

Simphonie definition file



AUTHOR	Sebastien Devaux
ABSTRACT	
KEYWORDS	simulation, simulation modelling platform, ECSS-E-ST-40-07C, kernel
CONTEXT	Component simphonie-1.0.0
PROCESSED	2025-03-23 16:59:09+01:00 / sdevaux@theia

ISSUE / REVISION STATUS RECORD		
Edition	Date	Changes summary
1	2024-12-27	Document creation

Summary

1	Introduction	4
1.1	SMP Level 1 features coverage	4
1.2	SMP Level 2 features coverage	4
1.3	Simphonie specific behaviors	4
1.4	SMP complements and extension	4
2	Design	5
2.1	Mandatory services bundle	5
2.2	Path parsing	6
2.3	Scheduler / Time Keeper coordination	7
2.4	Simulation start / stop control	8
2.5	Data propagation implicit schedule	9
2.6	aperiodic our out of clock scheduling	10
2.6.1	Simulation time change triggered by external event	10
2.6.2	Entry points scheduled on data event	10

List of Figures

1	Simulator services override	6
2	Path parsing state diagram	7
3	Scheduler / Time keeper coordination	8
4	Scheduler / Time keeper coordination	9

1 Introduction

1.1 SMP Level 1 features coverage

In this section, the requirements are defined only to link to SMP level clauses subset in order to organize test traceability and provide an overview of the SMP features coverage.

simph.smp.obj

Simphonie shall provide a concrete class for the Smp::IObject interface.

simph.smp.comp

Simphonie shall provide a concrete class for the Smp::IComponent interface.

1.2 SMP Level 2 features coverage

1.3 Simphonie specific behaviors

simph.sched.1

The scheduler shall emit an event named **Scheduler_PreEventExecute** before executing the next scheduled entry point.

simph.sched.2

The scheduler shall emit an event named **Scheduler_PostEventExecute** just after the execution of each scheduled entry point is completed.

simph.sched.3

The scheduler shall request simulator to hold simulation when there are no more events in the schedule queue.

1.4 SMP complements and extension

Here are defined some extended SMP features, that are features not defined by the SMP standard but considered to be needed to create a full featured simulator.

simph.esmp.1

The extended SMP feature libraries shall not link with the simphonie's kernel library.

The basic object implementation (object, component, collection, composites) can be shared, but the extended service shall rely only on the standard SMP interface when interacting with the simulator, models, components and other services to be hopefully reusable with any SMP compliant simulation infrastructure.

simph.esmp.svc.1

The simphonie's Smp::ISimulator implementation shall let the simulator integrator override the simphonie's implementation of the SMP mandatory services with any of its own implementation.

Of course, such service replacement may alter the way simphonie is working (and that is for that purpose), and the simulator integrator shall be aware of some constraints to keep its simulation system able to run properly:

- some simphonie's services are sharing somehow a contract for a good cooperation. For instance the scheduler and time keeper are cooperating through specific events that are not part of the SMP level 1 standard. Then replacing one of this two service requires the contract is maintained by the replacing service. Or both services that may have their own cooperation scheme shall be overridden.
- The service override is allowed only on the very early stage of the simulator build phase: services shall be added using AddService() method before the Publish step starts. After that the simulator behavior is undefined and the integrator shall assume the resulting behavior can suit its expectations.
- The type registry service is used to load libraries, making possible some types being registered before any service is added, then before a type registry replacement can be set making possible the loss of the early type registrations.

In the next requirements, the service word shall be understood in the extended way of set of feature. When deriving from `Smp::IService` is needed, specific requirements will be defined. For a given service, without such explicit requirement, the implementer can choose any of the following design:

- a single component derived from `Smp::IService`.
- a composite component and sub component with root composite derived from `Smp::IService`.
- a single component derived from `Smp::IModel`.
- a composite component and sub component with root composite derived from `Smp::IModel`.
- any other scheme of object composition as soon it can be inserted when into the simulator using the regular `Smp::ISimulator` interface.

