# Formal semantics for CMMN+

January 23, 2026

## 1 Introduction

This supplementary document provides the complete formal semantics for the R2 and R3 notational enhancements of the CMMN-based notational variant (CMMN+). Due to space limitations in the article, this document offers a detailed mathematical specification of the semantic structures underlying the proposed notational enhancements. These formal semantics do not introduce new execution semantics beyond standard CMMN, but instead provide a precise interpretation of the additional notational constructs and metadata introduced by CMMN+. The document is organized into two sections, corresponding to enhancements R2 and R3.

## 2 Formal Semantics for Notational Enhancement R2

The notational enhancement R2 of the CMMN-based notational variant (CMMN+) is presented using formal semantics.

First, the basic structure is introduced, namely the case model element, which is formally represented as: $M = \langle E, A, R \rangle$[1] where:

- $E$ is the set of all elements that constitute the model,
- $A$ is the set of all attributes of elements in $E$,
- $R$ is the set of all relations between elements in the model.

We define a new set $F$, which represents the set of all CaseFileItem elements, and specify that $F$ is a subset of the set $E$, as follows:

$$F \subseteq E,$$

[1] Angle brackets are commonly used for ordered tuples, meaning a sequence of elements where order matters. The set $M$ is represented with angle brackets to indicate a structured object composed of multiple components in a defined order, with each component carrying a specific meaning.

where each $f \in F$ belongs to the set of model elements $E$ and has attributes defined by the set $A$. Relations $R$ are defined as subsets of the Cartesian product of the element set $E$, written as:

$$R \subseteq E \times E,$$

and they may connect arbitrary elements of the model, including elements from the set $F$. The set of attributes $A$ includes both existing and newly introduced attributes for all model elements. Each attribute is represented as a function that maps an element $e \in E$ to its corresponding attribute value, where Values denote the universe of attribute values, defined as:

$$A = \{a_1, a_2, ..., a_k\}, \quad a_i : E \rightharpoonup Values,$$

which implies which implies that attribute values are defined only for elements for which the attribute is applicable. We further define a new set $D$, representing the set of all type definitions for CaseFileItem elements, defined by the `CaseFileItemDefinition` class, expressed as:

$$d \in D.$$

Since each CaseFileItem element is defined by attributes of both the `CaseFileItem` class and the `CaseFileItemDefinition` class, we formalize both classes and their respective attributes. Let $A_{\text{CaseFileItem}}$ denote the set of attributes defined in the `CaseFileItem` class, expressed as:

$$A_{\text{CaseFileItem}} = \{c_1, c_2, ..., c_k\},$$

where each attribute $c_i \in A_{\text{CaseFileItem}}$ is a function assigning a value to an element $f \in F$, written as:

$$c_i : F \rightharpoonup Values,$$

indicating that the attributes are specific to each instance $f \in F$ and bound to its contextual definition. Similarly, let $A_{\text{CaseFileItemDefinition}}$ denote the set of attributes defined in the `CaseFileItemDefinition` class, written as:

$$A_{\text{CaseFileItemDefinition}} = \{d_1, d_2, ..., d_m\},$$

where each attribute $d_i \in A_{\text{CaseFileItemDefinition}}$ is a function assigning a value to an element $d \in D$, defined as:

$$d_i : D \rightharpoonup Values,$$

indicating that these attributes specify the type definition for all CaseFileItem elements associated with the corresponding context.

Each element $f \in F$ is associated with exactly one type definition from the set $D$. This is formalized using the relation $r_{definition}$, which connects elements in $F$ to their corresponding type definitions in $D$:

$$r_{definition} \subseteq F \times D,$$

such that for every $f \in F$, the following holds:

$$\exists! d \in D,\ (f,d) \in r_{definition},$$

which means that there exists a specific $d \in D$ representing one of the type definitions assigned to the given CaseFileItem element. The pair $(f,d)$ belongs to the relation $r_{definition}$, linking the CaseFileItem element $f$ to its type definition $d$.

