HYBRIS – Efficient Analysis of Hybrid Systems

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

Univerrsität Bremen
Germany
{kirsten,bisanz,ulrichh,jp}@informatik.uni-bremen.de
Outline

1. HybridUML – Specification Formalism for Hybrid Systems


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 Profile

HybridUML Graphical Notation

HybridUML Textual Notation

HybridUML XMI (XML Metadata Interchange)

HybridUML Internal Data Structure

$\Phi$

$HL^3$ Program

$HL^3$ – Hybrid Low-Level Language

$HL^3$ Framework

HybridUML Models

HybridUML Graphical Notation

HybridUML Profile

HybridUML Graphical Notation

HybridUML Textual Notation

HybridUML XMI (XML Metadata Interchange)

HybridUML Internal Data Structure

$\Phi$

$HL^3$ Program

$HL^3$ – Hybrid Low-Level Language

$HL^3$ Framework

HybridUML Models – Graphical Notation
Class diagram of TrainController
class TrainController

: CloseRequestController
  closingFailed(...)
  activateCrossing(...)
  crossingActivated(...)
  crossingNotActivated(...)
  closeAcknowledged
  const
  vtpActive[]
  x
  v
  ra

: CrossingStatusController
  ra
  t_{closedTooLong}
  statusFailed(...)
  closeAcknowledged
  const
  crossingStatusReq(...)
  v
  crossingStatus(...)

: MovementController
  v
  brakingRequired
  a
  v_allowed
  const
  emergency()

: EmergencyController
  x
  emergency()
  v
  vtpActive[]
  ra

: EmergencyController
  v
  vtpActive[]
  ra

: BrakePointController
  t_{closedTooLong}
  brakingRequired
  const
  x
  v
  vtpActive[]

: TrainRadioController
  recvCrossing[](...)
  sendCrossing[](...)

: TrainRadioController
  recvCrossing[](...)
  sendCrossing[](...)

: TrainRadioController
  activateCrossing(...)
  crossingActivated(...)
  crossingNotActivated(...)
  crossingStatusReq(...)
  crossingStatus(...)

: CloseRequestController
  a_user

: UserInteractionController
  a_user

: LocalizationController
  x
  v
  v_allowed
  ra

Structure diagram of TrainController

28.09.2004

HYBRIS – Efficient Analysis of Hybrid Systems

J. Peleska, S. Bisanz
class TrainController

: CrossingStatusController
  \( t_{closedTooLong} \)
  statusFailed(…)
  closeAcknowledged
  crossingStatusReq(…)
  crossingStatus(…)

: MovementController
  \( v_{allowed} \)
  brakingRequired
  user
  emergency()
  const

: BrakePointController
  \( t_{closedTooLong} \)
  brakingRequired
  const
  x
  v
  vtpActive[]
  ra

Focus on BrakePointController

Zooming ...

Focus on BrakePointController
Focus on BrakePointController
Focus on BrakePointController
HybridUML Graphical Notation

statemachine brakePointControl

BrakingNotRequired

[ inv: (2) condBrakingNotRequired ]

[ (1) condBrakingRequired ]
/ brakingRequired := true

[ (2) condBrakingNotRequired ]
/ brakingRequired := false

BrakingRequired

[ inv: (1) condBrakingRequired ]

[ alge: (4) algeBrakePoint ]
[ flow: \( \forall c \in \{1..CROSSING\_COUNT\} \bullet t_{closedTooLong}[c] = -1 \) ]

[ alge: (5) algeStoppingDistance ]

[ alge: (6) algeStoppingDuration ]

[ alge: (7) algeGuaranteedPosition1 ]

[ alge: (8) algeGuaranteedPosition2 ]

Behaviour of BrakePointController

state machine brakePointControl

BrakingNotRequired

[ inv: (2) condBrakingNotRequired ]

[ (1) condBrakingRequired ]
/ brakingRequired := true

[ (2) condBrakingRequired ]
/ brakingRequired := false

BrakingRequired

[ inv: (1) condBrakingRequired ]

[ alge: (4) algeBrakePoint ]
[ flow: \( \forall c \in \{1..CROSSING\_COUNT\} \bullet t_{closed\_Too\_Long}[c] = -1 \) ]

