IST Project AMETIST

Real-time Service Allocation for Car Periphery Supervision

Preliminary Description (Deliverable No. 3.1.3)

Version	1.0
Date	17 Sep. 2002
Status	draft
Author(s)	Kowalewski, Rittel
Address	Robert Bosch GmbH Research and Development – FV/SLD PO Box 94 03 50 60461 Frankfurt am Main Germany
Phone	+49 (0)69 7909-530

File: AMETIST_CPSPrelimDescription_1_0.doc
Template: Bericht_RBSK_1_0.dot
Printed: 17.09.02

Change History

Version	Date	Status	Author	Changes
0.1	11.09.02	in preparation	Kowalewski	Document created. Sec. 2 is in large parts a translation of *M. Rittel: CPS-Übersicht, Version 1.1, 20 June 2002*.
1.0	17.09.02	draft	Kowalewski	First version for AMETIST partners. Comments by Dr. Hötzel (16.09.02) and M. Auerswald incorporated.

Table of Contents

Change History 2

Table of Contents 3

1. Introduction 4

2. Description of the CPS system 4

2.1 Context 4

2.2 Interfaces of the CPS system 5

2.2.1 Sensor interface 5

2.2.2 Collision object interface (COI) 5

2.2.3 Belt tensioner interface (BTI) 6

2.2.4 HMI interface 6

2.3 CPS system 6

2.3.1 Interface between sensor and ECU 7

2.3.2 Sensor 7

2.3.3 ECU 8

3. Timing 14

4. Problems 14

5. References 15

6. Abbreviations 15

7. Appendix 16

1. Introduction

This document provides a description of the case example "Real-time Service Allocation for Car Periphery Supervision". The term *Car Periphery Supervision* (*CPS*) subsumes the functionality and technology for obtaining information about the environment of a car. CPS is the basis for many driver assistance services, e.g., parking assistance, pre-crash detection, blind spot supervision, lane change assistant etc., most of which are still under development. There are different sensor technologies available for CPS realizations, e.g., ultrasonic, radar, lidar, infrared and video. In this example we concentrate on Short Range Radar (SRR) technology. Refer to [1] for an introduction to this technology and its use for CPS.

In the following, we consider a concrete basic architecture and use the term CPS to name the corresponding system. So, when we speak about the CPS system, we do not mean a general system for car periphery supervision but the particular example described in the following section. The description is rather abstract. It builds on a model developed by Michael Rittel at Bosch for use in a different project.

The CPS system considered here is reduced to one *sensor group*. A sensor group could be the set of front sensors and the corresponding controllers. There could be more than one sensor group in a car, e.g. for side or back supervision but this is not of interest for this example. Also, communication between groups is not considered.

This document is organized as follows. As mentioned above, the next section presents a component model of the CPS system under consideration. In Sec. 3 we briefly discuss the requirements for the scheduling of the CPS system. In Sec. 4 we provide some problem statements which can be taken as a starting point for the usage of the case example.

2. Description of the CPS system

In this section we present a rough model of the CPS system. It consists of components, their responsibilities, and the information exchanged between them. The model is abstract in the following sense: In terms of technology, the components could be systems made of hardware and software as well as pure software components. The information exchange could be realized physically via buses or by messages, shared data or alike in the software. Non of this is specified in the model.

The presentation of the model is hierarchical and top-down: We start with the context and then zoom further into the CPS system. For the graphical illustration of the system structure Michael Rittel chose a SDL-like fashion in which the information exchange takes place via signals sent from one component to another. The components (or "systems" or "subsystems") are symbolized by boxes, the signal directions by arrows and the signal content by names in brackets.

2.1 Context

Figure 1 presents the context of the CPS system. CPS communicates with the airbag system, the belt tensioner, and a human machine interface (HMI). The car velocity v_{self} is available to CPS via the CAN bus. Furthermore, there may be objects in the environment of the car which shall be detected by CPS.

