User's Guide for ContactCenters Simulation Library

API Specification for the Blend/multi-skill call center simulator

Version: March 17, 2014

Eric Buist

This is the API specification for the generic blend/multi-skill simulator using the ContactCenters library. This API describes the classes of this simulator, as well as its extensions.

Contents

Package umontreal.iro.lecuyer.contactcenters.msk

Provides a generic simulator for multi-skill and blend call centers. ContactCenters can be used directly to construct simulators for arbitrarily complex contact centers. See the `examples.pdf` document for examples of this. However, this requires programming and the resulting programs can become complex. This package provides a generic simulator adapted for call centers with multiple call types and agent groups, and using XML configuration files. It can be used for many simulation scenarios, and estimates a large set of performance measures.

This reference documentation covers all classes and methods in the simulator. It is targeted at developers who are using the tool in a program, or extending it. See the `guidemsk.pdf` document for a description of the model implemented by this simulator, and examples showing how to configure and use the tool from a user perspective.

The simulator implemented in this package is split into several components representing the model, the simulation logic, and the system managing statistics. The model regroups every entity of the call center, e.g., calls, agent groups, waiting queues, routers, etc. It also specifies how random numbers are generated throughout the simulation.

The simulation logic contains the necessary instructions to run the model in order to generate results. It defines the concept of a *step* and assigns *statistical periods* to calls. For a simulation with independent replications, each step corresponds to a replication while the statistical period is usually the period of arrival. For a simulation of a single period as it was infinite in the model, steps correspond to time intervals of a single long replication, and the statistical period is always 0.

The system managing statistics, on the other hand, is made of observers, and matrices of counters. Observers are registered to collect information about every call leaving the system, and the evolution of agent groups and waiting queues. All this information is used to update matrices of counters whose rows usually correspond to call types or agent groups, and columns, to periods. At the end of each simulation step, the values of the counters are added to matching matrices of statistical collectors. After the simulation is done, matrices of averages, sample variances, and other statistics can be obtained.

Figure 1 shows a UML diagram of the simulator's main classes. The model is implemented by classes in package `umontreal.iro.lecuyer.contactcenters.msk.model`, `CallCenter` being the main class. The simulation logics are implemented in package `umontreal.iro.lecuyer.contactcenters.msk.simlogic` with the `SimLogic` interface representing any simulation logic. The management of statistics is in package `umontreal.iro.lecuyer.contactcenters.msk.stat`. Matrices of counters are encapsulated in an object of `CallCenterMeasureManager`, while matrices of statistical probes are stored in an instance of a class implementing the `CallCenterStatProbes`. See the documentation of these packages for more information about these components and the classes.

The simulator provides three packages in addition to the packages providing the implementation for the main components. The package `umontreal.iro.lecuyer.contactcenters.msk.cv` provides an implementation of control variates to reduce the variance in simulations.

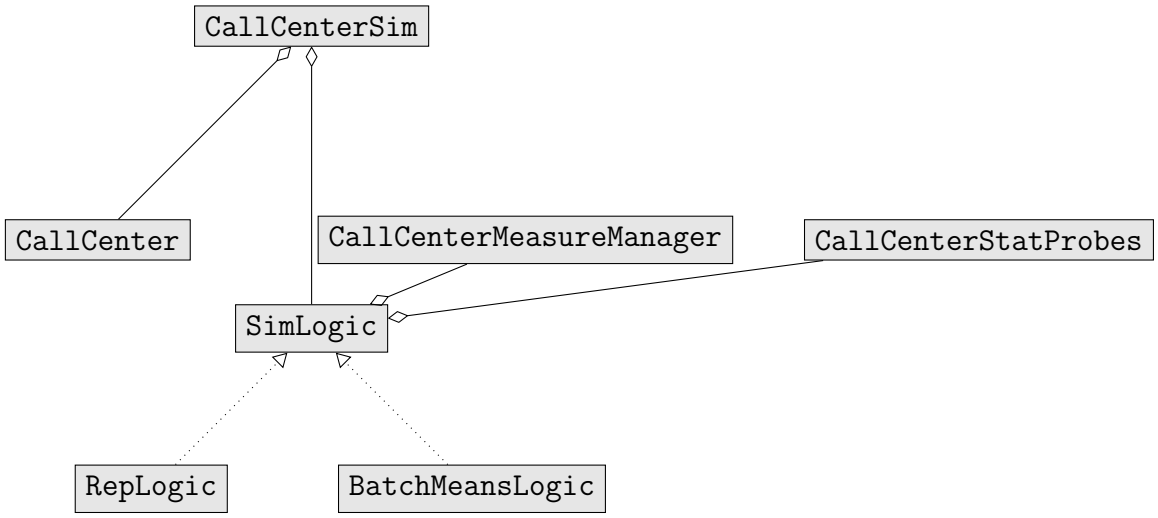

Figure 1: UML diagram of the generic blend/multi-skill simulator

The package `umontreal.iro.lecuyer.contactcenters.msk.spi` provides interfaces used when integrating a custom arrival process, routing or dialing policy into the simulator. The package `umontreal.iro.lecuyer.contactcenters.msk.conditions` implement conditions that can be used by some routing and dialing policies.

The class `CallCenterSim` represents the simulator as a whole, and implements the interface `ContactCenterSim`, which provides methods to perform simulations and obtain statistics in a standardized way. It provides a main method that can be used to call the simulator from the command line. The simulator can also be called from Java code.

CallCenterSim

Encapsulates all the components of the blend and multi-skill call center simulator, and provides methods to perform simulations and obtain results. This class uses the `CallCenter` class to implement a model, and a `SimLogic` implementation for the simulation logic. It also uses an implementation of `CallCenterStatProbes` for statistical collecting.

An object of this class is constructed using parameter objects usually read from XML files. The parameters of the model are stored into an instance of `CallCenterParams`, while the parameters of the experiment are encapsulated into an object of class `SimParams`. The classes `CallCenterParamsConverter` and `SimParamsConverter` can be used to read parameters from XML files.

After the simulator is constructed, it can be accessed in a standardized way through the `ContactCenterEval` interface, which defines methods to obtain global information about the call center, perform simulations, and retrieve matrices of statistics.

The `CallCenterSim` class also provides a `main` method accepting as arguments the name of the parameter files, performing a simulation, and showing results. This permits the simulator to be launched from the command-line.

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class CallCenterSim extends AbstractCallCenterSim
             implements ContactCenterSimWithObservations
```

Constructors

```
public CallCenterSim (CallCenterParams ccParams, SimParams simParams)
                      throws CallCenterCreationException
```

Constructs a new call center simulator using call center parameters `ccParams`, and simulation parameters `simParams`.

This calls `AbstractCallCenterSim.createModel (Simulator, CallCenterParams, RandomStreams)` to create the model, `AbstractCallCenterSim.createSimLogic (CallCenter, SimParams)` to create the simulation logic.

Parameters

`ccParams` the call center parameters.

`simParams` the simulation parameters.

```
public CallCenterSim (CallCenterParams ccParams, SimParams simParams,
                      RandomStreams streams) throws
                      CallCenterCreationException
```

Constructs a new call center simulator using call center parameters `ccParams`, simulation parameters `simParams`, and random streams `streams`.

This calls `AbstractCallCenterSim.createModel (Simulator, CallCenterParams, RandomStreams)` to create the model, `AbstractCallCenterSim.createSimLogic (CallCenter, SimParams)` to create the simulation logic.

Parameters

`ccParams` the call center parameters.

`simParams` the simulation parameters.

`streams` the random streams used by the simulator.

```
public CallCenterSim (Simulator sim, CallCenterParams ccParams, SimParams
                      simParams, RandomStreams streams) throws
                      CallCenterCreationException
```

Similar to `CallCenterSim (CallCenterParams, SimParams, RandomStreams)`, with the given simulator `sim`.

```
public CallCenterSim (Simulator sim, CallCenterParams ccParams, SimParams
                      simParams) throws CallCenterCreationException
```

Similar to `CallCenterSim (CallCenterParams, SimParams)`, with the given simulator `sim`.

Method

```
public static void main (String[] args)
```

Main method allowing to run this class from the command-line. The needed command-line arguments are the name of an XML file containing the non-stationary simulation parameters (root element `mskccparams`), and the name of a second XML file containing the simulation parameters (root elements `batchsimparams` or `repsimparams`).

Parameter

`args` the command-line arguments.

CallTracer

Observer sending any notified call to a contact trace facility. An object of this class is constructed using a `ContactTrace` instance as well as a `SimLogic` object. Each time a call exits the simulated system, a line is written to the associated trace, using information obtained from the contact object, and the simulation logic.

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class CallTracer implements ExitedContactListener, NewContactListener
```

Constructor

`public CallTracer (SimLogic simLogic, ContactTrace trace)`

Creates a new call tracer from the given simulation logic and trace.

Parameters

`simLogic` the simulation logic used to get step and period for the trace.

`trace` the object representing the call-by-call trace facility.

Methods

`public SimLogic getSimLogic()`

Returns the simulation logic associated with this call tracer.

`public ContactTrace getContactTrace()`

Returns the associated facility for contact-by-contact trace.

`public void register()`

Registers this call tracer with the model associated with the simulation logic returned by `getSimLogic()`. After this method is called, this listener is notified about every contact leaving the simulated system as well as any failed outbound call.

`public void unregister()`

Unregisters this call tracer with the model associated with the simulation logic returned by `getSimLogic()`.

CallCenterParamsConverter

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class CallCenterParamsConverter extends JAXBParamsConverter<
             CallCenterParams>
```

ParameterEstimator

Estimates the parameters of a call center model. This class defines a main method that loads a parameter file, estimates the parameters for probability distribution with associated data, and writes a new file for the same model, with the estimated parameters.

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class ParameterEstimator
```

Methods

```
public static void setParamsFromDefault (ArrivalProcessParams par,
                                         ArrivalProcessParams defaultPar)
```

If the basic parameters of arrival process `par` are not set, sets them to those of the default arrival process `defPar`.

Parameters

`par` parameters of the given arrival process

`defaultPar` parameters of the default arrival process

```
public static boolean estimateParameters (CallCenterParams ccParams)
                                           throws
                                          DistributionCreationException
```

Estimates the parameters for each element in the call center parameter objects for which raw observations are specified. Returns `true` if at least one parameter has been estimated by this method.

Parameter

`ccParams` the call center parameters.

Returns true if method was successfull.

Throws

`DistributionCreationException` if an error occurs during the creation of a distribution.

```
public static void main (String[] args)
```

Main method of this class taking, as arguments, the names of the input and the output files.

Parameter

`args` the arguments given to the program.

Throws

`IOException` if an I/O error occurs when reading or writing files.

`ParserConfigurationException` if an error occurs when parsing the XML file.

`SAXException` if an error occurs with SAX, when parsing the XML file.

`TransformerException` if an error occurs when creating the output XML file.

AbstractCallCenterSim

```
package umontreal.iro.lecuyer.contactcenters.msk;

public abstract class AbstractCallCenterSim extends AbstractContactCenterSim
                      implements ObservableContactCenterSim
```

Constructors

```
public AbstractCallCenterSim (CallCenterParams ccParams, SimParams
                               simParams) throws
                               CallCenterCreationException
```

Constructs a new call center simulator using call center parameters `ccParams`, and simulation parameters `simParams`.

This calls `createModel (Simulator, CallCenterParams, RandomStreams)` to create the model, `createSimLogic (CallCenter, SimParams)` to create the simulation logic.

Parameters

`ccParams` the call center parameters.

`simParams` the simulation parameters.

```
public AbstractCallCenterSim (CallCenterParams ccParams, SimParams
                               simParams, RandomStreams streams) throws
                               CallCenterCreationException
```

Constructs a new call center simulator using call center parameters `ccParams`, simulation parameters `simParams`, and random streams `streams`.

This calls `createModel (Simulator, CallCenterParams, RandomStreams)` to create the model, `createSimLogic (CallCenter, SimParams)` to create the simulation logic.

Parameters

`ccParams` the call center parameters.

`simParams` the simulation parameters.

`streams` the random streams used by the simulator.

```
public AbstractCallCenterSim (Simulator sim, CallCenterParams ccParams,
                               SimParams simParams) throws
                               CallCenterCreationException
```

Similar to `AbstractCallCenterSim (CallCenterParams, SimParams)`, with the given simulator `sim`.

```
public AbstractCallCenterSim (Simulator sim, CallCenterParams ccParams,
                               SimParams simParams, RandomStreams streams)
                                throws CallCenterCreationException
```

Similar to `AbstractCallCenterSim (CallCenterParams, SimParams, RandomStreams)`, with the given simulator `sim`.

Methods

`@Deprecated public CallCenter getModel()`

Use `getCallCenter()` instead.

`public CallCenter getCallCenter()`

Returns a reference to the model used by this simulator.

Returns a reference to the model.

`public SimLogic getSimLogic()`

Returns a reference to the simulation logic used by this simulator.

Returns a reference to the simulation logic.

```
protected CallCenter createModel (Simulator sim, CallCenterParams ccPs,
                                  RandomStreams streams) throws
                                  CallCenterCreationException
```

Constructs and returns the model of the call center used by this simulator. By default, this method constructs an instance of the `CallCenter` class, calls the `CallCenter.create()` method, and returns the resulting model object.

Parameters

`ccPs` the parameters of the call center.

`streams` the random streams.

Returns the constructed model.

`protected SimLogic createSimLogic (CallCenter model, SimParams simParams)`

Constructs and returns a `SimLogic` implementation for the simulation logic, using the given `model` and simulation parameters `simParams`.

By default, this method creates a `RepLogic` instance if `simParams` is an instance of `RepSimParams`, a `BatchMeansLogic` if `simParams` is an instance of `BatchSimParams`, and throws an exception otherwise.

Parameters

`model` the simulation model.

`simParams` the simulation parameters.

Returns the simulation logic.

CallCenterSimUtil

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class CallCenterSimUtil
```

Methods

```
public static int getRequiredNewSteps (Map<PerformanceMeasureType,
                                        MatrixOfStatProbes<?>> ccStat, List
                                        <SequentialSamplingParams> seqSamp,
                                        boolean verbose)
```

Computes the number of additional replications or batches required for reaching a certain precision.

Parameters

`ccStat` the statistical probes of the call center.

`seqSamp` the parameters for sequential sampling.

`verbose` determines if the method logs information about the number of required additional observations, for each tested performance measure.

Returns the number of additional observations required.

```
public static int checkCpuTimeLimit (double cpuTime, double limit, int
                                     steps, int nb, boolean verbose)
```

Corrects the number of observations required to approximately enforce the CPU time limit. This method estimates the CPU time for computing one observation by dividing the CPU time elapsed by `nb`, and estimates the maximal number of observations allowed without exceeding the CPU time limit. The method then returns this number, or `nb` if the limit is greater than `nb`.

Parameter

`nb` the computed number of additional observations.

Returns the corrected number of additional observations.

OldCallCenterParamsConverter

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class OldCallCenterParamsConverter
```

PeriodCovarianceEstimator

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class PeriodCovarianceEstimator implements MatrixOfObservationListener
```

CallCenterSimStrat

Defines a call center simulator using stratified sampling. This simulator stratifies on B, the busyness factor for inbound calls, and uses proportional allocation.

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class CallCenterSimStrat extends AbstractCallCenterSim
```

Constructor

```
public CallCenterSimStrat (CallCenterParams ccParams, StratSimParams
                           simParams) throws CallCenterCreationException
```

Constructs a new stratified call center simulator using the call center parameters `ccParams`, the simulation parameters `simParams`, and simulating `numStrata` strata.

Parameters

`ccParams` the call center parameters.

`simParams` the simulation parameters.

Methods

```
public void setProportionalAllocation()
```

Initializes the number of replications in each stratum for proportional allocation. This sets the number of replications n_s to n/m, where n is the total number of replications and m is the number of strata.

```
public void setOptimalAllocation (PerformanceMeasureType m, int r, int c,
                                  boolean cv)
```

Sets the number of replications in each stratum for optimal allocation minimizing the variance of performance measure of type `m`, at row `r` and column `c`. The boolean `cv` determines if control variables are used. One must call `makePilotRuns()` before calling this method.

Parameters

`m` the type of performance measure.

`r` the row index.

`c` the column index.

`cv` determines if control variables will be used.

```
public static void main (String[] args) throws IOException, JAXBException,
                         CallCenterCreationException
```

Main method allowing to run this class from the command-line. The needed command-line arguments are the name of an XML file containing the non-stationary simulation parameters (root element `mskccparams`), and the name of a second XML file containing the simulation parameters (root elements `batchsimparams` or `repsimparams`).

Parameter

`args` the command-line arguments.

CallCenterSimRQMC

Extends the `CallCenterSim` class for randomized Quasi-Monte Carlo simulation.

```
package umontreal.iro.lecuyer.contactcenters.msk;

public class CallCenterSimRQMC extends AbstractCallCenterSim
```

Constructor

```
public CallCenterSimRQMC (CallCenterParams ccParams, RepSimParams
                          simParams, int numPoints) throws
                          CallCenterCreationException
```

Constructs a new randomized Quasi-Monte Carlo call center simulator using the call center parameters `ccParams`, and simulation parameters `simParams`, with a point set containing `numPoints` points.

Parameters

`ccParams` the call center parameters.

`simParams` the simulation parameters.

`numPoints` the number of points in the point set.

Methods

```
public PointSet getPointSet()
```

Returns the point set used by this simulator.

Returns the point set being used.

```
protected PointSet createPointSet (int numPoints)
```

Creates the point set used for Quasi-Monte Carlo, which contains `numPoints` points. By default, this creates a Sobol sequence with one dimension, and containing `numPoints` points.

Parameter

`numPoints` the number of points in the point set.

Returns the constructed point set.

`protected void installPointSet()`

Configures the simulator for generating random numbers from the point set rather than from the default random streams. By default, this changes the busyness generator to obtain the busyness factor from the first dimension of the point set.

`protected void uninstallPointSet()`

Restors the simulator to stop using the point set.

`protected void randomizePointSet()`

Randomize the point set for a new macro-replication. By default, this applies an affine matrix scrambbling followed by a random digital shift.

```
public static void main (String[] args) throws IOException,
                         CallCenterCreationException, JAXBException
```

Main method allowing to run this class from the command-line. The needed command-line arguments are the name of an XML file containing the non-stationary simulation parameters (root element `mskccparams`), and the name of a second XML file containing the simulation parameters (root elements `batchsimparams` or `repsimparams`).

Parameter

`args` the command-line arguments.

Package `umontreal.iro.lecuyer.contactcenters.msk.model`

Provides the classes that implement the model of a call center with multiple call types and agent groups used by the blend and multi-skill simulator. The implemented model contains several elements: random streams for each type of random variate, factories for creating objects representing calls, arrival processes for generating the arrival times of inbound calls, dialers for producing outbound calls, agent groups, waiting queues, and a router. The parameters of the model are obtained using XML files transformed by JAXB into intermediate objects regrouped in an instance of the `CallCenterParams` class. The class `CallCenterParamsConverter` can be used to help in the conversion of XML data to an instance of `CallCenterParams`. The parameter objects are used to create the model, which provides methods to access parameters in a convenient way.

Arrival processes, dialers, agent groups, and the router are encapsulated in manager objects playing several roles: provide convenience methods for accessing parameters specific to the managed object, create the appropriate instance for the managed object, initialize it at the beginning of a simulation, and update its state throughout the simulation. All these manager objects can be accessed using methods in the `CallCenter` class, which is the central point of the model.

The model is usually created by the `CallCenterSim` class, and can be retrieved by its `getCallCenter` method. However, one may create a `CallCenter` object directly, and use it to get some information about the represented call center (e.g., mean service times), or perform custom simulations.

CallCenter

Represents the model of a call center with multiple call types and agent groups. The model encapsulates all the logic of the call center itself: a simulator with a clock and event list, a simulation event marking the change of periods, the call factories which create objects for every call, manager objects for arrival processes, dialers, agent groups, and the router, etc. A program can use methods in this class to obtain references to the call center objects, and retrieve their parameters, or register listeners to observe their evolution in time.

A model is created from an instance of `CallCenterParams`, and an instance of `RandomStreams`. After it is created using the `create()` method, it can be initialized for simulation using `initSim()`. The encapsulated period-change event, and managed arrival processes and dialers must then be started to schedule events before the simulation is started using `simulator().start()`.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallCenter extends AbstractContactCenterInfo
```

Constructors

`public CallCenter (CallCenterParams ccParams, RandomStreams streams)`

Constructs a new call center model from the call center parameters `ccParams`, and the random streams `streams`. This constructor assumes that `ccParams` is valid. The class `CallCenterParamsConverter` can be used to create valid objects from XML files.

Note that the `create()` method must be called after the model is constructed in order to create the model.

Parameters

`ccParams`

`streams`

Throws

`NullPointerException` if `ccParams` or `streams` are `null`.

`public CallCenter (Simulator sim, CallCenterParams ccParams, RandomStreams streams)`

Similar to `CallCenter (CallCenterParams, RandomStreams)`, with the given simulator `sim`.

`public CallCenter (CallCenterParams ccParams)`

Creates a call center model with parameters stored in `ccParams`, and using the default class of random stream `MRG32k3a`.

Note that the `create()` method must be called after the model is constructed in order to create the model.

Parameter

`ccParams` the parameters of the call center.

Throws

`NullPointerException` if `ccParams` is `null`.

`public CallCenter (Simulator sim, CallCenterParams ccParams)`

Similar to `CallCenter (CallCenterParams)`, with the given simulator `sim`.

Methods

`public final Simulator simulator()`

Returns the simulator associated with this call center model. The simulator is used to schedule events, and obtain simulation time when necessary.

Returns the associated simulator.

`public final void setSimulator (Simulator sim)`

Sets the simulator of this model to `sim`. After this method is called, the model should be reset using the `create()` method.

Parameter

`sim` the new simulator.

Throws

`NullPointerException` if `sim` is `null`.

`public double getTime (Duration d)`

Converts the given duration `d` to a time in the default time unit. This method calls `Duration.getTimeInMillis (Date)` using the date returned by `getStartingDate()`. It then uses `TimeUnit.convert (double, TimeUnit, TimeUnit)` to convert the obtained time in milliseconds to the default unit given by `getDefaultUnit()`.

Parameter

`d` the duration to be converted.

Returns the duration in the default time unit.

`public double[] getTime (Duration... d)`

Constructs and returns an array whose elements correspond to the time durations in the given array, converted to the default time unit. This method constructs an array with the same length as `d`, and sets element `i` of the target array to the result of `getTime (Duration)` called with `d[i]`.

Parameter

`d` the array of durations to convert.

Returns the array of converted durations.

`public double[][] getTime (Duration[][] d)`

Similar to `getTime (Duration...)`, for a 2D array.

Parameter

`d` the 2D array of durations to convert.

Returns the 2D array of converted durations.

`public double getTime (XMLGregorianCalendar xgcal)`

Converts the time returned by `CallCenterUtil.getTimeInMillis (XMLGregorianCalendar)` to the default time unit returned by `getDefaultUnit()`.

Parameter

`xgcal` the XML gregorian calendar representing a time.

Returns the time in the default time unit.

`public double[] getTime (XMLGregorianCalendar... d)`

Similar to `getTime (Duration...)`, for an array of XML gregorian calendars.

Parameter

`d` the array of times to convert.

Returns the array of converted times.

`public double[][] getTime (XMLGregorianCalendar[][] d)`

Similar to `getTime (XMLGregorianCalendar...)`, for a 2D array.

Parameter

`d` the 2D array of times to convert.

Returns the 2D array of converted times.

`public TimeUnit getDefaultUnit()`

Returns the default unit used for this call center. This corresponds to the unit for simulation time, and for output such as waiting times, excess times, etc.

Returns the default unit.

`public double getPeriodDuration()`

Returns the duration of main periods, expressed in the default time unit returned by `getDefaultUnit()`.

Returns the period duration.

`public Date getStartingDate()`

Returns the date corresponding to the environment being modeled. This corresponds to the date at which the preliminary period begins, with time set to midnight. This corresponds to the current date, at which this object was created, if no date was given explicitly in parameters.

Returns the date corresponding to the considered environment.

`public double getStartingTime()`

Returns the starting time of the first main period, expressed in the default time unit.

Returns the starting time of the first main period.

`public Date getMainPeriodStartingDate (int mp)`

Returns the date corresponding to the beginning of the main period `mp`. This method adds the starting time, and `mp` times the period duration to the date returned by `getStarting-Date()`, and returns the resulting date.

Parameter

`mp` the index of the main period.

Returns the date of the main period.

`public boolean isHorizonSpanningDays()`

Determines if the horizon of this model spans multiple days, i.e., if the period duration times the number of periods is larger than 24 hours. This method determines the starting dates of the first and last main periods, using `getMainPeriodStartingDate (int)`, and returns `true` if and only if the two dates have different days according to the Gregorian calendar.

Returns the success indicator of the test.

`public int getNumMatricesOfAWT()`

Returns the number of sets of parameters for the service level given by the user in parameter file.

Returns the number of sets of parameters for the service level.

`public String getMatrixOfAWTName (int m)`

Returns the name of the matrix of acceptable waiting time with index `m`, or `null` if no name was given in the parameter file.

Parameter

`m` the index of the matrix.

Returns the name corresponding to the matrix, or `null`.

Throws

`ArrayIndexOutOfBoundsException` if `m` is negative or greater than or equal to the value returned by `getNumMatricesOfAWT()`.

```
public ServiceLevelParamReadHelper getServiceLevelParams
(int m)
```

Returns the set of parameters `m` for the service level.

Parameter

`m` the index of the set.

Returns the set of parameters.

```
public CallCenterParams getCallCenterParams()
```

Returns the call center parameters associated with this model.

Returns the associated call center parameters.

```
public RandomStreams getRandomStreams()
```

Returns the random streams used by this model.

Returns the random streams used.

```
@Deprecated public RandomStreams getStreams()
```

Returns the random streams associated with this simulator.

Returns the associated random streams.

Deprecated Use `getRandomStreams()` instead.

```
public void setRandomStreams (RandomStreams streams)
```

Sets the random streams used by this model to `streams`. This method calls `RandomStreams.createStreams (CallCenterParams)` on the call center parameters to create necessary streams.

Note that the new random streams are used only after `create()` is called.

Parameter

`streams` the new random streams.

Throws

`NullPointerException` if `streams` id `null`.

`public void resetNextSubstream()`

Calls `RandomStream.resetNextSubstream()` on every random stream of this model.

`public void resetStartStream()`

Calls `RandomStream.resetStartStream()` on every random stream of this model.

`public void resetStartSubstream()`

Calls `RandomStream.resetStartSubstream()` on each random stream of this model.

```
public void reset (CallCenterParams ccParams1, RandomStreams streams1)
                    throws CallCenterCreationException
```

Recreates the model with new parameters. This class sets the call center parameters to `ccParams`, the random streams to `streams`, and calls `create()` to recreate the model.

Parameters

`ccParams1` the new call center parameters.

`streams1` the new random streams.

Throws

`NullPointerException` if `ccParams` or `streams` are `null`.

`public void create() throws CallCenterCreationException`

Calls `create (false)`.

Throws

`CallCenterCreationException` if an error occurs during the creation of the model.