[ alge: (5) algeStoppingDistance ]
[ alge: (7) algeGuaranteedPosition1 ]

[ alge: (6) algeStoppingDuration ]
[ alge: (8) algeGuaranteedPosition2 ]

Behaviour of BrakePointController

28.09.2004

HYBRIS – Efficient Analysis of Hybrid Systems

J. Peleska, S. Bisanz
statemachine brakePointControl

BrakingNotRequired

[ inv: (2) condBrakingNotRequired ]

[ (1) condBrakingRequired ]
/ brakingRequired := true

[ (2) condBrakingNotRequired ]
/ brakingRequired := false

BrakingRequired

[ inv: (1) condBrakingRequired ]

[ alge: (4) algeBrakePoint ]

Simplified behaviour of BrakePointController
HybridUML Internal Data Structure

HybridUML Profile

HybridUML Graphical Notation

HybridUML Textual Notation

HybridUML XMI (XML Metadata Interchange)

HybridUML Models – Internal Data Structure

$HL^3$ – Hybrid Low-Level Language

$HL^3$ Framework

$HL^3$ Program

$Φ$
HybridUML Internal Data Structure

- SignalConnector
  - SignalPort
    - VariablePort
      - VariableConnector
    - Property
      - PropertyValue
  - Property
- Agent
- Mode
- ControlPoint
- AlgebraicConstraint
- Transition
- FlowConstraint
- RTSignal
- InvariantConstraint
- RTTrigger
- GuardConstraint
- Action
: BrakePointController

- brakingRequired
- x
- v
- vtpActive[]
Internal Data Structure Representation of BrakePointController
brakePointControl

BrakingNotRequired
- condBrakingNotRequired
- brakingRequired := true
  - condBrakingNotRequired
  - brakingRequired := false

BrakingRequired
- condBrakingRequired
- inv: condBrakingRequired

[ alge: algeBrakePoint ]
Internal Data Structure Representation of BrakePointController
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)
  → executable code

- Executable code defines formal semantics, based on:
  - Runtime environment
  - Design pattern
Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool

trigger init(), notifyTrans(), update()

Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)

enables/disables via guard

trigger action a.action()

Periodic scheduling of integrate()

Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()

Scheduler

trigger init(), getSelection()

Transition
+id: transId
+c: Condition
+a: Action

Selector
+init(): bool
+getSelection(leftFlowPhase:bool): Selection

TimeService
+clustertime: nat
+hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()

getHL3Time()

put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte

Channel

rx(n:nodeId): seq of byte

ClusterCommunication
+tx(n:nodeId,d:seq of byte)

AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

Scheduler

getHL3Time()

put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte

ClusterCommunication
+tx(n:nodeId,d:seq of byte)

AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

28.09.2004

HYBRIS – Efficient Analysis of Hybrid Systems

J. Peleska, S. Bisanz
Runtime Environment
Provides low level constructs to be used by $HL^3$ programs: (1) Channel (2) Cluster-Communication (3) TimeService (4) Scheduler

Abstract Machine, Flow, Interface Module, Transition, Selector; read/write global state to/from instances of Channel
The HL³ Framework is a tool for efficient analysis of hybrid systems. It is designed to support the modeling and simulation of complex systems that exhibit both discrete and continuous behavior. The framework provides a high-level abstraction of the system components, including Abstract Machine, Flow, Interface Module, Transition, Selector, Cluster Communication, and Time Service.

- **Abstract Machine**: Represents the core of the system, defining the state and behavior of the system over time. It includes methods for initializing, updating, and querying the state of the system.

- **Flow**: Models the flow of information within the system, including methods for integrating data and checking conditions.

- **Interface Module**: Handles the interaction between the Abstract Machine and the external world, providing methods for processing data and managing the flow.

- **Transition**: Represents the change in state of the system, triggered by conditions and actions. It includes methods for setting up and triggering transitions.

- **Selector**: Facilitates the selection of specific paths in the system, enabling/disabling transitions via guard conditions.

- **Cluster Communication**: Manages the communication between different parts of the system, including methods for transmitting and receiving data.

- **Time Service**: Provides the timing infrastructure for the system, including methods for getting/settting HL3 time and synchronizing the system.

