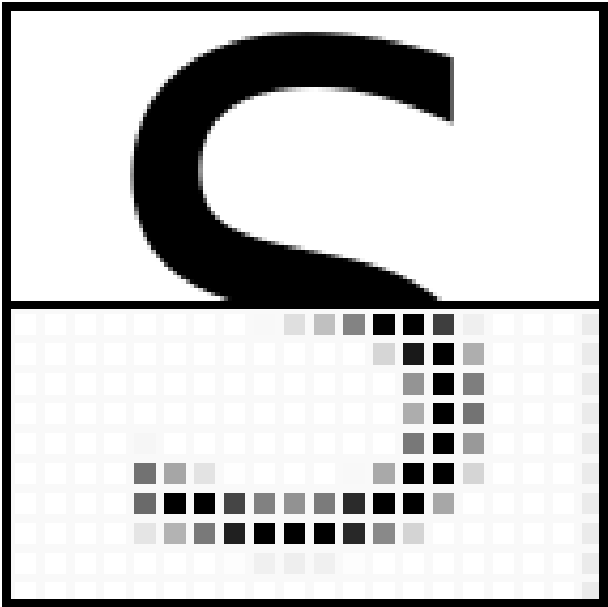


Simphonie definition file



AUTHOR	Sebastien Devaux
ABSTRACT	
KEYWORDS	simulation, simulation modelling platform, ECSS-E-ST-40-07C, kernel
CONTEXT	Component simphonie-1.0.12
PROCESSED	2025-07-17 07:44:00+02:00 / sdevaux@theia

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

Summary

1	Introduction	4
1.1	Reference	4
1.2	Scope	4
2	Requirements	4
2.1	SMP Level 1 features coverage	4
2.2	SMP Level 2 features coverage	4
2.3	Simphonie specific behaviors	4
2.3.1	Scheduler service	4
2.3.2	Smp::ISimulator implementation class design constraint	5
2.3.3	Logging service	5
2.3.4	Design constraints	7
2.4	SMP complements and extension	7
2.4.1	Design and implementation constraints	7
2.4.2	On event scheduling	7
2.4.3	Scheduling and entry point activation monitoring	8
2.5	Bundled service	8
2.5.1	Motivation and recommendations	8
2.5.2	Simulation stop control	8
2.5.3	Simulation real time synchronization service	9
2.5.4	Fields data recording service	9
3	Design	10
3.1	Mandatory services bundle	10
3.2	Path parsing	11
3.3	Scheduler / Time Keeper coordination	12
3.4	Simulation start / stop control	13
3.5	Data propagation implicit schedule	14
3.6	aperiodic out of clock scheduling	15
3.6.1	Simulation time change triggered by external event	15
3.6.2	Entry points scheduled on data event	15
3.7	Logging service	15
3.8	Scheduling Service	16
3.9	Storing System	18

List of Figures

1	Simulator services override	11
2	Path parsing state diagram	12
3	Scheduler / Time keeper coordination	13
4	Simulation start/stop sequence	14
5	Logger Front-end Architecture	15
6	Logger Back-end Architecture	16
7	Scheduler Architecture	16
8	StorageReader State Machine	18

1 Introduction

1.1 Reference

	Authors Reference	Title Edition
R1		<i>Simulation Modelling Platform (SMP)</i> https://ecss.nl/standard/ecss-e-st-40-07c-simulation-modelling-platform-2-march-2020/ <i>ECSS-E-ST-40-07C</i> 2 March 2020
R2	R. Gerhards:	<i>The Syslog Protocol</i> https://datatracker.ietf.org/doc/html/rfc5424 <i>RFC 5424</i> March 2009
R3		<i>High performance data management and storage suite</i> https://www.hdfgroup.org/solutions/hdf5/

1.2 Scope

This document is the definition file of the simphonie project. The purpose of the project is to provide a free simulation infrastructure implementation according the SMP standard [R1] . This definition file includes:

- additional requirements to complete the definition from the SMP standard.
- the design.
- a test plan.
- a user's guide, the targeted user being a simulation developer and integrator building its own simulator with SMP.

