Lando SSL Version 2 Well-formedness

September 9, 2024

1 Notation

Lists of t are written $[t]$; $[]_t$ is an empty list of t's, $x :: xs$ is the list with head x and tail xs, and $[x]$ is the singleton list containing x. $last(l)$ returns the last element of (necessarily non-empty) list l. An optional t is written $t?$; a present option value x is written $\lceil x \rceil$ and a missing option value of type t is written $\cdot_t$. We write $x.y$ for the projection of field y from a node x . In the pattern x@`Foo`$\{\ldots\}$, x is bound to the entire node labeled `Foo`. The pattern `(Foo|Bar)`$\{\ldots\}$ matches nodes labeled either `Foo` or `Bar`. Disjoint union of maps is written $\uplus$, and is undefined if the domains of the maps are not disjoint. If m_1 and m_2 are maps then $m_1 \lhd m_2$ (pronounced "m_1 shadows m_2") is $m_1 \cup \{(k, m_2(k)) | k \in Dom(m_2) - Dom(m_1)\}$. An empty map is written $\emptyset$. A judgement applies only if all premises are fully defined.

2 Raw AST Grammar

This is intended to correspond precisely to the `RawAST` structure built by the current parser, omitting source position tags and comments. The AST corresponds directly to the concrete syntax, with one exception: implicit bodies of `system`s and `subsystem`s are converted into explicit ones.

$$
\begin{array}{rcl}
s \in \mathit{source} & \rightarrow & \mathtt{Source}\{\mathtt{elems} : [\mathit{elem}]\} \\
e \in \mathit{elem} & \rightarrow & \mathtt{System}\{\mathtt{name} : \mathit{name}, \mathtt{abbrev} : \mathit{name}?, \mathtt{explanation} : \mathit{text}, \mathtt{indexings} : [\mathit{indexing}], \mathtt{body} : [\mathit{elem}]\} \\
 & \rightarrow & \mathtt{Subsystem}\{\mathtt{name} : \mathit{name}, \mathtt{abbrev} : \mathit{name}?, \mathtt{inherits} : [\mathit{qname}], \mathtt{clientOf} : [\mathit{qname}], \\
 & & \qquad \mathtt{explanation} : \mathit{text}, \mathtt{indexings} : [\mathit{indexing}], \mathtt{body} : [\mathit{elem}]\} \\
 & \rightarrow & \mathtt{SubystemImport}\{\mathtt{name} : \mathit{qname}, \mathtt{abbrev} : \mathit{name}?, \mathtt{clientOf} : [\mathit{qname}]\} \\
 & \rightarrow & \mathtt{Component}\{\mathtt{name} : \mathit{name}, \mathtt{abbrev} : \mathit{name}?, \mathtt{inherits} : [\mathit{qname}], \mathtt{clientOf} : [\mathit{qname}], \\
 & & \qquad \mathtt{explanation} : \mathit{text}, \mathtt{parts} : [\mathit{part}]\} \\
 & \rightarrow & \mathtt{ComponentImport}\{\mathtt{name} : \mathit{qname}, \mathtt{abbrev} : \mathit{name}?, \mathtt{clientOf} : [\mathit{qname}]\} \\
 & \rightarrow & \mathtt{Events}\{\mathtt{name} : \mathit{name}, \mathtt{events} : [\mathit{item}]\} \\
 & \rightarrow & \mathtt{Scenarios}\{\mathtt{name} : \mathit{name}, \mathtt{scenarios} : [\mathit{item}]\} \\
 & \rightarrow & \mathtt{Requirements}\{\mathtt{name} : \mathit{name}, \mathtt{requirements} : [\mathit{item}]\} \\
 & \rightarrow & \mathtt{Relation}\{\mathtt{name} : \mathit{qname}, \mathtt{inherits} : [\mathit{qname}], \mathtt{clientOf} : [\mathit{qname}]\} \\
x \in \mathit{indexing} & \rightarrow & \mathtt{Indexing}\{\mathtt{key} : \mathit{text}, \mathtt{values} : [\mathit{text}]\} \\
p \in \mathit{part} & \rightarrow & \mathtt{Constraint}\{\mathtt{text} : \mathit{text}\} \\
 & \rightarrow & \mathtt{Query}\{\mathtt{text} : \mathit{text}\} \\
 & \rightarrow & \mathtt{Command}\{\mathtt{text} : \mathit{text}\} \\
i \in \mathit{item} & \rightarrow & \mathtt{Item}\{\mathtt{id} : \mathit{name}, \mathtt{text} : \mathit{text}\} \\
u \in \mathit{uid} & & \text{(base type)} \\
t \in \mathit{text} & & \text{(base type)} \\
n \in \mathit{name} & & \text{(base type)} \\
q \in \mathit{qname} & = & [\mathit{name}] \\
c \in \mathit{comment} & & \text{(base type)}
\end{array}
$$

