TUM

INSTITUT FÜR INFORMATIK

From Semiformal Requirements To Formal Specification

Maria Spichkova

TUM-I1019
Oktober 10

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-10-I1019-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Contents

1 Introduction

In this paper we present an approach for translating semiformal specification in formal ones. This approach was applied on the case study during the project DENTUM between Denso Deutschland GmbH and the chair for Software & Systems Engineering at Technische Universität München (see [2]). The goal of this project was to define a methodology for the model-based development of automotive systems. This methodology was evaluated by developing an Adaptive Cruise Control (ACC) system with Pre-Crash Safety (PCS) functionality.

This document explains an extension part of the methodology as well as the corresponding part of the case study in detail to teach the reader the details of the work that has been done.

The starting point of these approach is a semiformal requirement specification the according to the ideas presented in [3] (see also Section 2).

On base of these requirements we specify the corresponding message sequence charts (MSCs, see [4, 5]) and translate them to a formal specification in Focus [1], a framework for formal specifications and development of interactive systems. If some missing requirements are found, they are added to the final version of the requirement specification. If some inconsistencies in the requirements are found, the corresponding requirements are corrected in the final version of the requirement specification. In the requirement specification we focus on two parts of it – logical interface and formulated requirements.

Given a system, represented in Focus, one can verify its properties by translating the specification to a Higher-Order Logic and subsequently using the theorem prover Isabelle/HOL or the point of disagreement will be found. The translation can be done according to the approach "Focus on Isabelle" [7]. Moreover, using this approach one can validate the refinement relation between two given systems, as well as make automatic correctness proofs of syntactic interfaces for specified system components. Having a Focus specification, we can schematically translate it to a specification in Hight-Order Logic and verify properties of the specified system.

To remove the technical details which belongs to the requirement specification from Denso Deutschland GmbH we have to replace here the real values by constants.

2 Semiformal Specification

Based on an initial set of requirements, the informal specification[1] is structured and subsequently specified with the help of pre-defined text patterns. Furthermore, the logical interface of the system as well as the main system states are identified. For this purpose we use a simplified versions of an approach presented in [2, 3], which we extended according to the needs of the overall development approach.

An informal specification consists of a set of words, which can be distinguished into two categories:

- Content words: these are system-specific words or phrases, e.g. *"system is initialized"* or *"Off-button is pressed"*. The set of all content words forms the logical interface of the system, which can be understood as some kind of (domain specific, system-dependent) glossary that must be defined in addition.
- Relation words (keywords): these are domain-independent words (e.g. "if", "then", "else"). These words form relationships between the content words.

[1] The presented methodology focuses only on functional requirements, including timing aspects.

A semiformal specification consists of a number of requirements described using the following textual patterns:[2]

WHILE ⟨*Some state*⟩
IF ⟨*Some event occurs or some state changes*⟩
THEN ⟨*Some event occurs or some state changes*⟩

An *event* describes a *point in time*, in which the system observes or does something; the duration of the event is not important, e.g., "driver presses a button". A *state* describes a system or component state *within some time period*, e.g., "a button is pressed". Strictly speaking, all states of a state space are disjunct, but in some cases it is more efficient to use a state hierarchy that must be described separately. For example, if we assume two disjoint states, "system is active" and "system is inactive", the first of them might have three more-specific substates: "system is in initialization state", "system performs some action A" and "system performs some action B"

The semiformal specification can be given in a simple tabular form with the following columns:

- ID: the unique requirement identifier.
- Description: the semiformal description of the requirement.
- Original Description: sentences from the informal specification, which were reformulated to the semiformal requirement.
- MSC (optional, only in case the MSC representation is used): name of MSC to which the semiformal requirement belongs.
- Remarks (optional): some remarks to the requirement.
- Alternative Description (optional): alternative formulation of the semiformal description of the requirement. This is needed if a team of development engineers cooperates for building this specification trying find the most appropriate formulation of specification.

Using such a simple tabular description to structure the information from the informal specification, we can find out quite fast, which information is missing. Furthermore, we identify possible synonyms that must be unified before proceeding to a formal specification. Analysis of the semiformal specification document should also yield sentences, which need to be reformulated or extended.

3 MSC Specifications

In this section we present the MSC specification which is based on the semiformal specification developed within the chair for Software & Systems Engineering during the project DENTUM. Some case study details are omitted here, we present only the final state of the specifications.

We use here a standard MSC notation, but for readability add also a color marking: blue hexagon denote the modes of ACC an PCS,

Using the MSCs we extend the standard notation by the following syntax:

- Blue hexagons denote states (modes of ACC an PCS) and correspond to local (state) variables in FOCUS.

[2] In some cases either the WHILE-part or the IF-part can be omitted.

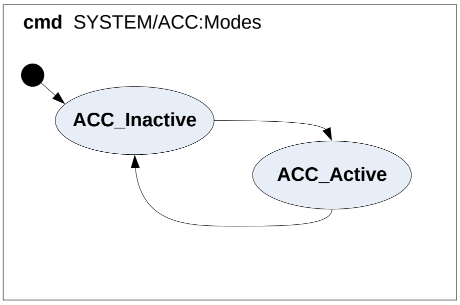
- ◇ Green hexagons denote “substates” to represent local variables variables and correspond to local (state) variables in Focus.
- ◇ Light blue rectangles denotes some operations and correspond to state changes in Focus.

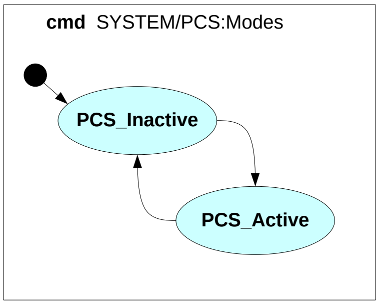
All the requirements that are added to the specification on the MSCs level are marked by red rectangles to show which kind of missed requirements can be find out using this approach.

3.1 Modes

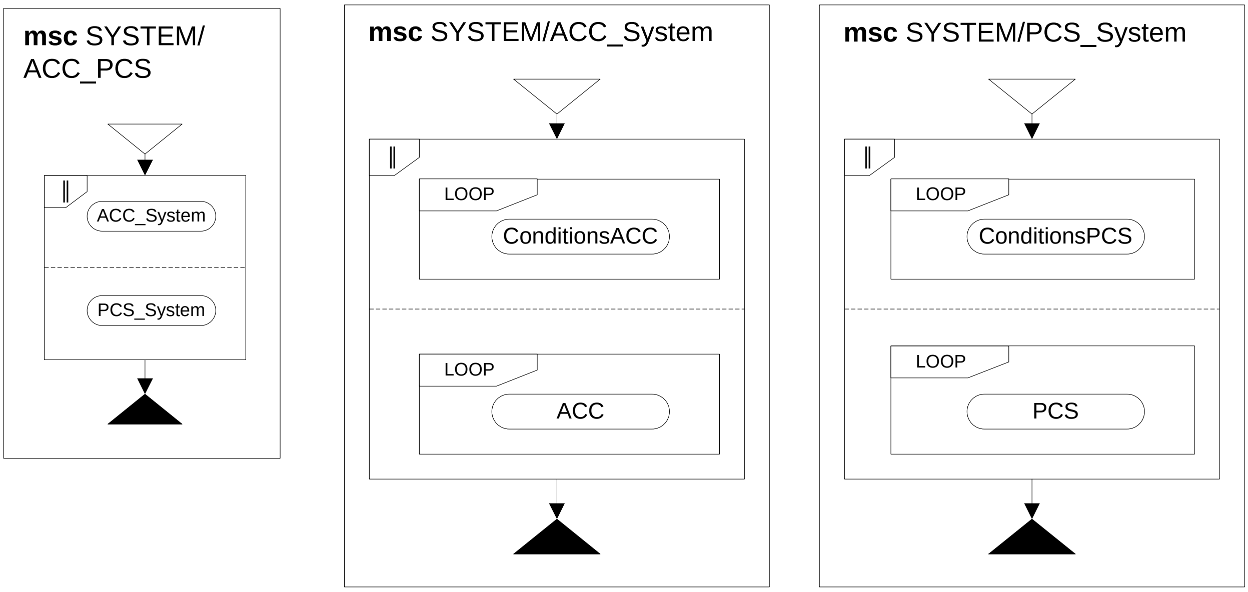
The ACC and the PCS have according to the semiformal requirement specification only two modes – *active* and *inactive*. This must be true also for the MSC and the Focus specifications.

The MSCs *ACC* : *Modes* and *PCS* : *Modes* are two state diagrams to represented the state transitions in general. They follows to the logical interface description in the requirements document.

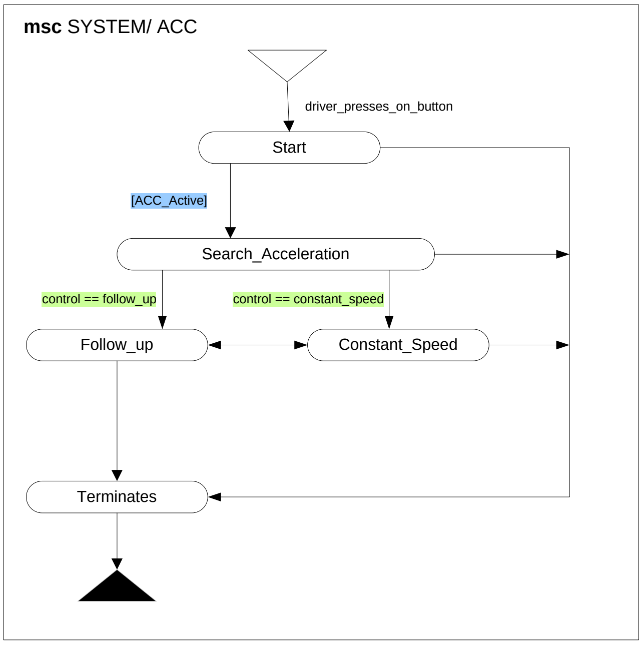


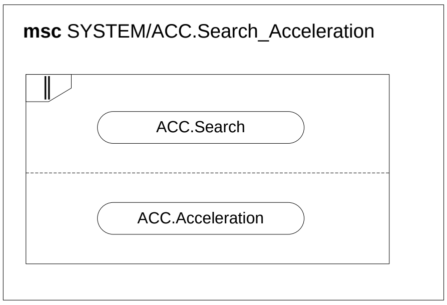


3.2 System workflow



The MSCs *ACC_PCS*, *ACC_System* and *PCS_System* describe the system workflow, where the MSCs *ACC* and *search_Acceleration* describe the workflow of the ACC part of the system. These MSCs do not correspond to some particular requirements, but follow from them implicit – they corresponds to the logical interface description in the requirements document. There are many ways to decompose the system workflow, we choose one of them.





