HYBRIS – Efficient Analysis of Hybrid Systems

Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska

Universität Bremen

Germany

{kirsten,bisanz,ulrichh,jp}@informatik.uni-bremen.de

Outline

1. HybridUML – Specification Formalism for Hybrid Systems

2. Automated Model-Based Code Generation – The Hybrid Low-Level Framework HL^3 as Compilation Target for Real-Time Specification Formalisms

3. Automated Testing against (Hybrid) Real-Time Specifications – Timed CSP, HybridCSP, HybridUML

4. Domain-Specific Descriptions for Railway Control Systems

HybridUML – Specification Formalism for Hybrid Systems

- Formal Semantics

- UML

HybridUML – Specification Formalism for Hybrid Systems

- Formal Semantics

  - + Hybrid automata (Henzinger)
  - + Hierarchic Modelling
  - + Semantics by Transformation $\Rightarrow HL^3$
  - + Semantically well-defined
  - + Executable in Hard Real-Time

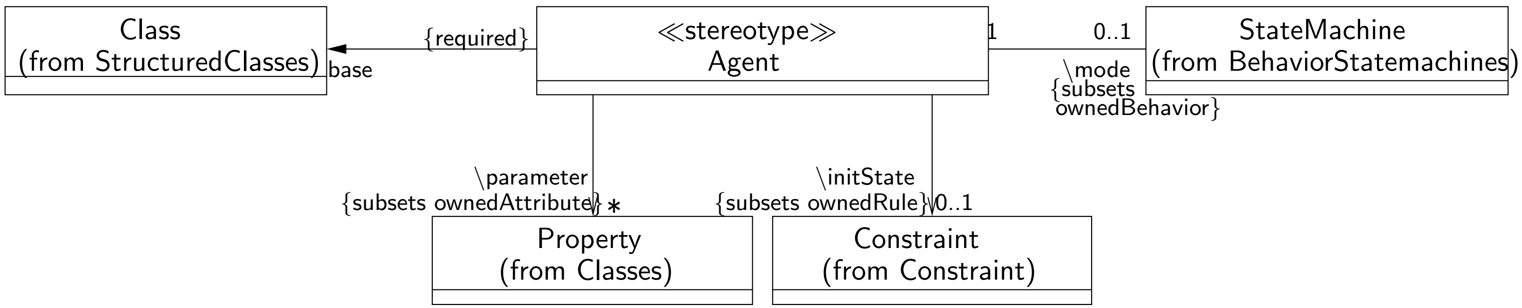
- UML

HybridUML – Specification Formalism for Hybrid Systems

- Formal Semantics

- UML

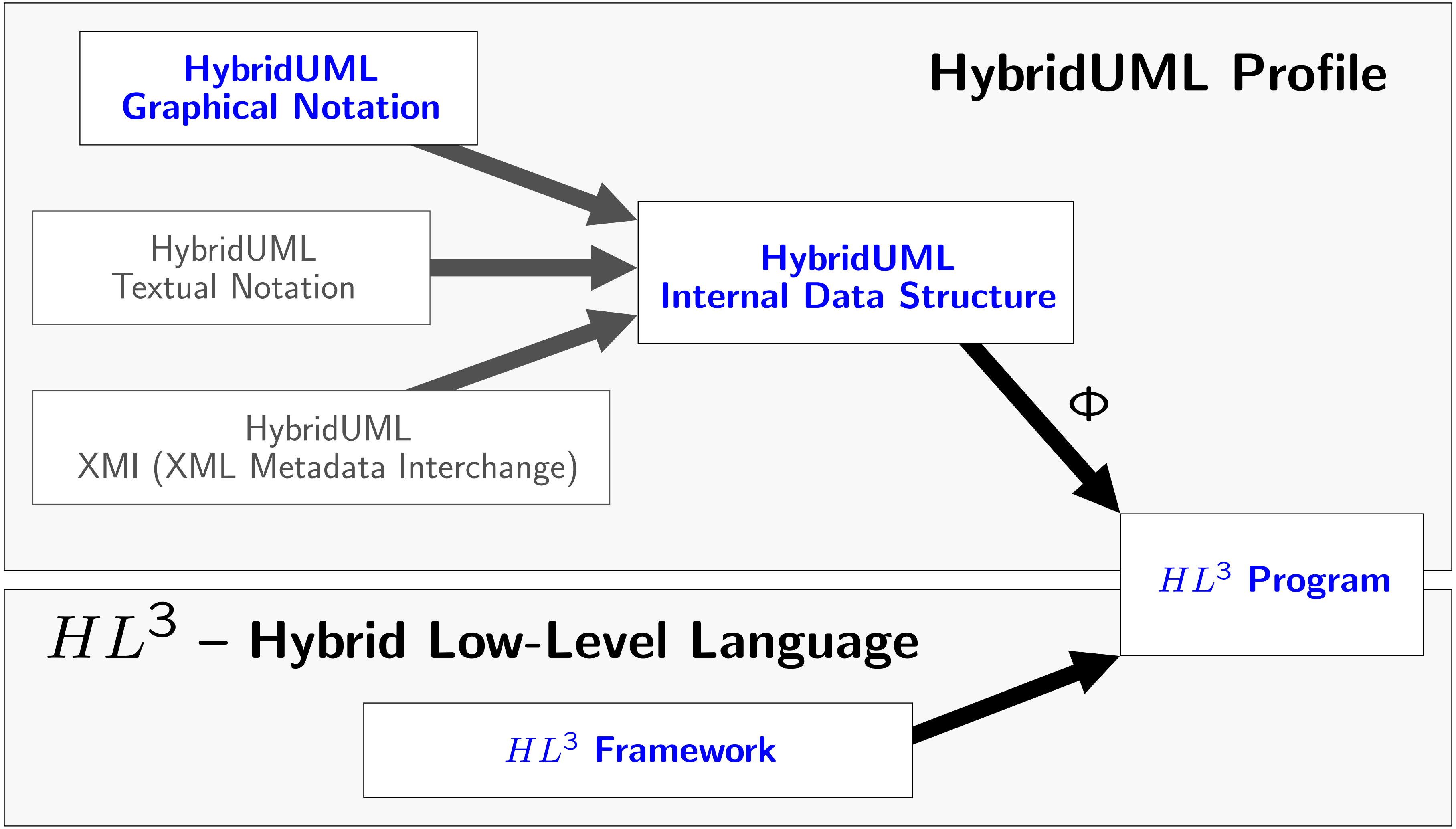
+ Customisation of UML 2.0 – HybridUML Profile:



Additional constraints, i.e. `self.mode->forAll(oclIsTypeOf(Mode))`

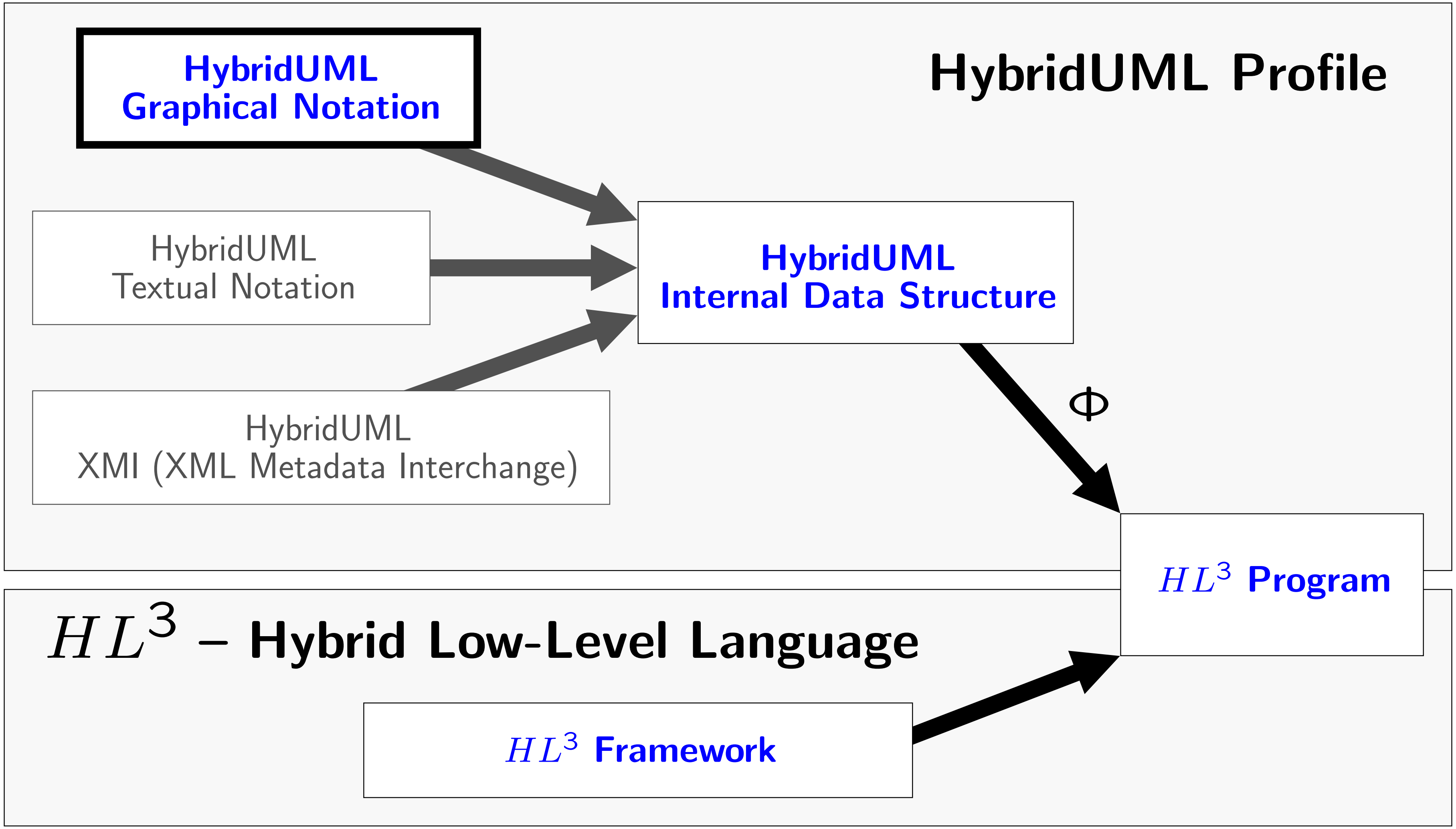
+ Enhanced by (time-continuous) Flow Constraints and Invariants

