kiltera user manual

Ernesto Posse

December 8, 2006

Contents

---

1 Introduction

This document describes how to use the kiltera interpreter and server. For information on the language itself, consult the language reference provided in the documentation (`doc/reference.pdf`.)

kiltera modules can be run in two modes:

- stand-alone, or
- as part of a system distributed over a network

The first mode is described in section 2. The second is described in section 3.

2 Using the kiltera interpreter

The basic mode of execution of kiltera modules is as stand-alone applications. In this mode, kiltera modules are executed in the command-line using the command

```
klt
```

This command is normally installed in the Python scripts directory. Usually this is `/usr/local/bin` or `/usr/bin` on Unix/Linux, and on Windows, the `Scripts` subdirectory of your Python installation (e.g. `C:\Program Files\Python24\Scripts`.) To find the precise location, go to the directory where you unpacked the distribution and type

```
cat install_data.py
```

The `install_scripts` field contains the directory where `klt` was installed. It is highly recommendable to add this directory to your path environment variable.[1]

2.1 Normal execution

The `klt` command is normally executed passing as parameter the name of the module to be interpreted, as follows:

`klt` <*file_name*>

It is recommendable to use `.klt` as extension for kiltera files.

If the execution of the module results in dead-lock (the interpreter hangs,) or is taking too much time, it can be interrupted by pressing `Ctrl-C`.

The interpreter can accept a number of different options, in which case the command has the form:

`klt` <*options*> <*file_name*>

The full list of options is summarized in section 4. You can also see the list of options by typing

```
klt -h
```

To see the version used, type

```
klt -V
```

To execute in verbose mode, which shows many internal messages, type

`klt -v`<*level*> <*file_name*>

where <*level*> is the level of detail in the messages. See section 4 for full details.

[1] On some machines the `klt` file might not be given the correct executable permissions, so you might have to execute it as `python` <*install_scripts*>`/klt` where <*install_scripts*> is the directory mentioned above. Note that executing `python` <*install_scripts*>`/klt.py` will not work. This is important since the files `klt` and `klt.py` are not the same.

2.2 Event traces

When executing a module it is very useful to obtain a log of the events which occurred. The interpreter provides three formats for such a trace:

- Short XML format
- Long XML format
- Tuple-based format

For each option, it is possible to either dump the trace into a file, or to standard output (which may be used in a pipe.)

Also, for each option it is possible to specify a format for the time tag in each event. This is described below.

To generate the trace in short XML format and save it into a file, type

`klt -X`<*trace_file*> <*module_file*>

To dump it to standard output, type

`klt -x` <*module_file*>

To generate the trace in long XML format and save it into a file, type

`klt -Y`<*trace_file*> <*module_file*>

To dump it to standard output, type

`klt -y` <*module_file*>

To generate the trace in tuple-based format and save it into a file, type

`klt -E`<*trace_file*> <*module_file*>

To dump it to standard output, type

`klt -e` <*module_file*>

In the generated trace, every event has a timestamp, or time-tag. For each option it is possible to specify a format for this time-tag in each event by using the `-P` option which has as argument the precision of the time, with the form <*length*>.<*decimals*>, or just .<*decimals*>. For example

```
klt -Xmytrace.trc -P7.3 mymodule.klt
```

2.3 Alternative main

A typical kiltera module has the form

```
module <name>

<top_level_function_definition_1>
<top_level_function_definition_2>
...
<top_level_function_definition_n>

<top_level_process_definition_1>
<top_level_process_definition_2>
...
<top_level_process_definition_m>
main
    <main_process>
```

which is short for

```
module <name>

<top_level_function_definition_1>
<top_level_function_definition_2>
...
<top_level_function_definition_n>

<top_level_process_definition_1>
<top_level_process_definition_2>
...
<top_level_process_definition_m>
process main[]:
    <main_process>
in
  main[]
```

It is possible to specify an alternative process as the "main" process to be executed by the interpreter by means of the `-m` command-line option, as follows:

```
klt -m<process_name> <file_name>
```

Where *<process_name>* must be defined in the module, otherwise there will be a runtime-error.

2.4 Pretty printer and syntax trees

The kiltera interpreter provides facilities to pretty-print the module, to print its abstract-syntax tree, to print its concrete syntax-tree and to print the results

of lexical analysis. These may be useful for dealing with syntax issues, or to be used by other tools such as editors, IDEs, or for module exchange.

To pretty-print the module, type

`klt -p` <*file_name*>

To print the abstract syntax tree, type

`klt -a` <*file_name*>

To print the concrete syntax tree, type

`klt -t` <*file_name*>

To print the full list of tokens produced by the lexer, type

`klt -l` <*file_name*>

2.5 Other options

If the kiltera prebuilt parser is corrupted, or if the installation process failed to build the parser, the interpreter may still build the parser on-line by typing

`klt -b` <*file_name*>

Keep in mind that this process may take several seconds.

It is also possible to see kiltera's internal parsing table by typing

`klt -T`

3 Using kiltera distributed over a network

kiltera sytems can be distributed over a network where modules can communicate with other possibly remote modules. In order to do this there must be a central *server* to which all modules in the system must be connected. Each module can then communicate with others via the server. The server is in charge of routing messages from each module to their destination[2].