The `Smp::IModel` or `Smp::IService` derivation shall be preferred for any component to be dynamically loaded through the library loading, factory and component instances creation features of the simulator (see `Smp::ISimulator`), in order to let the loaded feature being properly published and retrievable through the resolver service.

simph.esmp.hold.1

The extended SMP library shall provide a service letting control automatic transition of the simulator to stand by state.

simph.esmp.hold.2

The simulator hold control service shall publish a field named `endSimTime` defining when the simulation shall stop (as a simulation time reference).

simph.esmp.hold.3

The simulator hold control service shall request simulator hold when the simulation time reached the current value of its `endSimTime` field.

simph.esmp.datarec.1

The extended SMP library shall provide a data recording service able to record in file any data field of the simulator during the simulation.

simph.esmp.datarec.2

The data recording service shall publish a field named `file` to store the path of the file where data shall be recorded.

simph.esmp.fsched.1

Simphonie shall let the simulator integrator define entry point scheduling based on field related events.

This feature shall let define aperiodic entry points activation. The following event kinds and scheduling policies shall be considered:

- a value is pushed to some of the entry point owner's input field.
- same than previous, but not only a value is pushed, but the new pushed value is not equal to the current value.
- the entry point may be scheduled now (current simulation time) or with a delay. Activation count and periodicity may be considered as well.
- when the entry point activation is requested by many events at the same simulation time, the entry point shall be scheduled only once. Said another way, the entry point should not be scheduled several times for the same simulation time.

The above policy may be refined as more formal requirement in future editions. The main motivation for such feature is to trigger computation only when relevant on external asynchronous and aperiodic stimulation. The SMP discrete event simulation kernel may then handle computation pipelines according to any on-demand strategy similar to the very common observer/observable patterns or publish/subscribe patterns.

2 Design

2.1 Mandatory services bundle

The kernel library include an implementation for each of the mandatory SMP services:

- Logger
- Scheduler
- Resolver
- TimeKeeper

The services are created on the simulator construction to ensure availability at the earliest stages of the simulator build. However those service shall be replaceable

simph.esmp.svc.1

, meaning a user may substitute anyone with its own implementation using the **AddService** method. To do so, this method dynamically checks the type of the service to add for each mandatory service type. When it matches a type, the simphonie's default service implementation is deleted and replaced by the new one. This makes the eventual configuration applied to the deleted service be lost. Each mandatory service type is checked in sequence, making a single object implementing many of the mandatory services be properly registered as the right instance for the many services it implements.

This way of doing (creating default services and replace later) has been selected because of the few early simulator building action that may need the availability of such service before any new one can be added. In most case this is without significant effect, since almost nothing is done before the first service addition and services should be added and more specifically published before any other kinds of components can be published.

