Translation Rules

High Level

section *CircusBNFEncoding* **parents** *standard_toolkit*

$[Predicate, N, Expression, Paragraph, SchemaExp, Declaration]$

$$Command ::= spec\langle\langle \operatorname{seq} N \times Predicate \times Predicate \rangle\rangle \mid equals\langle\langle N \times \operatorname{seq} Expression \rangle\rangle$$

$$CParameter ::= shriek\langle\langle N \rangle\rangle \mid shriekRestrict\langle\langle N \times Predicate \rangle\rangle \mid bang\langle\langle Expression \rangle\rangle \mid dotParam\langle\langle Expression \rangle\rangle$$

$$Communication == N \times \operatorname{seq} CParameter$$

$$CSExpression ::= cs\langle\langle \operatorname{seq} N \rangle\rangle \mid csName\langle\langle N \rangle\rangle \mid union\langle\langle CSExpression \times CSExpression \rangle\rangle \mid intersect\langle\langle CSExpression \times CSExpression \rangle\rangle \mid subtract\langle\langle CSExpression \times CSExpression \rangle\rangle$$

$$Action ::= actSe\langle\langle SchemaExp \rangle\rangle \mid com\langle\langle Command \rangle\rangle \mid skip \mid stop \mid chaos \mid prefixExp\langle\langle Communication \times Action \rangle\rangle \mid guard\langle\langle Predicate \times Action \rangle\rangle \mid seqExp\langle\langle Action \times Action \rangle\rangle \mid extChoice\langle\langle Action \times Action \rangle\rangle \mid intChoice\langle\langle Action \times Action \rangle\rangle \mid actPar\langle\langle Action \times CSExpression \times Action \rangle\rangle \mid actInter\langle\langle Action \times Action \rangle\rangle \mid actHide\langle\langle Action \times CSExpression \rangle\rangle \mid mu\langle\langle N \times Action \rangle\rangle \mid actParam\langle\langle Declaration \times Action \rangle\rangle \mid actInst\langle\langle Action \times \operatorname{seq} Expression \rangle\rangle$$

$$GuardedAction ::= thenAct\langle\langle Predicate \times Action \rangle\rangle \mid thenActComp\langle\langle Predicate \times Action \times GuardedAction \rangle\rangle$$

$$PParagraph ::= pPar\langle\langle Paragraph \rangle\rangle \mid def\langle\langle N \times Action \rangle\rangle$$

$$Process ::= proc\langle\langle \operatorname{seq} PParagraph \times Action \rangle\rangle \mid procName\langle\langle N \rangle\rangle \mid procSeq\langle\langle Process \times Process \rangle\rangle \mid procExtChoice\langle\langle Process \times Process \rangle\rangle \mid procIntChoice\langle\langle Process \times Process \rangle\rangle \mid procPar\langle\langle Process \times CSExpression \times Process \rangle\rangle \mid procInter\langle\langle Process \times Process \rangle\rangle \mid procHide\langle\langle Process \times CSExpression \rangle\rangle \mid rename\langle\langle Process \times \operatorname{seq} N \times \operatorname{seq} N \rangle\rangle \mid procParam\langle\langle Declaration \times Process \rangle\rangle \mid procInstP\langle\langle Process \times \operatorname{seq} Expression \rangle\rangle \mid procGeneric\langle\langle \operatorname{seq} N \times Process \rangle\rangle \mid procInstG\langle\langle Process \times \operatorname{seq} Expression \rangle\rangle$$

$$ProcDefinition ::= pd\langle\langle N \times Process \rangle\rangle$$

$$ChanSetDefinition ::= csdName\langle\langle N \times CSExpression \rangle\rangle$$

$$SCDeclaration ::= chanName\langle\langle \mathrm{seq}\, N \rangle\rangle \mid chanNameWithType\langle\langle \mathrm{seq}\, N \times Expression \rangle\rangle \mid scSe\langle\langle SchemaExp \rangle\rangle$$

$$CDeclaration ::= scDecl\langle\langle SCDeclaration \rangle\rangle \mid multiDecl\langle\langle SCDeclaration \times CDeclaration \rangle\rangle$$

