Politecnico di Milano
Formal Methods for Concurrent and Real-Time Systems

Computer Controller Automatic Transmission
(Mandatory Part, Optional Part coming soon)

Alessandra Bonetto
Filippo Sironi
Matteo Villa

2009

Contents

List of Figures

Listings

1 CCAT Class

The *ComputerControlledAutomaticTransmission* class is formalized thanks to the code reported in Listing 1 while Figure 1 shows the *big picture* of our complete designed.

Listing 1: ComputerControlledAutomaticTransmission.trio

```
class ComputerControlledAutomaticTransmission

import:
    HydraulicSystem,
    PlanetaryGearSet,
    TransmissionControlUnit,
    VehicleSpeedSensor,
    EngineSpeedSensor;

signature:

visible:
    torqueConverterState,
    vehicleSpeed,
    engineSpeed;

temporal domain: real;

domains:
    TorqueConverterState: {Attached, Detached};

items:
    TD total torqueConverterState: TorqueConverterState;
    TD total vehicleSpeed: integer;
    TD total engineSpeed: integer;

modules:
    hydraulicSystem: HydraulicSystem;
    planetaryGearSet: PlanetaryGearSet;
    transmissionControlUnit: TransmissionControlUnit;
    vehicleSpeedSensor: VehicleSpeedSensor;
    engineSpeedSensor: EngineSpeedSensor;

connections:
    (direct EngineSpeedSensor.actualSpeed, engineSpeed)
    (direct vehicleSpeedSensor.actualSpeed, vehicleSpeed)
    (direct planetaryGearSet.transmissionShaftState,
       torqueConverterState)

    (direct planetaryGearSet.gearShift,
       hydraulicSystem.gearShift)
    (direct planetaryGearSet.gearDrive,
       hydraulicSystem.gearDrive)
    (direct planetaryGearSet.gearPark,
       hydraulicSystem.gearPark)
    (direct planetaryGearSet.gearReverse,
       hydraulicSystem.gearReverse)
    (direct hydraulicSystem.controlGearShift,
       transmissionControlUnit.controlGearShift)
    (direct transmissionControlUnit.receiveEngineSpeed,
       engineSpeedSensor.sendSpeed)
    (direct transmissionControlUnit.receiveVehicleSpeed,
       vehicleSpeedSensor.sendSpeed)

end
```