The HL³ Framework supports periodic scheduling of certain operations, such as polling and transmitting data, and enables/disables transitions via guard conditions. It is designed to be extensible, allowing for the addition of new components and functionalities as needed.
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)
- \((receiver, \text{timestamp}) \rightarrow \text{VisibilitySet}\)
**HL³ Framework**

- **AbstractMachine**
  - +id: amId
  - +amStatus: active|suspended|stopped
  - +init()
  - +notifyTrans(tr:transId)
  - +update()
  - +getTrans(id:amId): set of transId
  - +isFlowEnabled(id:amId): bool

- **Transition**
  - +id: transId
  - +c: Condition
  - +a: Action
  - evaluates condition
  - trigger action a.action()

- **Flow**
  - +id: flowId
  - +guard: bool
  - +frequency: int
  - +integrate(v:VisibilitySet)
  - integrates v
  - enables/disables via guard

- **Interface Module**
  - +id: ifmId
  - +freqPoll: int
  - +freqTx: int
  - +poll(v:VisibilitySet)
  - periodic scheduling
  - +transmit()
  - transmit()

- **Scheduler**
  - trigger init(), notifyTrans(), update()

- **Selector**
  - trigger init(), getSelection()

- **Channel**
  - get(id:amId + ifmId): seq of byte

- **ClusterCommunication**
  - tx(n:nodeId, d: seq of byte)
  - rx(n:nodeId): seq of byte

- **TimeService**
  - +clustertime: nat
  - +hl3time: timeTick
  - +getHL3Time(): timeTick
  - +setHL3Time(t:timeTick)
  - +synchronize()

- **Hybris – Efficient Analysis of Hybrid Systems**
  - J. Peleska, S. Bisanz

28.09.2004
Abstract Machine, Flow, Interface Module, Transition, Selector; read/write global state to/from instances of Channel

+id: transId
+c: Condition
+a: Action
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()

trigger init(), getSelection()

Periodic scheduling of poll(), transmit() Periodic scheduling of integrate()... Module
HL3 Framework
ClusterCommunication
TimeService
Scheduler
Channel
getHL3Time()
    getHL3Time()
Abstract Machine, Flow, Interface Module, 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 \( t_0 \)

**ClusterCommunication**

- AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel

**Scheduler**

- \( \text{init} \): bool
- trigger \( \text{init} \), \( \text{getSelection} \)

**Selector**

- \( \text{getSelection} \): Selection

**Interface Module**

- \( +\text{id}: \text{id} \)
- \( +\text{c}: \text{Condition} \)
- \( +\text{a}: \text{Action} \)

**Transition**

- \( +\text{id}: \text{id} \)
- \( +\text{freqPoll}: \text{freq} \)
- \( +\text{poll}(v:\text{VisibilitySet}) \)
- \( +\text{transmit}() \)

**Flow**

- \( +\text{id}: \text{id} \)
- \( +\text{guard}: \text{bool} \)
- \( +\text{frequency}: \text{freq} \)
- \( +\text{integrate}(v:\text{VisibilitySet}) \)

**Abstract Machine**

- \( +\text{id}: \text{id} \)
- \( +\text{status}: \text{active}|\text{suspended}|\text{stopped} \)
- \( +\text{init}() \)
- \( +\text{notifyTrans}(\text{tr}:\text{id}) \)
- \( +\text{update}() \)

- \( +\text{getTrans}(\text{id}:\text{id}) \): set of \( \text{id} \)
- \( +\text{isFlowEnabled}(\text{id}:\text{id}) : \text{bool} \)

**Channel**

- \( +\text{id}: \text{id} \)
- \( +\text{guard}: \text{bool} \)
- \( +\text{frequency}: \text{freq} \)
- \( +\text{integrate}(v:\text{VisibilitySet}) \)

- \( +\text{get\text{HL3Time}(): timeTick} \)
- \( +\text{set\text{HL3Time}(t:timeTick)} \)
- \( +\text{get\text{HL3Time}()} \)
- \( +\text{synchronize()} \)

**Cluster Communication**

- \( +\text{clustertime}: \text{nat} \)
- \( +\text{hl3time}: \text{timeTick} \)
- \( +\text{get\text{HL3Time}(): timeTick} \)
- \( +\text{set\text{HL3Time}(t:timeTick)} \)
- \( +\text{synchronize()} \)