3.3 ConditionsACC

The MSC *ConditionsACC* is based on the requirement *ACC startable*:

WHILE vehicle speed is higher than $CONST_{minaccspeed1}$*km/h and lower than* $CONST_{maxaccspeed}$*km/h*
THEN acc is startable

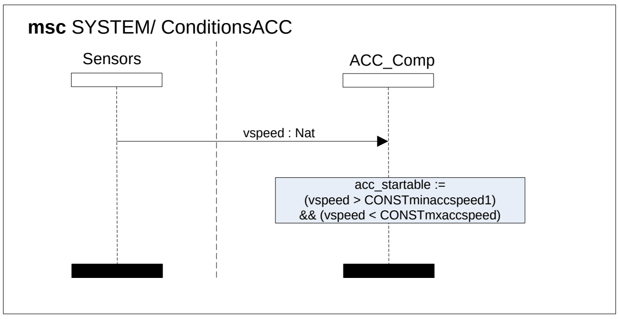
The natural language expression "vehicle speed" is represented in the MSC specification by a variable (signal) *vspeed* of type ℕ. Please note, that the signal *vspeed* is send every time unit.

From the natural language expression "acc is startable" follows, that we need to argue about an information, whether the ACC is startable or not. The most natural representation for this information is a boolean variable – we call it *acc_startable*. Now we can see, that the semiformal specification does not contain any explicit information, whether the ACC will be startable or not if the vehicle speed is lower than $CONST_{minaccspeed1}$km/h or higher than $CONST_{maxaccspeed}$km/h, but the original meaning of the requirement *ACC startable* is that the ACC is startable only if the vehicle speed is higher than $CONST_{minaccspeed1}$km/h and lower than $CONST_{maxaccspeed}$km/h. Thus, we need to add a new requirement to the semiformal specification:

WHILE vehicle speed is lower than $CONST_{minaccspeed1} km/h$ *and higher than* $CONST_{maxaccspeed} km/h$
THEN acc is non-startable

The logical variable *acc_startable* can be defined as follows:

$$(vspeed > CONST_{minaccspeed1})\&\&(vspeed < CONST_{maxaccspeed})$$

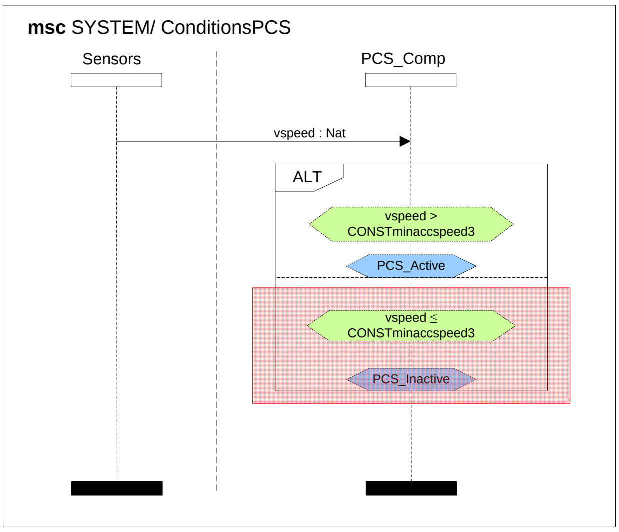

msc SYSTEM/ ConditionsACC
Sensors
ACC_Comp
vspeed : Nat
acc_startable :=
(vspeed > CONSTminaccspeed1)
&& (vspeed < CONSTmxaccspeed)

3.4 ConditionsPCS

The MSC *ConditionsPCS* is based on the requirement *PCS Active-Condition*:

WHILE vehicle speed is higher than $CONST_{minaccspeed3} km/h$
THEN pcs is active

The natural language expression “pcs is active” represents one of the states (modes) of the PCS: *PCS_Active*. We can see, that the semiformal specification does not contain any explicit information, whether the PCS will be active or inactive if the vehicle speed is lower than $CONST_{minaccspeed3}$ km/h, but the original meaning of the requirement *PCS Active-Condition* is that the PCS is active only if the vehicle speed is higher than $CONST_{minaccspeed3}$ km/h. Thus, we need to add a new requirement to the semiformal specification, and the MSC is extended by this information.



WHILE vehicle speed is lower than $CONST_{minaccspeed3} km/h$
THEN pcs is inactive

3.5 ACC.Start

The MSC *ACC.Start* is based on the requirement *ACC stars by driver interaction*

WHILE acc is startable
IF driver-presses-on-button
THEN acc starts

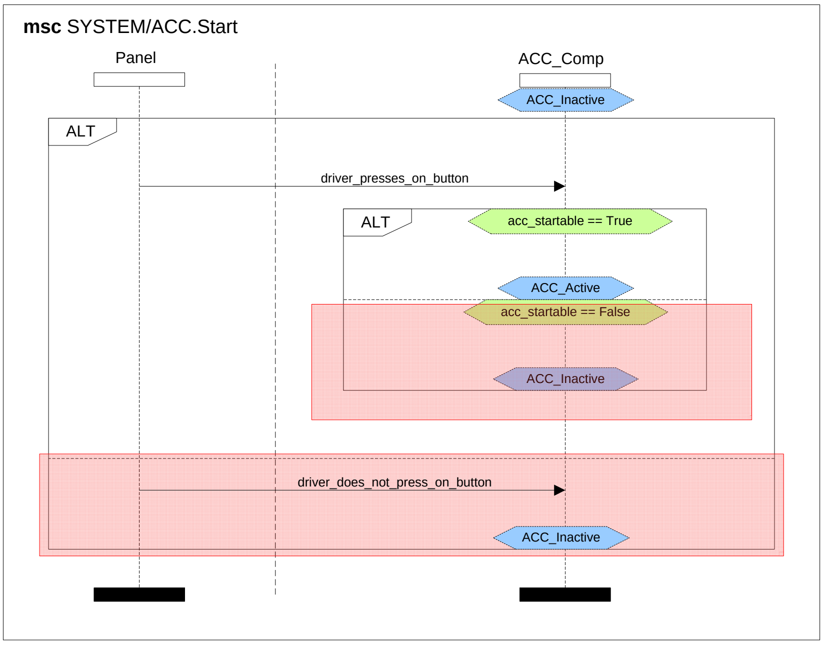
Here we do not have the information what happens, if the ACC is non-startable. We need to add a new requirement to the semiformal specification, and the MSC is extended by this information:

WHILE acc is non-startable
IF driver-presses-on-button
THEN acc does not start

The requirement *ACC stars by driver interaction only* was added to the final version of the semiformal specification by the same reason:

IF driver-doesn't-press-on-button
THEN acc doesn not start

This requirement is represented by the second ALT-part of the MSC *ACC.Start*.



The expressions “driver-presses-on-button”, “driver-doesn't-press-on-button” are specified by the corresponding events *driver_presses_on_button* and *driver_does_not_press_on_button*.

Please note, that according to the logical interface:

- either the event “driver_presses_on_button” or the event “driver_does_not_press_on_button” can happen,
- either the event “acc starts” or the event “acc doesn not start” can happen.

3.6 ACC.Acceleration

The MSC *ACC.Acceleration* is based on the requirement *ACC overruled by driver acceleration*:

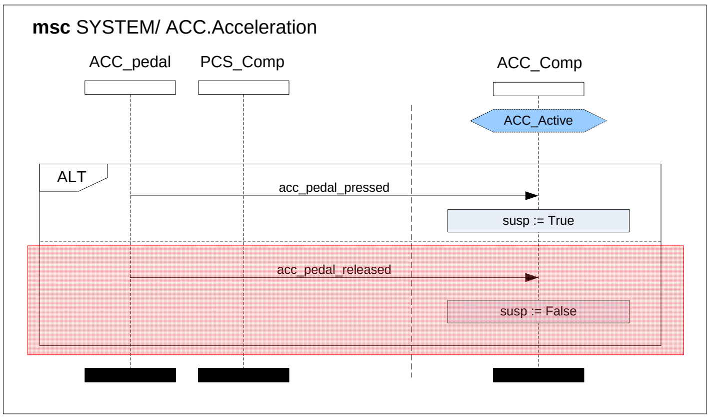
WHILE acc is active
IF driver-presses-acceleration-pedal
THEN acc MUST NOT brake

To represent the information “acc MUST NOT brake” we will use the variable *susp* of type $\mathbb{B}$ool. Its value will influence on the system behavior, described by the MSCs *Follow_up* and *Constant_Speed*, in the blocks[3] *AdjustVehicleSpeed2TargetSpeed* and *ControlDistance (current_distance)*

And again we need to add the requirement, which describes the opposite situation:

WHILE acc is active
IF driver-releases-acceleration-pedal
THEN acc CAN brake

The expressions “driver-presses-acceleration-pedal” and “driver-releases-acceleration-pedal” are represented in the MSC specification by the events *acc_pedal_pressed* and *acc_pedal_released* respectively.



[3] Please note, that these blocks are underspecified – the corresponding requirements are missed.