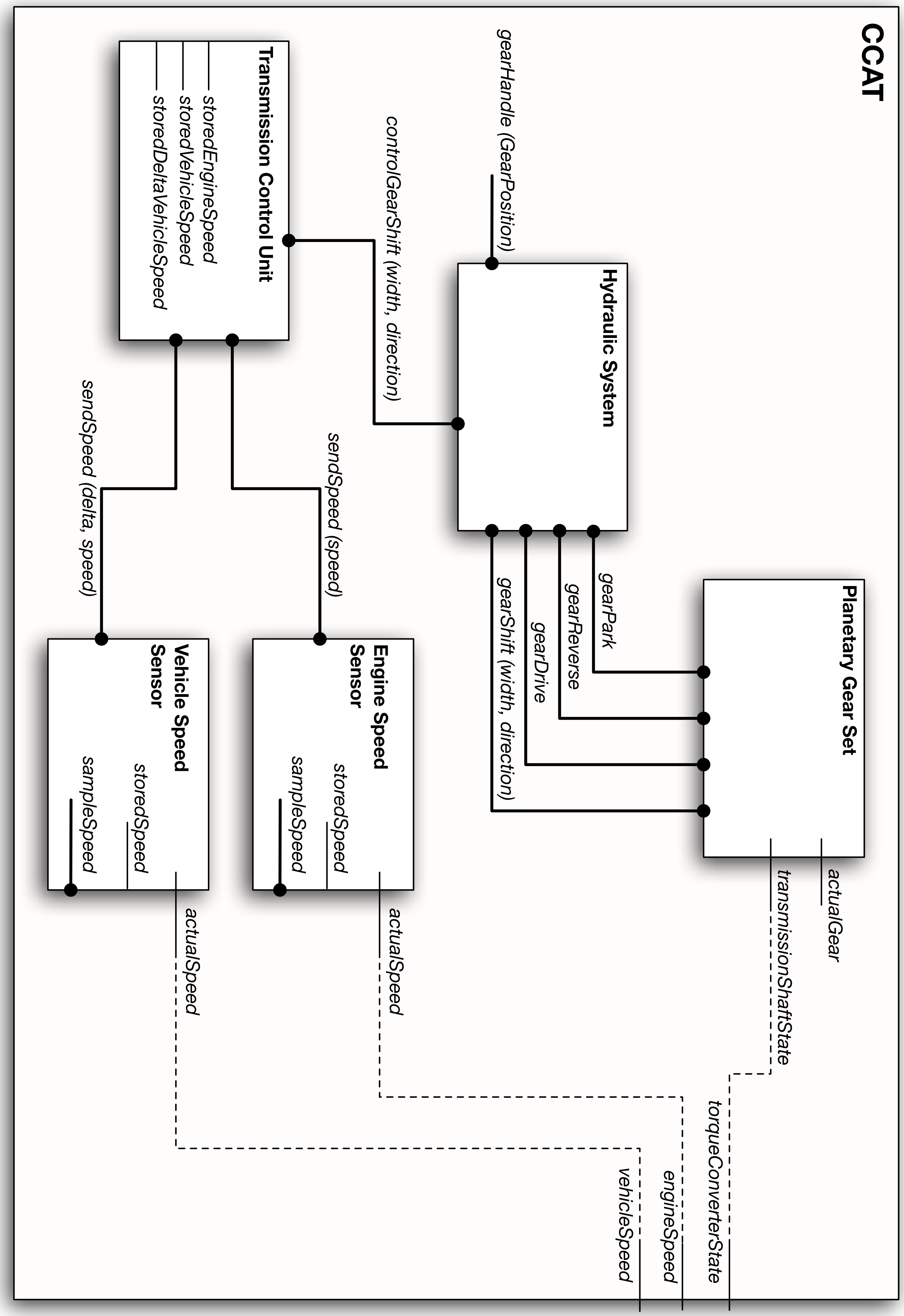

Figure 1: Computer Controller Automatic Transmission

2 Vehicle/EngineSpeedSensor Classes

The *VehicleSpeedSensor* is formalized thanks to the code reported in Listing 2 while the *EngineSpeedSensor* is formalized thanks to the code reported in Listing 3.

During the formalization of sensors we decided to simplify the design assuming that every time a `sampleSpeed` event occurs the state variable `actualSpeed` - which is `time dependent` and `total` - is automatically updated with the actual measured speed. This means we don't provide any axioms formalizing this behavior.

Moreover, we specified the starting point of the constant frequency sample chain saying that sometimes in the past there was a `sampleSpeed` occurrence. Further more, we guarante that `sampleSpeed` events will accure at constant frequency. In addition, if the sensor has memory we imposed that the `storedValue` is equal to 0. These can be consider just like the "initial conditions" of the system.

At the end, we guaranteed a sensor performs the needed action if and only if a sample event occur.

We didn't write any axioms specifying the fact that a `sendSpeed` event is mutually exclusive with itself due to the `total time dependent` parameter it accepts.

Listing 2: VehicleSpeedSensor.trio

```
class VehicleSpeedSensor (const sampleInterval, const
    sampleDelay)

signature:

visible:
    actualSpeed,
    sendSpeed;

temporal domain: real;

items:
    TI sampleInterval: real;
    TI sampleDelay: real;
    TD total storedSpeed: integer;
    TD total actualSpeed: integer;
    event sendSpeed (integer, integer);
    event sampleSpeed;

