# Simulation System Overview

Elaine Wah
ewah@umich.edu

Updated: May 12, 2014

## 1 Introduction

The HFT simulation system employs agent-based modeling and discrete-event simulation to model algorithmic trading in financial markets.

## 2 Overview

The simulation system consists of:

**Markets:** Two types of markets are implemented in the system, a continuous double auction (CDA) market and a call market (which matches orders at regular, fixed intervals).

**Agents:** There are three general types (i.e. roles) of agents in the system, which are distinguished by whether or not they have fast access to more than one market and whether or not they possess private valuations for the security.

- **Background traders:** These agents have (potentially undelayed) access to only a single market (which is specified at agent creation as their primary market). See ZIAgent, ZIRAgent, ZIPAgent, and AAAgent.
- **Market makers (MM):** These agents submit a ladder of buy and sell orders upon each reentry. See BasicMM, MAMM, and WMAMM.
- **High-frequency traders (HFT):** These agents have access to multiple markets (usually every market), which models the direct feeds that many HFTs have to exchanges. See LAAgent.

## 3 Discrete-Event Simulation

In our system, we employ *discrete-event simulation*, a paradigm that allows the precise specification of event occurrences. It is particularly effective for modeling current U.S. securities regulation, specifically Regulation NMS—which led to the creation of the Security Information Processor (SIP) and which mandates the routing of orders for best execution. Components of the simulation system include:

| **Activity** | **Description** |
|---|---|
| AgentArrival | agent arrives in a market (or markets) |
| AgentStrategy | agent executes its trading strategy |
| Clear | market clears any matching orders |
| Liquidate | agent liquidates any net position |
| LiquidateAtFundamental | agent liquidates inventory at fundamental value |
| ProcessQuote | QP (or SIP) updates its best market quotes |
| ProcessTransactions | TP (or SIP) updates its list of transactions |
| SendToQP | market sends a new quote to QP (or SIP) |
| SendToTP | market sends list of new transactions to TP (or SIP) |
| SubmitOrder | agent submits a limit order to a market |
| SubmitNMSOrder | agent submits a limit order, routed for best execution |
| WithdrawOrder | agent withdraws a specific order from a market |

Table 1: List of activities in the simulation system.

**Entity:** Objects present in the simulation system, e.g., traders, markets, quote processors (QP), transaction processors (TP), and the SIP, that perform actions that effects on other entities.

**Activity:** Actions that entities can execute.

**Event:** A sequence of activities happening at the same time. Maintains the order in which they occur.

**Event Queue:** Queue ordered by activity time that executes activities in the "proper" order, sequentially until empty. Multiple activities may occur during the same time step, and in most circumstances they execute in pseudo-random order according to the random number generator of the simulation. The random nature is meant to simulate the slight timing differences that occur in real life (nothing actually occurs at the same time).

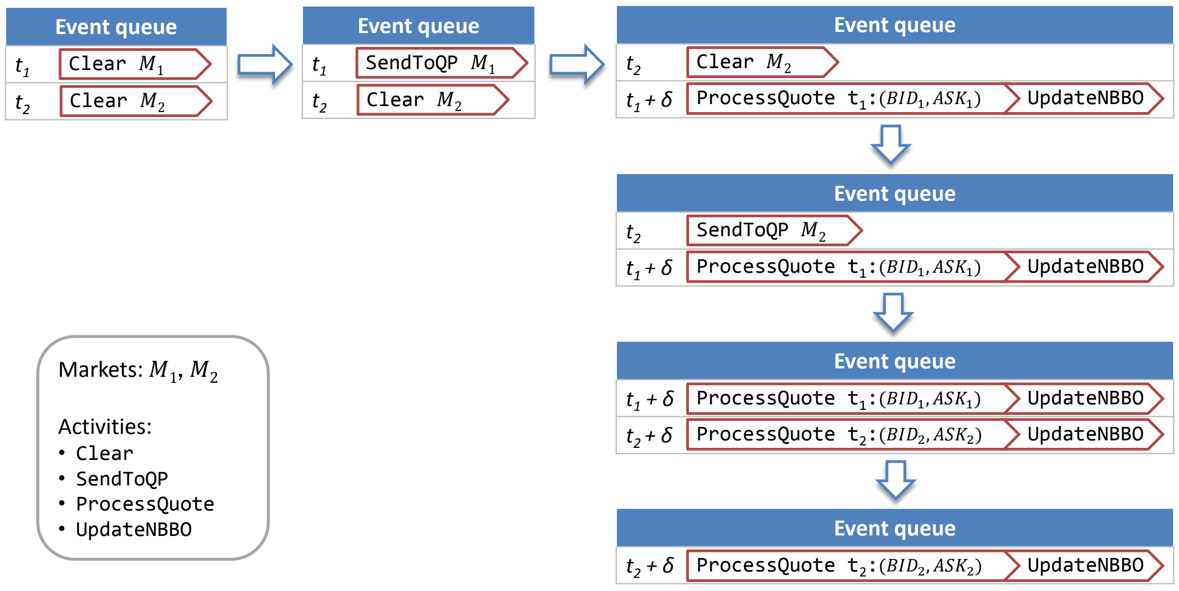
To summarize, an *event* consists of a sequence of *activities* that are to be executed by various *entities* (traders, markets, and the SIP).

