Package 'simmer'

October 6, 2016

Type Package

Version 3.4.4

Title Discrete-Event Simulation for R

Description A process-oriented and trajectory-based Discrete-Event Simulation (DES) package for R. Designed to be a generic framework, it leverages the power of 'Rcpp' to boost the performance, turning DES in R feasible. As a noteworthy characteristic, 'simmer' exploits the concept of trajectory: a common path in the simulation model for entities of the same type.

Encoding UTF-8

URL http://r-simmer.org

BugReports https://github.com/r-simmer/simmer/issues

Depends R (>= 3.1.2)

Imports Rcpp, R6, magrittr, methods, utils

Suggests parallel, dplyr, tidyr, ggplot2, scales, testthat, knitr, rmarkdown, covr

LinkingTo Rcpp (>= 0.12.0), BH (>= 1.58.0-1)

License MIT + file LICENSE

RoxygenNote 5.0.1

VignetteBuilder knitr

NeedsCompilation yes

Author Bart Smeets [aut, cph], Iñaki Ucar [aut, cph, cre]

Maintainer Iñaki Ucar <i.ucar86@gmail.com>

Repository CRAN

Date/Publication 2016-10-06 19:22:26

R topics documented:

add_generator *Add a generator*

Description

Define a new generator of arrivals in a simulation environment.

Usage

```
add_generator(env, name_prefix, trajectory, dist, mon = 1, priority = 0,
  preemptible = priority, restart = FALSE)
```

Arguments

env	the simulation environment.
name_prefix	the name prefix of the generated arrivals.
trajectory	the trajectory that the generated arrivals will follow (see create_trajectory).
dist	a function modelling the interarrival times (returning a negative value stops the generator).
mon	whether the simulator must monitor the generated arrivals or not (0 = no monitoring, 1 = simple arrival monitoring, 2 = level 1 + arrival attribute montoring)
priority	the priority of each arrival (a higher integer equals higher priority; defaults to the minimum priority, which is 0).
preemptible	if a seize occurs in a preemptive resource, this parameter establishes the minimum incoming priority that can preempt these arrivals (an arrival with a priority greater than preemptible gains the resource). In any case, preemptible must be equal or greater than priority, and thus only higher priority arrivals can trigger preemption.
restart	whether the activity must be restarted after being preempted.

Value

Returns the simulation environment.

See Also

Convenience functions: at, from, to, from_to.

add_resource *Add a resource*

Description

Define a new resource in a simulation environment.

Usage

```
add_resource(env, name, capacity = 1, queue_size = Inf, mon = TRUE,
  preemptive = FALSE, preempt_order = c("fifo", "lifo"),
  queue_size_strict = FALSE)
```

Arguments

env	the simulation environment.
name	the name of the resource.
capacity	the capacity of the server.
queue_size	the size of the queue.
mon	whether the simulator must monitor this resource or not.
preemptive	whether arrivals in the server can be preempted or not based on seize priorities.
preempt_order	if the resource is preemptive and preemption occurs with more than one arrival in the server, this parameter defines which arrival should be preempted first. It must be `fifo` (First In First Out: older preemptible tasks are preempted first) or `lifo` (Last In First Out: newer preemptible tasks are preempted first).
queue_size_strict	if the resource is preemptive and preemption occurs, this parameter controls whether the `queue_size` is a hard limit. By default, preempted arrivals go to a dedicated queue, so that `queue_size` may be exceeded. If this option is `TRUE`, preempted arrivals go to the standard queue, and the maximum `queue_size` is guaranteed (rejection may occur).

Value

Returns the simulation environment.

See Also

Convenience functions: `schedule`.

at *Arrivals at specific times*

Description

Generator convenience function to generate arrivals at specific times.

Usage

```
at(...)
```

Arguments

...	a vector or multiple parameters of times at which to initiate an arrival.

Value

Returns a generator function.

See Also

add_generator.

Examples

```
t0 <- create_trajectory() %>% timeout(0)
env <- simmer(verbose=TRUE) %>%
  add_generator("dummy", t0, at(0, c(1,10,30), 40, 43)) %>%
  run(100)
```

batch *Add a batch/separate activity*

Description

Collect a number of arrivals before they can continue processing or split a previously established batch.

Usage

```
batch(traj, n, timeout = 0, permanent = FALSE, name = "", rule = NULL)

separate(traj)
```

Arguments

traj	the trajectory object.
n	batch size, accepts a numeric.
timeout	set an optional timer which triggers batches every timeout time units even if the batch size has not been fulfilled, accepts a numeric (0 = disabled).
permanent	if TRUE, batches cannot be split.
name	optional string. Unnamed batches from different batch activities are independent. However, if you want to feed arrivals from different trajectories into a same batch, you need to specify a common name across all your batch activities.
rule	an optional callable object (a function) which will be applied to every arrival to determine whether it should be included into the batch, thus

Value

The trajectory object.