HybridUML – Specification Formalism for Hybrid Systems

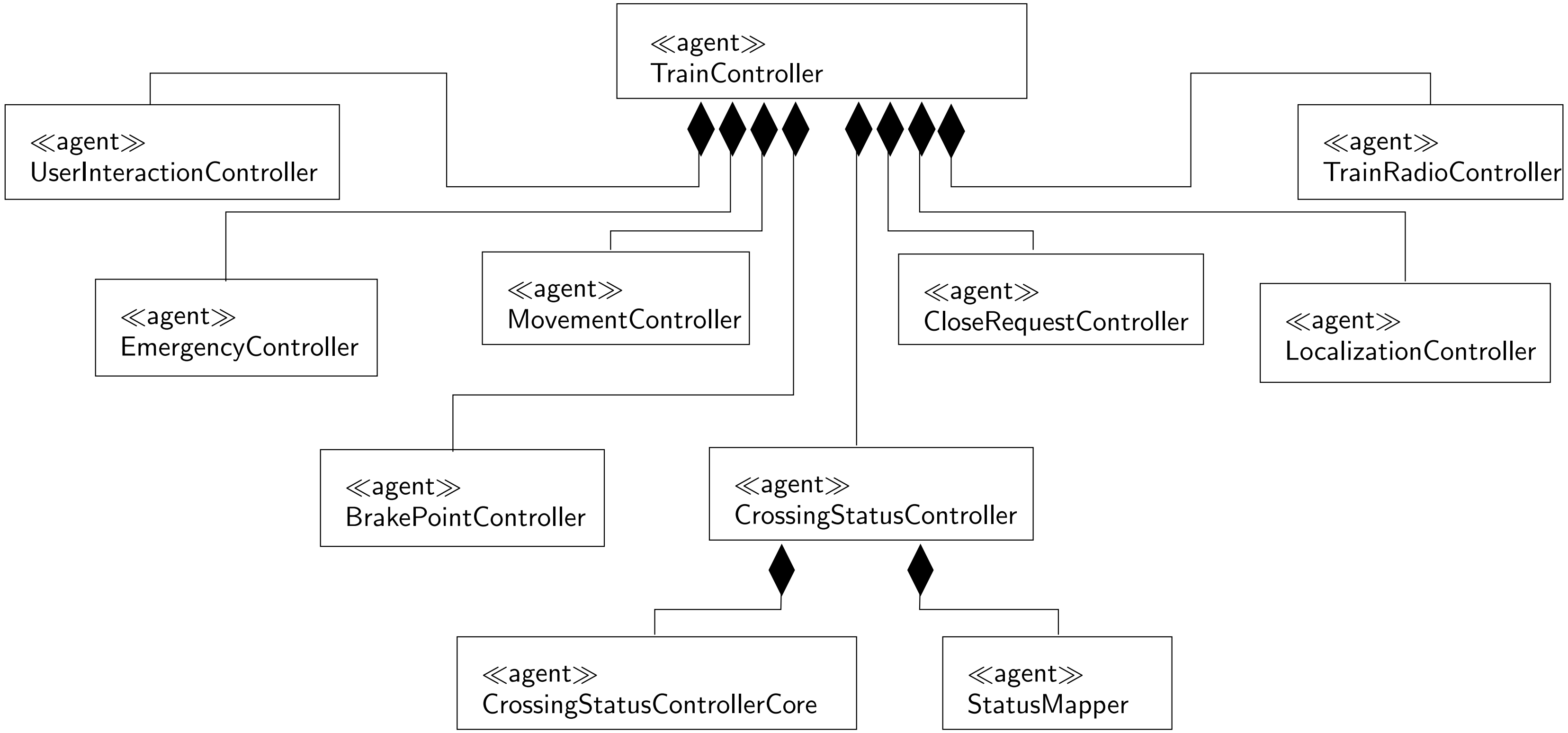


HybridUML Models

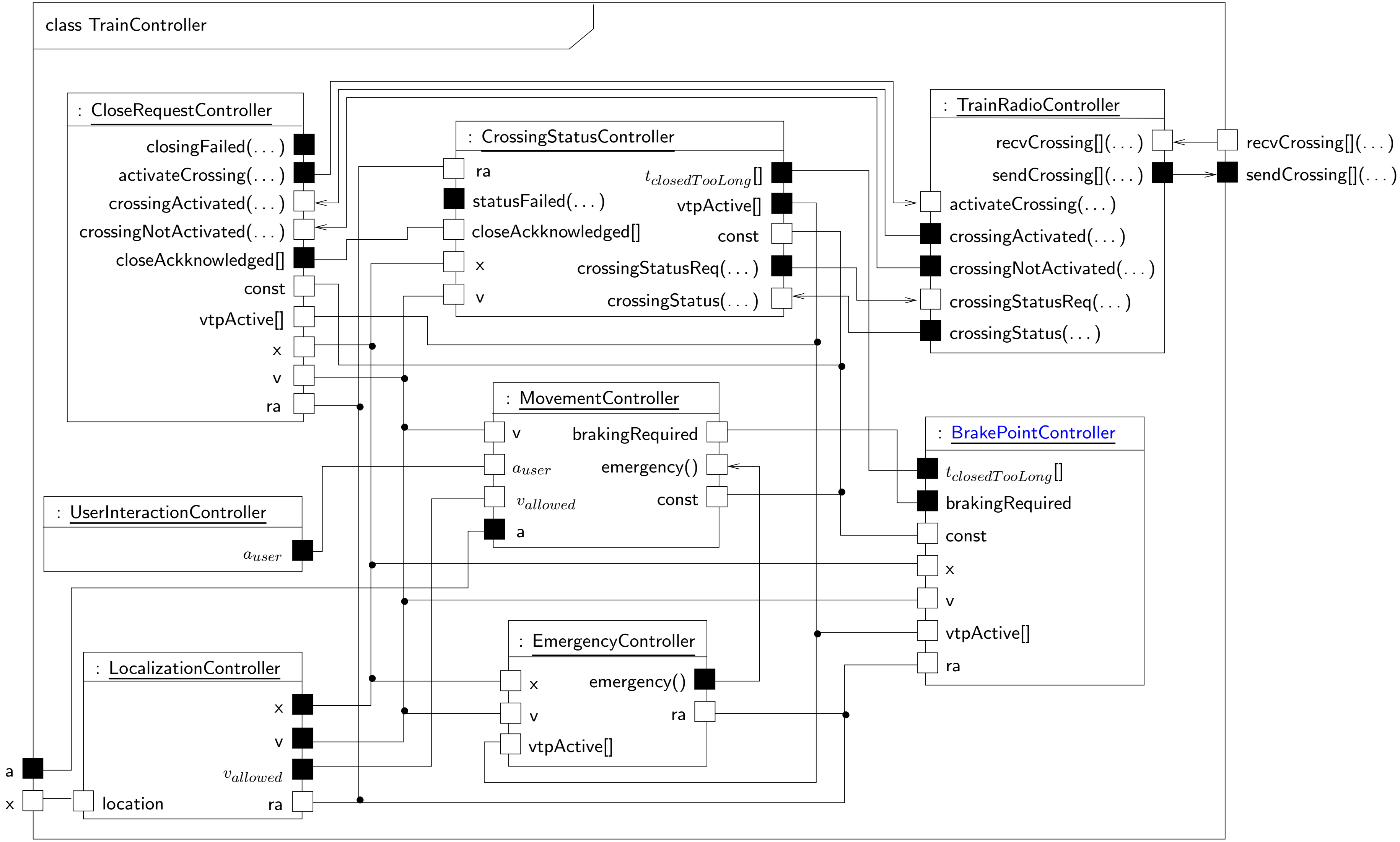
HybridUML Graphical Notation



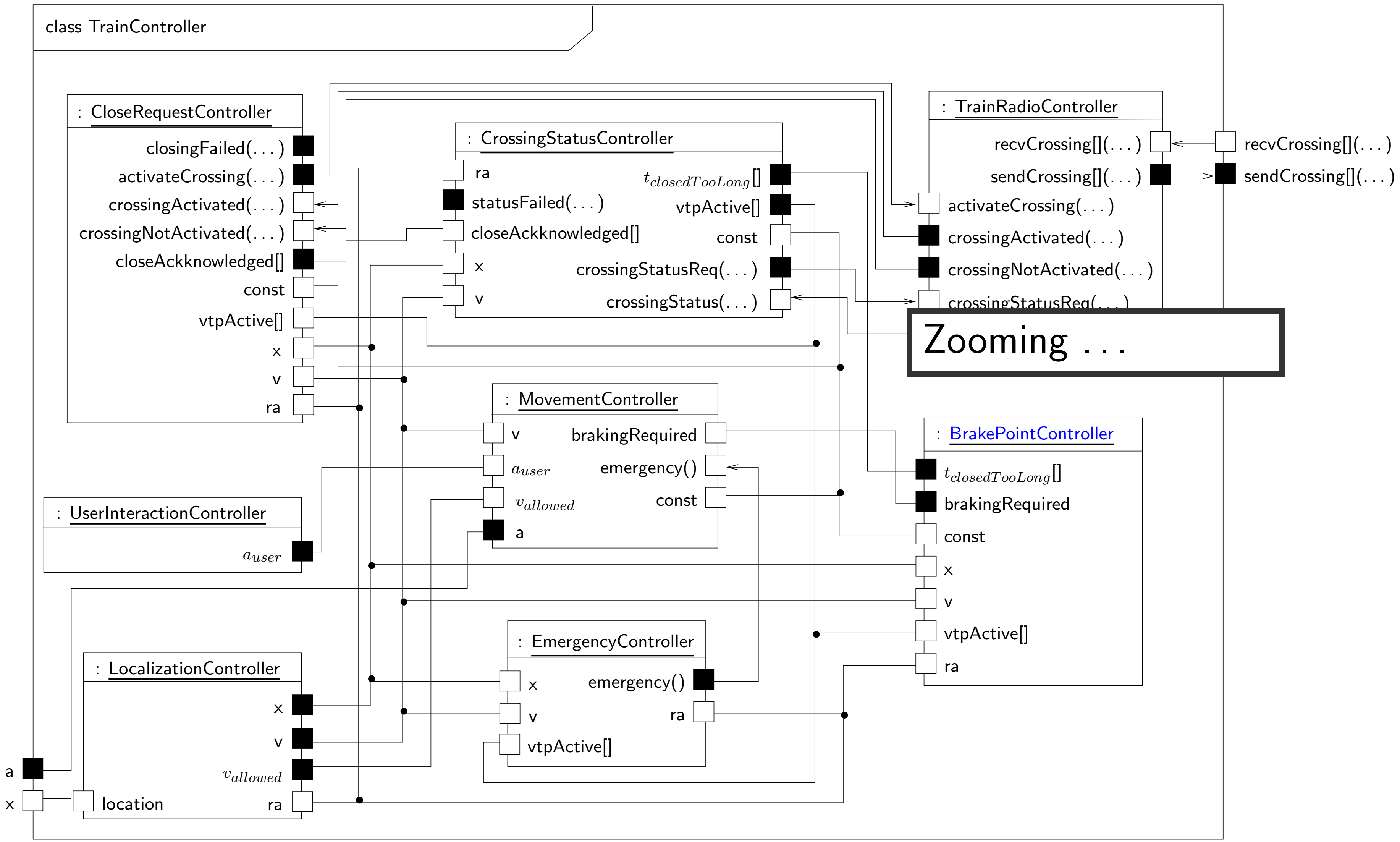
HybridUML Models – Graphical Notation



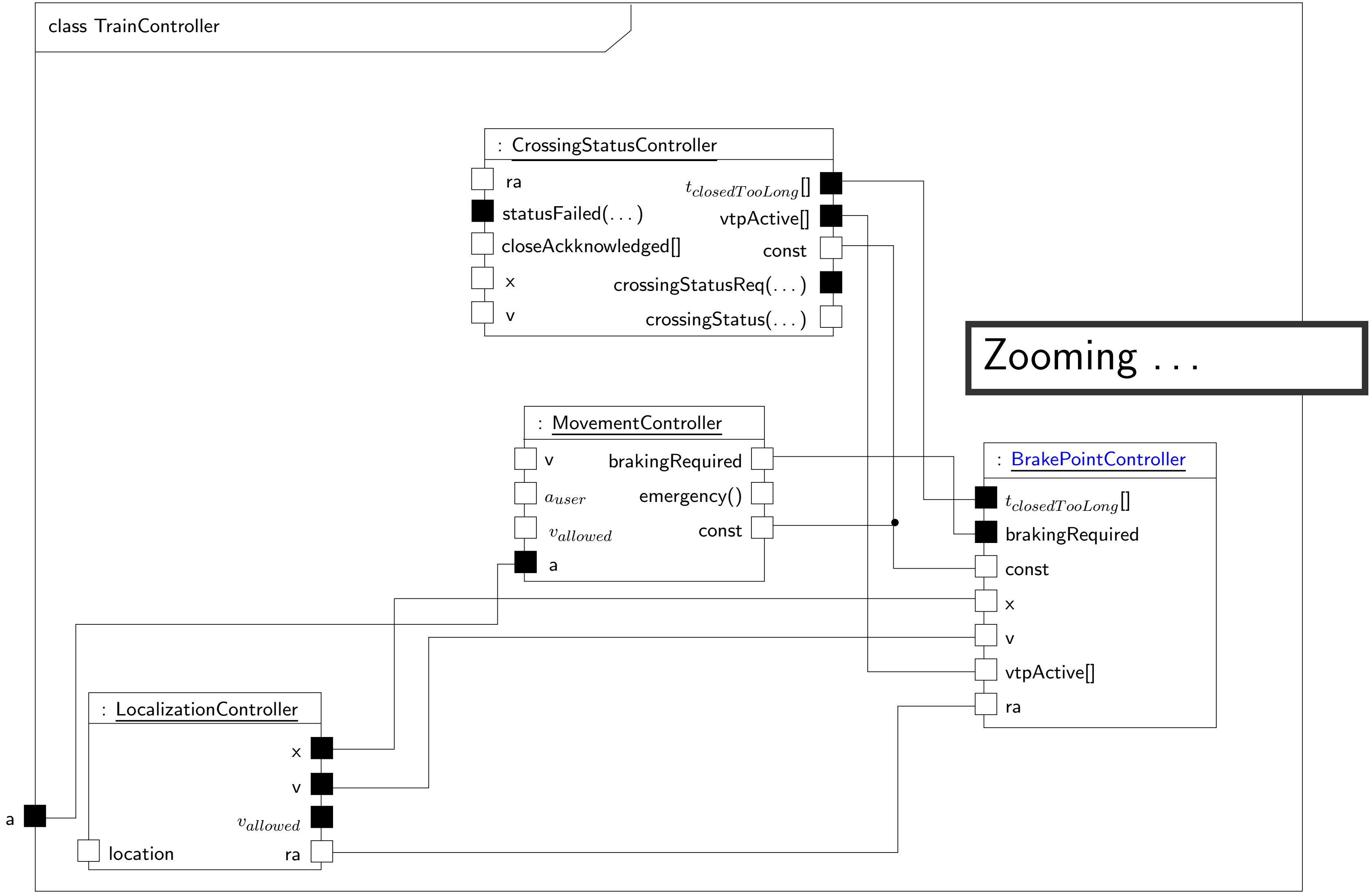
Class diagram of TrainController



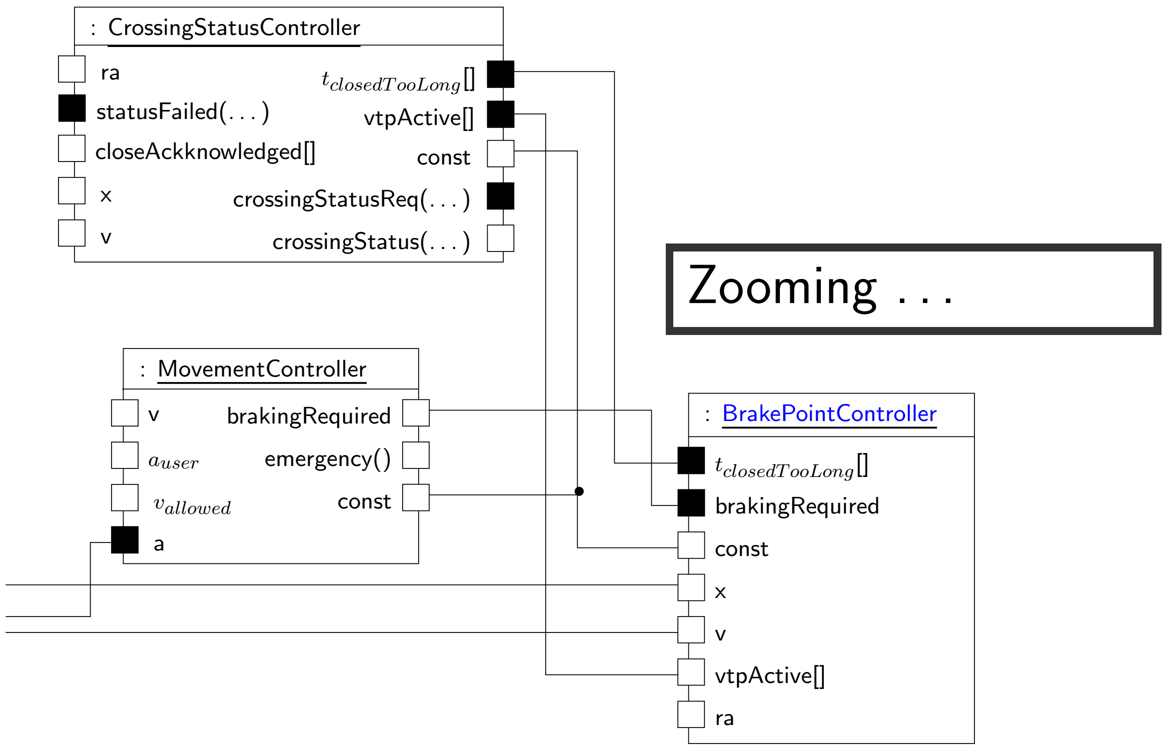
Structure diagram of TrainController



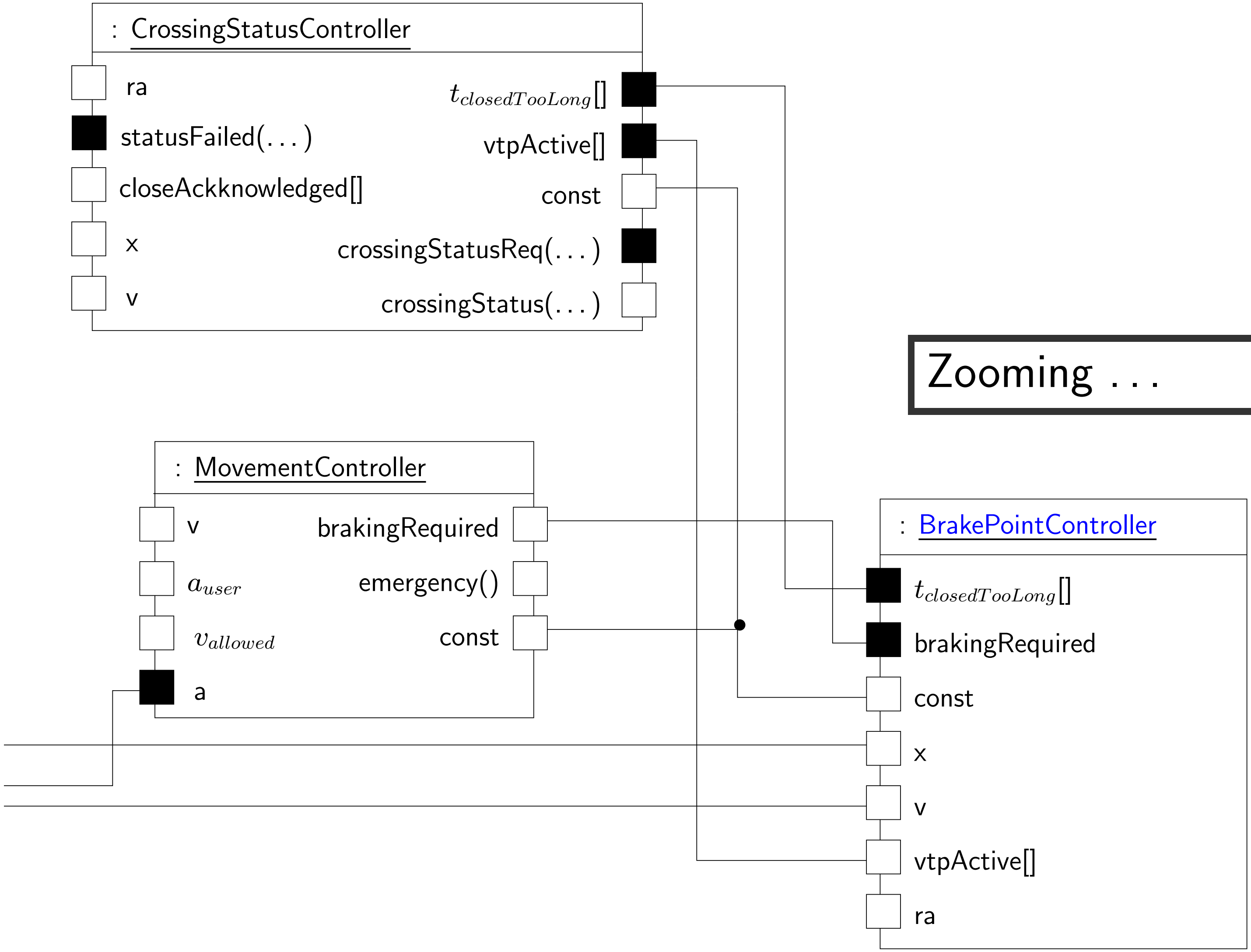
Structure diagram of TrainController



Focus on BrakePointController

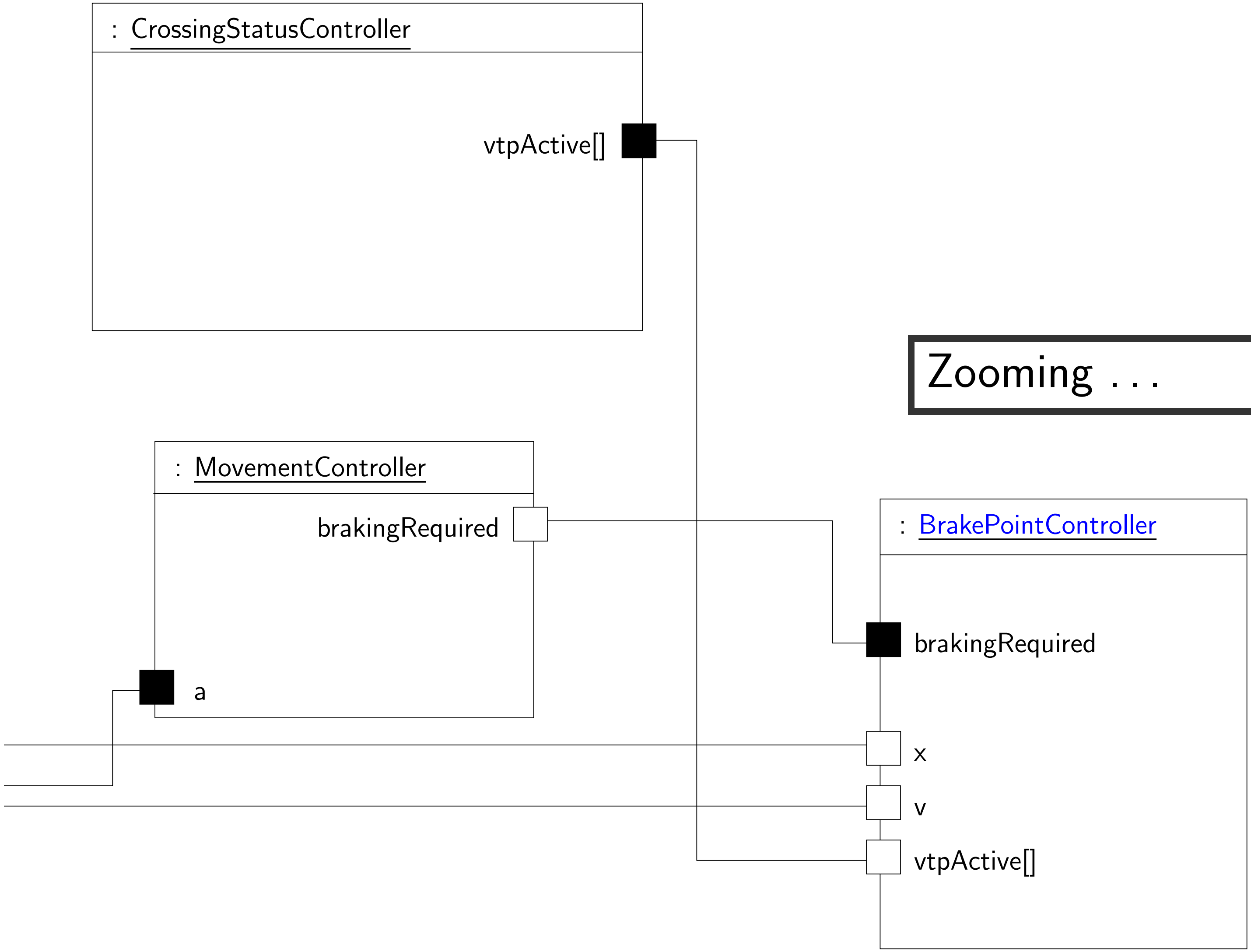


Focus on BrakePointController



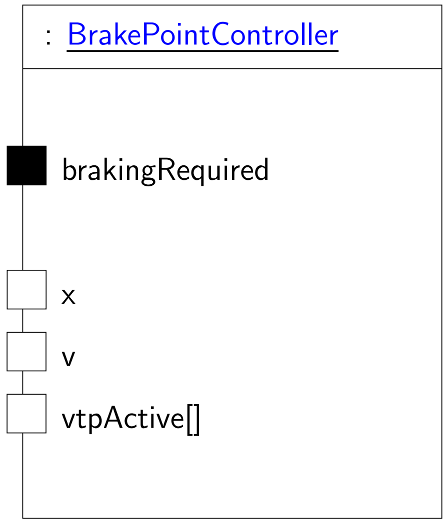
Zooming ...

Focus on BrakePointController

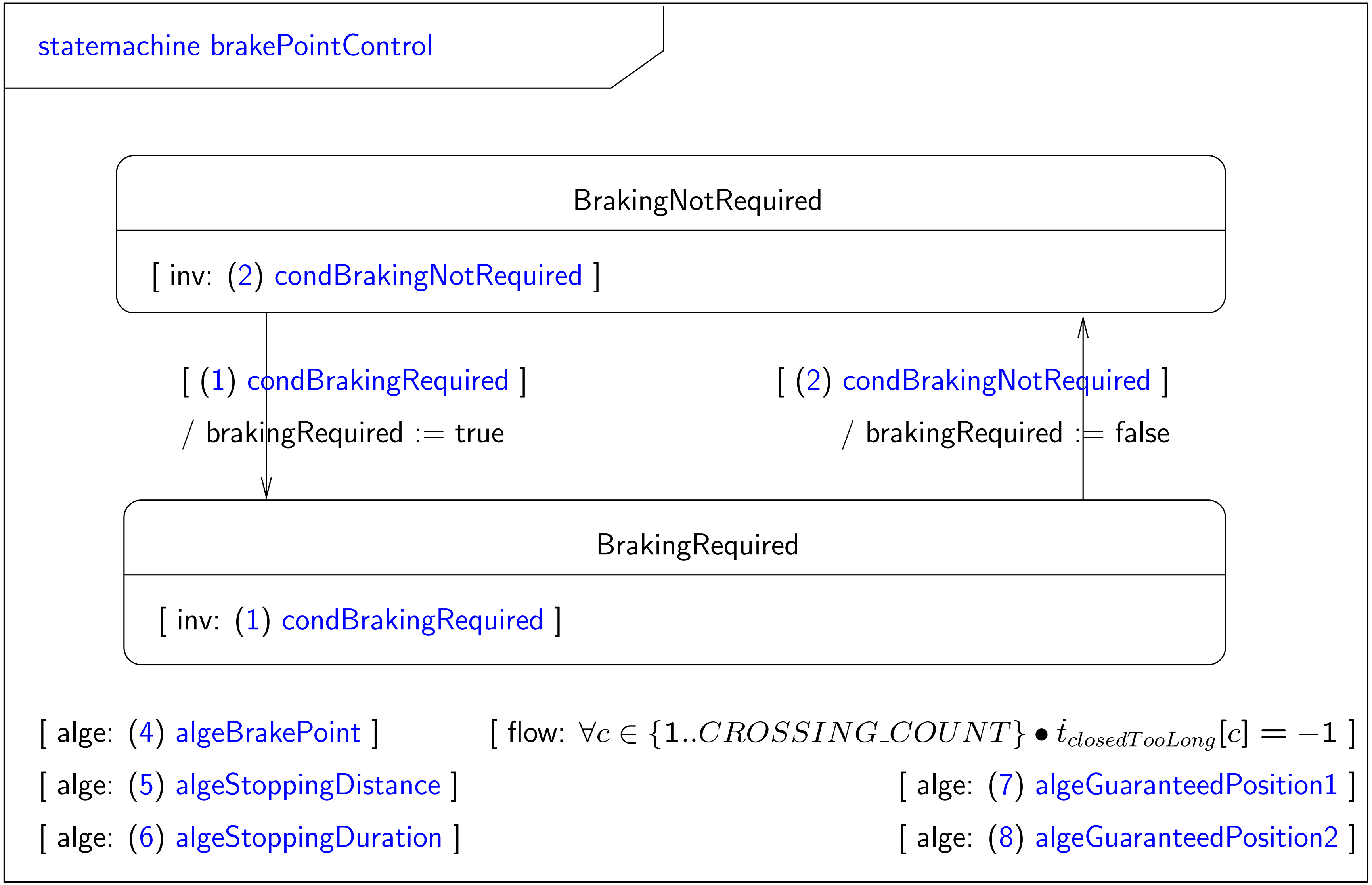


Zooming . . .

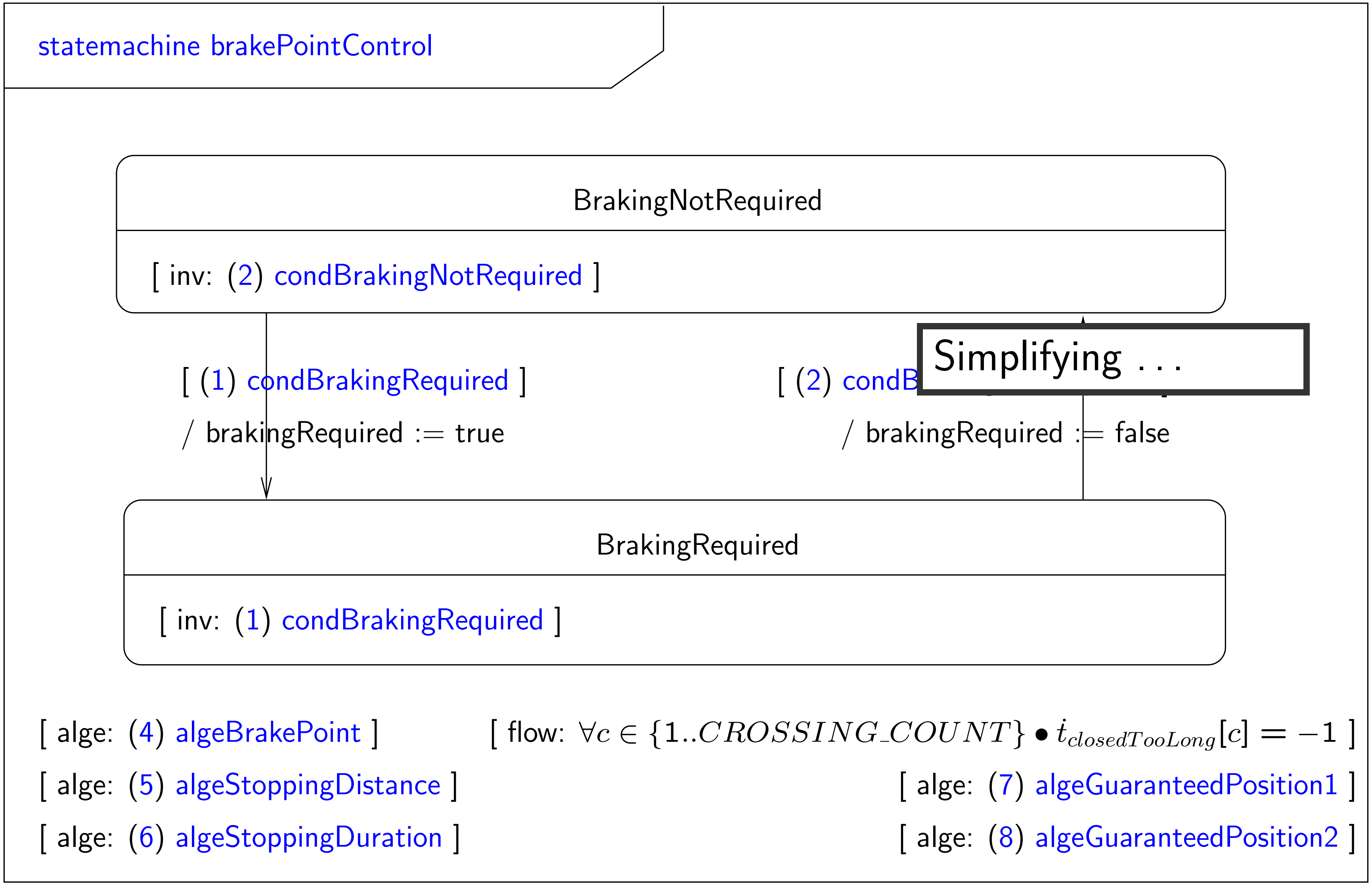
Focus on BrakePointController



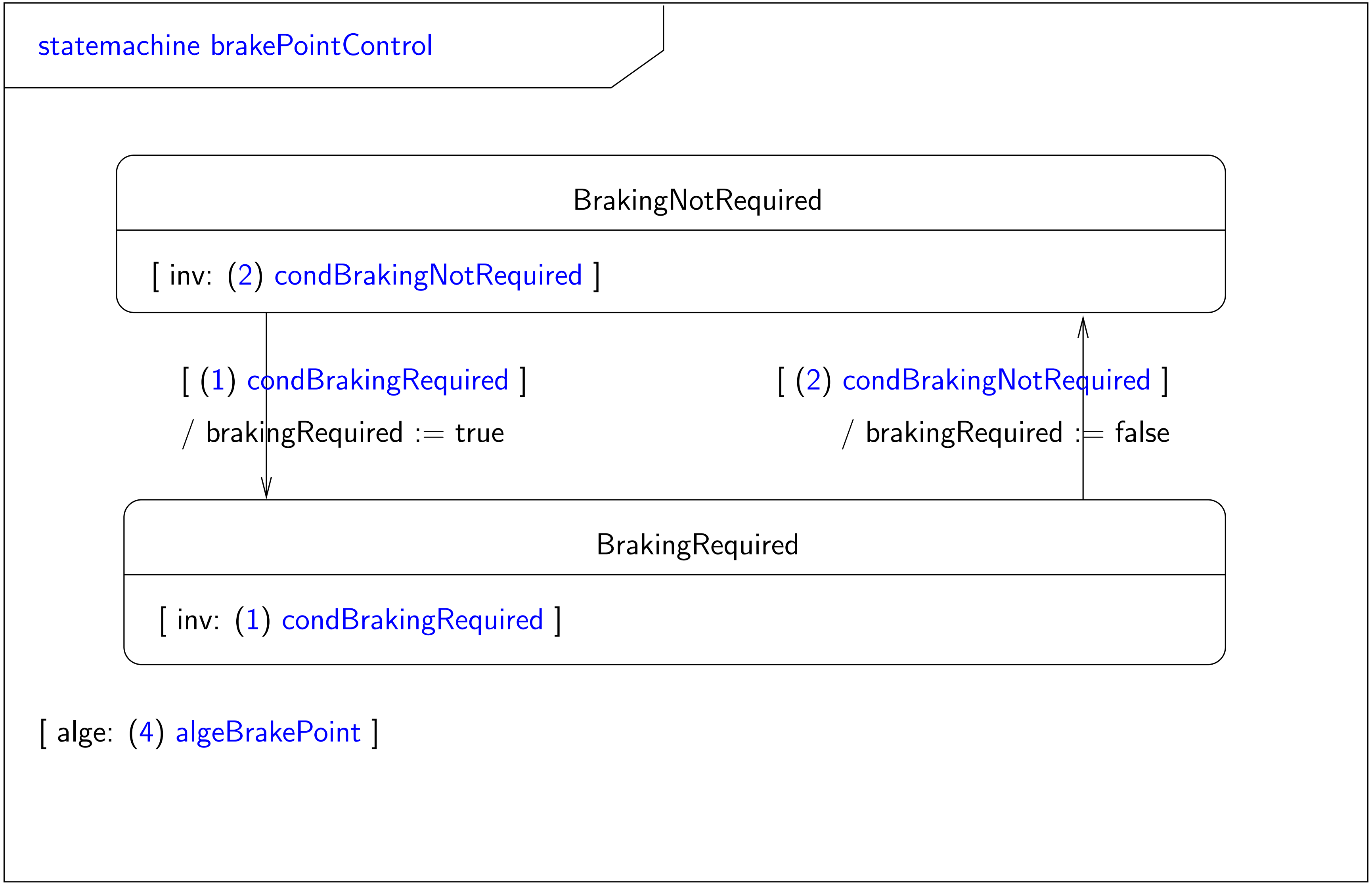
Focus on BrakePointController



Behaviour of BrakePointController

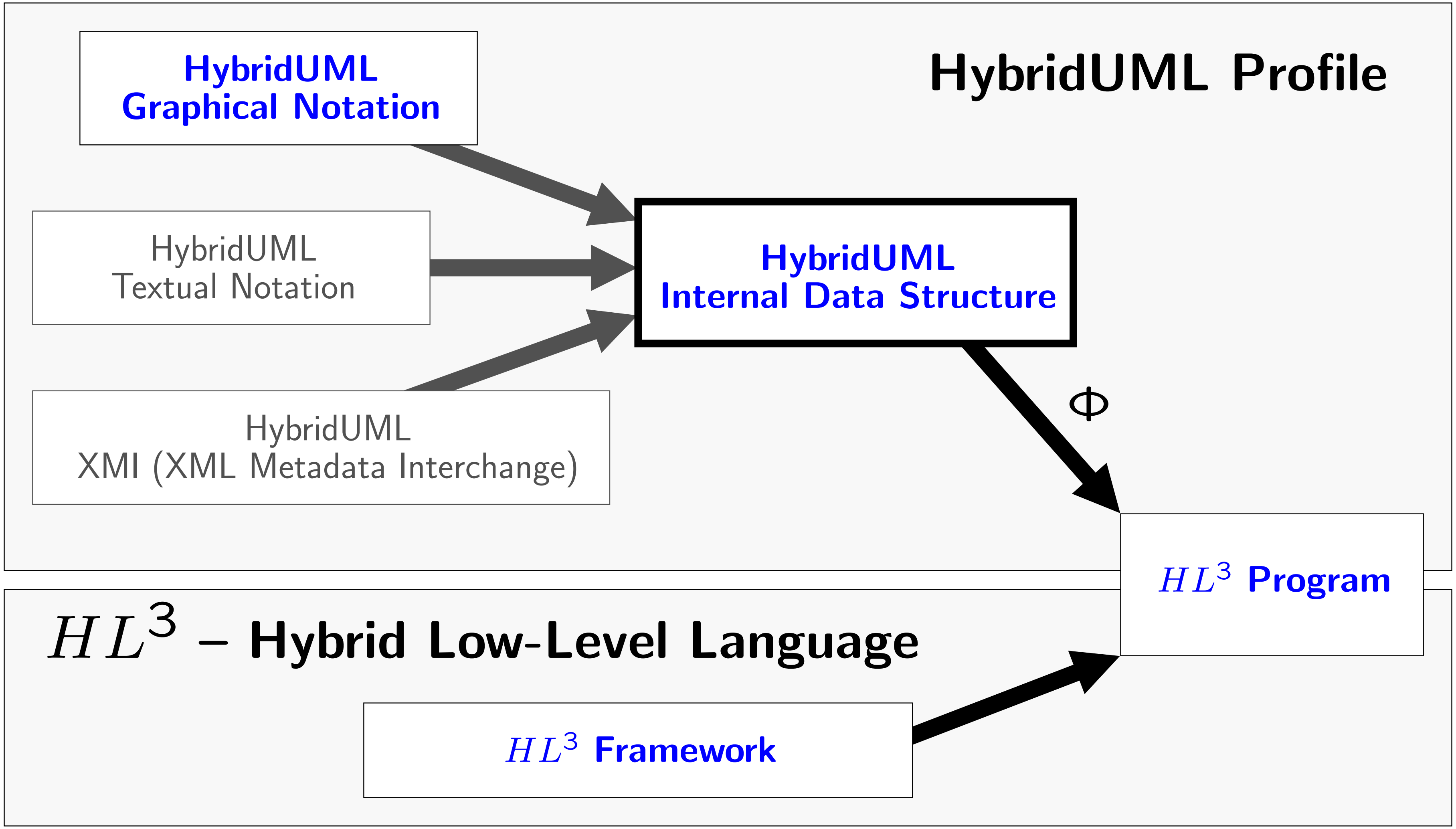


Behaviour of BrakePointController

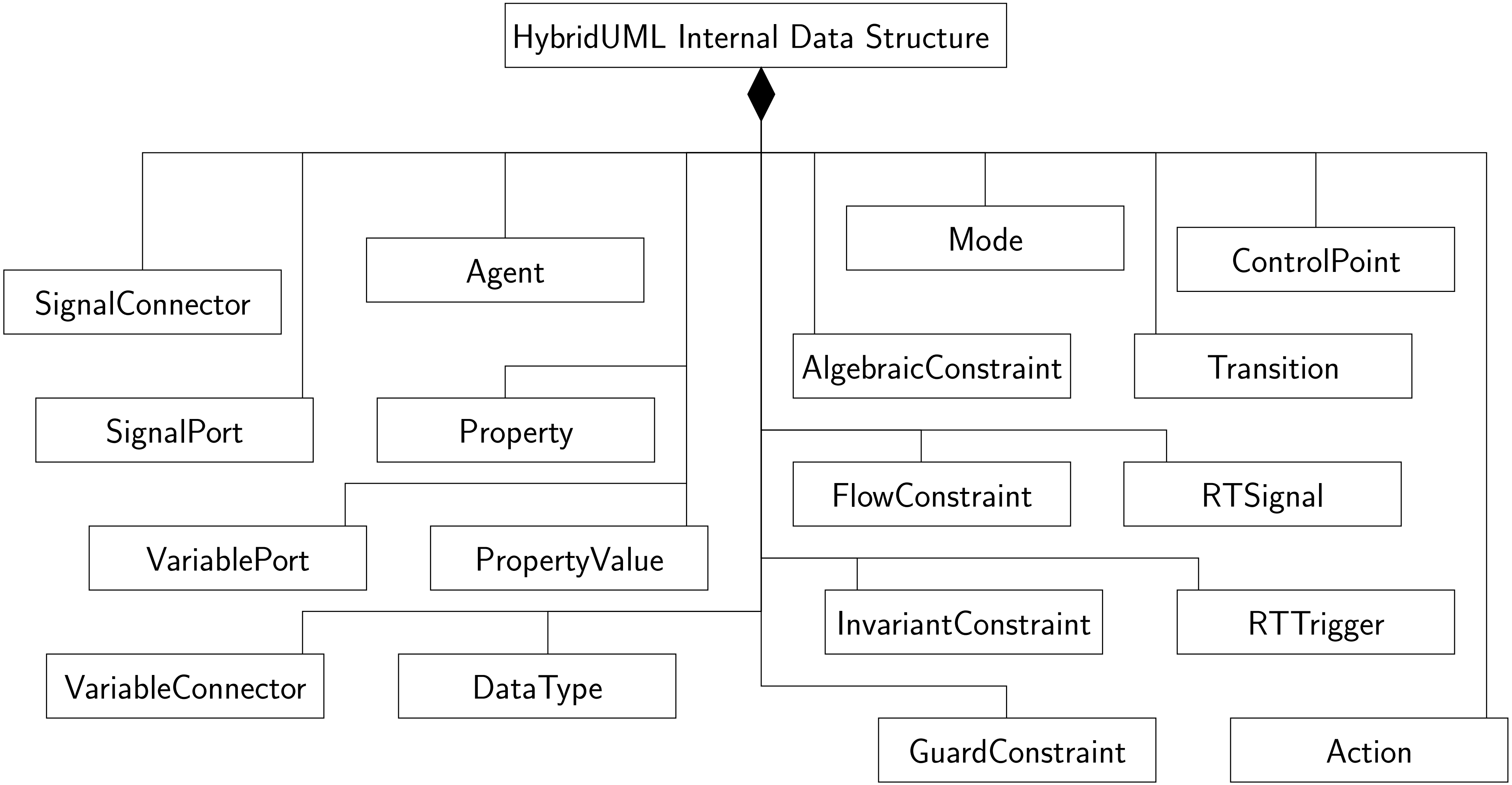


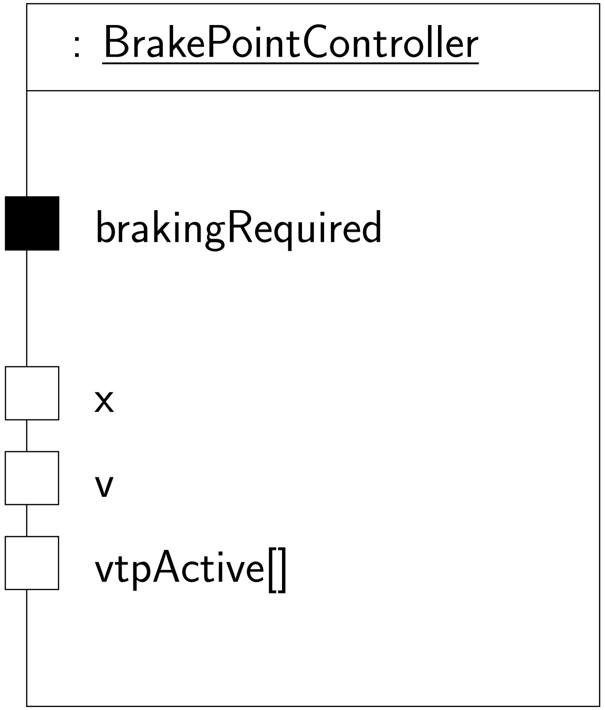
Simplified behaviour of BrakePointController