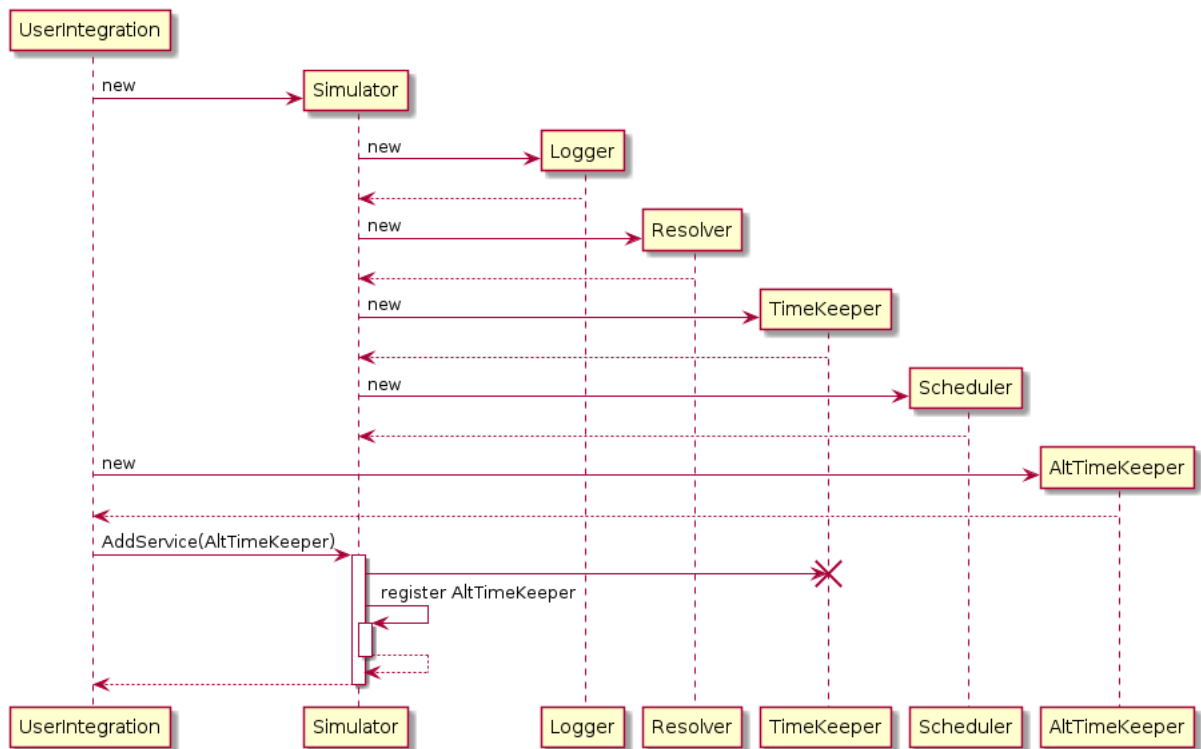


Figure 1: Simulator services override

2.2 Path parsing

The bundled resolver parses the pathes to be resolved according the following state diagram.