$$ChannelDefinition == CDeclaration$$

$$CircusParagraph ::= para\langle\langle Paragraph \rangle\rangle \mid chanDef\langle\langle ChannelDefinition \rangle\rangle \mid chanSetDef\langle\langle ChanSetDefinition \rangle\rangle \mid procDef\langle\langle ProcDefinition \rangle\rangle$$

$$CircusProgram == \mathrm{seq}\, CircusParagraph$$

section *SCJBNFEncoding* **parents** *standard_toolkit*

$[MethodBody, ClassBodyDeclaration, Identifier, MethodDeclaration, Long]$

$Run == MethodBody$
$ManagedThreadClassBody == Run \times \text{seq}\ ClassBodyDeclaration$
$ManagedThread == Identifier \times ManagedThreadClassBody$

$HandleAsyncEvent == MethodBody$
$HandleAsyncLongEvent == Long \times MethodBody$
$EventHandlerClassBody == HandleAsyncEvent \times \text{seq}\ ClassBodyDeclaration$
$OneShotEventHandler == Identifier \times EventHandlerClassBody$
$LongEventHandlerClassBody == HandleAsyncLongEvent \times \text{seq}\ ClassBodyDeclaration$
$AperiodicEventHandler ::= apehType\langle\langle Identifier \times EventHandlerClassBody\rangle\rangle \mid$
$\quad aplehType\langle\langle Identifier \times LongEventHandlerClassBody\rangle\rangle$
$PeriodicEventHandler == Identifier \times EventHandlerClassBody$
$EventHandler ::= pehDecl\langle\langle PeriodicEventHandler\rangle\rangle \mid$
$\quad apehDecl\langle\langle AperiodicEventHandler\rangle\rangle \mid$
$\quad osehDecl\langle\langle OneShotEventHandler\rangle\rangle$

$GetNextMission == MethodBody$
$MissionSequencerClassBody == GetNextMission \times \text{seq}\ ClassBodyDeclaration$
$MissionSequencer == Identifier \times MissionSequencerClassBody$

$NestedMissionSequencer == MissionSequencer$

$SchedulableObject ::= handler\langle\langle EventHandler\rangle\rangle \mid$
$\quad mt\langle\langle ManagedThread\rangle\rangle \mid$
$\quad nms\langle\langle NestedMissionSequencer\rangle\rangle$

$Cleanup == MethodBody$
$Initialize == MethodBody$
$MissionClassBody == Initialize \times Cleanup \times \text{seq}\ ClassBodyDeclaration$
$Mission == Identifier \times MissionClassBody$

$Cluster == Mission \times \text{seq}\ SchedulableObject$
$Tier == \text{seq}\ Cluster$

$$TopLevelMissionSequencer ::= NoSequencer \mid tlms\langle\langle MissionSequencer\rangle\rangle$$

$$ImmortalMemorySize == MethodDeclaration$$
$$InitializeApplication == MethodBody$$
$$GetSequencer == MethodBody$$
$$SafeletClassBody == InitializeApplication \times GetSequencer \times ImmortalMemorySize \times \text{seq}\, ClassBodyDeclaration$$
$$Safelet == Identifier \times SafeletClassBody$$

$$SCJProgram == Safelet \times TopLevelMissionSequencer \times \text{seq}\, Tier$$

section *TransSCJProg* **parents** *standard_toolkit*, *SCJBNFEncoding*, *CircusBNFEncoding*

$$controlTierSync : CSExpression$$
$$Tier0 : N$$
$$MissionIds : \text{seq } CircusParagraph$$
$$SchedulableIds : \text{seq } CircusParagraph$$

$$ServicesChan : \text{seq } CircusParagraph$$
$$GlobalTypes : \text{seq } CircusParagraph$$
$$JTime : \text{seq } CircusParagraph$$
$$PrimitiveTypes : \text{seq } CircusParagraph$$
$$Priority : \text{seq } CircusParagraph$$
$$PriorityQueue : FrameworkChan : MissionId : \text{seq } CircusParagraph$$
$$SchedulableId : \text{seq } CircusParagraph$$

$$ObjectFW : Process$$
$$ObjectChan : \text{seq } CircusParagraph$$
$$ObjectFWChan : \text{seq } CircusParagraph$$
$$ObjectMethChan : \text{seq } CircusParagraph$$
$$ThreadFW : Process$$
$$ThreadChan : \text{seq } CircusParagraph$$
$$ThreadFWChan : \text{seq } CircusParagraph$$
$$ThreadMethChan : \text{seq } CircusParagraph$$