3 Well-formedness Judgements

Although the AST does not explicitly include element labels, we assume that each node (in particular each *elem*) has a unique identity with a well-defined notion of equality.

$\Gamma : \mathit{name} \to \mathit{elem}$ is an environment (finite map) from names to their corresponding elements.

$\Phi : \mathit{elem} \to \Gamma$ is a finite map from (sub)systems to the environments they generate.

The function $\mathit{qlook}_\Gamma(q)$ resolved a qualified name q starting in environment Γ; it is defined thus:

$$\begin{array}{rcl} \mathit{qlook}_\Gamma([n]) & = & \Gamma(n) \\ \mathit{qlook}_\Gamma(n :: ns) & = & \mathit{qlook}_{\Phi(\Gamma(n))}(ns) \end{array}$$

$I : \mathit{elem} \times \mathit{elem}$ is an inheritance relation between elements, where $(e_1, e_2) \in I$ means e_1 inherits from e_2. The predicate $\mathit{nocycles}(I)$ holds when I has no cycles.

We assume the existence of a function $\mathit{qnames}(t)$ that returns the list of *qname*s mentioned in *text* t.

A specification s is well-formed if it is possible to find a top-level environment Γ_0, a global Φ_0, and a global inheritance relation I_0 (all implicitly threaded everywhere) such that $\vdash s$.

Below we provide a short description of each judgement:

- **Judgement 1** Global well-formedness
  - *A `Source` top level element containing a list of elements es is well formed if elements in es are well formed, there are no cycles, there is only one `System` element, and all elements in es are valid top level elements*
  - $\Gamma_0 \vdash es \Rightarrow \Gamma_0$: This premise checks the well-formedness of a list of elements es in the context of a top-level environment Γ_0. It asserts that es is well formed in the context of Γ_0
  - $\mathit{nocycles}(I)$: This states that the global inheritance relation I must not contain any cycles.
  - The third premise ensures uniqueness of `System` elements within es. It states that if e_1 and e_2 are both `System` elements within es, they must be the same entity ($e_1 = e_2$).
  - $\forall e \in es, \texttt{valid-toplevel}(e)$: This checks that every element e in es satisfies the `valid-toplevel` judgment, meaning each element is valid as a top-level construct.
  - $\vdash \texttt{Source}\{\texttt{elems} = es\}$: If all the above premises are satisfied, the conclusion states that a `Source` element containing a list of elements es is well-formed. This `Source` element acts as a container or root for the structure defined by the elements in es.

$$\frac{\begin{array}{c}\Gamma_0 \vdash es \Rightarrow \Gamma_0 \qquad \mathit{nocycles}(I) \\ \forall e_1, e_2 \in es, e_1 = \texttt{System}\{\ldots\} \wedge e_2 = \texttt{System}\{\ldots\} \implies e_1 = e_2 \qquad \forall e \in es, \texttt{valid-toplevel}(e)\end{array}}{\vdash \texttt{Source}\{\texttt{elems} = es\}} \tag{1}$$