---

branch *Add a branch activity*

Description

Define a fork with N alternative sub-trajectories.

Usage

```
branch(traj, option, continue, ...)
```

Arguments

traj	the trajectory object.
option	a callable object (a function) which must return an integer between 0 and N. A return value equal to 0 skips the branch and continues to the next activity. A returning value between 1 to N makes the arrival to follow the corresponding sub-trajectory.
continue	a vector of N booleans that indicate whether the arrival must continue to the main trajectory after each sub-trajectory or not.
...	N trajectory objects describing each sub-trajectory.

Value

The trajectory object.

clone *Add a clone/synchronize activity*

Description

A clone activity replicates an arrival n times (the original one + n-1 copies). A synchronize activity removes all but one clone.

Usage

```
clone(traj, n, ...)

synchronize(traj, wait = TRUE, mon_all = FALSE)
```

Arguments

traj	the trajectory object.
n	number of clones, accepts either a numeric or a callable object (a function) which must return a numeric.
...	optional parallel sub-trajectories. Each clone will follow a different sub-trajectory if available.
wait	if FALSE, all clones but the first to arrive are removed. if TRUE (default), all clones but the last to arrive are removed.
mon_all	if TRUE, get_mon_arrivals will show one line per clone.

Value

The trajectory object.

create_trajectory *Create a trajectory*

Description

This method initialises a trajectory object, which comprises a chain of activities that can be attached to a generator.

Usage

```
create_trajectory(name = "anonymous", verbose = FALSE)
```

Arguments

name	the name of the trajectory.
verbose	enable showing additional information.

Value

Returns an environment that represents the trajectory.

See Also

Methods for dealing with trajectories: get_head, get_tail, get_n_activities, join, seize, release, seize_selected, release_selected, select, set_prioritization, set_attribute, timeout, branch, rollback, leave, renege_in, renege_abort,clone, synchronize, batch, separate.

Examples

```
t0 <- create_trajectory("my trajectory") %>%
  ## add an intake activity
  seize("nurse", 1) %>%
  timeout(function() rnorm(1, 15)) %>%
  release("nurse", 1) %>%
  ## add a consultation activity
  seize("doctor", 1) %>%
  timeout(function() rnorm(1, 20)) %>%
  release("doctor", 1) %>%
  ## add a planning activity
  seize("administration", 1) %>%
  timeout(function() rnorm(1, 5)) %>%
  release("administration", 1)

t0

t1 <- create_trajectory("trajectory with a branch") %>%
  seize("server", 1) %>%
  # 50-50 chance for each branch
  branch(function() sample(1:2, 1), continue=c(TRUE, FALSE),
    create_trajectory("branch1") %>%
      timeout(function() 1),
    create_trajectory("branch2") %>%
      timeout(function() rexp(1, 3)) %>%
      release("server", 1)
  ) %>%
  # only the first branch continues here
  release("server", 1) %>%
  timeout(function() 2)

t1
```

---

from *Generate arrivals starting at a specified time*

Description

Generator convenience function to generate inter-arrivals with a specified start time.

Usage

```
from(start_time, dist, arrive = TRUE)
```

Arguments

start_time	the time at which to launch the initial arrival.
dist	a function modelling the interarrival times.
arrive	if set to TRUE (default) the first arrival will be generated at start_time and will follow dist from then on. If set to FALSE, will initiate dist at start_time (and the first arrival will most likely start at a time later than start_time).

Value

Returns a generator function.

See Also

add_generator.

Examples

```
t0 <- create_trajectory() %>% timeout(0)
env <- simmer(verbose=TRUE) %>%
  add_generator("dummy", t0, from(5, function() runif(1, 1, 2))) %>%
  run(10)
```

---

from_to *Generate arrivals starting and stopping at specified times*

Description

Generator convenience function to generate inter-arrivals with specified start and stop times.

Usage

```
from_to(start_time, stop_time, dist, arrive = TRUE)
```

Arguments

start_time	the time at which to launch the initial arrival.
stop_time	the time at which to stop the generator.
dist	a function modelling the interarrival times.
arrive	if set to TRUE (default) the first arrival will be generated at start_time and will follow dist from then on. If set to FALSE, will initiate dist at start_time (and the first arrival will most likely start at a time later than start_time).

Value

Returns a generator function.

See Also

add_generator.