28.09.2004

**HYBRIS – Efficient Analysis of Hybrid Systems**

J. Peleska, S. Bisanz
Periodic scheduling of poll() and transmit() enables/disables via guard evaluates condition evaluates condition

Abstract Machine
+id: amId
+amStatus: active|suspended|stopped
+init()
+notifyTrans(tr:transId)
+update()
+getTrans(id:amId): set of transId
+isFlowEnabled(id:amId): bool

Transition
+id: transId
+c: Condition
+a: Action
trigger init(), getSelection()

Flow
+id: flowId
+guard: bool
+frequency: int
+integrate(v:VisibilitySet)

Interface Module
+id: ifmId
+freqPoll: int
+freqTx: int
+poll(v:VisibilitySet)
+transmit()

Scheduler
trigger init(), getSelection()

Selector
trigger init(), getSelection(leftFlowPhase:boolean)

TimeService
+clustertime: nat
+hl3time: timeTick
+getHL3Time(): timeTick
+setHL3Time(t:timeTick)
+synchronize()

Channel
+e: set of ChannelEntry
+put(d:seq of byte,v:VisibilitySet)
+get(id:amId + ifmId): seq of byte
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte

ClusterCommunication
+tx(n:nodeId,d:seq of byte)
+rx(n:nodeId): seq of byte

HYBRIS – Efficient Analysis of Hybrid Systems
J. Peleska, S. Bisanz

Universität Bremen

28.09.2004

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
Abstract Machine, Flow, Interface Module, HL3 Framework
ClusterCommunication
Channel
Selector
Scheduler
Transition
TimeService

+ id: transId
+ c: Condition
+ a: Action

HL3 Framework

+ id: flowId
+ guard: bool
+ frequency: int
+ integrate(v: VisibilitySet)

**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)
The HL³ Framework is a tool for the analysis of hybrid systems. It consists of several components:

- **Abstract Machine**: Includes methods like `init()`, `notifyTrans()`, and `update()`. It has attributes such as `amId`, `amStatus`, `freqPoll`, `freqTx`, and methods for integrating and transmitting data.

- **Flow**: Contains methods like `poll()`, `transmit()`, and `integrate()`. It has attributes such as `flowId`, `guard`, `frequency`, and `condition`.

- **Interface Module**: Includes methods like `init()`, `put()`, `get()`, and `tx()`. It has attributes such as `ifmId`, `freqPoll`, and `freqTx`.

- **Transition**: Includes methods like `trigger()`, `getSelection()`, and `isFlowEnabled()`. It has attributes such as `transId` and `condition`.

- **Selector**: Includes methods like `trigger()`, `getSelection()`, and `isFlowEnabled()`. It has attributes such as `ifmId` and `amId`.

- **TimeService**: Includes methods like `getHL3Time()`, `setHL3Time()`, and `synchronize()`. It has attributes such as `clustertime` and `hl3time`.

- **Channel**: Includes methods like `put()`, `get()`, `tx()`, and `rx()`. It has attributes such as `flowId`, `guard`, `frequency`, and `data`.

- **ClusterCommunication**: Includes methods like `tx()`, `rx()`, and `getTrans()`. It has attributes such as `nodeId` and `data`.

The HL³ Framework enables/disables via guard, evaluates conditions, and triggers init(), getSelection(), and notifyTrans(). The scheduler reads/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

AbstractMachine:
- id: amId
- status: active|suspended|stopped
- init()
- notifyTrans(tr:transId)
- update()
- getTrans(id:amId): set of transId
- isFlowEnabled(id:amId): bool
- poll(v:VisibilitySet)
- transmit()

Interface Module
- id: ifmId
- freqPoll: int
- freqTx: int
- poll(v:VisibilitySet)
- transmit()

ClusterCommunication
- e: set of ChannelEntry
- put(d:seq of byte,v:VisibilitySet)
- get(id:amId + ifmId): seq of byte

Scheduler
- trigger init(), notifyTrans(), update()
Abstract Machine, 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
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
HYBRIS – Efficient Analysis of Hybrid Systems

J. Peleska, S. Bisanz

28.09.2004
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
$\Phi \rightarrow HL^3$ Program

HybridUML Profile

HybridUML Graphical Notation

HybridUML Textual Notation

HybridUML XMI (XML Metadata Interchange)