- **Judgement 2** Recursive well-formedness
  - *A list of elements* $e :: es$ *is well-formed if both* e *and* es *are well formed. The resulting environment* Γ *is a union of* Γ' *and* Γ''*. This means we can process the elements recursively.*
  - $\Gamma \vdash e \Rightarrow \Gamma'$: This premise asserts that a single element e is well-formed in the context of an environment Γ, resulting in a possibly modified environment Γ'.
  - $\Gamma \vdash es \Rightarrow \Gamma''$: Similarly, this checks the well-formedness of a list of elements es in the same initial environment Γ, resulting in another possibly modified environment Γ''. I1t's assessing the collective impact of the remaining elements in the list after e.
  - $\Gamma \vdash e :: es \Rightarrow \Gamma' \uplus \Gamma''$: The conclusion states that if both premises are true, then the entire list formed by concatenating e at the head of es ($e :: es$) is well-formed in the initial environment Γ. The resulting environment after processing this concatenated list is the disjoint union of Γ' and Γ'' (denoted as $\Gamma' \uplus \Gamma''$).

$$\frac{\Gamma \vdash e \Rightarrow \Gamma' \qquad \Gamma \vdash es \Rightarrow \Gamma''}{\Gamma \vdash e :: es \Rightarrow \Gamma' \uplus \Gamma''} \tag{2}$$

- **Judgement 3** Terminating condition for recursive well-formedness
  - An empty list of elements is always well formed.

$$\frac{}{\Gamma \vdash []_{elem} \Rightarrow \{\}} \tag{3}$$

- **Judgement 4** System well-formed
  - *A System element is well formed if all the preconditions are valid.*
  - $n \neq n_a$: This states that the name n of the `System` element must not be equal to its abbreviation n_a. This ensures distinct identifiers for the full name and the abbreviation.
  - $\Gamma \vdash t$: This checks the well-formedness of the `explanation` text t in the current environment Γ.
  - $\forall e \in es_b, \texttt{valid-contains}(e_s, e)$: For every element e in the body es_b of the `System`, this condition asserts that e can be contained within the the `System`
  - $\Phi_0(e_s) = \Gamma'$: This premise specifies that the environment generated by the `System` element e_s is Γ'
  - $\Gamma' \lhd \Gamma \vdash es_b \Rightarrow \Gamma'$: This checks the well-formedness of the body es_b of the `System` in an environment that is a combination of Γ' (specific to the `System`) and Γ (the broader environment), with Γ' taking precedence in case of overlaps.
  - If all the above premises are satisfied, then the `System` element e_s, defined with a name n, an optional abbreviation n_a, an explanation t, and a body es_b (along with other unspecified properties represented by $\ldots$), is well-formed in the environment Γ. Furthermore, the resulting environment from processing this `System` element maps both n and n_a to e_s.

$$\frac{n \neq n_a \qquad \Gamma \vdash t \qquad \forall e \in es_b, \texttt{valid-contains}(e_s, e) \qquad \Phi_0(e_s) = \Gamma' \qquad \Gamma' \lhd \Gamma \vdash es_b \Rightarrow \Gamma'}{\Gamma \vdash e_s@\texttt{System}\{\texttt{name} = n, \texttt{abbrev} = \lceil n_a \rceil, \texttt{explanation} = t, \texttt{body} = es_b, \ldots\} \Rightarrow \{n \mapsto e_s, n_a \mapsto e_s\}} \tag{4}$$

- **Judgement 5** System well-formed
  - *A System element is well formed if all the preconditions are valid.*
  - The same as Judgement 4 except for systems without the abbreviation n_a - in this case `abbrev` is empty (None).