```

```
axioms:
vars:
    deltaSpeed: integer;
    speed: integer;
formulae:
    SpeedValues:
        actualSpeed >= 0 and storedSpeed >= 0;

    BeginSample:
        SomP (storedSpeed = 0 & sampleSpeed);

    SamplingDefinition:
        sampleSpeed implies Futr (sampleSpeed,
           sampleInterval) and not Lasts (sampleSpeed,
           sampleInterval);

    SamplingAction:
        sampleSpeed implies Futr (deltaSpeed = actualSpeed
           - storedSpeed and speed = actualSpeed and
           sendSpeed (deltaSpeed, speed) and Lasts
           (storedSpeed = actualSpeed, sampleInterval),
           sampleDelay);

    SendSpeed:
        deltaSpeed = actualSpeed - storedSpeed and
           actualSpeed = speed and sendSpeed (deltaSpeed,
           speed) implies Past (sampleSpeed, sampleDelay);

end
```

Listing 3: EngineSpeedSensor.trio

```
class EngineSpeedSensor (const sampleInterval, const
   sampleDelay)

signature:

visible: actualSpeed, sendSpeed;

temporal domain: real;

items:
    TI sampleInterval: real;
    TI sampleDelay: real;
    TD total actualSpeed: integer;
    event sendSpeed (integer);
    event sampleSpeed;

axioms:
vars:
    speed: integer;
formulae:
    SpeedValues:
        actualSpeed >= 0;

    BeginSample:
        SomP (sampleSpeed);

    SamplingDefinition:
        sampleSpeed implies Futr (sampleSpeed,
            sampleInterval) and not Lasts (sampleSpeed,
            sampleInterval);

    SampleSpeedActions:
        sampleSpeed implies Futr (actualSpeed = speed and
            sendSpeed (speed), sampleDelay);

    SendSpeed:
        actualSpeed = speed and sendSpeed (speed) implies
            Past (sampleSpeed, sampleDelay);

end
```

3 PlanetaryGearSet Class

The *PlanetaryGearSet* class is formalized thanks to the code reported in Listing 4.

The Planetary Gear Set guarantees that every time a gear shift event occurs the `actualGear` will be maintained until the shift is finished.

Inside this component are defined all axioms limiting gear shifts to effective ones only (e.g. it is impossibile to shift down a gear if `actualGear` is `First`). The Planetary Gear Set permits to shift up to two gear at the same time (as the specification asks), however, the Transmission Control Unit doesn't use this possibility because in a real Planetary Gear Set this is not possibile.

Moreover, through the formalization of the Planetary Gear Set we impose that we can't receive a gear shift event if we are in the middle of a gear shift. Different gear shifting times are defined for different gears and different steps.

The gears `Drive`, `Park`, and `Reverse` can be selected if and only if the transmission shaft is decoupled from the engine.

The state of the Planetary Gear Set changes if and only if an event occurs.