```
public void create (boolean recreateStreams) throws
                    CallCenterCreationException
```

Constructs the elements of the call center. This method is called by the constructor or by `reset (CallCenterParams, RandomStreams)`. If `recreateStreams` is `true`, a new `RandomStreams` object is created and associated with this model; this results in a change of seeds for every random stream used. If `recreateStreams` is `false`, the same random streams are kept, and new ones are created just if needed.

Since this method recreates the complete structure of the call center, any listener observing the evolution of the model must be re-registered after this method returns.

Parameter

`recreateStreams` determines if random streams are recreated.

Throws

`CallCenterCreationException` if an error occurs during the creation of the model.

`public DialerObjects getDialerObjects()`

Returns the instance of `DialerObjects` associated with this model. If no such instance exists, it is constructed, stored for future use, and returned.

Returns eht dialer objects of this model.

`protected WaitingQueue createWaitingQueue (int q)`

Constructs and returns the `q`th waiting queue for this call center. By default, this returns an instance of `StandardWaitingQueue` which is a FIFO queue without priority.

Parameter

`q` the index of the created waiting queue.

Returns the constructed waiting queue.

`public void initSim()`

Initializes the model for a new simulation with a random busyness factor. This method first generates the busyness factor B using the generator returned by `getBusynessGen()`, or sets $B = 1$ if no generator was given in parameter file for the busyness factor. It then calls `initSim (double)` with the generated B to complete initialization.

`public double getBusynessFactor()`

Returns the current value of B used by arrival processes.

Returns the current value of B.

`public void initSim (double b1)`

Initializes the model for a new simulation setting the busyness factor of arrival processes to the given value `b`. This method initializes arrival processes, dialers, agent groups, waiting queues, and the router, without scheduling any event. Methods such as `PeriodChangeEvent.start()`, `ContactArrivalProcess.start()`, etc. must be used to schedule events before starting the simulation.

Parameter

`b1` the busyness factor used.

`public boolean isCallTransferSupported()`

Determines if this model supports call transfers. This returns `true` if `CallFactory.isCallTransferSupported()` returns `true` for at least one call factory returned by `getCallFactories()`.

Returns `true` if and only if this model supports call transfers.

`public boolean isVirtualHoldSupported()`

Determines if this model supports virtual holding. This returns `true` if `CallFactory.isVirtualHoldSupported()` returns `true` for at least one call factory returned by `getCallFactories()`.

Returns `true` if and only if this model supports call virtual holding.

`public AWTPeriod getAwtPeriod()`

Returns the object used to compute the AWT period of contacts. This method returns `null` unless `setAwtPeriod (AWTPeriod)` was called with a non-`null` value.

Returns the object for AWT periods.

`public void setAwtPeriod (AWTPeriod awtPeriod)`

Sets the object for computing AWT periods to `awtPeriod`.

Parameter

`awtPeriod` the object for computing AWT periods.

`public int getAwtPeriod (Contact contact)`

Returns the period index used to obtain the period-specific acceptable waiting time for contact `contact`. If `getAwtPeriod()` returns `null`, this method returns the main period index of the contact's arrival. Otherwise, it returns the result of `getAwtPeriod().getAwtPeriod (contact)`.

Parameter

`contact` the contact to be tested.

Returns the AWT period of the contact.

`public PeriodChangeEvent getPeriodChangeEvent()`

Returns a reference to the period-change event used by this model. This event occurs at the beginning of each period of the horizon, and triggers updates of some parameters such as the staffing in agent groups. One should start this event using `PeriodChangeEvent.start()` to simulate the horizon, or use `PeriodChangeEvent.setCurrentPeriod (int)` to simulate a single period as if it was infinite in the model.

Returns the period-change event.

`@Deprecated public void setRouter (Router router)`

Deprecated Use `RouterManager.setRouter (Router)` instead.

`public Router getRouter()`

Returns a reference to the router used by this model. This method calls `getRouterManager().getRouter`

`public RouterManager getRouterManager()`

Returns a reference to the router manager of this model.

`public RandomVariateGen getBusynessGen()`

Returns a reference to the random variate generator used for the global busyness factor B multiplying the arrival rates or number of arrivals of calls. This method returns `null` if no busyness factor is used.

Returns the random variate generator for busyness factor.

`public void setBusynessGen (RandomVariateGen bgen)`

Sets the random variate generator for the global busyness factor to `bgen`.

Parameter

`bgen` the new random variate generator.

`public int getNumDialers()`

Returns the maximal number of dialer managers in this model. This corresponds to the number of outbound call types plus the number of dialers that can generate calls of several types.

Returns the number of dialers.

`public DialerManager getDialerManager (int k)`

Returns the dialer manager with index `k`. The first K_O dialers generate outbound calls of a single type while other dialers can generate calls of several types. This method returns `null` if k is smaller than K_O, and no dialer dedicated to calls of outbound type k exists.

Parameter

`k` the index of the dialer manager.

Returns the dialer manager.

Throws

`ArrayIndexOutOfBoundsException` if `k` is negative, or greater than or equal to the value returned by `getNumDialers()`.

`public Dialer getDialer (int k)`

Returns the dialer with index `k`, or `null` if k is smaller than K_O, and no dialer is dedicated to outbound calls of type `k`. This method calls `getDialerManager (int)` with the given value of `k`, and returns the dialer associated with the returned dialer manager.

Parameter

`k` the index of the dialer.

Returns the dialer, or `null`.

`public DialerManager[] getDialerManagers()`

Returns the array of dialer managers in this model. The first K_O elements of the returned array represent dialers dedicated to a single type of outbound call, and may be `null` if no dialer is dedicated to a given call type.

`public Dialer[] getDialers()`

Returns an array containing the dialers of this model. This method calls `getDialerManagers()`, and creates an array with each element k being the dialer associated with the dialer manager k. As with `getDialerManagers()`, some elements in the returned array might be `null`.

`public int getNumArrivalProcesses()`

Returns the maximal number of arrival process managers in this model. This corresponds to the number of inbound call types plus the number of arrival processes that can generate calls of several types.

Returns the number of arrival processes.

`public ArrivalProcessManager getArrivalProcessManager (int k)`

Returns the arrival process manager with index `k`. The first K_I arrival processes generate inbound calls of a single type while other processes can generate calls of several types. This method returns `null` if k is smaller than K_I, and no arrival process dedicated to calls of inbound type k exists.

Parameter

`k` the index of the arrival process manager.

Returns the arrival process manager.

Throws

`ArrayIndexOutOfBoundsException` if `k` is negative, or greater than or equal to the value returned by `getNumArrivalProcesses()`.

`public ContactArrivalProcess getArrivalProcess (int k)`

Returns the arrival process with index `k`, or `null` if k is smaller than K_I, and no arrival process is dedicated to inbound calls of type `k`. This method calls `getArrivalProcessManager (int)` with the given value of `k`, and returns the arrival process associated with the returned manager.

Parameter

`k` the index of the arrival process.

Returns the arrival process, or `null`.

`public ArrivalProcessManager[] getArrivalProcesManagers ()`

Returns the array of arrival process managers in this model. The first K_{I} elements of the returned array represent arrival processes dedicated to a single type of inbound call, and may be `null` if no arrival process is dedicated to a given call type.

`public ContactArrivalProcess[] getArrivalProcesses()`

Returns an array containing the arrival processes of this model. This method calls `getArrivalProcesManagers()`, and creates an array with each element k being the arrival process associated with the manager k. As with `getArrivalProcesManagers()`, some elements in the returned array might be `null`.

`public AgentGroupManager getAgentGroupManager (int i)`

Returns the agent group manager with index `i`.

Parameter

`i` the index of the agent group.

Returns the agent group manager.

Throws

`ArrayIndexOutOfBoundsException` if `i` is negative, or greater than or equal to the value returned by `getNumAgentGroups()`.

`public AgentGroup getAgentGroup (int i)`

Returns the agent group with index `i`. This method is equivalent to calling `getAgentGroupManager (int)` and using `AgentGroupManager.getAgentGroup()`.

Parameter

`i` the index of the agent group.

Returns the agent group.

Throws

`ArrayIndexOutOfBoundsException` if `i` is negative, or greater than or equal to the value returned by `getNumAgentGroups()`.

`public AgentGroupManager[] getAgentGroupManagers()`

Returns an array containing the agent group managers of this model.

`public AgentGroup[] getAgentGroups()`

Returns an array containing the agent groups of this model.

`public WaitingQueue[] getWaitingQueues()`

Returns an array containing the waiting queues of this model.

`public WaitingQueue getWaitingQueue (int q)`

Returns the waiting queue with index `q` in this model.

Parameter

`q` the index of the waiting queue.

Returns a reference to the waiting queue.

Throws

`ArrayIndexOutOfBoundsException` if `q` is negative, or greater than or equal to the value returned by `getNumWaitingQueues()`.

`public CallFactory[] getCallFactories()`

Returns the array of call factories for this model.

`public CallFactory getCallFactory (int k)`

Returns the call factory generating calls of type `k` in this model.

Parameter

`k` the index of the call type.

Returns a reference to the call factory.

Throws

`ArrayIndexOutOfBoundsException` if `k` is negative, or greater than or equal to the value returned by `getNumContactTypes()`.

`public SegmentInfo[] getInContactTypeSegments()`

Returns an array containing information objects for all user-defined segments regrouping inbound contact types.

`public SegmentInfo getInContactTypeSegment (int k)`

Returns the information object for the `k`th user-defined segment regrouping inbound contact types.

Parameter

`k` the index of the user-defined segment.

Returns the segment information object.

Throws

`ArrayIndexOutOfBoundsException` if `k` is negative, or greater than or equal to `getNumInContactTypeSegments()`.

`public SegmentInfo[] getOutContactTypeSegments()`

Returns an array containing information objects for all user-defined segments regrouping outbound contact types.

`public SegmentInfo getOutContactTypeSegment (int k)`

Returns the information object for the `k`th user-defined segment regrouping outbound contact types.

Parameter

`k` the index of the user-defined segment.

Returns the segment information object.

Throws

`ArrayIndexOutOfBoundsException` if `k` is negative, or greater than or equal to `getNumOutContactTypeSegments()`.

`public SegmentInfo[] getContactTypeSegments()`

Returns an array containing information objects for all user-defined segments regrouping contact types.

`public SegmentInfo getContactTypeSegment (int k)`

Returns the information object for the `k`th user-defined segment regrouping contact types.

Parameter

`k` the index of the user-defined segment.

Returns the segment information object.

Throws

`ArrayIndexOutOfBoundsException` if `k` is negative, or greater than or equal to `getNumContactTypeSegments()`.

`public SegmentInfo[] getAgentGroupSegments()`

Returns an array containing information objects for all user-defined segments regrouping agent groups.

`public SegmentInfo getAgentGroupSegment (int i)`

Returns the information object for the `i`th user-defined segment regrouping agent groups.

Parameter

`i` the index of the user-defined segment.

Returns the segment information object.

Throws

`ArrayIndexOutOfBoundsException` if `i` is negative, or greater than or equal to `getNumAgentGroups()`.

`public SegmentInfo[] getMainPeriodSegments()`

Returns an array containing information objects for all user-defined segments regrouping main periods.

`public SegmentInfo getMainPeriodSegment (int p)`

Returns the information object for the `p`th user-defined segment regrouping main periods.

Parameter

`p` the index of the user-defined segment.

Returns the segment information object.

Throws

`ArrayIndexOutOfBoundsException` if `p` is negative, or greater than or equal to `getNumMainPeriods()`.

`public double getArrivalsMult()`

Returns the global multiplier applied to the arrival rates or number of arrivals for each arrival process in this model.

Returns the global multiplier for arrivals.

`public double getPatienceTimesMult()`

Returns the global multiplier for patience times which is applied on every generated patience time.

Returns the global multiplier for patience times.

`public double getServiceTimesMult()`

Returns the global multiplier for service times which is applied on every generated service time.

Returns the global multiplier for service times.

`public double getConferenceTimesMult()`

Returns the global multiplier for conference times of calls transferred to a new agent with the primary agent waiting for the secondary agent.

Returns the global multiplier for conference times.

`public double getPreServiceTimesNoConfMult()`

Returns the global multiplier for pre-service times of calls transferred to a new agent without the primary agent waiting for the secondary agent.

Returns the global multiplier for pre-service times.

`public double getTransferTimesMult()`

Returns the global multiplier applied on any generated transfer time.

Returns the global multiplier for transfer times.

`public double getPreviewTimesMult()`

Returns the global multiplier applied to all generated preview times of outbound calls.

Returns the global multiplier for preview times.

`public double getAgentsMult()`

Returns the global multiplier for the number of agents in any group during any period.

Returns the global multiplier for staffing.

`public void setArrivalsMult (double arrivalsMult)`

Sets the global multiplier for arrivals to `arrivalsMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`arrivalsMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setPatienceTimesMult (double patienceTimesMult)`

Sets the global multiplier for patience times to `patienceTimesMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`patienceTimesMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setServiceTimesMult (double serviceTimesMult)`

Sets the global multiplier for service times to `serviceTimesMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`serviceTimesMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setConferenceTimesMult (double conferenceTimesMult)`

Sets the global multiplier for conference times to `conferenceTimesMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`conferenceTimesMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setPreServiceTimesNoConfMult (double preServiceTimesNoConfMult)`

Sets the global multiplier for pre-service times to `preServiceTimesNoConfMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`preServiceTimesNoConfMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setTransferTimesMult (double transferTimesMult)`

Sets the global multiplier for transfer times to `transferTimesMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`transferTimesMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setPreviewTimesMult (double previewTimesMult)`

Sets the global multiplier for preview times to `previewTimesMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`previewTimesMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public void setAgentsMult (double agentsMult)`

Sets the global multiplier for the number of agents to `agentsMult`. This multiplier takes effect only after the next call to `initSim()`.

Parameter

`agentsMult` the new multiplier.

Throws

`IllegalArgumentException` if the given multiplier is negative.

`public boolean[][] getDefaultShiftMatrix()`

Returns the $J \times P$ default shift matrix used for any agent group with a schedule giving only a vector of agents. Here, J is the number of shifts in the matrix. Element (j, p) of the returned matrix is `true` if and only if agents on shift j work during main period p.

Returns the default matrix of shifts.

`public void resetAgentsMult()`

Sets the multiplier returned by `getAgentsMult()` to 1, and adjusts the multipliers for each agent group. Let m be the multiplier returned by `getAgentsMult()` before this method is called. This method changes the multiplier for each agent group i from m_i to $m * m_i$, and resets the global multiplier m to 1.

`public Class<? extends WaitingTimePredictor> getWaitingTimePredictorClass ()`

Returns the class of waiting time predictors used by some routing policies, and virtual holding.

Returns the class of predictor for waiting times.

`public int getQueueCapacity()`

Returns the current queue capacity in this model. This corresponds to the maximal total number of calls in queue at any time during the simulation,. An infinite queue capacity is represented by `Integer.MAX_VALUE`.

Returns the total queue capacity.

`public void setQueueCapacity (int q)`

Sets the total queue capacity to `q`.

Parameter

`q` the new queue capacity.

Throws

`IllegalArgumentException` if the given queue capacity is smaller than the current total number of calls in queue.

`public boolean isExponentialPatienceTime (int k, int mp)`

Determines if patience times for contacts of type `k` arrived during period `mp` are exponential, and returns the result of the test.

Parameters

`k` the tested contact type.

`mp` the tester arrival period.

Returns `true` if and only if patience times are exponential.

`public boolean[][] isExponentialPatienceTime()`

Returns an array containing `true` at position `[k][p]` if contacts of type `k` arrived during period `p` have exponential patience times.

Returns the status of patience times for all contact types and periods.

`public boolean isExponentialServiceTime (int k, int i, int mp)`

Determines if service times for contacts of type `k` arrived during period `mp`, and served by agents in group `i` are exponential, and returns the result of the test.

Parameters

`k` the tested contact type.

`i` the tested agent group.

`mp` the tested arrival period.

Returns `true` if and only if service times are exponential.

`public boolean[][][] isExponentialServiceTime()`

Returns an array containing `true` at position `[k][i][p]` if contacts of type `k` arrived during period `p`, and served by agents in group `i` have exponential service times.

Returns the status of service times for all contact types and periods.

`public Map<String, Object> getProperties()`

Returns a map containing the user-defined properties associated with this model.

Returns the map of user-defined properties.

CallCenterUtil

Provides helper static methoeds used for the initialization of call center models.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallCenterUtil
```

Methods

`public static int[] getIntArray (int[] array, int numPeriods)`

Constructs and returns an array containing `numPeriods` elements from the input array `array`. If the length of the given array is 0, this returns an empty array. Otherwise, if the length is 1, this returns an array of length `numPeriods` filled with `array[0]`. Otherwise, an array with the first `numPeriods` elements of `array` is constructed and returned.

Parameters

`array` the input array.

`numPeriods` the number of elements in the output array.

Returns the output array.

Throws

`IllegalArgumentException` if the given array is too short.

`public static double[] getDoubleArray (double[] array, int numPeriods)`

Similar to `getIntArray (int[], int)`, for an array of double-precision values.

Parameters

`array` the input array.

`numPeriods` the number of elements in the output array.

Returns the output array.

Throws

`IllegalArgumentException` if the given array is too short.

`public static GregorianCalendar getDate (XMLGregorianCalendar xgcal)`

Converts the given XML gregorian calendar into a Java gregorian calendar, with time reset to midnight relative to the timezone given in the XML gregorian calendar. If `xgcal` does not specify a timezone offset, the default offset of the system is used. If `xgcal` is `null`, the current date is used.

This method first creates a Java `GregorianCalendar` by using `XMLGregorianCalendar.toGregorianCalendar()` (this uses the default timezone offset if no offset was specified explicitly), or the no-argument constructor of `GregorianCalendar` if `xgcal` is `null` (this creates a calendar initialized to the current date and time). It then resets the time fields of the created calendar to midnight before returning it.

Parameter

`xgcal` the XML gregorian calendar to be converted to a date.

Returns the gregorian calendar representing the date.

`public static long getTimeInMillis (XMLGregorianCalendar xgcal)`

Returns the time duration, in milliseconds, elapsed between midnight and the time given by `xgcal`, at the date set by `xgcal`. This method uses `XMLGregorianCalendar.toGregorianCalendar (TimeZone, Locale, XMLGregorianCalendar)` with a default timezone corresponding to GMT, the default locale, and no default XML gregorian calendar. It then clears all fields of the resulting calendar corresponding to date components, and returns `Calendar.getTimeInMillis()`. If `xgcal` is `null`, this returns 0.

Parameter

`xgcal` the XML gregorian calendar.

Returns the time in milliseconds.

`public static String getCallTypeInfo (CallCenterParams ccParams, int k)`

Returns information about a call type `k` defined in call center parameters `ccParams`. This method returns a string of the form `call type k (name)` which is included in some error messages.

Parameters

`ccParams` the call center parameters.

`k` the index of the call type.

Returns the string representation for the call type.

`public static String getAgentGroupInfo (CallCenterParams ccParams, int i)`

Similar to `getCallTypeInfo (CallCenterParams, int)`, for agent group `i`. This method returns a string of the form `agent group i (name)` included in some error messages.

Parameters

`ccParams` the call center parameters.

`i` the index of the agent group.

Returns the string representation of the agent group.

`public static <K> Map<K, String> toStringValues (Map<? extends K, ? extends Object> map)`

Constructs and returns a map for which each entry (k, v') is created from entry (k, v) in map `map`, where k is a key, and v' is the string representation of the value v. The string representation of v is the string "`null`" if v is `null`, or the result of `v.toString()` if v is non-`null`.

Type parameter

`K` the type of keys in the maps.

Parameter

`map` the source map.

Returns the map with string representations as values.

MakeAgentAvailableEvent

Represents an event occuring when a disconnected agent becomes available again.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public final class MakeAgentAvailableEvent extends Event
```

Constructor

`public MakeAgentAvailableEvent (CallCenter model, Agent agent)`

Constructs an event making the agent `agent` in the model `model` available when it occurs.

Parameters

`model` the model the agent belongs to.

`agent` the agent that will be made available.

Methods

`public CallCenter getCallCenter()`

Returns the model associated with this event.

Returns the associated model.

`public Agent getAgent()`

Returns the agent associated with this event.

Returns the associated agent.

CallFactoryStreamType

Types of random streams for call factories.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public enum CallFactoryStreamType
```

Constants

`BALKTEST`
: Random stream for immediate abandonment.

`PATIENCE`
: Random stream for patience time, for contacts not abandoning immediately.

`SERVICE`
: Random stream for service time.

ArrivalProcessStreamType

Types of random streams for arrival processes.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public enum ArrivalProcessStreamType
```

Constants

`INTERARRIVAL`
: Stream for inter-arrival times.

`RATES`
: Stream for random arrival rates, in the case of doubly-stochastic arrival processes.

DialerStreamType

Types of random streams for dialers.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public enum DialerStreamType
```

Constants

`DIALDELAY`
: Random stream for dialing delays.

`REACHTEST`
: Random stream for testing if a call is reached or has failed.

AgentGroupStreamType

Types of random streams for agent groups.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public enum AgentGroupStreamType
```

Constants

`DISCONNECTTEST`
: Random stream for probability that an agent disconnects after some event occurs.

`DISCONNECTTIME`
: Random stream for the time an agent remains offline after it disconnects.

RandomStreams

Encapsulates the random streams used by the blend/multi-skill call center simulator. The model uses one random stream for each type of random variate for better synchronization when using common random numbers. This class creates, stores, and manages all these random streams.

Often, this class is not used directly since the `CallCenter` class provides a constructor which implicitly creates the random streams. However, it can be useful to get the `RandomStreams` object of a model, using the `CallCenter.getRandomStreams()` method, in order to retrieve the reference to a particular random stream, or to pass the random streams to a new model. Creating several models with the same random streams can improve synchronization when comparing systems with common random numbers.

However, if several instances of `CallCenter` are used in parallel, each instance should have its own random streams. The `clone()` method can be used if seeds must be shared between two instances of this class.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class RandomStreams implements Cloneable
```

Fields

`public static final int NUMFACTORYSTREAMS`

Number of random streams for a contact factory.

`public static final int NUMFACTORYSTREAMS2`

Number of random streams for a contact factory.

`public static final int NUMAPSTREAMS`

Number of random streams for arrival processes.

`public static final int NUMDIALERSTREAMS`

Number of streams for dialers.

`public static final int NUMAGENTGROUPSTREAMS`

Number of streams for agent groups.

Constructor

`public RandomStreams (RandomStreamFactory rsf, CallCenterParams ccParams)`

Creates a new set of random streams using the random stream factory `rsf`, and the call center parameters `ccParams`. The parameters are used to determine the number of call types, agent groups, etc., in order to set the number of streams of each type to create.

This method sets the random stream factory returned by `getRandomStreamFactory()`, and calls `createStreams (CallCenterParams)`.

Parameters

`rsf` the random stream factory used to create each `RandomStream` instance.

`ccParams` the parameters of the call center for which random streams are created.

Throws

`NullPointerException` if `rsf` or `ccParams` are `null`.

Methods

`public void createStreams (CallCenterParams ccParams)`

Creates the necessary random streams for supporting $K = K_\mathrm{I} + K_\mathrm{O}$ contact types, and I agent groups. This method reuses every stream associated with this object; it only creates new streams when needed. Consequently, it cannot be used to set new random seeds for every stream. Setting new seeds can be done by constructing a new `RandomStreams` instance.

Parameter

`ccParams` the parameters of the call center.

Throws

`NullPointerException` if `ccParams` is `null`.

`public RandomStreamFactory getRandomStreamFactory()`

Returns the random stream factory used by the `createStreams (CallCenterParams)` method of this object to create random streams.

Returns the associated random stream factory.

`public void setRandomStreamFactory (RandomStreamFactory rsf)`

Sets the associated random stream factory to `rsf`. The new factory will only affect streams created by subsequent calls to `createStreams (CallCenterParams)`, not already created streams.

Parameter

`rsf` the new random stream factory.

Throws

`NullPointerException` if `rsf` is `null`.

`public Set<RandomStream> getRandomStreamsInit()`

Returns the set regrouping random streams used during the initialization of replications only. Streams in this set can, for example, set the busyness factor for the day, the total (random) number of arrivals, etc.

Returns the set of random streams used for initialization.

`public Set<RandomStream> getRandomStreamsSim()`

Returns the set of random streams regrouping random streams used during the whole simulation. These streams may, for example, generate inter-arrival, patience, and service times.

Returns the set of random streams.

`public RandomStream getStreamCT()`

Returns the random stream used for generating contact type indices while the system is initialized non-empty, for a simulation on an infinite horizon using batch means.

Returns the random stream for contact type indices.

`public RandomStream getStreamB()`

Returns the random stream used for the global busyness factor. This stream is used only at the beginning of a replication, for a finite-horizon simulation, if a distribution was given for the busyness factor B of the day.

Returns the random stream used for the global busyness factor.

`public RandomStream getCallFactoryStream (int k, CallFactoryStreamType s)`

Returns the random stream of type `s` used by the contact factory with index `k`.

Parameters

`k` the index of the call factory.

`s` the type of the stream.

Returns the random stream.

`public RandomStream getCallFactoryStream2 (int k, CallFactoryStreamType2 s)`

Similar to `getCallFactoryStream (int, CallFactoryStreamType)`, for a complementary set of random streams. These streams, used for call transfer and virtual queueing, were added at a later time, so a second set was used to avoid changing the seeds of other streams.

Parameters

`k` the index of the call factory.

`s` the type of the complementary stream.

Returns the random stream.

```
public RandomStream getArrivalProcessStream (int ki,
                                             ArrivalProcessStreamType s)
```

Returns the random stream of type `s` used by the arrival process with index `ki`.

Parameters

`ki` the index of the arrival process.

`s` the type of the stream.

Returns the random stream.

```
public RandomStream getArrivalProcessPStream (int ki)
```

Returns the random stream used to select generated call type for the *ki*-th arrival process generating calls of multiple types.

Parameter

`ki` the index of the arrival process.

Returns the random stream.

```
public RandomStream getDialerStream (int ko, DialerStreamType s)
```

Returns the random stream of type `s` used by the dialer with index `ko`.

Parameters

`ko` the index of the dialer.

`s` the type of the stream.

Returns the random stream.

```
public RandomStream getDialerPStream (int ko)
```

Returns the random stream used to select generated call type for the *ko*-th dialer generating calls of multiple types.

Parameter

`ko` the index of the dialer.

Returns the random stream.

```
public RandomStream getAgentGroupStream (int i, AgentGroupStreamType s)
```

Returns the random stream of type `s` used by the agent group `i`.

Parameters

`i` the index of the agent group.

`s` the type of the stream.

Returns the random stream.

`public RandomStream getStreamAgentSelection()`

Returns the random stream used for agent selection during routing, if agent selection is randomized.

Returns the random stream used for agent selection.

`public RandomStream getStreamContactSelection()`

Returns the random stream used for contact selection during routing, if contact selection is randomized.

Returns the random stream used for contact selection.

`public RandomStreams clone()`

Creates a clone of this object and all the contained random streams. This method creates a copy of this object, and clones every random stream by casting them to `CloneableRandomStream` and calling `clone()`. Each generator in the cloned object has the same properties and seeds as the corresponding generator in the original object.

Throws

`ClassCastException` if at least one encapsulated random stream does not implement the `CloneableRandomStream` interface.

Call

Represents a call in the multi-skill call center simulator. A call is a special type of contact that stores the periods of its arrival, of its service startup and its service termination. These periods can be stored, because the model uses a single period-change event. A call also holds additional information such as transfer times, conference times, etc.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class Call extends Contact
```

Constructors

```
public Call (PeriodChangeEvent pce, int arrivalPeriod)
```

Equivalent to `Call (pce, arrivalPeriod, 1, 0)`.

Parameters

`pce` the period-change event associated with the call.

`arrivalPeriod` the period of arrival of the call.

Throws

`NullPointerException` if `pce` is `null`.

```
public Call (PeriodChangeEvent pce, int arrivalPeriod, int typeId)
```

Equivalent to `Call (pce, arrivalPeriod, 1, typeId)`.

Parameters

`pce` the period-change event associated with the call.

`arrivalPeriod` the period of arrival of the call.

`typeId` the type identifier of the call.

Throws

`NullPointerException` if `pce` is `null`.