$$\frac{\Gamma \vdash t \qquad \forall e \in es_b, \texttt{valid-contains}(e_s, e) \qquad \Phi_0(e_s) = \Gamma' \qquad \Gamma' \lhd \Gamma \vdash es_b \Rightarrow \Gamma'}{\Gamma \vdash e_s@\texttt{System}\{\texttt{name} = n, \texttt{abbrev} = \cdot_{name}, \texttt{explanation} = t, \texttt{body} = es_b, \ldots\} \Rightarrow \{n \mapsto e_s\}} \tag{5}$$

- **Judgement 6** Subsystem well-formed
  - *A `Subsystem` is well formed if all the preconditions are valid.*
  - The name and the abbreviated name must be different.
  - e_s must have only valid inherit relations qs_i
  - e_s must contain only valid elements es_b
  - e_s must have only valid client relations qs_c

– $\Phi_0(e_s) = \Gamma'$: This premise specifies that the environment generated by the `Subsystem` element e_s is Γ'

– $\Gamma' \lhd \Gamma \vdash es_b \Rightarrow \Gamma'$: The well-formedness of the body es_b of the `Subsystem` is checked in an environment that combines Γ' and Γ, with Γ' taking precedence.

– If all those conditions are met, then the subsystem is well formed.

$$\frac{\begin{array}{c} n \neq n_a \qquad \forall q \in qs_i, \texttt{valid-inherit}(e_s, qlook_\Gamma(q)) \qquad \forall q \in qs_c, \texttt{valid-client}(e_s, qlook_\Gamma(q)) \\ \Gamma \vdash t \qquad \forall e \in es_b, \texttt{valid-contains}(e_s, e) \qquad \Phi_0(e_s) = \Gamma' \qquad \Gamma' \lhd \Gamma \vdash es_b \Rightarrow \Gamma' \end{array}}{\Gamma \vdash e_s@\texttt{Subsystem}\left\{ \begin{array}{l} \texttt{name} = n, \texttt{abbrev} = \lceil n_a \rceil, \texttt{inherits} = qs_i, \\ \texttt{clientOf} = qs_c, \texttt{explanation} = t, \texttt{body} = es_b, \ldots \end{array} \right\} \Rightarrow \{n \mapsto e_s, n_a \mapsto e_s\}} \tag{6}$$

- **Judgement 7** Subsystem well-formed
  – *A `Subsystem` is well formed if all the preconditions are valid.*
  – The same as Judgement 6 except for subsystems without the abbreviation n_a (abbreviation will be empty).

$$\frac{\begin{array}{c} \forall q \in qs_i, \texttt{valid-inherit}(e_s, qlook_\Gamma(q)) \qquad \forall q \in qs_c, \texttt{valid-client}(e_s, qlook_\Gamma(q)) \\ \Gamma \vdash t \qquad \forall e \in es_b, \texttt{valid-contains}(e_s, e) \qquad \Phi_0(e_s) = \Gamma' \qquad \Gamma' \lhd \Gamma \vdash es_b \Rightarrow \Gamma' \end{array}}{\Gamma \vdash e_s@\texttt{Subsystem}\left\{ \begin{array}{l} \texttt{name} = n, \texttt{abbrev} = \cdot_{name}, \texttt{inherits} = qs_i, \\ \texttt{clientOf} = qs_c, \texttt{explanation} = t, \texttt{body} = es_b, \ldots \end{array} \right\} \Rightarrow \{n \mapsto e_s\}} \tag{7}$$

- **Judgement 8** Subsystem import
  – *A `SubsystemImport` is well formed if all the preconditions are valid.*
  – $qlook_{\Gamma_0}(q) = e$: This premise asserts that the qualified name q resolves to an element e in the top-level environment Γ_0.
  – $\forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))$: For each qualified name q in the list qs_c (which represents elements for which the imported subsystem is a client), the element corresponding to q must have a valid client relationship with e (the element resolved from q).
  – If the above premises are satisfied, then the `SubsystemImport` element e_s, defined with a reference name q, an optional abbreviation n_a, and a list of clients qs_c, is well-formed in the environment Γ. The resulting environment from processing this import maps the abbreviation n_a to the `SubsystemImport` element e_s.