Listing 4: PlanetaryGearSet.trio

```
class PlanetaryGearSet (const singleGearShiftDelay, const
   dualGearShiftDelay, const driveGearShiftDelay, const
   parkGearShiftDelay, const reverseGearShiftDelay)

signature:

visible:
    actualGear,
    transmissionShaftState;
    gearShift,
    gearDrive,
    gearPark,
    gearReverse,

temporal domain: real;

domains:
    Gear: {First, Second, Third, Park, Reverse};
    TransmissionShaftState: {Attached, Detached};
    ShiftWidth: 1..2;
    ShiftDirection: {Up, Down};

items:
    TI singleGearShiftDelay: real;
    TI dualGearShiftDelay: real;
    TI driveGearShiftDelay: real;
    TI parkGearShiftDelay: real;
    TI reverseGearShiftDelay: real;
    TD total actualGear: Gear;
    TD total transmissionShaftState: TransmissionShaftState;
    event gearShift (ShiftWidth, ShiftDirection);
    event gearDrive;
    event gearPark;
    event gearReverse;

axioms:
vars:
    gearShiftWidth: ShiftWidth;
    gearShiftWidth2: ShiftWidth;
    gearShiftDirection: ShiftDirection;
    gearShiftDirection2: ShiftDirection;
    gear: Gear;
formulae:
    Annotations:
        actualGear = Park implies transmissionShaftState = Detached;

    GearDriveShift:
        (actualGear = Reverse and gearDrive implies (Lasts (actualGear = Reverse, driveGearShiftDelay) and Futr (actualGear = First, driveGearShiftDelay))) and
        (actualGear = Park and gearDrive implies (Lasts (actualGear = Park, driveGearShiftDelay) and Futr (actualGear = First, driveGearShiftDelay))) and
        (actualGear = First or actualGear = Second or actualGear = Third implies not gearDrive) and
        (gearDrive iff transmissionShaftState = Detached);

    GearShiftsFirst:
        (actualGear = First implies Alw (not gearDrive and not ex gearShiftWidth (gearShiftDirection = Down and gearShift (gearShiftWidth, gearShiftDirection)))) and
        (actualGear = First and gearShiftWidth = 1 and
        gearShiftDirection = Up and gearShift
        (gearShiftWidth, gearShiftDirection) implies
        Lasts (actualGear = First,
        singleGearShiftDelay) and Futr (actualGear =
        Second, singleGearShiftDelay)) and
        (actualGear = First and gearShiftWidth = 2 and
        gearShiftDirection = Up and gearShift
        (gearShiftWidth, gearShiftDirection) implies
        Lasts (actualGear = First, dualGearShiftDelay)
        and Futr (actualGear = Third,
        dualGearShiftDelay);

    GearShiftsSecond:
        (actualGear = Second implies Alw (not gearDrive and
             not gearPark and not gearReverse and not ex
        gearShiftDirection (gearShiftWidth = 2 and
        gearShift (gearShiftWidth,
        gearShiftDirection)))) and
        (actualGear = Second and gearShiftWidth = 1 and
        gearShiftDirection = Up and gearShift
        (gearShiftWidth, gearShiftDirection) implies
        Lasts (actualGear = Second,
        singleGearShiftDelay) and Futr (actualGear =
        Third, singleGearShiftDelay)) and
        (actualGear = Second and gearShiftWidth = 1 and
        gearShiftDirection = Down and gearShift
        (gearShiftWidth, gearShiftDirection) implies
        Lasts (actualGear = Second,
        singleGearShiftDelay) and Futr (actualGear =
        First, singleGearShiftDelay));

    GearShiftsThird:
        (actualGear = Third implies Alw (not gearDrive and
        not gearPark and not gearReverse and not ex
        gearShiftWidth (gearShiftDirection = Up and
        gearShift (gearShiftWidth,
        gearShiftDirection)))) and
        (actualGear = Third and gearShiftWidth = 1 and
        gearShiftDirection = Down and gearShift
        (gearShiftWidth, gearShiftDirection) implies
        Lasts (actualGear = Third,
        singleGearShiftDelay) and Futr (actualGear =
        Second, singleGearShiftDelay)) and
```

```
        (actualGear = Third and gearShiftWidth = 2 and
            gearShiftDirection = Down and gearShift
            (gearShiftWidth, gearShiftDirection) implies