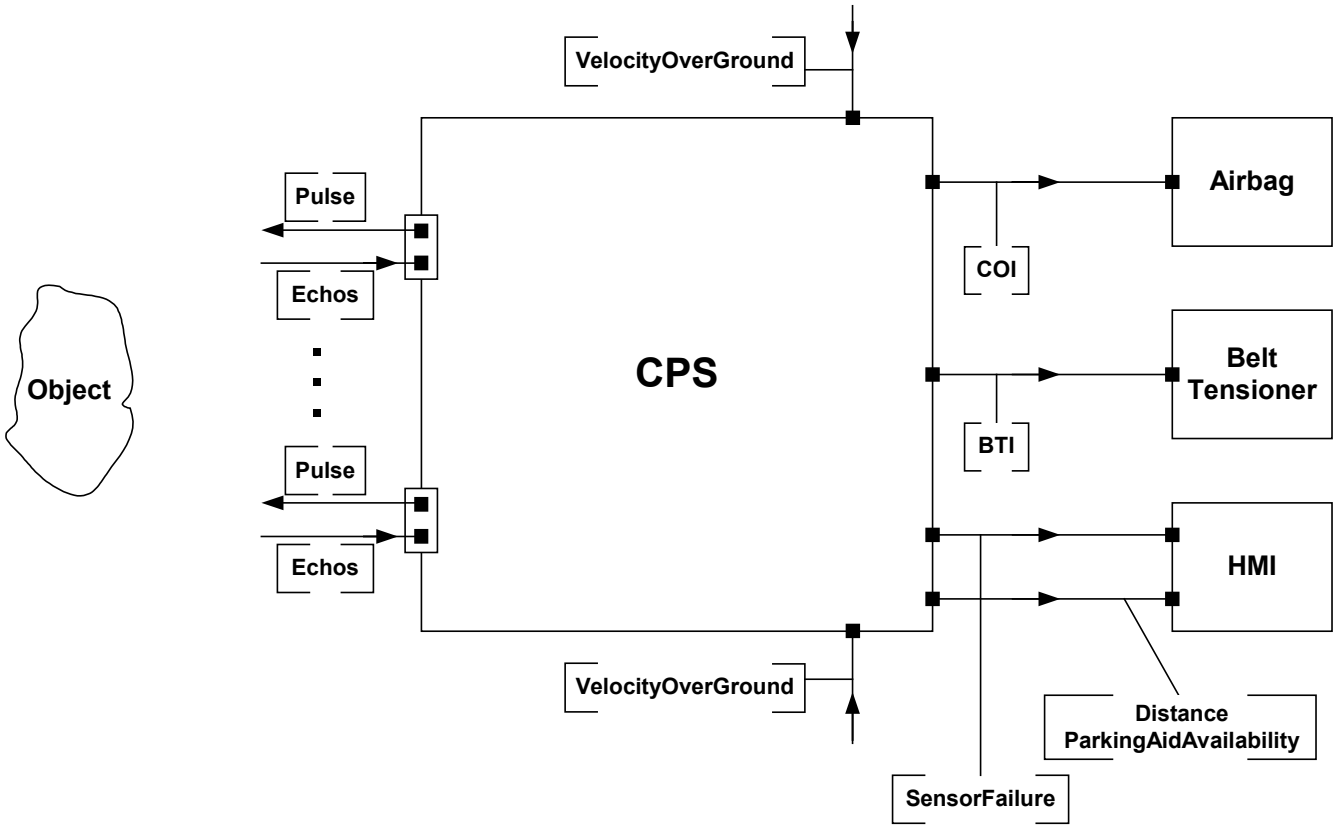

Figure 1: Context of the CPS system

2.2 Interfaces of the CPS system

2.2.1 Sensor interface

The CPS system is sending radar pulses out into the environment of the car and receives and processes the echoes which are reflected from objects in the environment. The range of the sensors under consideration is 0 to 7 m.

2.2.2 Collision object interface (COI)

CPS sends the following data to the airbag ECU:

- t_{ti} : Time to impact
- v_{crash} : Expected relative crash velocity
- Δy : Offset of place of impact from center
- α : Crash angle with respect to x-axis.

See Figure 8 for an illustration of the basic car geometry assumptions. The information must be delivered at least 10 ms before a predicted collision.

2.2.3 Belt tensioner interface (BTI)

A further task of CPS is to activate the belt tensioner via the BTI interface in case of a predicted collision. The activation must take place at least 110 ms before the predicted time of collision. After the collision or, if the predicted collision will not take place, respectively, CPS will deactivate the belt tensioner.

2.2.4 HMI interface

The following information is indicated to the user by CPS via the HMI interface:

- Failure of a sensor,
- Distance to next object, if parking aid is activated,
- Unavailability of the parking aid functionality.

2.3 CPS system

On the next level of detail, the CPS system can be refined into one ECU component and between 1 to 6 sensors, depending on the chosen configuration (see Figure 2).

