The `SequentialTaskCollection`

GC3: Grid Computing Competence Center,
University of Zurich

Oct. 1, 2012

Running jobs in sequence

A `SequentialTaskCollection` runs its tasks one at a time.

`SequentialTaskCollection` can alter the sequence on the fly, insert new stages while running and loop back.

Running jobs in sequence

After a task has completed, the `next` method is called with the index of the `finished` task in the ‘self.tasks‘ list

the return value of the `next` method is then made the collection `execution.state`.

If the returned state is ‘RUNNING‘, then the subsequent task is started, otherwise no action is performed.

Example of a SequentialTaskCollection

```
class DemoIteration (SequentialTaskCollection):
    def __init__(self, ...):

        # create initial task and register it
        initial_task = GdemoApplication(...)
        SequentialTaskCollection.__init__(
            self,
            [initial_task]
            )
```

Create initial list of tasks

```
class DemoIteration(SequentialTaskCollection):
  def __init__(self, ...):

      # create initial task and register it
       initial_task = GdemoApplication(...)
      SequentialTaskCollection.__init__(
          self,
          [initial_task]
          )
```

`next` method used to dynamically define new task

```
def next(self, iteration):
    last_application = self.tasks[iteration]

    if computed_value == self.limit:
        self.returncode = 0
        return Run.State.TERMINATED
    else:
        self.add(GdemoApplication(
                    computed_value,
                    self.increment,
                    iteration+1
                    )
                )
        return Run.State.RUNNING
```

use `iteration` to identify last terminated task

```
def next(self, iteration):
    last_application = self.tasks[iteration]

    # extract information from 'last_application'
    # and process it

    if computed_value == self.limit:
        self.returncode = 0
        return Run.State.TERMINATED
    else:
        self.add(GdemoApplication(
                    computed_value,
                    self.increment,
                    iteration+1
                    )
                )
        return Run.State.RUNNING
```

Define termination condition: `return Run.State.TERMINATED`

```
def next(self, iteration):
    last_application = self.tasks[iteration]

    if computed_value == self.limit:
        self.returncode = 0
        return Run.State.TERMINATED
    else:
        self.add(GdemoApplication(
                    computed_value,
                    self.increment,
                    iteration+1
                    )
                )
        return Run.State.RUNNING
```

an Example from gmhc-coev

```
def next(self, iteration):
  if self.generations_done < self.generations_to_do:
      self.add(
          GMhcCoevApplication(
            self.N,
            self.p_mut_coeff,
            self.choose_or_rand,
            self.sick_or_not,
            self.off_v_last,
            output_dir = os.path.join(
                         self.output_dir,
                         'tmp'),
            latest_work = latest_work,
            executable = self.executable,
            **self.extra)
            )
      return Run.State.RUNNING
```

...or add new task in the list and continue

```
def next(self, iteration):
    last_application = self.tasks[iteration]

    if computed_value == self.limit:
        self.returncode = 0
        return Run.State.TERMINATED
    else:
        self.add(GdemoApplication(
                    computed_value,
                    self.increment,
                    iteration+1
                    )
                )
        return Run.State.RUNNING
```

let's look inside `SequentialTaskCollection` class

```
class SequentialTaskCollection(TaskCollection):

 def update_state(self, **extra_args):
  ...
  task = self.tasks[self._current_task]
  task.update_state(**extra_args)
  ...
  elif (task.execution.state == Run.State.TERMINATED)
  ...
  nxt = self.next(self._current_task)
  ...
```

update_state called by Engine

```
class SequentialTaskCollection(TaskCollection):

 def update_state(self, **extra_args):
  ...
  task = self.tasks[self._current_task]
  task.update_state(**extra_args)
  ...
  elif (task.execution.state == Run.State.TERMINATED)
  ...
  nxt = self.next(self._current_task)
  ...
```

let's look inside SequentialTaskCollection class

```
class SequentialTaskCollection(TaskCollection):

 def update_state(self, **extra_args):
  ...
  task = self.tasks[self._current_task]
  task.update_state(**extra_args)
  ...
  elif (task.execution.state == Run.State.TERMINATED)
  ...
  nxt = self.next(self._current_task)
  ...
```

let's look inside SequentialTaskCollection class

```
class SequentialTaskCollection(TaskCollection):

 def update_state(self, **extra_args):
  ...
  task = self.tasks[self._current_task]
  task.update_state(**extra_args)
  ...
  elif (task.execution.state == Run.State.TERMINATED)
  ...
  nxt = self.next(self._current_task)
  ...
```