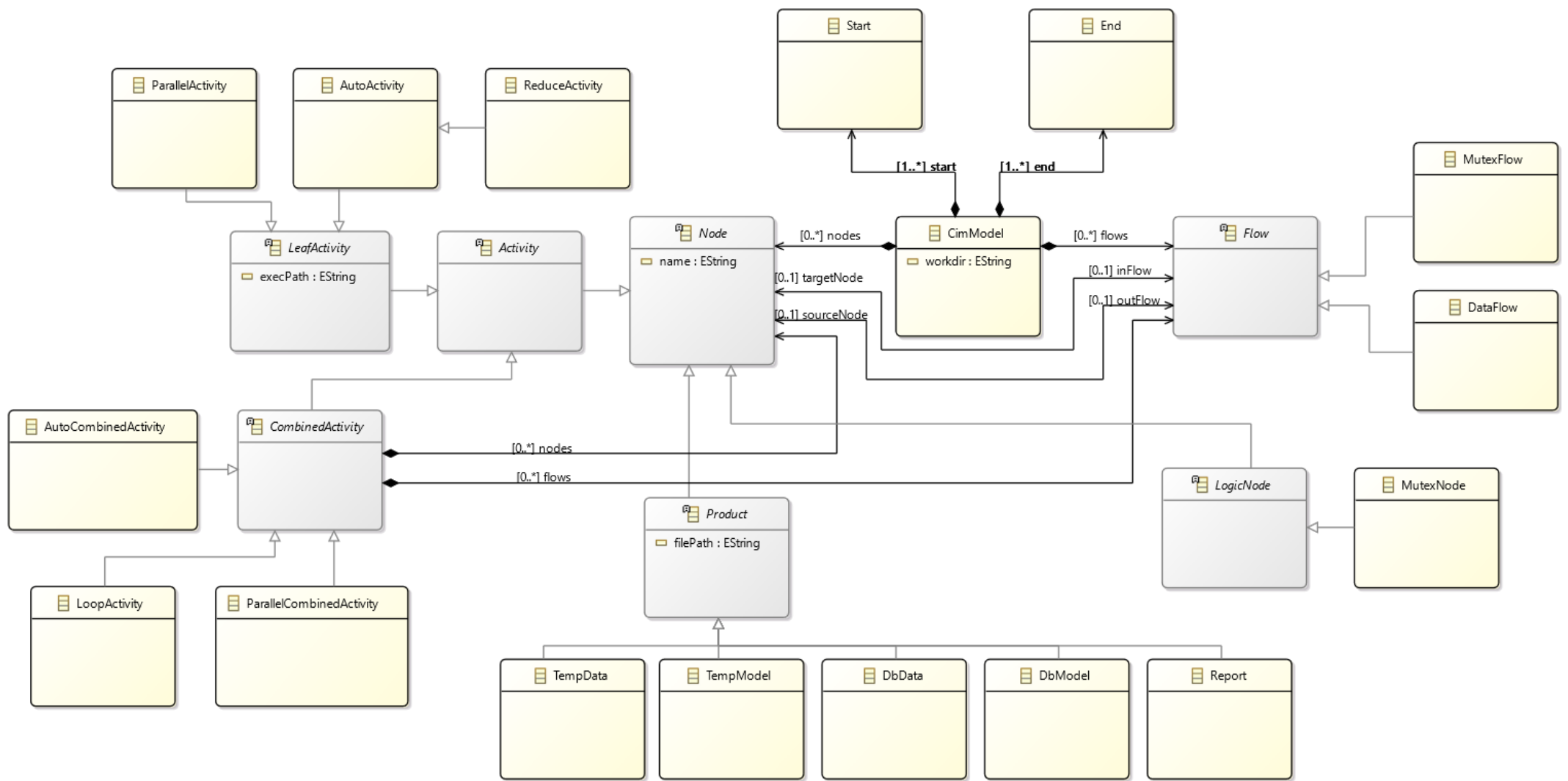


## ParaMDO-CIM DSML Definition

### 1. Grammar definition

Figure 1 shows the metamodel of ParaMDO-CIM.