$$SafeletFW : Process$$
$$SafeletFWChan : \text{seq } CircusParagraph$$
$$SafeletChan : \text{seq } CircusParagraph$$
$$SafeletMethChan : \text{seq } CircusParagraph$$

$$TopLevelMissionSequencerFW : Process$$
$$TopLevelMissionSequencerChan : \text{seq } CircusParagraph$$
$$TopLevelMissionSequencerFWChan : \text{seq } CircusParagraph$$

$$MissionSequencerChan : \text{seq } CircusParagraph$$
$$MissionSequencerFWChan : \text{seq } CircusParagraph$$
$$MissionSequencerMethChan : \text{seq } CircusParagraph$$

$$MissionFW : Process$$
$$MissionChan : \text{seq } CircusParagraph$$
$$MissionFWChan : \text{seq } CircusParagraph$$
$$MissionMethChan : \text{seq } CircusParagraph$$

$SchedulableChan$: seq $CircusParagraph$
$SchedulableMethChan$: seq $CircusParagraph$
$SchedulableFWChan$: seq $CircusParagraph$
$HandlerChan$: seq $CircusParagraph$
$HandlerFWChan$: seq $CircusParagraph$
$HandlerMethChan$: seq $CircusParagraph$

$PeriodicEventHandlerChan$: seq $CircusParagraph$
$PeriodicEventHandlerFW$: $Process$
$PeriodicEventHandlerFWChan$: seq $CircusParagraph$
$PeriodicParameters$: seq $CircusParagraph$

$AperiodicEventHandlerChan$: seq $CircusParagraph$
$AperiodicEventHandlerFW$: $Process$
$AperiodicLongEventHandlerMethChan$: seq $CircusParagraph$
$AperiodicParameters$: seq $CircusParagraph$

$OneShotEventHandlerChan$: seq $CircusParagraph$
$OneShotEventHandlerFW$: $Process$
$OneShotEventHandlerFWChan$: seq $CircusParagraph$
$OneShotEventHandlerMethChan$: seq $CircusParagraph$

$SchedulableMissionSequencerFW$: $Process$
$SchedulableMissionSequencerChan$: seq $CircusParagraph$
$SchedulableMissionSequencerFWChan$

$ManagedThreadFW$: $Process$
$ManagedThreadChan$: seq $CircusParagraph$
$ManagedThreadFWChan$: seq $CircusParagraph$
$ManagedThreadMethChan$: seq $CircusParagraph$

$framework : CircusProgram$

$framework = ServicesChan \frown GlobalTypes \frown JTime \frown PrimitiveTypes \frown Priority \frown$
$PriorityQueue \frown FrameworkChan \frown MissionId \frown SchedulableId \frown ObjectFW \frown$
$ObjectChan \frown ObjectFWChan \frown ObjectMethChanThreadFW \frown ThreadChan \frown$
$ThreadFWChan \frown ThreadMethChan \frown SafeletFW \frown SafeletFWChan \frown$
$SafeletChan \frown SafeletMethChan \frown TopLevelMissionSequencerFW \frown$
$TopLevelMissionSequencerChan \frown TopLevelMissionSequencerFWChan \frown$
$MissionSequencerChan \frown MissionSequencerFWChan \frown MissionSequencerMethChan \frown$
$MissionFW \frown MissionChan \frown MissionFWChan \frown MissionMethChan \frown$
$SchedulableChan \frown SchedulableMethChan \frown SchedulableFWChan \frown$
$HandlerChan \frown HandlerFWChan \frown HandlerMethChan \frown PeriodicEventHandlerChan \frown$
$PeriodicEventHandlerFW \frown PeriodicEventHandlerFWChan \frown PeriodicParameters \frown$
$AperiodicEventHandlerChan \frown AperiodicEventHandlerFW \frown$
$AperiodicLongEventHandlerMethChan \frown AperiodicParameters \frown$
$OneShotEventHandlerChan \frown OneShotEventHandlerFW \frown$
$OneShotEventHandlerFWChan \frown OneShotEventHandlerMethChan \frown$
$SchedulableMissionSequencerFW \frown SchedulableMissionSequencerChan \frown$
$SchedulableMissionSequencerFWChan \frown ManagedThreadFW \frown ManagedThreadChan \frown$
$ManagedThreadFWChan \frown ManagedThreadMethChan$