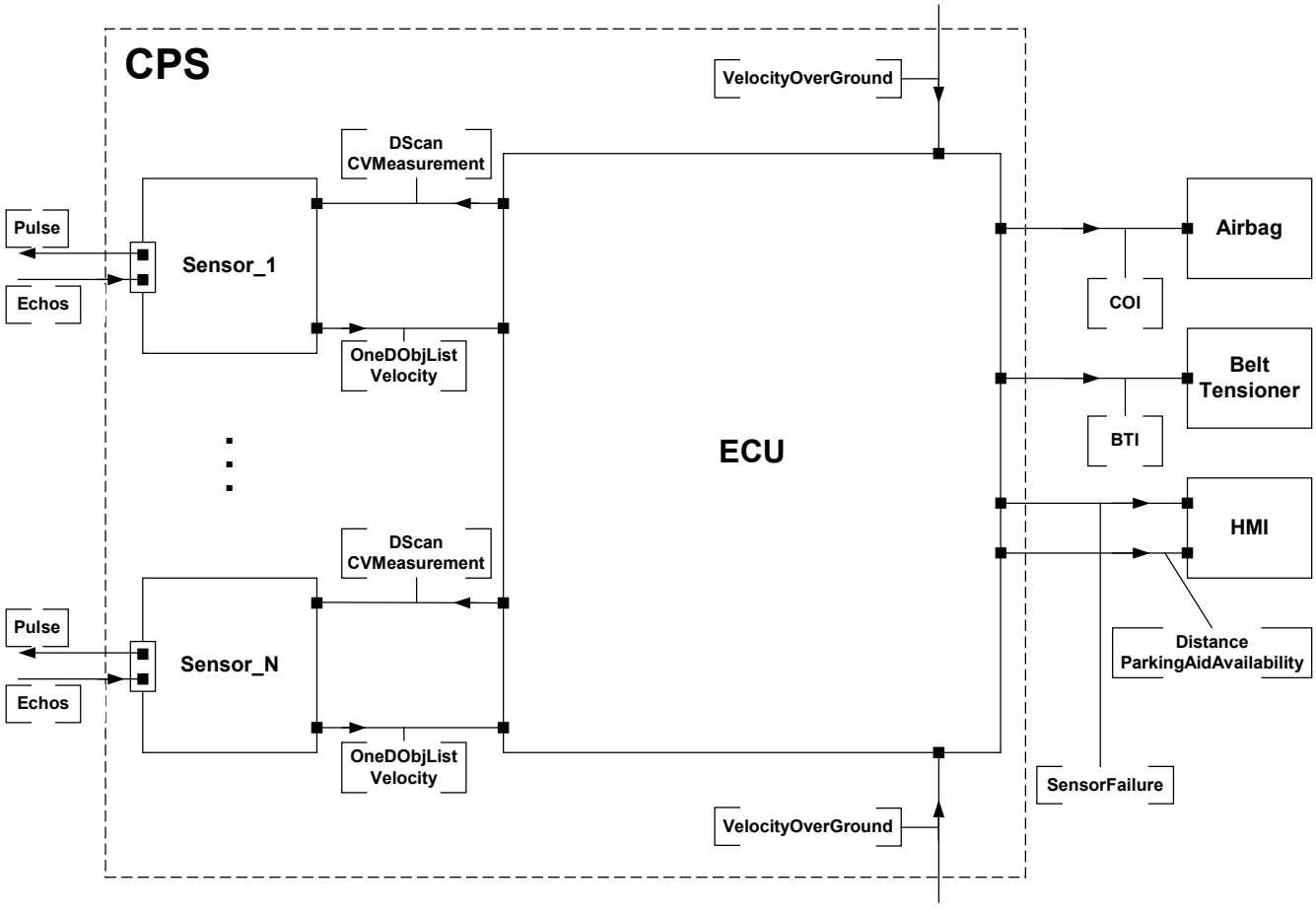

Figure 2: Structure of the CPS system

2.3.1 Interface between sensor and ECU

Each measurement and the corresponding data processing on the sensor is explicitly triggered by an order from the ECU. The order specifies whether a *DistanceScan* or a *CVMeasurement* shall be performed (see next section). The sensor generates the corresponding response which is a one-dimensional object list in case of a *DistanceScan* and a sequence of radial velocity measurements in case of a *CVMeasurement* (see Sections 2.3.2.2 and 2.3.2.3).

2.3.2 Sensor

The sensor component includes not only the equipment for sending and receiving radar signals but also a processor for basic data processing and control. The sensors have two processing modes. In the mode *DistanceScan*, the complete range of supervision will be scanned for objects. The range of the distance scan can be either from 0 to 7 m (normal case) or from 0 to 2m (after a *CVMeasurement*, see Sec. 2.3.3.1). In the mode *CVMeasurement*, a smaller range (from 1.41 to 0.69m) will be supervised by checking for closing objects at a number of range gates (see Section *CVModeProcessing* for a more detailed description).