Each $f \in F$ is described by a combination of attributes, where the attributes of the `CaseFileItem` class are represented as:

$$\{a_1(f), a_2(f), ..., a_k(f)\},$$

where each $a_i \in A_{\text{CaseFileItem}}$. This attribute combination also includes the attributes from the `CaseFileItemDefinition` class, which are linked to the type definition $d$ associated with a particular CaseFileItem element $f$, expressed as:

$$\{d_1(d), d_2(d), ..., d_m(d)\},$$

where $d_i \in A_{\text{CaseFileItemDefinition}}$. In summary, a complete CaseFileItem element can be expressed as:

$$CaseFileItem(f) = \langle A_{\text{CaseFileItem}}(f), A_{\text{CaseFileItemDefinition}}(d)\rangle,$$

where $A_{\text{CaseFileItem}}(f)$ denotes the values of the attributes from the `CaseFileItem` class for element $f$, and $A_{\text{CaseFileItemDefinition}}(d)$ denotes the attribute values from the `CaseFileItemDefinition` class, where $d$ is the definition linked to $f$. The CaseFileItem element is therefore described as a combination of attributes from both classes.

The attributes of the `CaseFileItem` class are represented by the set $A_{\text{CaseFileItem}}$, defined as:

$$A_{\text{CaseFileItem}} = \{ac_1, ac_2, ac_3, ac_4, ac_5, ac_6, ac_7, ac_8, ac_9, ac_{10}, ac_{11}, ac_{12}\}.$$

The individual attributes are presented below.

Attribute $ac_1 \equiv name : F \to \text{String}$
Meaning: Each element $f \in F$ has a name, represented as a string.
Rule: $name(f)$ returns a string that denotes the name of the element $f$.
Constraint: $\forall f \in F,\ name(f) \in \text{String} \land name(f) \neq \emptyset$.
Example: If $f_1 \in F$, then $name(f_1) =$ "Document".

Attribute $ac_2 \equiv multiplicity : F \to \text{MultiplicityEnum}$
Meaning: Specifies the possible number of instances of the element $f$.
Rule: $multiplicity(f)$ returns one of the defined values; the default is (1).
Constraint: $\forall f \in F,\ multiplicity(f) \in \text{MultiplicityEnum}$.
Example: If $f_2 \in F$ includes only one instance, then $multiplicity(f_2) =$ "1".
The enumeration MultiplicityEnum is defined as:

$$MultiplicityEnum = \{(0..1), (0..*), (1), (1..*)\}.$$

Attribute $ac_3 \equiv resourceState : F \rightarrow \text{ResourceStateEnum}$
Meaning: Defines the current life cycle state of the element $f$.
Rule: $resourceState(f)$ returns one of the defined values; the default is "Unspecified".
Constraint: $\forall f \in F,\ resourceState(f) \in \text{ResourceStateEnum}$.
Example: If $f_3 \in F$ represents a document under review, then $resourceState(f_3) =$ "InReview".
The enumeration ResourceStateEnum is defined as:

$$\begin{aligned} ResourceStateEnum = \{&Draft,\ InReview,\ InApproval,\ Approved,\ Published,\\ &Unpublished,\ Archived,\ Canceled,\ Unspecified,\\ &Unknown\}. \end{aligned}$$

Attribute $ac_4 \equiv locationRef : F \rightarrow \text{String}$
Meaning: Contains a reference with the logical location of the element $f$.
Rule: $locationRef(f)$ returns a valid textual string representing the location.
Constraint: $\forall f \in F,\ locationRef(f) \in (\text{String} \cup \{\emptyset\})$.
Example: If $f_4 \in F$, then $locationRef(f_4) =$ "C:/Documents/Contract.pdf".