3.7 ACC.Search

The MSC *ACC.Search* is based on the requirements *Follow-Up-Control Condition* and *Constant-Speed-Control Condition* which are alternative, together with initializing requirements *Initial Following-Distance* and *Initial Target-Speed*:

WHILE acc is active
IF target vehicle is detected
THEN acc is in follow-up-control

WHILE acc is active AND no target vehicle is detected
THEN acc is in constant-speed-control

WHILE acc is active
IF acc starts follow-up-control
THEN acc sets following distance to "middle"

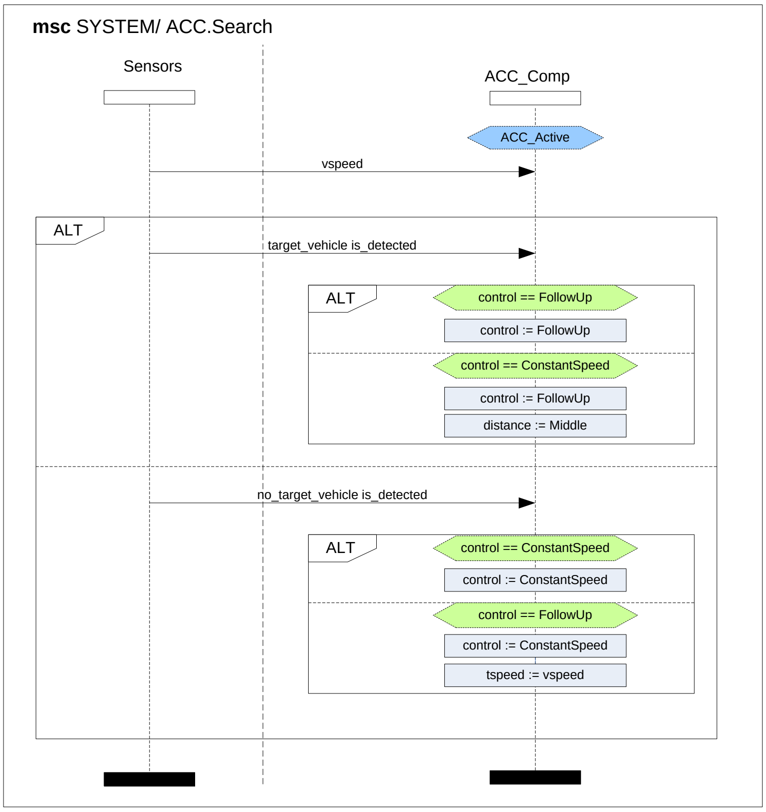
WHILE acc is active
IF acc starts constant-speed-control
THEN acc sets target speed to vehicle speed

The natural language expressions "target vehicle is detected" and "no target vehicle is detected" are represented by the signals *target_vehicle_is_detected* and *no_target_vehicle_is_detected* respectively. Please note, that every time unit either the signal *target_vehicle_is_detected* or the signal *no_target_vehicle_is_detected* is send. This information follows from the logical interface description.

To represent the information "acc is in follow-up-control" and "acc is in constant-speed-control" we use a variable, let call it *control*. This variable can have only two values, according to the two control modes of the ACC: *follow_up* and *constant_speed* (we define the corresponding data type *ControlType*).

We also need to introduce a new variable, let call it *tspeed*, to safe an information about the target speed.

We add the initializing requirements *acc sets following distance to "middle"* and *acc sets target-speed to current-speed* in the MSC by the corresponding hexagons immediately after the hexagons describing the substate from "constant speed" to "follow up" and from "follow up" to "constant speed" respectively.



3.8 ACC.Follow_up

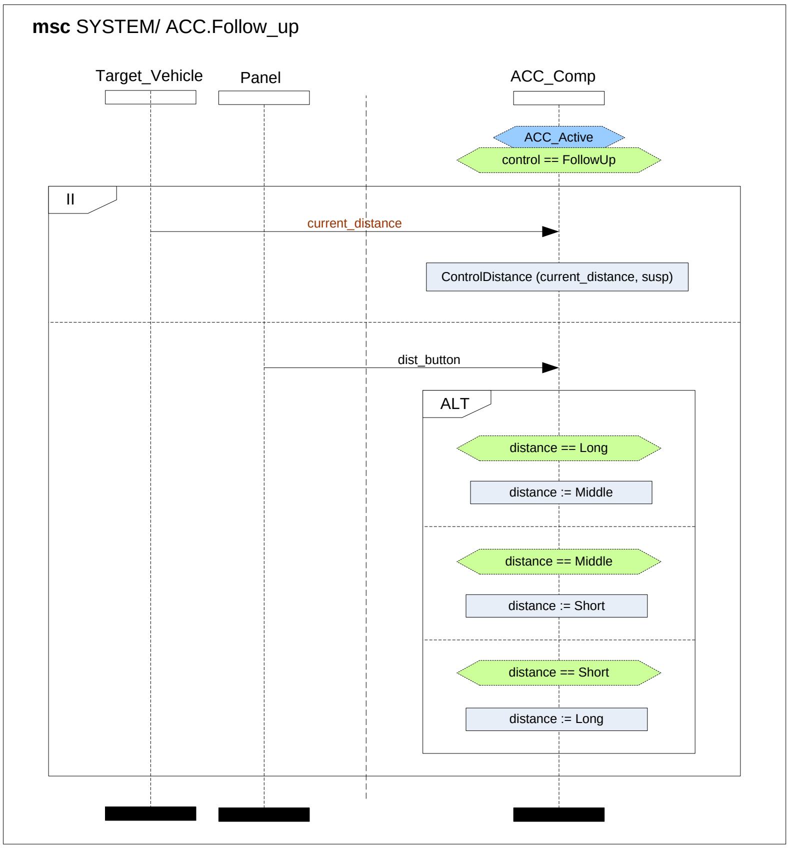
The MSC *ACC.Follow_up* is based on the requirements *Follow-Up-Control Behaviour*, *Changing Following-Distance (long ⇒ middle)*, *Changing Following-Distance (middle ⇒ short)* and *Changing Following-Distance (short ⇒ long)*.

WHILE acc is active AND acc is in follow-up-control
THEN acc controls following distance to target vehicle

WHILE acc is in follow-up-control AND following-distance is "long"
IF driver-presses-change-distance-button
THEN acc sets following distance to "middle"

WHILE acc is in follow-up-control AND following-distance is "middle"
IF driver-presses-change-distance-button
THEN acc sets following distance to "short"

WHILE acc is in follow-up-control AND following-distance is "short"
IF driver-presses-change-distance-button
THEN acc sets following distance to "long"



The expression "acc controls following distance to target vehicle" is underspecified in the semiformal specification. Thus, we will underspecify it also on the MSC diagram – we will represent it by the the MSC-block *ControlDistance*. As an input for this block the following distance to target vehicle will be taken – we represent it by the *current_distance* signal.

To represent the information "acc sets following distance to *middle*", "acc sets following distance to *short*" and "acc sets following distance to *long*" we introduce a variable, which will be of the data type *DistanceType*. Let call this variable *distance*.

The "change distance" requirements are alternative in the semiformal specification and they must be considered as concurrent to the *Follow-Up-Control Behaviour* requirement.

3.9 ACC.Constant_Speed

The MSC *ACC.Constant_Speed* is based on the requirements *Constant-Speed-Control Behavior*, *Incrementing Target-Speed* and *Decrementing Target-Speed*.

WHILE acc is in constant-speed-control
THEN adjust vehicle speed to target speed

WHILE acc is in constant-speed-control
IF driver-increases-target-speed
THEN acc increases target speed with $+CONST_{changespeed} km/h$

WHILE acc is in constant-speed-control
IF driver-decreases-target-speed
THEN acc decreases target speed with -$CONST_{changespeed}$*km/h*

The expression “adjust vehicle speed to target speed” is underspecified in the semiformal specification. Thus, we will underspecify it also on the MSC diagram – we will represent it by the the MSC-block *AdjustVehicleSpeed2RTargetSpeed*. As an input for this block the target speed and the current vehicle speed will be taken.

The requirements *Incrementing Target-Speed* and *Decrementing Target-Speed* are alternative. They both must be considered as concurrent to the requirement *Constant-Speed-Control Behavior*. The information “driver-increases-target-speed” and “driver-decreases-target-speed” will be represented by the signals *increase_speed* and *decrease_speed*, which can’t be send simultaneously (at the same time unit).

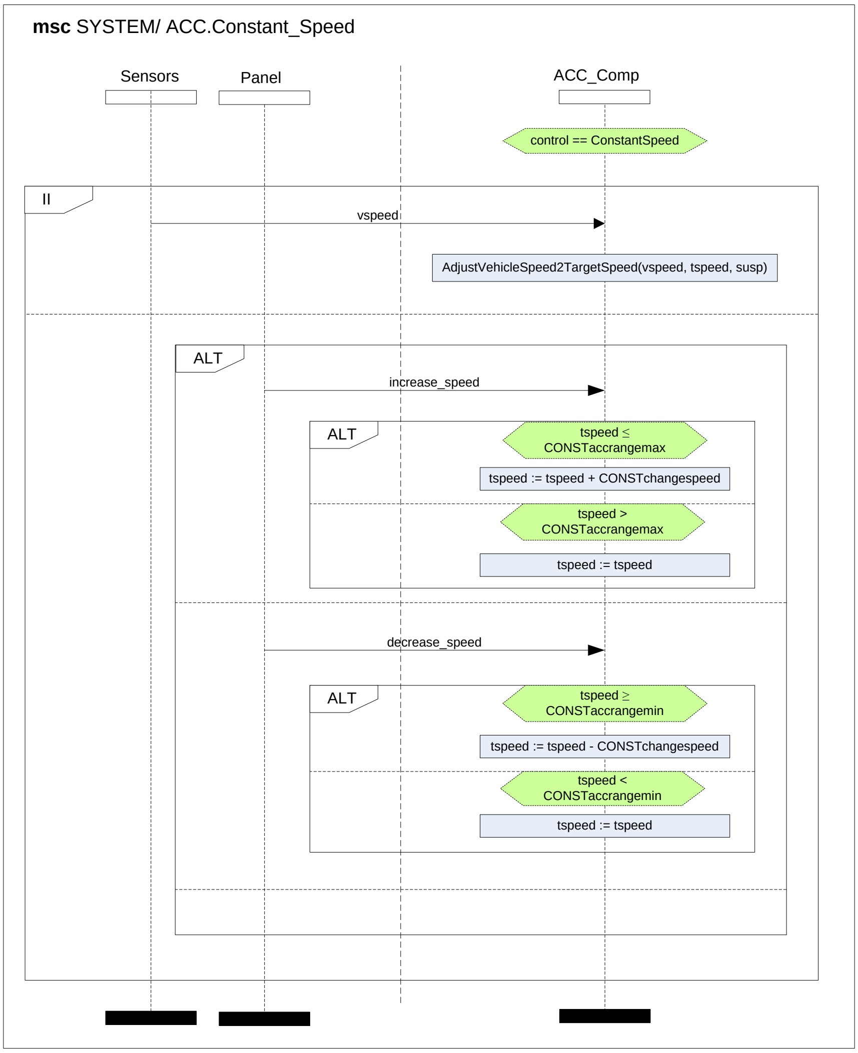
The comments “target speed cannot be increased to more than $CONST_{accrangemax}$km/h” and “target speed cannot be decreased to less than $CONST_{accrangemin}$km/h” from the semiformal specification must be added to this specification explicitly, as requirements. Optimized version of the requirements *Incrementing Target-Speed* and *Decrementing Target-Speed*:

WHILE acc is in constant-speed-control ***AND vehicle speed is lower or equal than*** $CONST_{accrangemax}$***km/h***
IF driver-increases-target-speed
THEN acc increases target speed with +$CONST_{changespeed}$*km/h*

WHILE acc is in constant-speed-control ***AND vehicle speed is higher than*** $CONST_{accrangemax}$***km/h***
IF driver-increases-target-speed
THEN target speed remains unchanged

WHILE acc is in constant-speed-control ***AND vehicle speed is higher or equal than*** $CONST_{accrangemin}$***km/h***
IF driver-decreases-target-speed
THEN acc decreases target-speed with -$CONST_{changespeed}$*km/h*

WHILE acc is in constant-speed-control ***AND vehicle speed is lower than*** $CONST_{accrangemin}$***km/h***
IF driver-decreases-target-speed
THEN target speed remains unchanged


msc SYSTEM/ ACC.Constant_Speed
Sensors
Panel
ACC_Comp
control == ConstantSpeed
II
vspeed
AdjustVehicleSpeed2TargetSpeed(vspeed, tspeed, susp)
ALT
increase_speed
ALT
tspeed ≤ CONSTaccrangemax
tspeed := tspeed + CONSTchangespeed
tspeed > CONSTaccrangemax
tspeed := tspeed
decrease_speed
ALT
tspeed ≥ CONSTaccrangemin
tspeed := tspeed - CONSTchangespeed
tspeed < CONSTaccrangemin
tspeed := tspeed

3.10 ACC.Terminates

The MSC *ACC.Terminates* is based on the requirements *ACC terminates by low speed*, *ACC terminates by driver pressing button*, *ACC terminates by driver brake pedal*, and *PCS-Brake suspends ACC*.

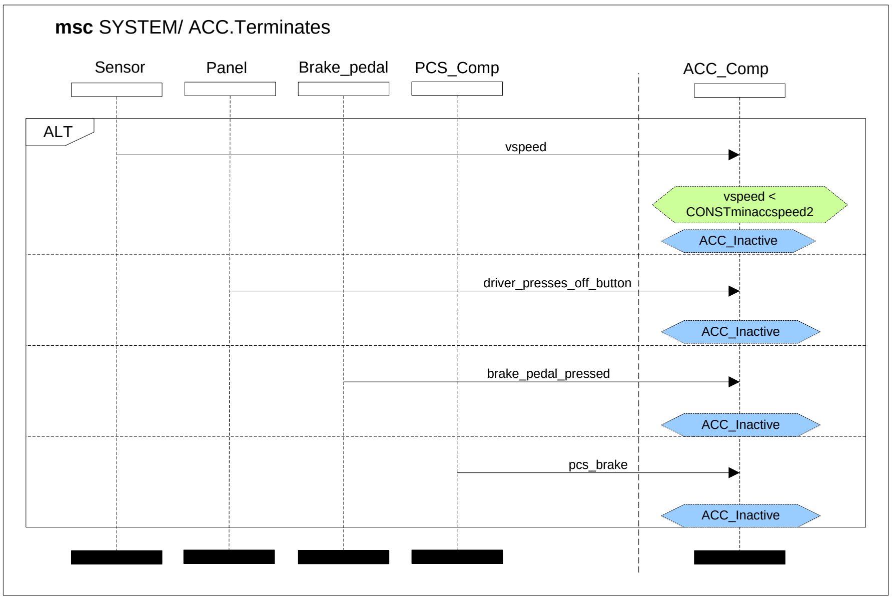
We delete from this requirements the while-part *WHILE acc is active*, because the result will be *acc_inactiv* – it does't matter, in which state it was before.

IF vehicle speed becomes lower than $CONST_{minaccspeed2} km/h$
THEN acc terminates

IF driver-presses-off-button
THEN acc terminates

IF driver operates brake pedal
THEN acc terminates

IF pcs-executes-brake
THEN acc terminates



The natural language expression “acc terminates” represents the information that the ACC comes to the state (mode) *ACC_Inactive*.

The expressions “driver-presses-off-button”, “driver operates brake pedal” and “pcs-executes-brake” are specified by the corresponding events *driver_presses_off_button*, *brake_pedal_pressed* and *pcs_brake*.

3.11 PCS

The MSC *PCS* is based on the requirements *PCS warning*, *PCS brake*, *PCS seat belt*, *PCS brake-end*, and *PCS-Brake overrules Driver-Acceleration*:

WHILE pcs is active
IF collision-time is smaller than limit
THEN pcs warns-driver-about-small-collision-time

WHILE pcs is active
IF collision-time becomes smaller than limit
THEN pcs-executes-break

WHILE pcs is active
IF collision-time becomes smaller than limit
THEN pcs rewinds-seatbelt

WHILE pcs-executes-brake
IF collision-time becomes larger than limit OR vehicle stops
THEN pcs ends-pcs-brake-control-execution AND pcs releases-seatbelt

WHILE pcs-brake-control-is-executed
IF driver brakes
THEN pcs-brake-control-execution-continues

To represent the information "collision-time is smaller than limit" we introduce a variable *collision_time* as well as a parameter *collision_time_limit*. The natural language expression "collision-time becomes smaller than limit" means, that at the previous moment (at the previous time unit) it was no brake execution.

The expression "pcs-executes-break" was discussed in Section 3.10.

The expression "pcs warns-driver-about-small-collision-time" is specified by the event *small_collision_time_warning*.

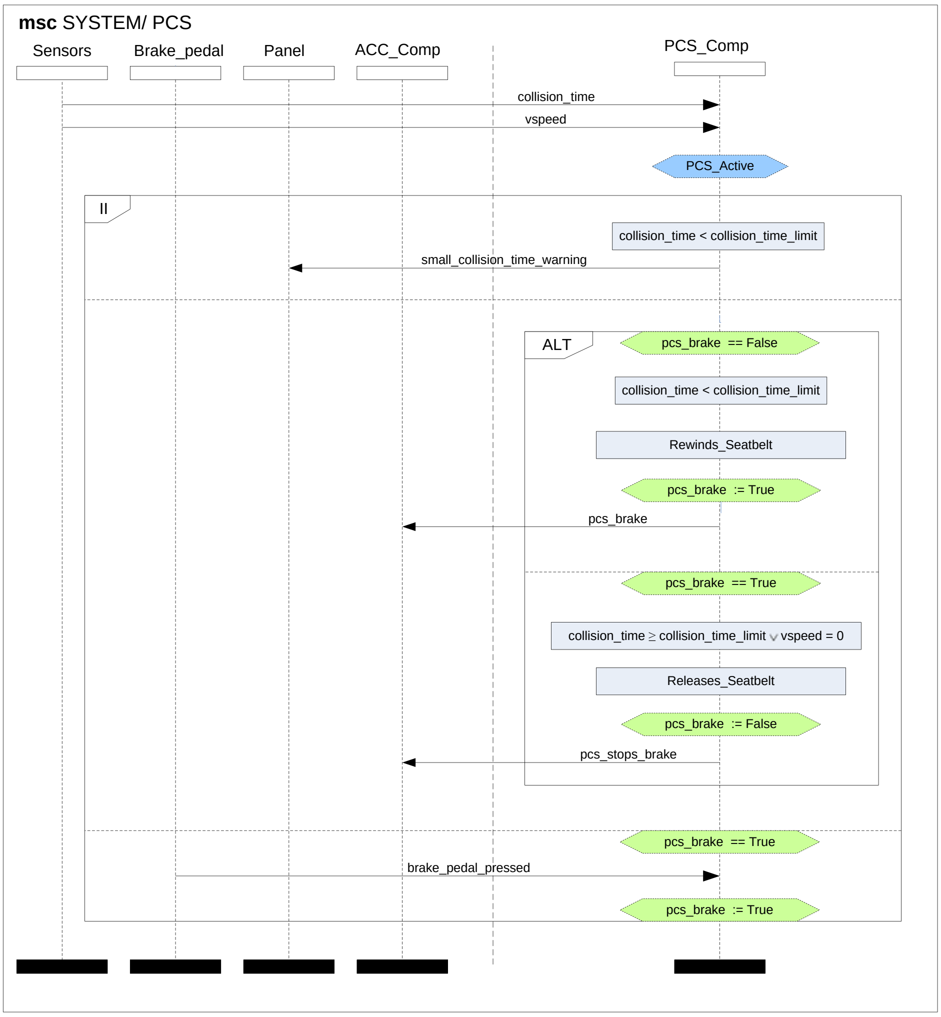
The natural language expression "'vehicle stops" is equal to the expression "vehicle speed is equal 0km/h" and we represent it by $vspeed == 0$.

Doing the MSC specification we found out that the following information is lost in semiformal specification:

- ◇ The following items are lost in the logical interface:
  - ▷ pcs-brake-control-is-executed,
  - ▷ pcs-brake-control-execution-continues,
  - ▷ pcs-executes-brake,
  - ▷ driver accelerates.

◇ The following actions must denote the same: “pcs-brake-control-is-executed”, “pcs-brake-control-execution-continues”, “pcs-executes-brake”.

These information was added to the logical interface description in the final version of the requirement specification.



