Data Collection in Repast Simphony

Antonio Bucchiarone

Fondazione Bruno Kessler, Trento – Italy
bucchiarone@fbk.eu

14 October 2020

Data Sources and Set

- Repast Simphony records data from *Data Sources.*
- ***Aggregate Data Sources****:* it receives a collection of objects (agents) and returns some aggregate value calculated over all the objects.
  - *Ex:* call a method on each object (agent) and return the maximum value.
- ***Non-Aggregate Data Sources****:* it takes a single object (agent) and returns a value.
  - Ex: call a method on an agent and return the result of that method call.
- **Data Set**: a template for producing *tabular data* where each *column* represents a data source and each *row* a value returned by that data source.

Writing Data

- Data will be *recorded* during the simulation run.
- Simphony can write data to both a *file* and the *console*.
- Files are created using the “File Sink” functionality.
  - Texts Sinks -> Add File Sink

Model Parameters

- Setting of the Initial number of *zombies* and *humans* (not fixed).

- A ***model parameter*** is parameter used by the model that a user can set via the GUI.

  - ***Name:*** a unique identifying name for the parameter.
  - **Display Name:** the label that will be used in the parameters panel for this model parameter.
  - **Type:** int, long, double, or string.
  - **Default Value:** the initial value of the parameter.
  - **Values [Optional]:** A space separated list of values of the chosen type. The parameter will be restricted to these values.

External Tools

- RStudio Statistical Computing Application
- Table of Agents and their properties
- Spreadsheet (Excel by default)
- JUNG (Internal Tools that provides some stats on networks)
- Export a Geography Layer to a Shapefile
- Weka Data Mining Analysis Application
- Pajek Network Analysis Application
- JoSQL (Runs SQL like queries on simulation components – contexts etc.)

Model Distribution

- Repast models can be distributed to model users via the *installation builder*.
- This features packs up your model and all of the software we need to run it into a single Java archive (“JAR”).
- The resulting installer can be executed on a any system with a Java version equal to or greater than the version used to compile the model.