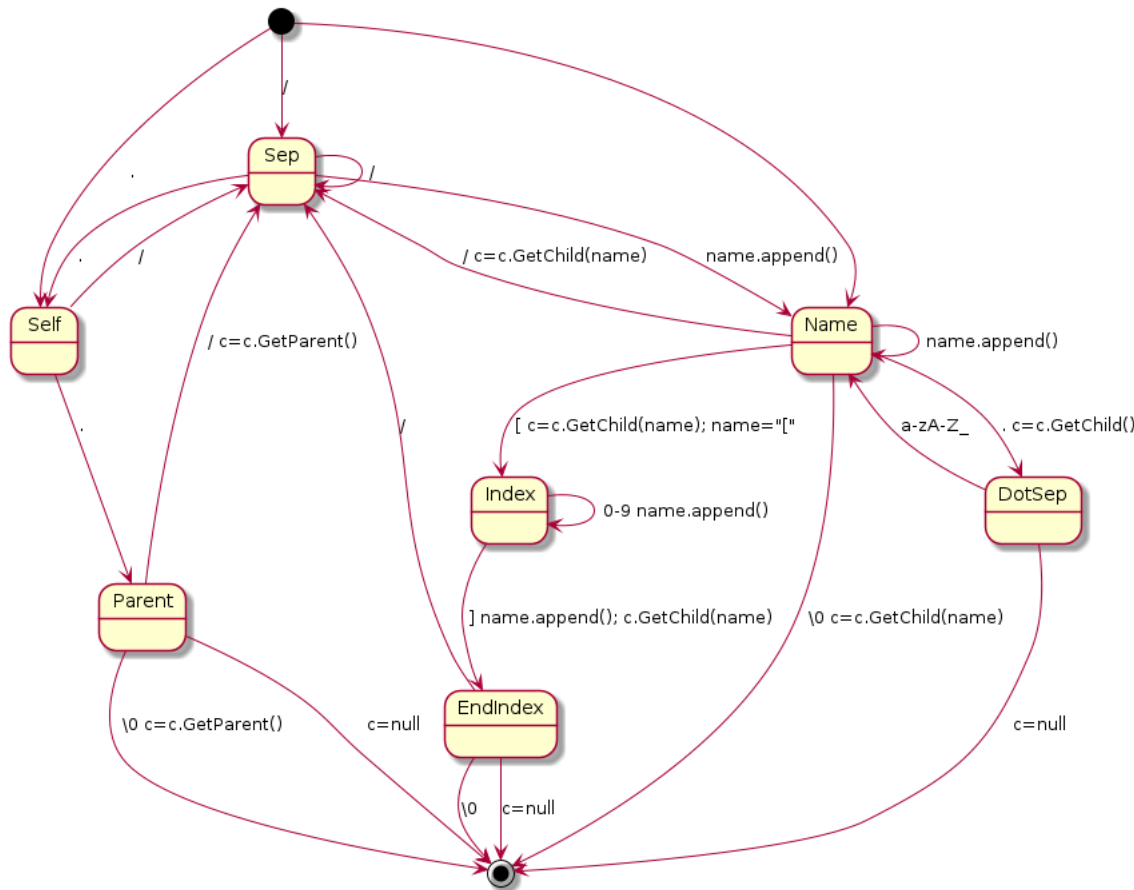


Figure 2: Path parsing state diagram

2.3 Scheduler / Time Keeper coordination

The scheduler / time keeper coordination relies on a specific event that is not defined in the SMP standard: **Scheduler_PreEventExecute** shall be registered in the event manager sometime during the simulator building phase. Then the following policy shall be applied to let the time keeper and scheduler fulfil each its own duty as defined in the SMP standard while being able to remain consistent with each other:

- the time keeper subscribes the **Scheduler_PreEventExecute** to trigger its own simulation time update process according the next scheduled event simulation time.
- the scheduler shall emit the **Scheduler_PreEventExecute** before running each scheduled entry point.

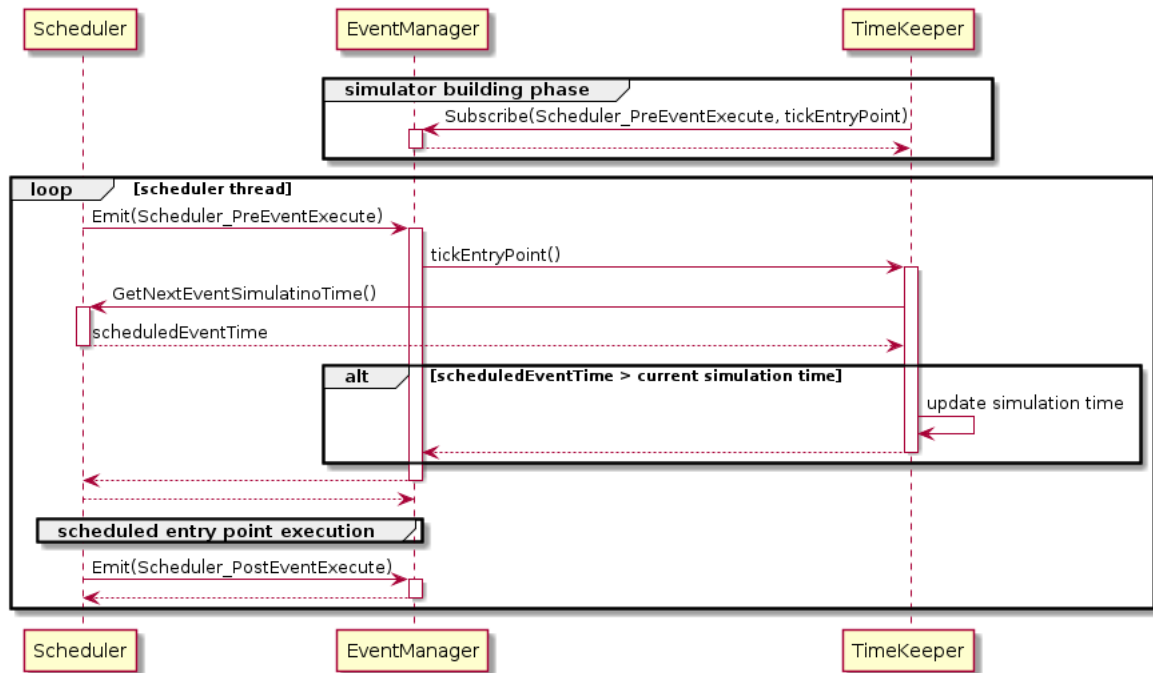


Figure 3: Scheduler / Time keeper coordination

Any alternate scheduler or time keeper service implementation may be used as soon the `Scheduler_PreEventExecute` event is handled applying the policy defined above in this section.