HybridUML Internal Data Structure

$\Phi$

$HL^3$ Program

$HL^3$ – Hybrid Low-Level Language

$HL^3$ Framework

HybridUML Models – $HL^3$ Program

28.09.2004

HYBRIS – Efficient Analysis of Hybrid Systems

J. Peleska, S. Bisanz
Transforming Internal Data Structure into a $HL^3$ program by applying the $HL^3$ Design Pattern.
Step 1 – Generate $HL^3$::AbstractMachine

→ AbstractMachine class: once
→ notifyTrans(...), update(...), ...: once
→ data structure for HybridUML Modes’ local control structure: once
→ AbstractMachine instances for HybridUML basic agents: for each HybridUML model
Abstract Machine example: BrakePointController
AbstractMachine example: BrakePointController
AbstractMachine example: BrakePointController
Φ → \(HL^3\) Program

Abstract Machine, Flow, Interface Module, Transition, Selector; read/write global state to/from instances of Channel
Step 2 – Generate $HL^3$::Flow

⇒ Flow class: once
⇒ integrate(...): once
→ Flow instances for HybridUML algebraic and flow constraints: for each HybridUML model
⇒ 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);
}
Abstract Machine, Flow, Interface Module, Transition, Selector; read/write global state to/from instances of Channel

AbstractMachine, Flow, InterfaceModule, Transition, Selector; read/write global state to/from instances of Channel
Step 3 – Generate HL3::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. ⇒ $e(c)$: Continuous step $c$ admissible?

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

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

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

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

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

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

5. The chosen step is taken, then proceeded with 1.
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 \( \text{Loc} \).

- A state \( \sigma \) maps (local and global) variable symbols to (appropriately typed) values.

- \( \Gamma \) is a global structure denoting the signal space.

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

\[ \Rightarrow \text{An agent configuration is a tuple (loc, } \sigma, \Gamma, t). \]
Discrete Transitions

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

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

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

\[
(\text{loc}, \sigma, \Gamma, t) \longrightarrow (\text{loc}, \sigma', \Gamma', t')
\]

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 \( \Gamma' \models \neg (\text{sig}, i) \) for all signals \( \text{sig} \) 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
CHARACTERISATION SET \( W = \{ <a> \} \) TESTS DEFINED BY \( T^W \)

TEST TRACES \( T1 = <x,a> \quad T2 = <a,a> \quad T3 = <e.u,a,a> \)
\( T4 = <e.u,x,a> \quad T5 = <e.u,e.w,a> \quad T6 = <e.z,a> \)

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 ⇒ 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 MAIN ROUTES:
1: S20−G21 (NORTH-SOUTH)
3: S21−G23 (SOUTH-NORTH)
4: S21−G25
5: S22−G23
6: S22−G21

TRAM MAINTENANCE SITE

S22
W118
W100

W102

TRAMWAY MAIN ROUTES:

ROUTE 1: S20−G21
ROUTE 2: S20−G25
ROUTE 3: S21−G23
ROUTE 4: S21−G25
ROUTE 5: S22−G23
ROUTE 6: S22−G21

G20.0 G20.1 G20.2
G20.3
G23.1
G23.0

G25.0 G25.1
G24.0 G24.1 G24.2

G21.0 G21.1
G22.0 G22.1 G22.2 G22.3

G23.1

W100

S21

S22

W118

W102

S20

G24.0 G24.1 G24.2 G24.3

G22.2

G21.0 G21.1

G20.0 G20.1 G20.2 G20.3

Tramway network
Tramway network – UML2.0 representation

- S20:Signal
- G200:Sensor
  - isEntrySensor = TRUE
- G201:Sensor
- G202:Sensor
- W102:Point
- G201:Sensor

Connections:
- connected-to: S20:Signal to G200:Sensor
- has-stem-at: G201:Sensor to W102:Point
- has-left-branch-at: W102:Point to G202:Sensor
- has-straight-branch-at: W102:Point to G201:Sensor

Tramway network – UML2.0 representation
Conclusion – Impact of the HYBRIS Project

HYBRIS

DG-F-SPP
1064

Basic Research

20 publications

Theory of Test Automation

Specification Formalisms

Real-Time Operating Systems

Model-based Software Development

Airbus
EU PROJECT
VICTORIA

Airbus
BMBF PROJECT
KATO

Siemens
DFG TRANSFER CENTRE
(planned for 2005)