- 1)  $Activity = \{activity_1, activity_2, \dots, activity_n\}$  represents the activity set in the CIM workflow, including the leaf and non-leaf activities.
- 2)  $Product = \{Product_1, Product_2, \dots, Product_n\}$  represents the data files set in the CIM workflow.
- 3)  $Flow = \{flow_1, flow_2, \dots, flow_n\}$  represents set of connections between nodes.
- 4)  $Logic = \{logic_1, logic_2, \dots, logic_n\}$  represents logic nodes set.
- 5)  $Start = \{start\}$  represents the start of a CIM workflow.
- 6)  $End = \{end\}$  represents the end of a CIM workflow.



**Figure 1 CIM Metamodel**

There are two types of Activity: leaf activities and non-leaf activities, which is called combined activities. Leaf activities refer to non-redivisible CIM activity nodes, while non-leaf activities contain a CIM subprocess. Leaf activities include automated activities, parallel automated activities and reductive activities, while non-leaf activities include parallel combinatorial activities, automated combinatorial activities, and loop activities. The specific activities are shown in Table 1.

**Table 1 CIM Activity Node Semantic**

<b>Element</b>	<b>Description</b>
Auto Activity	Represents one single executable program
Parallel Activity	Represents one single program to be deployed as multi-instance
Parallel Combined Activity	Represents a sequence of programs to be deployed as multi-instance
Auto Combined Activity	Represents a sequence of executable programs
Loop Activity	Represents a sequence of programs to be called for several times
Reduce Activity	Represents a reducer function. Can only be set as the last activity of parallel combined activity

There are 6 types of products, which is described in Table 2. Overall, there are two states of data, available and unavailable. The initial state attribute defines the available state of the product when the CIM workflow first starts. When data is out of the available state, a data available event is sent to all output data streams and when the data is used by the activity.

**Table 2 CIM Product Node Semantic**

<b>Element</b>	<b>Description</b>
Temp Data	Represents one data file generated during execution, which will not be persisted once the execution finishes
Database Data	Represents one data file generated during execution, which will be persisted
Report	Represents one report file generated during execution
Temp Model	Represents one model file generated during execution, which will not be persisted once the execution finishes
Database Model	Represents one model file generated during execution, which will be persisted

Start node and end node are necessary node of a CIM model which marks the start and end of a workflow.

**Table 3 CIM Start / End Description**

<b>Element</b>	<b>Description</b>
Start	Represents the start point of the model
End	Represents the end point of the model

Logic nodes are described in Table 4. Currently there is only one type of logic node which is a mutex node. The mutex node is used to avoid a race condition between activities.

**Table 4 CIM Logic Node Description**

<b>Element</b>	<b>Connections</b>	<b>Description</b>
Mutex	More than one connection to different activities	A mutex represents only one instance of the connected activities could be running at the same time

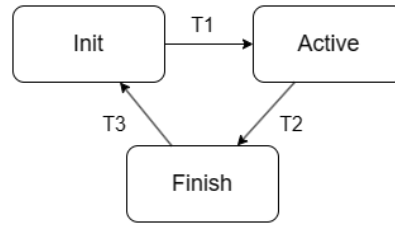
## 2. Semantics definition

The semantics of each modeling element is described as follows.

### 2.1 Basic semantics of activities

Attributes common to all types of activities are the identifier id, the activity name, which is used to store information about the activity.

The activity state consists of three states: initial, active and finish. The state transfer diagram is shown in Figure 2.



**Figure 2 State transfer of activity**

All activity start states are initial states.

➤ **T1 transfer condition** (*transfer condition for shifting from the initial state to the active state*): for an activity, there is an input data stream that is the starting data stream, or all input data are ready.

