14.2.2 Simulation & Engine

We start at the highest abstraction: Simulation & Engine. If we forget about the specific system types and multiple interferers we get the conceptualized view of a scenario. A victim system can be interfered by an interfering system.

On a very high level of abstraction an interference simulation consists of

1. victim.simulate, i.e. Configure the victim system;
2. interferer.simulate, i.e. Configure the interference system;
3. and collect the interference results based on the settings of the two systems.

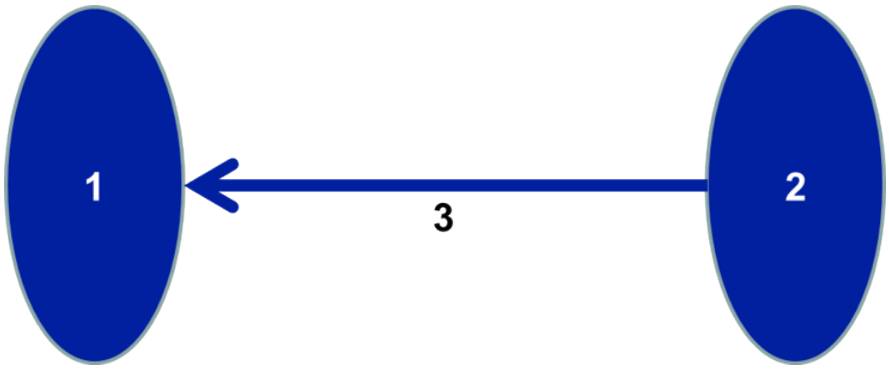

Figure 300: Abstraction level of an interference simulation

A closer look at the 3 steps: get the list of active receivers from the victim. Remember do not think of a specific system, if you for instance think of a Generic system you might say that this is never the case because there can be only one receiver. But there being only one is a special case of being many. And for Cellular systems there can definitely be many receivers (UE or BS). So the highlevel view of the algorithm is that there is a list of active receivers in the victim system.

Now we do the same (but opposite) for the interfering system: get the list of active transmitters. Again the list of active transmitters is the most general or highlevel notion of the algorithm which describes all possible cases (0, 1, or many).

Finally we combine all victim receivers with interfering transmitters and base the calculation of the interference with this as shown in Figure 301.

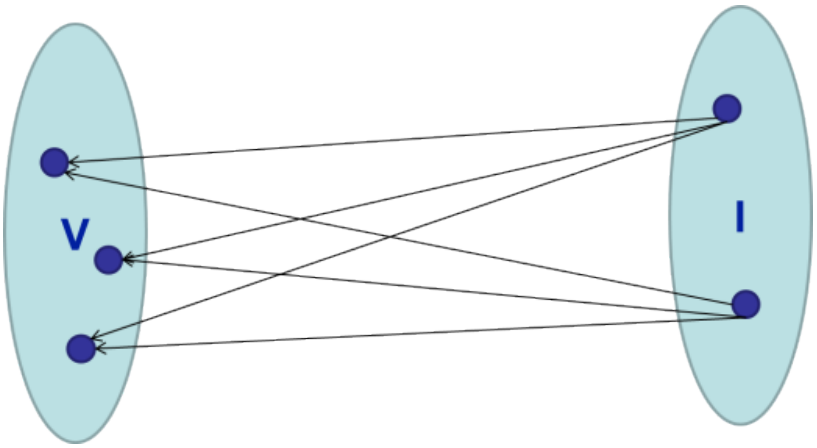

Figure 301: Abstraction level of IT->VR links

These IT->VR links are really what we are looking for and bases the interference calculations upon.

Given this the following results are possible:

- 0 – For some reason there was either 0 active receivers or 0 active transmitters in the victim, interfering system (respectively). Therefore we have no IT->VR links;
- 1 – There is precisely one active receiver and one active transmitter. This is most commonly the case for ”generic-generic” simulations;
- Many – There are many IT->VR links either because the systems are generic with many active transmitters or cellular (or maybe it is a future system not invented yet).

We have certainly described a lot about how SEAMCAT works but at no point it was needed to talk about any systems specifics. Of course during a simulation the choise of system will very much affect how it is done, but conceptualized like this there is no need.

On the highest level of abstraction we have the Simulation & Engine. “Simulation” holds the scenario setup and various hook for starting and completing a snapshot while “Engine” runs a simulation by controlling the steps in the simulation

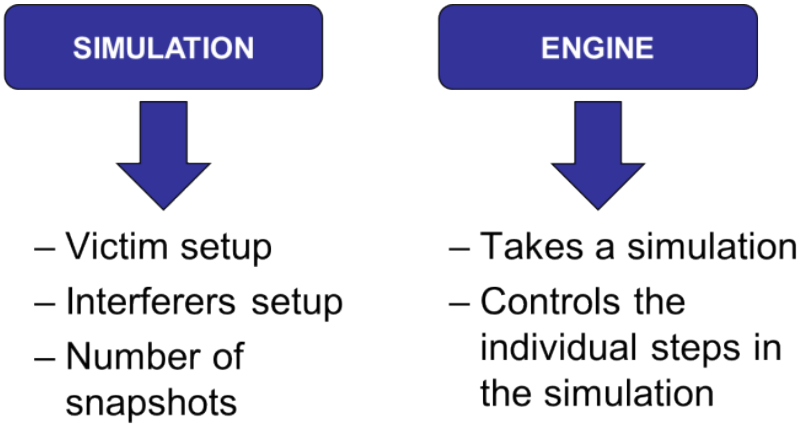

Figure 302: Simulation and Engine abstract representation

The figure below from the engine shows almost exactly the steps described above.

The real implementation is in InterferenceSimulationEngine.java take a look at it. Remind yourself that programming is all about details: there are so many other things going on than the actual loop.

```
public MutableEventResult single( VictimSystemSimulation victimSimulation, int eventNumber, Simulation simulation ) {
    fixSeed( simulation, eventNumber);

    MutableEventResult eventResult = new MutableEventResult(eventNumber, (WorkspaceScenario) simulation.getScenario());
    victimSimulation.simulate(eventResult);

    List<InterferenceLinkSimulation> iSims = new ArrayList<>();
    for (InterferenceLink interferenceLink : simulation.getScenario().getInterferenceLinks()) {
        InterferenceLinkSimulation iSim = simulation.getInterferenceLinkSimulation(interferenceLink);
        iSims.add(iSim);
        Point2D victimPos = victimSimulation.getSystemPosition( eventResult, interferenceLink);
        iSim.simulate(simulation.getScenario(), eventResult, interferenceLink, victimPos);
    }

    victimSimulation.collect(eventResult);

    simulation.eventComplete(eventResult, victimSimulation, iSims);

    return eventResult;
}
```

Figure 303: Engine code

The engine asks the simulation for the victim system simulation. So the simulation controls which victim simulation instance is handed to the engine, and the engine then uses that instance for doing its calculation. Therefore the engine is decoupled from the knowledge of which specific victim simulation is running (e.g. Cellular victim).

Revision #1
Created 2026-07-14 06:35:06 UTC by ECO TECH
Updated 2026-07-14 06:36:53 UTC by ECO TECH