$fwProcName : N \pfun N$
$appProcName : N \pfun N$
$mcbProcName : N \pfun N$
$lockProcName : N \pfun N$

$distinct\langle ran\ fwProcName, ran\ appProcName, ran\ mcbProcName, ran\ lockProcName\rangle$

$MakeProcDef : \mathrm{seq}\ Process \pfun \mathrm{seq}\ ProcDef$

$\forall\ procs : \mathrm{seq}\ Process \bullet$
$MakeProcDef(procs) = \langle ProcDef(head\ procs)\rangle \frown MakeProcDef(tail\ procs)$

$GenerateTierProcs : \mathrm{seq}\ Tier \pfun Process$

$GenerateTiers : \mathrm{seq}\ Tier \pfun \mathrm{seq}\ Process$

$GetEnv : SCJProgram \pfun Safelet \times TopLevelMissionSequencer \times \mathrm{seq}\ Tier$

$\forall\ scj : SCJProgram \bullet$
$\exists\ s : Safelet;\ tlms : TopLevelMissionSequencer;\ tiers : \mathrm{seq}\ Tier \bullet$
$GetEnv(scj) = (s, tlms, tiers)$

$$GenerateFWProcs : Safelet \times TopLevelMissionSequencer \times \operatorname{seq} Tier \pfun \operatorname{seq} Process$$

$$\begin{lstlisting}
∀ s : Safelet; tlms : TopLevelMissionSequencer; tiers : seq Tier
    ∃ fwProc : Process; controlTierProc : Process; tierProc : Process; tierProcs : seq Process |
        fwProc = procPar(controlTierProc, controlTierSync, tierProc) ∧
        controlTierProc =
            procPar(safeletFW, controlTierSync, topLevelMissionSequencerFW) ∧
        tierProc = GenerateTierProc(tiers)
        tierProcs = GenerateTiers(tiers) •
        GenerateFWProc(scj) = fwProc ⌢ tierProcs
\end{lstlisting}$$

$$GenerateAppProc : Safelet \times TopLevelMissionSequencer \times \operatorname{seq} Tier \pfun Process$$

$$GenerateMCBProc : Safelet \times TopLevelMissionSequencer \times \operatorname{seq} Tier \pfun Process$$

$$GenerateLockProc : Safelet \times TopLevelMissionSequencer \times \operatorname{seq} Tier \pfun \operatorname{seq} Process$$

$$TransClasses : Safelet \times TopLevelMissionSequencer \times \operatorname{seq} Tier \pfun CircusProgram$$

$$TransSCJProg : Identifier \times SCJProgram \pfun CircusProgram$$

$$\begin{lstlisting}
∀ scjProg : SCJProgram; name : Identifier •
    ∃ app : CircusProgram;
        program : CircusProgram; n : N; p : Process;
        appComms : CSExpression; mcbComms : CSExpression; lockComms : CSExpression
        fwProc : Process; appProc : Process; lockProc : Process; mcbProc : Process |
    app = TransClasses(GetEnv(scjProg)) ∧
    fwProc = GenerateFWProc(GetEnv(scjProg)) ∧
    appProc = GenerateAppProc(GetEnv(scjProg)) ∧
    mcbProc = GenerateMCBProc(GetEnv(scjProg)) ∧
    lockProc = GenerateLockProc(GetEnv(scjProg)) ∧
    program = procDef(pd(n,
        procHide(procPar(
            procHide(procPar(
                procHide(procPar(fwProc, appComms, appProc), appComms)
            , mcbComms, mcbProc), mcbComms),
        lockComms, lockProc), lockComms))) •
    TransSCJProg(name, scjProg) =
        framework ⌢ ⟨procDef(pd(fwProcName(n), fwProc))⟩⌢
            app ⌢ ⟨procDef(pd(appProcName(n), appProc))⟩⌢
            ⟨procDef(pd(mcbProcName(n), mcbProc))⟩⌢
            ⟨procDef(pd(lockProcName(n), lockProc))⟩⌢
            ⟨program⟩
\end{lstlisting}$$