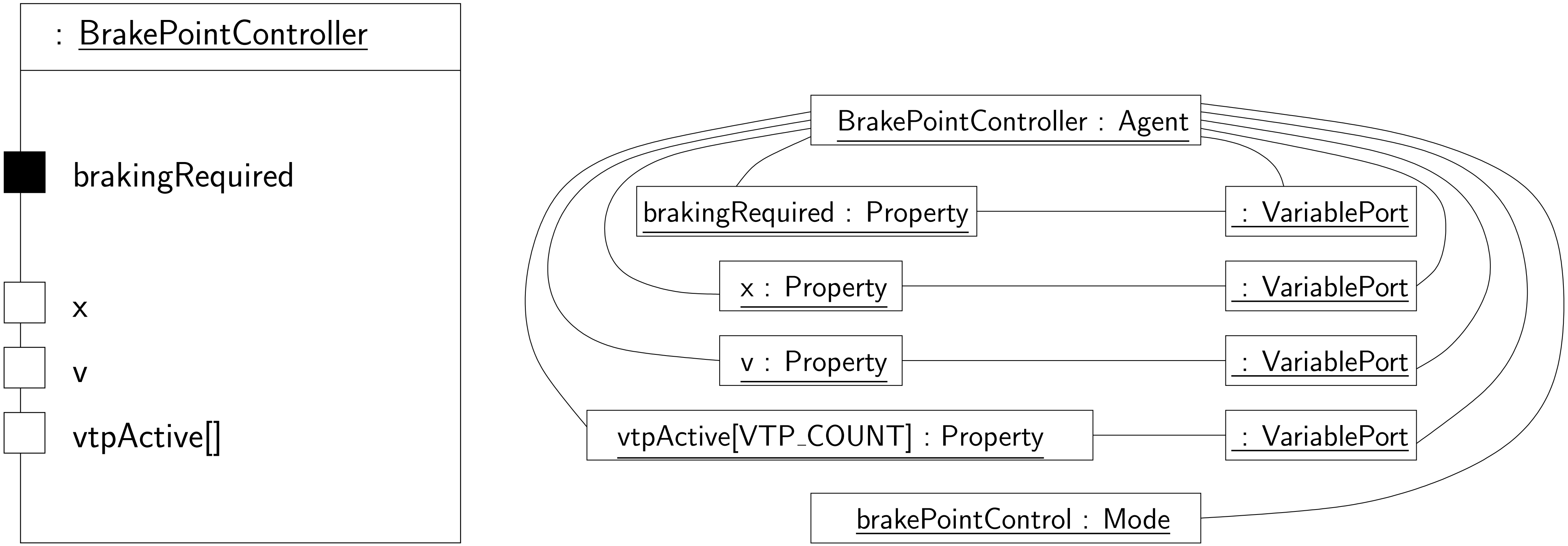
HybridUML Internal Data Structure



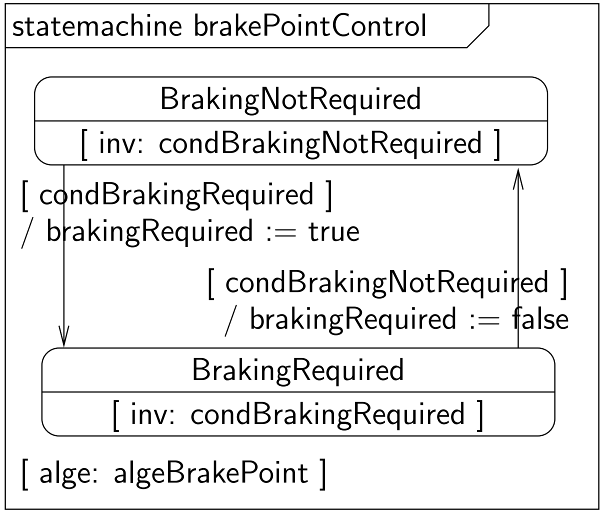
HybridUML Models – Internal Data Structure

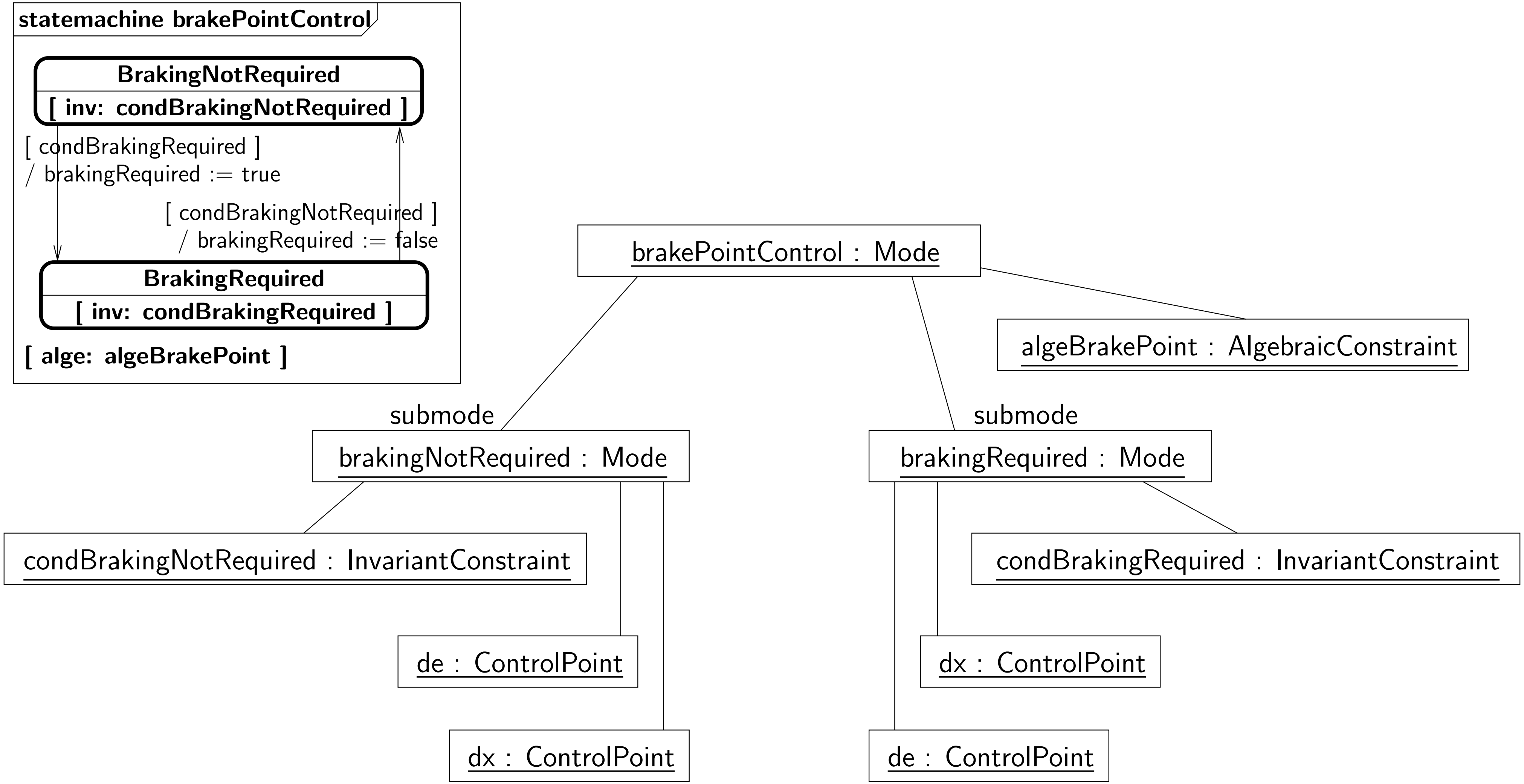

HybridUML Internal Data Structure
SignalConnector
Agent
Mode
ControlPoint
AlgebraicConstraint
Transition
SignalPort
Property
FlowConstraint
RTSignal
VariablePort
PropertyValue
InvariantConstraint
RTTrigger
VariableConnector
DataType
GuardConstraint
Action


: BrakePointController
brakingRequired
x
v
vtpActive[]



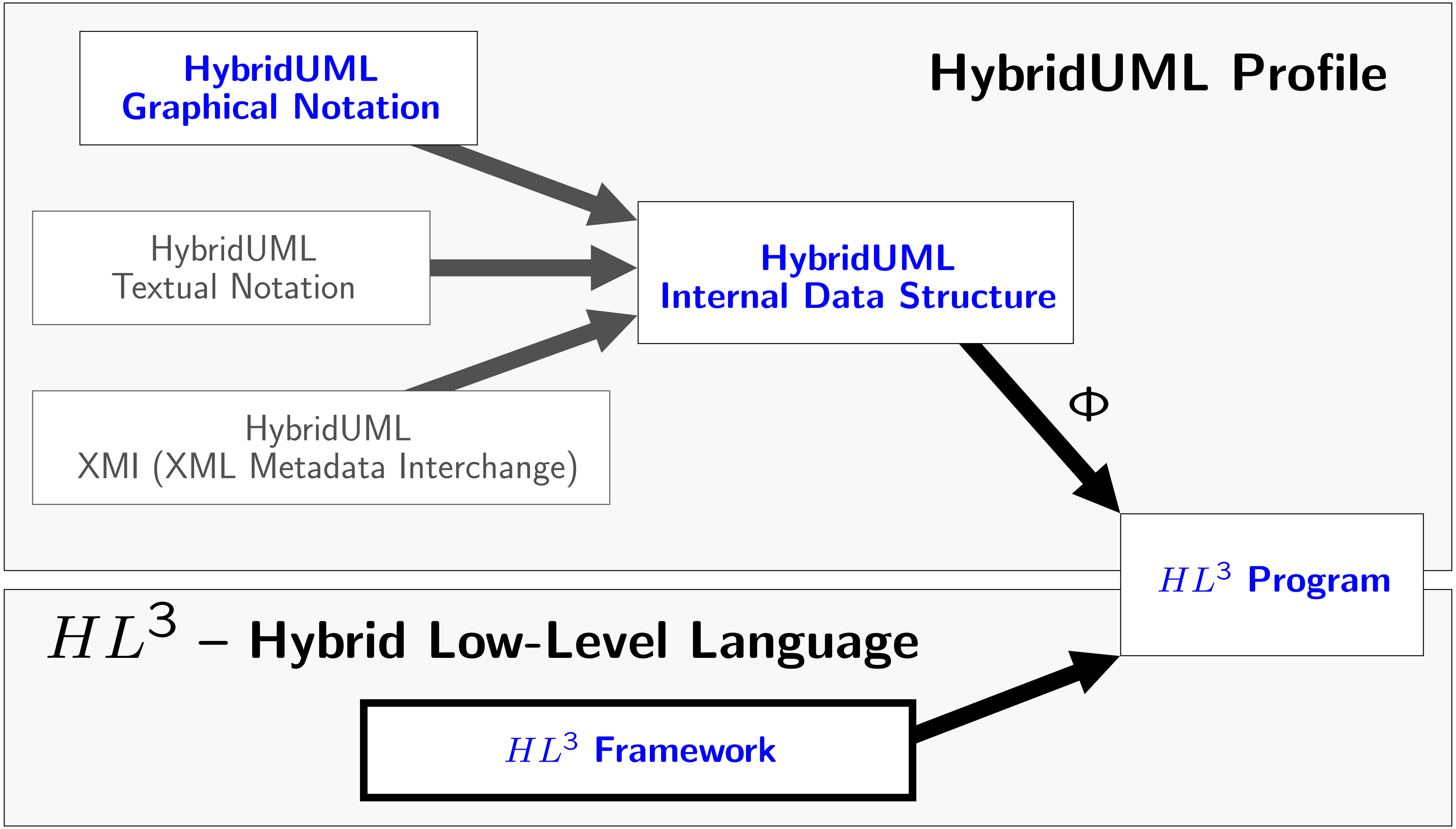
Internal Data Structure Representation of BrakePointController


statemachine brakePointControl
BrakingNotRequired
[inv: condBrakingNotRequired]
[condBrakingRequired]
/ brakingRequired := true
[condBrakingNotRequired]
/ brakingRequired := false
BrakingRequired
[inv: condBrakingRequired]
[alge: algeBrakePoint]



Internal Data Structure Representation of BrakePointController

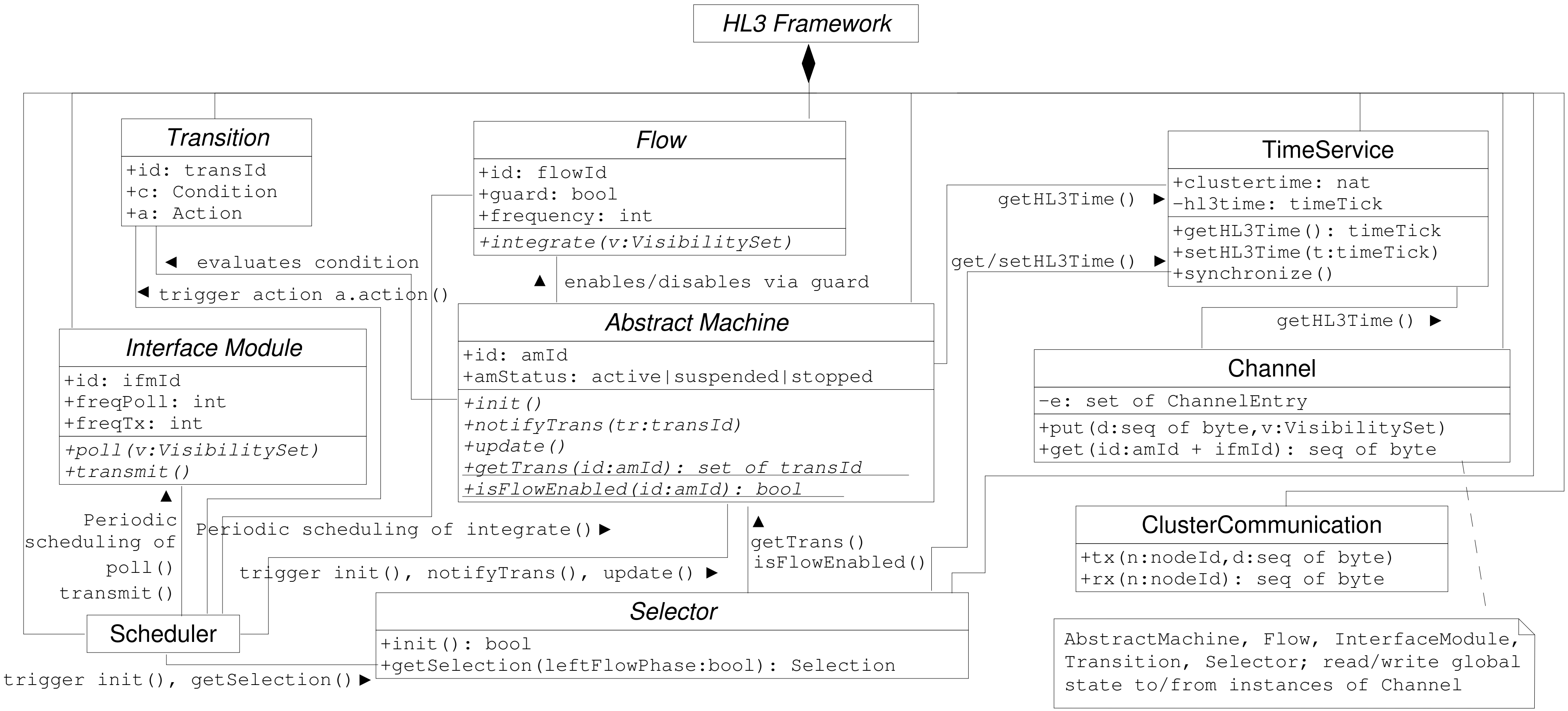
HL^3 Framework



HybridUML Models – HL^3 Framework

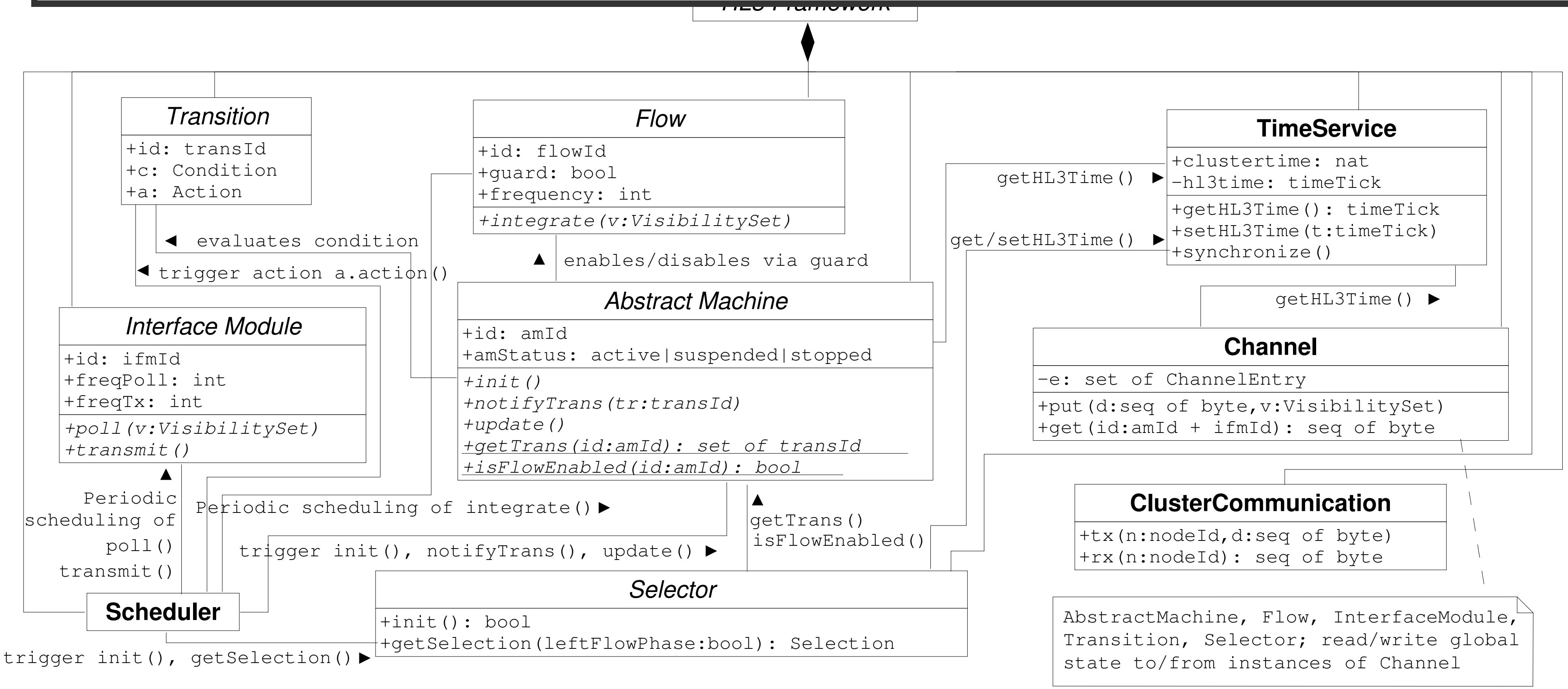
HL^3 Framework

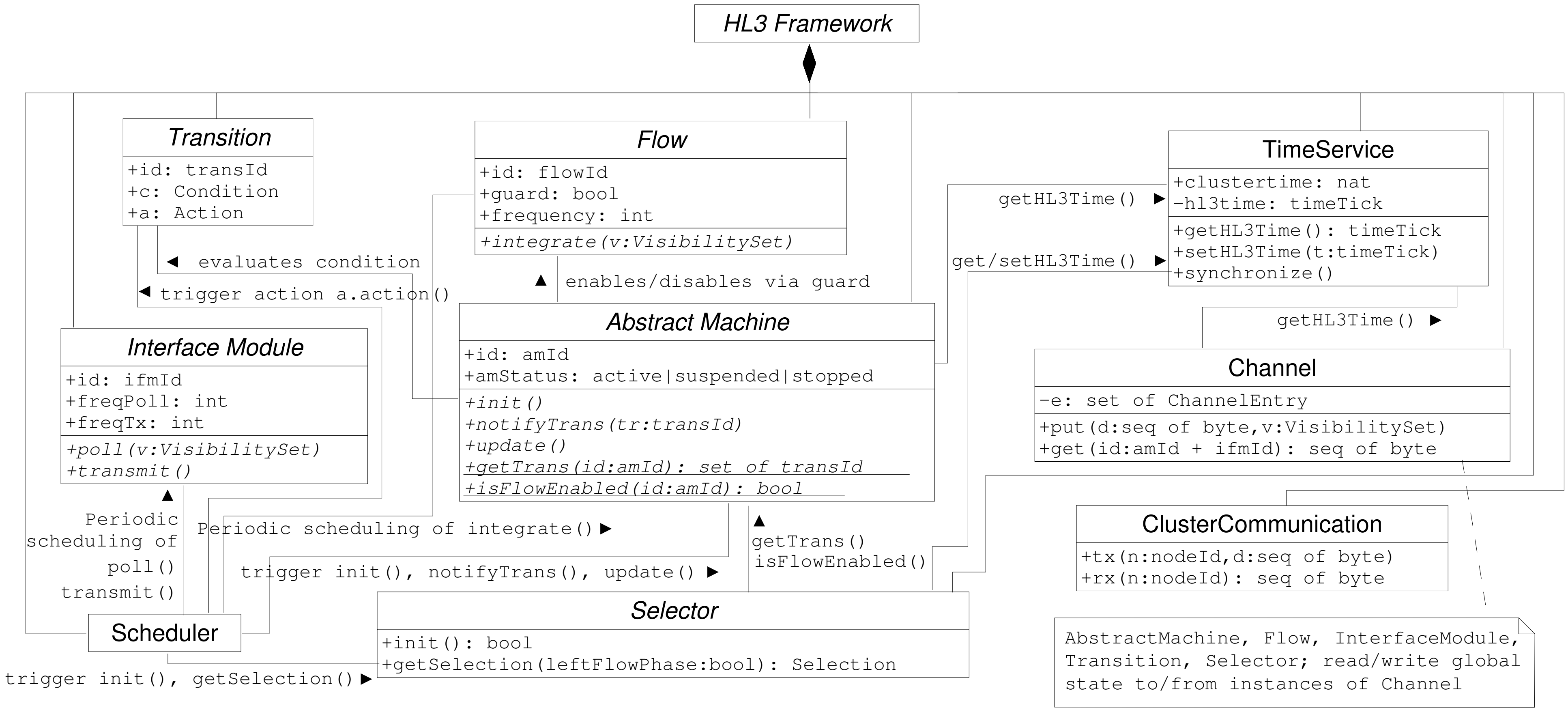
- Generic compilation target for hybrid high-level formalisms
- Development and test of embedded applications
- Transformation of high-level models (e.g. HybridUML specifications) $\longrightarrow$ executable code
- Executable code defines formal semantics, based on:
  - Runtime environment
  - Design pattern


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
getHL3Time()
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

Runtime Environment

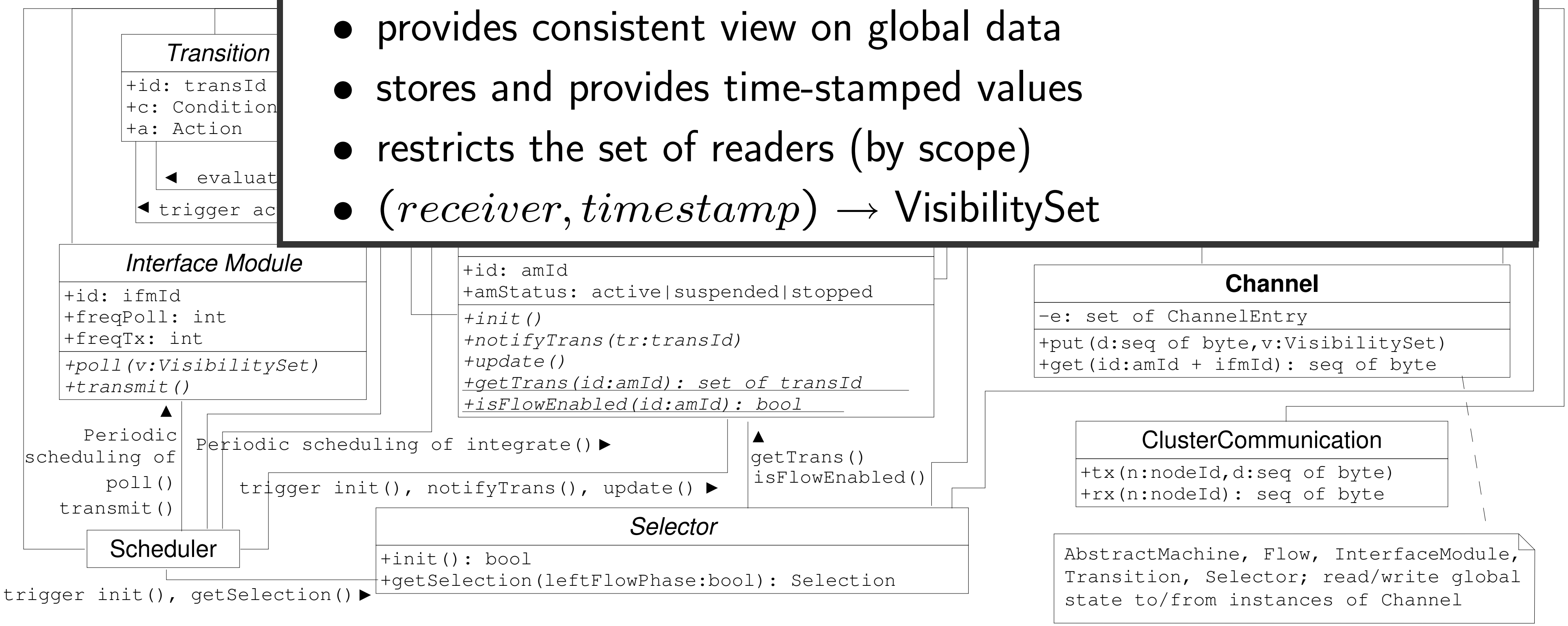
Provides low level constructs to be used by HL^3 programs: (1) Channel (2) Cluster-Communication (3) TimeService (4) Scheduler