$$\frac{qlook_{\Gamma_0}(q) = e \qquad \forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))}{\Gamma \vdash e_s@\texttt{SubsystemImport}\{\texttt{name} = q, \texttt{abbrev} = \lceil n_a \rceil, \texttt{clientOf} = qs_c\} \Rightarrow \{n_a \mapsto e_s\}} \tag{8}$$

- **Judgement 9** Subsystem import (no n_a)
  – *A `SubsystemImport` is well formed if all the preconditions are valid.*
  – Same as Judgement 8, except for imported subsystems without an abbreviation n_a.

$$\frac{qlook_{\Gamma_0}(q) = e \qquad \forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))}{\Gamma \vdash e_s@\texttt{SubsystemImport}\{\texttt{name} = q, \texttt{abbrev} = \cdot_{name}, \texttt{clientOf} = qs_c\} \Rightarrow \{last(q) \mapsto e_s\}} \tag{9}$$

- **Judgement 10** Component well-formed

- *A `Component` is well formed if all the preconditions are valid.*
- Distict name and and the abbreviated name
- Valid client and valid inherit relationships
- The explanation t must be well formed
- Each part p must be well formed
- If all those premises are met, the component e_c is well formed

$$\frac{\begin{array}{c} n \neq n_a \qquad \forall q \in qs_i, \texttt{valid-inherit}(e_c, qlook_\Gamma(q)) \\ \forall q \in qs_c, \texttt{valid-client}(e_c, qlook_\Gamma(q)) \qquad \Gamma \vdash t \qquad \forall p \in ps, \Gamma \vdash p \end{array}}{\Gamma \vdash e_c@\texttt{Component}\left\{ \begin{array}{l} \texttt{name} = n, \texttt{abbrev} = \lceil n_a \rceil, \texttt{inherits} = qs_i, \\ \texttt{clientOf} = qs_c, \texttt{explanation} = t, \texttt{parts} = ps \end{array} \right\} \Rightarrow \{n \mapsto e_c, n_a \mapsto e_c\}} \tag{10}$$

- **Judgement 11** Component well formed (no n_a)
  - *A `Component` is well formed if all the preconditions are valid.*
  - Same as Judgement 10, except for components without an abbreviated name.

$$\frac{\begin{array}{c} \forall q \in qs_i, \texttt{valid-inherit}(e_c, qlook_\Gamma(q)) \\ \forall q \in qs_c, \texttt{valid-client}(e_c, qlook_\Gamma(q)) \qquad \Gamma \vdash t \qquad \forall p \in ps, \Gamma \vdash p \end{array}}{\Gamma \vdash e_c@\texttt{Component}\left\{ \begin{array}{l} \texttt{name} = n, \texttt{abbrev} = \cdot_{name}, \texttt{inherits} = qs_i, \\ \texttt{clientOf} = qs_c, \texttt{explanation} = t, \texttt{parts} = ps \end{array} \right\} \Rightarrow \{n \mapsto e_c\}} \tag{11}$$

- **Judgement 12** Constraint well-formed
  - If the text t is well-formed, a `Constraint` is also well-formed

$$\frac{\Gamma \vdash t}{\Gamma \vdash \texttt{Constraint}\{\texttt{text} = t\}} \tag{12}$$

- **Judgement 13** Query well-formed
  - If the text t is well-formed, a `Query` is also well-formed

$$\frac{\Gamma \vdash t}{\Gamma \vdash \texttt{Query}\{\texttt{text} = t\}} \tag{13}$$

- **Judgement 14** Command well-formed
  - If the text t is well-formed, a `Command` is also well-formed

$$\frac{\Gamma \vdash t}{\Gamma \vdash \texttt{Command}\{\texttt{text} = t\}} \tag{14}$$