- (1) When there is a data flow whose starting point is the start node, if the non-leaf parent activity corresponding to the current process is active, the current process can be converted from the initial state to the active state.
- (2) For all the input data flow whose starting point is a data node:  $DataFlow = \{flow_1, flow_2, \dots, flow_n\}$ , we can get the input data of the current activity by the attribute of the starting node of the data flow:  $Data = \{data_1, data_2, \dots, data_n\}$ . For any  $data_i \in Data$ , the following condition should be satisfied: for all the input data flow connections of  $data_i$ :  $PreDataFlow = \{f_1, f_2, \dots, f_m\}$ , for any  $f_j \in PreDataFlow$ ,

the source node of  $f_j$  is activity node, then the state of such activity nodes should be finish.

➤ **T2 transfer condition** (*transfer condition for active to completed state*): completion of the execution of the activity.

- (1) If the current activity is an **Auto Activity**, the current activity needs to be executed to completion.
- (2) If the current activity is a **Reduce Activity**, all parallel instances of the parent activity need to be normalized to a single data output to complete the parent parallel combined activity.
- (3) If the current activity is a **Parallel Activity**, all parallel instances of the activity need to be executed.
- (4) If the current activity is an **Auto Combined Activity**, the subprocesses of the current activity need to be executed, i.e., the end node of the subprocesses needs to be in the end state.
- (5) If the current activity is a **Loop Activity**, the current iteration count should be greater than or equal to the maximum number of iterations of the loop and the loop exit node or end node of the subprocess should be in the end state.
- (6) If the current activity is a **Parallel Combined Activity**, it shall be satisfied that all parallel instances of the sub-process have completed execution, i.e., the end nodes of all instances are in the end state. And if the current activity contains a subsumption activity within the sub-process, the statute activity should also be in the end state.

➤ **T3 transfer condition** (*transfer condition for the completion of the state to the initial state*): the external loop activity starts a new iteration. The looping activity represents the execution of the subprocess in a loop iteration according to the configured maximum number of iterations. Before starting a new iteration, the state of all active nodes in the sub-process will be reset to the initial state.

## 2.2 Leaf activity semantics

Leaf activities contain **Auto Activity**, **Reduce Activity**, and **Parallel Activity**. Leaf activities are the basic unit of activity execution in the CIM process model that cannot be subdivided. Among them, the **Auto Activity** represents the activity of calling external program for automatic execution processing, the **Reduce Activity** is the special automatic activity required for the two-thread generalization within the parallel combination activity, and the **Parallel Activity** represents the special automatic activity for the creation of multiple instances to be executed in parallel.

**Auto Activity** is used to represent executable external programs and contain attributes related to calling external programs. The executable program address attribute represents the path of the external program to be invoked, and the execution parameter attribute contains a set of variables whose values will be obtained from the variable table when the actual invocation takes place, and the program path is spliced together as the actual invocation command. For example, if the program path is `dir/activity.exe`, and the execution parameters are `[a, b, c]`, and the values of `a`, `b`, and `c` in the variable table are 1, 2, and 3 respectively, then the actual invocation command is `dir/activity.exe 1 2 3`.

**Reduce Activity** is inherited from **Auto Activity**, so they have all the properties of **Auto Activity**, and the semantics of the properties inherited from **Auto Activity** are consistent with those of **Auto Activity**. Unlike the **Auto Activity**, the **Reduce Activity** is used to process two-by-two reduce operations on data generated by multiple instances, and can only appear at the end of the sub-processes of the **Parallel Combined Activity**, which needs to be connected to the end node through the data flow. That is, there exists a parallel combinatorial activity `A`, the number of instances of `A` is `n`, where the last activity in the sub-process of `A` is a reductive activity `RA`, and the set of the remaining activities is  $Activity = \{activity_1, activity_2, \dots, activity_m\}$ , where the successor activity of `RA` is  $activity_i$ , which generates  $data_i$  and  $data_i$  is the input data of `RA`. Then, since the actual sub-process has `n` process instances, there are `n` copies of  $data_i$ , denoted as  $data_{1i}, data_{2i}, \dots, data_{ni}$ , then the **Reduce Activity** will select 2



copies of  $data_i$  as inputs, and call the configured program to synthesize a new copy of the merged data of  $data_i$ . After one round of imputation,  $n/2$  copies of data will be obtained, which are written as  $data_{1i}, data_{2i}, \dots, data_{\frac{n}{2}i}$ , the imputation activity will take these  $n/2$  copies of data as new inputs, and call the imputation activity for  $n/4$  times, so as to halve the number of outputs  $data_i$ . The above process is called repeatedly until the final output  $data_i$  is a single copy of data.

