openPASS - Scheduler
Architectural perspective

„Old“ scheduler (before release 0.6)

• Priority lists are built for every timestep
• List management introduces a lot of runtime overhead
• Easy to adapt

„New“ scheduler

• Scheduled elements (tasks) are considered (semi-) static
• Based on their scope, tasks can be grouped into lists which are initialized once at start up
• Tasks and their containing lists are designed to allow expansion and automatic sorting based on task attributes during runtime
• This leads to support of static use cases (PCM) as well as dynamic ones (runtime spawning of stochastic agents)
• Small overhead introduced due to dynamically added tasks. Compared to the simulation time, list manipulation is only sporadic (i.e. a new agent is added) and fast.
• Handling of agent based and common tasks throughout the simulation
• Execution of a run is divided into 6 phases:
  - Bootstrap
  - Common
  - NonRecurring
  - Recurring
  - Finalize Recurring
  - Finalize

  ● One-time execution at start of simulation
  ● Processed at each timestep
  ● One-time execution after end condition is reached (i.e. simulation duration)
<table>
<thead>
<tr>
<th>Task type</th>
<th>Scope</th>
<th>Priority</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spawning</td>
<td>Triggers agent spawning during spawn timr, Parse agent tasks</td>
<td>4</td>
<td>(agents have to be instantiated first)</td>
</tr>
<tr>
<td>EventDetector</td>
<td>Execution of event detectors</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Manipulator</td>
<td>Execution of manipulators</td>
<td>2</td>
<td>(dependent on event detectors)</td>
</tr>
<tr>
<td>SyncGlobalData</td>
<td>Synchronization of global data</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Observation</td>
<td>Update of observation modules</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Trigger</td>
<td>Execution of components (trigger functions)</td>
<td></td>
<td>Dependent on component-priorities. Independent of non-component priorities.</td>
</tr>
<tr>
<td>Update</td>
<td>Execution of components (Update input/output)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase</td>
<td>Category</td>
<td>Task type</td>
<td>Comment</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>static</td>
<td>Observation</td>
<td>--</td>
</tr>
<tr>
<td>Common</td>
<td>static</td>
<td>Spawning, EventDetector, Manipulator</td>
<td>Initialization phase for whole simulator</td>
</tr>
<tr>
<td>NonRecurring</td>
<td>dynamic</td>
<td>Trigger, Update</td>
<td>Used to init components. Task is deleted after execution.</td>
</tr>
<tr>
<td>Recurring</td>
<td>dynamic</td>
<td>Trigger, Update</td>
<td>Execution of agent components</td>
</tr>
<tr>
<td>FinalizeRecurring</td>
<td>static</td>
<td>SyncGlobalData</td>
<td>Synchronization of world</td>
</tr>
<tr>
<td>Finalize</td>
<td>static</td>
<td>EventDetector, Manipulator, Observation</td>
<td>Execution after reach of end condition</td>
</tr>
</tbody>
</table>
**Tasks.h**

Specifies „how“ tasks look like.

Defines:
- TaskTypes and their static priorities
- constructors for different TaskItems based on TaskTypes specialising a generic TaskItem definition
- class „Tasks“ as TaskItem container

**TaskBuilder.h**

Helper class to:
- „create“ Bootstrap-, Common-, FinalizeRecurring- and Finalize-Tasks
- „create“ Manipulator- and EventDetector tasks.
- bind functions of above tasks to underlying TaskItems

**SchedulerTasks.h**

„Management“ class for:
- calculation of scheduler timestamps
- access of tasks for given timestamp (filtering)
- deletion of recurring/non-recurring tasks linked to an specific agent

**Scheduler.h** provides logic to function as “controller“ for :
- check of end condition “simulation duration"
- systematic execution of tasks (phases) by requesting tasks for a given timestamp from SchedulerTasks
- supervision of agent based tasks, spawning of agents, triggering of agent removal
- managing abortion of a task
Scheduler module hierarchy

"static" lists

"dynamic" lists
For each simulation run the following steps are made:

1. Instantiate TaskBuilder and create Bootstrap, Common, FinalizeRecurring and Finalize tasks.
2. Instantiate and fill SchedulerTasks with built task lists.
3. Initial spawning
4. Execute bootstrap tasks
5. Execute common tasks
6. Update SchedulerTasks (spawning/removing, change component tasks)
7. Execute component tasks (non-recurring, recurring, finalize recurring)
8. Make timestep
9. Repeat steps 5 till 8 until end condition is reached
10. Execute finalize tasks
1) Instantiate SpawnControl and TaskBuilder

2) Call TaskBuilder to create Bootstrap-, Common- FinalizeRecurring- and FinalizeTasks

3) Instantiate SchedulerTasks with above task lists and fixed update rate of 100ms

SpawnControl is used to generate log messages if a task aborts (e.g. Incomplete scenario, Agent generation error...). This is handled by checking the return value of each executed task. If a task returns false ParseAbortReason(spawnControl, currentTime) is called and an error message is created.

Each call generates and returns a List<TaskItems>. Each TaskItem has a fixed framework update rate of 100ms.

As per definition TaskItems need to bind executing functions interfaced as std::function<bool>().

If needed additional parameters can be linked via std::ref(param) (e.g. BootstrapTasks link ObservationNetworkInterface::UpdateTimeStep to runResults).

Tasks are converted to a multiset and implicitly sorted via overloaded operator< based on their priority and TaskType.

Timestamps for execution are calculated.
4) Execute Bootstrap tasks

5) Do until (time <= simulation end time):
   5.1) execute common tasks
   5.2) update agents

Generic function ExecuteTasks is called for all Bootstrap tasks.
This invoks the UpdateTimeStep-method of the ObservationNetwork. Logging path is set via referenced runResults.

SpawnControl::PullNewAgents is invoked. For each new Agent its modules are parsed and new TaskItems are created (implicit sorting based on the module priority due to multiset and overloaded operator<).
Recurring- / and non-recurring agent tasks are added to the scheduler.
Invalid agents are queued for removal via Worldinterface::QueueAgentRemove
5.3) execute non-recurring tasks

5.4) execute recurring tasks

6) update current timestamp

7) clear active events

8) execute FinalizeTasks (t > simulation duration)

Updated Non-recurring (agent) tasks are consumed and deleted after execution.

Updated recurring (agent) tasks are executed.

SchedulerTasks::GetNextTimestamp is called and all timestamps (considering delays and different cycle times of (updated) agent modules) are calculated. The next scheduled timestamp > current timestamp is returned.

EventNetwork::ClearActiveEvents is called.

All FinalizeTasks are executed once. End of current run.