

Translation Rules

High Level

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

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

Command ::= *spec*⟨⟨seq *N* × *Predicate* × *Predicate*⟩⟩ | *equals*⟨⟨*N* × seq *Expression*⟩⟩

CParameter ::= *shriek*⟨⟨*N*⟩⟩ | *shriekRestrict*⟨⟨*N* × *Predicate*⟩⟩ | *bang*⟨⟨*Expression*⟩⟩ |
dotParam⟨⟨*Expression*⟩⟩

Communication == *N* × seq *CParameter*

CSEExpression ::= *cs*⟨⟨seq *N*⟩⟩ | *csName*⟨⟨*N*⟩⟩ |
union⟨⟨*CSEExpression* × *CSEExpression*⟩⟩ |
intersect⟨⟨*CSEExpression* × *CSEExpression*⟩⟩ |
subtract⟨⟨*CSEExpression* × *CSEExpression*⟩⟩

Action ::= *actSe*⟨⟨*SchemaExp*⟩⟩ | *com*⟨⟨*Command*⟩⟩ | *skip* | *stop* | *chaos* |
prefixExp⟨⟨*Communication* × *Action*⟩⟩ |
guard⟨⟨*Predicate* × *Action*⟩⟩ | *seqExp*⟨⟨*Action* × *Action*⟩⟩ |
extChoice⟨⟨*Action* × *Action*⟩⟩ | *intChoice*⟨⟨*Action* × *Action*⟩⟩ |
actPar⟨⟨*Action* × *CSEExpression* × *Action*⟩⟩ | *actInter*⟨⟨*Action* × *Action*⟩⟩ |
actHide⟨⟨*Action* × *CSEExpression*⟩⟩ | *mu*⟨⟨*N* × *Action*⟩⟩ | *actParam*⟨⟨*Declaration* × *Action*⟩⟩ |
actInst⟨⟨*Action* × seq *Expression*⟩⟩

GuardedAction ::= *thenAct*⟨⟨*Predicate* × *Action*⟩⟩ |
thenActComp⟨⟨*Predicate* × *Action* × *GuardedAction*⟩⟩

PParagraph ::= *pPar*⟨⟨*Paragraph*⟩⟩ | *def*⟨⟨*N* × *Action*⟩⟩

Process ::= *proc*⟨⟨seq *PParagraph* × *Action*⟩⟩ | *procName*⟨⟨*N*⟩⟩ |
procSeq⟨⟨*Process* × *Process*⟩⟩ | *procExtChoice*⟨⟨*Process* × *Process*⟩⟩ |
procIntChoice⟨⟨*Process* × *Process*⟩⟩ | *procPar*⟨⟨*Process* × *CSEExpression* × *Process*⟩⟩ |
procInter⟨⟨*Process* × *Process*⟩⟩ | *procHide*⟨⟨*Process* × *CSEExpression*⟩⟩ |
rename⟨⟨*Process* × seq *N* × seq *N*⟩⟩ | *procParam*⟨⟨*Declaration* × *Process*⟩⟩ |
procInstP⟨⟨*Process* × seq *Expression*⟩⟩ | *procGeneric*⟨⟨seq *N* × *Process*⟩⟩ |
procInstG⟨⟨*Process* × seq *Expression*⟩⟩