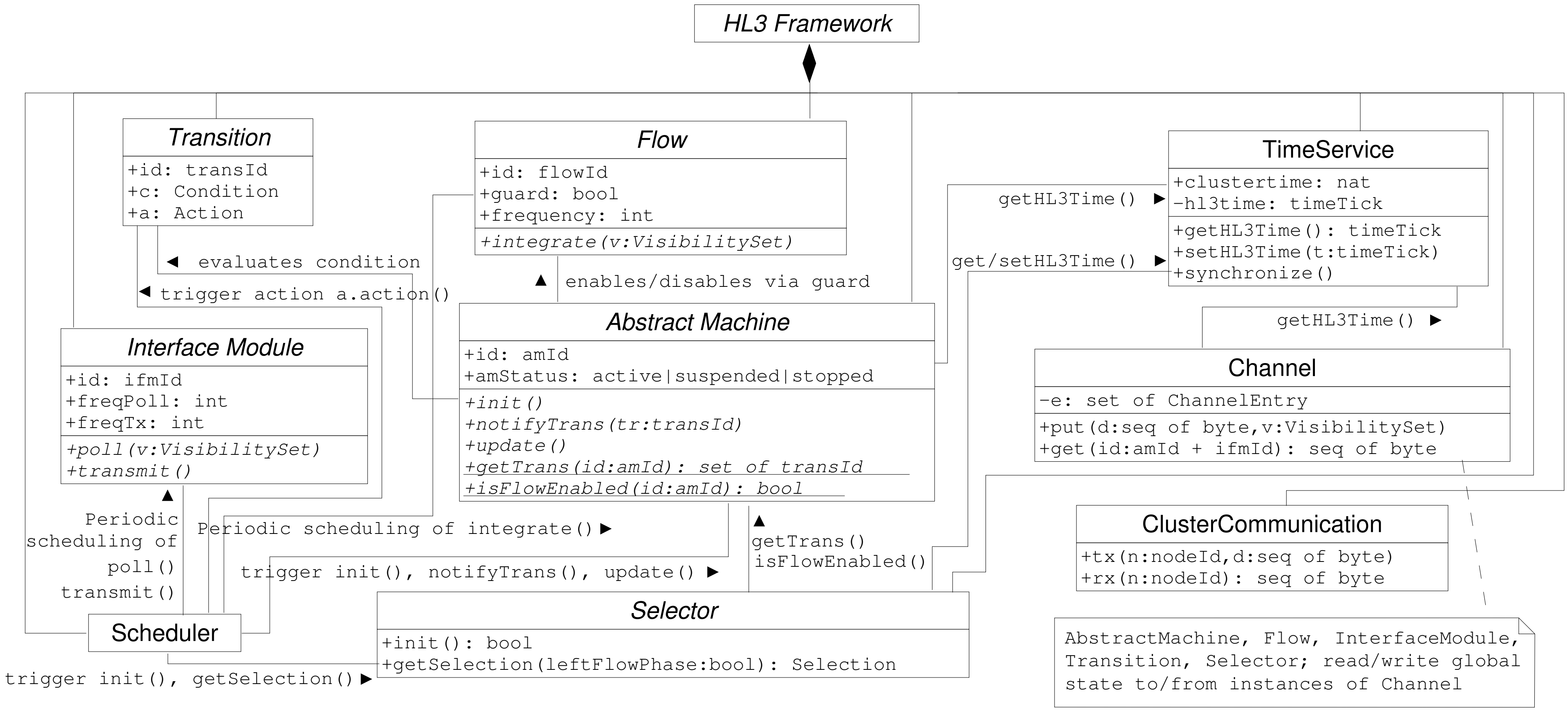


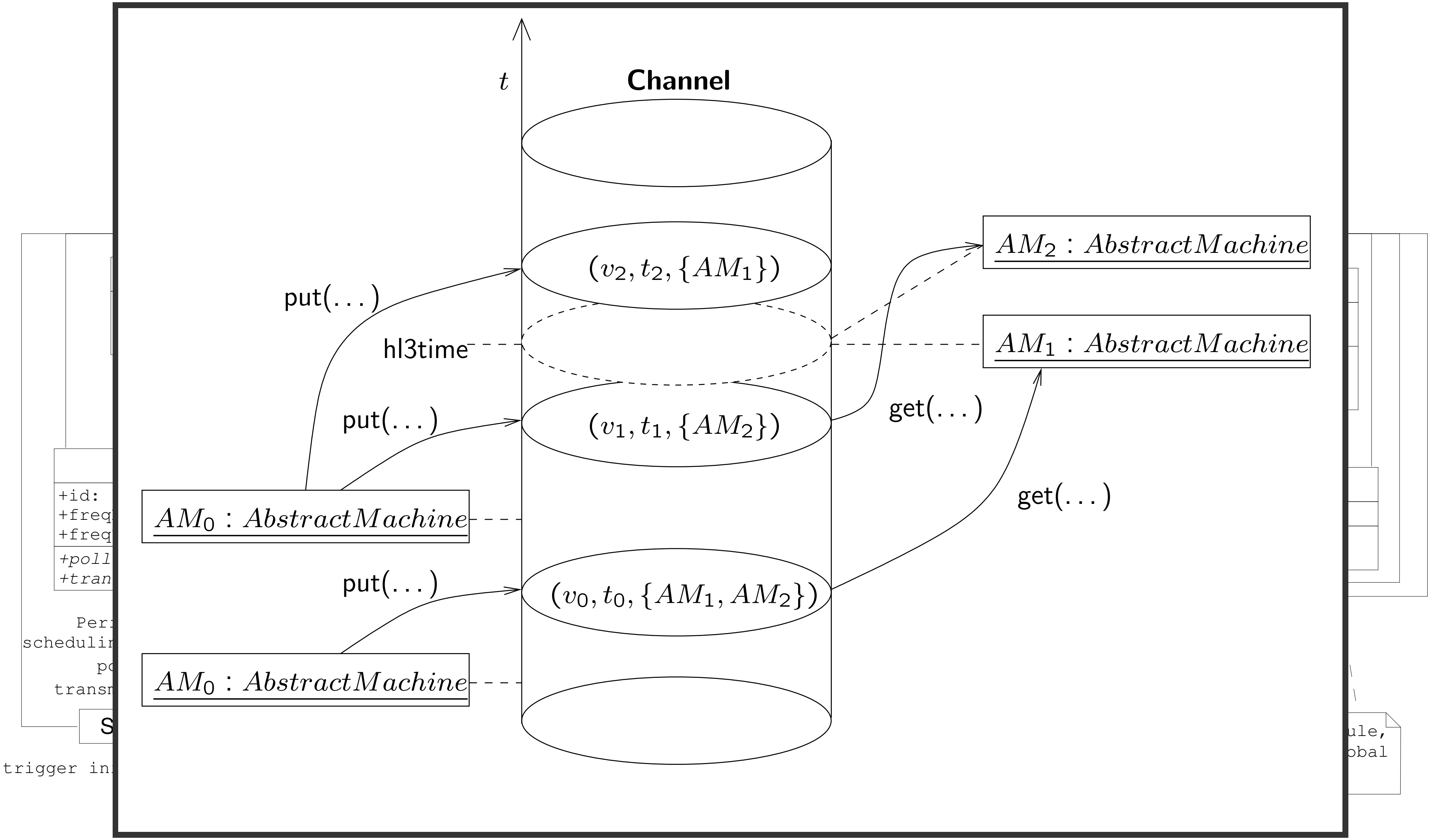

HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
getHL3Time()
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

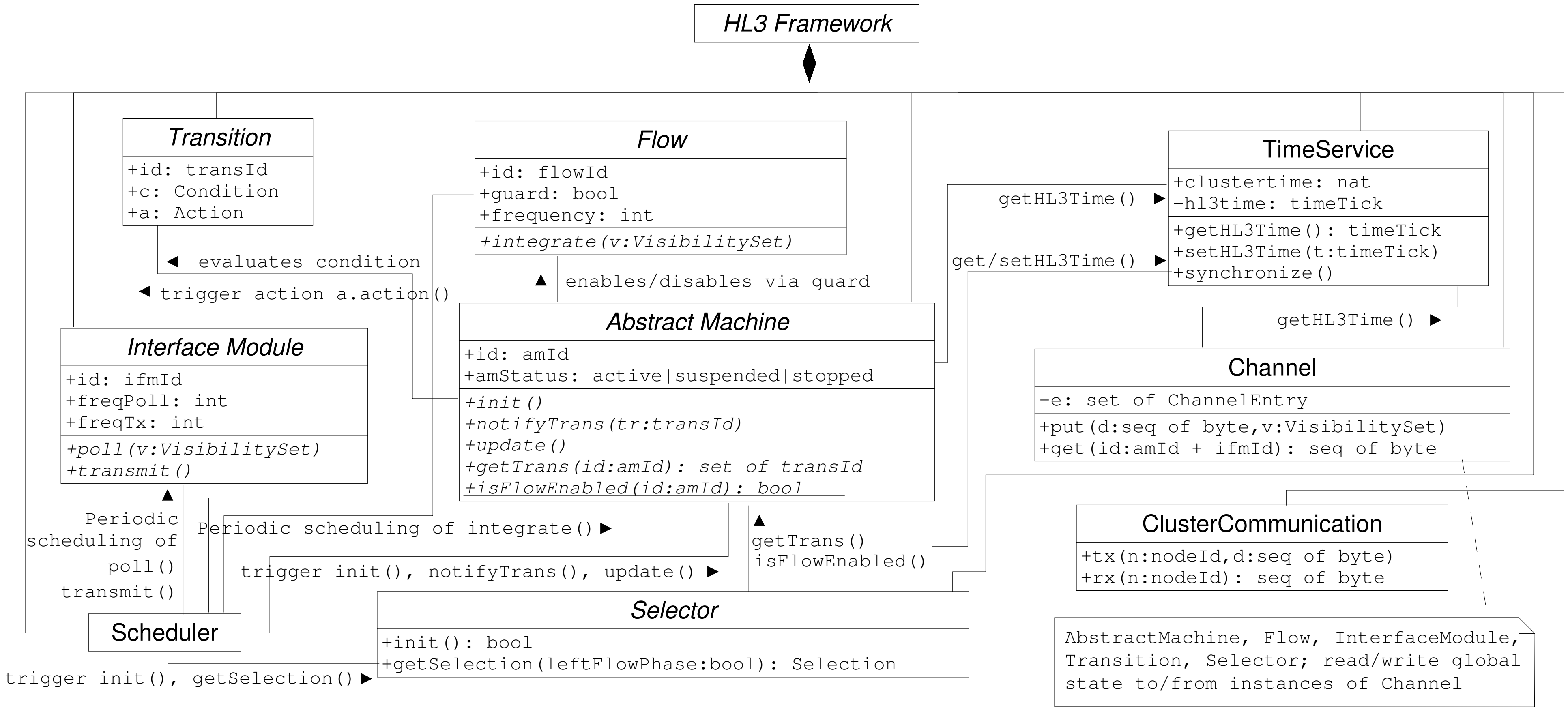
Channel:

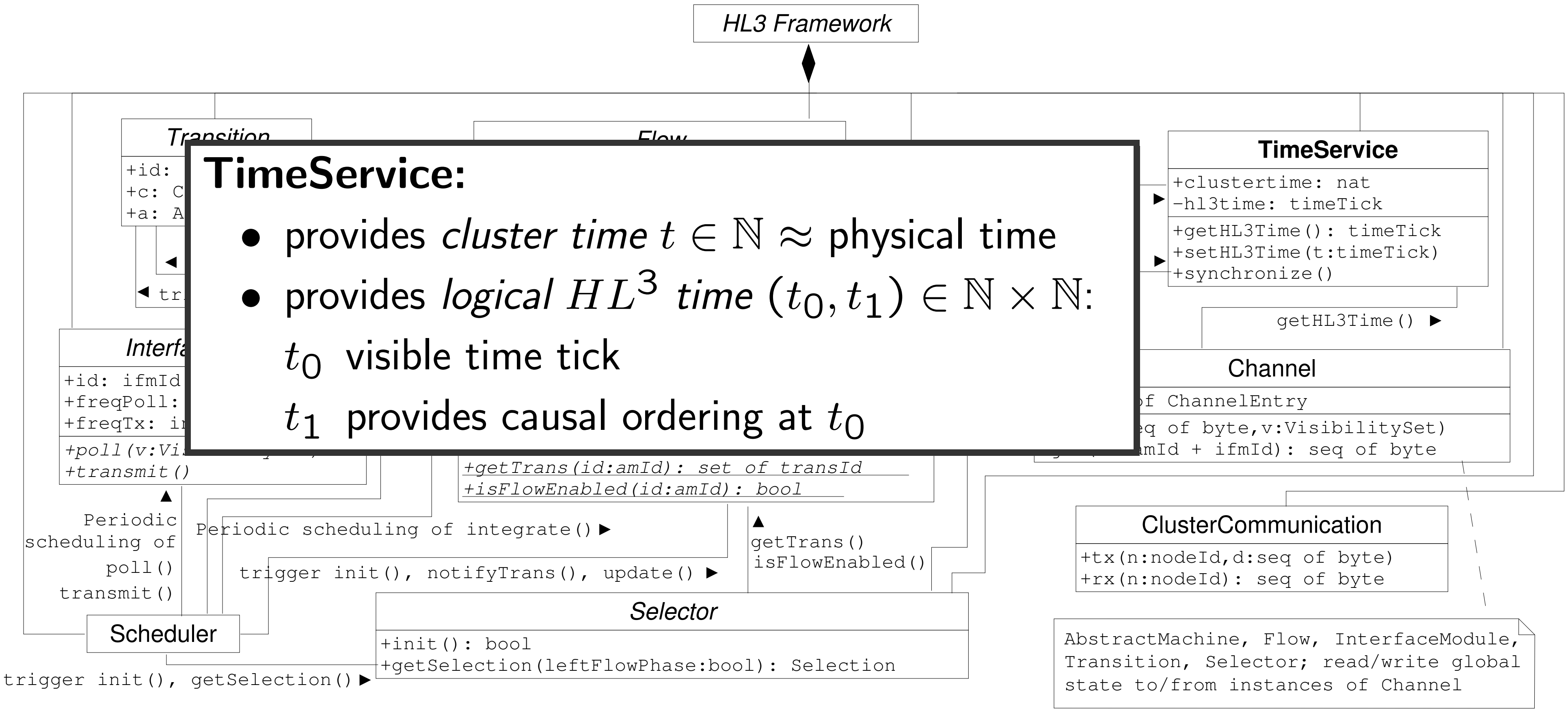
- represents a global variable or signal (of the high-level model)
- provides consistent view on global data
- stores and provides time-stamped values
- restricts the set of readers (by scope)
- $(receiver, timestamp) \rightarrow$ VisibilitySet




HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
getHL3Time()
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel


t
Channel
$(v_2, t_2, \{AM_1\})$
put(. . .)
hl3time
$AM_2 : AbstractMachine$
$AM_1 : AbstractMachine$
put(. . .)
$(v_1, t_1, \{AM_2\})$
get(. . .)
$AM_0 : AbstractMachine$
get(. . .)
put(. . .)
$(v_0, t_0, \{AM_1, AM_2\})$
$AM_0 : AbstractMachine$


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
getHL3Time()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

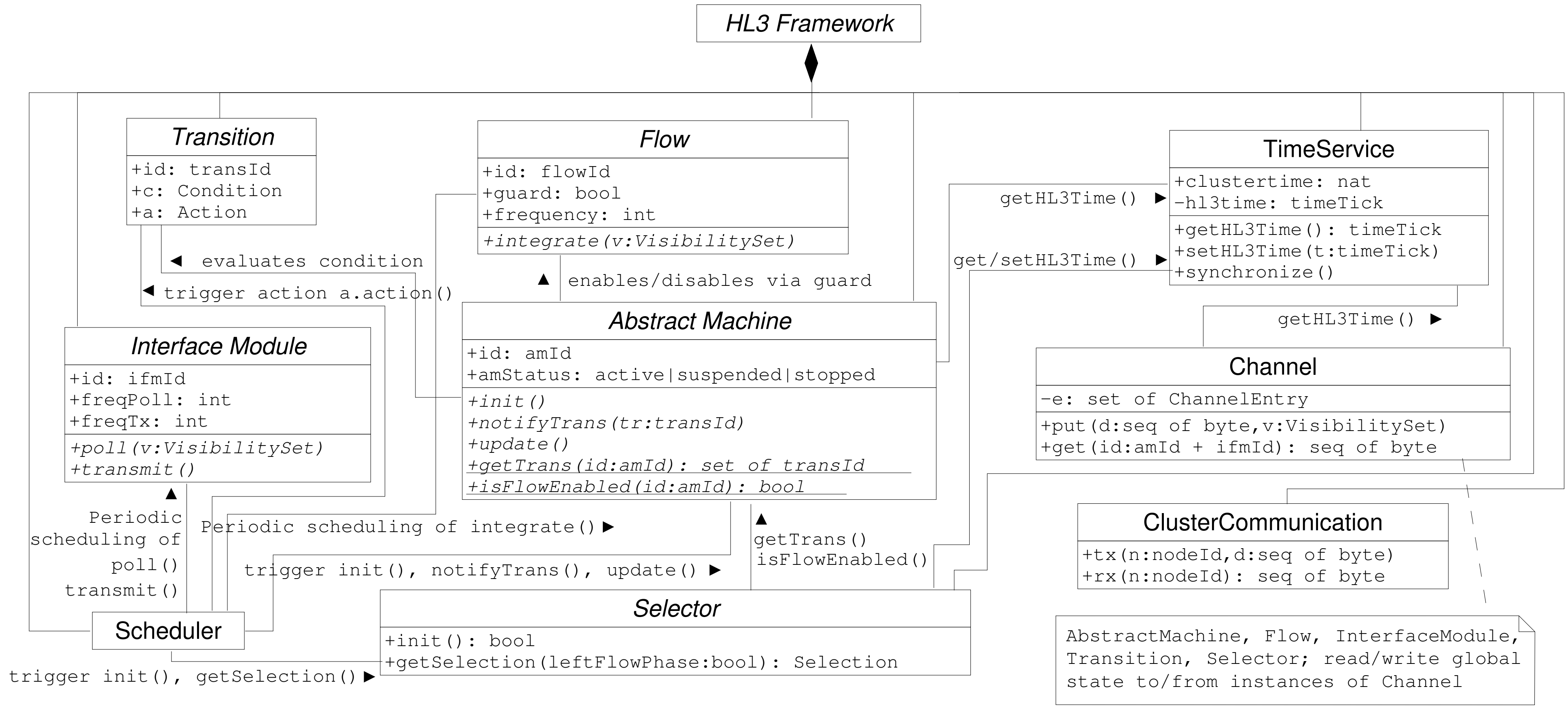


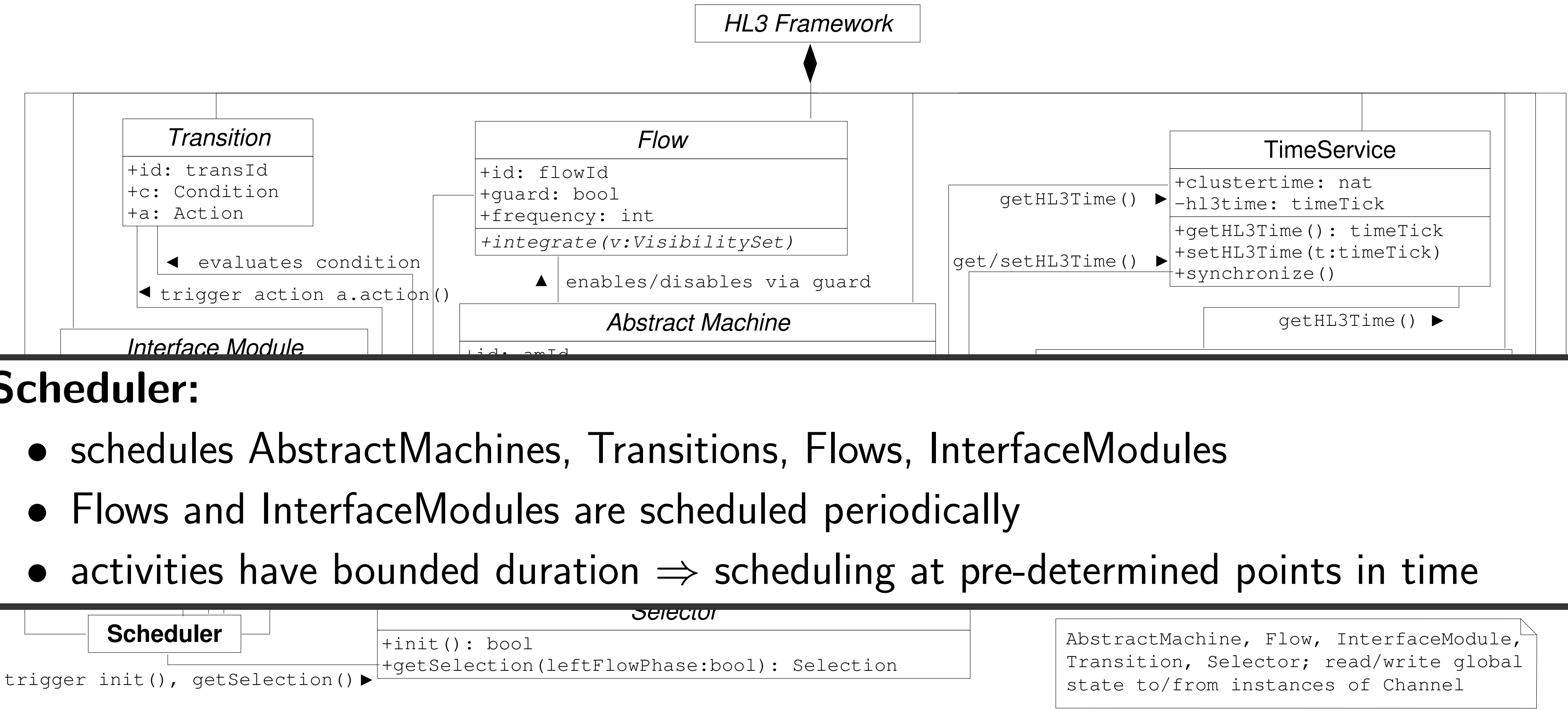
TimeService:

- provides *cluster time* $t \in \mathbb{N} \approx$ physical time
- provides *logical* HL^3 *time* $(t_0, t_1) \in \mathbb{N} \times \mathbb{N}$:

  t_0 visible time tick

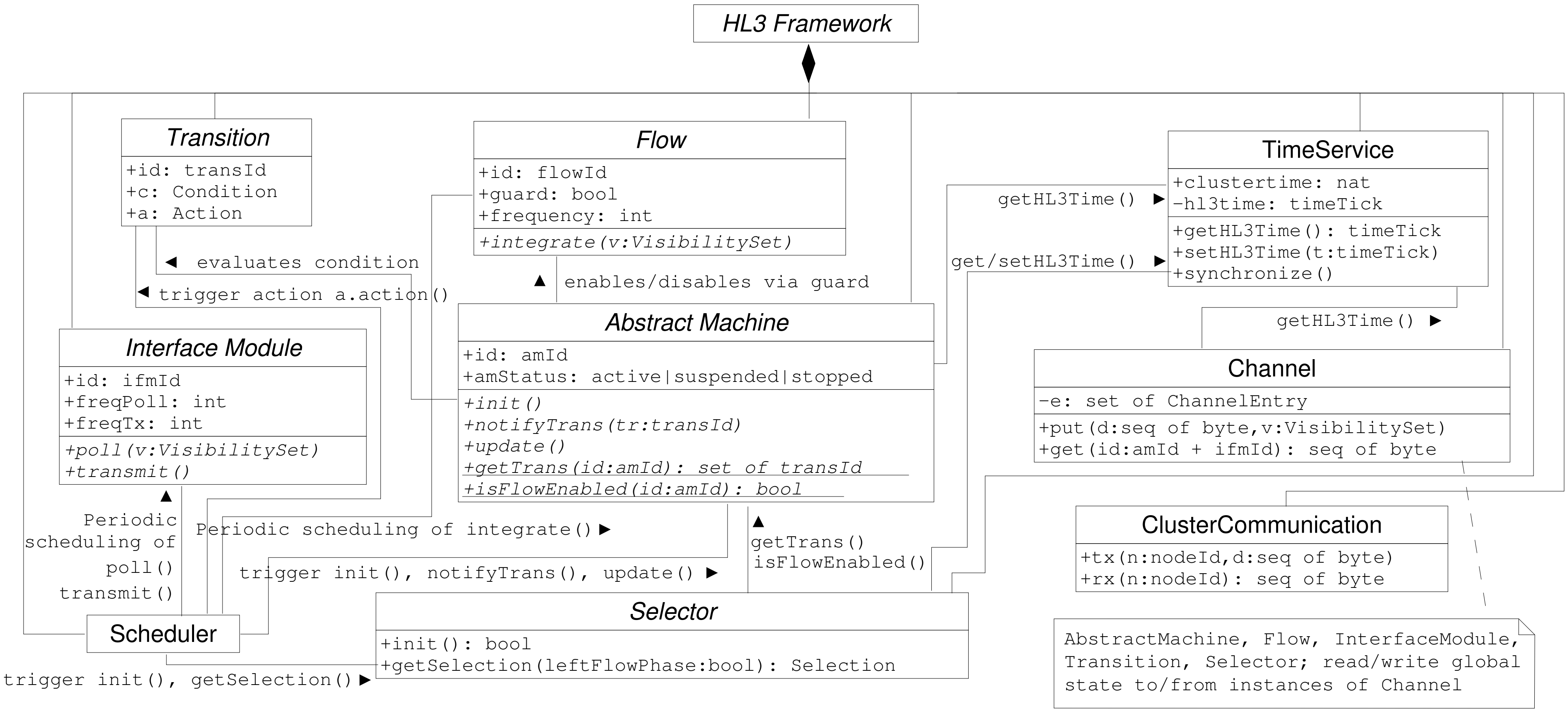
  t_1 provides causal ordering at t_0


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
getHL3Time()
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