2 Requirements

2.1 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.smpdk.1

The simphonie project shall include a SMP development kit aiming at providing the concrete classes of the SMP interfaces needed to develop some components (services and models) libraries.

simph.smpdk.2

The SMP development kit shall provide a concrete class for the Smp::IObject interface.

simph.smpdk.3

The SMP development kit shall provide a concrete class for the Smp::IComponent interface.

2.2 SMP Level 2 features coverage

SMP L2 is to be considered in further development increment.

2.3 Simphonie specific behaviors

2.3.1 Scheduler service

simph.sched.1

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

simp.h.sched.2

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

simp.h.sched.3

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

2.3.2 Smp::ISimulator implementation class design constraint

simp.h.isim.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.

2.3.3 Logging service

simp.h.log.1

Logger service Log method implementation shall be thread safe.

Logging may be requested from many threads, at least:

- the simulator main thread for tracing the simulator control operations.
- the scheduler thread for tracing events from the scheduled entry points and models.

simp.h.log.cfg.1

The logger service shall publish a string input field to receive the path of the file where to record the log events.

simp.h.log.cfg.2

The default configuration of the logger shall be to record in a file named as the host simulator and located in the current working directory.

simp.h.log.cfg.3

The logger shall publish a boolean input field to configure the console (standard output) activation.

simp.h.log.cfg.4

The logger shall publish a string input field to define the log events to network send configuration.

simp.h.log.cfg.5

The logger shall publish an input field to optionally configure a buffered asynchronous event handling (see `simp.h.log.cfg.unsync.1`)

simp.h.log.fld.1

The logger shall publish an output field that counts the log events of error kind.

simph.log.fld.2

The logger shall publish an output field that counts the log events of warning kind.

simph.log.fld.3

The logger shall publish an output field that counts the log events of information kind.

simph.log.fld.4

The logger shall publish an output field that counts the log events of debug kind.

simph.log.fld.5

The logger shall publish an output field that counts the log events of event kind.

simph.log.fld.6

The logger shall publish an output field that counts all the log events of any kind.

simph.log.fld.7

The logger shall provide a reset entry point that sets all the log counter to 0 when executed.

simph.log.ev.1

Each log event shall include a zulu time attribute. Its value is the zulu time (as define by the SMP standard) when the event is submitted to the logger service.

simph.log.ev.2

Each log event shall include a simulation time attribute. Its value is the current simulation time got from the timekeeper when the event is submitted to the logger service.

simph.log.ev.3

Each log event shall include a severity attribute. Its value is defined from the Smp::Services::LogMessageKind argument of the Log method from the Logger service.

simph.log.ev.4

Each log event shall include a sender identifier attribute. Its value is defined from the Smp::IObject* argument of the Log method from the Logger service.

simph.log.ev.5

Each log event shall include a message attribute. Its value is the string argument of the Log method from the logger service.

simph.log.ev.6

Each log event shall include a thread identifier attribute. Its value identify the thread calling the Log method from the Logger service.

simph.log.unsync.1

When configured asynchronous, The logging service shall send the log event to a queue to be consumed by a distinct thread.

When running real-time simulation, blocking write request to mass memory may imply wait states leaving not enough margins to fulfil the real time requirements.

simph.log.net.1

When send to network mode is activate, the logging service shall send the event to a remote logging service.

simph.log.net.2

The send to network logging mode shall support the syslog protocol [R2] .

Designer is free to build a composite logging service where all optional features are handled by a subcomponent and activated when the related component is added to the service.

2.3.4 Design constraints

simpl.dsg.1

The simphonie Smp::ISimulator implementation shall allow to create many instance in the same process without side effects from on instance to another.

This requirement does not forbid to link the many simulator instances together as soon it is done on purpose, for instance for:

- start/stop the instances together.
- synchronize the scheduler of each instances with each other.
- propagate data or events from one instance to another. On purpose means the many simulator instances shall be explicitly configured to do so. The foreseen use cases for such simulator instances linking are, multithreaded simulation, co-simulation running composites models of a complex system that can't fit a single threaded simulation.

2.4 SMP complements and extension

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

2.4.1 Design and implementation constraints

simpl.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.

simpl.esmp.2

The SMP complements and extension shall be defined as interface (C++ pure virtual class headers) only as the SMP standard itself is.

simpl.esmp.3

The SMP complements and extension shall not derive from the SMP interfaces.

