Package 'simmer'

July 28, 2017

Type Package

Title Discrete-Event Simulation for R

Version 3.6.3

Description A process-oriented and trajectory-based Discrete-Event Simulation (DES) package for R. It is designed as a generic yet powerful framework. The architecture encloses a robust and fast simulation core written in C++ with automatic monitoring capabilities. It provides a rich and flexible R API that revolves around the concept of trajectory, a common path in the simulation model for entities of the same type.

License MIT + file LICENSE

Encoding UTF-8

URL http://r-simmer.org, https://github.com/r-simmer/simmer

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

Depends R (>= 3.1.2)

Imports Rcpp, R6, magrittr, codetools

Suggests simmer.plot, parallel, dplyr, tidyr, testthat, knitr, rmarkdown, covr

LinkingTo Rcpp (>= 0.12.9), BH (>= 1.62.0-1)

ByteCompile yes

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2017-07-28 04:24:34 UTC

R topics documented:

`activate` *Activate/Deactivate Generators*

Description

Activities for activating or deactivating the generation of arrivals by name.

Usage

```
activate(.trj, generator)

deactivate(.trj, generator)
```

Arguments

.trj	the trajectory object.
generator	the name of the generator or a function returning a name.

Value

Returns the trajectory object.

See Also

set_trajectory, set_distribution.

---

add_generator *Add a Generator*

Description

Define a new generator of arrivals in a simulation environment.

Usage

```
add_generator(.env, name_prefix, trajectory, distribution, 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 trajectory).
distribution	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 monitoring)
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 *Convenience Functions for Generators*

Description

These convenience functions facilitate the definition of generators of arrivals for some common cases.

Usage

```
at(...)

from(start_time, dist, arrive = TRUE)

to(stop_time, dist)

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

Arguments

`...`	a vector or multiple parameters of times at which to initiate an arrival.
`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`).
`stop_time`	the time at which to stop the generator.
`every`	repeat with this time cycle.

Details

`at` generates arrivals at specific absolute times. `from` generates inter-arrivals following a given distribution with a specified start time. `to` generates inter-arrivals following a given distribution with a specified stop time. `from_to` is the union of the last two.

Value

Returns a generator function.

See Also

`add_generator`.

Examples

```
t0 <- trajectory() %>%
  timeout(0)

simmer() %>%
  add_generator("dummy", t0, at(0, c(1,10,30), 40, 43)) %>%
  run(100) %>%
  get_mon_arrivals()
t0 <- trajectory() %>%
  timeout(0)

simmer() %>%
  add_generator("dummy", t0, from(5, function() runif(1, 1, 2))) %>%
  run(10) %>%
  get_mon_arrivals()
t0 <- trajectory() %>%
  timeout(0)

simmer() %>%
  add_generator("dummy", t0, to(5, function() runif(1, 1, 2))) %>%
  run(10) %>%
  get_mon_arrivals()
t0 <- trajectory() %>%
  timeout(0)

# from 8 to 16 h every 24 h:
simmer() %>%
  add_generator("dummy", t0, from_to(8, 16, function() runif(1, 1, 2), every=24)) %>%
  run(48) %>%
  get_mon_arrivals()
```

---

batch *Batch/Separate Arrivals*

Description

Activities for collecting a number of arrivals before they can continue processing and splitting a previously established batch.

Usage

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

separate(.trj)
```

Arguments

.trj	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 or a callable object (a function) which must return 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

Returns the trajectory object.

---

branch *Fork the Trajectory Path*

Description

Activity for defining a fork with N alternative sub-trajectories.

Usage

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

Arguments

.trj	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

Returns the trajectory object.

clone *Clone/Synchronize Arrivals*

Description

Activities for defining a parallel fork and removing the copies. `clone` replicates an arrival `n` times (the original one + `n-1` copies). `synchronize` removes all but one clone for each set of clones.

Usage

```
clone(.trj, n, ...)

synchronize(.trj, wait = TRUE, mon_all = FALSE)
```

Arguments

`.trj`	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

Returns the trajectory object.

Extract.trajectory *Extract or Replace Parts of a Trajectory*

Description

Operators acting on trajectories to extract or replace parts.

Usage

```
## S3 method for class 'trajectory'
x[i]

## S3 method for class 'trajectory'
x[[i]]
```

```
## S3 replacement method for class 'trajectory'
x[i] <- value

## S3 replacement method for class 'trajectory'
x[[i]] <- value
```

Arguments

x	the trajectory object.
i	indices specifying elements to extract. Indices are `numeric` or `character` or `logical` vectors or empty (missing) or `NULL`. Numeric values are coerced to integer as by `as.integer` (and hence truncated towards zero). Negative integers indicate elements/slices to leave out the selection. Character vectors will be matched to the names of the activities in the trajectory as by `%in%`. Logical vectors indicate elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. An empty index will return the whole trajectory. An index value of `NULL` is treated as if it were `integer(0)`.
value	another trajectory object.

Value

Returns a new trajectory object.

See Also

`length.trajectory`, `get_n_activities`, `join`.

---

get_capacity *Get Resource Parameters*

Description

Getters for resources: server capacity/count and queue size/count.

Usage

