openPASS - Scheduler

intech

„Old“ scheduler (before release 0.6)

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

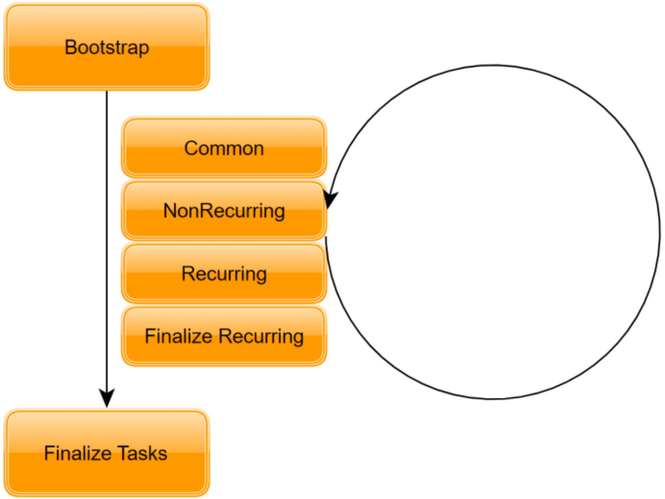
„New“ scheduler

- 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.

Architectural perspective

Basic principle

- **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 ●



Bootstrap

Common

NonRecurring

Recurring

Finalize Recurring

Finalize Tasks

● One-time execution at start of simulation

● Processed at each timestep

● One-time execution after end condition is reached (i.e. simulation duration)

Task type	Scope	Priority
Spawning	Triggers agent spawning during spawntimr, Parse agent tasks	4 (agents have to be instantiated first)
EventDetector	Execution of event detectors	3
Manipulator	Execution of manipulators	2 (dependent on event detectors)
SyncGlobalData	Synchronization of global data	1
Observation	Update of observation modules	0
Trigger	Execution of components (trigger functions)	Dependent on component-priorities. Independent of non-component priorities.
Update	Execution of components (Update input/output)	

Task types and phases

Phase	Category	Task type	Comment
Bootstrap	static	Observation	--
Common	static	Spawning, EventDetector, Manipulator	Initialization phase for whole simulator
NonRecurring	dynamic	Trigger, Update	Used to init components. Task is deleted after execution.
Recurring	dynamic	Trigger, Update	Execution of agent components
FinalizeRecurring	static	SyncGlobalData	Synchronization of world
Finalize	static	EventDetector, Manipulator, Observation	Execution after reach of end condition