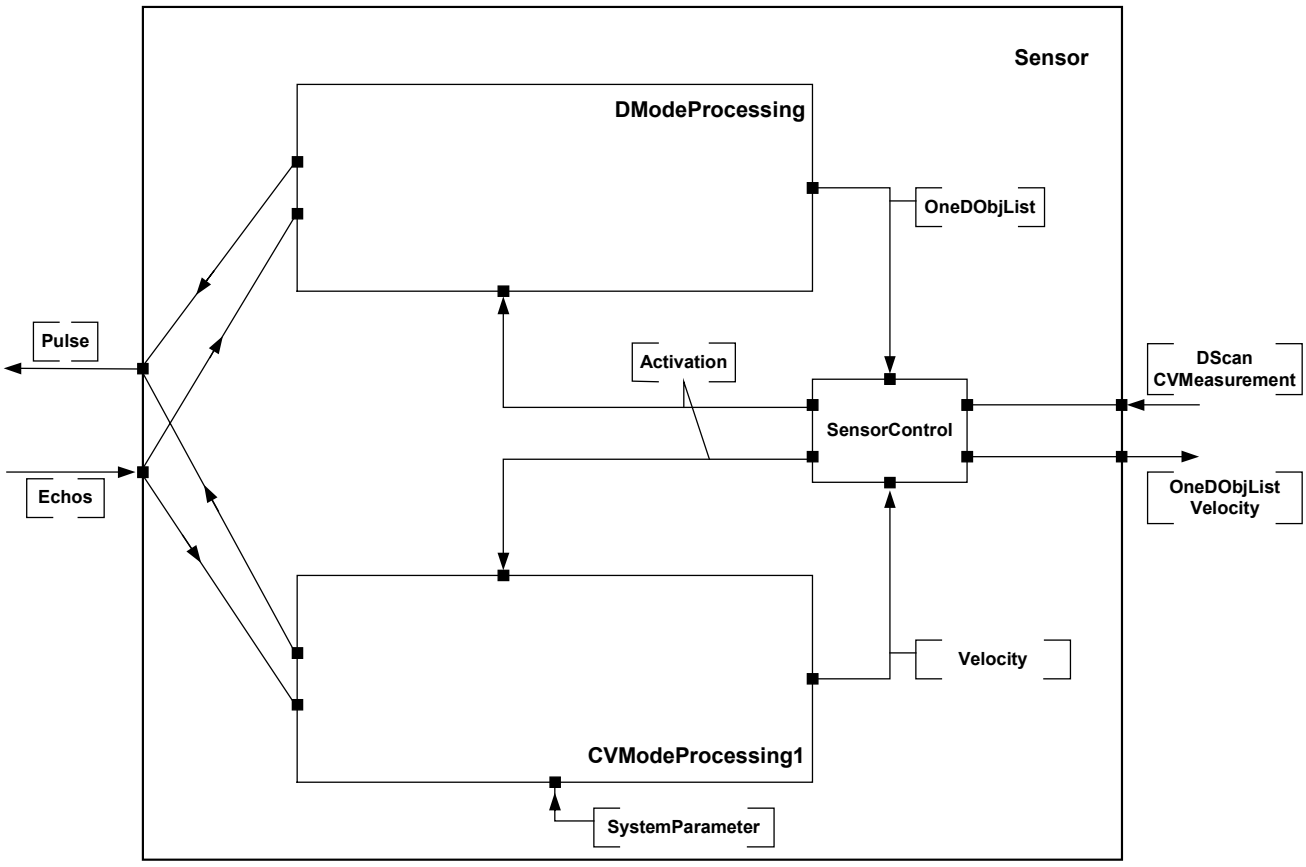

Figure 3: Structure of a sensor

2.3.2.1 SensorControl

The component *SensorControl* receives the measurement order from the ECU and activates the appropriate mode, that is *DModeProcessing* in case of a *DScan* command, and *CVModeProcessing* in case of a *CVMeasurement* command.

2.3.2.2 DModeProcessing

After executing the command *DScan*, the component *DModeProcessing* delivers a one-dimensional object list. A one-dimensional object list consists of maximally 8 objects each of which is characterized by the parameters

- r : radial distance between object and sensor,
- v_r : radial velocity,
- $[Q_r]$: quality of the radial distance measurement,
- $[Q_v]$: quality of the radial velocity measurement.

This list will be sent back by the sensor to the ECU as the response to a *DScan* command. The execution time of *DModeProcessing* is appr. $T_{\text{DModeProcessing}} \approx 3 .. 4.5$ ms.

2.3.2.3 CVModeProcessing

After receiving a *CVMeasurement* command, *CVModeProcessing* will not deliver one single result (like in the *DScan* case) but start to deliver a time series of maximally 10 radial velocity (v_r) values. The last value in this series is 0, no matter how long the sequence will be. The first value is the radial velocity of the next object passing the first *range gate* RG1 =1.41m. The second value is measured at the second range gate RG2 = 1.32m = 1.41m – 0.09m, and so on. Each further value will be measured at the next range gate which is located at the distance reduced by 0.09 m. In the whole, there are nine range gates (RG1 =1.41m, RG2 =1.32m, RG3 =1.23m, RG4 =1.14m, RG5 =1.05m, RG6 =0.96m, RG7 =0.87m, RG8 =0.78m, RG9 =0.69m).