4 FOCUS Specifications

4.1 Short Introduction to FOCUS

A distributed system in Focus is represented by its *components*[4]. Components that are connected by communication lines called *channels*, can interact or work independently of each other. The channels in Focus are *asynchronous communication links* without delays. They are directed, reliable, and order preserving. Via these channels components exchange information in terms of *messages* of specified types. The formal meaning of a Focus specification is a relation between the *communication histories* for the external input and output channels. The specifications can be structured into a number of formulas each characterizing a different kind of property, the most prominent classes of them are *safety* and *liveness properties*. Focus supports a variety of *specification styles* which describe system components by logical formulas or by diagrams and tables representing logical formulas. For a detailed description of Isabelle/HOL see [6] and [8].

4.1.1 Concept of Streams

The central concept in Focus are *streams*, that represent communication histories of *directed channels*. Streams in Focus are functions mapping the indexes in their domains to their messages. For any set of messages M, M^{ω} denotes the set of all streams, M^{∞} and M^{*} denote the sets of all infinite and all finite streams respectively. $M^{\underline{\omega}}$ denotes the set of all timed streams, $M^{\underline{\infty}}$ and $M^{\underline{*}}$ denote the sets of all infinite and all finite timed streams respectively. A *timed stream* is represented by a sequence of messages and *time ticks*, the messages are also listed in their order of transmission. The ticks model a discrete notion of time.

The timed domain is the most important one for representation of distributed systems with real-time requirements. Specifications of embedded systems must be *timed*, because by representing a real-time system as an untimed specification a number of properties of the system are loosed (e.g. the causality property) that are not only very important for the system, but also help us to make proofs easier. The definition in Isabelle/HOL of the Focus stream types is given below. Another ways of streams formalizations as well as the related work for the approach "Focus on Isabelle" are discussed in [7].

To simplify the specification of the real-time systems we introduce an additional Focus operator $\mathsf{ti}(s, n)$ that yields the list of messages that are in the timed stream s between the ticks $n - 1$ and n (at the nth time unit).

The predicate ts holds for a timed stream s, iff s is time-synchronous in the sense that exactly one message is transmitted in each time interval.

The Focus operator $\mathsf{msg}_n(s)$, which holds for a timed stream s, if this stream contains at every time unit at most n messages.

[4] A component in Focus means a "logical component" and not a physical one.

4.1.2 Specifications

Focus specifications can be *elementary* or *composite*. Any elementary Focus specification has the following syntax:

Name (Parameter_Declarations) ═══ Frame_Labels

in *Input_Declarations*

out *Output_Declarations*

Body

Name is the name of the specification; *Frame_Labels* lists a number of frame labels, e.g. *untimed*, *timed* or *time-synchronous*, that correspond to the stream types in the specification (see Sect. 4.1.1); *Parameter_Declarations* lists a number of parameters (optional); *Input_Declarations* and *Output_Declarations* list the declarations of input and output channels respectively. *Body* characterizes the relation between the input and output streams, and can be a number of formulas, or a table, or diagram or a combination of them. For any elementary timed parameterized specification S we define its semantics, written $[\![S]\!]$, to be the formula:

$$i_S \in I_S^{\infty} \ \wedge \ p_S \in P_S \ \wedge \ o_S \in O_S^{\infty} \ \wedge \ B_S \tag{1}$$

where i_S and o_S denote lists of input and output channel identifiers, I_S and O_S denote their corresponding types, p_S denotes the list of parameters and P_S denotes their types, B_S is a formula in predicate logic that describes the body of the specification S.

To define the semantics of a timed specification in Isabelle/HOL we introduce first tree predicates, inStream, outStream, and locStream over infinite timed streams. The predicate inStream/outStream/locStream is true, if the channel identifier corresponds to an input/output/local stream. Now we can define the semantics of an elementary timed specification with the input channels $i_1, \dots, i_n$ and the output channels $o_1, \dots, o_m$ (and with parameters $p_1, \dots, p_k$) in Isabelle/HOL in the same way as it is defined in Focus:

$$\bigwedge_{1}^{n} \mathsf{inStream}(i_j) \ \wedge \ \bigwedge_{1}^{m} \mathsf{outStream}(o_j) \ \wedge \ body \tag{2}$$

where the Isabelle/HOL predicate *body* describes here the relation (with k extra parameters) between the input and output streams and is equal modulo syntax to B_S that is conjunction of all propositions in the body of the specification S). The order of the parameters in the relation must be the following one: number of channels in the sheaf, input streams, specification parameters, output streams. For the proofs of the properties we need only the predicate *body*. Therefore, only this part will be denoted later as semantic of the specification. The conjunction of the predicates inStream/outStream/locStream will be defined in Isabelle/HOL separately[5], because this part will be used only to show that the correctness of syntactic interface.

[5] The signature of the corresponding predicate will be equal to the signature of the predicate *body*.

4.1.3 Data types

In this section we present the user defined Focus data types to model the system in a formal way.

First of all we need to define the sets of ACC and PCS states respectively. These sets must correspond to the sets of modes defined by MSCs *ACC:Modes* and *PCS:Modes*, thus, we need only to list these modes and:

$ACCState = \{ACC\_Inactive, ACC\_Active\}$
$PCSState = \{PCS\_Inactive, PCS\_Active\}$

We also need to define a data type to describe two kinds of control by ACC (according to the switch in the flow at the MSC *ACC*): follow up, control a constant speed. Let call it *ControlType*:

$ControlType = \{FollowUp, ConstantSpeed\}$

Th represent the three kinds of the distances to following up (a long, a middle and a short distance) we define the data type *DistanceType*:

$DistanceType = \{Long, Middle, Short\}$

The data type *ChangeSpeed* describes two commands/kinds of changing speed: increasing speed according to some acceleration/ speed difference value and decreasing speed according to some deceleration / speed difference value:

$ChangeSpeed = \{Increase, Decrease\}$

To describe the state of pedals and belts we will use respectively the data types *Pedal* and *Belt*:

$Pedal = \{Pressed, Released\}$
$Belt = \{Rewind, Release\}$

The data type *Event* ist needed to model simple signals that just indicate some event, the corresponding channel for a stream with the *Event*-type will be named by the concrete event we need to represent. The data type *Warn* is defined in a similar way.

$Event = \{event\}$
$Warn = \{Warning\}$

4.2 Designer Decisions

In this section we discuss the translation problems that can't be resolved schematically, because some refinement steps or designer decisions are needed.

It must be impossible to give two contradictory orders, therefore we need some constraint, which makes sure that the signals *increase_speed* and *decrease_speed* can't be send simultaneously (at the same time unit). We have two possibilities to solve it:

- Model them by two different channels and add the assumption, that the corresponding streams are disjoint.

- Model them by a single channel add the assumption, that the corresponding stream can have every time unit at most one message.

We choose the second way to model these signals.

The head (main) MSC *ACC_PCS* represents a general system behavior and is described by parallel combinations of two MSCs, *ACC_System* and *PCS_System*, each of them is again a parallel combinations – of MSCs in loop. Thus, four MSCs (*ConditionsACC*, *ACC*, *PCS* and *ConditionsPCS*) act parallel, where the main actions are described by *ACC* and *PCS*, whether *ConditionsACC* and *ConditionsPCS* describe only computation of local/state variables. The result of the computations in them depens only on the current speed, any previous values of the variables etc. are ignored. Thus, we have three possibilities:

- All these parts will be represented by a single Focus component (namely *System*, *acc_startable* and *psc_state* will be represented by local/state variables.
- These parts will be represented by two compoments: ACC (combination of *ConditionsACC* and *ACC*) and PCS (combination of *ConditionsPCS* and *PCS*), *acc_startable* and *psc_state* will be represented by local/state variables.
- Every part will be represented by a single Focus component, the component *System* will be composite one. The components *ConditionsACC* and *ConditionsPCS* will be weak causal, the components *ACC* and *PCS* will be strong causal (with delay equal to one time unit).

The first solution will give us a very unreadable specification and contradicts all the ideas of the modular representation. The both other solutions are appropriate but lead to different models. We present here both variants to show the main idea difference between them: the second variant is more appropriate for the small component and systems, the third variant is more appropriate for the large component and systems. This case study is rather small, therefore the second variant is optimal, but the third one is appropriate as well.

We can prove that the system modelled according to the third variant (see Section 4.3) is a refinement of the system modelled according to the second variant (see Section 4.4) and also a refinement of the system that can be modelled according to the first variant (because the system modelled according to the second variant is a refinement of the system that can be modelled according to the first variant).

We have two possibilities to represent a mode:

- by a local variable; this solution implies that by a readable model every change of the variable according to the corresponding input values will have influence only on the next time unit according to the strong causality property; this variant is more appropriate, if the current mode depents on the previous mode;
- by an abbreviation represented by the key words let or where ; this solution implies that every change of the corresponding input values can have influence also at the same time unit; this variant is more appropriate, if the current mode depends only on the current input values and does not depent on the previous mode;

The current ACC mode depends also from the previous one, therefore we need to model it in Focus by a local variable. The current PCS mode depends only on the input values, therefore we represent it by an abbreviation of the calculations over these values.

According to the MSC *ACC.Search* every time unit either the signal *target_vechicle_is_detected* or the signal *target_vechicle_is_NOT_detected* must be send. We represent both of them by a single time-synchronous stream *vechicleDetected* of the type $\mathbb{B}$ool, , where the value of message at every time unit t will be defined as follows: true corresponds to the signal *target_vechicle_is_detected*, false corresponds to the signal *target_vechicle_is_NOT_detected*.

The same situation is with the signals *acc_pedal_pressed* and *acc_pedal_released* (see MSC *ACC.Acceleration*) – these signals will be represented by a single time-synchronous stream *accPedal* of the type $\mathbb{B}$ool, where the value of message at every time unit t will be defined as follows: true corresponds to the signal *acc_pedal_pressed*, false corresponds to the signal *acc_pedal_released*.

Underspecified functions and predicates

We need to find out which functions and predicates are underspecified and decide whether we can let them underspecified on this phase. According to the semiformal specifications we have two underspecified functions:
$ControlDistance$ and
$AdjustVehicleSpeed2TargetSpeed$.
We do not add concrete specification for these functions here.