```
public Call (PeriodChangeEvent pce, int arrivalPeriod, double priority,
             int typeId)
```

Constructs a new call with period-change event `pce`, period of arrival `arrivalPeriod`, priority `priority`, and type identifier `typeId`. The period-change event is used to set the simulator associated with the call, and to determine periods of service termination or abandonment.

Parameters

`pce` the period-change event associated with the call.

`arrivalPeriod` the period of arrival of the call.

`priority` the priority of the call.

`typeId` the type identifier of the call.

Throws

`NullPointerException` if `pce` is `null`.

Methods

`public int getArrivalPeriod()`

Returns the period during which this call has arrived. This corresponds to the period during which the call object was constructed.

Returns the period during which the call arrived.

`public void setArrivalPeriod (int arrivalPeriod)`

Sets the period of arrival of this call to `arrivalPeriod`.

Parameter

`arrivalPeriod` the new period of arrival.

`public int getBeginServicePeriod()`

Returns the period at which the service of this call started, or -1 if this call was never served.

Returns the period at which the service of this call began.

`public void setBeginServicePeriod (int beginServicePeriod)`

Sets the period at which the service of this call begins to `beginServicePeriod`.

Parameter

`beginServicePeriod` the period at which the service of this call begins.

`public int getExitPeriod()`

Returns the period at which this call exited the system, or -1 if the call is still in the system.

Returns the period at which this call exited the system.

`public void setExitPeriod (int exitPeriod)`

Sets the period at which this call exits the system to `exitPeriod`.

Parameter

`exitPeriod` the period at which this call exits the system.

`public PeriodChangeEvent getPeriodChangeEvent()`

Returns the period-change event used to initializes the period at which the service begins, and at which this call exits.

Returns the period-change event.

`public void setPeriodChangeEvent (PeriodChangeEvent pce)`

Sets the period-change event of this call to `pce`.

Parameter

`pce` the period-change event associated with this call.

Throws

`NullPointerException` if `pce` is `null`.

`public boolean isRightPartyConnect()`

Determines if this call is a right party connect. By default, this method returns `true`, but for outbound calls, `OutboundCallFactory` can set this flag to `false` in order to generate a wrong party connect. This differs from a failed call, which is handled by the dialer itself, because an agent is needed to screen the call. The main use of the returned value is for statistical collecting.

Returns `true` if and only if this call is a right party connect, or an inbound call.

`public void setRightPartyConnect (boolean rightPartyConnect)`

Sets the indicator for right party connect to `rightPartyConnect`.

Parameter

`rightPartyConnect` the new value of the indicator.

See also `isRightPartyConnect()`

`public EndServiceEvent getPrimaryEndServiceEvent()`

If this object represents a transferred call, returns a reference to the end-service event representing the service with the primary agent, before the transfer. This end-service event is used to terminate the service with the primary agent after a conference time. This returns `null` if this object does not represent a transferred call, or if the primary agent does not wait for a secondary agent after the transfer.

Returns the end-service event associated with the primary agent for a transferred call.

`public void setPrimaryEndServiceEvent (EndServiceEvent primaryEndServiceEvent)`

Sets the end-service event associated with the primary agent for a transferred call to `primaryEndServiceEvent`.

Parameter

`primaryEndServiceEvent` the new end-service event.

`public double getUTransfer()`

Returns the random number used to test if a call is transferred after its service is over. This uniform is initialized by the call factory if call transfers are supported. Otherwise, it is set to 0.

Returns the uniform for deciding if the call is transferred.

`public void setUTransfer (double transfer)`

Sets the uniform for transfer decision to `transfer`.

Parameter

`transfer` the new uniform.

Throws

`IllegalArgumentException` if `transfer` is out of $[0, 1]$.

See also `getUTransfer()`

`public double getUTransferWait()`

Returns the uniform used to decide if the primary agent waits for a secondary agent after a transfer. This uniform is generated by the call factory only if call transfers are supported. If transfers are disabled, this method always returns 0.

Returns the uniform for deciding if the primary agent waits for the secondary agent with the caller.

`public void setUTransferWait (double transferWait)`

Sets the uniform for deciding if the primary agent waits for a secondary agent to `transferWait`.

Parameter

`transferWait` the new uniform.

Throws

`IllegalArgumentException` if `transferWait` is out of $[0, 1]$.

See also `getUTransferWait()`

`public double getUVQ()`

Returns the uniform used to decide if a call accepts to be called back (or join the virtual queue) if offered the possibility. This uniform is generated by the call factory if virtual queueing is used. If virtual queueing is disabled, this method always returns 0.

Returns the uniform for virtual queueing decision.

`public void setUVQ (double u)`

Sets the uniform for deciding if a call chooses to be called back to `u`.

Parameter

`u` the new uniform.

Throws

`IllegalArgumentException` if `u` is not in $[0, 1]$.

See also `getUVQ()`

`public double getUVQCallBack()`

Returns the uniform used to decide if a call returning from the virtual queue is successfully called back. This uniform is generated by the call factory, and is always 0 if virtual queueing is disabled.

Returns the uniform for call back success.

`public void setUVQCallBack (double u)`

Sets the uniform for call back success to `u`.

Parameter

`u` the new uniform.

Throws

`IllegalArgumentException` if `u` is not in $[0, 1]$.

See also `getUVQCallBack()`

`public ServiceTimes getConferenceTimes()`

Returns the conference times spent by a primary agent with a secondary before the service of this transferred call begins with the secondary agent. By default, this is set to 0. This time is set by the call factory if call transfers are enabled.

Returns an object storing the conference times.

`public ServiceTimes getPreServiceTimesNoConf()`

Returns the pre-service times with an agent. By default, this is set to 0.

Returns an object storing pre-service times.

`public ServiceTimes getTransferTimes()`

Returns the transfer times spent by primary agents to initiate call transfers. By default, this is set to 0.

Returns an object storing transfer times.

`public double getWaitingTimeVQ()`

Returns the time spent in virtual queue by this call. If virtual queueing is disabled, this method always returns 0.

Returns the waiting time of this call in virtual queue.

`public void setWaitingTimeVQ (double waitingTimeVQ)`

Sets the waiting time in virtual queue of this call to `waitingTimeVQ`.

Parameter

`waitingTimeVQ` the new waiting time in virtual queue.

`public int getTypeBeforeVQ()`

Returns the type of this call before entering virtual queue.

Returns the type identifier of this call before entering virtual queue.

`public void setTypeBeforeVQ (int beforeVQ)`

Sets the type of this call before it enters virtual queue to `beforeVQ`.

Parameter

`beforeVQ` the original type of this call.

CallFactory

Contact factory used to create the calls for the simulator, and to generate call-specific random variates such as patience times and service times. The call factory also contains any information related to call types, such as name, properties, and the probability distribution for patience and service times.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallFactory extends SingleTypeContactFactory
```

Constructor

```
public CallFactory (CallCenter cc, CallCenterParams ccParams,
                    CallTypeParams par, int k) throws
                    CallFactoryCreationException
```

Constructs a new call factory using the call center `cc`, the call center parameters `ccParams`, the call-type parameters `par`, and call type index `k`.

Parameters

`cc` the call center.

`ccParams` the call center parameters.

`par` the call-type parameters.

`k` the call type index.

Throws

`CallFactoryCreationException` if an exception occurs during the creation of the factory.

Methods

```
public void initTransferTargets (CallCenterParams ccParams, int k) throws
                                 CallFactoryCreationException
```

Constructs a contact factory used to generate calls resulting from transfers after service termination. This initialization is not included in the constructor, because all call factories must be created before the contact factory for transfered calls is constructed.

Parameters

`ccParams` the parameters of the call center.

`k` the identifier of the call type.

Throws

`CallFactoryCreationException` if an error occurs during the creation of the call factory.

`public CallCenter getCallCenter()`

Returns the call center object containing this call factory.

Returns the call center object for this factory.

`public void init()`

Initializes this call factory by setting the multipliers for patience and service times.

`public double getPatienceTimesMult()`

Returns the current multiplier for patience times for calls generated by this factory. Patience times are multiplied by this constant and the multiplier returned by `CallCenter.getPatienceTimesMult()`. The default value of this multiplier is 1.

Returns the multiplier for patience times.

`public void setPatienceTimesMult (double pgenMult)`

Sets the multiplier for patience times to `pgenMult`.

Parameter

`pgenMult` the new multiplier for patience times.

Throws

`IllegalArgumentException` if `pgenMult` is negative.

`public ServiceTimesManager getServiceTimesManager()`

Returns an object managing the random variate generators for regular service times.

Returns the service times manager.

`public ServiceTimesManager getConferenceTimesManager()`

Returns an object managing the random variate generators for conference times between primary and secondary agents.

Returns the conference times manager.

`public ServiceTimesManager getPreServiceTimesNoConfManager ()`

Returns an object managing the random variate generators for pre-service times with secondary agents if no conference with primary agents.

Returns the pre-service times manager.

`public ServiceTimesManager getTransferTimesManager()`

Returns an object managing the random variate generators for transfer times.

Returns the transfer times manager.

```
public static CallFactory create (CallCenter cc, CallCenterParams ccParams,
                                  int k) throws
                                  CallFactoryCreationException
```

Constructs a new call factory using call center `cc`, call center parameters `ccParams`, and call type index `k`. This returns an instance of this class for inbound call types, or an instance of `OutboundCallFactory` for outbound types. This method calls `init()` on the constructed factory before returning it.

Parameters

`cc` the call center model.

`ccParams` the call center parameters.

`k` the index of the call type.

Returns the constructed call factory.

Throws

`CallFactoryCreationException` if an exception occurs during the creation of the factory.

```
public String getName()
```

Returns the name of the call type associated with this call factory.

Returns the name of the call type.

```
public Map<String, Object> getProperties()
```

Returns the user-defined properties of the call type associated with this call factory.

Returns the user-defined properties of the call type.

```
public double getWeight()
```

Returns the default weight used when no per-period weight is available for the call type associated with this call factory.

Returns the weight of the call type.

```
public double getWeight (int mp)
```

Returns the weight of the associated call type during main period `mp`, or the result of `getWeight()` if no per-period weight was given.

Parameter

`mp` the index of the main period.

Returns the weight of the call type during the given main period.

```
public double getProbAbandon (int mp)
```

Returns the probability of balking for main period `mp`.

Parameter

`mp` the index of the main period.

Returns the probability of balking.

`public double getProbTransfer (int i, int mp)`

Returns the probability of transfer for a call of the associated type arrived during main period `mp`, and whose service finishes with a primary agent in group `i`.

Parameters

`i` the index of the agent group.

`mp` the index of the main period.

Returns the probability of transfer.

`public double getProbTransferWait (int i, int mp)`

Returns the probability of a primary agent waiting for transfer to finish, for a call of the associated type arrived during main period `mp`, and whose service finishes with a primary agent in group `i`.

Parameters

`i` the index of the agent group.

`mp` the index of the main period.

Returns the probability of waiting for transfer.

`public double getServiceTimesMultTransfer (int i, int mp)`

Returns the multiplier for service times of callers arrived during main period `mp`, and served by an agent in group `i` before a transfer to another agent occurs.

Parameters

`i` the index of the agent group.

`mp` the index of the main period.

Returns the service times multiplier.

`public boolean isCallTransferSupported()`

Determines if call transfer is supported by this call factory. This returns `true` if and only if `getProbTransfer (int, int)` returns a non-zero value for at least one pair (i, p), and `getTransferTargetFactory()` returns a non-`null` value.

`public boolean isVirtualHoldSupported()`

Determines if virtual holding (or virtual queueing) is supported for the associated call type. This returns `true` if and only if `getExpectedWaitingTimeThresh (int)` returns a finite value for at least one p, `getProbVirtualQueue (int)` returns a non-zero value for at least one p, and `getTargetVQType()` returns a non-negative value.

`public ContactFactory getTransferTargetFactory()`

Returns the contact factory used to generate transferred calls from calls of the associated type.

Returns the contact factory for transferred calls.

`public double getPatienceTimesMultNoVirtualQueue (int mp)`

Returns the multiplier for patience times for callers arrived during main period `mp`, and deciding not to join the virtual queue. The default multiplier is 1.

Parameter

`mp` the main period of arrival.

Returns the multiplier of the patience times.

`public double getPatienceTimesMultCallBack (int mp)`

Returns the multiplier of patience times for calls arriving during main period `mp`, joining the virtual queue, successuflly called back, and having to wait in regular queue. The default multiplier is 1.

Parameter

`mp` the main period of arrival.

Returns the multiplier of the patience times.

`public double getServiceTimesMultNoVirtualQueue (int i, int mp)`

Returns the multiplier of service times for callers arrived during main period `mp`, deciding not to join the virtual queue, and served by an agent in group `i`.

Parameters

`i` the index of the agent group.

`mp` the index of the main period.

Returns the service times multiplier.

`public double getServiceTimesMultCallBack (int i, int mp)`

Returns the multiplier of service times for callers arrived during main period `mp`, and served by an agent in group `i` after being called back.

Parameters

`i` the index of the agent group.

`mp` the index of the main period.

Returns the service times multiplier.

`public void multiplyServiceTimesNoVirtualQueue (Call call)`

Applies the multipliers returned by `getServiceTimesMultNoVirtualQueue (int, int)` to the given call `call`. This changes the service times returned by `Contact.getContact-Times()` for the given call.

Parameter

`call` the call whose service times are modified.

`public void multiplyServiceTimesCallBack (Call call)`

Similar to `multiplyServiceTimesNoVirtualQueue (Call)`, but using multipliers returned by `getServiceTimesMultCallBack (int, int)`.

Parameter

`call` the call whose service times are modified.

`public int getTargetVQType()`

Returns the index of the call type calls entering virtual queue are changed to.

Returns the target call type for virtual queueing.

`public void setTargetVQType (int targetVQType)`

Sets the target call type for virtual queueing to `targetVQType`.

Parameter

`targetVQType` the new target type.

`public double getProbVirtualQueue (int mp)`

Returns the probability that a caller arriving during main period `mp` accepts to enter virtual queue, and be called back later.

Parameter

`mp` the main period of arrival.

Returns the probability of entering virtual queue.

`public double getProbVirtualQueueCallBack (int mp)`

Returns the probability that a caller arriving during main period `mp` is successfully called back after joining the virtual queue.

Parameter

`mp` the main period of arrival.

Returns the probability of successful call back.

`public double getExpectedWaitingTimeThresh (int mp)`

Returns the threshold on the expected waiting time for determining if a caller arrived during main period `mp` has the possibility to be called back.

Parameter

`mp` the main period of arrival.

Returns the threshold on the expected waiting time.

`public double getExpectedWaitingTimeMult (int mp)`

Returns the multiplier for the expected waiting time used to determine the time spent by a caller arriving during main period `mp` in the virtual queue.

Parameter

`mp` the main period of arrival.

Returns the waiting time multiplier.

`public void setConferenceTimes (Call call)`

Generates conference times for the given call `call`, and adds these conference times to the regular service times.

Parameter

`call` the call being processed.

`public void setPreServiceTimesNoConf (Call call)`

Similar to `setConferenceTimes (Call)`, for pre-service times in the case when no conference occurs.

Parameter

`call` the call being processed.

`public MultiPeriodGen getPatienceTimeGen()`

Returns the patience time, converted to `MultiPeriodGen`. Note that calling `MultiPeriodGen.setMult (double)` on the returned instance is not recommended as the multipliers are reset by `init()`. One should use `setPatienceTimesMult (double)` or `CallCenter.setPatienceTimesMult (double)` to alter the multipliers of the patience times.

`public MultiPeriodGen getServiceTimeGen()`

Returns the random variate generator for the default service times used when no agent group specific service times are available.

Note that it is not recommended to use `MultiPeriodGen.setMult (double)` on the returned object. One should alter the multipliers provided by `getServiceTimesManager()` instead.

Returns the service time generator.

`public MultiPeriodGen[] getServiceTimeGenGroups()`

Similar to `SingleTypeContactFactory.getContactTimeGenGroups()`, but returns an array of `MultiPeriodGen` instead. The same note for multipliers as in method `getServiceTimeGen()` applies here.

Returns the array of service times.

`public boolean isDisableCallSource()`

Determines if calls of the associated type can be produced using a call source, e.g., an arrival process or a dialer. By default, this returns `false` for regular call types, and `true` for call types corresponding to transfer or virtual queueing targets.

`public void setDisableCallSource (boolean disableCallSource)`

Sets the indicator for disabled call source to `disableCallSource`.

See also `isDisableCallSource()`

`public boolean isExcludedFromStatTotal()`

Determines if calls of the associated type are excluded from the totals in statistical reports. By default, this returns `false` for regular call types, and `true` for call types corresponding to transfer or virtual queueing targets.

`public void setExcludedFromStatTotal (boolean excludedFromStatTotal)`

Sets the indicator for exclusion in totals to `excludedFromStatTotal`.

See also `isExcludedFromStatTotal()`

```
public static RandomTypeCallFactory createRandomTypeContactFactory
(CallCenter cc, List<ProducedCallTypeParams> types, RandomStream stream,
 boolean checkAgents) throws CallFactoryCreationException
```

Constructs and returns a contact factory that can produce calls of randomly selected types. This constructs and returns an instance of `RandomTypeCallFactory` by using the probabilities obtained by parsing the list `types`. Each element of this list gives a type identifier with associated probability of selection. See `RandomTypeCallFactory` for more information about how the selection is performed.

Parameters

`cc` the call center model.

`types` the list of produced contact types.

`stream` the random stream used to select contact type.

`checkAgents` determines if the call factory checks that there are agents capable of serving the call before producing a call of a given type.

Returns the contact type factory.

Throws

`CallFactoryCreationException` if an error occurs during the creation of the factory.

```
public static void checkInbound (int numInCallTypes, List<
                                 ProducedCallTypeParams> types)
```

For each element in the list `types`, tests that the type identifier returned by the `ProducedCallTypeParams.getType()` method is smaller than the given constant `numInCallTypes`. This condition is necessary for the indices to represent inbound call types.

Parameters

`numInCallTypes` the number of inbound call types.

`types` the list of call type records to test.

Throws

`IllegalArgumentException` if at least one type identifier is invalid.

```
public static void checkOutbound (int numInCallTypes, int numCallTypes,
                                  List<ProducedCallTypeParams> types)
```

For each element in the list `types`, tests that the type identifier returned by the `ProducedCallTypeParams.getType()` method is greater than or equal to `numInCallTypes` but smaller than `numCallTypes`. This condition is necessary for the indices to represent inbound call types.

Parameters

`numInCallTypes` the number of inbound call types.

`numCallTypes` the number of call types, inbound or outbound.

`types` the list of call type records to test.

Throws

`IllegalArgumentException` if `numInCallTypes` is greater than `numCallTypes`, or if at least one type identifier is invalid.

ServiceTimesManager

Manages the construction of service time generators specific to each agent, to each agent group also as well as a default generator used when no generator is available for a given agent or agent group. This class associates a multiplier to each such service time which can be used to alter the mean service time. One object of this class can be constructed for each part of the service time, e.g., the talk time, the transfer time, etc.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class ServiceTimesManager
```

Constructor

```
public ServiceTimesManager (CallCenter cc, String name, List<
                            ServiceTimeParams> pars, int k, RandomStream
                            sStream, double sgenMultAllGroups, int
                            numGroups) throws CallFactoryCreationException
```

Constructs a new service times manager using call center parameters `cc`. This method uses the given list of service time parameters `pars`, and the stream `sStream` to construct service time generators.

Parameters

`cc` the call center model.

`name` the name of the part of the service time this object concerns, used in error messages.

`pars` the service time parameters.

`k` the concerned call type.

`sStream` the random stream used to generate the service times.

`sgenMultAllGroups` the multiplier applied to all service time generators.

`numGroups` the number of agent groups.

Throws

`CallFactoryCreationException` if an error occurs during the construction of the service time manager.

Methods

```
public MultiPeriodGen getServiceTimeGen()
```

Returns the default service time generator used when no agent-group-specific service time is available.

Returns the default service time generator.

`public void setServiceTimeGen (MultiPeriodGen sgen)`

Sets the default service time generator to `sgen`.

Parameter

`sgen` the new default service time generator.

`public MultiPeriodGen getServiceTimeGen (int i)`

Returns the service time generator for agent group `i`. If no such generator is available, this returns the result of `getServiceTimeGen()`.

Parameter

`i` the tested agent group.

Returns the associated service time generator.

`public MultiPeriodGen[] getServiceTimeGenGroups()`

Returns an array containing the service time generators for each agent group. If no service time generator is associated with an agent group, the element at the corresponding position in the returned array is `null`.

Returns the array of service time generators.

`public void setServiceTimeGenGroups (MultiPeriodGen[] sgenGroups)`

Sets the service time generators to `sgenGroups` for agent groups.

Parameter

`sgenGroups` the new array of service time generators.

`public void setServiceTimeGen (int i, MultiPeriodGen gen)`

Sets the service time generator for agent group `i` to `gen`.

Parameters

`i` the index of the agent group.

`gen` the new generator.

`public double[] getServiceTimesGenGroupsMult()`

Returns an array containing the multiplier for each service time generator specific to an agent group.

Returns the array of service time multipliers.

`public void setServiceTimesGenGroupsMult (double[] sgenMultGroups)`

Sets the service time multipliers for the agent groups using the array `sgenMultGroups`.

Parameter

`sgenMultGroups` the array giving the multipliers.

`public double getServiceTimesMult()`

Returns the multiplier applied to the default service time generator.

Returns the multiplier for the default service time generator.

`public void setServiceTimesMult (double sgenMult)`

Sets the multiplier for the default service time generator to `sgenMult`.

Parameter

`sgenMult` the multiplier for the default service time multiplier.

`public double getServiceTimesMult (int i)`

Returns the service time multiplier specific to agent group `i`. This returns 1 if no generator is associated with specific agent groups.

Parameter

`i` the tested agent group.

Returns the multiplier.

`public void setServiceTimesMult (int i, double mult)`

Sets the service time multiplier specific to agent group `i` to `mult`.

Parameters

`i` the agent group identifier.

`mult` the new multiplier.

`public double getServiceTimesMultAllGroups()`

Returns the service time multiplier applied to the default generator, as well as all generators specific to agent groups.

Returns the global service time multiplier.

`public void setServiceTimesMultAllGroups (double sgenMultAllGroups)`

Sets the global multiplier applied to each service time generator managed by this object to `sgenMultAllGroups`.

Parameter

`sgenMultAllGroups` the new multiplier.

`public void init (double mult)`

Initializes this manager by setting the multipliers for the random variate generators. The used multiplier is the product of `mult`, the result of `getServiceTimesMultAllGroups()`, and the generator-specific multiplier. The value of `mult` corresponds to the global service time multiplier applying to all call types.

Parameter

`mult` the global multiplier.

`public void generate (ServiceTimes st)`

Uses the random variate generators attached with this service times manager to generate service times, and store the times in `st`.

Parameter

`st` the object holding service times.

RandomTypeCallFactory

This class is similar to `RandomTypeContactFactory`, but it allows the probability of generating each contact type to change from periods to periods, and possibly depends on the presence of agents in groups. More specifically, the factory contains a $K \times P$ 2D array giving a weight $p_{k,p}$ to each call type k and main period p. Each time a call is requested, the current main period is determined, and a weight is assigned to each call type. If the selection takes account of the presence of agents, weights corresponding to call types for which no agent is available are reset to 0. The weights are then summed up, and normalized to give probabilities which are used to select a call type.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class RandomTypeCallFactory implements ContactFactory
```

Constructor

```
public RandomTypeCallFactory (CallCenter cc, double[][] probMainPeriod,
                              RandomStream stream, boolean checkAgents)
```

Constructs a new random-type call factory using period-change event associated with `cc` to obtain the current main period, and random stream `stream` to generate random numbers. The probabilities of selection $p_{k,p}$ are initialized using the given `probMainPeriod` $K \times P$ 2D array as follows. For each factory `k`, $p_{k,p} = 0$ for $p = 1, \ldots, P$ if `probMainPeriod[k]` is `null` or has length 0. The probability $p_{k,p} = q$ for $p = 1, \ldots, P$ if `probMainPeriod[k]` has a single element q. Otherwise, $p_{k,p}$ is given by `probMainPeriod[k][p]`.

Parameters

`cc` the call center object.

`probMainPeriod` the main period and call factory specific probabilities.

`stream` the random stream used to generate random numbers.

`checkAgents` determines if the call factory checks that there are agents capable of serving the call before producing a call of a given type.

Throws

`NullPointerException` if any argument is `null`.

`IllegalArgumentException` if the lengths of `factories` and `probMainPeriod` are different.

Methods

`public int nextIndex()`

Generates and returns a new type identifier.

`public double[][] getProbPeriod()`

Returns a copy of the $K \times P$ 2D array giving the values of $p_{k,p}$.

`public double getProbPeriod (int k, int p)`

Returns the value of $p_{k,p}$.

`public RandomStream getStream()`

Returns the random stream used by this factory.

OutboundCallFactory

Represents a call factory for outbound calls. This extends `CallFactory` with parameters specific to outbound calls: the probability of right-party connect, and the generators for reach and fail times.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class OutboundCallFactory extends CallFactory
```

Constructor

```
public OutboundCallFactory (CallCenter cc, CallCenterParams ccParams,
                            OutboundTypeParams par, int k) throws
                            CallFactoryCreationException
```

Constructs a new call factory for outbound call.

Parameters

`cc` the call center.

`ccParams` the call center parameters.

`par` the parameters of the outbound call type.

`k` the index of the call type.

Throws

`CallFactoryCreationException` if an error occurs during the creation of the factory.

Methods

```
public double getProbReach (int mp)
```

Returns the probability of right party connect for this outbound call type during main period `p`.

Parameter

`mp` the index of the main period.

Returns the probability of right party connect.

```
public double getProbRPC (int mp)
```

Returns the probability of right party connect for this outbound call type during main period `p`.

Parameter

`mp` the index of the main period.

Returns the probability of right party connect.

`public MultiPeriodGen getReachGen()`

Returns the random variate generator for reach times.

Returns the random variate generator for reach times.

`public MultiPeriodGen getFailGen()`

Returns the random variate generator for fail times.

Returns the random variate generatof for fail times.

AgentGroupManager

Manages an agent group in the call center model. This class implements the mechanisms necessary to construct the agent group, and to update its state during the simulation. It also manages agent disconnection if it is enabled.

By default, this agent group manager sets the number of agents in the managed group to 0, and does not change it during simulation. However, subclasses such as `AgentGroupManagerWithStaffing` can override the `init()` method in order to set and update the number of agents.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class AgentGroupManager extends NamedInfo
```

Constructor

```
public AgentGroupManager (CallCenter cc, AgentGroupParams par, int i)
                          throws AgentGroupCreationException
```

Constructs a new agent group manager for the call center `cc`, agent group `i`, and based on agent group parameters `par`.

Parameters

`cc` the call center model.

`par` the parameters of the agent group to be managed.

`i` the index of the agent group.

Throws

`AgentGroupCreationException` if an error occurs while constructing the agent group manager, or the associated agent group.

Methods

```
public double getAgentsMult()
```

Returns the factor by which the number of agents in the managed group given in parameter file is multiplied. This multiplier is reset to 1 if the number of agents is changed programmatically by, e.g., `AgentGroupManagerWithStaffing.setStaffing (int[])`.

Returns the multiplier for the managed agent group.

```
public void setAgentsMult (double mult)
```

Sets the multiplier of the managed agent group to `mult`.

Parameter

`mult` the new multiplier.

`public void connectToRouter (Router router)`

Connects the managed agent group to the router `router` by using the `Router.setAgentGroup (int, AgentGroup)` method. If agent disconnection is enabled, this method also ensures that the listener handling disconnections is notified of events related to the agent group before the router.

Parameter

`router` the router the agent group is connected to.

`public double getWeight()`

Returns the weight associated with the managed agent group.

Returns the weight of the managed agent group.

`public int getSkillCount()`

Returns the skill count associated with the managed agent group, or `Integer.MAX_VALUE` if no skill count was set explicitly by the user.

This method is mainly for internal use; the recommended way to obtain the skill count is by using `RouterManager.getSkillCount (int)` after `RouterManager.initSkillCounts (RouterParams)` was called.

Returns the explicitly set skill count.

`public double getIdleCost()`

Returns the cost of an idle agent in the managed group during one simulation time unit.

Returns the cost of an idle agent.

`public double getBusyCost()`

Returns the cost of a busy agent in the managed group during one simulation time unit.

Returns the cost of a busy agent.

`public double getPerUseCost()`

Returns the cost incurred each time an agent in the managed group starts the service of a call.

Returns the per-use cost of agents in the managed group.

`public double getIdleCost (int mp)`

Returns the cost of an idle agent managed by this group during main period `mp`, during one simulation time unit. This returns the result of `getIdleCost()` if no per-period cost were given by the user in parameter file.

Parameter

`mp` the index of the tested main period.

Returns the idle cost.

`public double getBusyCost (int mp)`

Returns the cost of a busy agent managed by this group during main period `mp`, during one simulation time unit. This returns the result of `getBusyCost()` if no per-period cost were given by the user in parameter file.