- **Judgement 15** Component import
  - *A `ComponentImport` is well formed if all the preconditions are valid.*
  - Qualified name q resolves to an element e in the top level environment

- $\forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))$: For each qualified name q in the list qs_c (representing elements for which the imported component is a client), the element corresponding to q must have a valid client relationship with e.
- If the above defined premises are satisfied, the imported component is well-formed.

$$\frac{qlook_{\Gamma_0}(q) = e \qquad \forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))}{\Gamma \vdash e_c@\texttt{ComponentImport}\{\texttt{name} = q, \texttt{abbrev} = \lceil n_a \rceil, \texttt{clientOf} = qs_c\}\} \Rightarrow \{n_a \mapsto e_c\}} \tag{15}$$

- **Judgement 16** Component import without n_a
  - *A* `ComponentImport` *is well formed if all the preconditions are valid.*
  - Like Judgement 15, except covering cases where the imported component does not have an abbreviated name.

$$\frac{qlook_{\Gamma_0}(q) = e \qquad \forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))}{\Gamma \vdash e_c@\texttt{ComponentImport}\{\texttt{name} = q, \texttt{abbrev} = \cdot_{name}, \texttt{clientOf} = qs_c\}\} \Rightarrow \{last(q) \mapsto e_c\}} \tag{16}$$

- **Judgement 17** Events well-formed
  - If a list of events is is well formed, then an element e of type *Events* is well formed

$$\frac{\Gamma \vdash is \Rightarrow \Gamma'}{\Gamma \vdash e@\texttt{Events}\{\texttt{name} = n, \texttt{events} = is\} \Rightarrow \{n \mapsto e\} \uplus \Gamma'} \tag{17}$$

- **Judgement 18** Scenarios well-formed
  - If a list of scenarios is is well formed, then an element e of type *Scenarios* is well formed

$$\frac{\Gamma \vdash is \Rightarrow \Gamma'}{\Gamma \vdash e@\texttt{Scenarios}\{\texttt{name} = n, \texttt{scenarios} = is\} \Rightarrow \{n \mapsto e\} \uplus \Gamma'} \tag{18}$$

- **Judgement 19** Requirements well-formed
  - If a list of scenarios is is well formed, then an element e of type *Requirements* is well formed

$$\frac{\Gamma \vdash is \Rightarrow \Gamma'}{\Gamma \vdash e@\texttt{Requirements}\{\texttt{name} = n, \texttt{requirements} = is\} \Rightarrow \{n \mapsto e\} \uplus \Gamma'} \tag{19}$$

- **Judgement 20** Item well-formed
  - If the text t is well-formed, then an item i is well-formed.

$$\frac{\Gamma \vdash t}{\Gamma \vdash i@\texttt{Item}\{\texttt{id} = n, \texttt{text} = t\} \Rightarrow \{n \mapsto i\}} \tag{20}$$

- **Judgement 21** Relation well-formed
  - Qualified name q must be resolved to an element e
  - All inherit relations are valid

- All client relations are valid
- A well-formed relation does not add a new element to Γ, it only needs to be valid within the environment

$$\frac{qlook_\Gamma(q) = e \qquad \forall q \in qs_i, \texttt{valid-inherit}(e, qlook_\Gamma(q)) \qquad \forall q \in qs_c, \texttt{valid-client}(e, qlook_\Gamma(q))}{\Gamma \vdash \texttt{Relation}\{\texttt{name} = q, \texttt{inherits} = qs_i, \texttt{clientOf} = qs_c\} \Rightarrow \{\}} \tag{21}$$

- **Judgement 22** Text well-formed
  - This judgment defines the well-formedness criteria for a piece of text t focusing on the resolution of names within that text. It's a rule that ensures every name mentioned in the text t corresponds to an existing element in the current environment.
  - $\forall n_t \in qnames(t), \exists e, \Gamma(n_t) = e$: For each name n_t that appears within the text t (as identified by the function $qnames(t)$), there must exist an element e such that n_t resolves to e in the environment Γ. This means that every name referenced in t must have a corresponding, defined element in the current environment.
  - $\Gamma \vdash t$: If the above premise is satisfied, then the text t is considered well-formed in the environment Γ. This conclusion states that the text is valid within the context of the environment, assuming that all names it mentions are properly accounted for and linked to existing elements.