In each module, the link to the server is provided by the first port of its main process. If the module wants to send a message to some other module, it must be a pair of the form `(`<*destination*>`,` <*data*>`)`, where <*destination*> is a string with the name of the target module, and <*data*> is any kiltera value. For example, consider the following two modules which may be in remote machines:

[2] In the current version, only remote asynchronous communication is supported.

```
#File module1.klt on machine 1
module A
process main[server]:
  send ("B", "hello") to server

#File module2.klt on machine 2
module B
process main[server]:
  receive msg from server ->
    print msg
```

In order to run this system it is necessary to setup a kiltera server, as described in section 3.1 and then connecting each of the modules to the server as described in section 3.2.

If there are multiple machines executing a module with the same name, the message will go to any of them, chosen randomly. But it is possible to specify to which particular instance we want to send the message by providing the name of the machine containing the module in the <*destination*>, where the <*destination*> field is a string which has the form <*target-machine*>:<*module-name*>. For example

```
#File module1.klt
module A
process main[server]:
  send ("mach1.cs.mcgill.ca:B", "hello") to server
```

If there is more than one instance of the module in the target machine, the message will go to any of them.

3.1 The kiltera server

The kiltera server plays the role of message-dispatcher for a distributed kiltera system. The server is initiated by the command

```
kltserver
```

which is found in the same location as the `klt` command (see section 2.)

Normally it is enough to start the server on some machine (typically as a deamon process.)

The server accepts command-line options, in which case the command is typed as follows:

`kltserver` <*options*>

The full list of options is summarized in section 4. You can also see the list of options by typing

```
kltserver -h
```

To see the version used, type

```
kltserver -V
```

To execute in verbose mode, which shows many internal messages, type

```
kltserver -v<level>
```

where <*level*> is the level of detail in the messages. See section 4 for full details.

The server and the client modules communicate by a TCP socket. To do this the use a *port*. The port number must be the same for the server and clients. By default both clients and the server are setup to have the same port, but if for some reason communication fails over that port, it is possible to specify an alternative port using the -p option:

```
kltserver -p<number>
```

In which case the clients must start with the same port (see section 3.2 below.)

3.2 Connecting kiltera modules to the server

In order to connect a kiltera module to a server, the module must be executed with the `-r` option, as follows:

```
klt -r<server_address> <module_file_name>
```

where <*server_address*> is the (symbolic) address of the server, for example:

```
klt -rhost.cs.mcgill.ca module1.klt
```

If the server was started with an alternative port number, then this port number must be specified in the command-line as follows:

```
klt -r<server_address>:<port_number> <module_file_name>
```

For example, if the server was executed in 'host.cs.mcgill.ca' as

```
kltserver -p50006
```

then all the client modules must start as

```
klt -rhost.cs.mcgill.ca:50006 <module_name>
```

3.3 A note about time, mobility and dynamic structure

In the current version, the support for distributed computing is limited in the sense that it does not support distributed timing and full mobility. The system does not guarrantee cross-network time consistency. With respect to mobility, if a channel is sent to a remote site, the channel cannot be used for remote communication.

Nevertheless, the system does support another form of dynamic structure: it is possible to add and remove new client modules at runtime.

Option	Description
`-h`	Lists all the options.
`-V`	Prints the version number.
`-v`<*level*>	Verbose mode; shows various internal messages with the given level of detail, where <*level*> is of the form `all`, <*number*>, <*number*>`:`<*mask*> or `all:`<*mask*>, where <*mask*> is a combination of: `o` (only one level), `T` (time), `t` (thread), m (module), `c` (class), `f` (function/method) and `l` (line number).
`-a`	Print the module's abstract syntax tree.
`-b`	Build kiltera's parser instead of using the prebuilt parser.
`-e`, `-E`<*file*>	Print or save the event trace to <*file*> in tuple format.
`-l`	Print the list of tokens produced by the lexer.
`-m`<*name*>	Use <*name*> as the main process.
`-p`	Pretty-print the given module.
`-P`<*length*>.<*decimals*>	Use the given format for the timestamps in the trace.
`-r`<*server*>	Connect to the remote <*server*>.
`-r`<*server*>:<*port*>	Connect to the remote <*server*> using the given <*port*>.
`-t`	Print the module's concrete syntax tree.
`-T`	Print kiltera's parsing table.
`-x`, `-X`<*file*>	Print or save the event trace to <*file*> in short XML format.
`-y`, `-Y`<*file*>	Print or save the event trace to <*file*> in long XML format.

Table 1: `klt` command-line options.

4 Command-line options summary

Table 1 shows the command-line options for the kiltera interpreter summarized.

Table 2 shows the command-line options for the kiltera server summarized.

Option	Description
`-h`	Lists all the options.
`-V`	Prints the version number.
`-v`<*level*>	Verbose mode; shows various internal messages with the given level of detail, where <*level*> is of the form `all`, <*number*>, <*number*>`:`<*mask*> or `all:`<*mask*>, where <*mask*> is a combination of: `o` (only one level), `T` (time), `t` (thread), m (module), `c` (class), `f` (function/method) and `l` (line number).
`-p`<*number*>	Uses the port number given for the TCP socket.

Table 2: `kltserver` command-line options.