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

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

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

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

$ChannelDefinition == CDeclaration$

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

$CircusProgram == seq CircusParagraph$

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

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

$$\begin{aligned} Run &== MethodBody \\ ManagedThreadClassBody &== Run \times \text{seq } ClassBodyDeclaration \\ ManagedThread &== Identifier \times ManagedThreadClassBody \end{aligned}$$
$$\begin{aligned}
\textit{HandleAsyncEvent} &== \textit{MethodBody} \\
\textit{HandleAsyncLongEvent} &== \textit{Long} \times \textit{MethodBody} \\
\textit{EventHandlerClassBody} &== \textit{HandleAsyncEvent} \times \textit{seq ClassBodyDeclaration} \\
\textit{OneShotEventHandler} &== \textit{Identifier} \times \textit{EventHandlerClassBody} \\
\textit{LongEventHandlerClassBody} &== \textit{HandleAsyncLongEvent} \times \textit{seq ClassBodyDeclaration} \\
\textit{AperiodicEventHandler} &::= \textit{apehType} \langle\langle \textit{Identifier} \times \textit{EventHandlerClassBody} \rangle\rangle \mid \\
&\quad \textit{aplehType} \langle\langle \textit{Identifier} \times \textit{LongEventHandlerClassBody} \rangle\rangle \\
\textit{PeriodicEventHandler} &== \textit{Identifier} \times \textit{EventHandlerClassBody} \\
\textit{EventHandler} &::= \textit{pehDecl} \langle\langle \textit{PeriodicEventHandler} \rangle\rangle \mid \\
&\quad \textit{apehDecl} \langle\langle \textit{AperiodicEventHandler} \rangle\rangle \mid \\
&\quad \textit{osehDecl} \langle\langle \textit{OneShotEventHandler} \rangle\rangle
\end{aligned}$$
$$\begin{aligned} \text{GetNextMission} &== \text{MethodBody} \\ \text{MissionSequencerClassBody} &== \text{GetNextMission} \times \text{seq } \text{ClassBodyDeclaration} \\ \text{MissionSequencer} &== \text{Identifier} \times \text{MissionSequencerClassBody} \end{aligned}$$

NestedMissionSequencer == *MissionSequencer*

$$\begin{aligned} \text{SchedulableObject} ::= & \text{handler} \langle \langle \text{EventHandler} \rangle \rangle \mid \\ & \text{mt} \langle \langle \text{ManagedThread} \rangle \rangle \mid \\ & \text{nms} \langle \langle \text{NestedMissionSequencer} \rangle \rangle \end{aligned}$$
$$\begin{aligned} \textit{Cleanup} &== \textit{MethodBody} \\ \textit{Initialize} &== \textit{MethodBody} \\ \textit{MissionClassBody} &== \textit{Initialize} \times \textit{Cleanup} \times \textit{seq ClassBodyDeclaration} \\ \textit{Mission} &== \textit{Identifier} \times \textit{MissionClassBody} \end{aligned}$$
$$\begin{aligned} Cluster &== Mission \times \text{seq } SchedulableObject \\ Tier &== \text{seq } Cluster \end{aligned}$$

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

$ImmortalMemorySize == MethodDeclaration$

$InitializeApplication == MethodBody$

$GetSequencer == MethodBody$

$SafeletClassBody ==$

$InitializeApplication \times GetSequencer \times ImmortalMemorySize \times \text{seq } ClassBodyDeclaration$

$Safelet == Identifier \times SafeletClassBody$

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

section *TransSCJProg* **parents** *scj_prelude, SCJBNFEncoding, CircusBNFEncoding*

[*ID*]

controlTierSync : *CSExpression*
Tier0 : *N*
MissionIds : seq *CircusParagraph*
SchedulableIds : seq *CircusParagraph*
ThreadIds : seq *CircusParagraph*
ObjectIds : seq *CircusParagraph*

ServicesChan : seq *CircusParagraph*
GlobalTypes : seq *CircusParagraph*
JTime : seq *CircusParagraph*
PrimitiveTypes : seq *CircusParagraph*
Priority : seq *CircusParagraph*
PriorityQueue : seq *CircusParagraph*
FrameworkChan : seq *CircusParagraph*
MissionId : seq *CircusParagraph*
SchedulableId : seq *CircusParagraph*

ObjectFW : *CircusParagraph*
ObjectChan : seq *CircusParagraph*
ObjectFWChan : seq *CircusParagraph*
ObjectMethChan : seq *CircusParagraph*
ThreadFW : *CircusParagraph*
ThreadChan : seq *CircusParagraph*
ThreadFWChan : seq *CircusParagraph*
ThreadMethChan : seq *CircusParagraph*

SafeletFW : *CircusParagraph*
SafeletFWChan : seq *CircusParagraph*
SafeletChan : seq *CircusParagraph*
SafeletMethChan : seq *CircusParagraph*