Parameters

In most cases we need to decide what to use in our system specification: parameterized representation of system/components or global constants.

For the presented case study we chose the first variant and specify the PCS subsystem (and the whole system as well) using one parameter that represents a collision time limit (denoted by collision_time_limit signal in MSCs): $CTLimit$ of type $\mathbb{N}$.

Local variables

We also need to collect the set of local variables that we will need in our specifications. Let represent this collection by Table 1 to have an overview which local variable comes from which MSC as well as to which Focus component this variable must belong. For completeness we go through the whole list of MSCs, presenting in the table also such of them that do not imply any new variable.

At the end we need to define the initial values of our local variables. Let choose the following definitions:

- ◇ $control = ConstantSpeed$,
- ◇ $distance = Middle$,
- ◇ $pcsBrake =$ false,
- ◇ $tspeed = 0$.

To model the system in more readable way, we add to the ACC component a number of output channels that indicate the current values of the corresponding local variables. The general inteface of the component ACC and PCS (the relations between MSC signals and the Focus streams\channels) is presentey by Table 2.

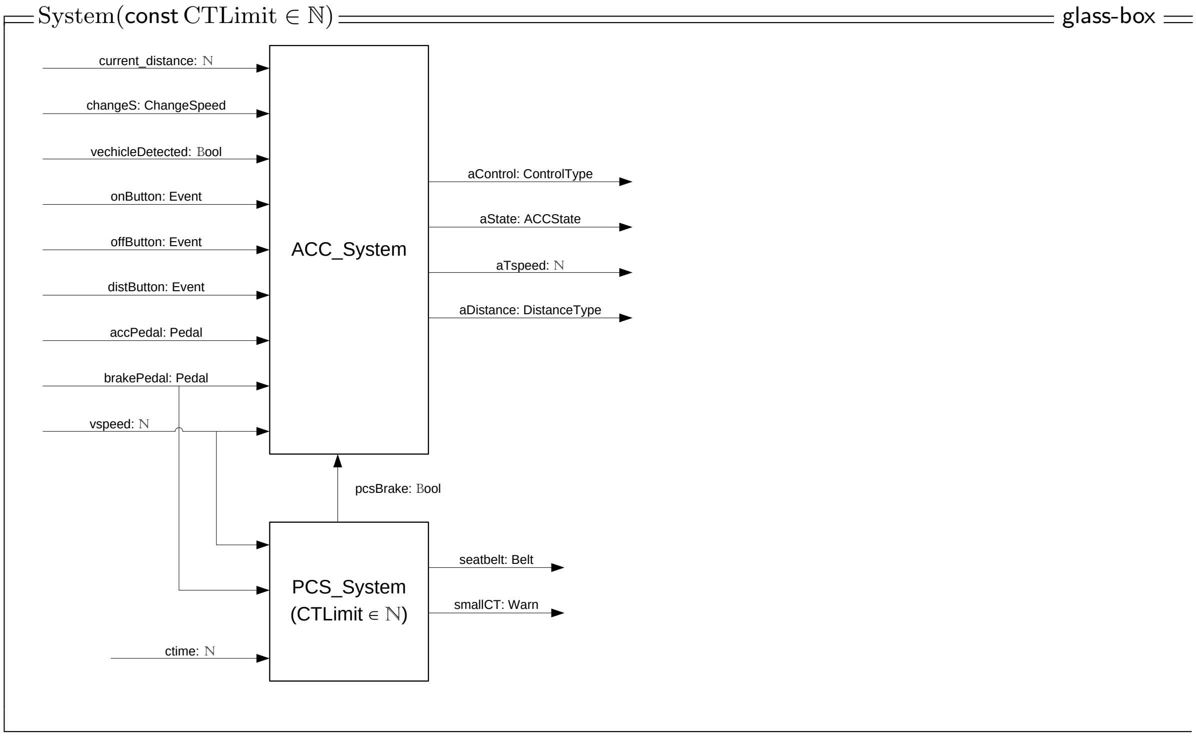
The system architecture will be represented by the following Focus specification *System*.

MSC	MSC Local Variable/Mode	FOCUS Local Variable	FOCUS component
ACC:Modes	(Mode)	$acc\_state \in ACCState$	ACC
PCS:Modes	(Mode)	–	–
ACC_PCS			
ACC_System			
PCS_System			
ConditionsACC	*acc_startable*	–	–
ACC	*control*	$control \in ControlType$	ACC
ACC.Start	*acc_startable*	–	–
ACC.Search_Acceleration			
ACC.Search	*control*	$control \in ControlType$	ACC
ACC.Acceleration	*susp*	$susp \in \mathbb{B}\text{ool}$	ACC
ACC.Follow_up	*control* *distance*	$control \in ControlType$ $distance \in DistanceType$	ACC ACC
ACC.Constant_Speed	*control* *tspeed*	$control \in ControlType$ $tspeed \in \mathbb{N}$	ACC ACC
ACC.Terminates			
ConditionsPCS			
PSC	*pcs_brake*	$pcs\_brake \in \mathbb{B}\text{ool}$	PCS

Table 1: Local Variables

MSC signal (or local variable)	FOCUS channel	From	To
curent_distance	*curent_distance*	Environment	ACC_ System
increase_speed, decrease_speed	*changeS*	Environment	ACC_ System
driver_presses_on_button	*onButton*	Environment	ACC_ System
driver_presses_off_button	*offButton*	Environment	ACC_ System
dist_button	*distButton*	Environment	ACC_ System
acc_pedal_pressed, acc_pedal_released	*accPedal*	Environment	ACC_ System
brake_pedal_pressed	*brakePedal*	Environment	ACC_ System PCS_ System
	vspeed	Environment	ACC_ System PCS_ System
collision_time	*ctime*	Environment	PCS_ System
pcs_brake	*pscBrake*	PCS_System	ACC_ System
control (local variable)	*aControl*	ACC_ System	Environment
(Mode of ACC)	*aState*	ACC_ System	Environment
tspeed (local variable)	*aTspeed*	ACC_ System	Environment
distance (local variable)	*aDistance*	ACC_ System	Environment
Rewinds_Seatbelt, Releases_Seatbelt	*seatbelt*	PCS_ System	Environment
small_collision_time_warning	*smallCT*	PCS_ System	Environment

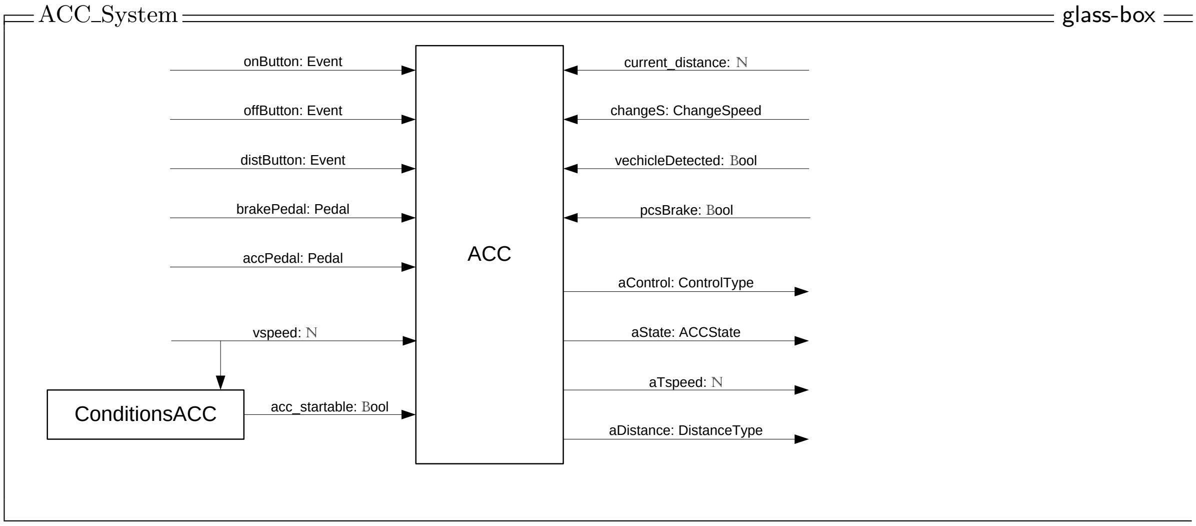
Table 2: MSC signals vs. Focus streams channels



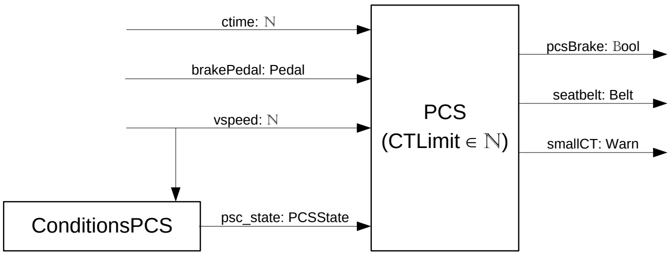
4.3 Specification of the System: 1. Model

Each of the compoments *ACC_System* and *PCS_System* is represented by a composition of two components: the correspondig Focus specifications are given below. Thus, we have here one decomposition layer more: *ACC_System* is a composition of *ConditionsACC* and *ACC* components, and *PCS_System* is a composition of *ConditionsPCS* and *PCS* components. The components *ConditionsACC*, *ACC ConditionsPCS* and *PCS* are elementary ones, where *ConditionsACC* and *ConditionsPCS* are weak-causal, and *ACC PCS* are strong causal.

In this model we have to define two more (local) channels that are presented in the 1. model by abbreviations of the computations over the corresponding input values: *acc_startable* : 𝔹ool and *pcs_state* : *PCSState*.



