Disruptions in SC-1 (aka Pharma 3)

Updated: 2022-08-09, for Pharma3 ver. 2.002

This document discusses the implementation of the disruptions in the SC-1 (aka Pharma3) application.

The `Disruptions` data structure

The `Disruptions` data structure (`edu.rutgers.pharma3.Disruptions`) implements the "schedule of disruptions", i.e. the list of disruption events that need to be modeled during the simulation. It stores a vector of `Disruption` elements, each of which represents a single disruptive event.

A `Disruption` is a structure with 4 elements:

- `time`: the time point when the disruption starts, on our standard simulation schale (days), same as `A1` in Abhisekh's spreadsheet. Typically, it's an integer value, i.e. refers to the beginning of a simulation day.
- `unit`: a string referring either to the production unit affected, or (for depletions of material from input buffers) the name of the product involved. In the former case, The name is identical to what would be returned by `Named.getName()` of the unit, e.g. `RawMaterialSupplier`. In the latter case, it is the named of the underlying CountableResource (not of the Batch resource that may be constructed on top of it), as per `Resource.getName()`, e.g. `Api` or `BulkDrug`.
- `type`: the type of the disruption, such as depletion, delay, adulteration etc.
- `magnitude`: a real number describing the "magnitude" of the disruption. Its semantics depends on the type of the disruption.

Types of disruptions

```
enum Type { Delay, ShipmentLoss, Adulteration, Depletion, Halt, };
```

- `Delay`: this is implemented, but not tested, because the current (Aug 2022) version of Abhisekh's 21-point scheme does not use it anymore. (At earlier version of the scheme, in July 2021, did). A disruption of this time would affect the transportation delay of batches shipped during this day, by making it longer by `magnitude` days.
- `Delay`: causes the loss of all shipments sent by this production unit during this days and several subsequent days. The total number of days so affected is given by `magnitude`.
- `Adulteration`: increases the probability that the products produced by a certain production unit will be faulty. this works slightly differently for Batch products (those with expiration dates and identified lots) and "fungible" ones (the packing material only). For the Batch products, this disruption applies to the batches produced during the day of the disruption; which of these batches are actually defective, will be only discovered during the QA (which may be some days later, due to the transportation delay). OTOH, because for fungible products we don't store any lot information, their disruption results in a higher fault rate in the products pasing the QA (testing) on the disruption days. The former approach is probably more realistic (the disruption, after all, affects manufacturing, rather than testing); the latter literally follows Abhisekh's write-up.
- `Adulteration`: destroys the specified amount (10^7 * `magnitude`) of the specified (by `unit`) resource in the input buffer of an FC unit that uses thats resource. For example, if `unit=RawMaterial`, we'll destroy some raw material in the input buffer of the ApiProduction unit.
  (Note: at present, using the name of the resource in the `unit` field is a sufficiently good identifier of what we want to destroy and where, because each resourse is used by only one unit of the FC subchain, and we don't do any destructions within the CMO subchain. If we have a more complex model, where it is important to distinguish "destruction of product X in the input buffer of element A" from "destruction of

product X in the input buffer of element B", then we may need to use a different notation, say, "A.B" in the `unit` field).

Creating a `Disruptions` object

One can create a `Disruptions` object in the Java API, by repeatedly use `Disruptions.add(...)` method.

One can also initialize a `Disruptions` object from a CSV file that describes a "disruption schedule". The file should have 4 columns, corresponding to the 4 fields of a `Disruption` object. Each line of the file corresponds to a single disruption event, i.e. a `Disruption` object. The file can contain an arbitrary number of lines, describing disruptions of arbitrary types affecting arbitrary supply chain elements to which they are applicable.

For sample dsisruption schedule files, see `config/dis.A2.csv`, `config/dis.A3.csv`, etc. For simplicity, each of these files contains disruptions of just one type, affecting just one unit. The naming of these files matches the naming of disruptions (A2 thru A21) in Abhisekh's CSV file.

How to run

One can specify the disruption file on the command line, along with the main config file: `./run3.sh -config config/pharma3.orig.csv -disrupt config/dis.A2.csv -until 100 > out.log`

How this all works

This is how the disruption simulation works in Pharma3 ver. 2.002. This probably will change somewhat in the future, for better integration with other Mason tools.

- The main application class `Demo extends SimState` has a field `Disruptions disruptions`, which, if it is not null, contains the disruptions schedule to apply in the current run. This field can be initialized when a `Demo` object is created. In the above example, it is initialized based on a disruptions file.
- The `Demo` object is available to all classes implementing the supply chain components (it is passed in as the argument to their `step()` method). Accordingly, each component that may be affected by one of our supported disruptions would check, in its `step()` method, whether a disruption of any of the applicable is scheduled for this component on the current simulation day. If it is, it will carry out appropriate actions, e.g. by setting flaggs that will modify the behavior of certain objects.
- The affected objects will modify their behavior as necessary, for a requested time period. For example, the ProdDelay class may produce "poor quality" lots for a specified number of days, i.e. lots marked with an `increaseInFaultRate` flag.