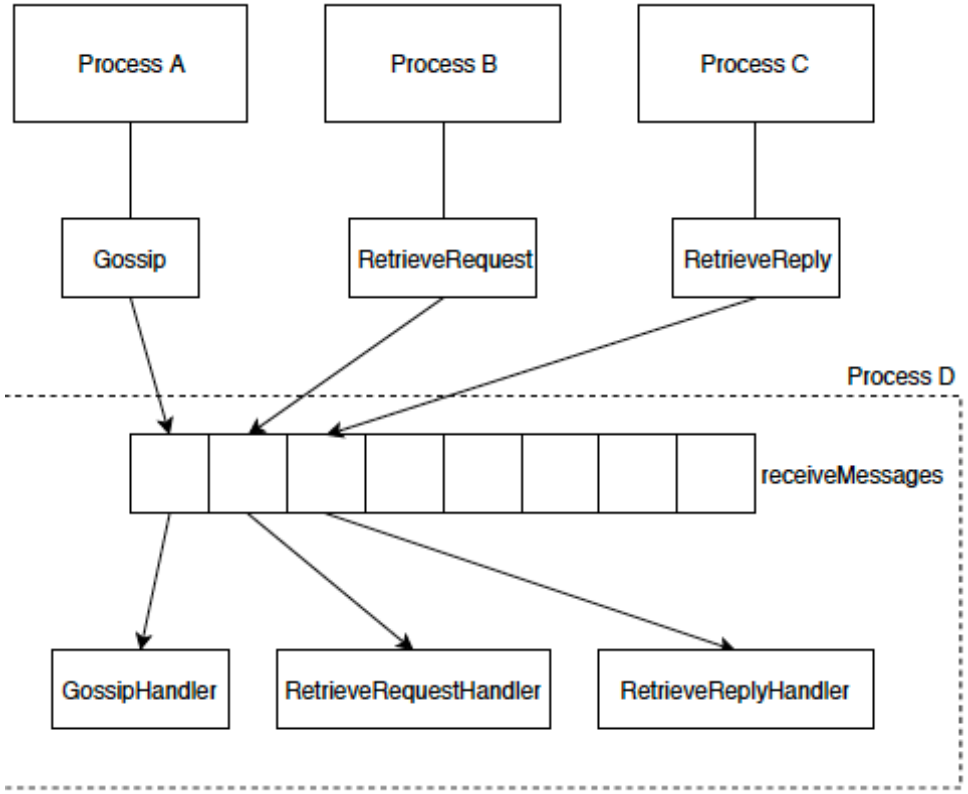
Lightweight Probabilistic Broadcast

FONDAZIONE BRUNO KESSLER

- Gossip-based algorithm.
- Reliability for scalability in distributed systems to parallelize very complex tasks.
- Let's see how to implement a simulator to evaluate the performance of the lightweight probabilistic broadcast algorithm.
  - Architecture of the simulator
  - Realtime Visualization
  - Parameter of the Simulation
  - Analysis

Architecture

- Process Agent
- LpbCastBuilder creates:
  - N instances of this agent
- No centralized memory.
- Interaction between processes is realized through message exchanges.
- Each process stores a queue of incoming messages.



Type of Messages

- **Gossip:** a message to periodically spread the information between processes.
- **RetrieveRequest:** a message to require the retransmission of a missing event.
- **RetrieveReply:** a message to send an event to a process that required it.

To simulate **network delays**, each message carries a field (*tick*) whose value denotes the *simulation tick* during which the message has to be processed.

At each **time step**, each process iterates through the queue containing the incoming messages and dispatches them to the handler responsible for their processing.

Buffers

- **retrieve:** the buffer used to collect missing events that might need to be recovered.

- **activeRetrieveRequest:** the buffer used to maintain the state of the requests which have already been issued in order to recover a missing event.

- When a process detects a missing event:
  - it adds it to the **retrieve** buffer.
  - If no gossip message carrying this event is received within a certain number of ticks, **the process starts the recovery phase**.
  - The missing event is then removed from the **retrieve** buffer and added to the **activeRetrieveRequest** buffer.
  - **……**

Performance Evaluation

- To analyze the performance of the protocol, the implementation leverages a particular AGENT, named **Collector.**

- It is used to gather information from the processes.
  - Number of times each event is delivered.
  - The tick during which each process becomes aware of a new subscriber.

- The visualization of the model exploits an agent, named **Visualization** to collect the actions performed during every tick by each process and visualize them on a display.

Real Time Visualization

- **How the simulation evolves** and how agents interact with each other.
- A **network of nodes** is displayed in order to visualize how processes relate in terms of message passing.
- The network is displayed by means of a **directed graph** in which **nodes** correspond to the **processes** and the **edges** represent **how processes are linked together** and how they exchange messages.

- Edge Types
  - **Light Gray Edges**: how processes are connected and how their view changes over time – to detect the presence of nodes isolated from the rest of the network.
  - **Colored edges**: the gossip of that particular event – new color represents the fact that another event is being considered.
  - **Thin red edges**: RetrieveRequest messages.
  - **Thin blue edges**: RetrieveReply messages.

Action Visualization

- To see action performed by a process
  - **Delivery**: when a process delivers the displayed event, it becomes of the same color used to represent the gossip message. After a specific amount of time has elapsed, the color of each node is reset to the default one and the visualization proceeds to show a new event.
  - **Submission**: the processes joining the network are briefly flashed in a bright red color.
  - **Unsubmission**: when a process performs a submission, its color changes to the light gray and after a short amount of time is removed from the view.

Parameter of the Simulator

- *Age-Based Message Purging Optimization [enabled]*: allows users to enable and disable the first optimization described in the paper which influences how the `events` buffer is trimmed.
- *Average Frequency Multiplier [0.9]*: this is a float value between 0 and 1, which is multiplied by the average frequency of a subscription as described in [1]. It influences the behaviour only if the frequency-based membership extension is enabled.
- *Buffer ArchivedEvents Max Size [25]*: the maximum length of the buffer `archivedEvents`, which stores messages in order to be able to satisfy retransmission requests.
- *Buffer EventIds Max Size [50]*: the maximum length of the buffer `eventIds`.
- *Buffer Events Max Size [5]*: the maximum length of the buffer `events`.