Examples

```
t0 <- create_trajectory() %>% timeout(0)
env <- simmer(verbose=TRUE) %>%
  add_generator("dummy", t0, from_to(5, 10, function() runif(1, 1, 2))) %>%
  run(100)
```

---

get_head *Get the first/last activity*

Description

Trajectory getters for obtaining the pointer to its first/last activity.

Usage

```
get_head(traj)

get_tail(traj)
```

Arguments

traj	the trajectory object.

Value

An external pointer to an activity object.

See Also

get_n_activities, join.

get_mon *Get statistics*

Description

Simulator getters for obtaining monitored data (if any) about arrivals, attributes and resources.

Usage

```
get_mon_arrivals(envs, per_resource = FALSE, ongoing = FALSE)

get_mon_attributes(envs)

get_mon_resources(envs, data = c("counts", "limits"))
```

Arguments

envs	the simulation environment (or a list of environments).
per_resource	if TRUE, statistics will be reported on a per-resource basis.
ongoing	if TRUE, ongoing arrivals will be reported. The columns end_time and finished of these arrivals are reported as NAs.
data	whether to retrieve the "counts", the "limits" or both.

Value

Return a data frame.

get_next_activity *Get the next/prev activity*

Description

It takes an external pointer to an activity an returns the next/prev activity.

Usage

```
get_next_activity(activity)

get_prev_activity(activity)
```

Arguments

activity	an external pointer to the activity.

Value

An external pointer to an activity object.

See Also

get_head, get_tail, print_activity.

---

get_n_activities *Get the number of activities*

Description

Trajectory getter for obtaining the total number of activities defined inside it.

Usage

```
get_n_activities(traj)
```

Arguments

traj	the trajectory object.

Value

The number of activities in the trajectory.

See Also

get_head, get_tail, join.

---

get_n_generated *Get the number of arrivals generated*

Description

Simulator getter for obtaining the number of arrivals generated by a generator by name.

Usage

```
get_n_generated(env, name)
```

Arguments

env	the simulation environment.
name	the name of the generator.

Value

Returns a numeric value.

join *Join trajectories*

Description

Concatenate any number of trajectories in the specified order.

Usage

```
join(...)
```

Arguments

...	trajectory objects.

Value

A new trajectory object.

See Also

get_head, get_tail, get_n_activities.

Examples

```
t1 <- create_trajectory() %>% seize("dummy", 1)
t2 <- create_trajectory() %>% timeout(1)
t3 <- create_trajectory() %>% release("dummy", 1)

join(t1, t2, t3)

create_trajectory() %>%
  join(t1) %>%
  timeout(1) %>%
  join(t3)
```

leave *Add a leave activity*

Description

Leave the trajectory with some probability.

Usage

```
leave(traj, prob)
```

Arguments

traj	the trajectory object.
prob	a probability or a function returning a probability.

Value

The trajectory object.

---

now *Get current time*

Description

Get the current simulation time.

Usage

```
now(env)
```

Arguments

env	the simulation environment.

Value

Returns a numeric value.

See Also

peek.

---

peek *Peek next events*

Description

Look for future events in the event queue and (optionally) obtain info about them.

Usage

```
peek(env, steps = 1, verbose = FALSE)
```

Arguments

env	the simulation environment.
steps	number of steps to peek.
verbose	show additional information (i.e., the name of the process) about future events.

Value

Returns numeric values if verbose=F and a data frame otherwise.

See Also

now.

---

plot_attributes *Plot evolution of attribute data*

Description

Plot the evolution of user-supplied attribute data.

Usage

```
plot_attributes(envs, keys = c())
```

Arguments

envs	a single simmer environment or a list of environments representing several replications.
keys	the keys of attributes you want to plot (if left empty, all attributes are shown).

Value

a ggplot2 object.

See Also

plot_resource_usage, plot_resource_utilization, plot_evolution_arrival_times.

plot_evolution_arrival_times *Plot evolution of arrival times*

Description

Plot the evolution of arrival related times (flow, activity and waiting time).

Usage

```
plot_evolution_arrival_times(envs, type = c("activity_time", "waiting_time",
  "flow_time"))
```

Arguments

envs	a single simmer environment or a list of environments representing several replications.
type	one of c("activity_time", "waiting_time", "flow_time").

Value

a ggplot2 object.

See Also

plot_resource_usage, plot_resource_utilization, plot_attributes.

plot_resource_usage *Plot usage of a resource over time*

Description

Plot the usage of a resource over the simulation time frame.

Usage

```
plot_resource_usage(envs, resource_name, items = c("system", "queue",
  "server"), steps = FALSE)
```

Arguments

envs	a single simmer environment or a list of environments representing several replications.
resource_name	the name of the resource (character value).
items	the components of the resource to be plotted.
steps	adds the changes in the resource usage.

