---

layout: default

title: Logical Viewpoint

nav\_order: 5

parent: SpesML Modeling Framework

grand\_parent: SpesML Concepts

permalink: /concepts/modeling\_framework/logical\_viewpoint.html

---

# Logical Viewpoint

This document describes the basic concepts that will be covered in the \_logical viewpoint\_ and how they are mapped to `SysML elements`.

\*Content:\*

- [Logical Viewpoint](#logical-viewpoint)

- [Logical Viewpoint: Overview](#logical-viewpoint-overview)

- [Logical Components and their relations to other model elements of the Logical Viewpoint](#logical-components-central-model-elements-of-the-logical-viewpoint)

- [Logical Context: The System and its Context in the Logical Viewpoint](#logical-context-the-system-and-its-context-in-the-logical-viewpoint)

- [Interface behavior of Logical Components](# interface -behavior-of-logical-components)

- [Decomposition of Logical Components](#decomposition-of-logical-components)

- [Tracing Relations between Logical Components and other Elements](#tracing-relations-between-logical-components-and-other-elements)

- [Modeling the Transition to the Technical Viewpoint](#modeling-the-transition-to-the-technical-viewpoint)

- [Well-formedness Rules](#well-formedness-rules)

# Logical Viewpoint: Overview

The logical viewpoint (LVP) describes how the system under development (SuD) can be structured in order to achieve the behavior which is specified in the [functional view](https://spesml.github.io/concepts/modeling\_framework/functional\_viewpoint.html) independent of the technical realization.

To do so, the LVP defines how the models of the Logical View (LV) describe the [logical system context](#logical-context-the-system-and-its-context-in-the-logical-viewpoint), the [decomposition](#decomposition-of-logical-components) of the SuD in logical components and sub-components and their respective [interface behavior](#modeling-interfaces-and-behavior-of-logical-components).

In the following we describe the basic concepts and models for the LV.

# Logical Components: Central Model Elements of the Logical Viewpoint

The central modeling concept of the logical viewpoint are the so-called \_logical components\_.

They strictly follow the [universal interface model](https://spesml.github.io/concepts/modeling\_framework/uim.html) including syntactic and semantic interface, causality and compositionality.

In the SpesML tooling a logical component is represented as a dedicated `SysML Block` as can be seen in the example in Figure 1:

![Logical Component](/images/logical\_viewpoint/logical-component.png){:width="400" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 1: </b><em>A SysML Block representing a logical component</em></div>

# Logical Context: The System and its Context in the Logical Viewpoint

The \_logical context\_ models the environment in which the SuD shall operate on a logical level.

This can be actors that interact with the SuD or external logical components.

One main objective in modeling the logical context is also to specify the logical interface of the SuD to the operational context on a logical (i.e. technology independent) level.

Hereby, the logical context model refines the context model of the [functional context](https://spesml.github.io/concepts/modeling\_framework/functional\_viewpoint.html#functional-context-the-system-and-its-context-in-the-functional-viewpoint) in the FV. Hereby, usual refinements concern the interfaces and interface types at the system boundary to the system context.

In SysML, the logical context is modelled using a specific `SysML Internal Block Diagram (IBD)` that can be created in the respective package as can be seen in Figure 2:

![Logical Context in the Logical Viewpoint](/images/logical\_viewpoint/logical-context\_creation.png){:width="250" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 2: </b><em>The logical context diagram can be modeled in the dedicated package of the logical view.</em></div>

As for the functional and technical view, the logical context defines the scope of what is \_inside\_ the system boundary and what is \_outside\_.

The logical components that shall be outside the system boundary (i.e. in the context) can be specified by a dedicated boolean property with name \_external\_ per component.

In the SpesML SysML-profile, the logical components for the SuD and the element of the context are connected in the *logical context diagram* as can be seen in Figure 3.

![Logical Context](/images/logical\_viewpoint/logical-context.png){:width="1000" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 3: </b><em>An example for a logical system context.</em></div>

More information on modeling the system context can be found [in the context documentation](https://spesml.github.io/concepts/modeling\_framework/context.html).

# Interface Behavior of Logical Components

As already mentioned, also the logical viewpoint is based on the concepts of the [universal interface model](https://spesml.github.io/concepts/modeling\_framework/uim.html) and it’s mapping to SysML.

This means, that interfaces of logical components consist of a \_syntactical\_ and a \_semantical\_ interface.

The \_syntactical\_ interfaces of logical components is modeled using \_logical interfaces\_ which are represented by `SysML Proxy Ports`.

Hereby, every logical interface needs to be assigned a \_logical interface type\_ which is defined as a set of `SysML Channels`.