TopLevelMissionSequencerFW : *CircusParagraph*
TopLevelMissionSequencerChan : seq *CircusParagraph*
TopLevelMissionSequencerFWChan : seq *CircusParagraph*

MissionSequencerChan : seq *CircusParagraph*
MissionSequencerFWChan : seq *CircusParagraph*
MissionSequencerMethChan : seq *CircusParagraph*

MissionFW : *CircusParagraph*
MissionChan : seq *CircusParagraph*
MissionFWChan : seq *CircusParagraph*
MissionMethChan : seq *CircusParagraph*

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

PeriodicEventHandlerChan : seq *CircusParagraph*
PeriodicEventHandlerFW : *CircusParagraph*
PeriodicEventHandlerFWChan : seq *CircusParagraph*
PeriodicParameters : seq *CircusParagraph*

AperiodicEventHandlerChan : seq *CircusParagraph*
AperiodicEventHandlerFW : *CircusParagraph*
AperiodicLongEventHandlerMethChan : seq *CircusParagraph*
AperiodicParameters : seq *CircusParagraph*

OneShotEventHandlerChan : seq *CircusParagraph*
OneShotEventHandlerFW : *CircusParagraph*
OneShotEventHandlerFWChan : seq *CircusParagraph*
OneShotEventHandlerMethChan : seq *CircusParagraph*

SchedulableMissionSequencerFW : *CircusParagraph*
SchedulableMissionSequencerChan : seq *CircusParagraph*
SchedulableMissionSequencerFWChan : seq *CircusParagraph*

ManagedThreadFW : *CircusParagraph*
ManagedThreadChan : seq *CircusParagraph*
ManagedThreadFWChan : seq *CircusParagraph*
ManagedThreadMethChan : seq *CircusParagraph*

framework : *CircusProgram*

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

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

distinct(*ran fwProcName*, *ran appProcName*, *ran mcbProcName*, *ran lockProcName*)

GenerateTierProcs : $\text{seq } \textit{Tier} \rightarrow \textit{Process}$

GenerateTiers : $\text{seq } \textit{Tier} \rightarrow \text{seq } \textit{Process}$

ProcessID : $N \rightarrow \textit{ID}$

ClusterEnv == $N \times \text{seq } N \times \text{seq } N \times \text{seq } N \times \text{seq } N \times \text{seq } N$

TierEnv == $\text{seq } \textit{ClusterEnv}$

FWEnv == $N \times \text{seq } \textit{TierEnv}$

BuildFWEnv : $\textit{SCJProgram} \rightarrow \textit{FWEnv}$

$\text{SafeletFWName} : N$
 $\text{TopLevelMissionSequencerFWNMame} : N$

$\text{GenerateFWProcs} : N \times \text{seq } \text{TierEnv} \rightarrow \text{seq } \text{Process}$

$\text{AppEnv} == N \times \text{seq } \text{Expression}$

$\text{AppProcEnv} == \text{AppEnv} \times \text{AppEnv} \times \text{seq seq } \text{AppEnv} \times \text{AppEnv}$

$\text{BuildAppProcEnv} : \text{SCJProgram} \rightarrow \text{AppProcEnv}$

$\text{GenerateAppProc} : \text{AppProcEnv} \rightarrow \text{Process}$

$[\text{Type}]$

$\text{BinderMethodEnv} == N \times \mathbb{F} N \times \mathbb{F} N \times \text{Type} \times \text{seq } \text{Type} \times \mathbb{B} \times N \times \text{Type} \times \text{Type}$

$\text{MCBEnv} == \text{seq } \text{BinderMethodEnv}$

$\text{BuildMCBEnv} : \text{SCJProgram} \rightarrow \text{seq } \text{BinderMethodEnv}$

$\text{GenerateMCBProc} : \text{seq } \text{BinderMethodEnv} \rightarrow \text{Process}$

$\text{LockingEnv} == \text{seq } \text{ThreadIds} \times \text{seq } \text{ObjectIds}$