Value

a ggplot2 object.

See Also

plot_resource_utilization, plot_evolution_arrival_times, plot_attributes.

`plot_resource_utilization` *Plot utilization of resources*

Description

Plot the utilization of specified resources in the simulation.

Usage

```
plot_resource_utilization(envs, resources)
```

Arguments

envs	a single simmer environment or a list of environments representing several replications.
resources	a character vector with at least one resource specified - e.g. "c('res1','res2')".

Value

a ggplot2 object.

See Also

plot_resource_usage, plot_evolution_arrival_times, plot_attributes.

`print_activity` *Print an activity*

Description

It can be used to visualise an activity's internal structure.

Usage

```
print_activity(activity)
```

Arguments

activity	an external pointer to the activity.

See Also

get_head, get_tail, get_next_activity, get_prev_activity.

---

renege_in *Add a renege activity*

Description

Set or unset a timer after which the arrival will abandon.

Usage

```
renege_in(traj, t, out = NULL)

renege_abort(traj)
```

Arguments

traj	the trajectory object.
t	timeout to trigger reneging, accepts either a numeric or a callable object (a function) which must return a numeric.
out	optional sub-trajectory in case of reneging.

Value

The trajectory object.

---

reset *Reset a simulator*

Description

Reset the following components of a simulation environment: time, event queue, resources, generators and statistics.

Usage

```
reset(env)
```

Arguments

env	the simulation environment.

Value

Returns the simulation environment.

See Also

onestep, run.

`resource` *Set/Get a resource's parameters*

Description

Simulator getters/setters for a resource's server capacity/count and queue size/count.

Usage

```
set_capacity(env, name, value)

get_capacity(env, name)

set_queue_size(env, name, value)

get_queue_size(env, name)

get_server_count(env, name)

get_queue_count(env, name)
```

Arguments

`env`	the simulation environment.
`name`	the name of the resource.
`value`	new value to set.

Value

Return the simulation environment (setters) or a numeric value (getters).

rollback *Add a rollback activity*

Description

Go backwards to a previous point in the trajectory. Useful to implement loops.

Usage

```
rollback(traj, amount, times = 1, check)
```

Arguments

traj	the trajectory object.
amount	the amount of activities (of the same or parent trajectories) to roll back.
times	the number of repetitions until an arrival may continue.
check	a callable object (a function) which must return a boolean. If present, the times parameter is ignored, and the activity uses this function to check whether the rollback must be done or not.

Value

The trajectory object.

run *Run the simulation*

Description

Execute steps until the given criterion.

Usage

```
run(env, until = 1000)

onestep(env)
```

Arguments

env	the simulation environment.
until	stop time.

Value

Returns the simulation environment.

See Also

reset.

---

schedule *Generate a scheduling object*

Description

Resource convenience function to generate a scheduling object from a timetable specification.

Usage

```
schedule(timetable, values, period = Inf)
```

Arguments

timetable	absolute points in time in which the desired value changes.
values	one value for each point in time.
period	period of repetition.

Value

Returns a Schedule object.

See Also

add_resource.

Examples

```
# Schedule 3 units from 8 to 16 h
#          2 units from 16 to 24 h
#          1 units from 24 to 8 h
capacity_schedule <- schedule(c(8, 16, 24), c(3, 2, 1), period=24)

env <- simmer() %>%
  add_resource("dummy", capacity_schedule)
```

`seize` *Add a seize/release activity*

Description

Activities for seizing/releasing a resource, by name or a previously selected one.

Usage

```
seize(traj, resource, amount = 1, continue = NULL, post.seize = NULL,
  reject = NULL)

seize_selected(traj, amount = 1, id = 0, continue = NULL,
  post.seize = NULL, reject = NULL)

release(traj, resource, amount = 1)

release_selected(traj, amount = 1, id = 0)
```

Arguments

`traj`	the trajectory object.
`resource`	the name of the resource.
`amount`	the amount to seize/release, accepts either a numeric or a callable object (a function) which must return a numeric.
`continue`	a boolean (if `post.seize` OR `reject` is defined) or a pair of booleans (if `post.seize` AND `reject` are defined) to indicate whether these subtrajectories should continue to the next activity in the main trajectory.
`post.seize`	an optional trajectory object which will be followed after a successful seize.
`reject`	an optional trajectory object which will be followed if the arrival is rejected.
`id`	selection identifier for nested usage.

Value

The trajectory object.

See Also

`select`.

select *Select a resource*

Description

Resource selector for a subsequent seize/release.