Task types and phases

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

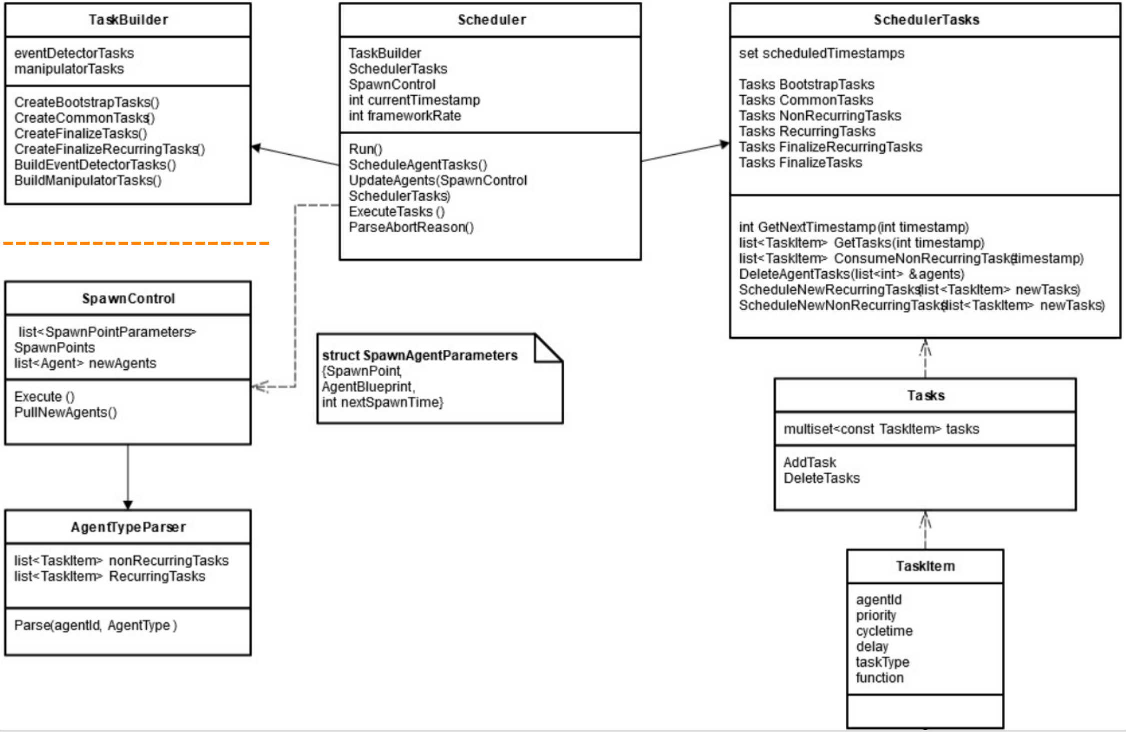
Scheduler.h provides logic to function as “controller” for :

- 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 modules

„static“ lists

„dynamic“ lists



TaskBuilder

eventDetectorTasks
manipulatorTasks

CreateBootstrapTasks()
CreateCommonTasks()
CreateFinalizeTasks()
CreateFinalizeRecurringTasks()
BuildEventDetectorTasks()
BuildManipulatorTasks()

Scheduler

TaskBuilder
SchedulerTasks
SpawnControl
int currentTimestamp
int frameworkRate

Run()
ScheduleAgentTasks()
UpdateAgents(SpawnControl
SchedulerTasks)
ExecuteTasks ()
ParseAbortReason()

SchedulerTasks

set scheduledTimestamps

Tasks BootstrapTasks
Tasks CommonTasks
Tasks NonRecurringTasks
Tasks RecurringTasks
Tasks FinalizeRecurringTasks
Tasks FinalizeTasks

int GetNextTimestamp(int timestamp)
list<TaskItem> GetTasks(int timestamp)
list<TaskItem> ConsumeNonRecurringTasks(timestamp)
DeleteAgentTasks(list<int> &agents)
ScheduleNewRecurringTasks(list<TaskItem> newTasks)
ScheduleNewNonRecurringTasks(list<TaskItem> newTasks)

SpawnControl

list<SpawnPointParameters>
SpawnPoints
list<Agent> newAgents

Execute ()
PullNewAgents()

struct SpawnAgentParameters
{SpawnPoint,
AgentBlueprint,
int nextSpawnTime}

Tasks

multiset<const TaskItem> tasks

AddTask
DeleteTasks

AgentTypeParser

list<TaskItem> nonRecurringTasks
list<TaskItem> RecurringTasks

Parse(agentId, AgentType)

TaskItem

agentId
priority
cycletime
delay
taskType
function

Scheduler module hierarchy

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

Scheduling

1) **Instantiate SpawnControl and TaskBuilder**

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 a error message is created.

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

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 funtions interfaced as std::function<bool()>.

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

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

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

Timestamps for execution are calculated.

Scheduler::Run

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

Scheduler::Run

Scheduler::Run

5.3) execute non-recurring tasks

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

5.4) execute recurring tasks

Updated recurring (agent) tasks are executed.

6) update current timestamp

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.

7) clear active events

EventNetwork::ClearActiveEvents is called.

8) execute FinalizeTasks (t > simulation duration)

All FinalizeTasks are executed once. End of current run.