Parameter

`mp` the index of the tested main period.

Returns the busy cost.

`public double getPerUseCost (int mp)`

Returns the cost incurred each time an agent in the managed group starts a service during main period `mp`. This method returns the result of `getPerUseCost()` if no per-period costs were given in parameter file.

Parameter

`mp` the index of the tested main period.

Returns the per-use cost.

`public double getWeight (int mp)`

Returns the weight of the managed agent group during main period `mp`. If no per-period weights were given in parameter file, this method returns the result of `getWeight()`.

Parameter

`mp` the index of the tested main period.

Returns the weight of the managed agent group.

`public int getMaxAgents()`

Returns the maximal number of agents in the managed group.

Returns the maximal number of agents in the managed group.

`public int getMinAgents()`

Returns the minimal number of agents in the managed group.

Returns the minimal number of agents in the managed group.

`public int getMaxAgents (int mp)`

Returns the maximal number of agents in the managed group during main period `mp`. This method returns the result of `getMaxAgents()` if no per-period maximum number of agents were given in parameter file.

Parameter

`mp` the index of the tested main period.

Returns the maximal number of agents.

`public int getMinAgents (int mp)`

Returns the minimal number of agents in the managed group during main period `mp`. This method returns the result of `getMinAgents()` if no per-period minimum number of agents were given in parameter file.

Parameter

`mp` the index of the tested main period.

Returns the minimal number of agents.

`public static AgentGroupManager create (CallCenter cc, AgentGroupParams par, int i) throws AgentGroupCreationException`

Constructs and returns a new agent group manager for call center `cc`, agent group with index `i`, and parameters `par`. If the given parameters contain a staffing, an instance of `AgentGroupManagerWithStaffing` is created. If the parameters contain a schedule, an instance of `AgentGroupManagerWithSchedule` is constructed. If the parameters contain information about individual agents, an `AgentGroupManagerWithAgents` object is created. Otherwise, a plain `AgentGroupManager` object is created. The created object can, depending on parameters, be converted to an instance of `AgentGroupManagerWithStaffing`. The constructed (or converted) object is returned.

Parameters

`cc` the call center model.

`par` the parameters of the agent group to be managed.

`i` the index of the agent group.

Throws

`AgentGroupCreationException` if an error occurs while constructing the agent group manager, or the associated agent group.

`protected AgentGroup createAgentGroup (AgentGroupParams par, int i) throws AgentGroupCreationException`

Constructs and returns the `i`th agent group for this call center. By default, this constructs an `AgentGroup` or `DetailedAgentGroup` instance, depending on the return value of the `AgentGroupParams.isDetailed()` method.

Parameter

`i` the agent group index.

Returns the constructed agent group.

`public CallCenter getCallCenter()`

Returns a reference to the call center containing this agent group manager.

`public RandomStream getProbDisconnectStream()`

Returns the random stream used to test if an agent disconnects after the end of a service.

`public void setProbDisconnectStream (RandomStream dpStream)`

Sets the random stream used to test if an agents disconnects after the end of a service to `dpStream`.

`public MultiPeriodGen getDisconnectTimeGen()`

Returns the random variate generator used for disconnect times.

`public AgentGroup getAgentGroup()`

Returns a reference to the managed agent group.

`public double[] getProbDisconnect()`

Returns an array giving the probabilities of disconnection, for each main period.

`public double getProbDisconnect (int mp)`

Returns the probability that an agent ending a service during main period `mp` disconnects for a random time.

`public void init()`

Calls `init` on the managed agent group.

`public int[] getStaffing()`

Returns the raw staffing of the managed agent group. The returned array gives the number of agents in the managed group during each main period in the model, before any multiplier is applied.

This method is mainly for internal use; the `getEffectiveStaffing()` method should be used instead to take multipliers into account.

The default behavior of this method is to return an array of 0's.

Returns the raw staffing for the managed agent group.

`public int getStaffing (int mp)`

Returns element `mp` of the array that would be returned by `getStaffing()`.

As with `getStaffing()`, this method is for internal use. The method `getEffective-Staffing (int)` should be used instead.

Parameter

`mp` the index of the tested main period.

Returns the raw staffing.

`public int[] getEffectiveStaffing()`

Returns the staffing determining the effective number of agents in the managed group for each main period in the model. This method calls `getStaffing()`, and multiplies each element of the returned array by $m * m_i$, where m is determined by `CallCenter.getAgentsMult()` and m_i is given by `getAgentsMult()`. The resulting numbers are rounded to the nearest integers, and stored in the array being returned.

Returns the effective staffing.

`public int getEffectiveStaffing (int mp)`

Returns element `mp` of the array that would be returned by `getEffectiveStaffing()`.

Parameter

`mp` the index of the tested main period.

Returns the effective staffing.

`public AgentGroupSchedule getSchedule()`

Returns the schedule associated with the managed agent group. This corresponds to the effective schedule if this object is an instance of `AgentGroupManagerWithSchedule`. If this object is an instance of `AgentGroupManagerWithStaffing` converted from an instance with schedule, this returns the schedule of the original agent group manager with schedule. Otherwise, this method returns `null`.

`public static boolean estimateParameters (AgentGroupParams par) throws DistributionCreationException`

Estimates parameters relative to the agent group described by `par`. The method estimates the parameters of the distribution for disconnect times. If the agent group has staffing information, the method then estimates `staffing` and `probAgents` from `staffingData` if `staffingData` is given. If scheduling information is used, the method calls `AgentGroupSchedule.estimateParameters (AgentGroupScheduleParams)` to complete parameter estimation.

Parameter

`par` the parameters of the agent group.

Returns `true` if and only if some parameters were estimated.

Throws

`DistributionCreationException` if an error occurs during parameter estimation.

AgentGroupManagerWithStaffing

Manages an agent group with a staffing vector giving the number of agents for each period. This manager stores the staffing vector and registers a period-change to update the staffing at the beginning of main periods.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class AgentGroupManagerWithStaffing extends AgentGroupManager
```

Constructors

`public AgentGroupManagerWithStaffing (CallCenter cc, AgentGroupParams par, int i) throws AgentGroupCreationException`

Calls the superclass' constructor, and extracts the staffing from `par`.

`public AgentGroupManagerWithStaffing (CallCenter cc, AgentGroupParams par, int i, int[] staffing) throws AgentGroupCreationException`

Similar to the first constructor `AgentGroupManagerWithStaffing (CallCenter, AgentGroupParams, int)`, but uses the given staffing vector instead of the one extracted from `par`.

Methods

`public void setStaffing (int[] staffing)`

Sets the staffing vector to `staffing`.

Parameter

`staffing` the new staffing vector.

`public void setStaffing (int mp, int staffing)`

Sets the staffing for main period `mp` to `staffing`.

Parameters

`mp` the index of the affected main period.

`staffing` the new staffing.

`public void setEffectiveStaffing (int[] staffing)`

Sets the effective staffing for the managed agent group to `staffing`. This method sets the staffing to `staffing` using `setStaffing (int[])`, but it also resets the value of the multiplier $m*m_i$ to 1. This makes sure that `AgentGroupManager.getEffectiveStaffing()` will return the same value as the staffing passed to this method.

Parameter

`staffing` the new effective staffing.

`public void setEffectiveStaffing (int mp, int ns)`

Similar to `setEffectiveStaffing (int[])`, for a single main period.

Parameters

`mp` the index of the affected main period.

`ns` the new number of agents.

`public int[] getCurNumAgents()`

Returns the number of agents in the managed group for the current simulation replication. If the number of agents is deterministic, this method returns the result of `AgentGroupManager.getEffectiveStaffing()`. Otherwise, it returns the current (random) number of agents for each main period.

Returns the number of agents in the current replication.

`public int getCurNumAgents (int mp)`

Similar to `getCurNumAgents()`, for a given main period `mp`.

Parameter

`mp` the index of the main period.

Returns the number of agents.

`public double[] getAgentProbability()`

Returns the per-period probabilities of presence for each agent in the group. If no such probabilities wre given by the user, this returns an array of 1's.

Returns the presence probability, for each main period.

`public double getAgentProbability (int mp)`

Similar to `getAgentProbability()`, for a given main period `mp`.

Parameter

`mp` the index of the main period.

Returns the presence probability.

`public void setAgentProbability (double[] prob)`

Sets the per-period presence probabilities of agents to `prob`.

Parameter

`prob` the per-period presence probabilities.

`public void setAgentProbability (int mp, double prob)`

Sets the presence probability of agents to `prob` for main period `mp`.

AgentGroupManagerWithSchedule

Manages an agent group whose member follow a given schedule. A schedule is composed of shifts that can start and end at arbitrary times during the simulation horizon. This agent group manager encapsulates a simulation event for each shift. This event is used to add or remove agents to the managed group during simulation.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class AgentGroupManagerWithSchedule extends AgentGroupManager
```

Constructor

```
public AgentGroupManagerWithSchedule (CallCenter cc, AgentGroupParams par,
                                      int i) throws
                                      AgentGroupCreationException
```

Constructs the new schedule-based agent group manager.

Parameters

`cc` the call center model.

`par` the agent group parameters.

`i` the index of the agent group.

Throws

`AgentGroupCreationException` if an exception occurs when creating the agent group.

Methods

```
protected AgentGroup createAgentGroup (AgentGroupParams par, int i)
                                       throws AgentGroupCreationException
```

Constructs and returns a detailed agent group, which is needed to add and remove agents.

```
public ScheduleShift[] getShifts()
```

Returns the shifts composing the schedule of the agents.

Returns the shifts composing the schedule.

```
public int getNumShifts()
```

Returns the number of shifts in the schedule.

Returns the number of shifts in the schedule.

`public ScheduleShift getShift (int i)`

Returns the shift with index `i`.

Parameter

`i` the index of the shift.

Returns the corresponding shift.

`public int[] getNumAgents()`

Returns a vector giving the raw number of agents for each shift. This method is for internal use; the method `getEffectiveNumAgents()` is recommended to take account of agents multipliers into account.

Returns the raw number of agents per shift.

`public int getNumAgents (int shift)`

Returns the raw number of agents in shift `shift`. The method `getEffectiveNumAgents (int)` can be used to take agents multipliers into account.

Parameter

`shift` the index of the shift.

Returns the raw number of agents on the shift.

`public int[] getEffectiveNumAgents()`

Returns the effective number of agents during each shift. This method calls `getNumAgents()`, and multiplies each element of the returned array by $m*m_i$, where m is determined by `CallCenter.getAgentsMult()` and m_i is given by `AgentGroupManager.getAgentsMult()`. The resulting numbers are rounded to the nearest integers, and stored in the array being returned.

Returns the effective number of agents during each shift.

`public int getEffectiveNumAgents (int shift)`

Similar to `getEffectiveNumAgents()`, for a specific shift `shift`.

Parameter

`shift` the index of the tested shift.

Returns the effective number of agents on the shift.

`public void setNumAgents (int[] numAgents)`

Sets the vector of raw numbers of agents to `numAgents`.

Parameter

`numAgents` the new vector of agents.

`public void setNumAgents (int shift, int n)`

Sets the raw number of agents in shift `shift` to `n`.

Parameters

`shift` the index of the affected shift.

`n` the new number of agents.

`public void setEffectiveNumAgents (int[] numAgents)`

Sets the effective number of agents for each shift of the managed agent group to `numAgents`. This method sets the number of agents to `numAgents` using `setNumAgents (int[])`, but it also resets the value of the multiplier $m * m_k$ to 1. This makes sure that `getEffectiveNumAgents()` will return the same value as the vector passed to this method.

Parameter

`numAgents` the new vector of agents.

`public void setEffectiveNumAgents (int shift, int n)`

Similar to `setEffectiveNumAgents (int[])`, but only sets the number of agents in shift `shift` to `n` instead of the number of agents in all shifts.

Parameters

`shift` the index of the affected shift.

`n` the new number of agents.

`public boolean[][] getShiftMatrix()`

Computes and returns the matrix of shifts. Element (j, p) of this $J \times P$ matrix, where J corresponds to the number of shifts and P, to the number of main periods, is `true` if and only if agents are scheduled to work on shift `j` during main period `p`.

`public int[][] getShiftMatrixInt()`

Similar to `getShiftMatrix()`, but returns a matrix of integers, with 0 meaning `false`, and 1 meaning `true`.

`public int[] getStaffing()`

Computes and returns the staffing vector. This corresponds to the column vector returned by `getNumAgents()` multiplied by the matrix returned by `getShiftMatrix()`.

AgentGroupManagerWithAgents

Manages an agent group with detailed information on each agent.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class AgentGroupManagerWithAgents extends AgentGroupManager
```

Constructor

```
public AgentGroupManagerWithAgents (CallCenter cc, AgentGroupParams par,
                                    int i) throws
                                    AgentGroupCreationException
```

Creates an agent group manager with the call center model `cc`, agent group parameters `par`, and agent group index `i`.

Parameters

`cc` the call center model.

`par` the agent group parameters.

`i` the agent group index.

Throws

`AgentGroupCreationException` if an error occurs during the creation of the agent group manager.

Methods

```
protected AgentGroup createAgentGroup (AgentGroupParams par, int i)
                                       throws AgentGroupCreationException
```

Constructs and returns a detailed agent group, which is needed to add and remove agents.

```
public AgentInfo[] getAgents()
```

Returns an array containing an information object for each agent in this group.

Returns the array of agent information objects.

```
public int getNumAgents()
```

Returns the number of agents in this group.

Returns the number of agents in this group.

```
public AgentInfo getAgent (int i)
```

Returns the agent with index `i` in this group.

Parameter

`i` the index of the agent.

Returns the agent information object.

`public boolean[][] getShiftMatrix()`

Computes and returns the shift matrix. Element (j, p) of this $J \times P$ matrix, where J corresponds to the number of shifts and P, to the number of main periods, is `true` if and only if agents are scheduled to work on shift `j` during main period `p`.

`public int[][] getShiftMatrixInt()`

Similar to `getShiftMatrix()`, but returns a matrix of integers, with 0 meaning `false`, and 1 meaning `true`.

TimeInterval

Represents a time interval.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class TimeInterval
```

Constructors

`public TimeInterval (CallCenter cc, TimeIntervalParams par)`

Constructs a time interval from the call center `cc`, and the parameters `par`. This constructor converts times in `par`, expressed as XML durations, to the default time unit used by call center `cc`. It then checks that the starting time of the interval is not greater than its ending time.

Parameters

`cc` the call center.

`par` the parameters.

`public TimeInterval (double startingTime, double endingTime)`

Constructs a new time interval from the given starting and ending times.

Parameters

`startingTime` the starting time.

`endingTime` the ending time.

Methods

`public double getStartingTime()`

Returns the starting time of this interval.

Returns the starting time.

`public double getEndingTime()`

Returns the ending time of this interval.

Returns the ending time.

`public static void checkIntervals (TimeInterval... intervals)`

Verifies that the intervals of the given array are non-decreasing and do not overlap. This method throws an illegal-argument exception if the check fails.

Parameter

`intervals` the array of intervals to check.

```
public static TimeInterval[] create (CallCenter cc, List<
                                     TimeIntervalParams> intervalList)
```

Constructs an array of time intervals from the list of interval parameters.

Parameters

`cc` the call center.

`intervalList` the list of interval parameters.

Returns the array of intervals.

ShiftPart

Represents the part of a shift in a schedule. A shift part is a time interval with an additional field giving its type.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class ShiftPart extends TimeInterval
```

Field

`public static String WORKING`
The text "Working".

Constructors

`public ShiftPart (CallCenter cc, ShiftPartParams par)`
Constructs a new shift part using the call ceneter `cc`, and parameters `par`.

Parameters

`cc` the call center.

`par` the parameters for the part.

`public ShiftPart (double startingTime, double endingTime, String type)`
Constructs a new shift part using the given starting time, ending time, and type.

Parameters

`startingTime` the starting time of the shift part.

`endingTime` the ending time of the shift part.

`type` the type of the part.

Methods

`public String getType()`
Returns the type associated with this shift part.

Returns the type of this shift part.

`public boolean isWorking()`
Determines if agents are working during this part of the shift. This method returns `true` if and only if the string returns by `getType()` is equal to `Working`, case insensitive.

Returns the success indicator of the test.

```
public static ShiftPart[] create1 (CallCenter cc, List<ShiftPartParams>
                                   intervalList)
```

Constructs an array of shift parts from the list of part parameters.

Parameters

`cc` the call center.

`intervalList` the list of part parameters.

Returns the array of shift parts.

ScheduleShift

Represents a shift in a schedule for agents. A shift contains an array of parts as well as an integer giving the number of agents scheduled on that shift.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class ScheduleShift
```

Constructors

`public ScheduleShift (CallCenter cc, ScheduleShiftParams par)`

Constructs a new shift from call center `cc`, and parameters `par`.

Parameters

`cc` the call center.

`par` the parameters.

`public ScheduleShift (ShiftPart[] parts, int numAgents, double probAgents)`

Constructs a new schedule shift from parts in the array `parts`, and with `numAgents` agents.

Parameters

`parts` the shift parts.

`numAgents` the number of agents.

Methods

`public int getNumAgents()`

Returns the number of agents on this shift.

Returns the number of agents on this shift.

`public void setNumAgents (int numAgents)`

Sets the number of agents on that shift to `numAgents`.

Parameter

`numAgents` the number of agents.

`public ShiftPart[] getParts()`

Returns an array containing the shift parts.

Returns the array of shift parts.

`public int getNumParts()`

Returns the number of parts for this shift.

Returns the number of parts.

`public ShiftPart getPart (int i)`

Returns the shift part with index `i`.

Parameter

`i` the index of the part.

Returns the shift part.

`public boolean[] getShiftVector (PeriodChangeEvent pce)`

Computes and returns the shift vector for this shift, relative to the period-change event `pce`. Element p of this P-dimensional vector, where P is the number of main periods is `true` if and only if agents are scheduled to work during main period p.

Parameter

`pce` the period-change event.

Returns the shift vector.

`public int[] getShiftVectorInt (PeriodChangeEvent pce)`

Similar to `getShiftVector (PeriodChangeEvent)`, but returns an array of integers rather than an array of booleans. Element p of the returned array contains 1 if agents are scheduled to work in main period p, and 0 otherwise.

Parameter

`pce` the period-change event.

Returns the shift vector.

`public double getAgentProbability()`

Returns the presence probability of each agent on that shift.

`public void setAgentProbability (double prob)`

Sets the presence probability of agents on this shift to `prob`.

`public static boolean estimateParameters (ScheduleShiftParams par)`

Estimates the `numAgents` and `probAgents` parameters of the shift described by `par` from the `numAgentsData` array of observations, assuming that the number of agents follows a binomial distribution and using the maximum likelihood method.

Parameter

`par` the parameters of the shift.

Returns `true` if and only if some parameters were estimated.

ShiftEvent

Represents a simulation event adding agents to a group at the beginning of working parts of a shift, and removing them at the end of working parts. The agents to be added or removed are stored into an internal array of `Agent` objects so the agents are reused from parts to parts of a given shift.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class ShiftEvent extends Event
```

Constructors

`public ShiftEvent (DetailedAgentGroup group, ScheduleShift shift)`

Constructs a new shift event managing agent group `group`, and using information in shift `shift`.

Parameters

`group` the agent group to which agents are added and removed.

`shift` the shift used to determine the number of agents and working parts.

`public ShiftEvent (DetailedAgentGroup group, Agent[] agents, ScheduleShift shift)`

Similar to `ShiftEvent (DetailedAgentGroup, ScheduleShift)`, except that the agents in array `agents` are added and removed to the group rather than an array of new `Agent` objects.

Methods

`public void init (RandomStream stream, double mult)`

Initializes this event with a new multiplier `mult`, and resets the internal part index. This method gets the number of agents on the associated shift, multiplies this number with `mult`, and rounds the result to the nearest integer; this gives the effective number of agents on the shift. The method then creates or updates an internal array of `Agent` objects which are added and removed from the associated group each time the event occurs. The array of agents is created or updated only if it does not exist yet, or if its length does not correspond to the effective number of agents on the shift.

Parameters

`stream` a random stream used to generate the number of agents when it is random.

`mult` the multiplier for the number of agents.

Throws

`IllegalArgumentException` if `mult` is negative.

`public void schedule()`

Schedules this event to occur at the next time it is needed to add or remove the associated agents from the attached group. If the simulation time is greater than the ending time of the last part of the shift, the event is not scheduled anymore. The method `init (RandomStream, double)` can be used to reset the event.

AgentInfo

Encapsulates the information concerning a specific agent in a call center model.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class AgentInfo extends NamedInfo
```

Constructor

`public AgentInfo (CallCenter cc, AgentParams par)`

Constructs a new agent information object using the call center model `cc`, and the agent parameters `par`.

Parameters

`cc` the call center model.

`par` the agent parameters.

Methods

`public Agent getAgent()`

Returns the agent associated with this object.

Returns the associated agent.

`public ScheduleShift getShift()`

Returns an object representing the shift of the agent associated with this object.

Returns the shift of this agent.

CallSourceManager

Represents information concerning a call source, i.e., an arrival process or a dialer.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallSourceManager extends NamedInfo
```

Constructor

`public CallSourceManager (CallCenter cc, CallSourceParams par)`

Constructs a new call source information object with the given call center and call source parameters.

Parameters

`cc` the call center model.

`par` the call source parameters.

Methods

`public boolean isSourceEnabled()`

Returns `true` if the concerned call source is enabled, i.e., if it produces calls.

Returns the status of the managed call source.

`public double[] getSourceToggleTimes()`

Returns the source toggle times. This array contains an even number of simulation times, each value representing a starting or stopping time.

Returns the source toggle times.

ArrivalProcessManager

Encapsulates the parameters of an arrival process, constructs the corresponding `ContactArrivalProcess` object, and updates its state during simulation.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class ArrivalProcessManager extends CallSourceManager
```

Field

```
public static RandomVariateGenParams s_bgenParams
```

This field is initialized by `estimateParameters (CallCenterParams, ArrivalProcessParams, int, double)` when the distribution of a busyness factor is estimated in addition to parameters of arrival process. In such a case, the field is initialized with the parameters of a random variate generator for the busyness factor. Otherwise, this field is `null`.

Constructor

```
public ArrivalProcessManager (CallCenter cc, ArrivalProcessParams par, int
                              k) throws ArrivalProcessCreationException
```

Constructs a new arrival process manager for the call center model `cc`, the parameters `par`, and with index `k`. If `k` is smaller than the number of inbound call types defined by the model, this creates an arrival process producing calls of the single type `k`. Otherwise, the arrival process can produce calls of multiple types.

Parameters

`cc` the call center model.

`par` the parameters of the arrival process.

`k` the index of the arrival process.

Throws

`ArrivalProcessCreationException` if an error occurs during the creation of the arrival process.

Methods

```
public double getArrivalsMult()
```

Returns the value of the multiplier for the arrival rates.

Returns the multiplier for the arrival rates.

`public void setArrivalsMult (double arrivalsMult)`

Sets the multiplier for arrival rates to `arrivalsMult`.

Parameter

`arrivalsMult` the new multiplier.

Throws

`IllegalArgumentException` if `arrivalsMult` is negative.

`public ContactArrivalProcess getArrivalProcess()`

Returns the associated arrival process.

Returns the associated arrival process.

```
protected ContactArrivalProcess createArrivalProcess (ArrivalProcessParams
                                                       par, int k,
                                                       ContactFactory
                                                       factory) throws
                                                       ArrivalProcessCreationException
```

Constructs and returns the arrival process to be managed. This method uses `ArrivalProcessParams.getType()` to get a type identifier for the arrival process. It then retrieves parameters and initializes an arrival process specific to the given type. If the name of the arrival process corresponds to a constant in `ArrivalProcessType`, the method handles its construction directly. Otherwise, it queries every factory registered using `addArrivalProcessFactory (ArrivalProcessFactory)` until it finds one capable of creating the arrival process. If no such factory can create the process, it uses the `ServiceLoader` class to find an arrival process factory. If that last step fails, an arrival-process creation exception is thrown.

Parameters

`par` the parameters of the arrival process.

`k` the call type identifier.

`factory` the call factory that will be attached to the new process.

Returns the constructed arrival process.

Throws

`ArrivalProcessCreationException` if an error occurs during the construction of the arrival process.

`public static void addArrivalProcessFactory (ArrivalProcessFactory apf)`

Registers the arrival process factory `apf` for arrival process managers. If the user-specified type of arrival process does not correspond to a predefined process, the registered factories are queried to find one capable of creating an arrival process. This method must be called before the call-center simulator is initialized.

Parameter

`apf` the new arrival process factory to register.

```
public static boolean estimateParameters (CallCenterParams ccParams,
                                          ArrivalProcessParams par, int
                                          numPeriods, double
                                          periodDuration)
```

Estimates the parameters of the arrival process described by `par`, for a call center with `numPeriods` main periods with duration `periodDuration`. The method replaces the data stored in `par` with estimated parameters using an algorithm depending on the type of arrival process. If returns `true` if parameters were estimated, and `false` if there were no parameters to estimate. If parameter estimation is needed but fails, an illegal-argument exception is thrown.

More specifically, the method returns `false` if no data is stored in the given parameter object. If the type of arrival process corresponds to a constant in `ArrivalProcessType`, parameter estimation is handled directly by this method. Otherwise, the method queries every arrival process factory registered using `addArrivalProcessFactory (ArrivalProcessFactory)` until it finds a factory capable of performing the estimation. If no such factory exists, it uses the `ServiceLoader` class to find a factory dynamically. If this last step fails, the method throws an illegal-argument exception.

Parameters

`par` the parameters of the arrival process.

`numPeriods` the number of main periods.

`periodDuration` the duration of main periods.

Returns a boolean indicating if parameter estimation was performed.

```
public void init (double b)
```

Initializes the managed arrival process by calling `ContactArrivalProcess.init (double)`. The busyness factor given to the arrival process is the argument `b` multiplied by the product of the multiplier returned by `getArrivalsMult()`, and the global multiplier returned by `CallCenter.getArrivalsMult()`. The expectation $\mathbb{E}[B]$ is also set to these product of multipliers, multiplied by the mean value of B that can be generated using the generator returned by `CallCenter.getBusynessGen()`.

Parameter

`b` the generated base busyness factor.