The implementers shall remain as free as possible for the design, a remains to be able to decide when a single class shall implements many interface or add the optional extended features by delegation or composition.

2.4.2 On event scheduling

simpl.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.

2.4.3 Scheduling and entry point activation monitoring

simph.esmp.msched.1

the schedule monitoring service shall include a method returning the actual state of the scheduler as a schedule list.

Each entry of the schedul list:

- schedule event id.
- entry point name or identifier.
- schedule parameter (period, count, etc).
- actual activation count.

It is assumed that the returned shchedule list may be absolute as soon it is returned, since when the simulation is running, the scheduler state and its internal scheduled event list evolves.

2.5 Bundled service

2.5.1 Motivation and recommendations

Implementing strictly the ECSS SMP standard is not enough to deliver a full featured simulation system. Additional services are needed. Such are omitted mostly because it does not engage simulation model interoperability and exchange capabilities that are the main purpose of the standard. Even there are no constraints at standard level, the implementer in the frame of the simphonie project shall consider to rely as much as possible on the standard itself with as less as possible coupling on the specific infrastructure design from simphonie. This means those service should be package in a library that can be loaded into any SMP infrastructure to enrich it the same way than it is as bundled with simphonie.

2.5.2 Simulation stop control

The simulation stop control service purpose is to let a simulator end the simulation by itself when some stop conditions are met.

simph.simctrl.1

The simulation control service shall request simulator hold (simulation stop) on user defined stop condition.

simph.simctrl.2

The simulation control service shall check the stop condition at least on simulation time change.

That means when many criteria shall evaluated to check the stop condition, it is not request to be done at every simulation state change, unless the condition include a criteria so transient that it shall be missed if its evaluation is delayed to next time change.

simph.simctrl.expr.1

The stop condition shall be defined from a boolean expression composed of any criteria defined as requirements of this section.

simph.simctrl.expr.2

The stop condition criteria shall accept a maximum simulation time criteria.

simph.simctrl.expr.3

The stop condition criteria shall accept the occurrence of a simulator event (events handled by the simulator's SMP Event Manager - see Smp::Srvices::IEventManager)

simph.simctrl.expr.4

The stop condition criteria shall accept condition on any model's published field's value (equality, inequality, greater, lower, etc).

2.5.3 Simulation real time synchronization service

simph.sync.1

The simulation synchronization service shall provide an entry point waiting for the Zulu time reaches the simulation time.

To run the simulation real time, the strategy is to schedule periodically an entry point to slow down an as fast as possible simulation to some how start the next simulation time only when enough real time as been spent. Since the Zulu time is the wall clock time having its absolute value, the purpose when saying the zulu time reaches the simulation time is not to wait until the Smp::Duration value are equals but rather the elapsed time since the simulation start is the same in both reference. This could be defined as wait until simulation time is equal to current zulu time minus the zulu time at simulation start.

simph.sync.2

The simulation synchronization service shall publish an output field for synchronization cycle overflow count.

simph.sync.3

The simulation synchronization service shall increment the cycle overflow count each time the expected zulu synchronization time is lesser the current zulu time when the synchronisation entry point is activated (leading to no wait).

simph.sync.4

The simulation synchronization service shall publish a Smp::Duration margin output field.

simph.sync.5

On synchronisation entry point start, the synchronization service shall set its margin output value as the difference between expected zulu time and current zulu time.

Monitoring the margin output field let the user track for the CPU load during the simulation. Positive values meaning there is time left at the end of the cycle, a negative value tells there is a cycle overflow and the amount of the time overflow.

simph.sync.6

The synchronization service shall be configurable or extensible to use any clock other than the simulation host computer's operating system's clock to define the zulu time reference.