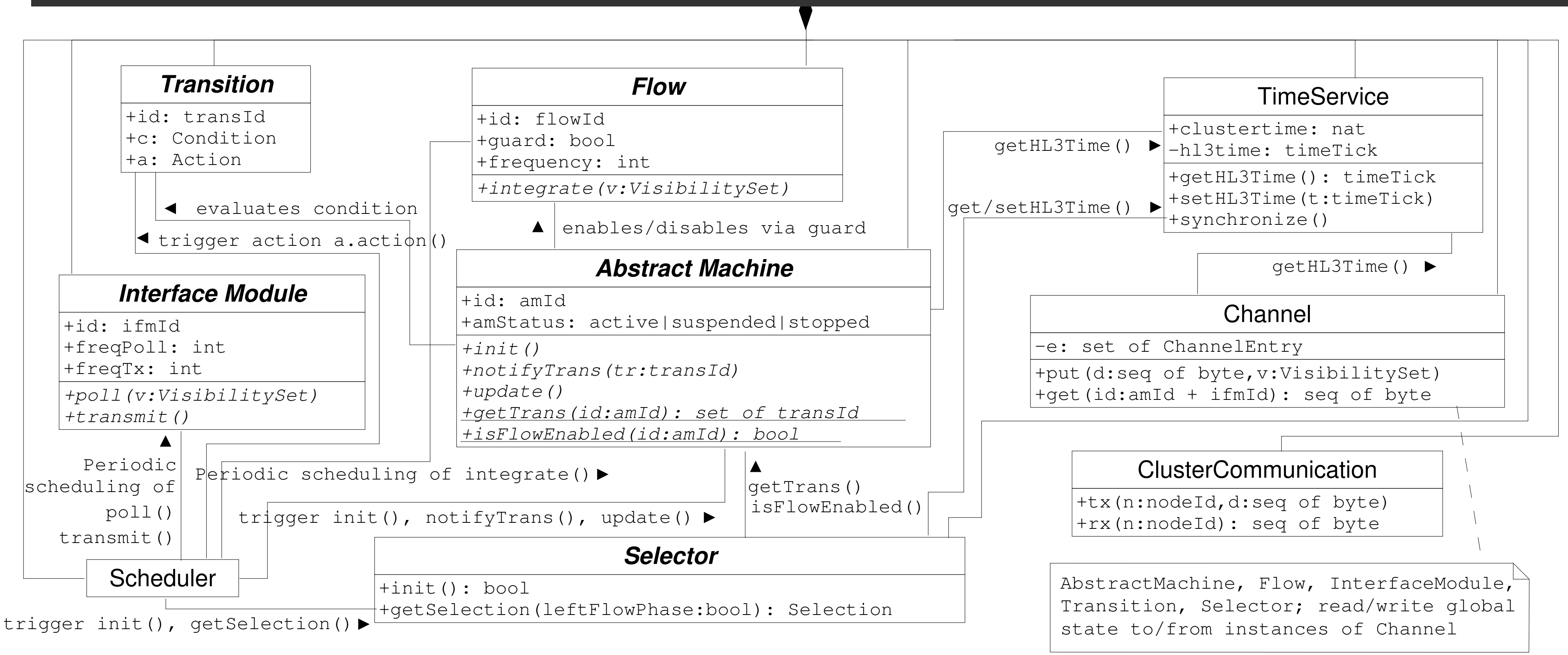


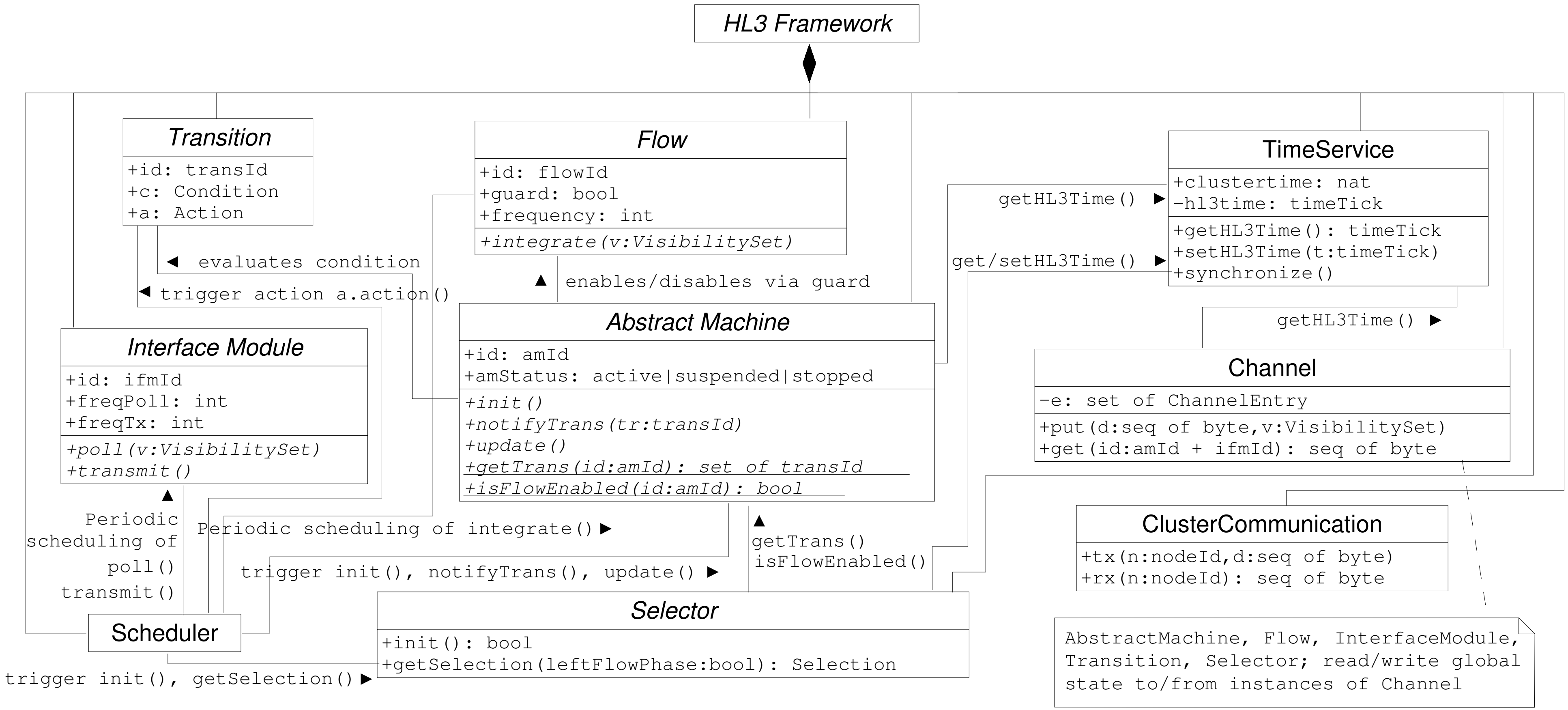
Scheduler:

- schedules AbstractMachines, Transitions, Flows, InterfaceModules
- Flows and InterfaceModules are scheduled periodically
- activities have bounded duration $\Rightarrow$ scheduling at pre-determined points in time


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
getHL3Time()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans() isFlowEnabled()
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

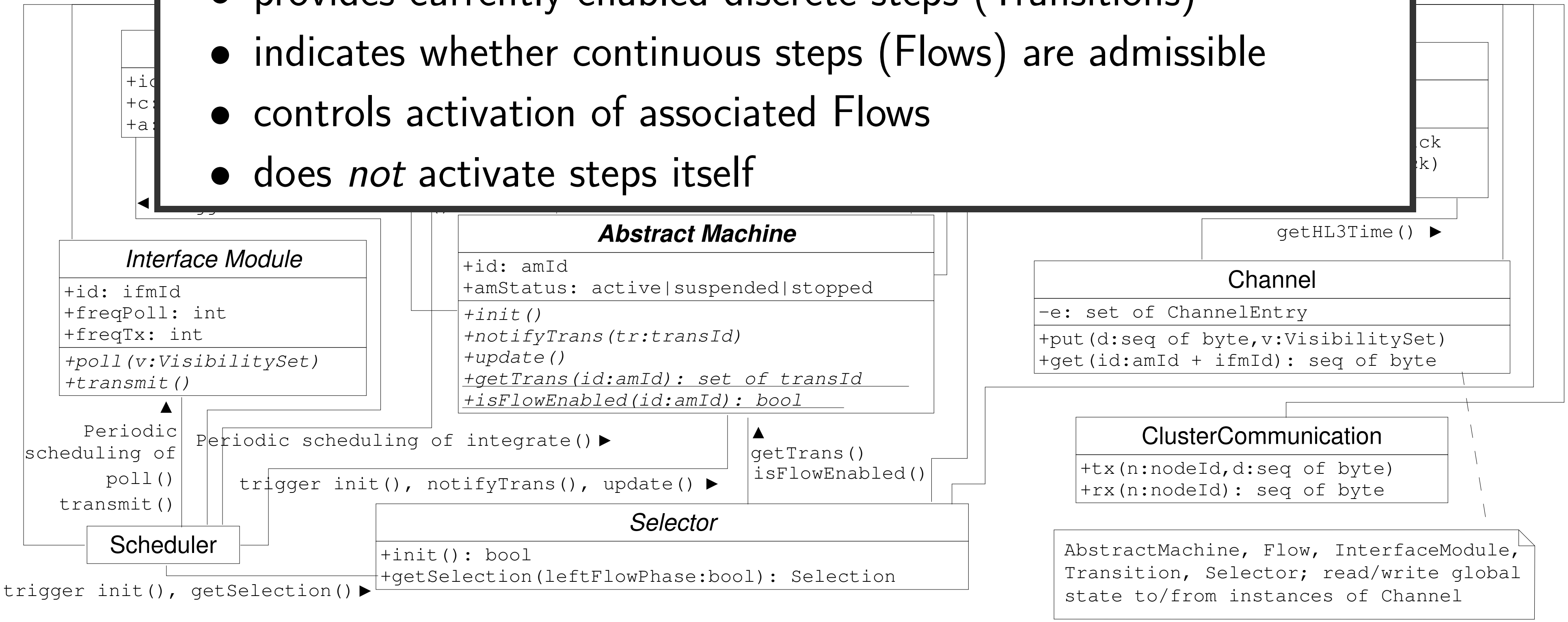
Design Pattern Proposes how to decompose (models of) a high-level formalism:
(1) AbstractMachine (2) Transition (3) Flow (4) InterfaceModule (per high-level formalism + per model)
(5) Selector (per high-level formalism)

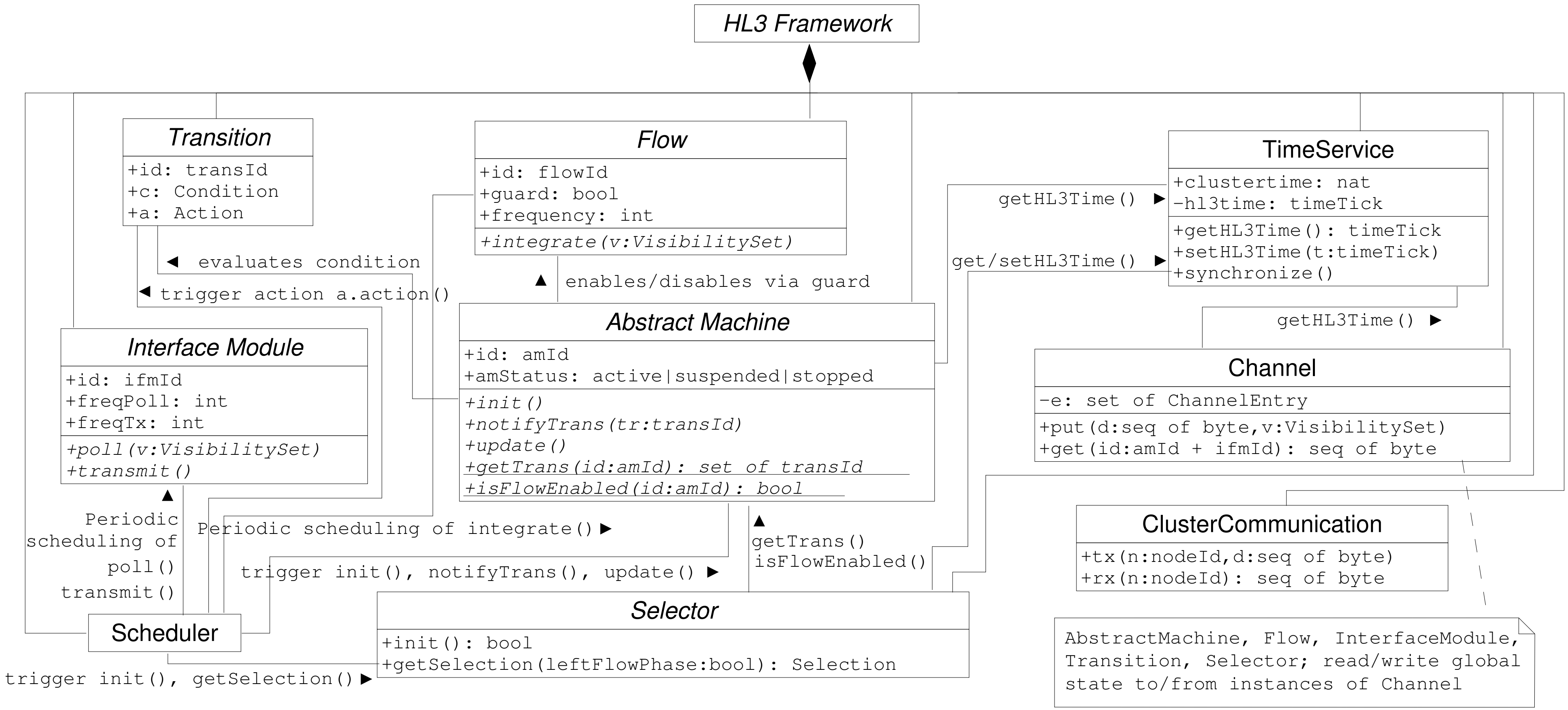


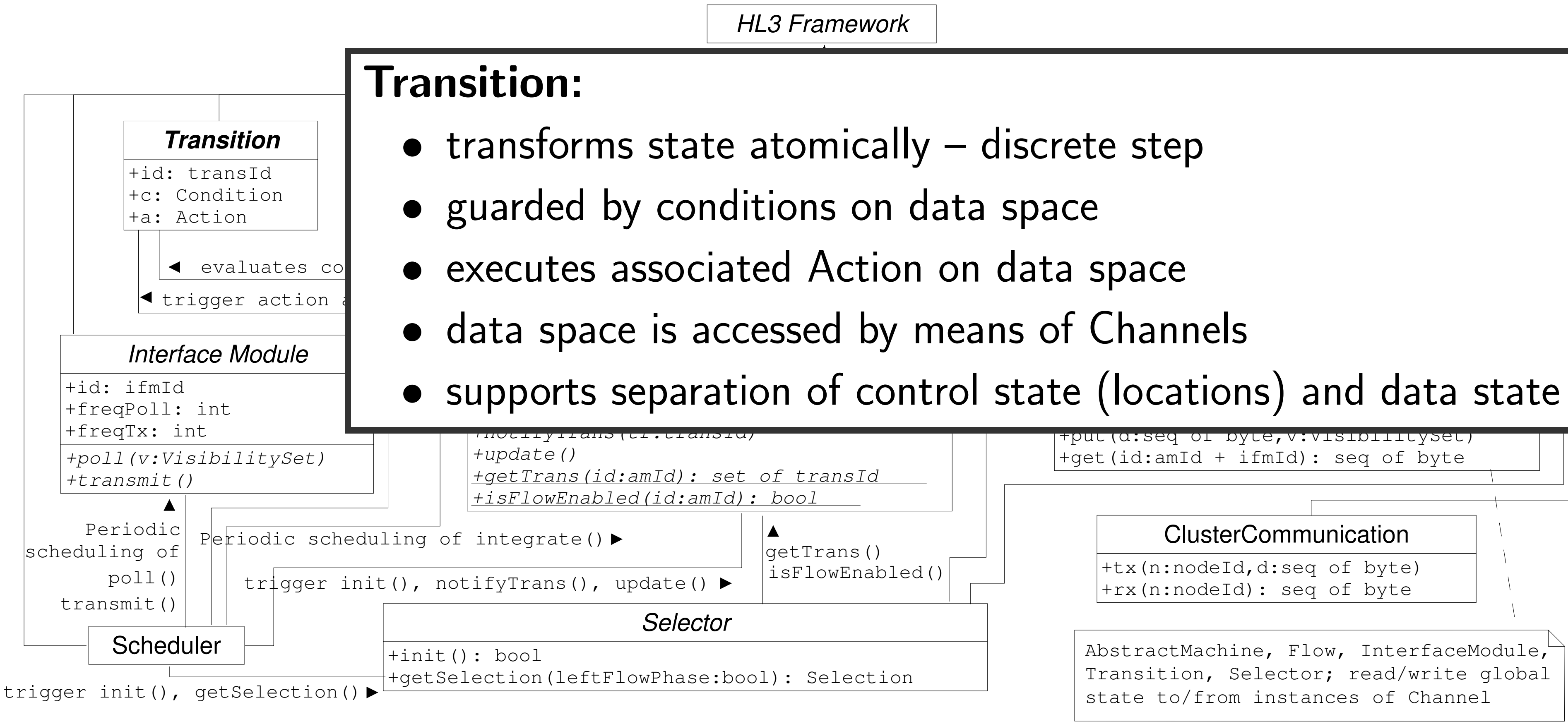

HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans() isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

AbstractMachine:

- represents local behaviour of sequential components
- encapsulates local control structure
- provides currently enabled discrete steps (Transitions)
- indicates whether continuous steps (Flows) are admissible
- controls activation of associated Flows
- does *not* activate steps itself

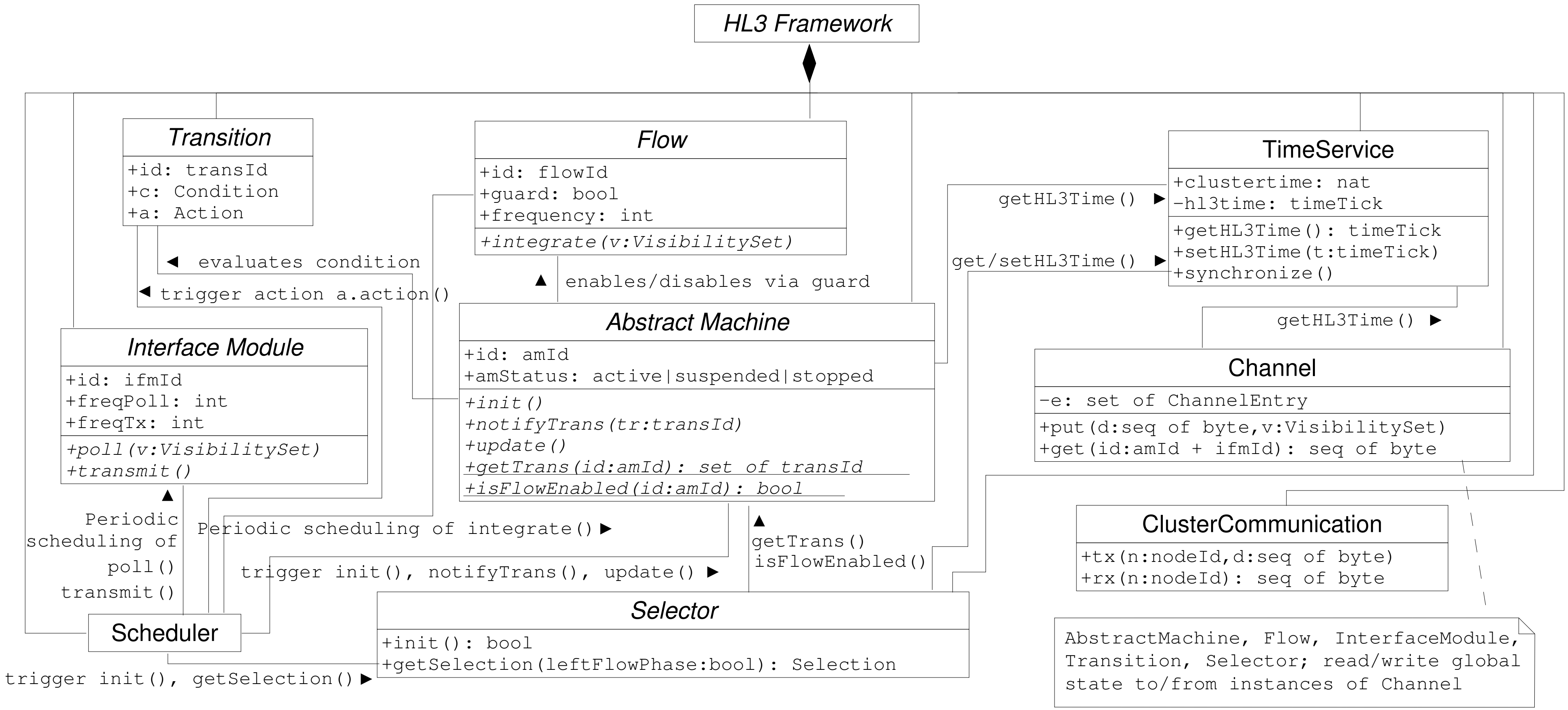


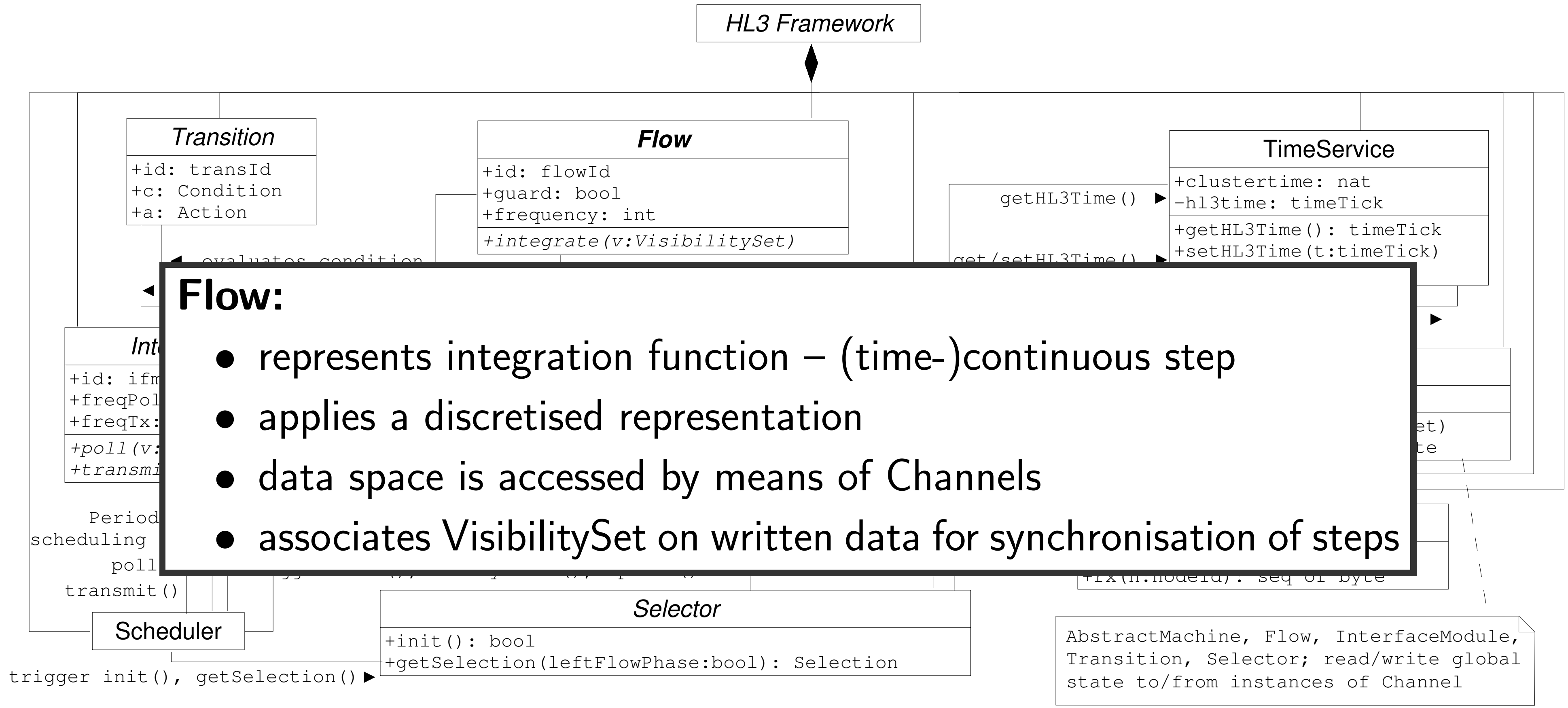

HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
getHL3Time()
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans() isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel



Transition:

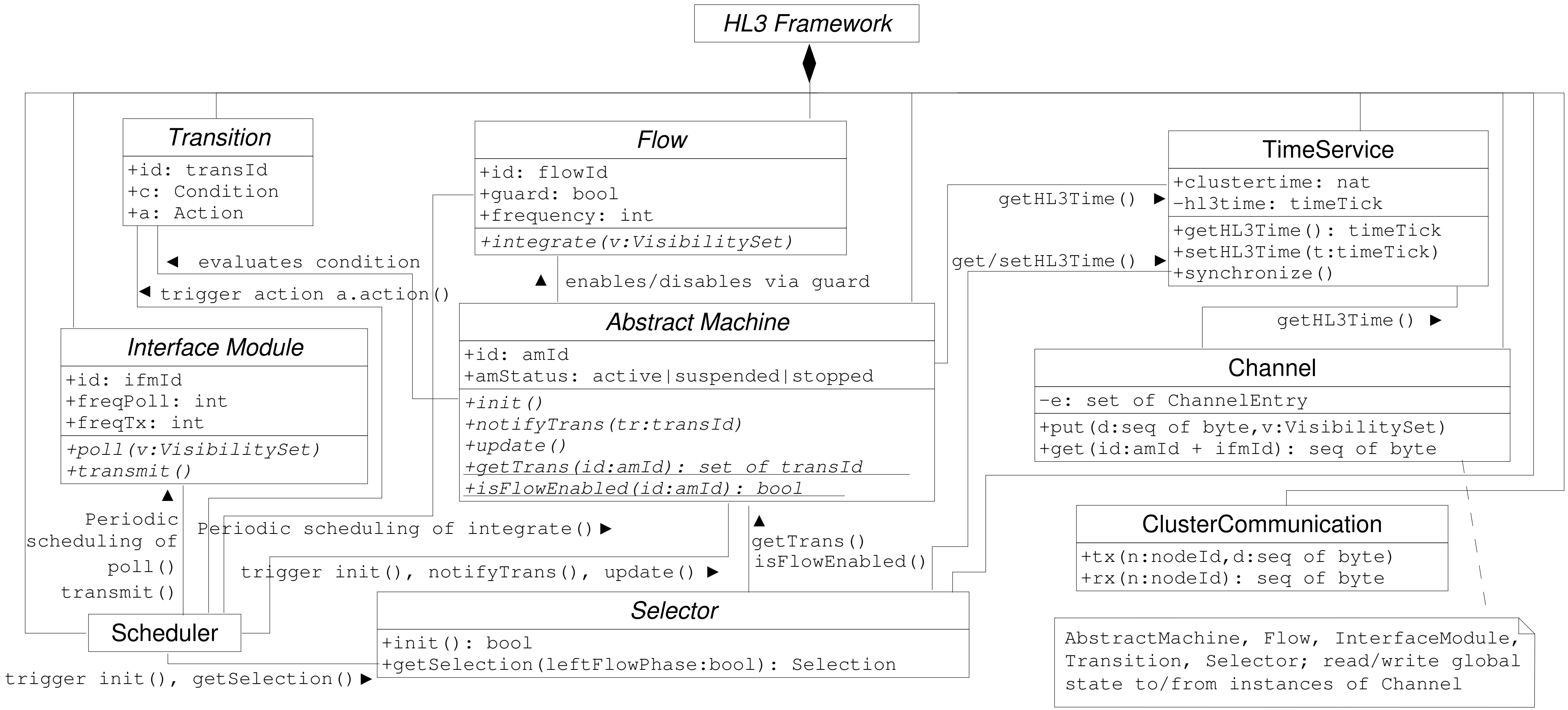
- transforms state atomically – discrete step
- guarded by conditions on data space
- executes associated Action on data space
- data space is accessed by means of Channels
- supports separation of control state (locations) and data state