            Lasts (actualGear = Third, dualGearShiftDelay)
            and Futr (actualGear = First,
            dualGearShiftDelay));

    GearShiftsReverse:
        (actualGear = Reverse implies Alw (not gearReverse
            and all gearShiftWidth, gearShiftDirection (not
            gearShift (gearShiftWidth,
            gearShiftDirection)))) and
        (actualGear = Reverse implies SomF (gearDrive or
            gearPark)) and (actualGear = First and
            gearReverse implies Lasts (actualGear = First,
            reverseGearShiftDelay) and Futr (actualGear =
            Reverse, reverseGearShiftDelay)) and
        (actualGear = Park and gearReverse implies Lasts
            (actualGear = Park, reverseGearShiftDelay) and
            Futr (actualGear = Reverse,
            reverseGearShiftDelay)) and
        (gearReverse iff transmissionShaftState =
            Detached);

    GearShiftsPark:
        (actualGear = Park implies Alw (not gearPark and
            all gearShiftWidth, gearShiftDirection (not
            gearShift (gearShiftWidth,
            gearShiftDirection)))) and
        (actualGear = Park implies SomF (gearDrive or
            gearReverse)) and
        (actualGear = First and gearPark implies Lasts
            (actualGear = First, parkGearShiftDelay) and
            Futr (actualGear = Park, parkGearShiftDelay))
            and
        (actualGear = Reverse and gearPark implies Lasts
            (actualGear = Reverse, parkGearShiftDelay) and
            Futr (actualGear = Park, parkGearShiftDelay)Futr
             (actualGear = Park, parkGearShiftDelay)) and
        (gearPark iff transmissionShaftState = Detached);
    GearShiftsTimings:
        all gearShiftDirection ((actualGear = First or
            actualGear = Second or actualGear = Third) and
            gearShiftWidth = 1 and gearShift
            (gearShiftWidth, gearShiftDirection) implies not
             Lasts (gearDrive or gearPark or gearReverse or
            ex gearShiftWidth2, gearShiftDirection2
            (gearShift (gearShiftWidth2,
            gearShiftDirection2)), singleGearShiftDelay))
            and
        all gearShiftDirection ((actualGear = First or
            actualGear = Third) and gearShiftWidth = 2 and
            gearShift (gearShiftWidth, gearShiftDirection)
            implies not Lasts (gearDrive or gearPark or
            gearReverse or ex gearShiftWidth2,
            gearShiftDirection2 (gearShift (gearShiftWidth2,
             gearShiftDirection2)), dualGearShiftDelay)) and
        ((actualGear = Reverse and gearDrive) implies not
            Lasts (gearDrive or gearPark or gearReverse or
            ex gearShiftWidth2, gearShiftDirection2
            (gearShift (gearShiftWidth2,
            gearShiftDirection2)), driveGearShiftDelay)) and
        ((actualGear = Reverse and gearPark) implies not
            Lasts (gearDrive or gearPark or gearReverse or
            ex gearShiftWidth2, gearShiftDirection2
            (gearShift (gearShiftWidth2,
            gearShiftDirection2)), parkGearShiftDelay)) and
        ((actualGear = Park and gearDrive) implies not
            Lasts (gearDriver or gearPark or gearReverse or
            ex gearShiftWidth2, gearShiftDirection2
            (gearShift (gearShiftWidth2,
            gearShiftDirection2)), driveGearShiftDelay)) and
        ((actualGear = Park and gearReverse) implies not
            Lasts (gearDrive or gearPark or gearReverse or
            ex gearShiftWidth2, gearShiftDirection2
            (gearShift (gearShiftWidth2,
            gearShiftDirection2)), reverseGearShiftDelay));

    Nothing:
        all gear (actualGear = gear and not (all
            gearShiftWidth, gearShiftDirection (gearShift
            (gearShiftWidth, gearShiftDirection)) or
            gearDrive or gearPark or gearReverse) implies
            UpToNow (actualGear = gear) and NowOn
            (actualGear = gear));