An exemplary definition of a logical interface and interface types can be seen in Figures 4 and 5:

![Logical Interface Type](/images/logical\_viewpoint/logical-intreface-type.png){:width="300" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 4: </b><em>Exemplary definition of a SpesML logical interface type</em></div>

![Logical Interface](/images/logical\_viewpoint/logical-intreface.png){:width="400" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 5: </b><em>Exemplary definition of a SpesML logical interface</em></div>

The \_semantical\_ interface of logical components is modelled by means of state machines that operate based on the component’s input streams and produce the component's output streams.

Hereby, the guards and actions can not only contain comparison and assignment operations, but also use [executable functions](https://spesml.github.io/concepts/modeling\_framework/executable\_functions.html).

More information on SpesML [state machines](https://spesml.github.io/concepts/modeling\_framework/state\_machines.html) can be found in the respective documentation article.

Figure 6 gives an example in the SpesML tool:

![State Machine](/images/logical\_viewpoint/state-machine.png){:width="1000" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 6: </b><em>An example for a SpesML state machine.</em></div>

# Decomposition of Logical Components

To structure the logical system architecture and to reduce complexity, all logical components can be decomposed into sub-components. This means that the behavior of a decompsed component results from the the composed behavior of the sub-components.

In the SpesML SysML-profile and plugin, decomposition is modeled by means of dedicated `SysML Internal Block Diagrams (IBDs)` in which `SysML Parts` are instantiated and connected for the subcomponents as can be seen in Figure 7:

![Composition](/images/logical\_viewpoint/decomposition.png){:width="1000" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 7: </b><em>Decomposition of a logical component into sub-components.</em></div>

Decomposition of logical components also involves the syntactic and semantic interface of components and sub-components:

Here, all inputs of the component need to be connected to an input of a sub-component and all outputs of a component need to be connected to an output of a subcomponent.

More detailed information can be found in the documentation of the [universal interface model](https://spesml.github.io/concepts/modeling\_framework/uim.html).

# Tracing Relations between Logical Components and other Elements

The logical components are traced to the whitebox functions which they \_realize\_ with a respective tracing matrix.

Hereby, we recommend to trace n whitebox-functions of the FV to exactly \_one\_ logical component (n:1 tracing).

In the SpesML profile and plugin, we use `SysML Matrixes` for this tracing as can be seen in Figure 8:

![Tracing from FV to LV](/images/logical\_viewpoint/tracing\_fvp-lvp.png){:width="600" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 8: </b><em>Tracing from the functional to the logical view.</em></div>

The technical realization of the logical components will be described by in the [technical view](https://spesml.github.io/concepts/modeling\_framework/technical\_viewpoint.html).

Hereby, we recommend to trace every logical component to exactly one technical component (1:1 tracing).

More detailed information on tracing in SpesML can be found [in the documentation of tracing](https://spesml.github.io/concepts/modeling\_framework/tracing.html).

# Modeling the Transition to the Technical Viewpoint

The logical view can also explicitly model certain aspects of the transition from the logical to the technical viewpoint, as is introduced [in the technical viewpoint documentation](https://spesml.github.io/concepts/modeling\_framework/tracing.html#definition-of-software-components-in-the-logical-architecture) in more detail..

In particular the identification of software sub-systems in the technical view can be modelled here, by decomposing logical components into different sub-components for software and non-software aspects.

These sub-components can then be regrouped in separate logical components that can then be technically modelled accordingly in the [technical view](https://spesml.github.io/concepts/modeling\_framework/technical\_viewpoint.html).

Figures 9 and 10 show how to model the transition to the technical view for software aspects:

![Specifying Software in the LV](/images/logical\_viewpoint/specify\_logical\_software.png){:width="400" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 9: </b><em>To model the transition from logical to technical viewpoint, logical components can be declared to become software in the technical view.</em></div>

![Software in the LV](/images/logical\_viewpoint/logical-software-diagram.png){:width="800" :class="img-responsive" style="display:block; margin-left:auto; margin-right:auto"}

<div align="center"><b>Figure 10: </b><em>Logical components which will become software in the technical view are indicated in diagrams with a yellow color.</em></div>

# Well-formedness Rules

I order to obtain a valid SpesML model when modeling logical views, there are several \_well-formedness rules (WFR)\_ that need to be adhered.

For the structural aspects and the syntactic interface modeling, this includes all [WFRs of the universal interface model] (https://spesml.github.io/concepts/modeling\_framework/uim.html).

For state machines and the semantic interface modeling, this includes the WFRs of [state machines](https://spesml.github.io/plugin/state\_machines.html) and [executable functions](https://spesml.github.io/plugin/executable\_functions.html#automatic-checks-well-formedness-rules).

Besides this, there are no dedicated well-formedness rules which are [specific for the logical viewpoint](https://spesml.github.io/concepts/modeling\_framework/rules.html).