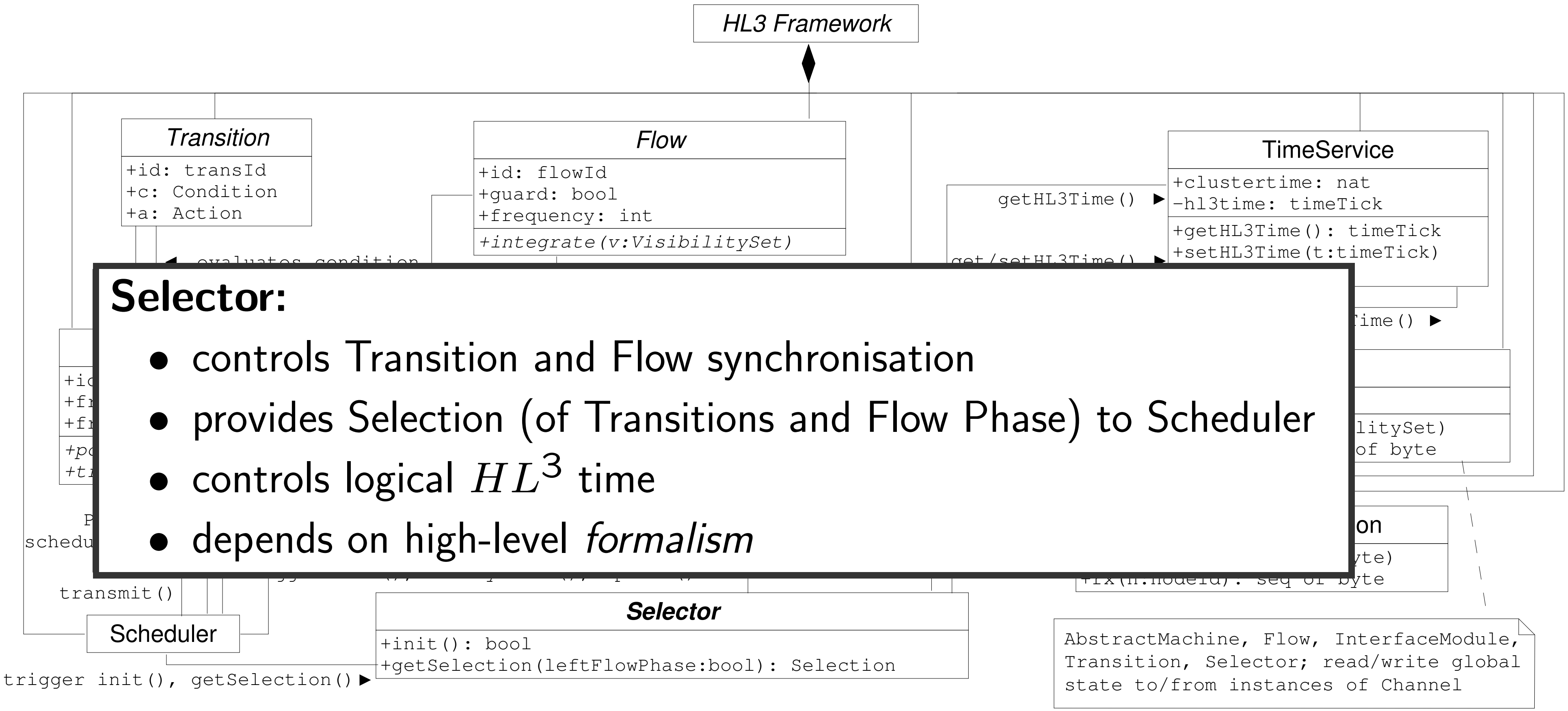

HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
getHL3Time()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel



Flow:

- represents integration function – (time-)continuous step
- applies a discretised representation
- data space is accessed by means of Channels
- associates VisibilitySet on written data for synchronisation of steps

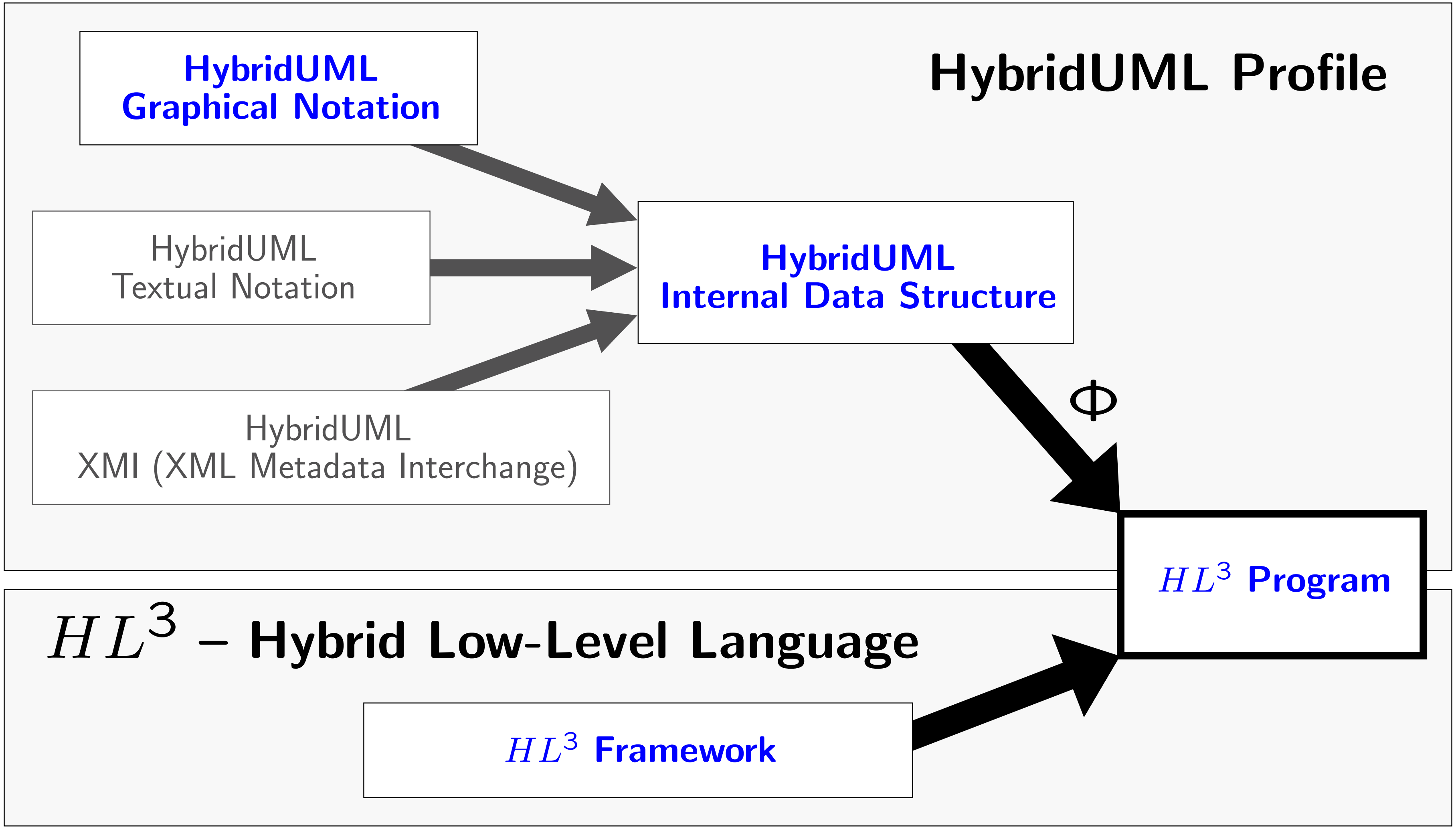

HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
getHL3Time()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans() isFlowEnabled()
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel



Selector:

- controls Transition and Flow synchronisation
- provides Selection (of Transitions and Flow Phase) to Scheduler
- controls logical HL^3 time
- depends on high-level *formalism*

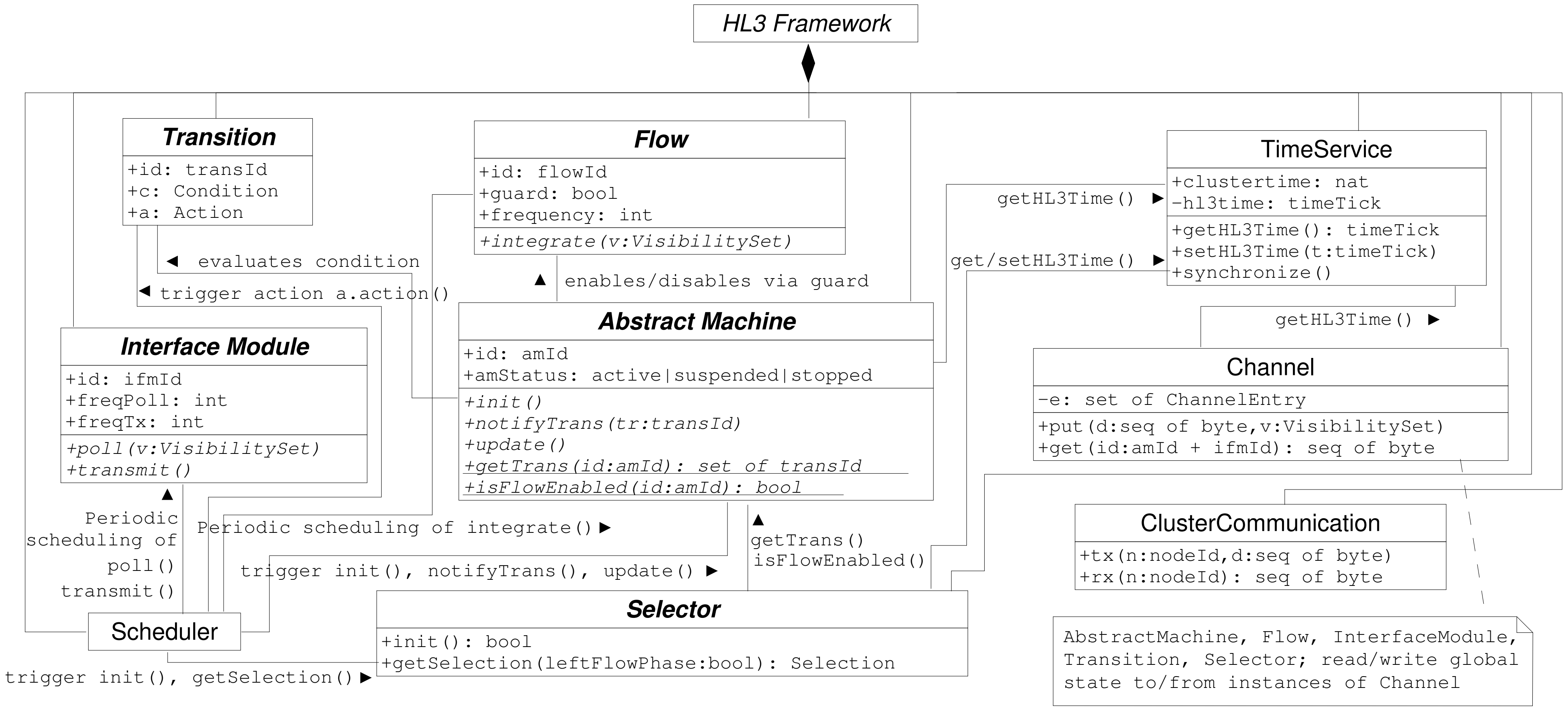
$\Phi \longrightarrow HL^3$ Program



HybridUML Models – HL^3 Program

Φ

Transforming Internal Data Structure into a HL^3 program by applying the HL^3 Design Pattern.


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
getHL3Time()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

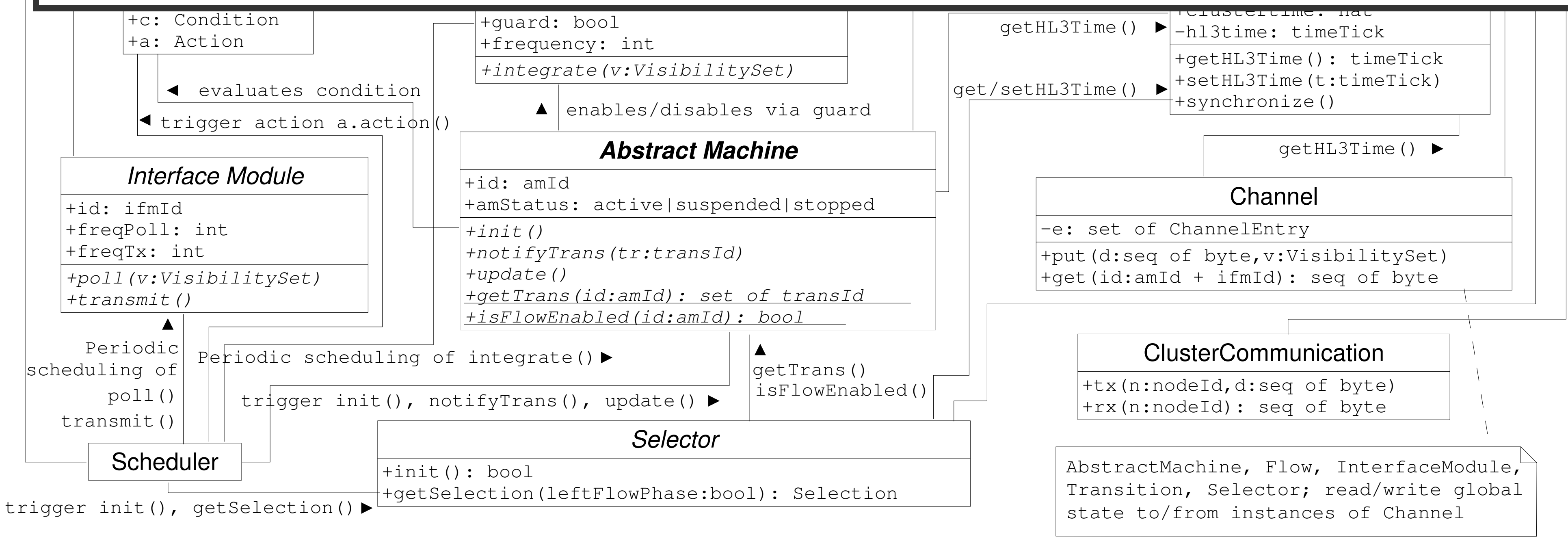
Step 1 – Generate HL^3`::AbstractMachine`

$\longrightarrow$ `AbstractMachine` class: once

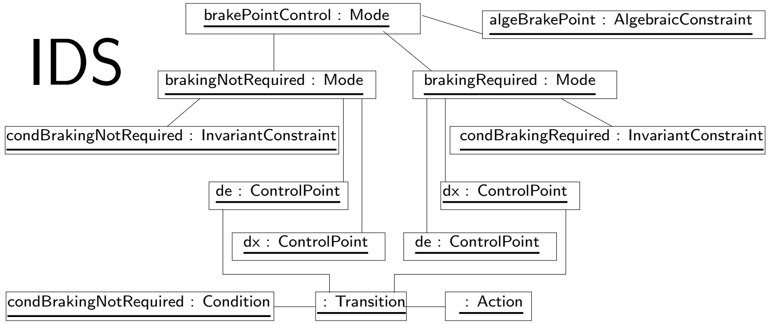
$\longrightarrow$ `notifyTrans(...)`, `update(...)`, ...: once

$\longrightarrow$ data structure for HybridUML Modes' local control structure: once

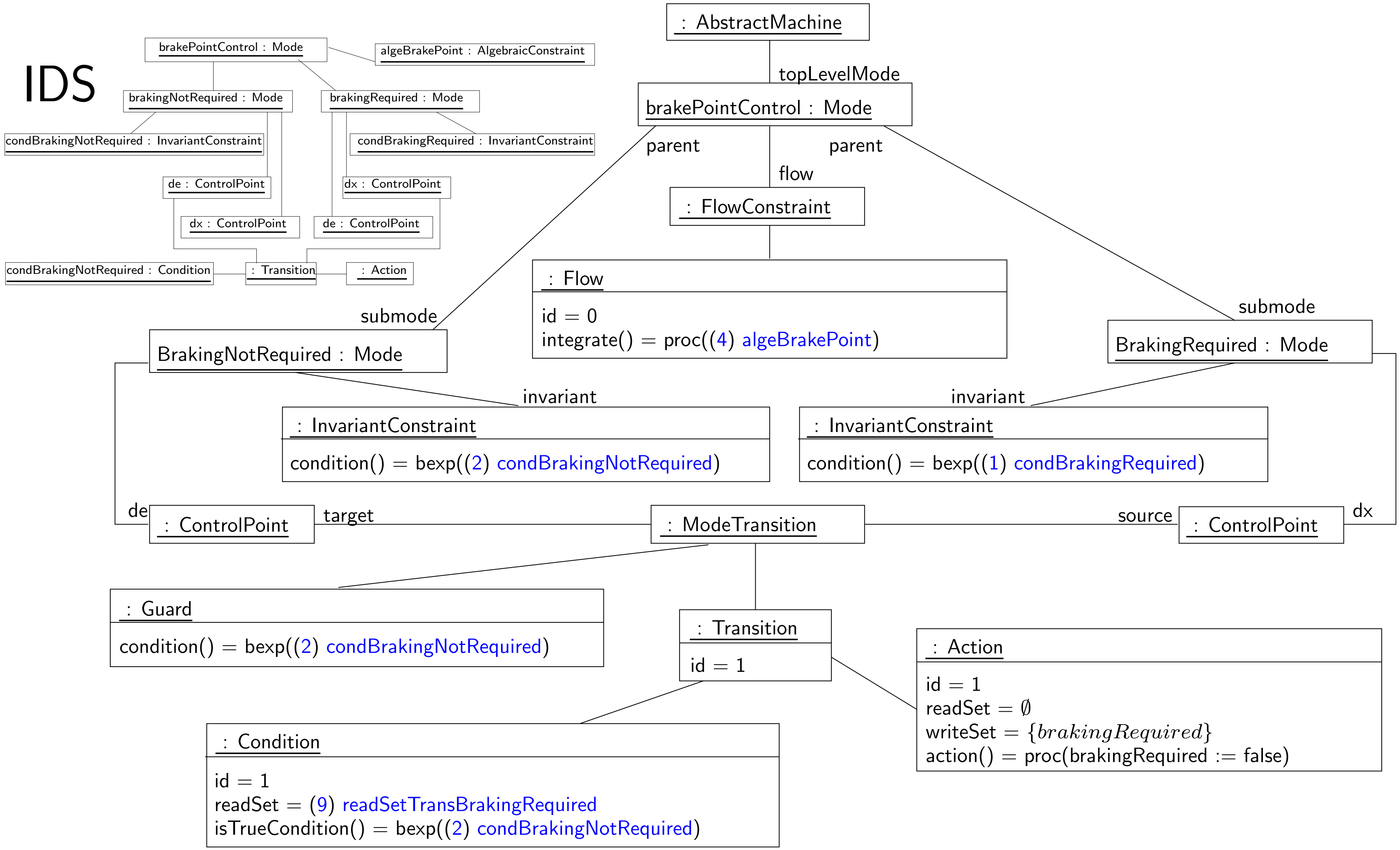
$\Longrightarrow$ AbstractMachine instances for HybridUML basic agents: for each HybridUML model



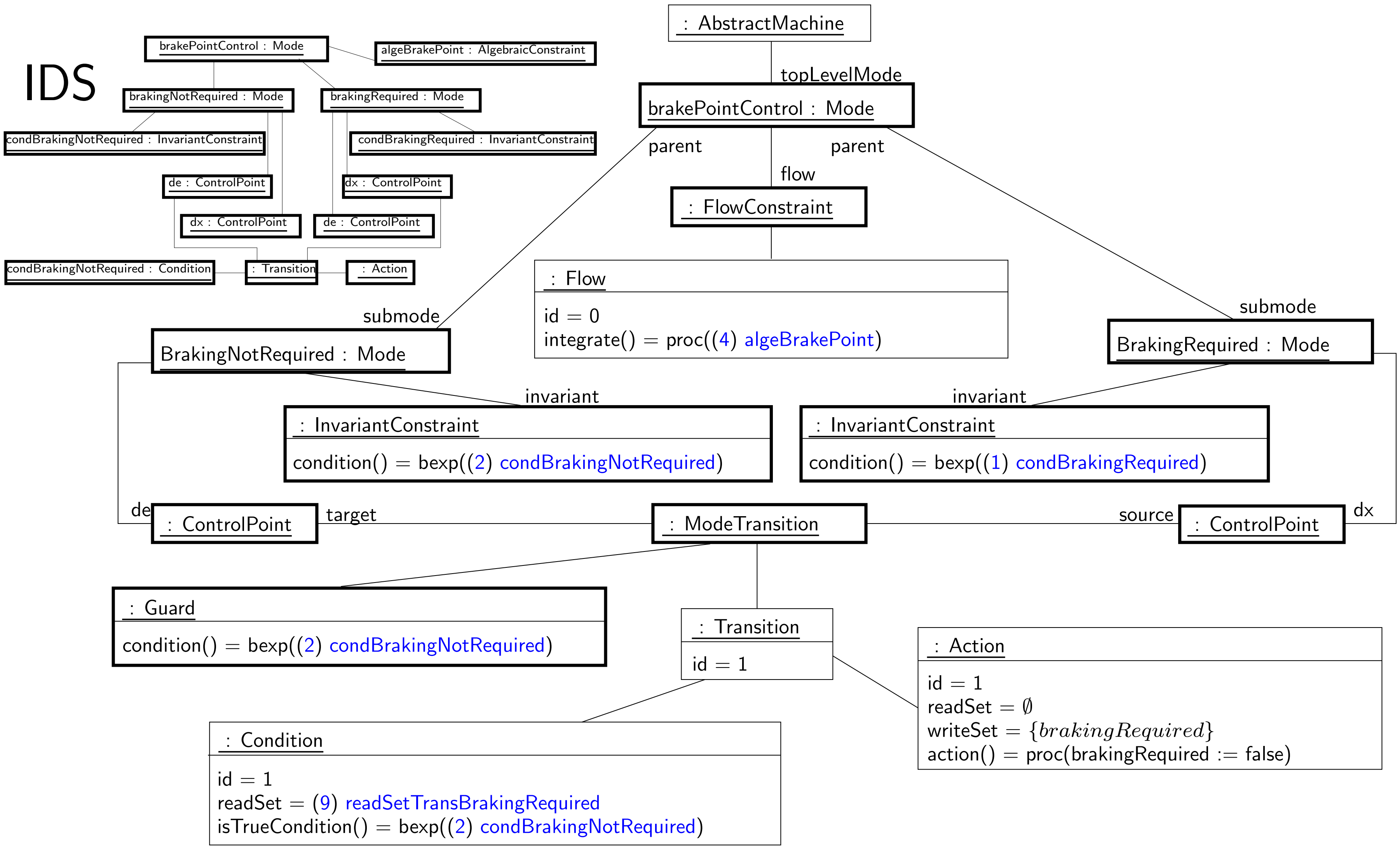
IDS



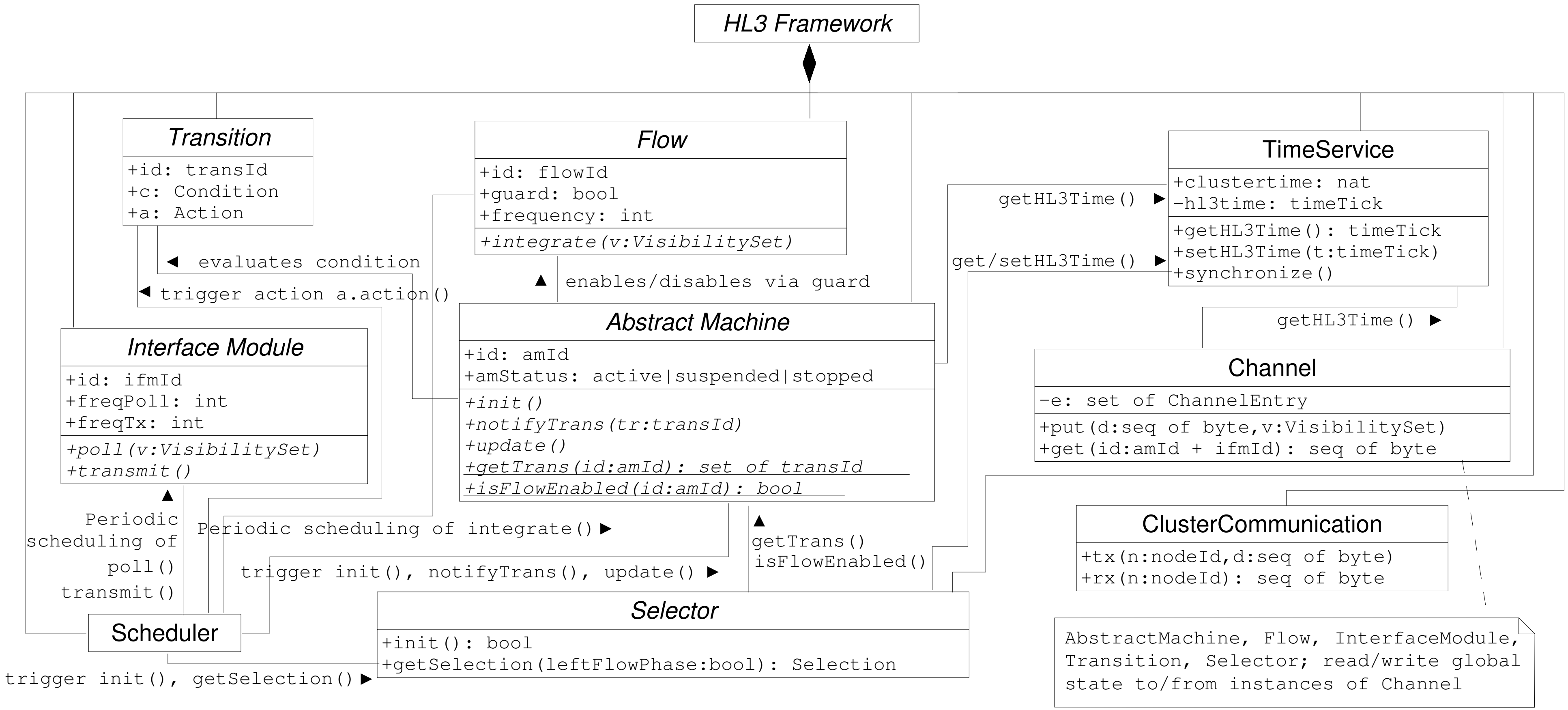
AbstractMachine example: BrakePointController