2.5.4 Fields data recording service

The data recording service defined here is a feature and shall not be understood as a monolithic component implementing the `SMP::IService` interface. All the requirements from this section may be provided by many specialized components, each handling only a subset of the requirement (basically, each output file format handled by a specialized data recording component).

simph.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.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.datarec.3

The data recording service shall provide a link / connect relationship to define what fields shall be recorded.

simpH.datarec.4

The data recording service shall provide an schedulable entry point to record when activated one snapshot of the value of each field connected to the recording service.

simpH.datarec.5

the data recording service shall include the current simulation time in each snapshot record (as defined in **simpH.datarec.4**).

simpH.datarec.6

When configured so, the data recording service shall record many fields set into as many recording files.

simpH.datarec.7

The data recording service shall accept a same field to be part of many data recording set.

simpH.datarec.8

The data recording service shall record each data recording set at ist own rate.

simpH.datarec.9

The data recording service shall accept the selection any data set recording format distinct from the other formats selected for other data sets (in the limit of how many file formats are supported).

Assuling two file formats A and B are supported, a summary of the few requirements above is a simulator integrator shall be able to define a recording configuration where for instance:

- fields f1, f2 and f3 are recorded at 10Hz using file format A.
- fields f1, f3 and f5 are recorded at 20Hz using file format B.
- fields f4, f5, f6 and f7 are recorded at 50Hz using file format A.

simpH.datarec.csv.1

When selected by configuration, the data recording service shall record data using CSV text file format.

simpH.datarec.csv.2

The CSV data recording service shall start the output file with a content index line, detailing what field is recorded in each column.

simpH.datarec.csv.3

When outputing a numeric field's value, the CSV data recording service shall include as much digit than the underlying numeric type can handle.

The implementer shall take care that the usual 3, 6 or 10 digit truncation or rounding frequently included in the default number formatting are not activated but all digits are streamed. However, this does not mean that extre zero shall be added in front or after the last decimal to fill the digit slots. 1.5 remains 1.5 not 00000001.5 neither 1.50000000.

simpH.datarec.hdf5.1

When selected by configuration, the data recording service shall record data using the HDF5 file format (see [R3]).

3 Design

3.1 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.