DialerManager

Manages a dialer performing outbound calls. An object of this class encapsulates the parameters specific to the dialer, and provides methods to construct the corresponding `Dialer` instance, and to update its state it during simulation.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class DialerManager extends CallSourceManager
```

Constructor

```
public DialerManager (CallCenter cc, DialerParams par, int k) throws
                      DialerCreationException
```

Constructs a new dialer manager using the call center model `cc`, the dialer parameters `par`, and dialer index `k`. If `k` is smaller than the number of outbound call types, this creates a dialer producing calls of a single type. Otherwise, this creates a dialer producing calls of randomly-chosen types.

Parameters

`cc` the call center model.

`par` the dialer's parameters.

`k` the index of the dialer.

Throws

`DialerCreationException` if a problem occurs during the creation of the dialer.

Methods

```
protected DialerPolicy createDialerPolicy (DialerParams par, DialerList
                                           dialerList) throws
                                           DialerCreationException
```

Constructs and returns an object representing the managed dialer's policy. This method uses `DialerParams.getDialerPolicy()` to get a type identifier for the dialer's policy. It then retrieves parameters and initializes a dialer's policy specific to the given type. If the name of the dialer's policy corresponds to a constant in `DialerPolicyType`, the method handles its construction directly. Otherwise, it queries every factory registered using `addDialer-PolicyFactory (DialerPolicyFactory)` until it finds one factory capable of constructing the policy. If no such factory can create the policy, it uses the `ServiceLoader` class to find a dialer's policy factory dynamically. If that last step fails, a dialer-creation exception is thrown.

Parameters

`par` the parameters of the dialer's policy.

`dialerList` the dialer's list.

Returns the constructed dialer's policy.

Throws

`DialerCreationException` if an error occurs during the creation of the dialer's policy.

`public static void addDialerPolicyFactory (DialerPolicyFactory dpf)`

Registers the dialer policy factory `dpf` for dialer managers. If the user-specified dialer policy does not correspond to a predefined policy, the registered factories are queried to find one capable of creating a dialer's policy. This method must be called before the call-center simulator is initialized.

Parameter

`dpf` the new dialer policy factory to register.

`public Dialer getDialer()`

Returns the dialer managed by this object.

Returns the managed dialer.

`public CallCenter getCallCenter()`

Returns a reference to the call center associated with this dialer manager.

Returns the associated call center.

`public AgentGroupSet getTargetSet()`

Returns a reference to the target set of agent groups associated with the managed dialer.

Returns the target set of agent groups.

`public int getServiceLevelIndex()`

Determines the 0-based index of the service-level information matrix used if the dialing policy in use takes service level (or acceptable waiting time) into account for its decisions.

Returns the index of the service-level information matrix.

`public int getNumCheckedPeriods()`

Determines the number of testing periods used by dialing policies taking cumulative statistics (service level, mismatch rate, etc.) into account for taking their decisions.

`public double getCheckedPeriodDuration()`

Determines the duration, in simulation time units, of the testing periods used by some dialing policies taking cumulative statistics into account.

`public double getSlInboundThresh()`

Determines the outbound-to-inbound threshold for the service level. When the service level goes below this threshold, some dialers start moving agents from outbound groups to inbound groups.

`public double getSlOutboundThresh()`

Determines the inbound-to-outbound threshold for the service level. When the service level goes above this threshold, some dialers start moving agents from inbound groups to outbound groups.

`public boolean isUseNumActionEvents()`

Determines if the dialer subtracts the number of calls for which dialing is in progress from the number of calls to dial. When dial delays are large enough for the dialer to start often while phone numbers are being composed, the agents of the call center might receive too many calls to serve, which results in a large number of mismatches. If this attribute is set to `true` (the default), the dialer will take into account the number of calls for which dialing is in progress while determining the number of additional calls to dial.

DialerObjects

Regroups objects used by dialers. This class encapsulates the testing set containing all the agent groups, and value generators for the reaching probability, and reaching and failing times. These parameters are the same for every dialer, but they are not needed if no dialer is used.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class DialerObjects
```

Constructor

`public DialerObjects (CallCenter cc)`

Constructs a new set of dialer objects from the given call center model.

Parameter

`cc` the call center model.

Methods

`public AgentGroupSet getAgentGroupTestSet()`

Returns the testing set of agent groups used by some dialing policies.

Returns the testing set of agent groups.

`public ValueGenerator getProbReachGen()`

Returns the value generator giving the probability of right party connect for any outbound call. The probability often depends on the call type and period of arrival of the call.

Returns the value generator for the probability of right party connect.

`public ValueGenerator getReachTimeGen()`

Returns the value generator giving the needed time for a caller to be reached. By using a value generator, the distribution of this (random) time can depend on the call type and period of arrival.

Returns the value generator for the reach times.

`public ValueGenerator getFailTimeGen()`

Returns the value generator for the needed time for an outbound call to fail. This method is similar to `getReachTimeGen()`, for fail times.

Returns the value generator for fail times.

CallNotifierForBadContactMismatchRate

Exited-contact and new-contact listeners used to update the state of the `BADCONTACTMISMATCHRATE` dialer's policy. This listener calls `BadContactMismatchRatesDialerPolicy.notifyInbound-Contact (Contact, boolean)`, and `BadContactMismatchRatesDialerPolicy.notifyOutbound-Contact (Contact, boolean)` methods when failed contacts are notified, or when other contacts exit. This listener should be registered with the router and with the dialer to receive failed calls.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallNotifierForBadContactMismatchRate implements
             ExitedContactListener, NewContactListener
```

Constructor

`public CallNotifierForBadContactMismatchRate (DialerManager dialerManager)`

Constructs a new call notifier for the dialer manager `dialerManager`.

Parameter

`dialerManager` the associated dialer manager.

CallNotifierForAgentsMove

Exited-contact listener used to update the state of the `AGENTSMOVE` dialer's policy during the simulation. This listener collects statistics about exiting calls to get estimates of the service level in a time window, which is used to determine if the dialer performs inbound-to-outbound, or outbound-to-inbound moves. After this listener is constructed, it should be registered with the router.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallNotifierForAgentsMove implements ExitedContactListener
```

Constructor

`public CallNotifierForAgentsMove (DialerManager dialerManager)`

Constructs a new call notifier for the dialer manager `dialerManager`.

Parameter

`dialerManager` the associated dialer manager.

DialerLimit

Represents a limit on the number of calls to dial. Such a limit is described by a time interval on which it applies, the maximal number of outbound calls allowed for this dialer during the interval, and the call types on which the limit applies. This class extends the `TimeInterval` class for the information about the time interval on which the limit applies.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class DialerLimit extends TimeInterval
```

Constructor

`public DialerLimit (CallCenter cc, DialerLimitParams par)`

Constructs a new dialer limit using the call center `cc`, and limit parameters `par`.

Parameters

`cc` the call center model.

`par` the limit parameters.

Methods

`public int getValue()`

Returns the maximal number of calls of the specified typeset during the given interval.

Returns the value of the limit.

`public int[] getTypes()`

Returns an array giving the list of call types on which the limit applies.

Returns the list of call types on which the limit applies.

`public boolean hasType (int k)`

Returns `true` if and only if this limit applies to call type `k`. This method always returns `false` for inbound call types.

Parameter

`k` the tested call type.

Returns the success indicator of the test.

DialerListWithLimits

Represents a dialer list imposing limits on the number of calls to dial.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class DialerListWithLimits implements DialerList
```

Constructors

```
public DialerListWithLimits (CallCenter cc, int k, DialerLimitParams...
                             limits)
```

Constructs a new dialer list with limits for the call center `cc`, call type `k`, and limits `limits`.

Parameters

`cc` the call center model.

`k` the call type identifier.

`limits` the dialer's limits.

```
public DialerListWithLimits (CallCenter cc, RandomTypeCallFactory factory,
                             DialerLimitParams... limits)
```

Constructs a new dialer list with limits for the call center `cc`, the call factory `factory` which generates calls of random types, and the limits `limits`.

Parameters

`cc` the call center model.

`factory` the random-type call factory.

`limits` the dialer's limits.

RouterManager

Manages the creation of the router as well as the data structures containing routing information. This class provides the necessary facility to read and validate routing tables stored in `RouterParams` instances, construct missing routing tables according to the rules specified in `RouterParams`, and create the appropriate `Router` instance used for simulation.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class RouterManager
```

Constructor

`public RouterManager (CallCenter cc, RouterParams par) throws RouterCreationException`

Constructs a new router manager using the call center model `cc`, and the router's parameters `par`.

Parameters

`cc` the call center model.

`par` the router's parameters.

Throws

`RouterCreationException` if a problem occurs during the construction of the router.

Methods

`public Router getRouter()`

Returns a reference to the router managed by this object.

Returns the managed router.

`public void setRouter (Router router)`

Sets the managed router to `router`.

Parameter

`router` the new managed router.

`public void initTypeToGroupMap (RouterParams par)`

Initializes the type-to-group map from the router parameters `par`, or constructs a new type-to-group map from other information if `RouterParams.getRoutingTableSources()` defines the `typeToGroupMap` attribute. This method does nothing if a type-to-group map was already constructed. The obtained type-to-group map can be accessed through `getTypeToGroupMap()` after this method succeeds, and an illegal-argument exception is thrown if this method fails.

Parameter

`par` the router's parameters.

`public void initGroupToTypeMap (RouterParams par)`

Initializes the group-to-type map from the router parameters `par`, or constructs a new group-to-type map from other information if `RouterParams.getRoutingTableSources()` defines the `groupToTypeMap` attribute. This method does nothing if a group-to-type map was already constructed. The obtained group-to-type map can be accessed through `getGroupToTypeMap()` after this method succeeds, and an illegal-argument exception is thrown if this method fails.

Parameter

`par` the router's parameters.

`public void initRanksTG (RouterParams par)`

Initializes the type-to-group matrix of ranks from the router parameters `par`, or constructs a new type-to-group matrix of ranks from other information if `RouterParams.getRoutingTableSources()` defines the `ranksTG` attribute. This method does nothing if a type-to-group matrix of ranks was already constructed. The obtained matrix can be accessed through `getRanksTG()` after this method succeeds, and an illegal-argument exception is thrown if this method fails.

Parameter

`par` the router's parameters.

`public void initRanksGT (RouterParams par)`

Initializes the group-to-type matrix of ranks from the router parameters `par`, or constructs a new group-to-type matrix of ranks from other information if `RouterParams.getRoutingTableSources()` defines the `ranksGT` attribute. This method does nothing if a group-to-type matrix of ranks was already constructed. The obtained matrix can be accessed through `getRanksGT()` after this method succeeds, and an illegal-argument exception is thrown if this method fails.

Parameter

`par` the router's parameters.

`public void initRanksGTUpdate (RouterParams par)`

Initializes the auxiliary group-to-type matrix of ranks associated with minimal waiting times. This method does nothing if no `ranksGTUpdate` elements is given in `par`. These matrices can be retrieved using the `getRanksGTDelay()` method.

Parameter

`par` the router's parameters.

`public void initWeightsTG (RouterParams par)`

Initializes the type-to-group matrix of weights using the router's parameters `par`. If no such matrix is defined, a matrix filled with 1's is created. An illegal-argument exception is thrown if any error occurs during the construction and validation of the matrix of weights. The matrix can be accessed using the `getWeightsTG()` method if this method succeeds.

Parameter

`par` the router's parameters.

`public void initWeightsGT (RouterParams par)`

Initializes the group-to-type matrix of weights using the router's parameters `par`. If no such matrix is defined in `par`, one is initialized from the queue weights. An illegal-argument exception is thrown if any error occurs during the construction and validation of the matrix of weights. The matrix can be accessed using the `getWeightsGT()` method if this method succeeds.

Parameter

`par` the router's parameters.

`public void initDelays (RouterParams par)`

Initializes the matrix of delays from the router's parameters `par`. If no matrix of delays is defined in `par`, this method creates a $I \times K$ matrix filled with 0's. The constructed matrix can be accessed using `getDelaysGT()`.

Parameter

`par` the router's parameters.

`public void initQueueWeights (RouterParams par)`

Initializes the queue weights using the router's parameters `par`. If no queue weights are specified in `par`, this method creates a vector of queue weights using the weights associated with each call type. The queue weights can be accessed using the `getQueueWeights()` method if this method succeeds.

Parameter

`par` the router's parameters.

`public void initIncidenceMatrixTG (RouterParams par)`

Initializes the type-to-group incidence matrix from the router parameters `par`, or constructs a new type-to-group incidence matrix from other information if `RouterParams.getRoutingTableSources()` defines the `incidenceMatrixTG` attribute. This method does nothing if a type-to-group incidence matrix was already constructed. The obtained matrix can be accessed through `getIncidenceMatrixTG()` after this method succeeds, and an illegal-argument exception is thrown if this method fails.

Parameter

`par` the router's parameters.

`public void initIncidenceMatrixGT (RouterParams par)`

Initializes the group-to-type incidence matrix from the router parameters `par`, or constructs a new group-to-type incidence matrix from other information if `RouterParams.getRoutingTableSources()` defines the `incidenceMatrixGT` attribute. This method does nothing if a group-to-type incidence matrix was already constructed. The obtained matrix can be accessed through `getIncidenceMatrixGT()` after this method succeeds, and an illegal-argument exception is thrown if this method fails.

Parameter

`par` the router's parameters.

`public void initSkillCounts (RouterParams par)`

Initializes the skill counts using the router's parameters `par`. If no skill count is specified in `par`, the skill counts are initialized from agent groups' `skillCount` attribute. If the skill count is not specified explicitly for at least one agent group, the group-to-type incidence matrix is initialized and used to count the number of call types accessible for this agent group.

Parameter

`par` the router's parameters.

`public void initStages (RouterParams par)`

Initializes the routing stages for the overflow-and-priority routing policy from parameters in `par`. This method does nothing if the stages, returned by the `getRoutingStages()` method, are already initialized. Otherwise, it processes parameters in `par` to initialize the stages.

Parameter

`par` the routing parameters.

`protected Router createRouter (RouterParams par) throws RouterCreationException`

Constructs and returns the router to be managed. This method uses `RouterParams.getRouterPolicy()` to get a type identifier for the router's policy. It then retrieves parameters and initializes a router specific to the given type. If the name of the policy corresponds to a constant in `RouterPolicyType`, the method handles its construction directly. Otherwise, it queries every factory registered using `addRouterFactory (RouterFactory)` until it gets one capable of creating the policy. If no such factory exists, it uses the `ServiceLoader` class to find a router policy factory dynamically. If that last step fails, the method throws a router-creation exception.

Parameter

`par` the parameters of the router.

Returns the constructed router.

Throws

`RouterCreationException` if an error occurs during the construction.

`public static void addRouterFactory (RouterFactory rf)`

Registers the router factory `rf` for router managers. If the user-specified router policy does not correspond to a predefined policy, the registered factories are queried to find one capable of creating a router. This method must be called before the call-center simulator is initialized.

Parameter

`rf` the new router factory to register.

`public void initAgentsPrefBased (RouterParams par, AgentsPrefRouter router1)`

Initializes an agents preference-based router using the parameters `par`. This method sets the score type for contact and agent selection as well as random streams for randomized selections if it is enabled.

Parameters

`par` the router's parameters.

`router1` the router object.

`public String getRegion (int id)`

Returns the region name corresponding to region identifier `id`. This must be called after `initTypeRegions()` or `initGroupRegions()`, and throws a `NoSuchElementException` if no region name has been associated with the given identifier.

Parameter

`id` the region identifier.

Returns the corresponding region name.

Throws

`NoSuchElementException` if no region name is associated with the corresponding region identifier.

`public int getRegion (String regStr)`

Returns the region identifier corresponding to the region name `regStr`. This method must be called only after `initTypeRegions()` or `initGroupRegions()`, and throws a `NoSuchElementException` if no identifier is associated with `regStr`.

Parameter

`regStr` the tested region name.

Returns the corresponding region identifier.

Throws

`NoSuchElementException` if no region identifier is associated with the given region name.

`public void clearRegionMap()`

Clears the internal region map used by `initTypeRegions()` and `initGroupRegions()`. After this method is called, it is not possible to get the region name corresponding to the region identifiers.

`public void initTypeRegions()`

Initializes the call type region identifiers used by the local-specialist routing policy. This method obtains a region name for each call type, and maps each identical name to the same integer. At the end of this process, the array returned by `getTypeRegions()` associates a region identifier to each call type.

The region name of a call type is computed as follows. First, the call type factory is obtained using `CallCenter.getCallFactory (int)`. If the properties returned by `CallFactory.getProperties()` contains a property named `region`, its value is used as the region string. Otherwise, the name of the call type, returned by `CallFactory.getName()`, is split using the semicolon as a delimiter, and the region corresponds to the string following the semicolon.

`public void initGroupRegions()`

Initializes the agent group region identifiers used by the local-specialist routing policy. This method obtains a region name for each agent group, and maps each identical name to the same integer. At the end of this process, the array returned by `getGroupRegions()` associates a region identifier to each call type.

The region name of an agent group is computed as follows. First, the agent group manager is obtained using `CallCenter.getAgentGroupManager (int)`. If the properties returned by `getProperties()` contains a property named `region`, its value is used as the region string. Otherwise, the name of the call type, returned by `getName()`, is split using the semicolon as a delimiter, and the region corresponds to the string following the semicolon.

`public int[][] getGroupToTypeMap()`

Returns the currently used group-to-type map. If `initGroupToTypeMap (RouterParams)` or `setGroupToTypeMap (int[][])` were never called, this method returns `null`. Otherwise, this method returns an array of I arrays giving an order list of call types for each agent group.

Returns the currently used group-to-type map.

`public void setGroupToTypeMap (int[][] groupToTypeMap)`

Sets the group-to-type map to `groupToTypeMap`.

Parameter

`groupToTypeMap` the new group-to-type map.

`public boolean[][] getIncidenceMatrixGT()`

Returns the currently used group-to-type incidence matrix. If `initIncidenceMatrixGT (RouterParams)` or `setIncidenceMatrixGT (boolean[][])` were never called, this method returns `null`. Otherwise, this returns a $I \times K$ incidence matrix.

Returns the currently used group-to-type incidence matrix.

`public void setIncidenceMatrixGT (boolean[][] incidenceMatrixGT)`

Sets the group-to-type incidence matrix to `incidenceMatrixGT`.

Parameter

`incidenceMatrixGT` the group-to-type incidence matrix.

`public boolean[][] getIncidenceMatrixTG()`

Returns the currently used type-to-group incidence matrix. If `initIncidenceMatrixTG (RouterParams)` or `setIncidenceMatrixTG (boolean[][])` were never called, this method returns `null`. Otherwise, this returns a $K \times I$ incidence matrix.

Returns the currently used type-to-group incidence matrix.

`public void setIncidenceMatrixTG (boolean[][] incidenceMatrixTG)`

Sets the type-to-group incidence matrix to `incidenceMatrixTG`.

Parameter

`incidenceMatrixTG` the type-to-group incidence matrix.

`public double[] getQueueWeights()`

Returns the currently used queue weights vector. If `initQueueWeights (RouterParams)` or `setQueueWeights (double[])` were never called, this method returns `null`. Otherwise, element `k` of the returned array gives the weight for contact type `k` when entering in queue.

Returns the vector of queue weights.

`public void setQueueWeights (double[] queueWeights)`

Sets the vector of queue weights to `queueWeights`.

Parameter

`queueWeights` the new vector of queue weights.

`public double[][] getRanksGT()`

Returns the currently used group-to-type matrix of ranks. If `initRanksGT (RouterParams)` or `setRanksGT (double[][])` were never called, this method returns `null`. Otherwise, it returns a $I \times K$ matrix of ranks.

Returns the currently used group-to-type matrix of ranks.

`public void setRanksGT (double[][] ranksGT)`

Sets the group-to-type matrix of ranks to `ranksGT`.

Parameter

`ranksGT` the new matrix of ranks.

`public SortedMap<Double, double[][]> getRanksGTDelay()`

Returns a map giving the auxiliary matrices of ranks with associated minimal waiting times. Each entry of the returned map has a key giving the minimal waiting time, and a value corresponding to the matrix of ranks. If no auxiliary matrix of ranks were given in routing parameters, this returns an empty map.

Returns the map of auxiliary matrices of ranks.

`public double[][] getRanksTG()`

Returns the currently used type-to-group matrix of ranks. If `initRanksTG (RouterParams)` or `setRanksTG (double[][])` were never called, this method returns `null`. Otherwise, it returns a $K \times I$ matrix of ranks.

Returns the currently used type-to-group matrix of ranks.

`public void setRanksTG (double[][] ranksTG)`

Sets the type-to-group matrix of ranks to `ranksTG`.

Parameter

`ranksTG` the new matrix of ranks.

`public int[] getSkillCounts()`

Returns the currently used skill counts. This method returns `null` if `initSkillCounts (RouterParams)` or `setSkillCounts (int[])` were never called. Otherwise, it returns an array whose element `i` gives the skill count for agent group `i`.

Returns the currently used array of skill counts.

`public int getSkillCount (int i)`

Returns the skill count for agent group `i`, i.e., the number of call types agents in this group can serve. This method returns `Integer.MAX_VALUE` if `initSkillCounts (RouterParams)` or `setSkillCounts (int[])` were never called.

Parameter

`i` the index of the agent group.

Returns the skill count.

`public void setSkillCounts (int[] skillCounts)`

Sets the currently used skill counts to `skillCounts`.

Parameter

`skillCounts` the new skill counts.

`public int[] getTypeRegions()`

Returns the currently used type regions vector. This method returns `null` if `initTypeRegions()` or `setTypeRegions (int[])` were never called. Otherwise, index `k` of the returned array gives the region identifier for calls of type `k`.

Returns the vector of type regions.

`public void setTypeRegions (int[] typeRegions)`

Sets the vector of type regions to `typeRegions`.

Parameter

`typeRegions` the new vector of type regions.

`public int[] getGroupRegions()`

Returns the currently used group regions. This method returns `null` if `initGroupRegions()` or `setGroupRegions (int[])` were never called. Otherwise, index `i` of the returned array gives the region identifier for agents in group `i`.

Returns the currently used group regions.

`public void setGroupRegions (int[] groupRegions)`

Sets the currently used group regions to `groupRegions`.

Parameter

`groupRegions` the new group regions.

`public int[][] getTypeToGroupMap()`

Returns the currently used type-to-group map. If `initTypeToGroupMap (RouterParams)` or `setTypeToGroupMap (int[][])` were never called, this method returns `null`. Otherwise, it returns an array of K arrays giving an ordered list of agent groups for each call type.

Returns the currently used type-to-group map.

`public void setTypeToGroupMap (int[][] typeToGroupMap)`

Sets the currently used type-to-group map to `typeToGroupMap`.

Parameter

`typeToGroupMap` the new type-to-group map.

`public double[][] getWeightsGT()`

Returns the currently used group-to-type matrix of weights. If `initWeightsGT (RouterParams)` or `setWeightsGT (double[][])` were never called, this method returns `null`. Otherwise, it returns a $I \times K$ matrix of weights.

Returns the currently used group-to-type matrix of weights.

`public void setWeightsGT (double[][] weightsGT)`

Sets the group-to-type matrix of weights to `weightsGT`.

Parameter

`weightsGT` the new matrix of weights.

`public double[][] getWeightsTG()`

Returns the currently used type-to-group matrix of weights. If `initWeightsTG (RouterParams)` or `setWeightsTG (double[][])` were never called, this method returns `null`. Otherwise, it returns a $K \times I$ matrix of weights.

Returns the currently used type-to-group matrix of weights.

`public void setWeightsTG (double[][] weightsTG)`

Sets the type-to-group matrix of weights to `weightsTG`.

Parameter

`weightsTG` the new matrix of weights.

`public double[][] getDelaysGT()`

Returns the currently used group-to-type delays matrix. If `initDelays (RouterParams)` or setDelaysGT (double[][]) were never called, this method returns `null`. Otherwise, it returns a $I \times K$ matrix of delays expressed in the default time unit of the simulator.

Returns the currently used group-to-type delays matrix.

`public void setDelaysGT (double[][] delaysGT)`

Sets the group-to-type delays matrix to `delaysGT`.

Parameter

`delaysGT` the new delays matrix.

CallCenterRoutingStageInfo

Provides information on a routing stage, for the `OverflowAndPriorityRouter` router. The information includes a waiting time, and a list of routing cases which are used to compute the functions returning vectors of ranks.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallCenterRoutingStageInfo implements RoutingStageInfo
```

Constructor

```
public CallCenterRoutingStageInfo (CallCenter cc, int k,
                                   RoutingStageParams par)
```

Constructs call canter routing stage from the model `cc`, and parameters `par`.

Parameters

`cc` the call center model.

`par` the parameters for the routing stage.

RoutingCase

Represents a routing case part of a routing stage, for the `OverflowAndPriorityRouter`. A case is defined by a condition, represented by an instance of `Condition`, and vectors of ranks for agent selection, and queue priorities. An instance with condition set to `null` is also possible to represent the default case.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class RoutingCase
```

Constructors

`public RoutingCase (CallCenter cc, int k, RoutingCaseParams par)`

Constructs a new routing case using the call center model `cc`, and parameters `par`. The vectors of ranks are extracted directly from `par` while the condition is parsed with the help of `ConditionUtil.createCondition (CallCenter, int, ConditionParams)`.

Parameters

`cc` the call center model.

`k` the call type for which the routing case concerns.

`par` the case parameters.

`public RoutingCase (CallCenter cc, int k, DefaultCaseParams par)`

Similar to constructor `RoutingCase (CallCenter, int, RoutingCaseParams)`, for the default case with no condition.

`public RoutingCase (Condition cond, double[] aRanks, double[] qRanks)`

Creates a new routing case with condition `cond`, and vectors of ranks `aRanks` and `qRanks` for agent selection and queue priority.

Methods

`public Condition getCondition()`

Returns the condition associated with this case, or `null` for the default case.

`public double[] getAgentGroupRanks()`

Returns the vector of ranks for agent selection, for this routing case.

`public double[] getQueueRanks()`

Returns the vector of ranks for queue priority, for this routing case.

`public boolean isAgentGroupRanksRelative()`

Determines if the vector of ranks for agent groups is relative for this routing case.

`public boolean isQueueRanksRelative()`

Same as `isAgentGroupRanksRelative()`, for the vector of ranks of waiting queues.

CallTransferManager

Implements the necessary logic for call transfer from primary to secondary agents.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallTransferManager
```

VirtualHoldManager

Implements the necessary logic for virtual holding, also called virtual queueing.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class VirtualHoldManager
```

Constructor

```
public VirtualHoldManager (CallCenter cc) throws
                           CallCenterCreationException
```

Constructs a new virtual hold manager for the call center model `cc`.

Method

```
public void init()
```

Initializes the internal variables of this manager for a new simulation.

SegmentInfo

Represents information about a user-defined segment regrouping some indexed entities such as call types, agent groups, or periods. Each segment has a name, optional user-defined properties, and a list of indices.

Segment information is extracted from a `SegmentParams` instance which is read from a XML file by JAXB. The method `getValues()` can be used to obtain the indices regrouped by the segment, while `containsValue (int)` tests if a specific index is contained in the segment.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class SegmentInfo extends NamedInfo
```

Constructor

`public SegmentInfo (SegmentParams par)`

Constructs a new segment information object from the segment parameters `par`.

Parameter

`par` the segment parameters.

Throws

`IllegalArgumentException` if some segment parameters wre invalid.

Methods

`public int[] getValues()`

Returns the reference to an array containing the list of values in this segment. The returned array can be modified without affecting the internal array in this object.

Returns the list of indices.

`public int getNumValues()`

Returns the number of different values in this segment.

Returns the number of values.

`public int getMinValue()`

Returns the minimal index in this segment.

Returns the minimal index.

`public int getMaxValue()`

Returns the maximal index in this segment.

Returns the maximal index.

```
public boolean containsValue (int i)
```

Tests if the index `i` is included in the list of values associated with this segment. Returns `true` if and only if `i` is included in the list of values returned by `getValues()`.

Parameter

`i` the tested index.

Returns the success indicator of the test.

```
public static SegmentInfo[] getSegments (Collection<? extends
                                         SegmentParams> par)
```

Converts the given collection of segment parameters into an array of segment information objects. This method first creates an array of segment information objects whose length corresponds to the size of the given collection. It then iterates over the collection, and creates one information object for each parameter object in the collection. The constructed array is then returned.

Parameter

`par` collection of segment parameters.

Returns the corresponding array of segment information objects.

Throws

`IllegalArgumentException` if an error occurs during the creation of a segment information object.

```
public static void checkRange (int lower, int upper, SegmentInfo...
                               segments)
```

Checks that the minimal value stored in all the segments `segments` is greater than or equal to `lower`, and the maximal value is smaller than `upper`. If this condition is violated for at least one segment, an illegal-argument exception is thrown. This method is used for validating parameters when the call center model is constructed. For example, it is used to ensure that segments of inbound call types does not contain any value greater than or equal to K_{I}.

Parameters

`lower` the lower bound (inclusive).

`upper` the upper bount (non-inclusive).

`segments` the array of segments to test.

Throws

`IllegalArgumentException` if at least one segment contains an out-of-bounds value.

```
public static DoubleMatrix2D addRowSegments (DoubleMatrix2D mat,
                                             DoubleDoubleFunction func,
                                             SegmentInfo... segments)
```

Calls `addRowSegments (mat, func, null, segments)`.

Parameters

`mat` the matrix to process.

`func` the function f.

`segments` the segments for which rows are added in the matrix.