**Parallel Activity** is a special kind of automatic activity, based on the inheritance of automatic activity attributes, there are several other attributes such as packing size, fixed data, operating system, and task serial number variable. Among them, the packing size is an integer variable, the value of which indicates that more than one input data file will be packed as a group to create an activity instance for computational processing; the fixed data is a data node type, which indicates that all activity instances need fixed data as input data; the operating system attribute is a string type, which is used to identify the platform on which the program can be executed; and the task serial number variable is a macro variable type, which can be accessed by different activity instances to obtain the current instance in all the instances. Different activity instances can access this macro variable to obtain the serial number of the current instance among all instances.

**Parallel Activity** are only allowed to have a single data input with a path to a folder, the path to other input data must be to a specific file. The activity will get the number of files in the input batch of data based on the data input whose path is a folder and decide the number of parallel task instances to be created based on the packing size. If there are  $m$  files under the input data folder, denoted as  $F = \{f_1, f_2, \dots, f_m\}$ , and the packing size is denoted as  $n$ , then the engine will divide the  $m$  files equally into  $\lceil m/n \rceil$  parts, i.e.,  $\{f_1, f_2, \dots, f_n\}, \{f_{n+1}, f_{n+2}, \dots, f_{2n}\}, \dots, \{f_{(\lceil m/n \rceil - 1)n + 1}, f_{(\lceil m/n \rceil - 1)n + 2}, \dots, f_m\}$ . The engine creates the corresponding  $\lceil m/n \rceil$  activity instances, each of which is processed with one document as input. Each activity instance can access the current task serial number through the task serial number macro variable,  $1, 2, \dots, \lceil m/n \rceil$  respectively.

## 2.3 Auto Combined Activity semantics

**Auto Combined activity** is the most basic non-leaf activities, containing a sub-process that serves the purpose of hierarchical modeling and aids in understanding, and has no substantive special semantics, and is used to represent non-leaf activities that can be automated for complete execution.

## 2.4 Parallel Combined Activity semantics

**Parallel Combined Activity** is a special type of **Auto Combined Activity** that represents the creation of multiple instances of sub-processes to be executed in parallel and may contain internal induction activities. The **Parallel Combined Activity** contains several special attributes such as data mapping method, number of task instances variable, and task number variable. The data mapping method is of enumeration type, which can be chosen from equalization score and full data; the task instance number variable is of integer variable type; and the task sequence number is of macro variable type.

If the value of the number of instances variable is  $n$ , the engine will create  $n$  sub-process instances to be executed in parallel, which are independent of each other and have independent node states. The role of the task serial number variable is the same as the **Parallel Activity**, which is used for each instance to obtain the serial number of the current instance among all instances, and the values that  $n$  instances can obtain through the variable are  $1, 2, \dots, n$ . The data mapping method is to realize the same single procedure with multiple data streams in parallel, and when the data mapping method is equally divided, assume that under the path of the activity's input data There are a total of  $m$  files, denoted as  $F = \{f_1, f_2, \dots, f_m\}$ , then the engine will divide these  $m$  data files into  $n$  equal parts, i.e.