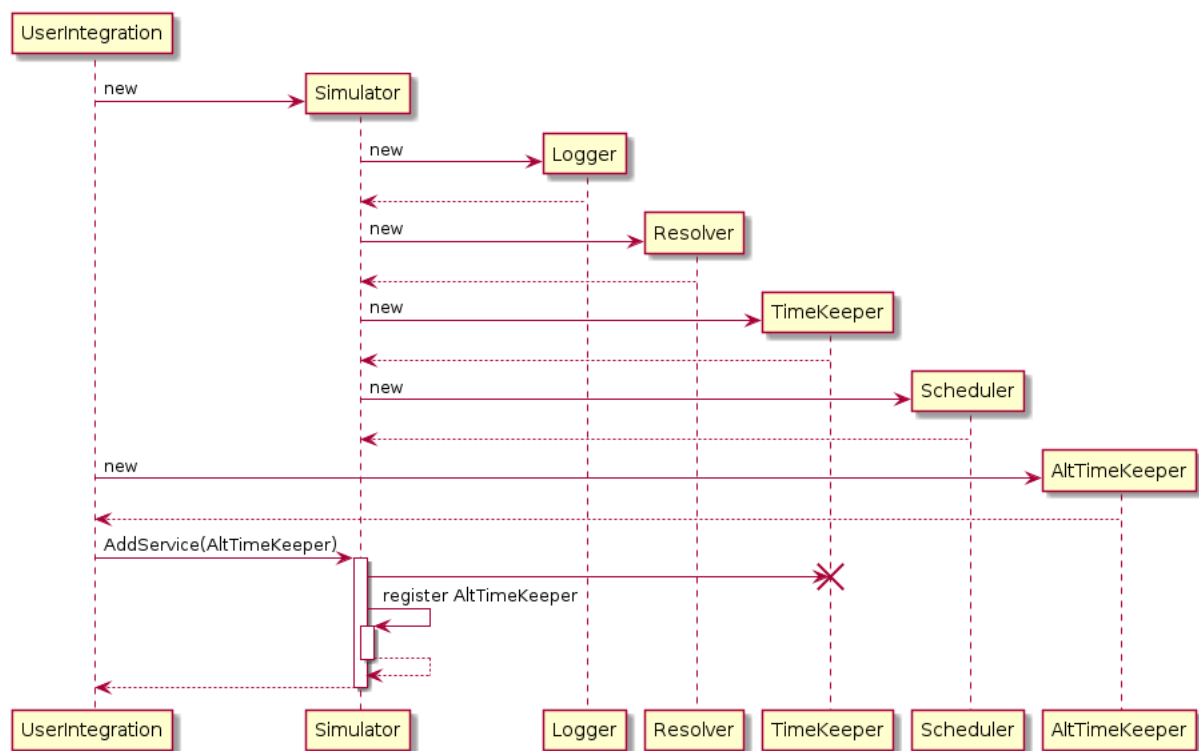


Figure 1: Simulator services override

3.2 Path parsing

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

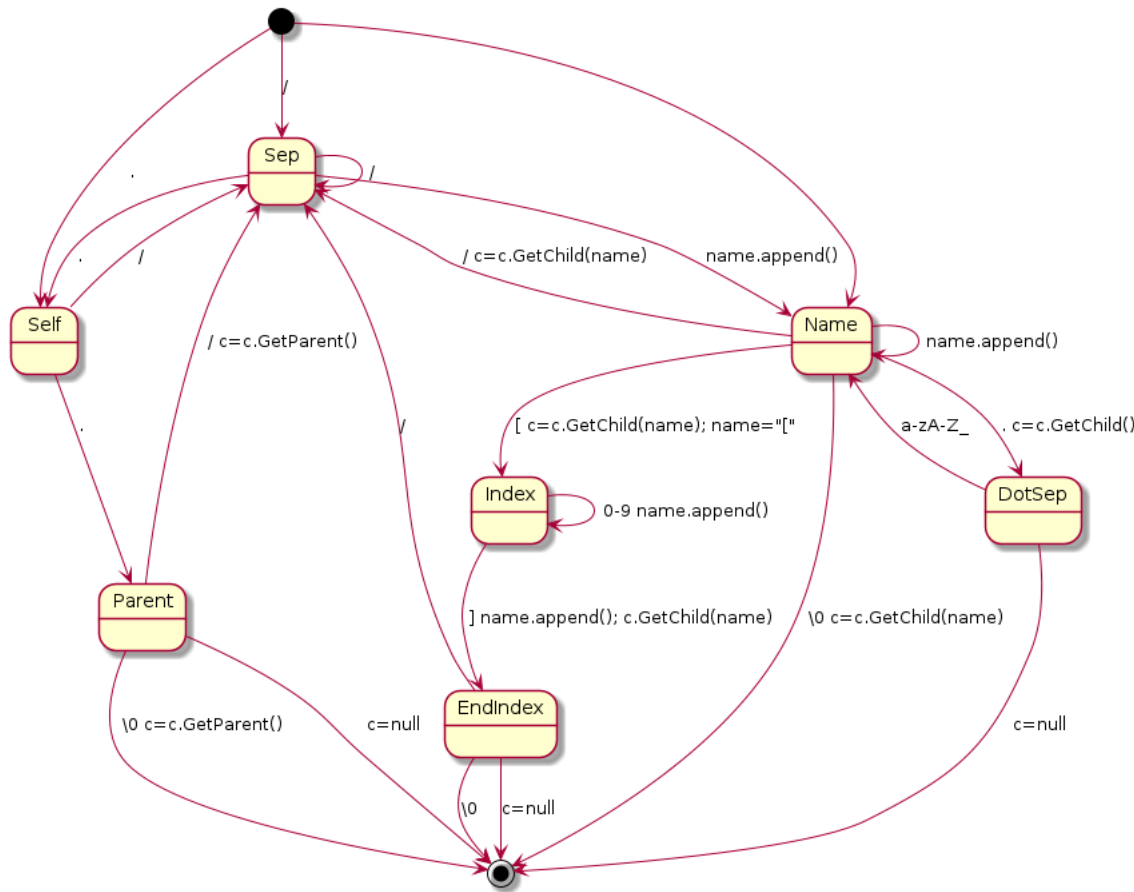


Figure 2: Path parsing state diagram

3.3 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.