Attribute $ac_5 \equiv access : F \rightarrow \text{AccessEnum}$
Meaning: Specifies access rights to the element $f$.
Rule: $access(f)$ returns one of the defined values; the default is "Unspecified".
Constraint: $\forall f \in F,\ access(f) \in \text{AccessEnum}$.
Example: If $f_5 \in F$ represents a document that is public and read-only, then $access(f_5) =$ "Public_ReadOnly".
The enumeration AccessEnum is defined as:

$$\begin{aligned} AccessEnum = \{&Private\_ReadOnly,\ Private\_Editable,\ Public\_ReadOnly,\\ &Public\_Editable,\ Unspecified,\ Unknown\}. \end{aligned}$$

Attribute $ac_6 \equiv author : F \rightarrow \text{String}$
Meaning: Specifies the author who created or is associated with the element $f$.
Rule: $author(f)$ returns a valid textual string representing the name or identifier of the author.
Constraint: $\forall f \in F,\ author(f) \in (\text{String} \cup \{\emptyset\})$.
Example: If $f_6 \in F$ represents a document created by "John Doe", then $author(f_6) =$ "John Doe".

Attribute $ac_7 \equiv version : F \rightarrow \text{String}$
Meaning: Specifies the version of the element $f$.
Rule: $version(f)$ returns a string representing the version.
Constraint: $\forall f \in F,\ version(f) \in (\text{String} \cup \{\emptyset\})$.
Example: If $f_7 \in F$ represents a document in version "1.0", then $version(f_7) =$ "1.0".

Attribute $ac_8 \equiv definitionRef : F \rightarrow D$
Meaning: Each element $f$ must be associated with exactly one type definition.

Rule: $definitionRef(f)$ returns a reference to the type definition $d$.
Constraint: $\forall f \in F,\ (definitionRef(f) \in D) \wedge (definitionRef(f) \neq \emptyset) \wedge (\exists! d \in D,\ definitionRef(f) = d)$.
Example: If $f_8 \in F$, then $definitionRef(f_8) = d_1$, where $d_1 \in D$.

Attribute $ac_9 \equiv children : F \to \mathcal{P}(F)$[2]
Meaning: Indicates the subset of $f$ that are its direct or indirect children.
Rule: $children(f)$ returns the set of children of $f$. If there are none, it returns $\emptyset$.
Constraint: $\forall f \in F,\ children(f) = \emptyset \vee children(f) \subseteq F$.
Example: If $f_9 \in F$, then $children(f_9) = \{f_5, f_6\}$.

Attribute $ac_{10} \equiv parent : F \to F \cup \{\emptyset\}$
Meaning: Indicates the parent of the element $f$, if it exists.
Rule: $parent(f)$ returns the parent. If none exists, it returns $\emptyset$.
Constraint: $\forall f \in F,\ parent(f) \in (F \cup \{\emptyset\})$.
Example: If $f_{10} \in F$, then $parent(f_{10}) = f_9$.

Attribute $ac_{11} \equiv targetRefs : F \to \mathcal{P}(F)$
Meaning: Indicates the set of other elements $f$ that the element $f$ refers to as targets.
Rule: $targetRefs(f)$ returns a set of references to other $f$ elements. If there are none, it returns $\emptyset$.
Constraint: $\forall f \in F,\ targetRefs(f) = \emptyset \vee targetRefs(f) \subseteq F$.
Example: If $f_{11} \in F$, then $targetRefs(f_{11}) = \{f_{12}, f_{13}\}$.

Attribute $ac_{12} \equiv sourceRef : F \to F \cup \{\emptyset\}$
Meaning: Indicates the source of the element $f$, if it exists.
Rule: $sourceRef(f)$ returns a reference to the source element $f$. If none exists, it returns $\emptyset$.
Constraint: $\forall f \in F,\ sourceRef(f) \in (F \cup \{\emptyset\})$.
Example: If $f_{12} \in F$, then $sourceRef(f_{12}) = f_{11}$.

The *CaseFileItemDefinition* class is represented by the set $A_{CaseFileItemDefinition}$, which includes all attributes defined within this class:

$$A_{CaseFileItemDefinition} = \{ad_1, ad_2, ad_3, ad_4, ad_5, ad_6, ad_7, ad_8\}$$

The attributes are described in the following section.