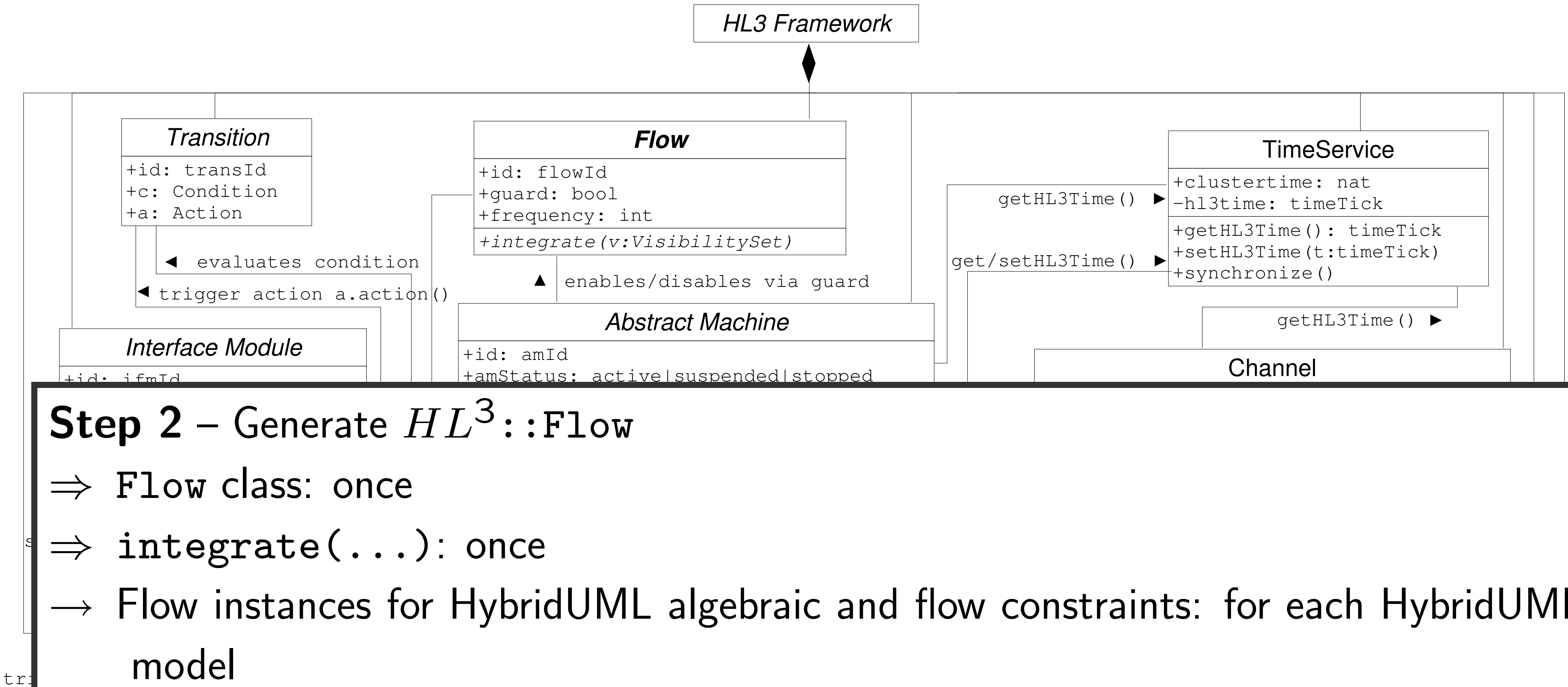


AbstractMachine example: BrakePointController



AbstractMachine example: BrakePointController


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
getHL3Time()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel



Step 2 – Generate HL^3`::Flow`

$\Rightarrow$ `Flow` class: once

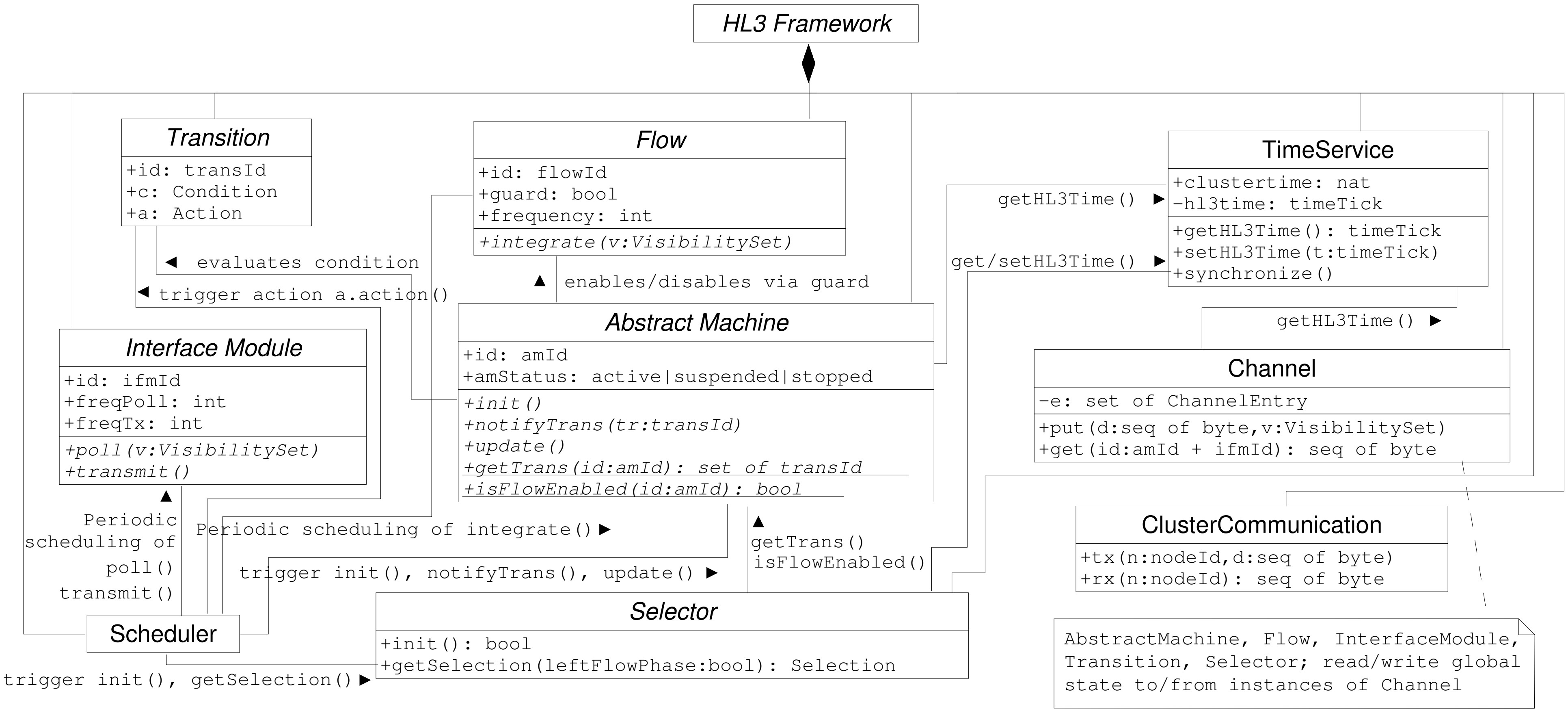
$\Rightarrow$ `integrate(...)`: once

$\rightarrow$ Flow instances for HybridUML algebraic and flow constraints: for each HybridUML model

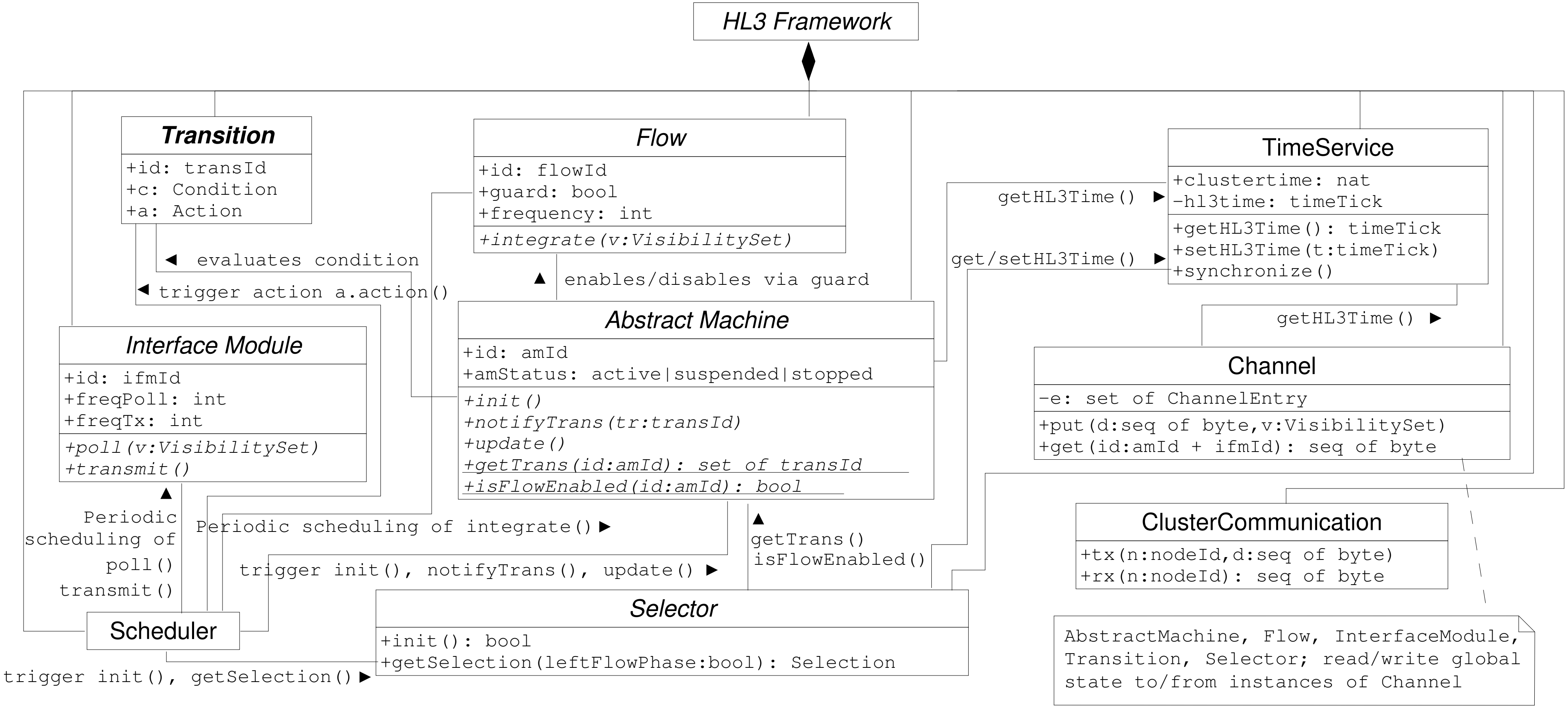
$\Rightarrow$ integration operations for each instance: for each HybridUML model

```
class Flow {
public:
  static void flowBrakePoint1 (const VisibilitySet &);
  static void flowBrakePoint2 (const VisibilitySet &);
  /* ... */
private:
  /** The flow's condition that guards if the flow is active or not. */
  bool m_guard;
  /** The associated abstract machine. */
  amId_t m_amId;
  /** The function that internally executes the integration step. */
  void (*const m_pItgrFct)(const VisibilitySet &);
public:
  /** Executes an integration step. */
  void integrate (const VisibilitySet &v) {if (m_guard) (*m_pItgrFct)(v);}
};
```

```
void Flow::flowBrakePoint1 (const VisibilitySet &visSet) {
  /* read values needed for calculation */
  RouteAtlas &ra = *(RouteAtlas*)Channel::ra.get(m_amId);
  float &v = *(float*)Channel::v.get(m_amId);
  GlobalConstants &gc = *(GlobalConstants*)Channel::gc.get(m_amId);
  /* prepare result */
  byteSeq_t byteSeq(sizeof(float));
  float &brakePoint1 = (float&)byteSeq;
  /* do calculate */
  brakePoint1
    = ra.vtp[1].x - (ra.vtp[1].v * ra.vtp[1].v - v * v) / 2 * gc.a_min;
  /* write result */
  Channel::brakePoint1.put(byteSeq, visSet);
}
```


HL3 Framework
Transition
+id: transId
+c: Condition
+a: Action
Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)
TimeService
+clustertime: nat
-hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()
getHL3Time()
get/setHL3Time()
evaluates condition
trigger action a.action()
enables/disables via guard
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool
Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()
Channel
-e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte
Periodic scheduling of poll() transmit()
Periodic scheduling of integrate()
trigger init(), notifyTrans(), update()
getTrans()
isFlowEnabled()
Scheduler
Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection
trigger init(), getSelection()
AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

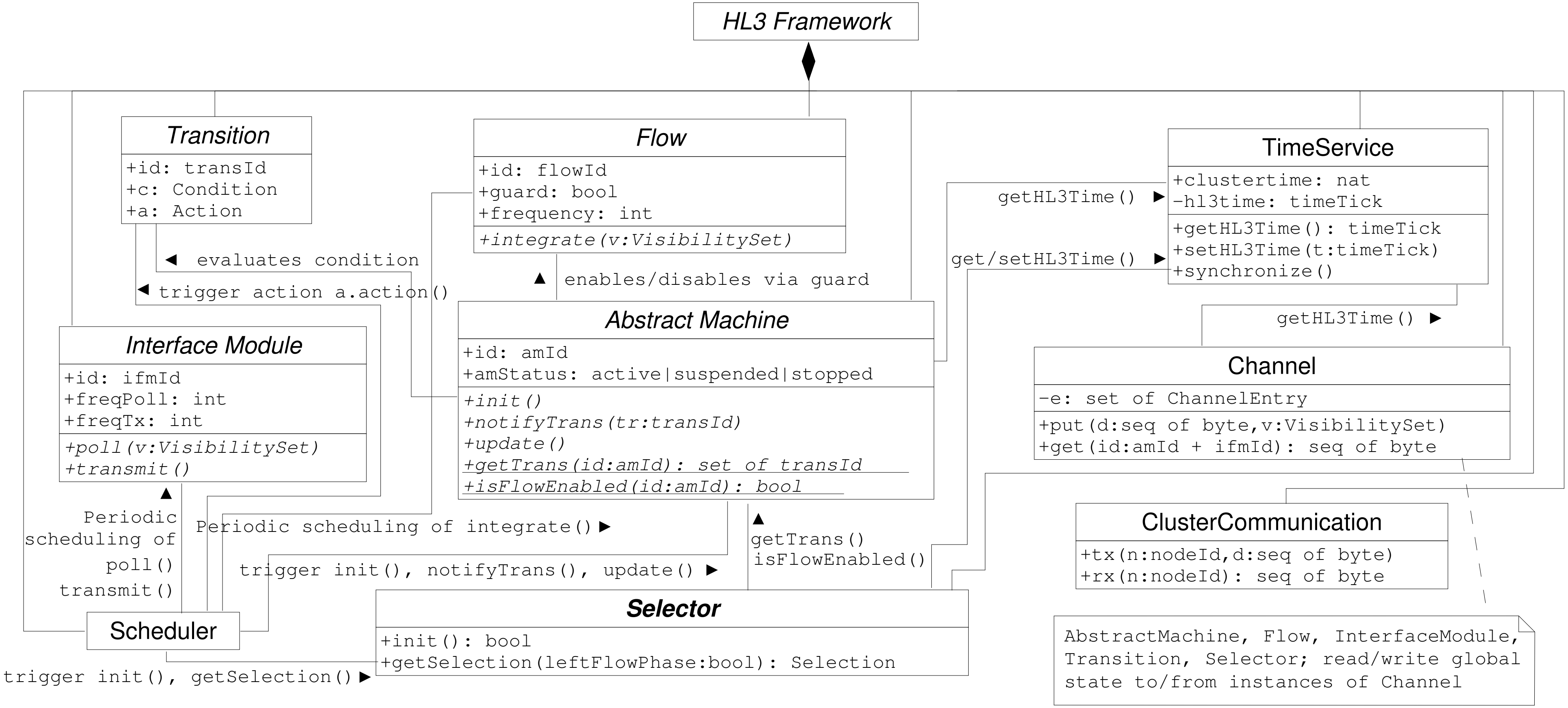
Step 3 – Generate HL^3::`Transition`...



Step 4 – Generate HL^3::Selector

→ `Selector` class: once

⇒ `getSelection(...)`, `...`: once



HybridUML Selector – `getSelection(...)`

HybridUML: interleaving of discrete and continuous steps

Discrete step
- A single basic agent fires one transition.

Continuous step
- All agents synchronously let time pass.
- Active algebraic and flow conditions are applied.
- All invariants of current mode configurations must be satisfied.

1. Active invariants are checked. $\Rightarrow e(c)$: Continuous step c admissible?

2. Set $T_{enabled}$ of enabled transitions is calculated.

3. $|T_{enabled}| > 0 \Rightarrow$ One enabled transition $t \in T_{enabled}$ is chosen non-deterministically.

4. $e(c) \wedge \neg(|T_{enabled}| > 0)$ — c is chosen.

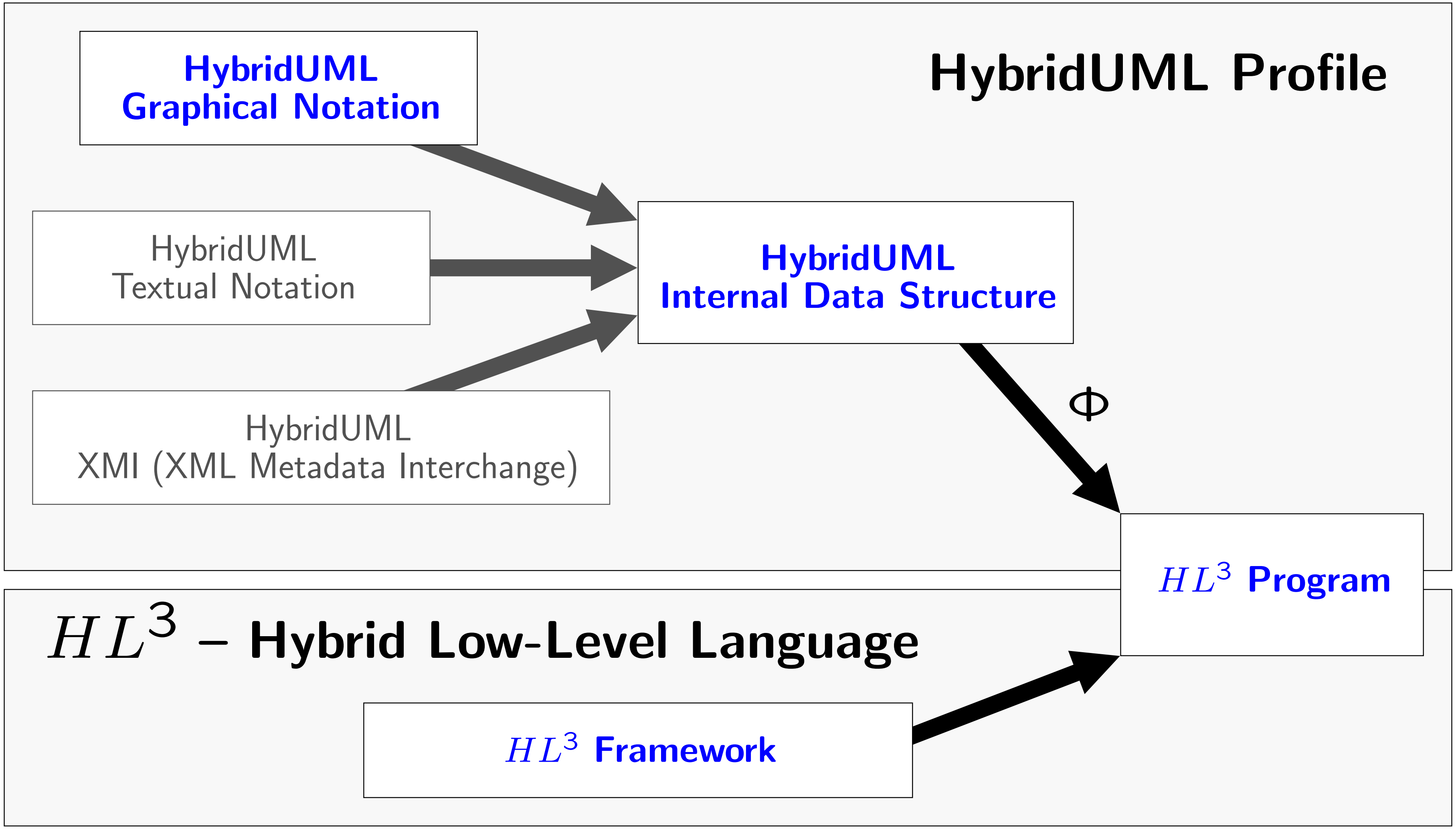
   $\neg e(c) \wedge |T_{enabled}| > 0$ — t is chosen.

   $e(c) \wedge (|T_{enabled}| > 0)$ — Non-deterministic choice of t or c.

   $\neg e(c) \wedge \neg(|T_{enabled}| > 0)$ — Deadlock.

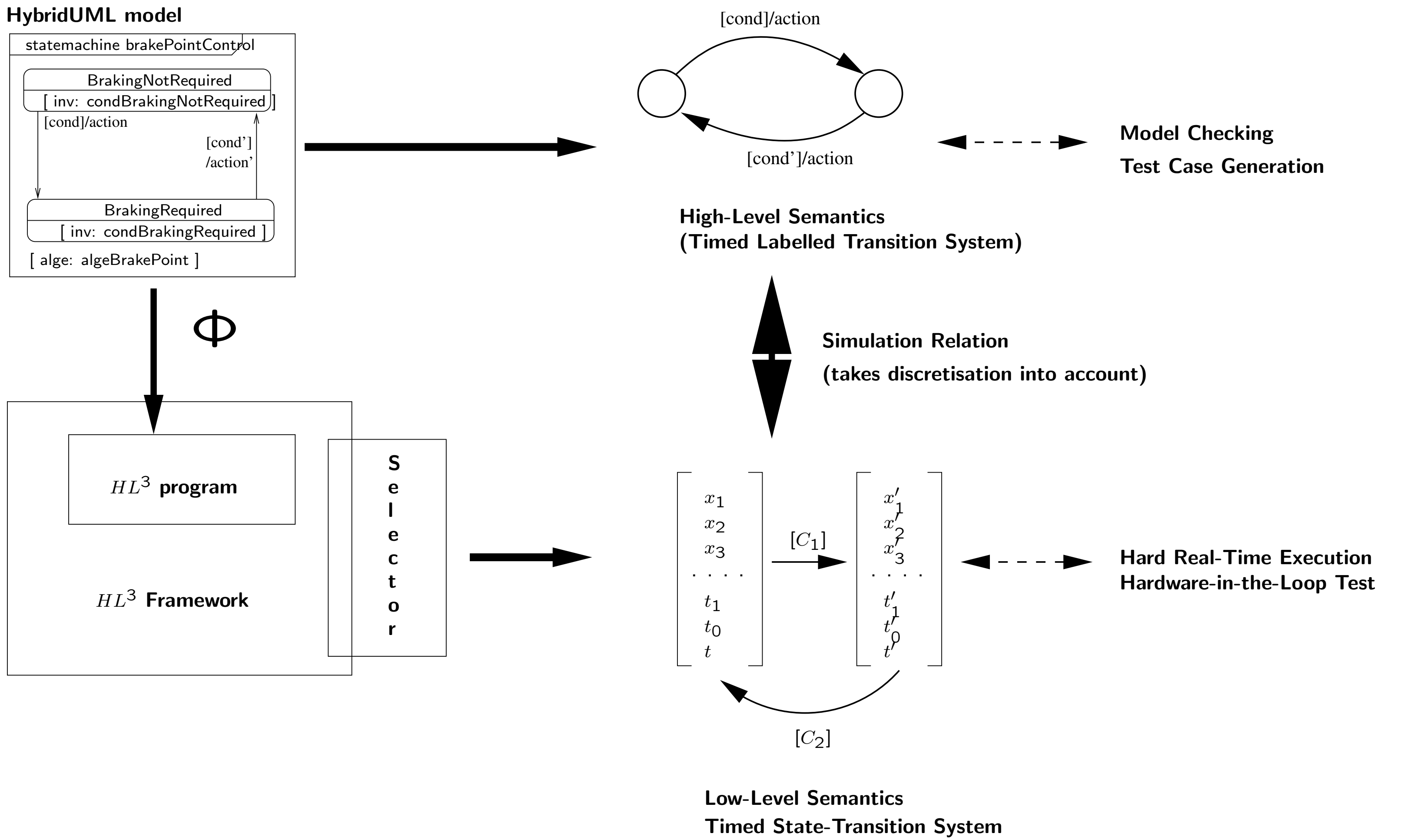
5. The chosen step is taken, then proceeded with 1.

. . . High-Level Semantics of HybridUML



HybridUML Models

. . . High-Level Semantics of HybridUML



HybridUML Model – High-Level Model – HL^3 Program

HybridUML Semantics

Semantics of agents are induced by (flat) Timed Labelled Transition System (LTS).

- Hierarchical structure of modes is encoded in *location* set Loc.

- A state σ maps (local and global) variable symbols to (appropriately typed) values

- Γ is a global structure denoting the signal space.

- Execution with respect to some global clock value $t \in \mathbb{R}_+$.

$\Longrightarrow$ An agent configuration is a tuple (loc, σ, Γ, t).

Discrete Transitions

A discrete transition step labelled $c[b]/f(\sigma); d$ between two nodes loc and loc' in agent i is thus handled as transition

$$(loc, \sigma, \Gamma, t) \xrightarrow{c} (loc', \sigma', \Gamma', t)$$

if $\Gamma \models (c, i)$ and $\sigma \models b$,
with $\sigma' = f(\sigma)$, $\Gamma' \models \neg(c, i)$ and $\Gamma \models (d, j)$ for all agents j.

Flow Transitions

A flow transition f acts on all time-continuous variables simultaneously.

We have a transition

$$(loc, \sigma, \Gamma, t) \longrightarrow (loc, \sigma', \Gamma', t')$$

if the flow f is defined on the interval $[0, t' - t]$ and enabled on the interval $[t, t']$, with $f(0) = \sigma$, $f(t' - t) = \sigma'$, and $\Gamma' \models \neg(sig, i)$ for all signals sig and all agents j.