2.4 Simulation start / stop control

According to the SMP standard, the simulator shall emit events when starting and stopping the simulation:

- on simulation start request, if the current state allows it:
 - first `SMP_LeaveStandby`
 - then `SMP_EnterExecuting`
- on simulation stop request, if current state allows it:
 - first `SMP_LeaveExecuting`
 - then `SMP_EnterStandby`

On start request, the simulator preforms the follwing actions:

- schedule a start entry point immediately (to be first to execute).
- emit the `SMP_LeaveStandby` event.
- emit the `SMP_EnterExecuting` event.
- wait for the start process completion.

The start entry point shall:

- save the current thread identifier. Since it is executed by the scheduler thread, this will be used on stop to detect the Hold method is called by a scheduled entry point, or externally.
- notify the Start caller thread to release it.

The Simulator's Hold method shall stop the simulation according its boolean argument:

- when true: stop shall be immediate, this as soon the currently executed scheduled entry point is completed.

- when false: stop shall be delayed when all entry point scheduled for the current simulation time are completed.

To do so, the simulator on Hold request, according the boolean argument, schedules its own stop processing entry point:

- immediatly (being the first entry point in the schedule list), when argument is true.
- at simulation time now (being the last for the current simulation time in the schedule list), when the Hold argument is false.

Once the stop entry point is scheduled, the Hold method shall wait until the stop entry point is completed (if not triggered by the scheduler thread itself), then it emits the **SMP_EnterStandby** event.

The stop processing entry point shall emit the **SMP_LeaveExecuting** event, then it shall release the Hold caller thread and the simulator shall then switch to standby state emitting the related event.

The scheduler shall:

- subscribe to the **SMP_EnterExecuting** event to start its own scheduling thread.
- subscribe to the **SMP_LeaveExecuting** event to change its running status and end its own scheduling thread once the currently executing entry point is completed.