```
get_capacity(.env, resource)

get_queue_size(.env, resource)

get_server_count(.env, resource)

get_queue_count(.env, resource)
```

Arguments

.env	the simulation environment.
resource	the name of the resource.

Value

Return a numeric value.

See Also

set_capacity, set_queue_size.

---

get_mon *Monitoring Statistics*

Description

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

Returns a data frame.

get_n_generated *Get Process Parameters*

Description

Getters for processes (generators and arrivals) number of arrivals generated by a generator, the name of the active arrival, an attribute from the active arrival or a global one, and prioritization values.

Usage

```
get_n_generated(.env, generator)

get_name(.env)

get_attribute(.env, key, global = FALSE)

get_global(.env, key)

get_prioritization(.env)
```

Arguments

.env	the simulation environment.
generator	the name of the generator.
key	the attribute name.
global	if TRUE, the attribute will be global instead of per-arrival.

Details

get_n_generated returns the number of arrivals generated by a given generator.

get_name returns the number of the running arrival. get_attribute returns a running arrival's attribute or a global one. If the provided key was not previously set, it returns a missing value. get_global is a shortcut for get_attribute(global=TRUE). get_prioritization returns a running arrival's prioritization values. get_name, get_attribute and get_prioritization are meant to be used inside a trajectory; otherwise, there will be no arrival running and these functions will throw an error.

See Also

set_attribute, set_prioritization.

join *Join Trajectories*

Description

Concatenate any number of trajectories in the specified order.

Usage

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

Arguments

...	trajectory objects.

Value

Returns a new trajectory object.

See Also

[.trajectory, [[.trajectory, length.trajectory, get_n_activities.

Examples

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

join(t1, t2, t3)

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

leave *Leave the Trajectory*

Description

Activity for leaving the trajectory with some probability.

Usage

```
leave(.trj, prob)
```

Arguments

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

Value

Returns the trajectory object.

---

length.trajectory *Number of Activities in a Trajectory*

Description

Get the number of activities in a trajectory. `length` returns the number of first-level activities (sub-trajectories not included). `get_n_activities` returns the total number of activities (sub-trajectories included).

Usage

```
## S3 method for class 'trajectory'
length(x)

get_n_activities(x)
```

Arguments

x	the trajectory object.

Value

Returns a non-negative integer of length 1.

See Also

`[.trajectory`, `[[.trajectory`, `join`.

log_ *Logging*

Description

Activity for displaying messages preceded by the simulation time and the name of the arrival.

Usage

```
log_(.trj, message)
```

Arguments

.trj	the trajectory object.
message	the message to display, accepts either a string or a callable object (a function) which must return a string.

Value

Returns the trajectory object.

now *Simulation 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.

renege_in *Renege on some Condition*

Description

Activities for setting or unsetting a timer or a signal after which the arrival will abandon.

Usage

```
renege_in(.trj, t, out = NULL)

renege_if(.trj, signal, out = NULL)

renege_abort(.trj)
```

Arguments

.trj	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.
signal	signal to trigger reneging, accepts either a string or a callable object (a function) which must return a string.

Value

Returns the trajectory object.

See Also

send

---

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.

rollback *Rollback a Number of Activities*

Description

Activity for going backwards to a previous point in the trajectory. Useful to implement loops.

Usage

```
rollback(.trj, amount, times = Inf, check = NULL)
```

Arguments

.trj	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

Returns the trajectory object.

run *Run a Simulation*

Description

Execute steps until a given criterion.

Usage

```
run(.env, until = 1000, progress = NULL, steps = 10)

onestep(.env)
```

Arguments

.env	the simulation environment.
until	stop time.
progress	optional callback to show the progress of the simulation. The completed ratio is periodically passed as argument to the callback.
steps	number of steps to show as progress (it takes effect only if progress is provided).

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 *Seize/Release Resources*

Description

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

Usage

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

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

release(.trj, resource, amount = 1)

release_selected(.trj, amount = 1, id = 0)
```

Arguments

.trj	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

Returns the trajectory object.

See Also

select, set_capacity, set_queue_size, set_capacity_selected, set_queue_size_selected.

select *Select Resources*

Description

Activity for selecting a resource for a subsequent seize/release or setting its parameters (capacity or queue size).

Usage

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

Arguments

.trj	the trajectory object.
resources	one or more resource names, or a callable object (a function) which must return one or more resource names.
policy	if resources is a character vector, this parameter determines the criteria for selecting a resource among the set of policies available: 'shortest-queue' selects the least busy resource, 'round-robin' selects the resources in order cyclically, 'first-available' selects the first resource available, and 'random' selects one at random.
id	selection identifier for nested usage.

Value

Returns the trajectory object.