PBSE

Technology Transfer Projects

Industrial Projects

Verified Systems Real-Time Test Engines

Verified Systems Real-Time Test Language RTTL

Siemens A2SME

Airbus A380 HW/SW Integration Tests

TTTech: Tests for Time Triggered Technology (planned)

Conclusion – Impact of the HYBRIS Project

28.09.2004

HYBRIS – Efficient Analysis of Hybrid Systems

J. Peleska, S. Bisanz
Contributions by . . .

Literature


The end.
condBrakingRequired ≡
\[ \exists i \in \{1..VTP\_COUNT\} \bullet \]
\[ brakePoint[i].x \leq x \land ra.vtp[i].v < v \land ra.vtp[i].x > x \land \]
\[ (vtpActive[i] \lor (3) \text{condTooLate}) \]

condBrakingNotRequired ≡
\[ \forall i \in \{1..VTP\_COUNT\} \bullet \]
\[ \neg (brakePoint[i].x \leq x \land ra.vtp[i].v < v \land ra.vtp[i].x > x \land \]
\[ (vtpActive[i] \lor (3) \text{condTooLate})) \]
condTooLate ≡
ra.vtp[i].type = VTP_TYPE.CROSSING∧
((x_{closedTooLong,1}[ra.vtp[i].cr.id] ≤ ra.vtp[i].cr.x_{end} + const.l
 ∧ t_{closedTooLong}[ra.vtp[i].cr.id] ≤ t_{brake})
∨ (x_{closedTooLong,2}[ra.vtp[i].cr.id] ≤ ra.vtp[i].cr.x_{end} + const.l
 ∧ t_{closedTooLong}[ra.vtp[i].cr.id] > t_{brake}))
algeBrakePoint \equiv \\
\forall i \in \{1..VTP\_COUNT\} \bullet brakePoint[i] = ra.vtp[i].x - \frac{ra.vtp[i].v^2-v^2}{2.\text{const.}a_{\text{min}}}

(4)
\begin{align*}
\text{algeStoppingDistance} & \equiv \\
    s_{brake} &= \frac{-v^2}{2 \cdot \text{const}.a_{\text{min}}} \\
\text{algeStoppingDuration} & \equiv \\
    t_{brake} &= \frac{-v}{\text{const}.a_{\text{min}}}
\end{align*}
\textbf{algeGuaranteedPosition1} \equiv \\
\forall c \in \{1..\text{CROSSING\_COUNT}\} \bullet \\
x_{\text{closedTooLong},1}[c] = x + \frac{\text{const.a}_{\text{min}}}{2} \cdot t_{\text{closedTooLong}}[c]^2 + v \cdot t_{\text{closedTooLong}}[c] \\

\textbf{algeGuaranteedPosition2} \equiv \\
\forall c \in \{1..\text{CROSSING\_COUNT}\} \bullet \\
x_{\text{closedTooLong},2}[c] = x + s_{\text{brake}} + (t_{\text{closedTooLong}}[c] - t_{\text{brake}}) \cdot \text{const.v}_{\text{pass}}
HybridUML Graphical Notation

Universität Bremen

RouteAtlas

VelocityTargetPoint

vtp

VTP_COUNT

vtp

VTP_TYPE

}}
dataType

RouteAtlas

x:Real

v:Real

type: VTP_TYPE

id:Integer

vtp

Crossing

{{{dataType}}}

Crossing

x_beginning:Real

x_end:Real

t_yellow:Real

t_gate:Real

t_approach:Real

id:Integer

vtp

cr

CROSSING_COUNT

1

0..1

Datatypes

\[\text{readSetTransBrakingRequired} \equiv \{ v | \exists i \in \{1..VTP\_COUNT\} \bullet \]
\[v \equiv \text{chan}(vtpActive[i]) \lor v \equiv \text{chan}(brakePoint[i]) \lor \]
\[v \equiv \text{chan}(\text{ra.vtp}[i].v) \lor v \equiv \text{chan}(\text{ra.vtp}[i].x)\]
\[\} \cup \{ \text{chan}(x), \text{chan}(v) \}\]