PCS_System(const CTLimit $\in \mathbb{N}$) — glass-box



ConditionsACC — timed

in $vspeed : \mathbb{N}$

out $acc\_startable : \mathbb{B}\mathrm{ool}$

asm ts($vspeed$)

gar

$\forall\, t \in \mathbb{N}$:

$\mathsf{ti}(acc\_startable, t) = \langle \mathsf{ft.ti}(vspeed, t) > 45) \;\wedge\; (\mathsf{ft.ti}(vspeed, t) < 110)\rangle$

ConditionsPCS — timed

in $vspeed : \mathbb{N}$

out $pcs\_state : PCSState$

asm ts($vspeed$)

gar

$\forall\, t \in \mathbb{N}$:

$\mathsf{ti}(pcs\_state, t) =$ if ft.ti($vspeed, t$) > 30 then $\langle PCSActive \rangle$ else $\langle PSCInactive \rangle$ fi

ACC — timed

$$\begin{array}{ll}
 & vspeed, curent\_distance:\ \mathbb{N};\ acc\_startable, vechicleDetected:\ \mathbb{B}\mathrm{ool};\\
\mathsf{in} & onButton, offButton, distButton : Event;\\
 & accPedal, brakePedal : Pedal;\ changeS : ChangeSpeed\\
\mathsf{out} & aState : ACCState;\ aControl : ControlType;\ aTspeed, aDistance : \mathbb{N}
\end{array}$$

$$\begin{array}{ll}
\mathsf{local} & acc\_state \in ACCState;\ susp \in \mathbb{B}\mathrm{ool};\\
 & control \in ControlType;\ distace \in DistanceType;\ tspeed \in \mathbb{N}
\end{array}$$

$$\begin{array}{ll}
\mathsf{init} & acc\_state = ACCInactive;\ susp = \mathsf{false};\\
 & control = ConstantSpeed;\ distance = Middle;\ tspeed = 0
\end{array}$$

$$\begin{array}{ll}
\mathsf{asm} & \mathsf{ts}(vspeed)\ \wedge\ \mathsf{ts}(vechicleDetected)\ \wedge\ \mathsf{ts}(curent\_distance)\ \wedge\ \mathsf{ts}(acc\_startable)\\
 & \mathsf{msg}_1(onButton)\ \wedge\ \mathsf{msg}_1(offButton)\ \wedge\ \mathsf{msg}_1(brakePedal)\ \wedge\ \mathsf{msg}_1(accPedal)\ \wedge\ \mathsf{msg}_1(changeS)
\end{array}$$

$$\begin{array}{l}
\mathsf{gar}\quad \forall\, t \in \mathbb{N}:\\
\quad \mathsf{ti}(aState, t) = \langle acc\_state\rangle\ \wedge\ \mathsf{ti}(aControl, t) = \langle control\rangle\ \wedge\ \mathsf{ti}(aTspeed, t) = \langle tspeed\rangle\ \wedge\ \mathsf{ti}(aDistance, t) = \langle distance\rangle\\
\\
\quad acc\_state = ACCInactive \rightarrow\\
\qquad \mathsf{ti}(onButton, t) \neq \langle\rangle \rightarrow\\
\qquad\quad (\mathsf{ti}(acc\_startable, t) = \langle\mathsf{true}\rangle\ \wedge\ acc\_state' = ACCActive)\\
\qquad\quad \vee\ (\mathsf{ti}(acc\_startable, t) = \langle\mathsf{false}\rangle\ \wedge\ acc\_state' = ACCInactive)\\
\qquad \wedge\\
\qquad \mathsf{ti}(onButton, t) = \langle\rangle \rightarrow acc\_state' = ACCInactive\\
\\
\quad (\mathsf{ft.ti}(vspeed, t) < 40\ \vee\ \mathsf{ti}(offButton, t) \neq \langle\rangle\ \vee\\
\quad\ \mathsf{ti}(brakePedal, t) = \langle Pressed\rangle\ \vee\ \mathsf{ti}(pcsBrake, t) = \langle\mathsf{true}\rangle)\\
\qquad \rightarrow acc\_state' = ACCInactive\\
\\
\quad (acc\_state = ACCActive \rightarrow\\
\qquad (\mathsf{ti}(accPedal, t) = \langle Pressed\rangle \rightarrow susp' = \mathsf{true})\ \vee\ (\mathsf{ti}(pcsBrake, t) = \langle\mathsf{true}\rangle \rightarrow susp' = \mathsf{true}) \vee\\
\qquad (\mathsf{ti}(accPedal, t) = \langle Released\rangle \rightarrow susp' = \mathsf{false})\ \vee\ (\mathsf{ti}(pcsBrake, t) = \langle\mathsf{false}\rangle \rightarrow susp' = \mathsf{false})\\
\\
\quad acc\_state = ACCActive \rightarrow\\
\qquad \mathsf{ft.ti}(vechicleDetected, t) = \mathsf{true} \wedge\\
\qquad\quad ((control = FollowUp \wedge control' = FollowUp) \vee\\
\qquad\quad (control = ConstantSpeed\ \wedge\ control' = FollowUp \wedge distace' = Middle))\\
\qquad \vee\\
\qquad \mathsf{ft.ti}(vechicleDetected, t) = \mathsf{false} \wedge\\
\qquad\quad ((control = ConstantSpeed \wedge control' = ConstantSpeed) \vee\\
\qquad\quad (control = FollowUp \wedge control' = ConstantSpeed \wedge tspeed' = \mathsf{ft.ti}(vspeed, t)))\\
\\
\quad acc\_state = ACCActive\ \wedge\ control = FollowUp \rightarrow\\
\qquad ControlDistance(\mathsf{ft.ti}(curent\_distance, t), susp) \wedge\\
\qquad \mathsf{ti}(distButton, t) \neq \langle\rangle \rightarrow\\
\qquad\quad (distance = Long\ \wedge\ distance' = Middle) \vee\\
\qquad\quad (distance = Middle\ \wedge\ distance' = Short) \vee\\
\qquad\quad (distance = Short\ \wedge\ distance' = Long)\\
\\
\quad control = ConstantSpeed \rightarrow\\
\qquad AdjustVehicleSpeed2TargetSpeed(\mathsf{ft.ti}(vspeed, t), tspeed, susp)\\
\qquad \wedge\\
\qquad ((\mathsf{ti}(changeS, t) = \langle Increase\rangle \rightarrow\\
\qquad\quad tspeed \leq 95\ \wedge\ tspeed' = tspeed + 5\ \ \vee\ \ tspeed > 95\ \wedge\ tspeed' = tspeed) \vee\\
\qquad\ (\mathsf{ti}(changeS, t) = \langle Decrease\rangle \rightarrow\\
\qquad\quad tspeed \geq 95\ \wedge\ tspeed' = tspeed - 5\ \ \vee\ \ tspeed < 55\ \wedge\ tspeed' = tspeed) \vee\\
\qquad\ \mathsf{ti}(changeS, t) = \langle\rangle)
\end{array}$$

```
$\mathrm{PCS}(\mathsf{const}\ \mathrm{CTLimit} \in \mathbb{N})$                                    timed
in     $vspeed, ctime : \mathbb{N};\ \ pcs\_state : PCSState;\ \ brakePedal : Pedal$
out    $smallCT : Warn;\ \ pcsBrake : \mathbb{B}\mathrm{ool};\ \ seatbelt : Belt$

local  $lbrake \in \mathbb{B}\mathrm{ool};$

init   $lbrake = \mathsf{false};$

asm    $\mathsf{ts}(ctime) \ \wedge\ \mathsf{ts}(vspeed) \ \wedge\ \mathsf{msg}_1(brakePedal);$

gar    $\forall\, t \in \mathbb{N}:$
   $\mathsf{ti}(pcs\_state, t) = \langle PCSActive \rangle \rightarrow$
      $\mathsf{ft.ti}(ctime, t) < CTLimit \rightarrow \mathsf{ti}(smallCT, t+1) = \langle Warning \rangle$
      $\wedge$
      $lbrake = \mathsf{false} \ \wedge\ \mathsf{ft.ti}(ctime, t) < CTLimit \rightarrow$
         $\mathsf{ti}(seatbelt, t+1) = \langle Rewind \rangle \ \wedge\ lbrake' = \mathsf{true} \ \wedge\ \mathsf{ti}(pcsBrake, t+1) = \langle \mathsf{true} \rangle$
      $\wedge$
      $brake = \mathsf{true} \ \wedge\ (\mathsf{ft.ti}(ctime, t) \geq CTLimit \ \vee\ \mathsf{ti}(vspeed, t) = 0) \rightarrow$
         $\mathsf{ti}(seatbelt, t+1) = \langle Release \rangle \ \wedge\ lbrake' = \mathsf{false} \ \wedge\ \mathsf{ti}(pcsBrake, t+1) = \langle \mathsf{false} \rangle$
      $\wedge$
      $lbrake = \mathsf{true} \ \wedge\ \mathsf{ti}(brakePedal, t) = \langle Pressed \rangle \rightarrow lbrake' = \mathsf{true}$
```