$$\frac{\forall n_t \in qnames(t), \exists e, \Gamma(n_t) = e}{\Gamma \vdash t} \tag{22}$$

- **Judgement 23** Valid top-level
  - $\texttt{valid-toplevel((System|Subsystem|...)}\{\ldots\}\texttt{)}$: The rule states that the described elements are always valid as top-level elements. The $\{\ldots\}$ indicates that the specific contents of these elements are not relevant to this judgment

$$\frac{}{\texttt{valid-toplevel((System|Subsystem|Component|Events|Scenarios|Requirements|Relation)}\{\ldots\}\texttt{)}} \tag{23}$$

- **Judgement 24** Valid elements of a `System`
  - This rule states that a `System` can contain only `Subsystems`, imported `Subsystems` or `Relations`

$$\frac{}{\texttt{valid-contains}(\texttt{System}\{\ldots\}, \texttt{(Subsystem|SubsystemImport|Relation)}\{\ldots\})} \tag{24}$$

- **Judgement 25** Valid elements of a `SubSystem`
  - This rule states that a `SubSystem` contains only `Subsystems`, imported `Subsystems`, `Components`, imported `Components`, and elements defined in Judgement 26.

$$\frac{}{\texttt{valid-contains}(\texttt{Subsystem}\{\ldots\}, \texttt{(Subsystem|SubsystemImport|Component|ComponentImport)}\{\ldots\})} \tag{25}$$

- **Judgement 26** Valid elements of a `SubSystem`
  - This rule states that a `SubSystem` contains only `Scenarios`, `Requirements`, `Events`, `Relations`, and elements defined in Judgement 25.

$$\frac{}{\texttt{valid-contains(Subsystem}\{\ldots\}\texttt{, (Scenarios|Requirements|Events|Relation)}\{\ldots\}\texttt{)}} \tag{26}$$

- **Judgement 27** Well-formed inheritance
  - $(e_1, e_2) \in I_0$: This premise states that there is an inheritance relationship between two elements e_1 and e_2 in the global inheritance relation I_0. In other words, e_1 inherits from e_2.
  - `valid-inherit`(e_1@`Component`$\{\ldots\}$, e_2@`Component`$\{\ldots\}$): If the above premise is satisfied, this conclusion asserts that the inheritance relationship between e_1 and e_2 is valid, where both e_1 and e_2 are `Component` elements. The $\{\ldots\}$ again implies that the specific contents or attributes of the `Component` elements are not relevant to this judgment.

$$\frac{(e_1, e_2) \in I_0}{\texttt{valid-inherit}(e_1\texttt{@Component}\{\ldots\}, e_2\texttt{@Component}\{\ldots\})} \tag{27}$$

- **Judgement 28** Client of an (imported) subsystem
  - This rule describes which elements can be a client of an (imported) subsystem.
  - `Sbstm|SbstmImprt` stands for `Subsystem|SubsystemImport`

$$\frac{}{\texttt{valid-client((Sbstm|SbstmImprt)}\{\ldots\}\texttt{, (Sbstm|SbstmImprt|Component|ComponentImport)}\{\ldots\}\texttt{)}} \tag{28}$$

- **Judgement 29** Client of an (imported) component
  - This rule describes which elements can be a client of an (imported) component.
  - `Cmpnt|CmpntImprt` stands for `Component|ComponentImport`

$$\frac{}{\texttt{valid-client((Cmpnt|CmpntImprt)}\{\ldots\}\texttt{, (Subsystem|SubsystemImport|Cmpnt|CmpntImprt)}\{\ldots\}\texttt{)}} \tag{29}$$