## 3.1 Activities

Each activity has a timestamp and each is associated with at least one entity present in the simulation system. Note that activities may be chained (the next activity is inserted at the end of the current one). Activities may also be executely immediately, as each entity has a reference to the event scheduler (responsible for inserting activities). See Table 1 for a list of activities in the system.

## 3.2 Example

To control the latency of the SIP as well as general market access to quote and transaction information, we specify a set of activities: SendToQP, SendToTP, ProcessQuote, and ProcessTransactions. Two of these activities can be seen in Figure 1. The SendToQP activity is



Event queue
$t_1$ Clear $M_1$
$t_2$ Clear $M_2$
Event queue
$t_1$ SendToQP $M_1$
$t_2$ Clear $M_2$
Event queue
$t_2$ Clear $M_2$
$t_1+\delta$ ProcessQuote $t_1:(BID_1, ASK_1)$ UpdateNBBO
Event queue
$t_2$ SendToQP $M_2$
$t_1+\delta$ ProcessQuote $t_1:(BID_1, ASK_1)$ UpdateNBBO
Event queue
$t_1+\delta$ ProcessQuote $t_1:(BID_1, ASK_1)$ UpdateNBBO
$t_2+\delta$ ProcessQuote $t_2:(BID_2, ASK_2)$ UpdateNBBO
Event queue
$t_2+\delta$ ProcessQuote $t_2:(BID_2, ASK_2)$ UpdateNBBO
Markets: $M_1$, $M_2$
Activities:
• Clear
• SendToQP
• ProcessQuote
• UpdateNBBO

Figure 1: Event queue with an example sequence of activities to update the NBBO quote, in which the quote processor in question is the one in the SIP.

inserted when a market updates its quote at time $t$. Once the quote processor (QP) within the SIP gets the information it inserts a ProcessQuote activity to execute at time $t+\delta$ in the future to account for the delay caused by processing the information. When ProcessQuote is executed, the QP updates its stored information on the best market quotes. When agents query the SIP for market information, they will only get the most recent information that the SIP has processed after $t+\delta$, not all of the market quotes at the current time.

## 4 Simulation Specification File

See the simulation spec documentation for details on specifying the simulation and environment parameters.

## 5 Running a Simulation

1. To run a basic simulation, create a directory to store the simulations (e.g. `simulations`) and then create a directory for your specific environment configuration inside it:

```
pwd                   # should print your hft folder
mkdir simulations     # should be already created in the repo
mkdir simulations/test
```

2. Then copy the default `simulation_spec.json` file into the folder you just created:

```
cp docs/simulation_spec.json simulations/test/
```

3. Make any tweaks to the specification you want using your favorite text editor.
4. Use the following command to run your simulation:

   ```
   ./run-hft.sh simulations/test <number of simulations to run>
   ```

   For example:

   ```
   ./run-hft.sh simulations/test 100
   ```

5. The generated observations from 100 simulation runs should be saved within the `simulations/test` directory. All log files, if logging is enabled, will be saved in the `simulations/test/logs` directory.

Refer to the observation file documentation for details on interpreting the generated observation files and the logging documentation for reading the log files.

More advanced users can use `run-local-hft.sh` which permits specifying the jar with which you wish to run simulations.