Attribute $ad_1 \equiv name : D$[3] $\to$ String
Meaning: Each definition $d \in D$ has a name represented as a string.
Rule: $name(d)$ returns a string indicating the name of the definition.
Constraint: $\forall d \in D,\ name(d) \in String \wedge name(d) \neq \emptyset$.

[2] $\mathcal{P}$ denotes the powerset and is commonly used in set theory to indicate that the attribute may contain multiple or no elements.
[3] D denotes the set of CaseFileItemDefinition instances

Example: If $d_1 \in D$ represents the definition type "Document", then $name(d_1) =$ "Document".

Attribute $ad_2 \equiv definitionType : D \to \mathrm{URI}$
Meaning: Specifies a URI that identifies the definition $d$ for a data element $f$.
Rule: $definitionType(d)$ returns one of the predefined values, with "Unspecified" as the default.
Constraint: $\forall d \in D,\ definitionType(d) \in URI$.
Example: If $d_2 \in D$ represents the definition "Document", then
$definitionType(d_2) =$ "http://example.org/Document".
The set of allowed values for the URI is defined as:

$$\begin{aligned} URI = \{&Folder\ in\ CMIS,\ Document\ in\ CMIS,\\ &Relationship\ in\ CMIS,\ XML - Schema\ Element,\\ &XML - Schema\ Complex\ Type, XML - Schema\ Simple\ Type,\\ &Unspecified,\ Unknown\}. \end{aligned}$$

Attribute $ad_3 \equiv structureRef : D \to \mathrm{QName}$
Meaning: Specifies a qualified name (QName) that refers to the concrete structure of the definition.
Rule: $structureRef(d)$ returns a valid QName.
Constraint: $\forall d \in D,\ structureRef(d) \in (QName \cup \{\emptyset\})$.
Example: If $d_3 \in D$ represents a definition that refers to an XML Schema, then
$structureRef(d_3) =$ "xs:ComplexType".

Attribute $ad_4 \equiv description : D \to \mathrm{String}$
Meaning: Contains the textual description of the definition $d$.
Rule: $description(d)$ returns a valid string.
Constraint: $\forall d \in D,\ description(d) \in (String \cup \{\emptyset\})$.
Example: If $d_4 \in D$ represents a definition for a document, then
$description(d_4) =$ "Definition for a legal contract".

Attribute $ad_5 \equiv status : D \to StatusEnum$
Meaning: Specifies the current status of the definition $d$ in its lifecycle.
Rule: $status(d)$ returns one of the predefined values; the default is "Unspecified".
Constraint: $\forall d \in D,\ status(d) \in StatusEnum$.
Example: If $d_5 \in D$ represents a currently active definition, then
$status(d_5) =$ "Active".
The enumeration for possible values is defined as:

$$StatusEnum = \{Active, Deprecated, Archived, Unspecified, Unknown\}.$$

Attribute $ad_6 \equiv version : D \to \mathrm{String}$
Meaning: Specifies the version of definition $d$.
Rule: $version(d)$ returns a string representing the version.

Constraint: $\forall d \in D,\ version(d) \in (\text{String} \cup \{\emptyset\})$.
Example: If $d_6 \in D$ represents a definition with version "2.0", then $version(d_6) = $ "2.0".

Attribute $ad_7 \equiv importRef : D \rightarrow Import \cup \{\emptyset\}$
Meaning: Reference to an external definition for $d$.
Rule: $importRef(d)$ returns an `Import` reference or $\emptyset$ if undefined.
Constraint: $\forall d \in D,\ importRef(d) \in (Import \cup \{\emptyset\})$.
Example: If $d_7 \in D$ uses an external schema, then $importRef(d_7) = $ "http://example.org/schema".

Attribute $ad_8 \equiv properties : D \rightarrow \mathcal{P}(Property)$
Meaning: A set of additional properties for the definition $d$.
Rule: $properties(d)$ returns a set of `Property`.
Constraint: $\forall d \in D,\ properties(d) = \emptyset \vee properties(d) \subseteq Property$.
Example: If $d_8 \in D$ has one property, then $properties(d_8) = \{$"Size"$\}$.