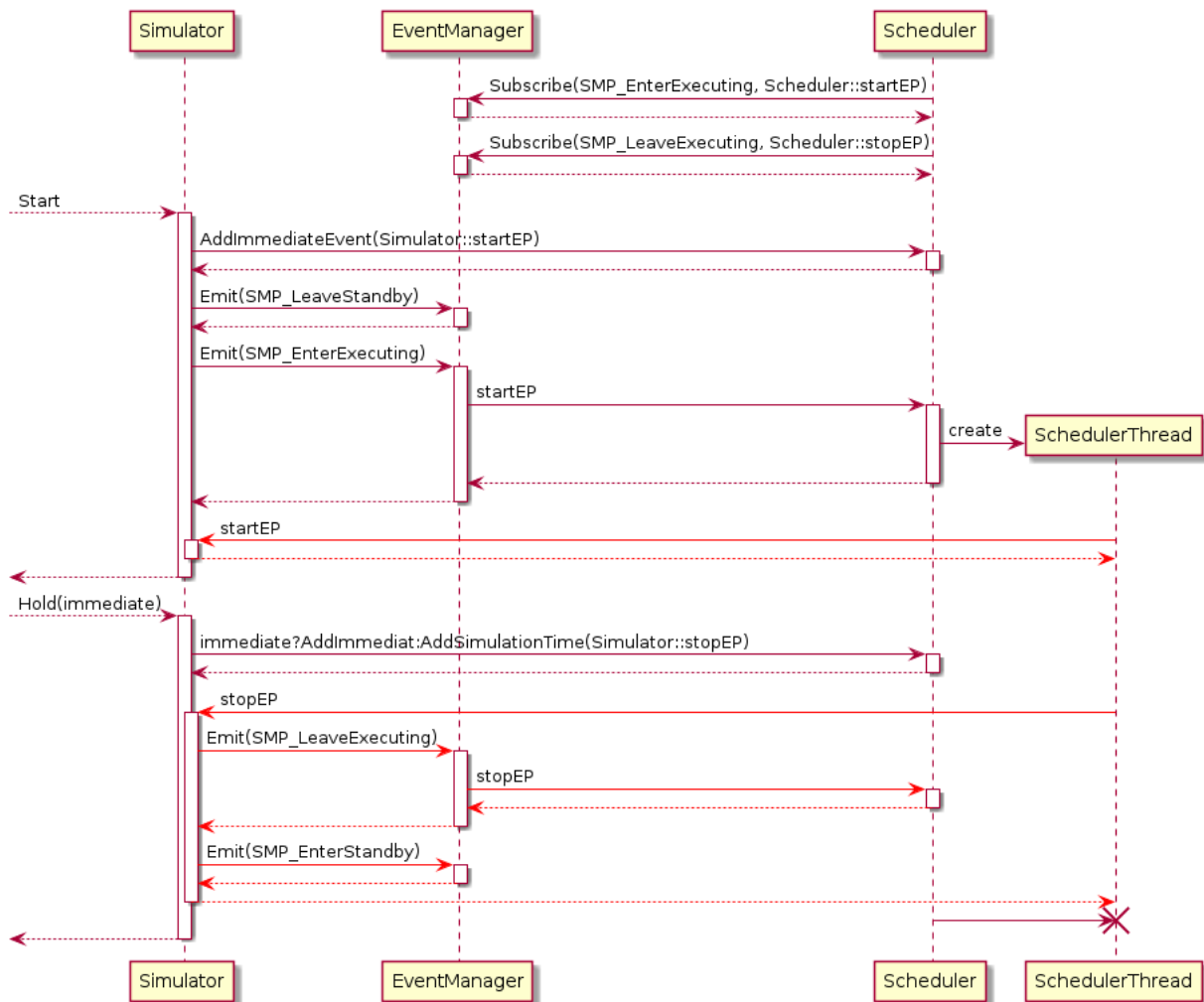


Figure 4: Scheduler / Time keeper coordination

2.5 Data propagation implicit schedule

When an entry point execution is completed, each of its owner's output field Push method is invoked to forward the possibly updated data to each connected input field.

2.6 aperiodic our out of clock scheduling

2.6.1 Simulation time change triggered by external event

Implementation strategy does not need a specific scheduler. The aperiodic and asynchronous event will synchronized the as fast as possible default scheduler behavior through the following strategy based on two entry points:

- 1st, a synchronization entry point that shall wait (block) until the next event that shall trigger the simulation time change. This synchronization entry point should be scheduled at a simulation time when all activities to be done before the expected event shall be completed. In most case when the simulation time is related to external events, the release entry point should be scheduled at Zulu time of the release event, making the simulation time reached the release event zulu time.
- 2nd, a time change entry point. This entry point does almost nothing, it is scheduled by the previous entry point, once the event is received an deblocking the waiting state at the simulation time value equal to the simulation time supposed to be reached when the synchronization event is received. This quite dummy entry point jut let have in the schedule queue list an event at the right time to trigger the simulation time change at the right value, even when no events are previously scheduled for the to reach simulation time. It finally reschedule the synchronisation entry point at the simulation time from which no more entry points are scheduled until the next synchronisation time.

2.6.2 Entry points scheduled on data event

In order to let entry point be activated (and consume CPU ressource) only when relevent, that is when the computed output shall be refreshed because of the update of an input, some specific Smp::IField subclass shall be implemented to override the SetValue() method. Then when a value is pushed by a connected output field:

- an assigned entry point is scheduled if any.
- this schedule is requested on various condition with any valid schedule option
 - only when value is change and not when new value is equal to current value.
 - immediate, now or delay.
 - with a period and repetition count.
- how to restrict the schedule when many fields are assign to a single entry point and avoid many shcedule of the same entry point at the same time [TBD1] .