Returns the matrix with extra rows.

```
public static DoubleMatrix2D addRowSegments (DoubleMatrix2D mat,
                                             DoubleDoubleFunction func,
                                             boolean[] globalSegmentValues,
                                             SegmentInfo... segments)
```

Constructs and returns a matrix with all the rows in `mat`, extra rows corresponding to the segments in `segments`, and an additional row representing the aggregation of all rows in the original matrix. Let `mat` be a $a \times b$ matrix. If $a \leq 1$, the method returns `mat` unchanged. Otherwise, it creates a new matrix M with $a+s+1$ rows and b columns, where s is the length of the `segments` array. Let $m_{i,j}$ be the element in `mat` at position (i, j), for $i = 0, \ldots, a-1$ and $j = 0, \ldots, b-1$, and let $M_{i,j}$ be an element in the resulting matrix, with $i = 0, \ldots, a+s$, and $j = 0, \ldots, b-1$. Then, for any $j = 0, \ldots, b-1$,

$$M_{i,j} = \begin{cases} f(0, m_{i,j}) & \text{for } i = 0, \ldots, a-1, \\ f_{l=0}^{a-1} m_{l,j} s_{i-a,l} & \text{for } i = a, \ldots, a+s, \end{cases}$$

where

$$f_{i=a}^{b} x_i s_i = \begin{cases} f(f_{i=a}^{b-1} x_i s_i, x_b) & \text{if } a < b \text{ and } s_b = 1, \\ f_{i=a}^{b-1} x_i s_i & \text{if } a < b \text{ and } s_b = 0, \\ f(0, x_a) & \text{if } a = b \text{ and } s_a = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $f : \mathbb{R}^2 \to \mathbb{R}$ is a function, and $s_{r,i} = 1$ if index i is included in the rth segment, and 0 otherwise. For $r = 0, \ldots, s-1$, $s_{r,i}$ is determined using the `containsValue (int)` method of `segments[r]` while $s_{s,i}$ is 1 if and only if `globalSegmentValues[i]` is `true`. If `globalSegmentValues` is `null`, $s_{s,i}$ is set to 1 for all i in the last row.

Usually, `func` which represents f is set to `Functions.plus`, or `Functions.max`. In the former case, $f(x, y) = x + y$, and $M_{i,j}$ is

$$\sum_{l=0}^{a-1} m_{l,j} s_{i-a,l}.$$

Parameters

`mat` the matrix to process.

`func` the function f.

`globalSegmentValues` determines which rows are summed up in the global segment.

`segments` the segments for which rows are added in the matrix.

Returns the matrix with extra rows.

```
public static DoubleMatrix2D addRowSegments (DoubleMatrix2D mat, int
                                             numGroups,
                                             DoubleDoubleFunction func,
                                             boolean[]
                                             globalSegmentValues1, boolean[]
                                             globalSegmentValues2,
                                             SegmentInfo[] segments1,
                                             SegmentInfo[] segments2)
```

Constructs and returns a matrix with all the rows in `mat`, and extra rows corresponding to the segments in `segments1` and `segments2`. Let `mat` be a $(a*c)\times b$ matrix. If $a*c \leq 1$, the method returns `mat` unchanged. Otherwise, it creates a new matrix M with $(a+s_1+1)(c+s_2+1)$ rows and b columns, where s_1 and s_2 are the lengths of the `segments1` and `segments2` arrays, respectively. Let $m_{i,j,p}$ be element at row $i*c+j$ and column p in `mat`, with $i=0,\ldots,a-1$ and $j=0,\ldots,c-1$, and $p=0,\ldots,b-1$. Also let $M_{i,j,p}$ be element at row $i*(c+s_2-1)+j$ and column p in M, with $i=0,\ldots,a+s_1$, and $j=0,\ldots,c+s_2$. Then,

$$M_{i,j,p} = \begin{cases} f(0, m_{i,j,p}) & \text{if } i < a \text{ and } j < c, \\ f_{l=0}^{c-1} m_{i,l,p} s_{2,j-c,l} & \text{if } i < a \text{ and } j = c,\ldots,c+s_2, \\ f_{l=0}^{a-1} m_{l,j,p} s_{1,i-a,l} & \text{if } i = a,\ldots,a+s_1 \text{ and} j < c, \\ f_{l_1=0}^{a-1}(f_{l_2=0}^{c-1} m_{l_1,l_2,p} s_{2,j-c,l_2}) s_{1,i-a,l_1} & \text{otherwise.} \end{cases}$$

Here, $s_{d,r,i}$ is 1 if and only if segment r in dimension d contains element i, for $d=1,2$. In particular, $s_{1,r,i}$, for $r=0,\ldots,s_1-1$, is determined using `segments1[r]` while $s_{2,r,i}$, for $r=0,\ldots,s_2-1$, is set using `segments2[r]`. The variables $s_{1,s_1,i}$ and $s_{2,s_2,i}$ are set using `globalSegmentValues1[i]` and texttttglobalSegmentValues2[i] respectively, or 1 if the corresponding array is `null`. The definition of $f_{i=a}^{b} x_i s_i$ is the same as in method `addRowSegments (DoubleMatrix2D, DoubleDoubleFunction, boolean[], SegmentInfo...)`.

Parameters

`mat` the matrix to process.

`numGroups` the value of c, a being determined using `mat`.

`func` the function f.

`globalSegmentValues1` determines which rows are summed up in the global segment for the first dimension.

`globalSegmentValues2` determines which rows are summed up in the global segment for the second dimension.

`segments1` the segments for which rows are added in the matrix, for the first dimension.

`segments2` the segments for which rows are added in the matrix, for the second dimension.

Returns the matrix with extra rows.

```
public static DoubleMatrix2D addColumnSegments (DoubleMatrix2D mat,
                                                DoubleDoubleFunction func,
                                                SegmentInfo... segments)
```

Similar to `addRowSegments (DoubleMatrix2D, DoubleDoubleFunction, SegmentInfo...)`, for adding extra columns to matrix `mat`.

Parameters

`mat` the matrix to process.

`func` the function f.

`segments` the segments for which rows are added in the matrix.

Returns the matrix with extra rows.

CallCenterCreationException

This exception is thrown when a problem occurs during the creation of a call center model.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallCenterCreationException extends Exception
```

CallFactoryCreationException

This exception is thrown when a problem occurs during the creation of a call factory.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class CallFactoryCreationException extends Exception
```

ArrivalProcessCreationException

This exception is thrown when a problem occurs during the creation of an arrival process.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class ArrivalProcessCreationException extends Exception
```

DialerCreationException

This exception is thrown when a problem occurs during the creation of a dialer.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class DialerCreationException extends Exception
```

RouterCreationException

This exception is thrown when a problem occurs during the creation of the router.

```
package umontreal.iro.lecuyer.contactcenters.msk.model;

public class RouterCreationException extends Exception
```

Package umontreal.iro.lecuyer.contactcenters.msk.simlogic

Provides classes implementing the logic containing the necessary instructions to simulate a model of a call center, and interact with facilities for statistical collecting. Such a logic can simulate independent replications using a given model, or split a single long replication into time intervals. In both cases, the simulation can be divided into steps corresponding to either replication, either time intervals also called batches.

To perform an experiment, a simulation logic is first initialized, which resets the state of its associated model to an empty system. The logic can then be used to simulate a certain number of steps.

The simulation logic interacts with the statistical collecting facilities in the following ways. First, any simulation can provide a period index for each observed call. This period index often corresponds to the period of arrival of the call, but it can also be the period at which the service ends, a fixed value (for simulations with batch means), etc. A simulation logic makes matrices of counters available for statistical collectors. When such a matrix is required, the logic may perform some processing such as aggregating columns or normalizing values with respect to time. After each step, the simulation logic adds observations to statistical collectors.

The simulation logic is represented by an object implementing the `SimLogic` interface which inherits interfaces in the package `umontreal.iro.lecuyer.contactcenters.msk.stat` for interaction with statistical collecting facilities. This package provides two implementations of this interface: `RepLogic` for simulating independent replications, and `BatchMeansLogic` for simulations with batch means of a single period as if it was infinite in the model.

Moreover, an object implementing the `SimLogicListener` interface can be registered with a simulation logic and used to monitor the simulated steps. The `SimLogicBase` class also provides some support methods for simulation logics.

SimLogic

Represents a simulation logic performing a certain type of experiment on a model of a call center. This interface defines methods to perform simulations, obtain the statistical period of contacts, transform matrices of counters into matrices of observations ready to be added to statistical collectors, and update some simulation parameters.

```
package umontreal.iro.lecuyer.contactcenters.msk.simlogic;

public interface SimLogic extends StatPeriod
```

Methods

`public CallCenter getCallCenter()`

Returns the model associated with this simulation logic.

Returns the associated model.

`public SimParams getSimParams()`

Returns the parameters associated with this simulation logic.

Returns the associated parameters.

`public CallCenterMeasureManager getCallCenterMeasureManager ()`

Returns an object containing the counters updated throughout the simulation.

Returns the call center measures.

`public CallCenterStatProbes getCallCenterStatProbes()`

Returns the call center statistical probes used by this simulation logic.

Returns the call center statistical probes.

`public void reset (PerformanceMeasureType... pms)`

Resets the simulation logic for a new experiment after the model has been reset. This method should update or recreate the associated counters and statistical probes, since the size of the model may have changed.

`public void init()`

Initializes the simulation logic for a new experiment. In particular, this resets the event list of the simulator, the state of the model, and the current number of completed steps to 0.

`public void simulate (int numSteps)`

Simulates `numSteps` steps, and updates observations in statistical collectors as well as the number of completed steps returned by `getCompletedSteps()`. Usually, this method simulates the required number of replications, and adds one observation to each statistical collector of the matrices returned by `getCallCenterStatProbes()`.

Note that this method may be called several times during a simulation experiment using sequential sampling. For this reason, one should take account of every observation collected since the last call to `init()`.

`public int getCompletedSteps()`

Returns the number of completed simulation steps.

Returns the number of completed steps.

`public boolean isSteadyState()`

Determines if this simulator performs a steady-state simulation.

Returns `true` if this is a steady-state simulator, `false` otherwise.

`public void formatReport (Map<String, Object> evalInfo)`

Adds the information specific to this simulation logic into the evaluation information map of the simulator. The keys and values of this map are listed at the beginning of the simulation report.

`public int[] getStaffing()`

Returns the staffing vector used by this simulator. This vector has the same format as the `EvalOptionType.STAFFINGVECTOR` evaluation option.

Returns the staffing vector.

`public void setStaffing (int[] staffing)`

Sets the staffing vector used by this simulator to `staffing`. This vector has the same format as the `EvalOptionType.STAFFINGVECTOR` evaluation option.

Parameter

`staffing` the new staffing vector.

`public int[][] getStaffingMatrix()`

Gets the staffing matrix for the simulated model. The returned 2D array has the format specified by `EvalOptionType.STAFFINGMATRIX`.

Returns the 2D array representing the staffing matrix.

`public void setStaffingMatrix (int[][] staffing)`

Sets the 2D array representing the staffing matrix to `staffing`.

Parameter

`staffing` the new staffing matrix.

`public int[][] getScheduledAgents()`

Returns the 2D array of scheduled agents for each shift and each agent group. Element (i, j) of the returned array contains the number of agents scheduled in group i during shift j.

Returns the scheduled agents.

`public void setScheduledAgents (int[][] ag)`

Sets the number of scheduled agents for each group and shift using the given 2D array.

Parameter

`ag` the array of scheduled agents.

`public int getCurrentMainPeriod()`

Returns the current period used by this simulator. If this simulator is not steady-state, this throws an `UnsupportedOperationException`.

Returns the current period.

`public void setCurrentMainPeriod (int mp)`

Sets the current period for this simulator to `p`. If this simulator is not steady-state, this throws an `UnsupportedOperationException`.

Parameter

`mp` the new current period.

`public boolean seemsUnstable()`

Returns `true` if, after the simulation, the system seems unstable. This is applicable for steady state simulations only.

Returns the result of the stability check.

`public void registerListeners()`

Registers any listener required by the simulator from the model.

`public void unregisterListeners()`

Disconnects every listener registered by the simulator from the model.

`public boolean isVerbose()`

Determines if the simulation logic is in verbose mode.

Returns the status of the verbose mode.

`public void setVerbose (boolean verbose)`

Sets the verbose indicator to `verbose`.

Parameter

`verbose` the value of the indicator.

`public void addSimLogicListener (SimLogicListener l)`

Registers the listener `l` to be notified about the progress of the simulator.

Parameter

`l` the listener to be notified.

Throws

`NullPointerException` if `l` is `null`.

`public void removeSimLogicListener (SimLogicListener l)`

Removes the listener `l` from the list of listeners registered with this simulator.

Parameter

`l` the listener being removed.

`public void clearSimLogicListeners()`

Removes all the listeners registered with this simulator.

`public List<SimLogicListener> getSimLogicListeners()`

Returns the listeners registered with this simulator.

Returns the list of registered listeners.

`public boolean isAborted()`

Determines if the simulation has been aborted by using the `setAborted (boolean)` method.

Returns `true` if the simulation was aborted, `false` otherwise.

`public void setAborted (boolean aborted)`

Aborts the current simulation.

SimLogicListener

Represents an observer of the progress of a simulation.

```
package umontreal.iro.lecuyer.contactcenters.msk.simlogic;

public interface SimLogicListener
```

Method

`public void stepDone (SimLogic sim)`

Indicates that a step was done by the simulator `sim`. One can use `ContactCenterSim.getCompletedSteps()` to obtain the number of completed steps.

Parameter

`sim` the contact center simulation logic.

SimLogicBase

Provides some basic methods for implementing the `SimLogic` interface. This class encapsulates a boolean variable indicating if the simulation was aborted by some thread as well as a list of observers notified at each simulation step.

```
package umontreal.iro.lecuyer.contactcenters.msk.simlogic;

public class SimLogicBase
```

RepLogic

Implements the logic for a simulation with independent replications. For each replication, this logic initializes the model to an empty state, and simulates the entire horizon, i.e., a single day, week, month, etc., depending on the model's parameters. Statistics are collected in every period.

```
package umontreal.iro.lecuyer.contactcenters.msk.simlogic;

public class RepLogic extends RepSim
             implements SimLogic
```

Constructor

`public RepLogic (CallCenter cc, RepSimParams simParams, PerformanceMeasureType... pms)`

Constructs a new simulation logic for independent replications, using the model `cc`, the simulation parameters `simParams`, and estimating performance measures of all types listed in `pms`.

Parameters

`cc` the simulated model.

`simParams` the simulation parameters.

`pms` the estimated performance measures.

Methods

`public int getNumPeriodsForCounters()`

This method returns $P + 2$, the number of periods.

`public int getNumPeriodsForCountersAwt()`

This method returns P', the number of segments regrouping main periods.

`public int getStatPeriod (Contact contact)`

By default, this returns the period of arrival of the given contact. However, this can be changed using the `PerPeriodCollectingMode` attribute in the `repSimParams` parameter file.

`public int getStatPeriodAwt (Contact contact)`

Returns the result of `getAwtPeriod (Contact)`.

`public int getStatPeriod()`

Returns the index of the current period.

`public int getAwtPeriod (Contact contact)`

Computes the statistical period p of the contact by calling `getStatPeriod (Contact)`, and converts p to a main period using `PeriodChangeEvent.getMainPeriod (int)`.

`public int getGlobalAwtPeriod()`

This returns $P' - 1$.

BatchMeansLogic

Implements the logic for a simulation with batch means. This logic simulates a single long replication which is divided into time intervals called batches. The logic uses matrices of counters with a single column for storing values for the current real batch. In the notation of the super class `BatchMeansSim`, these counters are used to generate the $\mathbf{V}_j$ vectors containing statistics for real batches. When batch aggregation is disabled, these counters are used directly to make the matrices of observations which correspond to the $\mathbf{X}_r$ vectors. When batch aggregation is enabled, the vectors of counts for each real batch are added to intermediate lists of statistical probes, and matrices of observations are constructed by aggregating some of these vectors. The operator used for aggregation is the sum, but it can also be the maximum for some statistics such as the maximal number of busy agents, maximal queue size, and maximal waiting time. For more information about batch aggregation, see the documentation of the super class `BatchMeansSim`.

```
package umontreal.iro.lecuyer.contactcenters.msk.simlogic;

public class BatchMeansLogic extends BatchMeansSim
             implements SimLogic
```

Constructor

```
public BatchMeansLogic (CallCenter cc, BatchSimParams simParams,
                        PerformanceMeasureType... pms)
```

Constructs a new simulation logic for batch means, using the model `cc`, the simulation parameters `simParams`, and estimating performance measures of all types listed in `pms`.

Parameters

`cc` the simulated model.

`simParams` the simulation parameters.

`pms` the estimated performance measures.

Methods

```
public int getNumPeriodsForCounters()
```

Returns 1.

```
public int getNumPeriodsForCountersAwt()
```

Returns 1.

```
public int getStatPeriod (Contact contact)
```

Returns the result of `getStatPeriod()`.

`public int getStatPeriod()`

Returns 0 if the warmup period is over, or -1 otherwise.

`public int getStatPeriodAwt (Contact contact)`

Returns the same value as `getStatPeriod (Contact)`.

`public int getAwtPeriod (Contact contact)`

This returns $P' - 1$.

`public int getGlobalAwtPeriod()`

This returns $P' - 1$.

`public int computeMaxQueueSizeThresh()`

Computes and returns the maximal queue size threshold before a simulated system is declared unstable. By default, this returns $20000 + 1000\sqrt{N}$ where N is the total number of agents.

Returns the maximal queue size threshold.

Package umontreal.iro.lecuyer.contactcenters.msk.stat

Provides utility classes to manage statistics in the blend/multi-skill call center simulator. The system for managing statistics is split into two parts: counters updated throughout the simulation, and collectors updated only at the end of steps using the values of counters. Counters and collectors are regrouped into matrices whose rows correspond to call types, agent groups, or (call type, agent group) pairs, and columns represent time intervals. We now examine how counters and collectors are managed and interact in more details.

The abstract class `CallCenterMeasureManager` represents the matrices of counters. It can be used to list the supported types of measures, and return matrices of values for any supported type. Getting a matrix of values is done by reading the corresponding counters, and performing some computations such as regrouping periods or normalizing with respect to time. The exact computation depends on application and thus on the concrete subclass. Usually, the matrices of counters contain one column per period, and matrices of statistical collectors have one column per main period, plus an extra column representing the whole horizon.

The measure manager also encapsulates some observers linked to the call center model in order to collect the appropriate statistics. These observers use an instance of `StatPeriod` to obtain the statistical period of any processed call. The call center measure manager also includes an instance of `CallByCallMeasureManager`, which regroups every counter containing sums with one (possibly 0) term for each simulated call.

On the other hand, the interface `CallCenterStatProbes` represents a set of matrices of statistical collectors. The most common implementation of this interface is `SimCallCenterStat` which provides a method `addObs` to add matrices of counters, obtained using an instance of `CallCenterMeasureManager`, to the corresponding matrices of collectors. Other implementations of the interface can be used to collect statistics about statistics, e.g., averages of averages, variances, etc., combine the information given by two instances of `SimCallCenterStat`, etc. This can be used to apply some variance reduction techniques such as stratification and randomized quasi-Monte Carlo methods.

AWTPeriod

Represents an object capable of computing a period index to get the acceptable waiting time of a contact. In general, the acceptable waiting time may depend on the call type and a period index. The period index often corresponds to the period of arrival, but it can be set to a fixed value in some cases. An implementation of this interface maps a contact object to a period index corresponding to the correct acceptable waiting time.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public interface AWTPeriod
```

Methods

`public int getAwtPeriod (Contact contact)`

Returns the index of the main period for the acceptable waiting time of contact `contact`. The returned index must not be smaller than 0 or greater than P, where P is the number of main periods. If this method returns P, the acceptable waiting time for all periods is used.

Parameter

`contact` the contact being queried.

Returns the main period index for the acceptable waiting time.

`public int getGlobalAwtPeriod()`

Returns the index for the acceptable waiting time for all periods.

Returns the main period index for the acceptable waiting time.

StatPeriod

Represents an object capable of assigning a statistical period to any observed call. An object implementing this interface is used by `CallCenterMeasureManager` and other associated observers to separate calls in periods.

Note that the values returned by `getNumPeriodsForCounters()`, `getNumPeriodsForCountersAwt()`, and `needsSlidingWindows()` should never change from call to call, for a given object implementing this interface.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public interface StatPeriod extends AWTPeriod
```

Methods

`public int getNumPeriodsForCounters()`

Returns the number of periods in usual matrices of counters updated throughout the simulation. Usually, this corresponds to $P + 2$, the total number of periods, but this returns 1 for a steady-state simulation over a single period.

Returns the number of periods for matrices of counters.

`public int getNumPeriodsForCountersAwt()`

Similar to `getNumPeriodsForCounters()`, for matrices of counters using acceptable waiting times. This usually returns P', the total number of segments regrouping main periods. But this returns 1 for a steady-state simulation over a single period.

Returns the number of periods for matrices of counters.

`public int getStatPeriod (Contact contact)`

Returns the statistical period of a contact `contact`. If a negative index is returned for a given contact, this contact is not counted in statistics. This often corresponds to the period during which the contact arrives, but this always returns 0 for steady-state simulations.

`public int getStatPeriodAwt (Contact contact)`

Similar to `getStatPeriod (Contact)`, for a statistic using an acceptable waiting time. If a negative index is returned for a given contact, this contact is not counted in statistics. Often, this returns `getStatPeriod (Contact)` minus 1.

`public boolean needsStatForPeriodSegmentsAwt()`

Determines if statistics for segments regrouping main periods are collected for measure types using acceptable waiting times. Usually, statistics are collected for each main period, and sums are computed at a later time if needed. However, statistics based on acceptable waiting times cannot be summed, because the AWT may change from periods to periods in general. This method thus indicates if observers must collect observations for groups of main periods in addition to the statistical periods of calls.

`public int getStatPeriod()`

Returns the default statistical period. This usually corresponds to the current period.

`public boolean needsSlidingWindows()`

Determines if sliding windows are needed by statistical counters using an object implementing this interface to get the statistical periods of calls. Usually, the period index returned by `getStatPeriod (Contact)` is never greater than the integer returned by `getNumPeriodsForCounters()`, and the same relationship holds for `getStatPeriodAwt (Contact)` and `getNumPeriodsForCountersAwt()`. However, this assumption can be violated if one needs to get real-time statistics concerning the last observed periods. In such cases, matrices of counters need to be implemented using sliding windows: when the index a statistical period becomes higher than the number of stored periods, the first periods are discarded. This method determines if such sliding windows are needed.

MeasureType

Defines the types of matrices of measures, or raw statistics, supported by the call center simulator. During simulation, matrices of counters are updated in order to get matrices of observations which are added to statistical probes. Each matrix of counters regroups counts for a certain type of measure, e.g., the number of served calls, the sum of waiting times, the total time spent by busy agents, etc. Each row of such a matrix concerns a call type, agent group or (call type, agent group) pair, while each column concerns a period. If a single period is simulated, all matrices contain a single column.

There are two types of matrices of counters: a regular type for most statistics, and a special type for statistics based on an acceptable waiting time. Regular matrices have $P + 2$ columns, e.g., one column per period, and a certain number of R of rows. When such a matrix of counters is updated, only one element is changed; this ensures that the matrix update does not take too much time. When the matrix is transformed into a matrix of observations, only results for the P main periods are retained, and aggregates are computed for segments regrouping main periods. Aggregates are also computed for rows, which results in the matrix of observations having extra rows.

Matrices of counters using acceptable waiting times are different, because rows and columns cannot be aggregated to make matrices of observations. Aggregation cannot be done, because each counter may be updated with a different acceptable waiting time in general.

This type can be determined for any enum constant by getting its associated row type, using `getRowType (false)`. The matrix type is AWT-based only if its associated row type is `RowType.INBOUNDTYPEAWT`.

The operator used for aggregation is often the sum, but this can also be the maximum for some types of measures. This operator can be obtained using the `getAggregation-Function()`.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public enum MeasureType
```

Constants

`MAXBUSYAGENTS`

`MAXQUEUESIZE`

`MAXWAITINGTIMEABANDONED`

`MAXWAITINGTIMESERVED`

NUMABANDONED

NUMABANDONEDAFTERAWT

NUMABANDONEDBEFOREAWT

NUMARRIVALS

NUMBLOCKED

NUMBUSYAGENTS

NUMDELAYED

NUMSCHEDULEDAGENTS

NUMSERVED

NUMSERVEDAFTERAWT

NUMSERVEDBEFOREAWT

NUMTRIEDDIAL

NUMWORKINGAGENTS

NUMWRONGPARTYCONNECTS

QUEUESIZE

SUMEXCESSTIMESABANDONED

SUMEXCESSTIMESSERVED

SUMSERVED

SUMSERVICETIMES

SUMWAITINGTIMESABANDONED

SUMWAITINGTIMESSERVED

SUMWAITINGTIMESVQABANDONED

`SUMWAITINGTIMESVQSERVED`

`SUMSQUAREDIFFESTREALWAITINGTIMESSERVED`

`SUMSQUAREDIFFESTREALWAITINGTIMESABANDONED`

`SUMSQUAREDIFFESTREALWAITINGTIMESVQABANDONED`

`SUMSQUAREDIFFESTREALWAITINGTIMESVQSERVED`

Methods

`public DoubleDoubleFunction getAggregationFunction()`

Returns the functions which is applied in order to aggregate two values of counters of this type. This usually returns `Functions.plus`, but this can also return `Functions.max` for example with `MAXWAITINGTIMEABANDONED`.

`public RowType getRowType (boolean contactTypeAgentGroup)`

Returns the row type for this type of measure. If `contactTypeAgentGroup` is `true`, this returns the row type when statistics are collected separately for (call type, agent group) pairs. Otherwise, this returns the row type when statistics are counted only for call types.

Parameter

`contactTypeAgentGroup`

Returns the row type for this measure type.

`public TimeNormalizeType getTimeNormalizeType()`

Returns a constant indicating how time normalization should be perform on matrix of counters of this type.

CallCenterMeasureManager

Encapsulates the matrices of counters collecting observations during simulation, and provides methods to determine which types of counters are supported, and to extract matrices of observations from the counters.

This class encapsulates observers used to update counters. Therefore, any instance of this class should be registered with the call center using the `registerListeners()` for listeners to be registered.

Each matrix of counters has a type represented by an enum constant in `MeasureType`. This type determines the role played by rows in the matrix of counters. The user can determine for which type of measures statistics are collected by giving a list of `MeasureType` instances to the constructor of `CallCenterMeasureManager`. This list can be retrieved by using the `getMeasures()` method.

The columns correspond to time intervals which are determined with the help of a `StatPeriod` implementation. Such an implementation gives the number of needed time intervals as well as a function mapping each contact, and each simulation time, to one of the columns. Usually, there is one column per period. The `StatPeriod` implementation of a measure manager can be obtained using the `getStatPeriod()` method.

The raw matrices of counters can be obtained using //important de noter the `getMeasureMatrix (MeasureType)` method. However, most measure managers regroup periods and normalizes values with respect to time in order to prepare matrices of observations for statistical collectors. This preparation is performed by the method `getValues (MeasureType, boolean)`.

The number of columns in the matrices of observations, the way periods are regrouped, and how time is normalized are determined by the subclass implementing the `getNumPeriodsForStatProbes()`, `getValues (MeasureType, boolean)`, and `timeNormalize (MeasureType, DoubleMatrix2D)` abstract methods. These methods need to be overridden by a concrete subclass.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public abstract class CallCenterMeasureManager
```

Constructors

`public CallCenterMeasureManager (CallCenter cc, StatPeriod statP, boolean contactTypeAgentGroup)`

Creates a measure manager for all possible types of measures on the call center model `cc`, and using `statP` to obtain the statistical periods of calls. The boolean `contactTypeAgentGroup` is used to determine if matrices of counters contain rows of type (call type, agent group). See the constructor `CallByCallMeasureManager.CallByCallMeasureManager (CallCenter, StatPeriod, boolean)` for more information about this.

Parameters

`cc` the call center model.

`statP` the object used to obtain statistical periods.

`contactTypeAgentGroup` determines if rows of type (call type, agent group) are needed.

```
public CallCenterMeasureManager (CallCenter cc, StatPeriod statP, boolean
                                 contactTypeAgentGroup,
                                 PerformanceMeasureType[] pms)
```

Similar to `CallCenterMeasureManager (CallCenter, StatPeriod, boolean)`, for a given subset of the types of performance measures. The subset is obtained by calling the `getMeasureTypes (PerformanceMeasureType...)` static method.

```
public CallCenterMeasureManager (CallCenter cc, StatPeriod statP, boolean
                                 contactTypeAgentGroup, Collection<
                                 MeasureType> measures)
```

Similar to `CallCenterMeasureManager (CallCenter, StatPeriod, boolean)`, for a given collection of measure types.