# 3 Formal semantics for Notational Enhancement R3

The notational enhancement R3 of the CMMN-based notational variant (CMMN+) is presented using formal semantics.

First, the basic structure is defined, namely the case model element, which is formally represented as $M = \langle E, A, R \rangle$, where:

- $E$ is the set of all elements that constitute the model,
- $A$ is the set of all attributes of the elements in $E$,
- $R$ is the set of relations or values that can be assigned between a role and an element.

The case model element ($M \in E$) is a special element of the model that represents the entire case process and contains other elements.

By associating an icon (*icon*) to $M$, we define the interpretation function $f_{raci} : M \rightarrow T$, where $T$ represents a specific subset of elements from $E$. The set $T$ includes those elements in $E$ for which role assignments are represented. The function $f_{raci}$ scans the entire model $M$, extracts all eligible elements, and maps them by deriving a representation of the responsibilities associated with roles and elements. The result of $f_{raci}(M)$ is a RACI table that provides a descriptive overview of elements, roles, and their corresponding responsibilities in the model $M$.

From the user's perspective: when the user activates $f_{raci}(M)$ (e.g., clicks the icon), the function evaluates all elements in the set $T$ and for each $t_j \in T$, determines which roles, defined in $V$, participate. For each element $t_j \in T$, the function identifies and represents the responsibilities associated with the roles that participate. Formally, this can be expressed as $icon(M) = f_{raci}(M)$, where $f_{raci}(M)$ is a function that derives a representation of the responsibilities associated with each sub-element of the model $M$.

We define the set of roles as $V = \{v_1, v_2, \ldots, v_n\}$, where each $v_i$ represents an individual role in the model. The set of elements for which role assignments are represented is defined as $T$, where $T \subseteq E$[4] within the model $M$.

The function $rac$ displays all responsibilities in the model and provides a representation of the relationships between roles and elements with the following structure:

$$rac : V \times T \rightharpoonup \{R, A, S, C, I\}$$

where $R$ is Responsible, $A$ is Accountable, $S$ is Support, $C$ is Consulted, and $I$ is Informed. An example of the function is $rac(v_1, t_1) = R$, which means that role $v_1$ is responsible for executing task $t_1$. Role responsibilities over elements can be represented with the semantic function $f_{raci}(M)$, defined as:

$$f_{raci}(M) = \{(v_i, t_j, rac(v_i, t_j)) \mid (v_i, t_j) \in dom(rac)\}$$

This function provides the tuples used to build the RACI table for the elements in $T$ of the model $M$. The icon ($icon$) on $M$ serves as an interactive component that enables access to $f_{raci}(M)$. Semantically, clicking the icon triggers the function, expressed as:

$$click(icon) \Rightarrow f_{raci}(M)$$

From the perspective of CMMN formal semantics, the notational addition of the icon to the case model element for displaying the RACI table is equivalent to adding a function that derives a representation of the responsibilities associated with relevant elements in the model.

For different element types, different subsets of RACI values are represented in the RACI table. Below are elements with their corresponding RACI values, according to $raciVals : ElementType \to P(R, A, S, C, I)$.

- For the user task element $UO$, all RACI values are represented:
  $raciVals(UO) = \{R, A, S, C, I\}$
- For the manual task element $RO$, all RACI values are represented:
  $raciVals(RO) = \{R, A, S, C, I\}$
- For the user event listener element $P$, only the value *Responsible* is represented:
  $raciVals(P) = \{R\}$

These sets indicate which RACI values are represented for each element type in the RACI table.

[4]The set $T$ is a subset of $E$. All elements in $T$ are also in $E$, but $E$ may contain additional elements that are not part of $T$.

# 4 Conclusion

The formal semantics defined in this document provide a rigorous basis for the proposed CMMN-based notational variant (CMMN+). By explicitly defining the structural and behavioral properties of the enhanced constructs, we facilitate precise model interpretation, tool support, and theoretical analysis. They complement the visual notation by ensuring precise interpretation of CMMN+-specific constructs while preserving the original execution semantics of CMMN.