Low Level

- $Method : MethodDeclaration \pfun (Name, Params, ReturnType, Body)$: translates an active application method into a *Circus* action
- $DataMethod : MethodDeclaration \pfun$: translates data methods into an *OhCircus* method
- $MethodBody : Block \pfun \seq CircExpression$: translates a Java block, for example a method body
- $Registers : Block \pfun \seq Name$: extracts the Names of the schedulables registered in a Java block
- $Returns : Block \pfun \seq Name$: extracts the Names of the variables retuned in a Java block
- $Variable : (Name, Type, InitExpression) \pfun (CircName, CircType, CircExpression)$: translates a variable
- $Parameters : (Name, Params, ReturnType, Body) \pfun \seq CircParam$: translates a list of method parameters
- $[\![Name]\!]_{name}$: translates the *name* to a Z identifier
- $[\![varType]\!]_{type}$: translates types
- $[\![expr]\!]_{expression}$: translates expressions

Auxiliary Functions

- *IdOf*(*name*): yields the identifier of a component called *name*
- *ObjectIdOf*(*name*): yields the identifier of the *Object* process of a component called *name*
- *ThreadIdOf*(*name*): yields the identifier of the *Thread* process of a component called *name*
- *MethodName*(*method*): yields the method name of *method*
- *MethodsOf*(*name*) : yeilds a sequence of methods from the class *name*

Pattern Matching Rules

Safelet

```
1  public class Identifier implements Safelet
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     initializeApplication
10
11    getSequencer
12
13    AppMeth_1
14    ...
15    AppMeth_n
16 }
```

process $[\![Identifier]\!]_{Name}$ $App \mathrel{\widehat=}$ $[\![[\![ConstructorDeclaration]\!]_{Method}]\!]_{Parameters}$ **begin**

State

$this : \text{ref } [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new } [\![Identifier]\!]_{name}\ Class()$

$$InitializeApplication \mathrel{\widehat=} \left(\begin{array}{l} initializeApplicationCall \longrightarrow \\ [\![[\![InitializeApplication]\!]_{Method}]\!]_{MethBody} \\ initializeApplicationRet \longrightarrow \\ \textbf{Skip} \end{array} \right)$$

$$GetSequencer \mathrel{\widehat=} \left(\begin{array}{l} getSequencerCall \longrightarrow \\ getSequencerRet\,!\ [\![GetSequencer]\!]_{Returns} \longrightarrow \\ \textbf{Skip} \end{array} \right)$$

$[\![AppMeth\_1]\!]_{Method}$

...

$[\![AppMeth\_n]\!]_{Method}$

$$
\begin{array}{l}
Methods \mathrel{\widehat{=}} \\
\left(\begin{array}{l}
GetSequencer \\
\Box \\
InitializeApplication \\
\Box \\
MethName(AppMeth\_1) \\
\Box \\
\ldots \\
MethName(AppMeth\_n) \\
\ldots
\end{array}\right) \; ; \; Methods
\end{array}
$$

- $(Init \; ; \; Methods) \;\; \triangle(end\_safelet\_app \longrightarrow \mathbf{Skip})$

end

Mission Sequencer

```
1  public class Identifier extends MissionSequencer
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     getNextMission
10
11    AppMeth_1
12    ...
13    AppMeth_n
14 }
```

process $[\![Identifier]\!]_{Name}\ App \mathrel{\widehat=} [\![\ [\![ConstructorDeclaration]\!]_{Method}]\!]_{Parameters}$ **begin**

State

$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$

$GetNextMission \mathrel{\widehat=} \textbf{var}\ ret : MissionID \bullet$

$$\begin{pmatrix} getNextMissionCall\,.\,IdOf(Identifier) \longrightarrow \\ ret := this\,.\,getNextMission(); \\ getNextMissionRet\,.\,IdOf(Identifier)\,!\,ret \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$[\![AppMeth\_1]\!]_{Method}$

$\ldots$

$[\![AppMeth\_n]\!]_{Method}$

$Methods \mathrel{\widehat=}$

$$\begin{pmatrix} GetNextMission \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} ;\ Methods$$