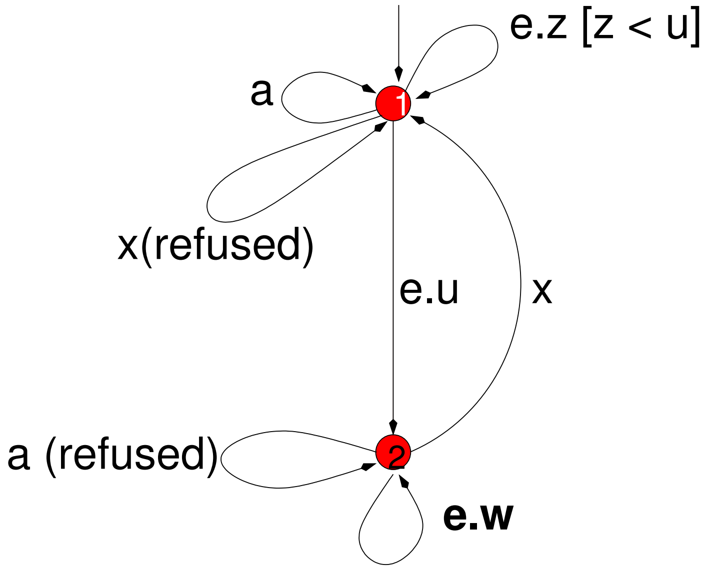
Automated Testing against (Hybrid) Real-Time Specifications

- Automated Testing against Timed CSP specifications
  - **W-Method** style theorem: Correct coverage of testing tree and characterisation set implies **correctness** of system under test in the sense of **Timed Failures Refinement**
  - Efficient test data selection heuristics for systems with constant timer durations
- Test data selection for time-continuous observables:
  - Select piecewise smooth curves through tangent vector fields on differentiable manifolds
  - Equivalence between specification and implementation may be characterised as isometry between manifolds – metric tensor maps coordinates of physical state space to coordinates of implementation state space

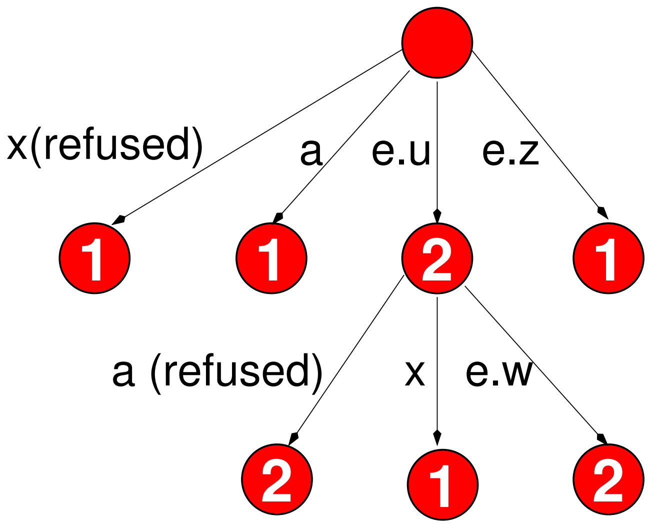
Automated Testing against (Hybrid) Real-Time Specifications

Conjecture: We cannot find a weaker relation than isometry for implementation correctness

TG(QU)



TRANSITION COVER T



CHARACTERISATION SET W = { <a> } **TESTS DEFINED BY** T^W

TEST TRACES T1 = <x,a> T2 = <a,a> T3 = <e.u,a,a>

T4 = <e.u,x,a> T5 = <e.u,e.w,a> T6 = <e.z,a>

complete set of tests if SUT has the same number of states as Q

Domain-specific description of railway control systems

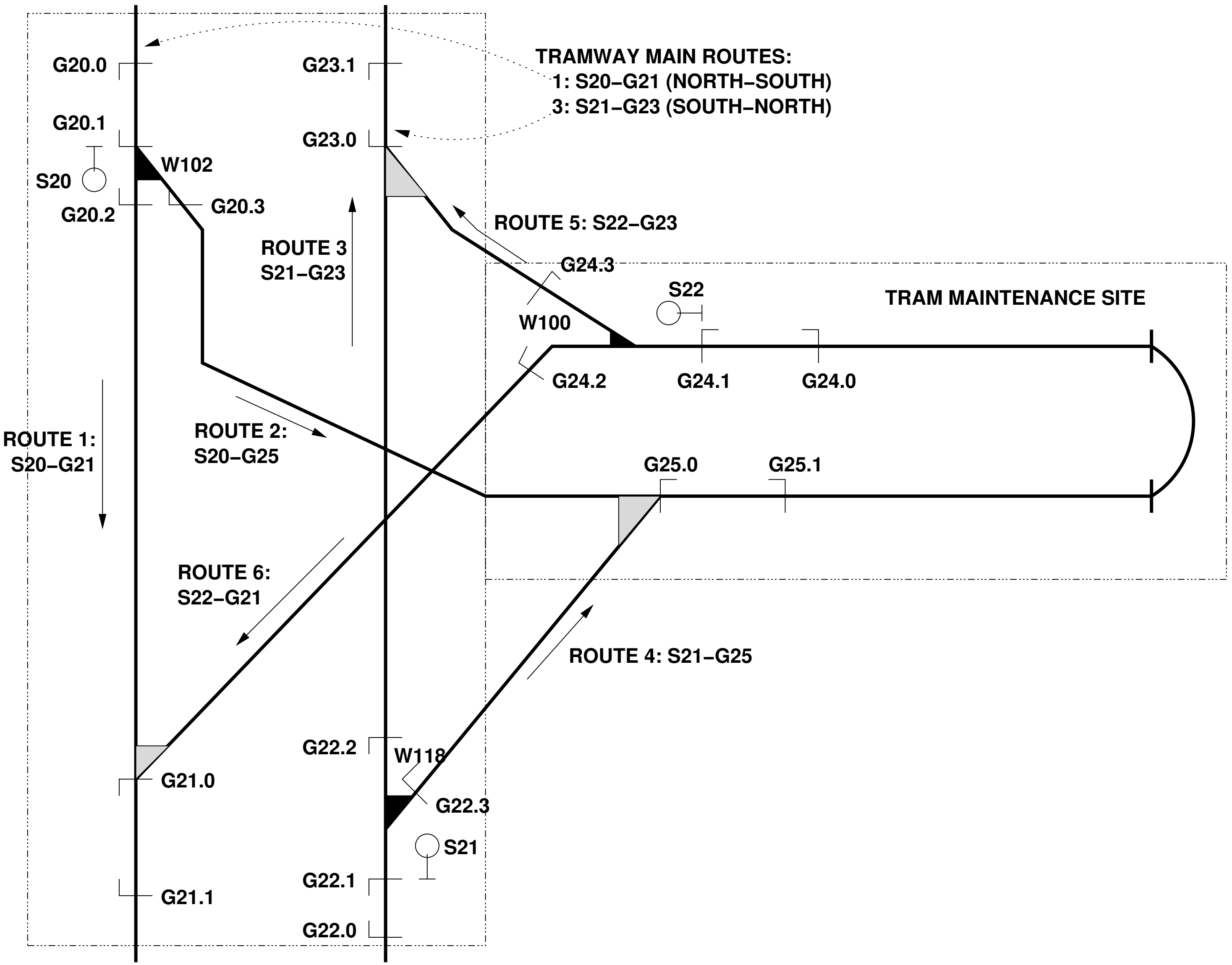
- Motivation:
  - Wide-spectrum formalism are suitable for most application domains, but require considerable IT expertise $\Rightarrow$ too complicated to be used as a means of communication between domain experts and IT specialists
  - Reason: language elements of wide-spectrum formalisms do not map directly onto concrete objects of the application domain

- Objectives of domain-specific formalisms:
  - Facilitate communication between domain experts and IT specialists by using terms and objects of the application domain in a direct way.
  - Define formal meaning of domain-specific descriptions by mapping into formal model or into wide-spectrum language with well-defined semantics

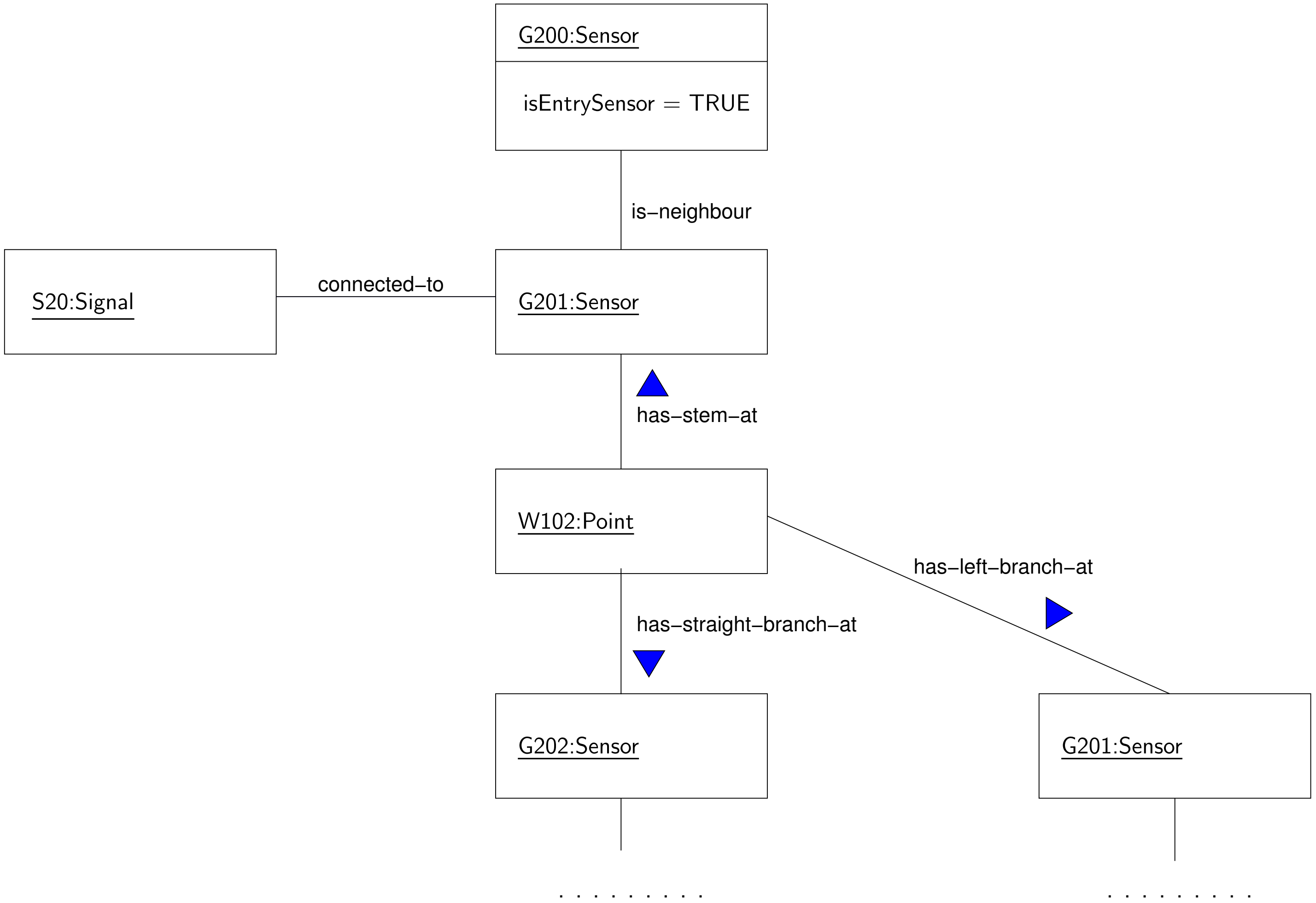
- Question: Is UML suitable for domain-specific descriptions?

- Answer: Yes – but we need UML2.0 with full profiling support:
  - Explain terms and objects of the application domain by means of the UML2 profile mechanism
  - Semi-formal semantics is given by the UML2.0-style profile description
  - Formal semantics is available, as long as constructs used in profile have formal meaning – for example, HybridUML constructs
  - Use domain-specific icons to depict the language elements of the new profile

- Examples from the railway domain:
  - Abstract railway networks are diagrams with nodes **Signal**, **Point**, **Sensor**, **Track Segment** and associations such as **connected-to**
  - These nodes are derived from HybridUML stereotype **Agent** – that is, from **Class** and **StateMachine**
  - Generic transition rules are encoded by StateMachines associated with each type of agent
  - A project-specific railway network is a concrete object diagram instantiated from the Agent diagram
  - Concrete object behaviour is specified by inserting concrete object identifications into StateMachines

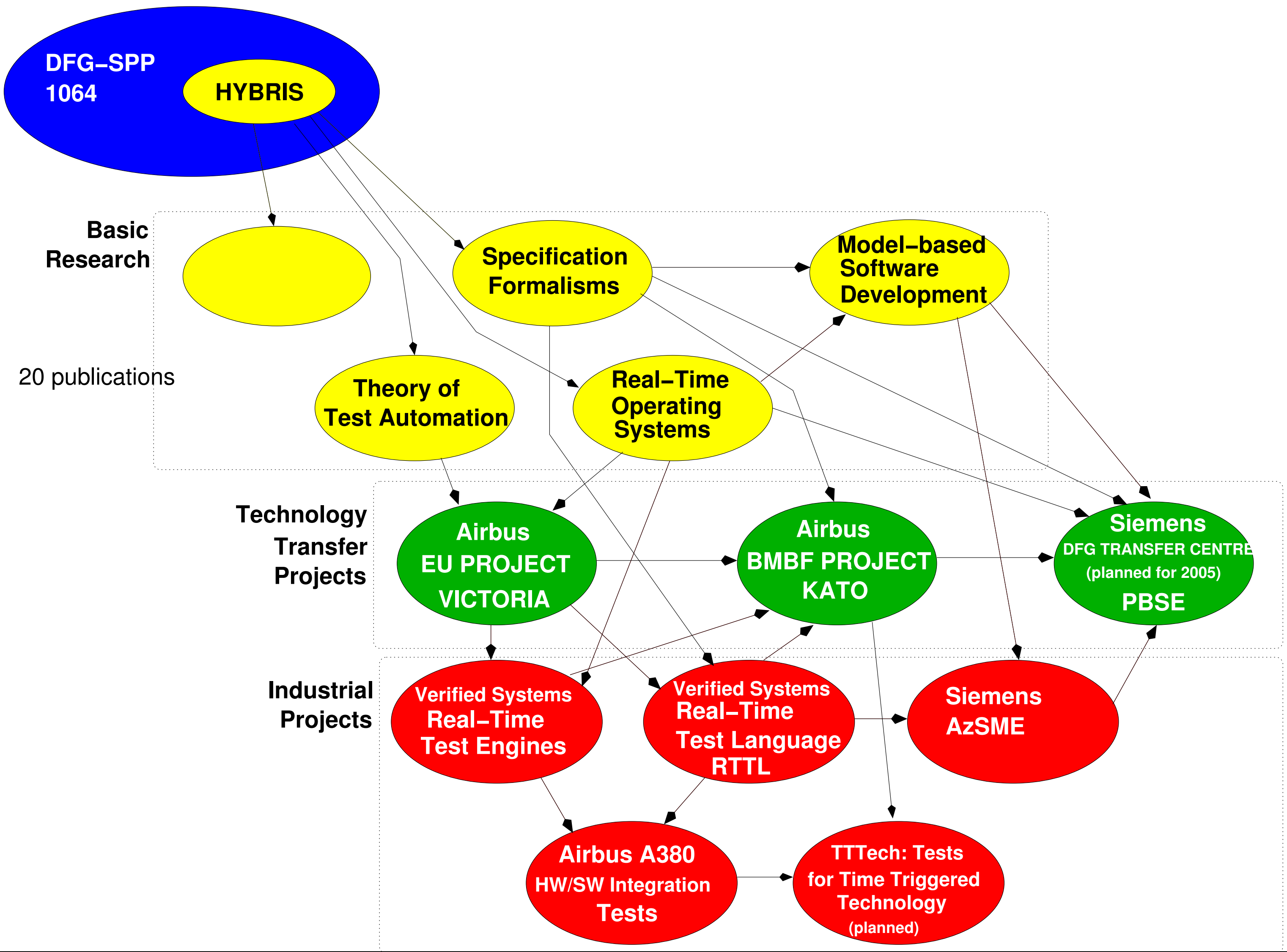


Tramway network



Tramway network – UML2.0 representation

Conclusion – Impact of the HYBRIS Project



Contributions by . . .

Peter Amthor – Kirsten Berkenkötter – Stefan Bisanz – Bettina Buth – Markus Dahlweid – Rolf Drechsler – Christof Efkemann – Ingo Fiß – Daniel Große – Ulrich Hannemann – Anne E. Haxthausen – Detlef Kendelbacher – Oliver Meyer – Jan Peleska – Holger Schlingloff – Uwe Schulze – Aliki Tsiolakis

Literature

[1] Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska. Executable HybridUML and its application to train control systems. In H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, editors, *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*. Springer Verlag, September 2004. ISBN 3-540-23135-8.

[2] Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska. Spezifikation von Echtzeit-Automatisierungssystemen mit HybridUML. *atp – Automatisierungstechnische Praxis*, 46(8):54–60, August 2004. ISSN 0178-2320.

[3] Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska. HybridUML Profile for UML 2.0. SVERTS Workshop at the ≪UML≫ 2003 Conference, October 2003. http://www-verimag.imag.fr/EVENTS/2003/SVERTS/.

[4] Jan Peleska. Formal methods for test automation - hard real-time testing of controllers for the airbus aircraft family. In *Proc. of the Sixth Biennial World Conference on Integrated Design & Process Technology (IDPT2002), Pasadena, California*. Society for Design and Process Science, June 2002. To appear.

[5] J. Peleska and A. Tsiolakis. Automated Integration Testing for Avionics Systems. In *Proceedings of the 3rd ICSTEST – International Conference on Software Testing*, April 2002.

[6] J. Peleska, S. Bisanz, I. Fiß, and M. Endreß. Non-Standard Graphical Simulation Techniques for Test Specification Development. In H. Szczerbicka, editor, *Modelling and simulation: A tool for the next millenium. 13th European Simulation Multiconference 1999*, volume 1, pages 575–580, Delft, 1999. Society for Computer Simulation International.

[7] A. Haxthausen and J. Peleska. Formal Development and Verification of a Distributed Railway Control System. In *Proceedings of First FMERail Workshop*, 1998.

[8] A. E. Haxthausen and J. Peleska. Formal Development and Verification of a Distributed Railway Control System. In *Proceedings of Formal Methods World Congress FM'99*, number 1709 in Lecture Notes in Computer Science, pages 1546 – 1563. Springer-Verlag, 1999.

[9] J. Peleska, A. Baer, and A. E. Haxthausen. Towards Domain-Specific Formal Specification Languages for Railway Control Systems. In *Proceedings of the 9th IFAC Symposium on Control in Transportation Systems 2000, June 13-15, 2000, Braunschweig, Germany*, pages 147–152, 2000.

[10] A. E. Haxthausen and J. Peleska. Formal Development and Verification of a Distributed Railway Control System. *IEEE Transaction on Software Engineering*, 26(8):687–701, 2000.

[11] A. E. Haxthausen and J. Peleska. Formal Methods for the Specification and Verification of Distributed Railway Control Systems: From Algebraic Specifications to Distributed Hybrid Real-Time Systems. In *Forms '99 - Formale Techniken für die Eisenbahnsicherung Fortschritt-Berichte VDI, Reihe 12, Nr. 436*, pages 263–271. VDI-Verlag, Düsseldorf, 2000.

[12] A. E. Haxthausen and J. Peleska. A Domain Specific Language for Railway Control Systems. In *Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology, (IDPT2002), Pasadena, California*, June 23-28 2002.

[13] A. E. Haxthausen and J. Peleska. Generation of Executable Railway Control Components from Domain-Specific Descriptions. In *Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'2003), Budapest/Hungary*, pages 83–90. L'Harmattan Hongrie, May 15-16 2003.

[14] A. E. Haxthausen and J. Peleska. Automatic Verification, Validation and Test for Railway Control Systems based on Domain-Specific Descriptions. In *Proceedings of the 10th IFAC Symposium on Control in Transportation Systems.* Elsevier Science Ltd, Oxford, 2003.

[15] A. E. Haxthausen J. Peleska, D. Große and Rolf Drechsler. Automated Verification for Train Control Systems. In *Submitted to Forms '04*, 2004.

The end.

$$\textbf{condBrakingRequired} \equiv \tag{1}$$
$$\exists i \in \{1..VTP\_COUNT\} \bullet$$
$$brakePoint[i].x \leq x \wedge \mathsf{ra}.vtp[i].v < v \wedge \mathsf{ra}.vtp[i].x > x \wedge$$
$$(vtpActive[i] \vee (3)\ \mathsf{condTooLate})$$

$$\textbf{condBrakingNotRequired} \equiv \tag{2}$$
$$\forall i \in \{1..VTP\_COUNT\} \bullet$$
$$\neg(brakePoint[i].x \leq x \wedge \mathsf{ra}.vtp[i].v < v \wedge \mathsf{ra}.vtp[i].x > x \wedge$$
$$(vtpActive[i] \vee (3)\ \mathsf{condTooLate}))$$

$$
\begin{aligned}
&\textbf{condTooLate} \equiv && (3)\\
&\mathsf{ra}.vtp[i].type = VTP\_TYPE.CROSSING \wedge \\
&((x_{closedTooLong,1}[\mathsf{ra}.vtp[i].cr.id] \leq \mathsf{ra}.vtp[i].cr.x_{end} + const.l \\
&\quad \wedge t_{closedTooLong}[\mathsf{ra}.vtp[i].cr.id] \leq t_{brake}) \\
&\quad \vee (x_{closedTooLong,2}[\mathsf{ra}.vtp[i].cr.id] \leq \mathsf{ra}.vtp[i].cr.x_{end} + const.l \\
&\quad \wedge t_{closedTooLong}[\mathsf{ra}.vtp[i].cr.id] > t_{brake}))
\end{aligned}
$$

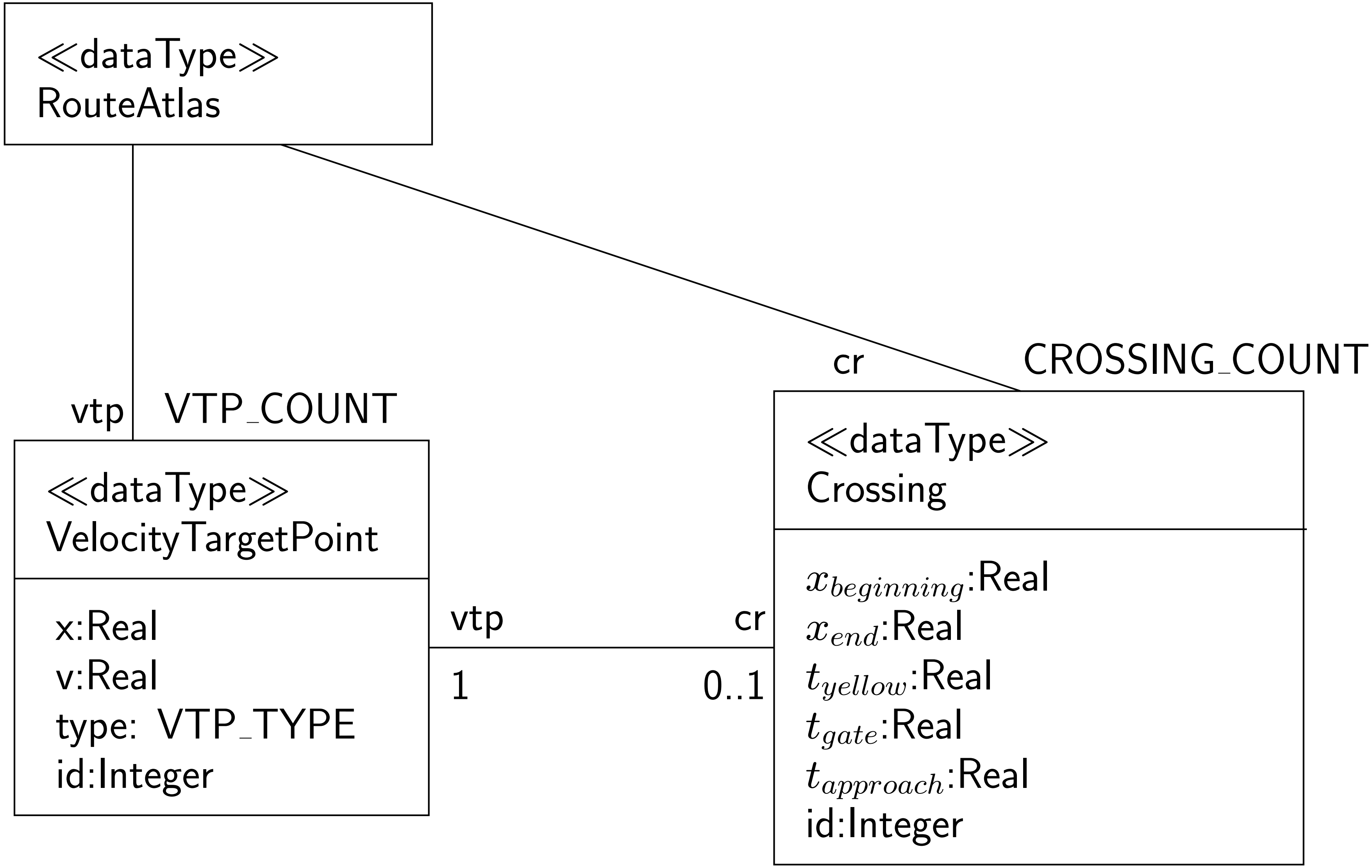
$$\textbf{algeBrakePoint} \equiv \tag{4}$$
$$\forall i \in \{1..VTP\_COUNT\} \bullet brakePoint[i] = \mathsf{ra}.vtp[i].x - \frac{\mathsf{ra}.vtp[i].v^2 - v^2}{2 \cdot const.a_{min}}$$

$$\textbf{algeStoppingDistance} \equiv \quad (5)$$
$$s_{brake} = \frac{-v^2}{2 \cdot const.a_{min}}$$

$$\textbf{algeStoppingDuration} \equiv \quad (6)$$
$$t_{brake} = \frac{-v}{const.a_{min}}$$

$$\textbf{algeGuaranteedPosition1} \equiv \tag{7}$$
$$\forall c \in \{1..CROSSING\_COUNT\} \bullet$$
$$x_{closedTooLong,1}[c] = x + \frac{const.a_{min}}{2} \cdot t_{closedTooLong}[c]^2 + v \cdot t_{closedTooLong}[c]$$

$$\textbf{algeGuaranteedPosition2} \equiv \tag{8}$$
$$\forall c \in \{1..CROSSING\_COUNT\} \bullet$$
$$x_{closedTooLong,2}[c] = x + s_{brake} + (t_{closedTooLong}[c] - t_{brake}) \cdot const.v_{pass}$$


≪dataType≫
RouteAtlas
vtp
VTP_COUNT
cr
CROSSING_COUNT
≪dataType≫
VelocityTargetPoint
x:Real
v:Real
type: VTP_TYPE
id:Integer
vtp
1
cr
0..1
≪dataType≫
Crossing
$x_{beginning}$:Real
x_{end}:Real
t_{yellow}:Real
t_{gate}:Real
$t_{approach}$:Real
id:Integer

Datatypes

$$
\begin{aligned}
&\textbf{readSetTransBrakingRequired} \equiv \\
&\{v \mid \exists i \in \{1..VTP\_COUNT\} \bullet \\
&\quad v \equiv chan(vtpActive[i]) \vee v \equiv chan(brakePoint[i]) \vee \\
&\quad v \equiv chan(ra.vtp[i].v) \vee v \equiv chan(ra.vtp[i].x) \\
&\} \cup \{chan(x), chan(v)\}
\end{aligned} \tag{9}
$$