Methods

```
public static PerformanceMeasureType[] getSupportedPerformanceMeasures
()
```

Returns the array of all types of performance measures supported by this measure manager.

```
public static MeasureType[] getMeasureTypesPm (PerformanceMeasureType pm)
```

Returns the types of counters needed to estimate the particular type of performance measure `pm`.

Parameter

`pm` the tested type of performance measure.

Returns the array of needed types of counters.

```
public static Set<MeasureType> getMeasureTypes (PerformanceMeasureType...
                                                pms)
```

Returns the types of counters needed to estimate all the performance measures in `pms`.

Parameter

`pms` the tested types of performance measures.

Returns the set of measure types.

`public StatPeriod getStatPeriod()`

Returns the object determining how columns of matrices of counters are mapped to time intervals.

`public CallByCallMeasureManager getCallByCallMeasureManager ()`

Returns the call-by-call measure manager used by this object.

`public boolean isContactTypeAgentGroup()`

Returns `true` if this group of call center measures contains matrices whose rows correspond to counters concerning (contact type, agent group) pairs. If no matrix with rows of type (contact type, agent group) is present, this returns `false`.

`public boolean hasMeasureMatricesFor (PerformanceMeasureType pm)`

Determines if this simulator computes the measure matrices required to estimate performance measures of type `pm`.

Parameter

`pm` the tested type of performance measures.

Returns `true` of the measures can be estimated, `false` otherwise.

`public void initMeasureMatrices()`

Initializes the measure matrices defined by this object.

`public void finishCurrentPeriod()`

Indicates the end of the current statistical period, whose index p is returned by `StatPeriod.getStatPeriod()`. This method updates the columns p of matrices of counters containing integrals with respect to simulation time. These matrices contain, for example, the time-average queue size, time-average number of busy agents, etc.

`public void updateCurrentPeriod()`

Updates the current statistical period. For any period p preceding the current statistical period, this method fills up the columns p of matrices of counters containing integrals with respect to simulation time. It also initializes the maximal queue size and maximal number of busy agents for the current statistical period.

`public MeasureType[] getMeasures()`

Returns an array containing all the measure types supported by this object.

Returns an array of measure types.

`public boolean hasMeasureMatrix (MeasureType mt)`

Determines if this object has a measure matrix for the measure type `mt`.

Parameter

`mt` the tested measure type.

Returns `true` if and only if a measure matrix of the tested type is available.

`public MeasureMatrix getMeasureMatrix (MeasureType mt)`

Returns the measure matrix corresponding to the measure type `mt`. This method is mainly used by the `getValues (MeasureType, boolean)` method of subclasses. One should call `getValues (MeasureType, boolean)` instead to get matrices of counters from measure types.

Parameter

`mt` the tested measure type.

Returns the measure matrix.

`public IntegralMeasureMatrix<GroupVolumeStatMeasureMatrix>[] getGroupVolumeStats()`

Returns the array of integral measure matrices used to compute measures related to agent groups. Each element of this array corresponds to an agent group.

Returns the integral measure matrices for agent groups.

`public IntegralMeasureMatrix<QueueSizeStatMeasureMatrix>[] getQueueSizeIntegralStats()`

Return the array of integral measure matrices used to compute queue sizes. Each element of this array corresponds to a waiting queue.

Returns the integral measure matrices for waiting queues.

`public void registerListeners()`

Registers listeners required to get statistics during simulation.

`public void unregisterListeners()`

Unregisters listeners required to get statistics during simulation.

`public abstract int getNumPeriodsForStatProbes()`

Returns the number of periods in matrices of statistical probes used to collect statistics about the simulation. This usually returns P', the number of segments regrouping main periods. However, for steady-state simulations, this returns 1.

Returns the number of periods for statistics.

`public abstract DoubleMatrix2D getValues (MeasureType mt, boolean norm)`

Converts a matrix of counters constructed during the simulation to a matrix of double-precision observations to be added to a matching matrix of tallies. The format of raw measures stored into the matrix of counters is specific to the simulation type. This method formats these measures into a matrix with one row for each measure type, and one column for each segment of main periods.

If `norm` is `true`, the measures are normalized to the default time unit if they correspond to durations. This normalization is performed by calling `timeNormalize (MeasureType, DoubleMatrix2D)`. Otherwise, time durations are relative to the length of the corresponding period.

Matrices of counters have a number of periods depending on the type of measures collected. The output matrix of observations has `getNumPeriodsForStatProbes()` columns. See the documentation of `MeasureType` for more information about measure types.

Parameters

`mt` the measure type queried.

`norm` determines if normalization to default time unit is done.

Returns the matrix of values.

`public abstract void timeNormalize (MeasureType mt, DoubleMatrix2D m)`

Normalizes the measures in `m` using simulation time. This method must normalize time durations to the default simulation time unit by dividing every value by the correct period duration. The given matrix should have `getNumPeriodsForStatProbes()` columns.

Parameters

`mt` the type of measure being processed.

`m` the matrix of values, obtained by `getValues (MeasureType, boolean)`.

CallByCallMeasureManager

Contains and updates call-by-call measures for a call center model. This includes the number of arrivals, the number of services, etc. Any object of this class encapsulates matrices of sums for each type of call-by-call measure. It is also an exited-contact listener which can be notified each time a call leaves the system, for statistical collecting.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class CallByCallMeasureManager implements ExitedContactListener
```

Constructors

```
public CallByCallMeasureManager (CallCenter cc, StatPeriod statP, boolean
                                 contactTypeAgentGroups)
```

Constructs an observer for all supported types of call-by-call measures, for the call center model `cc`, and using `statP` to obtain the statistical period of each counted call.

Many counters concerning a call type can be separated into I counters, one for each agent group. This can be useful to obtain statistics concerning specific (call type, agent group) pairs, but this requires more memory. The boolean argument `contactTypeAgentGroups` determines if this separation is needed. If (call type, agent group) statistics are needed, this argument is `true`. Otherwise, it is `false`.

Parameters

`cc` the call center model.

`statP` the object used to get statistical periods of calls.

`contactTypeAgentGroups` determines if statistics for (call type, agent group) pairs are needed.

```
public CallByCallMeasureManager (CallCenter cc, StatPeriod statP, boolean
                                 contactTypeAgentGroups, Collection<
                                 MeasureType> measures)
```

Similar to constructor `CallByCallMeasureManager (CallCenter, StatPeriod, boolean)`, but restricts the counters to the given collection of measure types.

Methods

```
public StatPeriod getStatPeriod()
```

Returns the simulation logic associated with this object.

Returns the associated simulation logic.

`public boolean isContactTypeAgentGroup()`

Returns `true` if this group of call center measures contains matrices whose rows correspond to counters concerning (contact type, agent group) pairs. If no matrix with rows of type (contact type, agent group) is present, this returns `false`.

`public void initMeasureMap (Map<MeasureType, MeasureMatrix> measureMap)`

Initializes the given map `measureMap` with the measure matrices declared by this class. Each key of the map must be an instance of `MeasureType` while values are instances of `MeasureMatrix`.

Parameter

`measureMap` the map to be initialized.

`public void init()`

Initializes every measure matrices defined by this object.

BusyAgentsChecker

Computes the maximal number of busy agents for every agent group and statistical period, during the simulation. An object of this class registers as a listener for every agent group. Each time a contact enters service, the object then checks that the number of busy agents is not greater than the current maximum, and updates the maximum if necessary. When the model is simulated over multiple periods, such maxima are computed for each period. A busy-agents checker is also a period-change listener, because at the beginning of periods, it needs to set the per-period initial maxima to the current number of busy agents.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public final class BusyAgentsChecker implements AgentGroupListener,
                   MeasureMatrix
```

Constructor

`public BusyAgentsChecker (CallCenter cc, StatPeriod statP)`

Constructs a new busy-agents checker using call center `cc`, and object `statP` to obtain statistical periods.

Methods

`public void init()`

Initializes the counters to 0.

`public void register()`

Registers this busy-agents checker with the associated call center model. The method adds this object to the list of observers for all agent groups of the model, and registers itself as a period-change listener.

`public void unregister()`

Unregisters this busy-agents checker with the associated model. This method performs the reverse task of `register()`.

QueueSizeChecker

Computes the maximal queue size for every waiting queue and statistical period, during the simulation. An object of this class registers as a listener for every waiting queue of the model. Each time a contact enters a queue, the object checks that the queue size is not greater than the current maximum, and updates the maximum if necessary. When the model is simulated over multiple periods, such maxima are computed for each period. A queue-size checker is also a period-change listener, because at the beginning of periods, it needs to set the per-period initial maxima to the current queue size.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public final class QueueSizeChecker implements WaitingQueueListener,
                   MeasureMatrix
```

Constructor

`public QueueSizeChecker (CallCenter cc, StatPeriod statP)`

Constructs a new queue-size checker using call center `cc`, and object `statP` to obtain statistical periods.

Methods

`public void init()`

Resets the values of maxima to 0.

`public void register()`

Registers this queue-size checker with the associated call center model. The method adds this object to the list of observers for all waiting queues of the model, and registers itself as a period-change listener.

`public void unregister()`

Unregisters this queue-size checker with the associated model. This method performs the reverse task of `register()`.

CallCounter

Defines a new-contact listener for counting calls. This encapsulates a measure matrix with K rows and a column for each statistical period. Each time a new contact is notified, the element with row k and column p is incremented, where k is the type of the new contact and p is its statistical period.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class CallCounter implements NewContactListener
```

Constructor

`public CallCounter (CallCenter cc, StatPeriod statP, MeasureType mt)`

Constructs a new call counter using call center `cc`, for type of measure `mt`, and using `statP` to obtain statistical periods. The measure type is used to determine if we have a measure using AWT, for which statistical periods are different than with regular measures.

Parameters

`cc` the call center model.

`statP` the object for obtaining statistical periods of calls.

`mt` the type of measure for the counter.

Methods

`public SumMatrix getCount()`

Returns the matrix containing the counts.

`public void init()`

Initializes the call counter.

OutboundCallCounter

Defines a new-contact listener that counts the number of outbound calls. This object encapsulates a measure matrix containing K_O lines. When a contact of type k is notified, it is added in row $k - K_I$ of the matrix, and column corresponding to its statistical period.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class OutboundCallCounter implements NewContactListener
```

Constructor

`public OutboundCallCounter (CallCenter cc, StatPeriod statP)`

Constructs a new call counter for the call center model `cc`, and using `statP` to get statistical periods of calls.

Parameters

`cc` the call center model.

`statP` the object for obtaining statistical periods.

Methods

`public SumMatrix getCount()`

Returns the sum matrix that contains the counts.

Returns the sum matrix.

`public void init()`

Initializes the sum matrix for counting contacts.

CallCenterStatProbes

Encapsulates collectors containing statistics about a simulated call center. This interface specifies a method mapping types of performance measures to matrices of statistical probes. These matrices are constructed and updated internally by the implementation. The updating method, which is implementation-specific, often uses another set of call center probes, or measures from a simulation logic.

The main implementation of this interface is `SimCallCenterStat`, which uses an instance of `CallCenterMeasureManager` to obtain observations for statistical collectors.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public interface CallCenterStatProbes
```

Methods

`public void init()`
Initializes the statistical collectors contained in this object.

`public PerformanceMeasureType[] getPerformanceMeasures()`
Returns the types of performance measures contained into the implemented set of call center probes. If the implementing group of probes does not contain any matrix of statistical probes, this method must return an array with length 0 rather than `null`.

Returns the supported types of performance measures.

`public boolean hasPerformanceMeasure (PerformanceMeasureType pm)`
Determines if the implementing set of call center probes contains a matrix of probes for the performance measure `pm`. This method returns `true` if and only if `getPerformanceMeasures()` returns an array containing `pm`.

Parameter

`pm` the type of performance measure.

Returns `true` if the measures are computed by the simulator, `false` otherwise.

`public Map<PerformanceMeasureType, MatrixOfStatProbes<?>> getMatricesOfStatProbes()`
Returns a map containing the matrix of statistical probes for each type of performance measure.

Returns the map of statistical probes.

`public MatrixOfStatProbes<?> getMatrixOfStatProbes (PerformanceMeasureType pm)`
Returns a matrix of statistical probes corresponding to the given type `pm` of performance measure. If the type `pm` is not supported, this method throws a `NoSuchElementException`.

Parameter

`pm` the type of performance measure.

Returns the matrix of statistical probes.

Throws

`NoSuchElementException` if the type of performance measure is not supported.

`public MatrixOfTallies<?> getMatrixOfTallies (PerformanceMeasureType pm)`

Returns a matrix of tallies corresponding to the given type `pm` of performance measure. This method usually calls `getMatrixOfStatProbes (PerformanceMeasureType)` and casts the results into a matrix of tallies.

Parameter

`pm` the type of performance measure.

Returns the matrix of tallies.

Throws

`NoSuchElementException` if the type of performance measure is not supported.

`public MatrixOfTallies<TallyStore> getMatrixOfTallyStores (PerformanceMeasureType pm)`

Returns a matrix of tallies corresponding to the given type `pm` of performance measure. This method usually calls `getMatrixOfStatProbes (PerformanceMeasureType)` and casts the results into a matrix of tallies that can store their observations.

Parameter

`pm` the type of performance measure.

Returns the matrix of tallies.

Throws

`NoSuchElementException` if the type of performance measure is not supported.

`public MatrixOfFunctionOfMultipleMeansTallies<?> getMatrixOfFunctionOfMultipleMeansTallies (PerformanceMeasureType pm)`

Returns a matrix of function of multiple means tallies corresponding to the given type `pm` of performance measure. This method usually calls `getMatrixOfStatProbes (PerformanceMeasureType)` and casts the results into a matrix of tallies.

Parameter

`pm` the type of performance measure.

Returns the matrix of tallies.

Throws

`NoSuchElementException` if the type of performance measure is not supported.

```
public DoubleMatrix2D getAverage (PerformanceMeasureType pm)

public DoubleMatrix2D getVariance (PerformanceMeasureType pm)

public DoubleMatrix2D getVarianceOfAverage (PerformanceMeasureType pm)

public DoubleMatrix2D getMin (PerformanceMeasureType pm)

public DoubleMatrix2D getMax (PerformanceMeasureType pm)

public DoubleMatrix2D[] getConfidenceInterval (PerformanceMeasureType pm,
                                               double level)
```

AbstractCallCenterStatProbes

This base class defines two maps that contain the statistical probes being managed. The first map, `tallyMap`, associates types of performance measures with matrices of tallies. The second map, `fmmTallyMap`, binds types of performance measures with matrices of function of multiple means tallies. The methods in this class assume that every type of performance measure do not appear in both maps.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class AbstractCallCenterStatProbes implements CallCenterStatProbes
```

Fields

`protected Map<PerformanceMeasureType, MatrixOfTallies<?>> tallyMap`

Map associating types of performance measures with matrices of tallies.

`protected Map<PerformanceMeasureType, MatrixOfFunctionOfMultipleMeansTallies<?>> fmmTallyMap`

Map associating types of performance measures with matrices of function of multiple means tallies.

SimCallCenterStat

Represents call center statistics obtained directly via call center measures. An instance of this class is created using an instance of `CallCenterMeasureManager`. Each time the `addObs()` method is called, the counters are read from the call center measures, and added to associated collectors.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class SimCallCenterStat extends AbstractCallCenterStatProbes
```

Constructor

```
public SimCallCenterStat (CallCenter cc, CallCenterMeasureManager ccm,
                          boolean keepObs, boolean normalizeToDefaultUnit,
                          PerformanceMeasureType... pms)
```

Constructs a new simulation-based call center statistics object. The constructor queries the given simulation logic for the supported measure types, and creates the necessary statistical probes.

Parameter

`ccm` the simulation logic.

Methods

```
public CallCenterMeasureManager getCallCenterMeasureManager
()
```

Returns the call center measures used for to collect observations.

```
public void addObs()
```

Adds new observations obtained via measure matrices. This uses `CallCenterMeasureManager.getValues (MeasureType, boolean)` to convert every available matrix of measures into matrices of double-precision values, and adds the resulting matrices to matrices of Tallies.

```
public void recomputeTimeAggregates()
```

Recomputes time-aggregate statistics in a setting where the number of observations in statistical collectors differs from periods to periods. This method processes each matrix of Tallies containing observations for several periods in the following way. Assuming that the processed matrix contains $P + 1$ columns, for each row r, this method gets the average for each period $p = 0, \ldots, P - 1$ and adds them up to get the time-aggregate average. Then, the Tally at position (r, P) is reset and the newly computed time-aggregate average is added.

ChainCallCenterStat

Combines the matrices of statistical probes from two call center statistical objects. Two implementations of `CallCenterStatProbes` are associated with each instance of this class. Each time a matrix of statistical probes is queried, this class queries the first inner call center statistic object. If the matrix is available, it returns it, otherwise, it queries the second inner object. This results in combining the statistics available in both objects.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class ChainCallCenterStat implements CallCenterStatProbes
```

Constructor

```
public ChainCallCenterStat (CallCenterStatProbes stat1,
                            CallCenterStatProbes stat2)
```

Constructs a new chained call center statistical object from inner objects `stat1` and `stat2`.

Parameters

`stat1` the first statistical object.

`stat2` the second statistical object.

StatType

Represents a type of statistic used by `StatCallCenterStat`.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public enum StatType
```

Constants

`AVERAGE`
: Average.

`VARIANCE`
: Variance.

`STANDARDDEVIATION`
: Standard deviation.

`VARIANCEOFAVERAGE`
: Variance divided by the number of observations in the inner tally.

`STANDARDDEVIATIONOFAVERAGE`
: Standard deviation divided by the square root of the number of observations in the inner tally.

StatCallCenterStat

Represents a set of statistical probes containing other statistics as observations. An object of this class is constructed from another implementation of `CallCenterStatProbes`, and a type of statistic to collect. For each matrix of probes defined in the inner implementation, a clone is made and stored into this object. When `addStat()` is called, each matrix of probes in the inner call center statistical object is retrieved, an intermediate matrix of observations is constructed, and the resulting matrix is added into the corresponding clone stored into this object.

This class is used for stratified sampling and randomized Quasi-Monte Carlo simulation as follows. Each time a macroreplication or stratum is simulated, statistics from a stratum or a randomization of a point set are available in an inner `CallCenterStatProbes` implementation. This class can obtain the averages, the variances, or the standard deviations from the probes, and add them to other matrices of statistical probes. This results in averages of averages, averages of variances, etc.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class StatCallCenterStat extends AbstractCallCenterStatProbes
```

Constructor

```
public StatCallCenterStat (CallCenterStatProbes stat, StatType statType,
                           boolean fmm)
```

Constructs a new group of call center statistical probes taking the observations from the inner set of probes `stat`, and collecting the statistic `statType`. This constructor creates a matrix of statistical probes for each performance measure defined in `stat`, by using the `clone` method. However, if `fmm` is `false`, no matrix of statistical probes is added for functions of multiple means tallies.

Parameters

`stat` the input call center statistical probes.

`statType` the type of statistic collected.

`fmm` determines if functions of multiple means are processed.

Method

```
public void addStat()
```

Adds new statistics to the probes defined by this object. Each matrix of probes in the inner call center statistical object is retrieved, an intermediate matrix of observations is constructed, and the resulting matrix is added into the corresponding clone stored into this object.

The way the matrix of observations is constructed depends on the type of input matrix of probes. For matrices of tallies, element (i, j) of the matrix of observations is given by computing a statistic (average, variance, etc.) on the observations of the tally (i, j) of the inner matrix of tallies. For matrices of functions of multiple means tallies, an array of tallies corresponds to each element (i, j), and a statistic is extracted from each element of this array of tallies. This results in a 3D array compatible with `MatrixOfFunctionOfMultiple-MeansTallies.add (double[][][])`.

CovFMMCallCenterStat

Represents a set of probes that collect covariances in matrices of functions of multiple means tallies. An instance of this class is constructed using a `CallCenterStatProbes` object. For each matrix of functions of multiple means tallies defined in the inner set of probes, this class can extract the covariance matrix of the functions' domain, and add these covariances into matrices of tallies. This results in averages of covariances which are useful for estimating the variance of stratified estimators.

More specifically, let $\mathbf{X} \in \mathbb{R}^d$ be a vector used to compute the function associated with position (r, c) in a matrix of performance measures m. Let $\mathbf{\Sigma}_{\mathrm{X}}$ be the matrix of covariances of $\mathbf{X}$. We suppose that $\mathbf{X}_0, \ldots, \mathbf{X}_{k-1}$ are i.i.d. and $\mathbf{X}_s$ is an average of n_s vectors. The average covariance is

$$\mathbf{\Sigma}_{\mathrm{X}} = \frac{1}{k} \sum_{s=0}^{k-1} \mathbf{\Sigma}_{\mathrm{X},s}$$

and the average weighted covariance is

$$\mathbf{\Sigma}_{\mathrm{X}} = \frac{1}{k} \sum_{s=0}^{k-1} \mathbf{\Sigma}_{\mathrm{X},s}/n_s.$$

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class CovFMMCallCenterStat
```

Constructor

`public CovFMMCallCenterStat (CallCenterStatProbes stat, boolean varWeighted)`

Constructs a new group of statistical probes for covariances from the inner call center statistics `stat`. If `varProp` is `true`, the returned covariances correspond to the proportional allocation in stratification. Otherwise, they depend on the number of observations in each stratum.

Parameters

`stat` the call center statistical object.

`varWeighted` the proportional allocation indicator.

Methods

`public void covariance (PerformanceMeasureType pm, int row, int col, DoubleMatrix2D cov)`

Returns the covariance matrix for the function of multiple means tally corresponding to the element `(row, col)` of the matrix of performance measures `pm`. If covariances are not queried for proportional allocation, the covariances are divided by the number of observations in the encapsulated tallies.

Parameters

`pm` the type of performance measure.

`row` the row in the matrix.

`col` the column in the matrix.

`cov` the 2D matrix filled with covariances.

`public void init()`

Initializes every matrix of tallies encapsulated in this object.

`public void addStat()`

Adds new observations in each associated matrix of tallies. This method is called at the end of each stratum or macroreplication and extracts the covariances from the matrices of functions of multiple means tallies. It then adds the covariances to the encapsulated matrices of tallies. If covariances are not queried for proportionnal allocation, the covariances are divided by the number of observations before they are added to the tallies, resulting in a weighted sum of covariances.

TimeNormalizeType

Possible type of time normalizations after a matrix of counters is obtained.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public enum TimeNormalizeType
```

Constants

`NEVER`

The values of the counters are never normalized with respect to time. This applies to measures not corresponding to counts, e.g., the maximal waiting time.

`ALWAYS`

Time normalization is always applied, because the counters are less sensible if not normalized. This applies, in particular, to the time-average number of agents.

`CONDITIONAL`

Time normalization is performed depending on a user-defined parameter, `SimParams.isNormalizeToDefaultUnit()`.

CallCenterStatWithSlidingWindows

Contains the necessary logic for computing statistics in time windows, for a call center model. Some routing or dialing policies might take decisions based on some statistics collected during the last few minutes of operation of the call center. This class provides the necessary tools for collecting such statistics. One first constructs an instance using a call center model, a number of periods, and a period duration. The instance is then registered with the model when statistics are needed, by using `registerListeners()`. The method `getStat()` can then be called at any time to obtain the statistics in the last time periods. Internally, this class uses counters with sliding windows to collect the observations.

```
package umontreal.iro.lecuyer.contactcenters.msk.stat;

public class CallCenterStatWithSlidingWindows implements Initializable,
            ToggleElement
```

Constructor

```
public CallCenterStatWithSlidingWindows (CallCenter cc, double
                                         periodDuration, int numPeriods,
                                         boolean contactTypeAgentGroup,
                                         PerformanceMeasureType... pms)
```

Constructs a new call center statistical collector with sliding windows, for the call center model `cc`, a window of `numPeriods` periods of `periodDuration` time units, and for performance measures of type `pms`. The boolean `contactTypeAgentGroup` determines if rows of type (call type, agent group) are needed or not for performance measures concerning call types.

Parameters

`cc` the call center model.

`periodDuration` the duration of the statistical periods.

`numPeriods` the number of statistical periods.

`contactTypeAgentGroup` determines if (call type, agent group) rows are needed.

`pms` the types of performance measures for which statistics are needed.

Methods

```
public void registerListeners()
```

Registers listeners with the call center model in order to collect observations.

```
public void unregisterListeners()
```

Unregisters the listeners with the call center model. This method may be used when the process using the statistics is stopped, in order to avoid unnecessary collecting of observations.

`public void init()`

Resets the internal statistical counters. This method should be called at the beginning of the simulation.

`public CallCenterStatProbes getStat()`

Initializes an object containing the statistics in the last periods. The matrices of statistical collectors in the returned object contain a single column corresponding to the statistics.

Package umontreal.iro.lecuyer.contactcenters.msk.cv

Contains classes used for simulation with control variables.

CVBetaFunction

Represents an object returning the β function used for control variables.

```
package umontreal.iro.lecuyer.contactcenters.msk.cv;

public interface CVBetaFunction
```

Method

```
public double getBeta (PerformanceMeasureType m, int row, int col, int cv,
                       int obs)
```

Returns the β function for a performance measure.

Parameters

`m` the type of performance measure.

`row` the row in the matrix of statistical probes.

`col` the column in the matrix of statistical probes.

`cv` the index of the control variable.

`obs` the observation this function is applied to.

Returns the value of β.

CVCallCenterStat

Represents call center statistics on which control variables are applied. An instance of this class is constructed from a `CallCenterStatProbes` object, and defines statistical probes for controlled estimators. When `applyControlVariables (PerformanceMeasureType, MatrixOfTallies, CVBetaFunction)` is called, observations are extracted from the probes in the inner `CallCenterStatProbes` object, and controlled observations are added to the encapsulated probes. This way, control variables are applied after the simulation is finished and do not require modifying the simulator.

```
package umontreal.iro.lecuyer.contactcenters.msk.cv;

public class CVCallCenterStat implements CallCenterStatProbes
```

Constructor

```
public CVCallCenterStat (SimLogic sim, CallCenterStatProbes inStat,
                         boolean fmm, ControlVariable... cvs)
```

Constructs a new CV call center statistical object using the simulation logic `sim`, taking statistics in the `stat` object, and applying the control variables `cvs`. If `fmm` is `false`, functions of multiple means tallies are ignored. Otherwise, control variables are applied on them when appropriate.

Parameters

`sim` the simulation logic.

`inStat` the call center statistics.

`fmm` if control variables are applied on functions of multiple averages.

`cvs` the array of control variables to apply.

Methods

```
public double[][][] getBetas (PerformanceMeasureType pm)
```

Returns the $\boldsymbol{\beta}$ arrays for performance measure of type `pm`. Element `[r][c][i]` of the returned array corresponds to the ith control variable applied to the performance measure of type `pm`, at position (`r`, `c`).

Parameter

`pm` the type of performance measure.

Returns the $\boldsymbol{\beta}$ vectors.

`public DoubleMatrix2D[][] getBetaMatrixFmm (PerformanceMeasureType pm)`

Returns a 2D array of matrices representing the β constants for the control variables applied to the components of functions of multiple means represented by the type of performance measure `pm`. Element `[r][c]` of the returned array contains the $\boldsymbol{\beta}$ matrix for performance measure at position (`r`, `c`).

Parameter

`pm` the type of performance measure.

Returns the $\boldsymbol{\beta}$ matrices.

`public void applyControlVariables()`

Equivalent to `applyControlVariables (null)`.

`public void applyControlVariables (CVBetaFunction cvBeta)`

Applies the control variables for the supported estimators. If `betaFunc` is non-`null`, it is used for obtaining the β constants. Otherwise, the constants are estimated from the statistics.

Parameter

`cvBeta` the beta function calculator, or `null`.

ControlVariable

Represents a type of control variable that can be applied on all performance measures supported by a call center simulator. An implementation of this interface obtains (or computes) observations of a centered control variable $\tilde{C}(m, r, c) = C(m, r, c) - E[C(m, r, c)]$ for performance measure type m, row r, and column c. Obtaining the centered CVs is usually done by querying some statistical collectors, but sometimes, sums may be computed. The exact control variable used might depend on the performance measure, e.g., the number of arrivals for calls of a given type, and the expectation might change from observations to observations. The only important point is to have $E[\tilde{C}(m, r, c)] = 0$ for each observation when the CV is applicable.