$\bullet\ (Init\ ;\ Methods)\ \ \triangle(end\_sequencer\_app\,.\,IdOf(Identifier) \longrightarrow \textbf{Skip})$

end

Mission

```
1  public class Identifier extends Mission
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     initialize
10
11    cleanUp
12
13    AppMeth_1
14    ...
15    AppMeth_n
16 }
```

process $[\![Indentifier]\!]\ App \mathrel{\widehat=} [\![\ [\![\ ConstructorDeclaration\ ]\!]_{Method}\ ]\!]_{Parameters}$ **begin**

State

$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$

$$InitializePhase \mathrel{\widehat=} \begin{pmatrix} initializeCall\,.\,IdOf(Indentifier) \longrightarrow \\ [\![initialize]\!]_{Registers}\ initializeRet\,.\,IdOf(Indentifier) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$CleanupPhase \mathrel{\widehat=} \begin{pmatrix} cleanupMissionCall\,.\,IdOf(Indentifier) \longrightarrow \\ cleanupMissionRet\,.\,IdOf(Indentifier)\,!\,\textbf{True} \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$[\![AppMeth\_1]\!]_{Method}$

...

$[\![AppMeth\_n]\!]_{Method}$

$$Methods \mathrel{\widehat=} \begin{pmatrix} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} ;\ Methods$$

• $(Init\ ;\ Methods)\ \ \triangle(end\_mission\_app\ .\ IdOf(Identifier) \longrightarrow \textbf{Skip}$

end

Handlers

```
1  class Identifier extends HandlerType
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     handleAsyncEvent
10
11    AppMeth_1
12    ...
13    AppMeth_n
14 }
```

process $[\![PName]\!]\ App \mathrel{\widehat=} [\![\ [\![\ ConstructorDeclaration\ ]\!]_{Method}\ ]\!]_{Parameters}$ **begin**

State

$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$

$handleAsyncEvent \mathrel{\widehat=}$
$\begin{pmatrix} handleAsyncEventCall\,.\,IdOf(PName) \longrightarrow \\ [\![\ [\![\ HandleAsyncBody\ ]\!]_{Method}\ ]\!]_{MethBody}; \\ handleAsyncEventRet\,.\,IdOf(PName) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$

$[\![AppMeth\_1]\!]_{Method}$

$\ldots$

$[\![AppMeth\_n]\!]_{Method}$

$Methods \mathrel{\widehat=}$
$\begin{pmatrix} handleAsyncEvent \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix}$; $Methods$

• $(Init\ ;\ Methods)\ \ \triangle(end\_\ [\![\ HandlerTypeIdOf(PName)\ ]\!] \longrightarrow \textbf{Skip})$

end

Managed Thread

```
1  public class Identifier extends ManagedThread
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     run
10
11    AppMeth_1
12    ...
13    AppMeth_n
14 }
```

$$\textbf{process}\ [\![PName]\!]\ App \mathrel{\widehat=} [\![\ [\![\ ConstructorDeclaration\ ]\!]_{Method}\ ]\!]_{Parameters}\ \textbf{begin}$$

State

$$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$$

$$\textbf{state}\ State$$

Init

$$State'$$
$$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$$

$$Run \mathrel{\widehat=} \begin{pmatrix} runCall\,.\,IdOf(PName) \longrightarrow \\ [\![\ [\![\ run\ ]\!]_{Method}\ ]\!]_{MethBody}; \\ runRet\,.\,IfOf(PName) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$[\![AppMeth\_1]\!]_{Method}$$

$$\ldots$$

$$[\![AppMeth\_n]\!]_{Method}$$

$$Methods \mathrel{\widehat=} \begin{pmatrix} Run \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix};\ Methods$$

$$\bullet\ (Init\ ;\ Methods)\ \ \triangle(end\_managedThread\_app\,.\,IdOf(PName) \longrightarrow \textbf{Skip})$$

$$\textbf{end}$$

Data Class

class $[\![PName]\!]_{name}$ *Class* $\widehat{=}$ **begin**

state *State*

$[\![VarName]\!]_{name} : [\![VarType]\!]_{type}$

state *State*

initial *Init*

$State'$

$[\![VarName]\!]'_{name} = [\![VarInit]\!]_{expression}$

$[\![DataMeth1]\!]_{dataMeth}$
$[\![DataMeth2]\!]_{dataMeth}$
...

- **Skip**

end