The length of the sequence depends on whether the object of interest will pass all the range gates or not. If the object "disappears" (for example because it changed its trajectory so that it left the visibility range of the sensors), the value 0 is sent as the last element of the measurement sequence. This can happen after all of the range gates, this means the length of the sequence can vary between minimally one 0 value and maximally nine "real" values and one 0 value. Of course, the time of the velocity measurements at the range gates depends on the speed of the object and cannot be predicted exactly. For this reason, there is a timer $T_{\text{CVRGMon}} = 25$ ms which is started at each range gate. If it elapses before the collision object reaches the next range gate, the object is regarded as having disappeared and a value 0 is sent as described above.

The knowledge that there are nine range gates and their position is stored in the sensor and in the EEPROM of the ECU.

2.3.3 ECU

This component comprises the system subcomponents which are realized on the ECU hardware (not the hardware itself). They are shown by Figure 4 and Figure 5. With one exception (see Sec. 2.3.3.2), these subcomponents are specific for the sensor group.

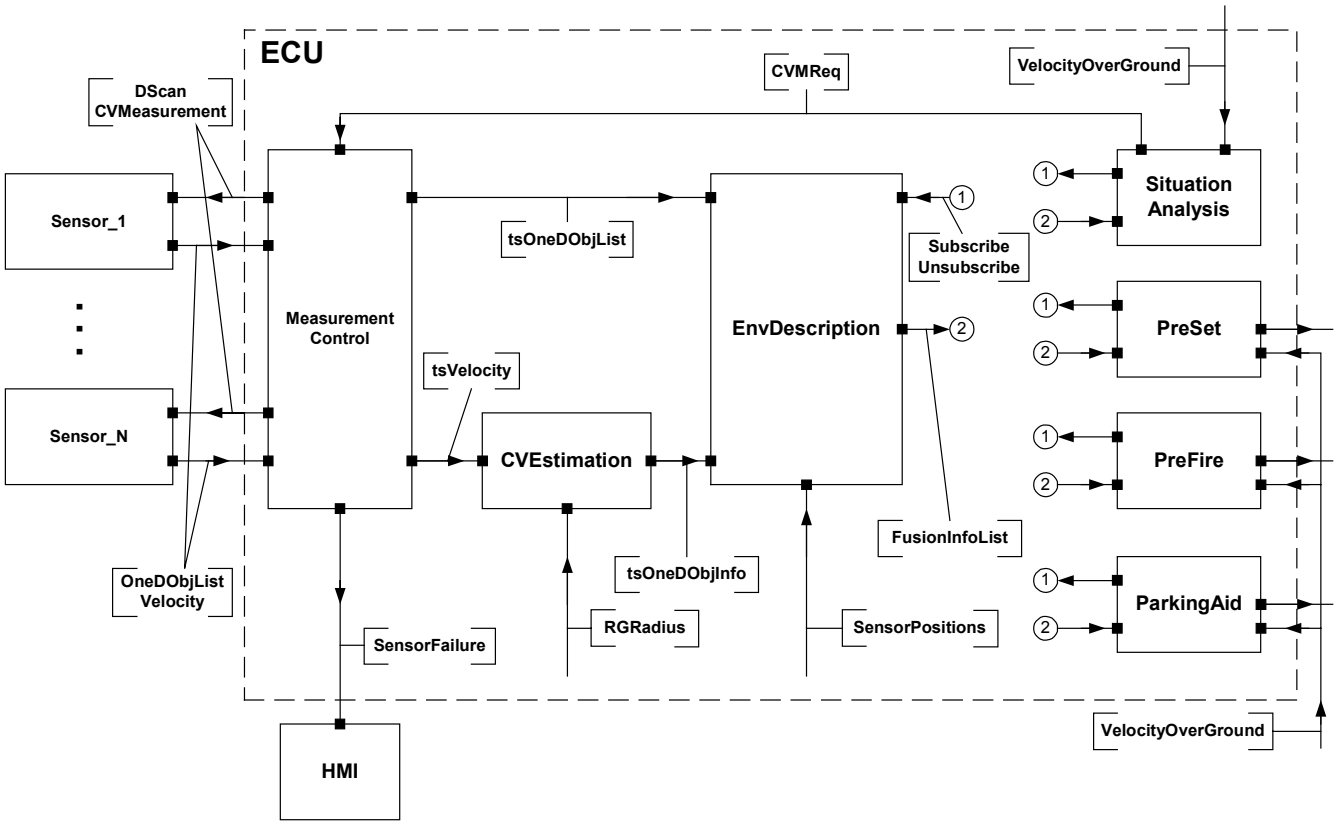

Figure 4: Structure of the ECU software, part 1

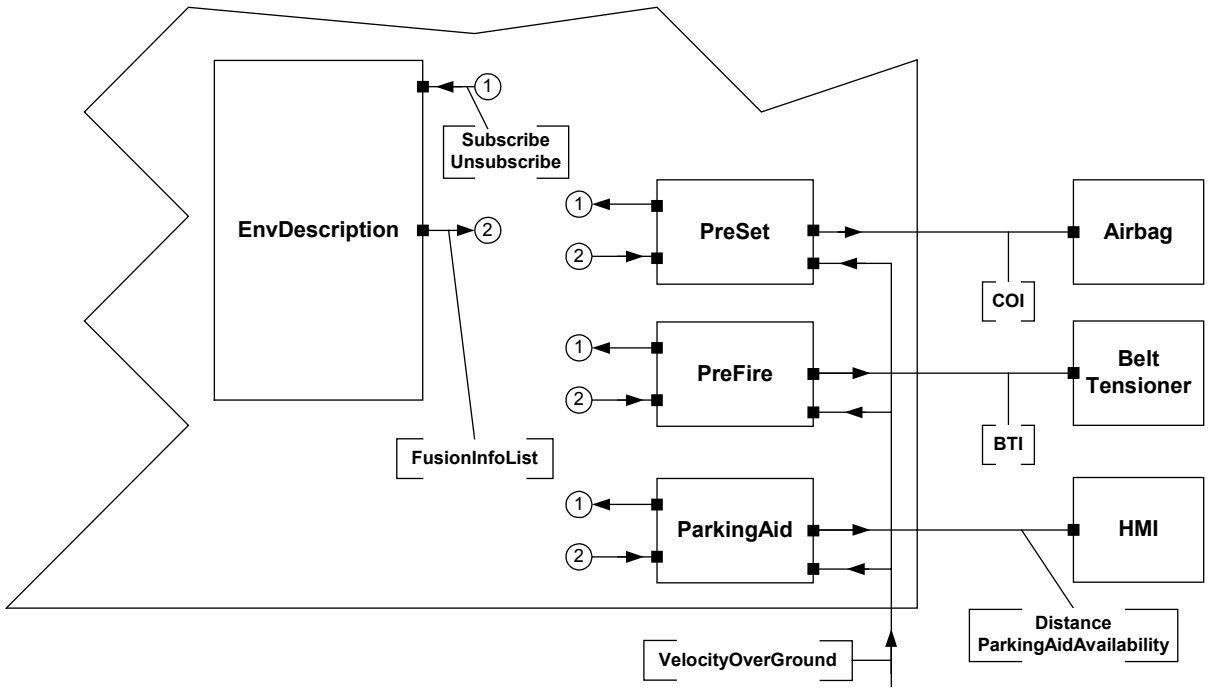

Figure 5: Structure of the ECU software, part 2

2.3.3.1 MeasurementControl

The component *MeasurementControl* sends the commands *DScan* or *CVMeasurement* to all sensors of the sensor group and receives the resulting sensor responds. It assigns the sensor number and a time stamp to each measurement and passes it on to other components for further processing.

MeasurementControl takes care that all sensors of one sensor group are in the same measurement mode. In the normal case, *MeasurementControl* is operating the sensors in *DistanceScan* mode by sending out *DScan* commands every 15 ms to all sensors. (Each sensor is connected to the central control unit by an exclusive direct connection.). The *DScan* command has a parameter specifying the range of supervision which is 7 m in the normal case.