$\{f_1, f_2, \dots, f_{\lfloor m/n \rfloor}\}, \{f_{\lfloor m/n \rfloor + 1}, f_{\lfloor m/n \rfloor + 2}, \dots, f_{2\lfloor m/n \rfloor}\}, \dots, \{f_{(n-1)\lfloor m/n \rfloor + 1}, f_{(n-1)\lfloor m/n \rfloor + 2}, \dots, f_m\}$ , with each activity being able to obtain a separate piece of input data for independent computation processing. When the data mapping method is full-volume data, all  $n$  activity instances can access all  $m$  input data files, and it is necessary for each activity

instance to reasonably select the input data for processing according to the current task instance number, to realize a more complex data mapping method.

## 2.5 Loop Activity semantics

The Loop activity is inherited from the Auto Combined activity and contains a subprocess. The loop activity contains three special properties: loop start variable, loop end variable, and loop count variable. The loop start variable and loop end variable are shaped variables, and the loop count variable is of macro variable type.

The engine will serialize the execution of the subprocess iteratively, at the beginning of each iteration, all nodes in the subprocess will be reset to the initial state, and the loop count macro variable can be accessed in the subprocess to get the current iteration.

## 2.6 Start / End semantics

A start node is a node that identifies the location of the start of a process. The start node has one and only one data stream output, and the end point of the data stream is of type activity, indicating the first activity of the process that begins execution.

An end node is a node that identifies the end position of an activity. The end node has one and only one input data stream, and the current process instance ends when the input data stream is available, i.e., when the state of the node at the start of the input data stream is a completion state.

The start and end nodes are non-redivisible basic nodes with two states: initial and end. The default state of all three nodes at the beginning is initial.

- Initial to completion state transfer conditions:
  - 1) Start node: the parent activity belonging to it changes from initial state to active state
  - 2) End node: for the only input data flow DataFlow of the end node, the state of the DataFlow start node is completed.
- The completion of the state transfer to the initial state is subject to any of the following conditions:
  - 1) The parent activity re-shifts from the completion state to the initial state

- 2) The parent activity is a recurring activity and a new round of iteration begins

## 2.7 Mutex semantics

A mutex node is used to control the mutex execution of an activity and has a mutex id attribute. The mutex id is a string expression type. Mutex nodes can be connected to several mutex streams, and the other end of the mutex streams is connected to the activity node, remember that mutex node mutex through the mutex streams connected to the set of activities as  $Activity = \{activity_1, activity_2, \dots, activity_n\}$ , then all the activities in the *Activity* collection need to compete for the mutex lock represented by the mutex before execution. Mutex nodes are stateless nodes, no need to consider the state transfer condition.

For example, in the root activity Root process, there exists a parallel combination activity *pa1*, with a number of parallel instances of *pa1* is *m*, and a task number variable named *var1*. There exists a parallel combination activity *pa2* in the sub-process of *pa1*, with a number of parallel instances of *pa2* is *n*, and a task number variable named *var2*. The *pa2* sub-process contains two activities, *a1* and *a2*, and there exists a mutual exclusion node, *mutex1*. connected to *a1* and *a2* via a mutex stream. Then there are  $m * n$  activities *a1* and *a2*, denoted as

$$\{a1_{1,1}, a1_{2,1}, \dots, a1_{m,1}, a1_{1,2}, \dots, a1_{m,2}, \dots, a1_{m,n}\}$$

and

$$\{a2_{1,1}, a2_{2,1}, \dots, a2_{m,1}, a2_{1,2}, \dots, a2_{m,2}, \dots, a2_{m,n}\}$$

Their mutual exclusion relationship is controlled by the mutual exclusion id expression of mutual exclusion node *mutex1*. When the content of the mutex id expression of the mutex node *mutex1* is *test@var1#*, then the engine generates *m* instances of mutex lock ids  $MutexID = \{test@var1\#1, test@var1\#2, \dots, test@var1\#m\}$ , for  $\forall mutexid_i \in MutexID$ , the set of activities that need to compete for *mutexid<sub>i</sub>* corresponding to the mutex lock is  $MutexActivity_i = \{a1_{i,1}, a1_{i,2}, \dots, a1_{i,n}\} \cup \{a2_{i,1}, a2_{i,2}, \dots, a2_{i,n}\}$ .