```

```
end
```

4 HydraulicSystem Class

The *HydraulicSystem* class is formalized thanks to the code reported in Listing 5.

The first assumption we made before modelling the Hydraulic System was that every valve and electrovalve configuration imposes the same fluid propagation delay; this means that for every command that the Hydraulic System propagates the delay will always be the same. This behavior is formalized with the `time independent constant fluidPropagationDelay`.

The *manual valve*, which permit the driver to manually select the gear mode, is modelled thanks to the `gearHandle` event and the `GearHandle` axiom. During the time in which the Hydraulic System propagate a command there can be no `gearHandle` event which somehow means the fluid propagation is faster then the driver reaction time (which is a realistic assumption).

Moreover, thanks to the `MutualExclusion` axiom, it's impossibile to generate two `gearHandle` event at the same time which means that the gear handle can't be for example in Park and Drive mode at the same instant.

Listing 5: HydraulicSystem.trio

```
class HydraulicSystem (const fluidPropagationDelay)

signature:

visible:
    gearHandle,
    gearShift,
    gearDrive,
    gearPark,
    gearReverse,
    controlGearShift;

temporal domain: real;

domains:
    GearPosition: {Drive, Park, Reverse};
    ShiftWidth: 1..2;
    ShiftDirection: {Up, Down};

items:
    TI fluidPropagationDelay: real;
    event gearHandle (GearPosition);
    event gearShift (ShiftWidth, ShiftDirection);
    event gearDrive;
    event gearPark;
    event gearReverse;
    event controlGearShift (ShiftWidth, ShiftDirection);

axioms:
vars:
    gear: GearPosition;
    gear2: GearPosition;
    gearShiftWidth: ShiftWidth;
    gearShiftWidth2: ShiftWidth;
    gearShiftDirection: ShiftDirection;
    gearShiftDirection2: ShiftDirection;
formulae:
    GearHandleCommand:
        (gear = Drive and gearHandle (gear) implies not
            Lasts (all gear2 (gear2 <> gear implies
            gearHandle (gear2) or ex gearShiftWidth,
            gearShiftDirection (controlGearShift
            (gearShiftWidth, gearShiftDirection))),
            fluidPropagationDelay) and Futr (gearDrive,
            fluidPropagationDelay) and
        (gear = Park and gearHandle (gear) implies not
            Lasts (all gear2 (gear2 <> gear implies
            gearHandle (gear2) or ex gearShiftWidth,
            gearShiftDirection (controlGearShift
            (gearShiftWidth, gearShiftDirection))),
            fluidPropagationDelay) and Futr (gearPark,
            fluidPropagationDelay) and
        (gear = Reverse and gearHandle (gear) implies not
            Lasts (all gear2 (gear2 <> gear implies
            gearHandle (gear2) or ex gearShiftWidth,
            gearShiftDirection (controlGearShift
            (gearShiftWidth, gearShiftDirection))),
            fluidPropagationDelay) and Futr (gearReverse,
            fluidPropagationDelay);

    PropagateGearShiftCommand:
        all gearShiftWidth, gearShiftDirection
            (controlGearShift (gearShiftWidth,
            gearShiftDirection) implies not Lasts (gearDrive
             or gearPark or gearReverse or ex
            gearShiftWidth2, gearShiftDirection2
            (controlGearShift (gearShiftWidth2,
            gearShiftDirection2)), fluidPropagationDelay)
            and Futr (gearShift (gearShiftWidth,
            gearShiftDirection), fluidPropagationDelay));

    MutualExclusions:
        all gear (gearHandle (gear) implies all gear2 (gear
            <> gear2 implies not gearHandle (gear2)));

end
```

5 TransmissionControlUnit Class

The *TransmissionControlUnit* class is formalized thanks to the code reported in Listing 6.

Our first formalization of the Transmission Control Unit didn't take in account the possibility to have asynchronous sensors; the latest version of the Transmission Control Unit permits to manage asynchronous sensors thanks to internal memory modelled with three `time dependent total` values.

When handle the necessity to scale gears till the `First` with the assumption that the human reaction is way slower than sampling frequency and mechanical reactions, so, when the vehicle stops, the axiom which handle the gear scale manage to be "active" the necessary amount of times to scale all the gears.

The Transmission Control Unit guarantees that it doesn't raise more than one gear shift event per instant and it receives at most one event per instant from each sensor (this is described also in Section 2 and so guaranteed in VehicleSpeedSensor and EngineSpeedSensor class).