A switch to the mode *CVMeasurement* will be triggered by the component *SituationAnalysis*. In response, the component *MeasurementControl* will send a command *CVMeasurement* to all sensors as soon as the current *DistanceScan* is finished. After the *CVMeasurement* is finished, *MeasurementControl* will send out *one DScan* command with the supervision range parameter set to 2 m. This shall make sure that objects closely following the originally tracked object will be detected quickly.

The responses on the *DScan* commands (i.e., the one-dimensional object list) get assigned the sensor number and a time stamp, and will be passed on to the component *EnvDescription*. The responses on the *CVMeasurement* commands (i.e., the radial velocities at each range gate) also get assigned the sensor number and a time stamp, and will be passed on to the component *CVEstimation*.

There are two watchdog timers for the *DScan* and the *CVMeasurement* command. If a sensor does not send a respond to a *DScan* command before the timer T_{DScan} is elapsed, or if it does not send a response to a *CVMeasurement* command before the timer T_{CVM} is elapsed, a corresponding sensor failure will be indicated to the HMI.

2.3.3.2 CVEstimation

Although the component *CVEstimation* is specific for each sensor and not for the sensor group, it will be executed on the ECU for reasons of time stamp accuracy and performance. *CVEstimation* processes the radial velocities (*tsVelocity*) and produces a vector *tsOneDObjInfo* consisting of the elements

- S_{No} : Sensor number
- *TS* : Time stamp
- |Δy| : Offset of place of impact from center (absolute value)
- |y| : Absolute value of coordinate on *y*-axis (see Sec. 7)

To perform its task, *CVEstimation* needs to know the radii at the nine range gates. *tsOneDObjInfo* will be send to the component *EnvDescription*. *CVEstimation* will produce an output for each input coming from *MeasurementControl*.