$\text{BuildLockEnv} : \text{SCJProgram} \rightarrow \text{LockingEnv}$

$\text{GenerateLockProc} : \text{LockingEnv} \rightarrow \text{seq } \text{CircusParagraph}$

$\text{TransClasses} : \text{SCJProgram} \rightarrow \text{CircusProgram}$

$TransSCJProg : Identifier \times SCJProgram \rightarrow CircusProgram$

$\forall scjProg : SCJProgram; name : Identifier \bullet$
 $\exists app : CircusProgram;$
 $program : CircusProgram; n : N; p : Process;$
 $appComms : CSEexpression; mcbComms : CSEexpression; lockComms : CSEexpression$
 $fwProcs : seq Process; appProc : Process; lockProcs : seq CircusParagraph; mcbProc : Process;$
 $app = TransClasses(scjProg) \wedge$
 $fwProcs = GenerateFWProcs(BuildFWEnv(scjProg)) \wedge$
 $appProc = GenerateAppProc(BuildAppProcEnv(scjProg)) \wedge$
 $mcbProc = GenerateMCBProc(BuildMCBEnv(scjProg)) \wedge$
 $lockProcs = GenerateLockProc(BuildLockEnv(scjProg)) \wedge$
 $program = \langle procDef(pd(name,$
 $\quad procHide(procPar($
 $\quad \quad procHide(procPar($
 $\quad \quad \quad procHide(procPar(fwProcs, appComms, appProc), appComms)$
 $\quad \quad \quad , mcbComms, mcbProc), mcbComms),$
 $\quad \quad \quad lockComms, lockProc), lockComms)))) \bullet$
 $TransSCJProg(name, scjProg) =$
 $framework \wedge \langle procDef(pd(fwProcName(n), fwProcs)) \rangle \wedge$
 $app \wedge \langle procDef(pd(appProcName(n), appProc)) \rangle \wedge$
 $\langle procDef(pd(mcbProcName(n), mcbProc)) \rangle \wedge$
 $\langle procDef(pd(lockProcName(n), lockProcs)) \rangle \wedge$
 $program$

Low Level

- $Method : MethodDeclaration \mapsto (Name, Params, ReturnType, Body) :$ translates an active application method into a *Circus* action
- $DataMethod : MethodDeclaration \mapsto :$ translates data methods into an *OhCircus* method
- $MethodBody : Block \mapsto \text{seq } CircExpression :$ translates a Java block, for example a method body
- $Registers : Block \mapsto \text{seq } Name :$ extracts the Names of the schedulables registered in a Java block
- $Returns : Block \mapsto \text{seq } Name :$ extracts the Names of the variables returned in a Java block
- $Variable : (Name, Type, InitExpression) \mapsto (CircName, CircType, CircExpression) :$ translates a variable
- $Parameters : (Name, Params, ReturnType, Body) \mapsto \text{seq } CircParam :$ translates a list of method parameters
- $\llbracket Name \rrbracket_{name} :$ translates the *name* to a Z identifier
- $\llbracket varType \rrbracket_{type} :$ translates types
- $\llbracket expr \rrbracket_{expression} :$ 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 $\llbracket Identifier \rrbracket_{Name} App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters} \mathbf{begin}$

State
this : ref $\llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
State '
this := **new** $\llbracket Identifier \rrbracket_{name} Class()$

InitializeApplication $\hat{=}$
 $\left(\begin{array}{l} initializeApplicationCall \longrightarrow \\ \llbracket \llbracket InitializeApplication \rrbracket_{Method} \rrbracket_{MethBody} \\ initializeApplicationRet \longrightarrow \\ \mathbf{Skip} \end{array} \right)$

GetSequencer $\hat{=}$
 $\left(\begin{array}{l} getSequencerCall \longrightarrow \\ getSequencerRet ! \llbracket GetSequencer \rrbracket_{Returns} \longrightarrow \\ \mathbf{Skip} \end{array} \right)$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$$Methods \hat{=} \left(\begin{array}{l} GetSequencer \\ \square \\ InitializeApplication \\ \square \\ MethName(AppMeth_1) \\ \square \\ \dots \\ MethName(AppMeth_n) \\ \dots \end{array} \right) ; Methods$$

$$\bullet (Init ; Methods) \triangle (end_safelet_app \longrightarrow \mathbf{Skip})$$