## 2.8 Product node semantics

Product nodes contain five specific types: temporary data, database data, reports, temporary models, and database models. These five types of data nodes have identical execution semantics for the process engine, and the difference between them is mainly for the business to distinguish between different types of data files and serve to increase the readability of the process model. The data nodes all contain a path attribute of string type. The path can be configured as a file path or a folder path in the system. When the path is configured as a folder path, the data represents the collection of all files in that folder. Temporary data contains a Boolean attribute of null data. When this attribute is true, the data node does not represent any data files and only serves to perform in-process messaging.

The data node contains two states: unavailable and available. The unavailable state means that the data is not ready to be used as input by the activity. The available state means that the data file is ready to be used by an activity or logical node along the output data stream.

Before a process can be executed, all data nodes need to be initialized. That is, the data nodes under the configured path where the file is not empty and no input data flow exists are set to the completion state, and the remaining data nodes are set to the available state. When the start node of the input data stream of a data node is in the completion state, the data node state is set to the completion state.

## 2.9 Variable semantics

Variables consist of five types of variables: string variables, integer variables, floating point variables, array type variables, and macro variables. The first four types are user-defined and initialized variables with initial values that can be read by all activities and logic nodes in the process, and can be modified by activities in the process. Macro variables do not have an initial value and are maintained by the engine. The activities and logical nodes in the process can only read macro variables and have no permission to modify them. Macro variables are mainly used to identify the loop count of cyclic activities and the task number of parallel automatic/parallel combined activities.

Variables are not visual nodes in the CIM process view and therefore have no state and need not be considered for state transfer conditions.






### 3. Graphical Modeling Definition

An example of the node object diagram in the CIM view is shown in Table 5, and an example of the connectivity diagram in the CIM view is shown in Table 6.

**Table 5 CIM Node Icons**

Node	Icon	Node	Icon
Activity		Temp Data	
Auto Activity		Database Data	
Parallel Activity		Temp Model	
Combined Activity		Database Model	
Auto Combined Activity		Report	
Loop Activity		Mutex	
Parallel Combined Activity		Start	
Reduce Activity		End	

**Table 6 CIM Connection Icons**

<b>Connection</b>	<b>Icon</b>	<b>Connection</b>	<b>Icon</b>
DataFlow		TrueFlow	
RefFlow		MutexFlow	
FalseFlow			



## 4. Model Checking Rules

The modeling process needs to check the model to ensure that the model definition satisfies the structural integrity rules of the modeling language, which lays the foundation for the correctness of the model analysis and model transformation.

The model parameter integrity check is used to check whether there are syntax errors in the parameters of the elements in the model, such as whether there is a parameter file whose path field is empty.

Table 7 defines the parameter integrity checking rules for the model. It is used to check whether the model parameters are complete or not. Table 8 defines the rules for structural integrity checking of the model, which is used to check whether the structure of the model is complete in the modeling syntax relations, e.g., to check whether there is a process view with no start node.

**Table 7 Parameter Integrity Checking Rules**

No.	Checking Rule
1	An element exists and an empty name field reports an error.
2	The root activity of the modeling project exists, and an empty workdir field reports an error.
3	An error is reported if there is a stored data and the path field is empty.
4	An error is reported if there is a loop activity with null loop start variable, end variable, and loop count variable fields.
5	An error is reported if there is a parallel combination activity or a parallel automation activity with empty task instance count variable and task serial number variable fields.
6	An error is reported if a conditional branch node exists and its Execution Condition field is empty.
7	An error is reported if there is a Mutually Exclusive node with an empty Mutually Exclusive id field.

**Table 8 Structural Integrity Checking Rules**

No.	Checking Rule
1	View exists, there is no start node or there is no end node, then an error is reported.
2	Activity exists with no reachable path that can be reached from the start node or no reachable path that can reach the end node.
3	Presence of Combination Activity/Automatic Combination Activity/Cyclic Activity with no leaf activity reports an error.