Listing 6: TransmissionControlUnit.trio

```
class TransmissionControlUnit

signature:

visible:
    controlGearShift,
    receiveEngineSpeed,
    receiveVehicleSpeed;

temporal domain: real;

domains:
    ShiftWidth: 1..2;
    ShiftDirection: {Up, Down};

items:
    TD total storedEngineSpeed: integer;
    TD total storedDeltaVehicleSpeed: integer;
    TD total storedVehicleSpeed: integer;
    event controlGearShift (ShiftWidth, ShiftDirection);
    event receiveEngineSpeed (integer);
    event receiveVehicleSpeed (integer, integer);

axioms:
vars:
    engineSpeed: integer;
    engineSpeed1: integer;
    engineSpeed2: integer;
    deltaVehicleSpeed: integer;
    deltaVehicleSpeed1: integer;
    deltaVehicleSpeed2: integer;
    vehicleSpeed: integer;
    vehicleSpeed1: integer;
    vehicleSpeed2: integer;
    gearShiftWidth1: ShiftWidth;
    gearShiftWidth2: ShiftWidth;
    gearShiftDirection1: ShiftDirection;
    gearShiftDirection2: ShiftDirection;
formulae:
    GearShifts:
        (receiveEngineSpeed (engineSpeed) and
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed) and engineSpeed >= 3000 and
            vehicleSpeed > 0 implies gearShiftWidth1 = 1 and
             gearShiftDirection1 = Up and controlGearShift
            (gearShiftWidth1, gearShiftDirection1)) and
        (receiveEngineSpeed (engineSpeed) and all
            deltaVehicleSpeed, vehicleSpeed (not
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed)) and engineSpeed >= 3000 and
            storedVehicleSpeed > 0 implies gearShiftWidth1 =
             1 and gearShiftDirection1 = Up and
            controlGearShift (gearShiftWidth1,
            gearShiftDirection1)) and
        (all engineSpeed (not receiveEngineSpeed
            (engineSpeed)) and receiveVehicleSpeed
            (deltaVehicleSpeed, vehicleSpeed) and
            storedEngineSpeed >= 3000 and vehicleSpeed > 0
            implies gearShiftWidth1 = 1 and
            gearShiftDirection1 = Up and controlGearShift
            (gearShiftWidth1, gearShiftDirection1)) and
        (receiveEngineSpeed (engineSpeed) and
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed) and engineSpeed <= 1500 and
            deltaVehicleSpeed <= 0 implies gearShiftWidth1 =
             1 and gearShiftDirection1 = Down and
            controlGearShift (gearShiftWidth1,
            gearShiftDirection1)) and
        (receiveEngineSpeed (engineSpeed) and all
            deltaVehicleSpeed, vehicleSpeed (not
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed)) and engineSpeed <= 1500 and
            storedDeltaVehicleSpeed <= 0 implies
            gearShiftWidth1 = 1 and gearShiftDirection1 =
            Down and controlGearShift (gearShiftWidth1,
            gearShiftDirection1)) and
        (all engineSpeed (not receiveEngineSpeed
            (engineSpeed)) and receiveVehicleSpeed
            (deltaVehicleSpeed, vehicleSpeed) and
            storedEngineSpeed <= 1500 and (deltaVehicleSpeed
             <= 0 or vehicleSpeed = 0) implies
            gearShiftWidth1 = 1 and gearShiftDirection1 =
            Down and controlGearShift (gearShiftWidth1,
            gearShiftDirection1)) and
        (receiveEngineSpeed (engineSpeed) and
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed) and engineSpeed <= 1500 and
            deltaVehicleSpeed > 0 implies all
            gearShiftWidth1, gearShiftDirection1 (not
            controlGearShift (gearShiftWidth1,
            gearShiftDirection1))) and
        (receiveEngineSpeed (engineSpeed) and all
            deltaVehicleSpeed, vehicleSpeed (not
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed)) and engineSpeed <= 1500 and
            storedDeltaVehicleSpeed >= 0 and
            storedVehicleSpeed > 0 implies all
            gearShiftWidth1, gearShiftDirection1 (not
            controlGearShift (gearShiftWidth1,
            gearShiftDirection1))) and
        (all engineSpeed (not receiveEngineSpeed
            (engineSpeed)) and receiveVehicleSpeed
            (deltaVehicleSpeed, vehicleSpeed) and
            storedEngineSpeed <= 1500 and deltaVehicleSpeed
            >= 0 and vehicleSpeed > 0 implies all
            gearShiftWidth1, gearShiftDirection1 (not