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 $\llbracket Identifier \rrbracket_{Name} App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State

this : ref $\llbracket Identifier \rrbracket_{name} Class$

state *State*

Init

State'
this := **new** $\llbracket Identifier \rrbracket_{name} Class()$

GetNextMission $\hat{=} \mathbf{var} \text{ ret} : MissionID \bullet$

$$\left(\begin{array}{l} getNextMissionCall . IdOf(Identifier) \longrightarrow \\ ret := this . getNextMission(); \\ getNextMissionRet . IdOf(Identifier) ! ret \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

Methods $\hat{=}$

$$\left(\begin{array}{l} GetNextMission \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right); Methods$$

$\bullet (Init ; Methods) \triangle (end_sequencer_app . IdOf(Identifier) \longrightarrow \mathbf{Skip})$

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 $\llbracket Identifier \rrbracket App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
this : ref $\llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
State '
this := **new** $\llbracket Identifier \rrbracket_{name} Class()$

InitializePhase $\hat{=}$

$$\left(\begin{array}{l} initializeCall . IdOf(Identifier) \longrightarrow \\ \llbracket initialize \rrbracket_{Registers} initializeRet . IdOf(Identifier) \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

CleanupPhase $\hat{=}$

$$\left(\begin{array}{l} cleanupMissionCall . IdOf(Identifier) \longrightarrow \\ cleanupMissionRet . IdOf(Identifier) ! \mathbf{True} \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$Methods \hat{=} \left(\begin{array}{l} InitializePhase \\ \square \\ CleanupPhase \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right) ; Methods$

• $(Init ; Methods) \triangle (end_mission_app . IdOf(Identifier) \longrightarrow \mathbf{Skip}$

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 $\llbracket PName \rrbracket App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
this : ref $\llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
State '
this := **new** $\llbracket Identifier \rrbracket_{name} Class()$

handleAsyncEvent $\hat{=}$

$$\left(\begin{array}{l} handleAsyncEventCall . IdOf(PName) \longrightarrow \\ \llbracket \llbracket HandleAsyncBody \rrbracket_{Method} \rrbracket_{MethBody}; \\ handleAsyncEventRet . IdOf(PName) \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

Methods $\hat{=}$

$$\left(\begin{array}{l} handleAsyncEvent \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right); Methods$$

• (*Init* ; *Methods*) $\triangle (end_ \llbracket HandlerTypeIdOf(PName) \rrbracket \longrightarrow \mathbf{Skip})$

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 }

```

process $\llbracket PName \rrbracket App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
 $this : \text{ref } \llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
 $State'$
 $this := \text{new } \llbracket Identifier \rrbracket_{name} Class()$

$Run \hat{=}$

$$\left(\begin{array}{l} runCall . IdOf(PName) \longrightarrow \\ \llbracket \llbracket run \rrbracket_{Method} \rrbracket_{MethBody}; \\ runRet . IfOf(PName) \longrightarrow \\ \text{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$Methods \hat{=}$

$$\left(\begin{array}{l} Run \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right); Methods$$

• $(Init ; Methods) \triangle (end_managedThread_app . IdOf(PName) \longrightarrow \text{Skip})$

end

Data Class

class $\llbracket PName \rrbracket_{name}$ *Class* $\hat{=}$ **begin**

state *State*

$\llbracket VarName \rrbracket_{name} : \llbracket VarType \rrbracket_{type}$

state *State*

initial *Init*

State '

$\llbracket VarName \rrbracket'_{name} = \llbracket VarInit \rrbracket_{expression}$

$\llbracket DataMeth1 \rrbracket_{dataMeth}$

$\llbracket DataMeth2 \rrbracket_{dataMeth}$

...

• **Skip**

end