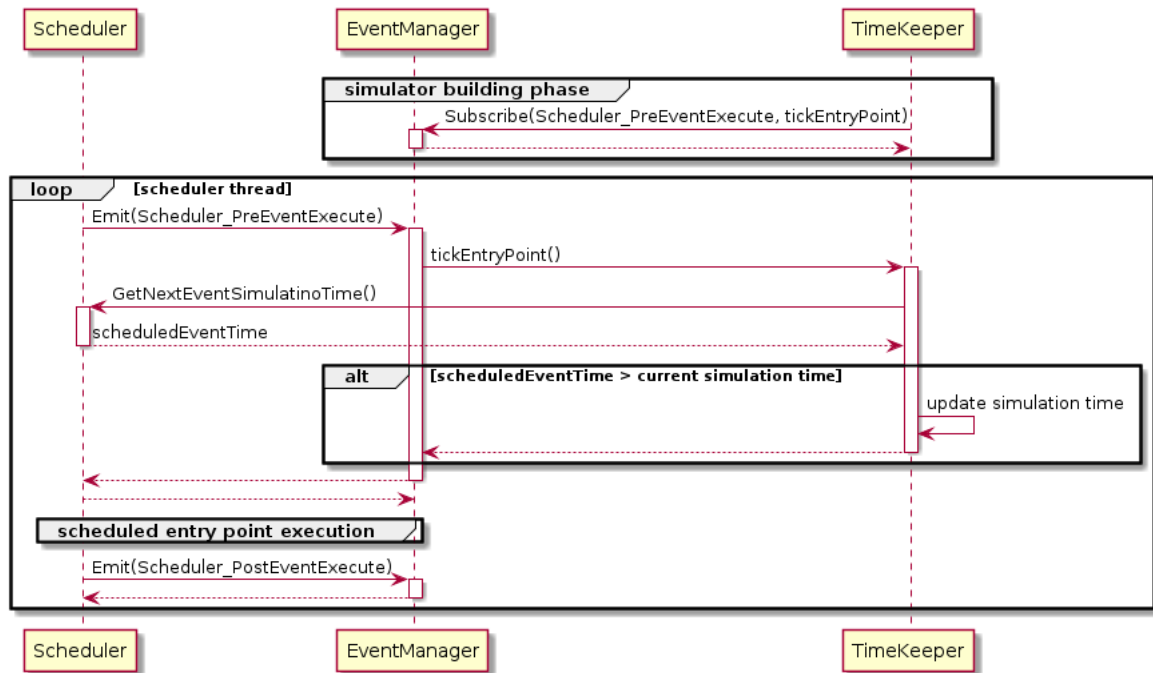


Figure 3: Scheduler / Time keeper coordination

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.

3.4 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.

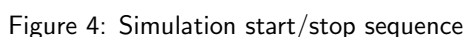
The Simulator's Hold method shall stop the simulation according its boolean argument:

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

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

- 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.



8f4041c- ©2019-2025 Sebastien Devaux

3.6 aperiodic out of clock scheduling

3.6.1 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.

3.6.2 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] .

3.7 Logging service

The logging service is splitted into a front-end logger (c.f. class named Logger) and several back-ends ones. The front-end logger acts as an abstraction layer as, regardless of how the log messages are processed, the system has to call its log function to log anything. Then, the logs are spread to all the back-ends for further handling. The communication between both sides is achieved by a standardization of the logs (c.f. class named LoggerEvent). The standardization provides an access to the logs' content as well as one of their string representation: back-ends can either use this string or build their own one. Back-ends should be child components of the front-end logger, contained in the "Backends" container.

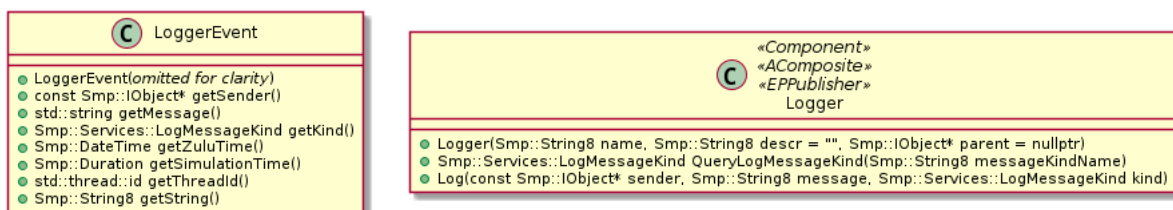


Figure 5: Logger Front-end Architecture

Simphonie provides four differents back-ends:

- LoggerOStream: push error logs to stderr, debug ones to stdout, and the others to stdout.