2.3.3.3 EnvDescription

The component *EnvDescription* performs a fusion of the one-dimensional and sensor-specific information generated by the *DistanceScans* or the *CVMeasurements*. The result of this fusion is a higher value description of the car environment which abstracts from the concrete sensors and which can be used by the component *SituationAnalysis* and the applications *PreSet*, *PreFire* and *ParkingAid*. To get this information, *SituationAnalysis* and the applications have to register to *EnvDescription* via a a publisher-subscriber mechanism.

EnvDescription consists of two subcomponents, *DFusion* and *VFusion* (see Figure 6). *DFusion* fuses the information from the *DistanceScans*, *VFusion* from the *CVMeasurements*. Both components require the information about the sensor positions in *x*, *y* coordinates.

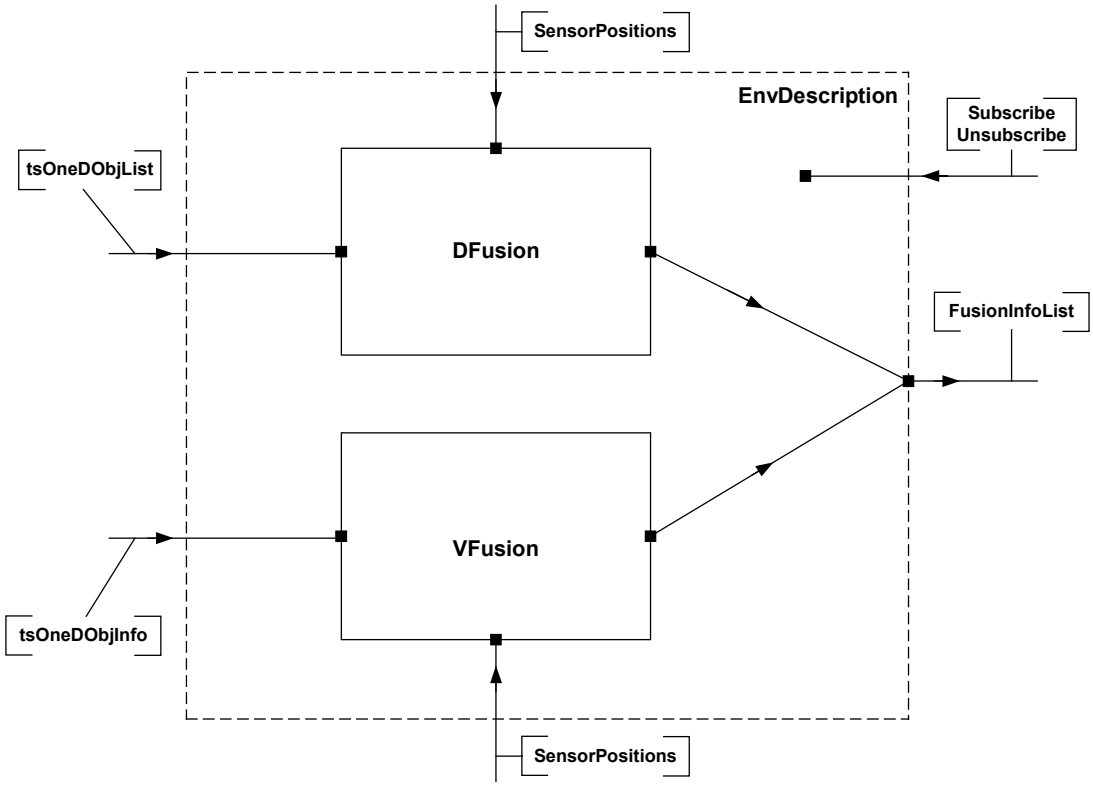

Figure 6: Structure of the component EnvDescription

2.3.3.3.1 DFusion

The component *DFusion* consists of the subcomponents *Triangulation* and *DistProcessing*. The decision which of these two subcomponents will process the incoming data depends only on the type of data. There is no external control mechanisms, and the introduction of two subcomponents merely serves to build a better model of the fusion algorithm.

2.3.3.3.1.1 Triangulation

In the component *Triangulation*, one-dimensional object lists will be fused to two-dimensional object lists. An object in two-dimensional object list is described by five vector elements:

- t_m : Time of measurement (often identical to *TS*, but may be result of extrapolation)
- x : x coordinate
- y : y coordinate
- v_x : velocity in x direction
- v_y : velocity in y direction.