See Also

seize_selected, release_selected, set_capacity_selected, set_queue_size_selected.

send *Inter-arrival Communication*

Description

These activities enable asynchronous programming. send() broadcasts a signal or a list of signals. Arrivals can subscribe to signals and (optionally) assign a handler with trap(). Note that, while inside a batch, all the signals subscribed before entering the batch are ignored. Upon a signal reception, the arrival stops the current activity and executes the handler (if provided). Then, the execution returns to the activity following the point of the interruption. untrap() can be used to unsubscribe from signals. wait() blocks until a signal is received.

Usage

```
send(.trj, signals, delay = 0)

trap(.trj, signals, handler = NULL, interruptible = TRUE)

untrap(.trj, signals)

wait(.trj)
```

Arguments

.trj	the trajectory object.
signals	signal or list of signals, accepts either a string, a list of strings or a callable object (a function) which must return a string or a list of strings.
delay	optional timeout to trigger the signals, accepts either a numeric or a callable object (a function) which must return a numeric.
handler	optional trajectory object to handle a signal received.
interruptible	whether the handler can be interrupted by signals.

Value

Returns the trajectory object.

See Also

renege_if

---

set_attribute *Set Attributes*

Description

Activity for modifying an arrival's attribute in the form of a key/value pair.

Usage

```
set_attribute(.trj, key, value, global = FALSE)

set_global(.trj, key, value)
```

Arguments

.trj	the trajectory object.
key	the attribute name.
value	the value to set, accepts either a numeric or a callable object (a function) which must return a numeric.
global	if TRUE, the attribute will be global instead of per-arrival.

Details

`set_global` is a shortcut for `set_attribute(global=TRUE)`.

Value

Returns the trajectory object.

See Also

`get_attribute`.

`set_capacity` *Set Resource Parameters*

Description

Activities for modifying a resource's server capacity or queue size, by name or a previously selected one.

Usage

```
set_capacity(.trj, resource, value)

set_capacity_selected(.trj, value, id = 0)

set_queue_size(.trj, resource, value)

set_queue_size_selected(.trj, value, id = 0)
```

Arguments

`.trj`	the trajectory object.
`resource`	the name of the resource.
`value`	new value to set.
`id`	selection identifier for nested usage.

Value

Returns the trajectory object.

See Also

`select`, `seize`, `release`, `seize_selected`, `release_selected`, `get_capacity`, `get_queue_size`.

set_prioritization *Set Prioritization Values*

Description

Activity for modifying an arrival's prioritization values.

Usage

```
set_prioritization(.trj, values)
```

Arguments

.trj	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

Returns the trajectory object.

See Also

get_prioritization.

set_trajectory *Set Generator Parameters*

Description

Activities for modifying a generator's trajectory or distribution by name.

Usage

```
set_trajectory(.trj, generator, trajectory)

set_distribution(.trj, generator, distribution)
```

Arguments

.trj	the trajectory object.
generator	the name of the generator or a function returning a name.
trajectory	the trajectory that the generated arrivals will follow.
distribution	a function modelling the interarrival times (returning a negative value stops the generator).

Value

Returns the trajectory object.

See Also

activate, deactivate.

---

simmer **simmer***: 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 like **SimPy** or **SimJulia**, it leverages the power of **Rcpp** to boost the performance and 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. It is pretty flexible and simple to use, and leverages the chaining/piping workflow introduced by the **magrittr** package.

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.

Author(s)

Iñaki Ucar and Bart Smeets

See Also

simmer's homepage http://r-simmer.org and GitHub repository https://github.com/r-simmer/simmer.

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, get_server_count, get_queue_count.

Examples

```
## Not run:
# introduction to simmer
vignette("A-introduction")

# more vignettes
vignette(package = "simmer")

## End(Not run)

t0 <- 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)) %>%
  run(until=80)
```

---

timeout *Delay*

Description

Activity for inserting delays and execute user-defined tasks.

Usage

```
timeout(.trj, task)
```

Arguments

.trj	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

Returns the trajectory object.

`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

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

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: `[.trajectory`, `[[.trajectory`, `length.trajectory`, `get_n_activities`, `join`, `seize`, `release`, `seize_selected`, `release_selected`, `select`, `set_capacity`, `set_queue_size`, `set_capacity_selected`, `set_queue_size_selected`, `set_prioritization`, `activate`, `deactivate`, `set_trajectory`, `set_distribution`, `set_attribute`, `timeout`, `branch`, `rollback`, `leave`, `renege_in`, `renege_if`, `renege_abort`, `clone`, `synchronize`, `batch`, `separate`, `send`, `trap`, `untrap`, `wait`, `log_`.

Examples

```
t0 <- 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 <- trajectory("trajectory with a branch") %>%
  seize("server", 1) %>%
  # 50-50 chance for each branch
  branch(function() sample(1:2, 1), continue=c(TRUE, FALSE),
    trajectory("branch1") %>%
      timeout(function() 1),
    trajectory("branch2") %>%
      timeout(function() rexp(1, 3)) %>%
      release("server", 1)
  ) %>%
  # only the first branch continues here
  release("server", 1) %>%
  timeout(function() 2)

t1
```

`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

```
## Not run:
library(parallel)

mm1 <- 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()
})

## End(Not run)
```

Index