- **LoggerFile**: save the logging events to a file. The filepath is published as an input field.
- **LoggerNetwork**: send the logs to a remote logging service that support the Syslog Protocol [R2] . Connection configuration is published as an input field. This back-end is not yet implemented.
- **LoggerAsync**: store the logs in a buffer to be consumed later. Buffer's size is published as an input field.

Users can create their own back-ends by implementing the `ILoggerBackend` interface. Moreover, users are free to use whatever they want as back-ends' can be duplicated or can stay unused.

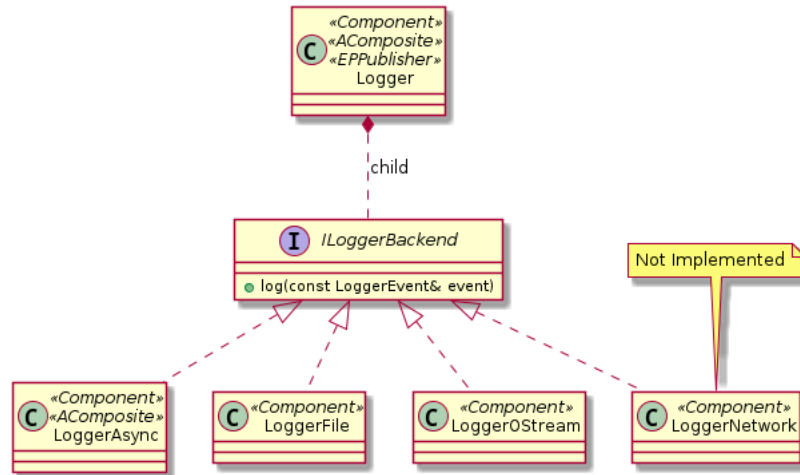


Figure 6: Logger Back-end Architecture

3.8 Scheduling Service

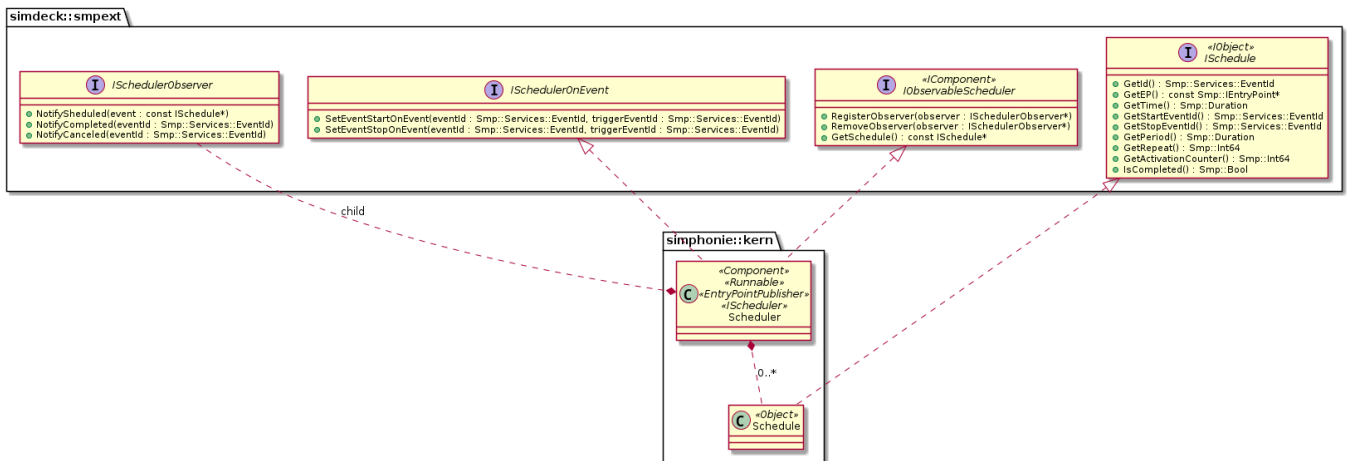


Figure 7: Scheduler Architecture

3.9 Storing System