Usage

```
select(traj, resources, policy = c("shortest-queue", "round-robin",
  "first-available", "random"), id = 0)
```

Arguments

traj	the trajectory object.
resources	one or more resource names, or a callable object (a function) which must return a resource name to select.
policy	if resources is a vector of names, this parameter determines the criteria for selecting a resource among the set of policies available; otherwise, it is ignored.
id	selection identifier for nested usage.

Value

The trajectory object.

See Also

seize_selected, release_selected.

set_attribute *Add a set attribute activity*

Description

Modify an attribute in the form of a key/value pair.

Usage

```
set_attribute(traj, key, value)
```

Arguments

traj	the trajectory object.
key	the attribute key (coerced to a string).
value	the value to set, accepts either a numeric or a callable object (a function) which must return a numeric.

Value

The trajectory object.

`set_prioritization` *Add a set prioritization activity*

Description

Modify the arrival's prioritization values.

Usage

```
set_prioritization(traj, values)
```

Arguments

`traj`	the trajectory object.
`values`	expects either a vector/list or a callable object (a function) returning a vector/list of three values `c(priority, preemptible, restart)`. A negative value leaves the corresponding parameter unchanged. See `add_generator` for more information about these parameters.

Value

The trajectory object.

`simmer` *Create a simulator*

Description

This method initialises a simulation environment.

Usage

```
simmer(name = "anonymous", verbose = FALSE)
```

Arguments

`name`	the name of the simulator.
`verbose`	enable showing activity information.

Value

Returns a simulation environment.

See Also

Methods for dealing with a simulation environment: reset, now, peek, onestep, run, add_resource, add_generator, get_mon_arrivals, get_mon_attributes, get_mon_resources, get_n_generated, get_capacity, get_queue_size, set_capacity, set_queue_size, get_server_count, get_queue_count.

Examples

```
t0 <- create_trajectory("my trajectory") %>%
  ## add an intake activity
  seize("nurse", 1) %>%
  timeout(function() rnorm(1, 15)) %>%
  release("nurse", 1) %>%
  ## add a consultation activity
  seize("doctor", 1) %>%
  timeout(function() rnorm(1, 20)) %>%
  release("doctor", 1) %>%
  ## add a planning activity
  seize("administration", 1) %>%
  timeout(function() rnorm(1, 5)) %>%
  release("administration", 1)

env <- simmer("SuperDuperSim") %>%
  add_resource("nurse", 1) %>%
  add_resource("doctor", 2) %>%
  add_resource("administration", 1) %>%
  add_generator("patient", t0, function() rnorm(1, 10, 2))

env %>% run(until=80)

plot_resource_usage(env, "doctor")
```

---

timeout *Add a timeout activity*

Description

Insert delays and execute user-defined tasks.

Usage

```
timeout(traj, task)
```

Arguments

traj	the trajectory object.
task	the timeout duration supplied by either passing a numeric or a callable object (a function) which must return a numeric (negative values are automatically coerced to positive).

Value

The trajectory object.

`to` *Generate arrivals stopping at a specified time*

Description

Generator convenience function to generate inter-arrivals with a specified stop time.

Usage

```
to(stop_time, dist)
```

Arguments

`stop_time`	the time at which to stop the generator.
`dist`	a function modelling the interarrival times.

Value

Returns a generator function.

See Also

`add_generator`.

Examples

```
t0 <- create_trajectory() %>% timeout(0)
env <- simmer(verbose=TRUE) %>%
  add_generator("dummy", t0, to(5, function() runif(1, 1, 2))) %>%
  run(10)
```

`wrap` *Wrap a simulation environment*

Description

This function extracts the monitored data from a simulation environment making it accessible through the same methods. Only useful if you want to parallelize heavy replicas (see the example below), because the C++ simulation backend is destroyed when the threads exit.

Usage

```
wrap(env)
```

Arguments

env	the simulation environment.

Value

Returns a simulation wrapper.

See Also

Methods for dealing with a simulation wrapper: get_mon_arrivals, get_mon_attributes, get_mon_resources, get_n_generated, get_capacity, get_queue_size, get_server_count, get_queue_count.

Examples

```
library(parallel)

mm1 <- create_trajectory() %>%
  seize("server", 1) %>%
  timeout(function() rexp(1, 2)) %>%
  release("server", 1)

envs <- mclapply(1:4, function(i) {
  simmer("M/M/1 example") %>%
    add_resource("server", 1) %>%
    add_generator("customer", mm1, function() rexp(1, 1)) %>%
    run(100) %>%
    wrap()
})

plot_resource_usage(envs, "server")
```

Index