            controlGearShift (gearShiftWidth1,
            gearShiftDirection1))) and
        (receiveEngineSpeed (engineSpeed) and
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed) and engineSpeed >= 1500 and
```

```
            engineSpeed < 3000 implies all gearShiftWidth1,
            gearShiftDirection1 (not controlGearShift
            (gearShiftWidth1, gearShiftDirection1))) and
        (receiveEngineSpeed (engineSpeed) and all
            deltaVehicleSpeed, vehicleSpeed (not
            receiveVehicleSpeed (deltaVehicleSpeed,
            vehicleSpeed)) and engineSpeed >= 1500 and
            engineSpeed < 3000 implies all gearShiftWidth1,
            gearShiftDirection1 (not controlGearShift
            (gearShiftWidth1, gearShiftDirection1))) and
        (all engineSpeed (not receiveEngineSpeed
            (engineSpeed)) and receiveVehicleSpeed
            (deltaVehicleSpeed, vehicleSpeed) and
            storedEngineSpeed >= 1500 and storedEngineSpeed
            < 3000 implies all gearShiftWidth1,
            gearShiftDirection1 (not controlGearShift
            (gearShiftWidth1, gearShiftDirection1))) and
        (all engineSpeed (not receiveEngineSpeed
            (engineSpeed)) and all deltaVehicleSpeed,
            vehicleSpeed (not receiveVehicleSpeed
            (deltaVehicleSpeed, vehicleSpeed)) implies all
            gearShiftWidth1, gearShiftDirection1 (not
            controlGearShift (gearShiftWidth1,
            gearShiftDirection1)));

    ReceivingEventAction:
        all deltaVehicleSpeed1, vehicleSpeed1
            (receiveVehicleSpeed (deltaVehicleSpeed1,
            vehicleSpeed1) implies Until
            (storedDeltaVehicleSpeed = deltaVehicleSpeed1
            and storedVehicleSpeed = vehicleSpeed1, ex
            deltaVehicleSpeed2, vehicleSpeed2
            (receiveVehicleSpeed (deltaVehicleSpeed2,
            vehicleSpeed2)))) and
        all engineSpeed1 (receiveEngineSpeed
            (engineSpeed1) implies Until (storedEngineSpeed
            = engineSpeed1, ex engineSpeed2
            (receiveEngineSpeed (engineSpeed2))));

    MutualExclusions:
        all gearShiftWidth1, gearShiftDirection1
            (controlGearShift (gearShiftWidth1,
            gearShiftDirection1) implies all
            gearShiftWidth2, gearShiftDirection2
            (gearShiftWidth1 <> gearShiftWidth2 and
            gearShiftDirection1 <> gearShiftDirection2
            implies not controlGearShift (gearShiftWidth2,
            gearShiftDirection2))) and
        all engineSpeed1 (receiveEngineSpeed
            (engineSpeed1) implies all engineSpeed2
            (engineSpeed2 <> engineSpeed1 implies not
            receiveEngineSpeed (engineSpeed2))) and
        all deltaVehicleSpeed1, vehicleSpeed1
            (receiveVehicleSpeed (deltaVehicleSpeed1,
            vehicleSpeed1) implies all deltaVehicleSpeed2,
            vehicleSpeed2 (deltaVehicleSpeed2 <>
            deltaVehicleSpeed1 and vehicleSpeed2 <>
            vehicleSpeed1 implies not receiveVehicleSpeed
            (deltaVehicleSpeed2, vehicleSpeed2)));

end
```

6 Annotations

During the last phase of our modelling we decided not to formalize the *Torque Converter* and this decision depends on the way the Torque Converter works.

The Torque Converter is a mechanical component that works coupling and decoupling the *Transmission Shaft* and the *Engine Shaft*. It solves is duty without the necessity to receive commands from any component of the system and this is the cause we have decided to remove it from our model.

Anyway, the state of the Torque Converter is really important for the system since it gives information that permits to insert or not to insert some gears and other details that aren't taken into account in this project.

7 Properties

Listing 7: Property 1

```
actualGear = First and controlGearShift (1, Up) implies
    Futr (actualGear = Second, fluidPropagationDelay +
    singleGearShiftDelay)
```

Listing 8: Property 2

```
gear = Park and gearHandle (gear) and Futr (actualGear =
    Park, fluidPropagationDelay + parkGearShiftDelay) iff
    Lasts (transmissionShaftState = Detached,
    fluidPropagationDelay + parkGearShiftDelay)
```