4.4 Specification of the System: 2. Modell

Each of the compoments ACC_System (combination of $ConditionsACC$ and ACC) and PCS_System (combination of $ConditionsPCS$ and PCS) is defined by an elementary component, there is no more decomposition layers.

```
$\mathrm{PCS}(\mathsf{const}\ \mathrm{CTLimit} \in \mathbb{N})$                                    timed
in     $vspeed, ctime : \mathbb{N};\ \ pcs\_state : PCSState;\ \ brakePedal : Pedal$
out    $smallCT : Warn;\ \ pcsBrake : \mathbb{B}\mathrm{ool};\ \ seatbelt : Belt$

local  $lbrake \in \mathbb{B}\mathrm{ool};$

init   $lbrake = \mathsf{false};$

asm    $\mathsf{ts}(ctime) \ \wedge\ \mathsf{ts}(vspeed) \ \wedge\ \mathsf{msg}_1(brakePedal);$

gar    $\forall\, t \in \mathbb{N}:$
   $pcs\_state = PCSActive \rightarrow$
      $\mathsf{ft.ti}(ctime, t) < CTLimit \rightarrow \mathsf{ti}(smallCT, t+1) = \langle Warning \rangle$
      $\wedge$
      $lbrake = \mathsf{false} \ \wedge\ \mathsf{ft.ti}(ctime, t) < CTLimit \rightarrow$
         $\mathsf{ti}(seatbelt, t+1) = \langle Rewind \rangle \ \wedge\ lbrake' = \mathsf{true} \ \wedge\ \mathsf{ti}(pcsBrake, t+1) = \langle \mathsf{true} \rangle$
      $\wedge$
      $brake = \mathsf{true} \ \wedge\ (\mathsf{ft.ti}(ctime, t) \geq CTLimit \ \vee\ \mathsf{ti}(vspeed, t) = 0) \rightarrow$
         $\mathsf{ti}(seatbelt, t+1) = \langle Release \rangle \ \wedge\ lbrake' = \mathsf{false} \ \wedge\ \mathsf{ti}(pcsBrake, t+1) = \langle \mathsf{false} \rangle$
      $\wedge$
      $lbrake = \mathsf{true} \ \wedge\ \mathsf{ti}(brakePedal, t) = \langle Pressed \rangle \rightarrow lbrake' = \mathsf{true}$
where $pcs\_state = \mathsf{if}\ \ \mathsf{ft.ti}(vspeed, t) > 30\ \mathsf{then}\ \ PCSActive\ \mathsf{else}\ \ PSCInactive\ \mathsf{fi}$
```

```
ACC                                                                  timed

in     $vspeed, curent\_distance : \mathbb{N};\ vechicleDetected : \mathbb{B}\mathrm{ool};\ onButton, offButton, distButton : Event;$
       $accPedal, brakePedal : Pedal;\ changeS : ChangeSpeed$
out    $aState : ACCState;\ aControl : ControlType;\ aTspeed, aDistance : \mathbb{N}$

local  $acc\_state \in ACCState;\ susp \in \mathbb{B}\mathrm{ool};$
        $control \in ControlType;\ distace \in DistanceType;\ tspeed \in \mathbb{N}$

init   $acc\_state = ACCInactive;\ susp = \mathsf{false};$
        $control = ConstantSpeed;\ distance = Middle;\ tspeed = 0$

asm    $\mathsf{ts}(vspeed)\ \wedge\ \mathsf{ts}(vechicleDetected)\ \wedge\ \mathsf{ts}(curent\_distance)$
       $\mathsf{msg}_1(onButton)\ \wedge\ \mathsf{msg}_1(offButton)\ \wedge\ \mathsf{msg}_1(brakePedal)\ \wedge\ \mathsf{msg}_1(accPedal)\ \wedge\ \mathsf{msg}_1(changeS)$

gar    $\forall\, t \in \mathbb{N}:$
  $\mathsf{ti}(aState, t) = \langle acc\_state\rangle\ \wedge\ \mathsf{ti}(aControl, t) = \langle control\rangle\ \wedge\ \mathsf{ti}(aTspeed, t) = \langle tspeed\rangle\ \wedge\ \mathsf{ti}(aDistance, t) = \langle distance\rangle$

    $acc\_state = ACCInactive \rightarrow$
      $\mathsf{ti}(onButton, t) \neq \langle\rangle \rightarrow$
        $(acc\_startable\ \wedge\ acc\_state' = ACCActive)\ \vee\ (\neg acc\_startable\ \wedge\ acc\_state' = ACCInactive)$
      $\wedge$
      $\mathsf{ti}(onButton, t) = \langle\rangle \rightarrow acc\_state' = ACCInactive$

    $(\mathsf{ft.ti}(vspeed, t) < 40\ \vee\ \mathsf{ti}(offButton, t) \neq \langle\rangle\ \vee$
     $\mathsf{ti}(brakePedal, t) = \langle Pressed\rangle\ \vee\ \mathsf{ti}(pcsBrake, t) = \langle\mathsf{true}\rangle)$
      $\rightarrow acc\_state' = ACCInactive$

    $(acc\_state = ACCActive \rightarrow$
      $(\mathsf{ti}(accPedal, t) = \langle Pressed\rangle \rightarrow susp' = \mathsf{true})\ \vee\ (\mathsf{ti}(pcsBrake, t) = \langle\mathsf{true}\rangle \rightarrow susp' = \mathsf{true})\vee$
      $(\mathsf{ti}(accPedal, t) = \langle Released\rangle \rightarrow susp' = \mathsf{false})\ \vee\ (\mathsf{ti}(pcsBrake, t) = \langle\mathsf{false}\rangle \rightarrow susp' = \mathsf{false})$

    $acc\_state = ACCActive \rightarrow$
      $\mathsf{ft.ti}(vechicleDetected, t) = \mathsf{true}\ \wedge$
        $((control = FollowUp \wedge control' = FollowUp)\ \vee$
        $(control = ConstantSpeed\ \wedge\ control' = FollowUp \wedge distace' = Middle))$
      $\vee$
      $\mathsf{ft.ti}(vechicleDetected, t) = \mathsf{false}\ \wedge$
        $((control = ConstantSpeed \wedge control' = ConstantSpeed)\ \vee$
        $(control = FollowUp \wedge control' = ConstantSpeed \wedge tspeed' = \mathsf{ft.ti}(vspeed, t)))$

    $acc\_state = ACCActive\ \wedge\ control = FollowUp \rightarrow$
      $ControlDistance(\mathsf{ft.ti}(curent\_distance, t), susp)\wedge$
      $\mathsf{ti}(distButton, t) \neq \langle\rangle \rightarrow$
        $(distance = Long\ \wedge\ distance' = Middle)\ \vee$
        $(distance = Middle\ \wedge\ distance' = Short)\ \vee$
        $(distance = Short\ \wedge\ distance' = Long)$

    $control = ConstantSpeed \rightarrow$
      $AdjustVehicleSpeed2TargetSpeed(\mathsf{ft.ti}(vspeed, t), tspeed, susp)$
      $\wedge$
      $((\mathsf{ti}(changeS, t) = \langle Increase\rangle \rightarrow$
        $tspeed \leq 95\ \wedge\ tspeed' = tspeed + 5\ \ \vee\ \ tspeed > 95\ \wedge\ tspeed' = tspeed)\ \vee$
       $(\mathsf{ti}(changeS, t) = \langle Decrease\rangle \rightarrow$
        $tspeed \geq 95\ \wedge\ tspeed' = tspeed - 5\ \ \vee\ \ tspeed < 55\ \wedge\ tspeed' = tspeed)\ \vee$
       $\mathsf{ti}(changeS, t) = \langle\rangle)$
where $acc\_startable = (\mathsf{ft.ti}(vspeed, t) > 45)\ \wedge\ (\mathsf{ft.ti}(vspeed, t) < 110)$
```

5 Summary

This paper presents an approach for translating semiformal specification in formal ones. For a semiformal representation two approaches were taken: an approach from A. Fleischmann [3] and a message sequence charts representation. A semiformal specification represented by one of these approaches can be translated to a formal specification in FOCUS, a a framework for formal specifications and development of interactive systems.

This approach was applied on the case study during the project DENTUM between Denso Deutschland GmbH and the chair for Software & Systems Engineering at Technische Universität München (see [2]). The goal of this project was to define a methodology for the model-based development of automotive systems. This methodology was evaluated by developing an Adaptive Cruise Control (ACC) system with Pre-Crash Safety (PCS) functionality.

References

[1] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer-Verlag, 2001.

[2] M. Feilkas, A. Fleischmann, F. Hölzl, C. Pfaller, K. Scheidemann, M. Spichkova, and D. Trachtenherz. A Top-Down Methodology for the Development of Automotive Software. Technical report, Technische Universität München, 2009.

[3] Andreas Fleischmann. *Model-based formalization of requirements of embedded automotive systems*. PhD thesis, Technische Universität München, 2008.

[4] ITU-TS. Recommendation Z.120: Message Sequence Chart (MSC). Geneva, 1996.

[5] Ingolf Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[6] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[7] M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, Technische Universität München, 2007.

[8] M. Wenzel. *The Isabelle/Isar Reference Manual*. Technische Universität München, 2004.

A Meaning of the used variables and streams: Summary table

Semiformal Specification	MSC	Focus
vehicle speed	vspeed: $\mathbb{N}$	$vspeed : \mathbb{N}^{\underline{\infty}}$
target speed	tspeed: $\mathbb{N}$	$tspeed : \mathbb{N}^{\underline{\infty}}$
acc is startable	acc_startable: $\mathbb{B}$ool	$acc\_startable : \mathbb{B}\mathrm{ool}^{\underline{\infty}}$
driver-presses-on-button	driver_presses_on_button : Event	$onButton : Event^{\underline{\infty}}$
driver-does-not-press-on-button	driver_does_not_press_on_button: Event	$onButton : Event^{\underline{\infty}}$
driver-presses-off-button	driver_presses_off_button: Event	$offButton : Event^{\underline{\infty}}$
driver operates brake pedal	brake_pedal_pressed : Event	$brakePedal : Pedal^{\underline{\infty}}$
pcs-executes-brake	pcs_brake: Event	$pcs\_brake : \mathbb{B}\mathrm{ool}^{\underline{\infty}}$
driver-presses-acceleration-pedal	acc_pedal_pressed : Event	$accPedal : Pedal^{\underline{\infty}}$
driver-releases-acceleration-pedal	acc_pedal_released : Event	$accPedal : Pedal^{\underline{\infty}}$
MUST NOT brake, CAN brake	susp: $\mathbb{B}$ool	$susp : \mathbb{B}\mathrm{ool}$
target vehicle is detected	target_vehicle_is_detected : Event	$vehicleDetected : \mathbb{B}\mathrm{ool}^{\underline{\infty}}$
no target vehicle is detected	no_target_vehicle_is_detected :Event	$vehicleDetected : \mathbb{B}\mathrm{ool}^{\underline{\infty}}$
acc is in follow-up-control	control: ControlType	$control : ControlType$
acc is in constant-speed-control	control: ControlType	$control : ControlType$
following distance to target vehicle	current_distance: $\mathbb{N}$	$current\_distance : \mathbb{N}^{\underline{\infty}}$
acc sets following distance to “middle”	distance: DistanceType	$distance : DistanceType$
acc sets following distance to “short”	distance: DistanceType	$distance : DistanceType$
acc sets following distance to “long”	distance: DistanceType	$distance : DistanceType$
driver-increases-target-speed	increase_speed: Event	$changeS : changeSpeed^{\underline{\infty}}$
driver-decreases-target-speed	increase_speed: Event	$changeS : changeSpeed^{\underline{\infty}}$