```
package umontreal.iro.lecuyer.contactcenters.msk.cv;

public interface ControlVariable
```

Methods

`public boolean appliesTo (PerformanceMeasureType pm)`

Determines if this control variable can be applied to the type `pm` of performance measure.

Parameter

`pm` the type of performance measure.

Returns `true` if the control variable can be applied, `false` otherwise.

`public boolean appliesTo (SimLogic sim, PerformanceMeasureType pm, int row, int col)`

Tests if the control variable can be applied to the performance measure of type `pm` at row `row` and column `column` when using the simulation logic `sim`.

Parameters

`sim` the simulation logic.

`pm` the type of performance measure.

`row` the row index.

`col` the column index.

Returns the result of the test.

`public int numberObs (SimLogic sim, CallCenterStatProbes inStat, PerformanceMeasureType pm, int row, int col)`

Returns the number of observations for the control variable used for the performance measure of type `pm`, at row `row` and column `col`. If no control variable of the type represented by this implementation is used with the specified performance measure, this returns 0.

Parameters

`sim` the simulation logic.

`inStat` the call center statistics.

`pm` the type of performance measure.

`row` the row index.

`col` the column index.

Returns the number of observations of the control variable.

```
public double getObs (SimLogic sim, CallCenterStatProbes inStat,
                      PerformanceMeasureType pm, int row, int col, int
                      index)
```

Returns the centered observation with index `index` of the control variable used for the type of performance measure `pm` at row `row` and column `col`.

Parameters

`sim` the simulation logic.

`inStat` the call center statistics.

`pm` the type of performance measure.

`row` the row index.

`col` the column index.

`index` the index of the observation.

Returns the observation.

```
public void init (SimLogic sim)
```

Initializes any data structure used by this control variable.

Parameter

`sim` the simulation logic.

NumArrivalsCV

Represents the control variable A, which is the number of arrivals of inbound contacts. When applied to a performance measure concerning inbound contact type k during period p, the used CV is the number of arrived contacts of type k during period p. For outbound contact types, the total number of arrived inbound contacts is used.

```
package umontreal.iro.lecuyer.contactcenters.msk.cv;

public class NumArrivalsCV implements ControlVariable
```

Package umontreal.iro.lecuyer.contactcenters.msk.conditions

Defines an interface for representing a condition on calls, and provides some implementations of the interface. A condition is an object that can be checked against any call of the model. When the condition is verified for a given call, we say that it applies for this call. Alternatively, some conditions may not depend on a call and rather checks the global state of the system. Conditions can be used to affect the behavior of routing or dialing policies.

The package provides the `Condition` interface to represent a condition. The other classes provide implementations of this interface for, e.g., conditions on the queue sizes, the number of free agents in group, etc. The class `ConditionUtil` is also provided to help with the creation of condition objects from `ConditionParams` instances created by parsing XML configuration files.

Condition

Represents a condition that can be checked on a given contact. Often, the test performed by such a condition is simple, e.g., the condition applies if the number of queued contacts of the type of the tested contact is greater than a threshold.

However, some conditions require complex state information, such as statistics observed during some time periods. In such cases, mechanisms need to be initialized at the beginning of simulation steps, and started during time intervals the condition is used. For this, the condition object might implement the `Initializable` and `ToggleElement` interfaces in addition to this interface. The simulator calls `init()` on each initializable condition, then `start()` for each condition implementing `ToggleElement`.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public interface Condition
```

Method

`public boolean applies (Contact contact)`

Checks the represented condition for the given contact `contact`, and returns `true` if and only if the condition applies. Some conditions depend on the state of the system rather than a particular contact. In such cases, the contact object can be ignored.

Parameter

`contact` the contact on which to check the condition.

Returns the success indicator of the test.

OrCondition

Represents a condition checking that at least one of a list of conditions applies.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class OrCondition implements Condition, Initializable, ToggleElement
```

Constructor

`public OrCondition (Condition... condList)`

Constructs a new or condition based on the list of conditions `condList`.

Parameter

`condList` the list on conditions used to perform the test.

Methods

`public Condition[] getConditions()`

Returns the associated list of conditions.

`public void start()`

Calls `start` for each associated condition implementing the `ToggleElement` interface.

`public void stop()`

Calls `stop` for each associated condition implementing the `ToggleElement` interface.

`public void init()`

Calls `stop()`, then calls `init` for each initializable condition associated with this object.

AndCondition

Represents a condition that checks if all conditions of a given list applies.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class AndCondition implements Condition, Initializable, ToggleElement
```

Constructor

`public AndCondition (Condition... condList)`

Constructs a new and condition using the list of conditions `condList`.

Parameter

`condList` the list of conditions to check.

Methods

`public Condition[] getConditions()`

Returns the associated list of conditions.

`public void start()`

Calls `start` for each associated condition implementing the `ToggleElement` interface.

`public void stop()`

Calls `stop` for each associated condition implementing the `ToggleElement` interface.

`public void init()`

Calls `stop()`, then calls `init` for each initializable condition associated with this object.

QueueSizesCondition

Represents a condition comparing the size of a waiting queue with the size of another queue. Let $Q_q(t)$ be the queue size of queue q at time t, and $\cdot$ be a relationship. The condition checks that $Q_{q_1}(t) \cdot Q_{q_2}(t)$ for fixed values of q_1, q_2, and $\cdot$. The relationship can be $<$, $>$, $=$, $\leq$, or $\geq$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class QueueSizesCondition extends TwoIndicesInfo
             implements Condition
```

Constructor

```
public QueueSizesCondition (CallCenter cc, int q1, int q2, Relationship
                            rel)
```

Constructs a new condition on the queue size for the call center model `cc`, first waiting queue `q1`, second waiting queue `q2`, and relationship `rel`.

Parameters

`cc` the call center model.

`q1` the index of the first waiting queue.

`q2` the index of the second waiting queue.

`rel` the relationship used to perform the comparison.

Method

```
public CallCenter getCallCenter()
```

Returns a reference to the call center associated with this condition.

QueueSizesWithTypesCondition

Represents a condition on queue sizes possibly for specific call types. This is similar to `QueueSizesCondition`, except that the compared queue sizes are determined using an index and a call type. If the given call type is non-negative, the compared size is the number of calls in the identified queue of the identified type. Otherwise, the total number of calls in the identified queue is used.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class QueueSizesWithTypesCondition extends QueueSizesCondition
```

Constructor

```
public QueueSizesWithTypesCondition (CallCenter cc, int q1, int q2, int k1,
                                     int k2, Relationship rel)
```

Constructs a new condition on the queue size for call center `cc`, using queue indices `q1` and `q2`, the call type indices `k1` and `k2`, and the relationshiop `rel`.

Parameters

`cc` the call center model.

`q1` the index of the first waiting queue.

`q2` the index of the second waiting queue.

`k1` the index of the first call type.

`k2` the index of the second call type.

`rel` the relationship used to perform the comparison.

Methods

```
public int getFirstType()
```

Returns the call type index for the first compared waiting queue.

```
public int getSecondType()
```

Returns the call type index for the second compared waiting queue.

NumFreeAgentsCondition

Represents a condition comparing the number of free agents in two groups of a model. Let $N_{\mathrm{F},i}(t)$ be the number of free agents in group i at time t, and $\cdot$ be a relationship. The condition checks that $N_{\mathrm{F},i_1}(t) \cdot N_{\mathrm{F},i_2}(t)$ for fixed values of i_1, i_2, and $\cdot$. The relationship can be $<$, $>$, $=$, $\leq$, or $\geq$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class NumFreeAgentsCondition extends TwoIndicesInfo
             implements Condition
```

Constructor

`public NumFreeAgentsCondition (CallCenter cc, int i1, int i2, Relationship rel)`

Constructs a new condition on agent groups for the call center model `cc`, agent groups with indices `i1` and `i2`, and comparing with relationship `rel`.

Parameters

`cc` the call center model.

`i1` the index of the first agent group.

`i2` the index of the second agent group.

`rel` the relationship used for the comparison.

Method

`public CallCenter getCallCenter()`

Returns a reference to the call center associated with this condition.

TwoIndicesInfo

Stores information about two indices and a relationship. This is similar to the JAXB-derived `TwoIndicesParams` class, except that the indices are stored into `int` fields instead of `Integer` fields. This class is used as a base for some condition objects, and to hold information about conditions on statistics.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class TwoIndicesInfo
```

Constructor

`public TwoIndicesInfo (int i1, int i2, Relationship rel)`
Constructs a new data object holding indices `i1` and `i2`, as well as relationship `rel`.

Methods

`public int getFirstIndex()`
Returns the value of i_1.

`public int getSecondIndex()`
Returns the value of i_2.

`public Relationship getRelationship()`
Returns the relationship to be tested.

IndexThreshInfo

Stores information about an index, a threshold, and a relationship. This is similar to the JAXB-derived `IndexThreshParams` class, except that the index and threshold are stored into fields of built-in types rather than wrappers. This class is used as a base for some condition objects, and to hold information about conditions on statistics.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class IndexThreshInfo
```

Constructor

`public IndexThreshInfo (int i, double threshold, Relationship rel)`
Constructs a new object holding the index `i`, the threshold `threshold`, and the relationship `rel`.

Methods

`public int getIndex()`
Returns the value of i.

`public double getThreshold()`
Returns the value of η.

`public Relationship getRelationship()`
Returns the relationship to be tested.

FracBusyAgentsCondition

Represents a condition comparing the fraction of busy agents for two groups. Let i_1 and i_2 be indices of agent groups and $\cdot$ be a relationship. This condition applies if and only if $\frac{N_{\mathrm{B},i_1}(t)}{N_{i_1}(t)+N_{\mathrm{G},i_1}(t)} \cdot \frac{N_{\mathrm{B},i_2}(t)}{N_{i_2}(t)+N_{\mathrm{G},i_2}(t)}$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class FracBusyAgentsCondition extends TwoIndicesInfo
             implements Condition
```

Constructor

```
public FracBusyAgentsCondition (CallCenter cc, int i1, int i2,
                                Relationship rel)
```

Constructs a new condition on the fraction of busy agents for call center `cc`, agent groups with indices `i1` and `i2`, and relationship `rel`.

Parameters

`cc` the call center model.

`i1` the index of the first agent group.

`i2` the index of the second agent group.

`rel` the relationship used for comparison.

Method

```
public CallCenter getCallCenter()
```

Returns a reference to the call center associated with this condition.

FracBusyAgentsWithTypesCondition

Represents a condition comparing the fraction of busy agents in two groups, possibly restricted to specific call types. This is similar to `FracBusyAgentsCondition`, except that the number of busy agents serving a contact of a given type can be used rather than the total number of busy agents. More specifically, the fraction of busy agents for group i_1 is determined using the number of busy agents serving calls of type k_1. If $k_1 < 0$, the total number of busy agents is used instead. A similar logic is used to get the fraction of busy agents in group i_2.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class FracBusyAgentsWithTypesCondition extends FracBusyAgentsCondition
```

Constructor

```
public FracBusyAgentsWithTypesCondition (CallCenter cc, int i1, int i2,
                                         int k1, int k2, Relationship rel)
```

Constructs a new condition on the fraction of busy agents for call center `cc`, agent groups `i1` and `i2`, call types `k1` and `k2`, and using relationship `rel` for comparison.

Parameters

`cc` the call center model.

`i1` the index of the first agent group.

`i2` the index of the second agent group.

`k1` the index of the first call type.

`k2` the index of the second call type.

`rel` the relationship used for comparison.

Methods

```
public int getFirstType()
```

Returns the call type index for the first compared agent group.

```
public int getSecondType()
```

Returns the call type index for the second compared agent group.

QueueSizeThreshCondition

Represents a condition comparing the size of a waiting queue with a fixed threshold. Let $Q_q(t)$ be the queue size of queue q at time t, and $\cdot$ be a relationship. The condition checks that $Q_q(t) \cdot \eta$ for fixed values of q, η, and $\cdot$. The relationship can be $<$, $>$, $=$, $\leq$, or $\geq$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class QueueSizeThreshCondition implements Condition
```

Constructor

```
public QueueSizeThreshCondition (CallCenter cc, int index, int threshold,
                                 Relationship rel)
```

Constructs a new condition on the queue size for the call center model `cc`, first waiting queue `index`, threshold `threshold`, and relationship `rel`.

Parameters

`cc` the call center model.

`index` the index of the waiting queue.

`threshold` the threshold.

`rel` the relationship used to perform the comparison.

Methods

```
public CallCenter getCallCenter()
```

Returns a reference to the call center associated with this condition.

```
public int getIndex()
```

Returns the value of q.

```
public int getThreshold()
```

Returns the value of η.

```
public Relationship getRelationship()
```

Returns the relationship to be tested.

QueueSizeThreshWithTypeCondition

Represents a condition comparing the number of calls of a given type in a given queue with a threshold. This is similar to `QueueSizeThreshCondition`, with the possibility to restrict the number of queued calls to a given type.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class QueueSizeThreshWithTypeCondition extends
             QueueSizeThreshCondition
```

Constructor

```
public QueueSizeThreshWithTypeCondition (CallCenter cc, int index, int
                                         type, int threshold, Relationship
                                         rel)
```

Constructs a new condition on the queue size for the call center `cc`, queue with index `index`, calls of type `type`, with threshold `threshold`, and using relation `rel` for comparison.

Parameters

`cc` the call center model.

`index` the index of the waiting queue.

`type` the call type index.

`threshold` the threshold.

`rel` the relationship used to perform the comparison.

Method

```
public int getType()
```

Returns the type identifier for which this condition is evaluated.

NumFreeAgentsThreshCondition

Represents a condition comparing the number of free agents in a groups of a model with a fixed threshold. Let $N_{\mathrm{F},i}(t)$ be the number of free agents in group i at time t, and $\cdot$ be a relationship. The condition checks that $N_{\mathrm{F},i}(t) \cdot \eta$ for fixed values of i, η, and $\cdot$. The relationship can be $<$, $>$, $=$, $\leq$, or $\geq$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class NumFreeAgentsThreshCondition implements Condition
```

Constructor

```
public NumFreeAgentsThreshCondition (CallCenter cc, int i, int threshold,
                                     Relationship rel)
```

Constructs a new condition on agent group for the call center model `cc`, agent group with index `i`, threshold `threshold`, and comparing with relationship `rel`.

Parameters

`cc` the call center model.

`i` the index of the agent group.

`threshold` the threshold.

`rel` the relationship used for the comparison.

Methods

```
public CallCenter getCallCenter()
```

Returns a reference to the call center associated with this condition.

```
public int getIndex()
```

Returns the value of i.

```
public int getThreshold()
```

Returns the value of η.

```
public Relationship getRelationship()
```

Returns the relationship to be tested.

QueueSizeThreshWithTypeCondition

Represents a condition comparing the number of calls of a given type in a given queue with a threshold. This is similar to `QueueSizeThreshCondition`, with the possibility to restrict the number of queued calls to a given type.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class QueueSizeThreshWithTypeCondition extends
             QueueSizeThreshCondition
```

Constructor

```
public QueueSizeThreshWithTypeCondition (CallCenter cc, int index, int
                                         type, int threshold, Relationship
                                         rel)
```

Constructs a new condition on the queue size for the call center `cc`, queue with index `index`, calls of type `type`, with threshold `threshold`, and using relation `rel` for comparison.

Parameters

`cc` the call center model.

`index` the index of the waiting queue.

`type` the call type index.

`threshold` the threshold.

`rel` the relationship used to perform the comparison.

Method

```
public int getType()
```

Returns the type identifier for which this condition is evaluated.

NumFreeAgentsThreshCondition

Represents a condition comparing the number of free agents in a groups of a model with a fixed threshold. Let $N_{\mathrm{F},i}(t)$ be the number of free agents in group i at time t, and $\cdot$ be a relationship. The condition checks that $N_{\mathrm{F},i}(t) \cdot \eta$ for fixed values of i, η, and $\cdot$. The relationship can be $<$, $>$, $=$, $\leq$, or $\geq$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class NumFreeAgentsThreshCondition implements Condition
```

Constructor

```
public NumFreeAgentsThreshCondition (CallCenter cc, int i, int threshold,
                                     Relationship rel)
```

Constructs a new condition on agent group for the call center model `cc`, agent group with index `i`, threshold `threshold`, and comparing with relationship `rel`.

Parameters

`cc` the call center model.

`i` the index of the agent group.

`threshold` the threshold.

`rel` the relationship used for the comparison.

Methods

```
public CallCenter getCallCenter()
```

Returns a reference to the call center associated with this condition.

```
public int getIndex()
```

Returns the value of i.

```
public int getThreshold()
```

Returns the value of η.

```
public Relationship getRelationship()
```

Returns the relationship to be tested.

FracBusyAgentsThreshCondition

Represents a condition comparing the fraction of busy agents in a group with a threshold. Let i be the index of an agent group, η be a threshold, and $\cdot$, a relationship. The condition applies if and only if $\frac{N_{\mathrm{B},i}(t)}{N_i(t)+N_{\mathrm{G},i}(t)} \cdot \eta$.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class FracBusyAgentsThreshCondition extends IndexThreshInfo
             implements Condition
```

Constructor

```
public FracBusyAgentsThreshCondition (CallCenter cc, int i, double
                                      threshold, Relationship rel)
```

Constructs a new condition on the fraction of busy agents for the call center model `cc`, the agent group index `i`, the threshold `threshold`, and for which comparisons are made using relationship `rel`.

Parameters

`cc` the call center model.

`i` the index of the agent group.

`threshold` the threshold on the fraction of busy agents.

`rel` the relationship used for comparison.

Method

```
public CallCenter getCallCenter()
```

Returns a reference to the call center associated with this condition.

ConditionUtil

Provides helper methods to construct condition objects using `ConditionParams` instances usually parsed from XML configuration files. The main method of this class is `createCondition (CallCenter, int, ConditionParams)` which uses other methods to make a `Condition` object out of the information provided by the parameters.

```
package umontreal.iro.lecuyer.contactcenters.msk.conditions;

public class ConditionUtil
```

Methods

```
public static Condition createCondition (CallCenter cc, int k,
                                         ConditionParams par)
```

Constructs a condition object for call center `cc`, and using parameters `par`. The type of the returned value depends on the parameters in `par`. If the `either` or `all` elements are set, the method calls `createOrCondition (CallCenter, int, ConditionParamsList)` or `createAndCondition (CallCenter, int, ConditionParamsList)`, respectively, and returns the result. If the `queueSizes` element is set, the method returns the result of `createQueueSizeCondition (CallCenter, TwoIndicesWithTypesParams)`. If the `queueSizeThresh` element is set, the method returns the result of `createQueueSizeThreshCondition (CallCenter, IndexThreshIntWithTypeParams)`. If `numFreeAgents` is set, the method returns the result of `createNumFreeAgentsCondition (CallCenter, TwoIndicesParams)`. If `numFreeAgentsThresh` is set, the method returns the result of `createNumFreeAgentsThreshCondition (CallCenter, IndexThreshIntParams)`. If `fracBusyAgents` is set, the method returns the result of `createFracBusyAgentsCondition (CallCenter, TwoIndicesWithTypesParams)`. If `fracBusyAgentsThresh` is set, the method returns the result of `createFracBusyAgentsThreshCondition (CallCenter, IndexThreshWithTypeParams)`. The result of `createCustomCondition (CallCenter, int, Named)` is returned if `custom` is set.

Parameters

`cc` the call center model.

`k` the index of the call type for which the condition concerns.

`par` the parameters from which conditions are created.

Returns the condition obtained from parameters.

Throws

`IllegalArgumentException` if a problem occurs during the creation of the condition.

`public static Condition createCondition (CallCenter cc, int k, JAXBElement <?> el)`

Similar to `createCondition (CallCenter, int, ConditionParams)`, but from a `JAXBElement` instance. The type of the condition created depends on the name of the element, obtained using `JAXBElement.getName()`. The value of the element, obtained using `JAXBElement.getValue()`, is cast to the appropriate class, and needed creation method is called.

Parameters

`cc` the call center model.

`k` the index of the call type for which the condition concerns.

`el` the JAXB element corresponding to the condition.

Returns the created condition object.

Throws

`IllegalArgumentException` if a problem occurs during the creation of the condition.

`public static OrCondition createOrCondition (CallCenter cc, int k, ConditionParamsList par)`

Creates an "or" condition from the call center model `cc`, and the parameters `par`. The parameter object encapsulates a list of `JAXBElement` representing parameters for a condition. The method uses `createCondition (CallCenter, int, JAXBElement)` to convert this element into a `Condition` object, and gathers the created objects into an array used to create the returned instance of `OrCondition`.

Parameters

`cc` the call center model.

`k` the index of the call type for which the condition concerns.

`par` the parameters for the condition.

Returns the condition object.

Throws

`IllegalArgumentException` if a problem occurs during the creation of one of the associated conditions.

`public static AndCondition createAndCondition (CallCenter cc, int k, ConditionParamsList par)`

Similar to `createOrCondition (CallCenter, int, ConditionParamsList)`, for an "and" condition.

`public static QueueSizesCondition createQueueSizeCondition (CallCenter cc, TwoIndicesWithTypesParams par)`

Creates a condition on the queue size from parameters `par`, and call center model `cc`. The method uses `par` to obtain the indices q_1 and q_2 as well as the relationship to compare with.

Parameters

`cc` the call center model.

`par` the condition parameters.

Returns the new condition object.

```
public static QueueSizeThreshCondition createQueueSizeThreshCondition
(CallCenter cc, IndexThreshIntWithTypeParams par)
```

Creates a condition on the queue size from parameters `par`, and call center model `cc`. The method uses `par` to obtain the index q, threshold η, and relationship to compare with.

Parameters

`cc` the call center model.

`par` the condition parameters.

Returns the new condition object.

```
public static NumFreeAgentsCondition createNumFreeAgentsCondition
(CallCenter cc, TwoIndicesParams par)
```

Similar to `createQueueSizeCondition (CallCenter, TwoIndicesWithTypesParams)`, for a condition on the number of free agents.

```
public static NumFreeAgentsThreshCondition
createNumFreeAgentsThreshCondition (CallCenter cc, IndexThreshIntParams
                                    par)
```

Similar to `createQueueSizeThreshCondition (CallCenter, IndexThreshIntWithTypeParams)`, for a condition on the number of free agents.

```
public static FracBusyAgentsCondition createFracBusyAgentsCondition
(CallCenter cc, TwoIndicesWithTypesParams par)
```

Creates a new condition on the fraction of busy agents using the call center model `cc`, and the parameters in `par`.

Parameters

`cc` the call center model.

`par` the condition parameters.

Returns the new condition object.

```
public static FracBusyAgentsThreshCondition
createFracBusyAgentsThreshCondition (CallCenter cc,
                                     IndexThreshWithTypeParams par)
```

Creates a new condition on the fraction of busy agents using the call center model `cc`, and the parameters in `par`.

Parameters

`cc` the call center model.

`par` the condition parameters.

Returns the new condition object.

```
public static Condition createCustomCondition (CallCenter cc, int k, Named
                                                par)
```

Calls `createCustom (Condition.class, cc, k, par)`.

Parameters

`cc` the call center model.

`k` the index of the call type concerning the condition.

`par` the parameters for the custom condition.

Returns an instance representing the custom condition.

```
public static <T> T createCustom (Class<T> base, CallCenter cc, int k,
                                  Named par)
```

Creates an object of base class `base`, from the parameter object `par`, and using the call center model `cc`. The method first uses the name associated with `par` as a class name for `Class.forName (String)`. It then checks that the corresponding class is a subclass of or implements `base`. If this is true, it searches for a constructor, and calls it to create an instance. The method looks for the following signatures, and the given order of priority: (`CallCenter`, `int`, `Map`), (`CallCenter`, `int`), (`CallCenter`, `Map`), (`int`, `Map`), (`CallCenter`), (`int`), (`Map`), and (). The last signature corresponds to the no-argument constructor. The instance of `CallCenter` is `cc` while the map is created by using `ParamReadHelper.unmarshalProperties (PropertiesParams)` on the properties associated with `par`.

Parameters

`base` the base class to be used.

`cc` the call center model.

`k` the index of the call type concerning the condition.

`par` the parameters for the custom condition.

Returns an instance representing the custom object.

```
public static boolean applies (double v1, double v2, Relationship rel)
```

Returns `true` if and only if a condition comparing `v1` and `v2` based on relationship `rel` applies.

Package `umontreal.iro.lecuyer.contactcenters.msk.spi`

Service provider interfaces for creating custom arrival processes, router's or dialer's policies. When one of these interfaces are implemented, the implementation is registered using the appropriate static method, or packaged as a Java extension with information for the service loading API. See `ServiceLoader` for more information on this.

DialerPolicyFactory

Provdes a method to create a dialer from the user-specified parameters.

```
package umontreal.iro.lecuyer.contactcenters.msk.spi;

public interface DialerPolicyFactory
```

Method

```
public DialerPolicy createDialerPolicy (CallCenter cc, DialerManager dm,
                                        DialerParams par) throws
                                        DialerCreationException
```

Constructs and returns a dialer policy for the call center model `cc` and the dialer parameters `par`. This method uses the result of `DialerParams.getDialerPolicy()` as a policy identifier given by the user. It returns a dialer policy if that particular dialer policy identifier is supported. Otherwise, it returns `null`. A dialer-creation exception is thrown only if the implementation supports the creation of the policy, but fails due to some error such as bad parameters.

Parameters

`cc` the call center model.

`par` the dialer's parameters.

Returns the new dialer's policy, or `null`.

RouterFactory

Provdes a method to create a router from the user-specified parameters.

```
package umontreal.iro.lecuyer.contactcenters.msk.spi;

public interface RouterFactory
```

Method

```
public Router createRouter (CallCenter cc, RouterManager rm, RouterParams
                            par) throws RouterCreationException
```

Constructs and returns a router for the call center model `cc` and the router parameters `par`. This method uses the `RouterParams.getRouterPolicy()` method to get the name of the router's policy given by the user, and creates a router object if it supports that particular policy name. Otherwise, it returns `null`. A router-creation exception is thrown only if the given routing policy is supported by the implementation, but some error occurs during the construction of the router, e.g., invalid parameters.

Parameters

`cc` the call center model.

`par` the router's parameters.

Returns the new router, or `null`.

ArrivalProcessFactory

Provdes a method to create an arrival process from the user-specified parameters.

```
package umontreal.iro.lecuyer.contactcenters.msk.spi;

public interface ArrivalProcessFactory
```

Methods

```
public ContactArrivalProcess createArrivalProcess (CallCenter cc,
                                                    ArrivalProcessManager
                                                    am,
                                                    ArrivalProcessParams
                                                    par) throws
                                                    ArrivalProcessCreationException
```

Constructs and returns an arrival process for the call center model `cc` and the arrival process parameters `par`. This method uses the `ArrivalProcessParams.getType()` to get the type string of the arrival process given by the user, and returns an arrival process if it supports that particular type identifier. Otherwise, it returns `null`. An arrival-process-creation exception is thrown only if the given arrival process is supported by the implementation, but some error occurs during the construction of the arrival process, e.g., invalid parameters.

Parameters

`cc` the call center model.

`par` the router's parameters.

Returns the new router, or `null`.

```
public boolean estimateParameters (ArrivalProcessParams par, int[][] data,
                                   double periodDuration)
```

Estimates the parameters of an arrival process using the data given in the 2D array `data`. The given array is a $N \times P$ matrix where N is the number of vectors of observations, and P is the number of main periods. If estimation is successful, the method updates the parameter object `par` with the estimated parameters, and returns `true`. Otherwise, it throws an illegal-argument exception. The method returns `false` if it does not recognize the type of arrival process described by `par`.

Parameters

`par` the parameters of the arrival process.

`data` the 2D array of vectors of observations.

`periodDuration` the duration of main periods, in simulation time units.

Returns the sucess indicator of the estimation.

Throws

`IllegalArgumentException` if an error occurs during parameter estimation.