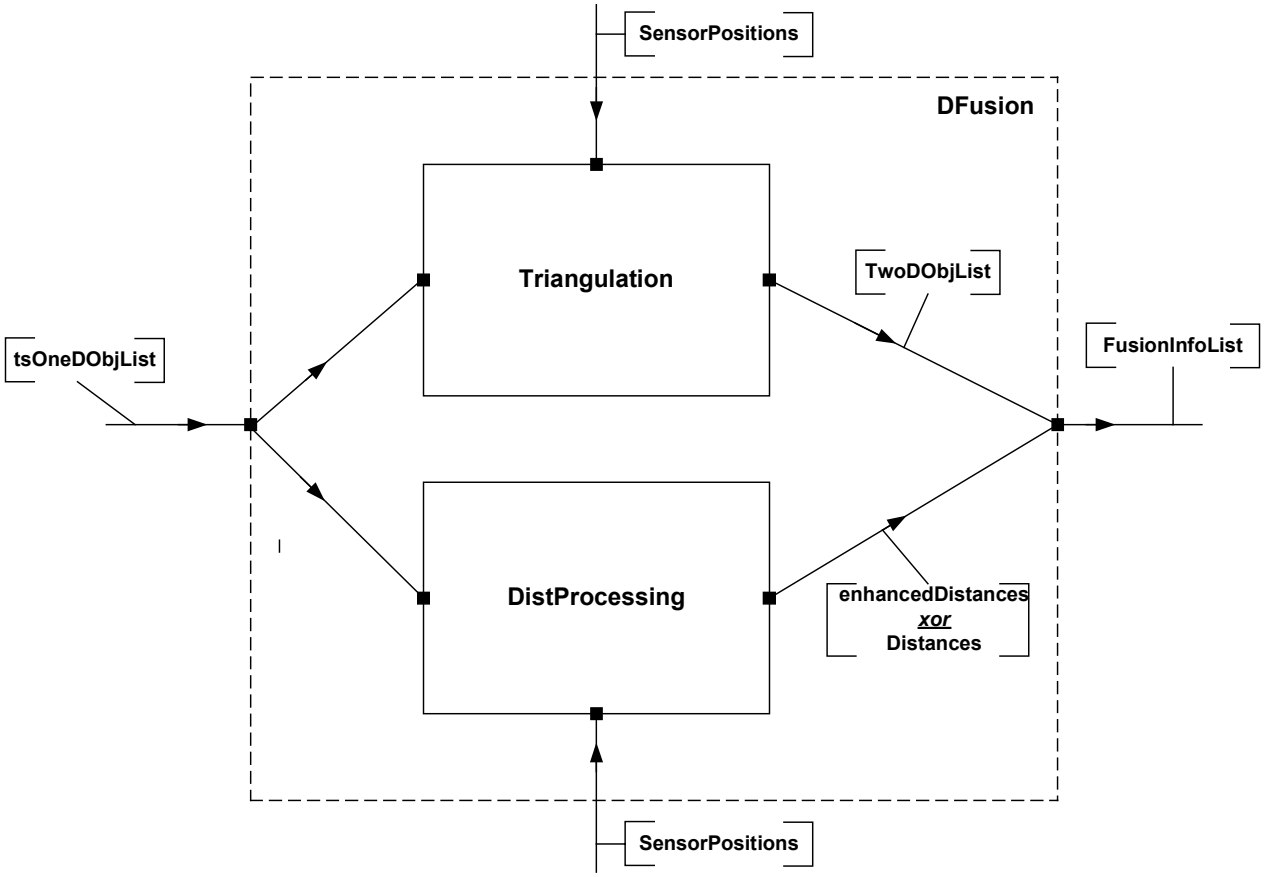

Figure 7: Structure of the component DFusion

2.3.3.3.1.2 DistProcessing

The component *DistProcessing* is relevant only if an object is detected by only one sensor. In this case, if possible, the distance information will be complemented by an offset information derived from the sensor position and the overlap with the neighboring sensors. The object information is then a four-tuple

- t_m : time of measurement
- r : radial distance of object from sensor
- v_r : radial velocity in sensor direction
- *offset* : offset information

If offset information cannot be derived, the incoming information will only be re-formatted to match the output format of *DistProcessing*. In this case a one-dimensional object is represented by the three-tuple (t_m, r, v_r) and has no additional information compared to the input data.

2.3.3.3.2 VFusion

VFusion processes the one-dimensional object data from a *CVMeasurement* sequence. Since the overlap between the sensor regions at this distance range is comparably small, data from different sensors are not fused (e.g., for

triangulation). The fusion is performed for each sensor individually. This means that there are maximally nine input values from a *CVMeasurement* sequence which have to be processed.

VFusion processes each input value as soon as it comes in. After having received and used the second input value of one sensor, *VFusion* delivers the first output value to the applications *PreSet* and *PreFire*. With each following value, the output is updated. So, maximally eight output values are produced for one *CVMeasurement*. *VFusion* delivers the following output information:

- t_m : time of measurement
- $|\Delta y|$: Offset of place of impact from center (absolute value)
- $|v|$: Radial velocity (absolute value)
- α : Crash angle with respect to *x*-axis.
- t_{ti} : Time to impact

The applications *PreSet* and *PreFire* continuously process the output of *VFusion*. It is possible they have to take action before the *CVmeasurement* sequence has completely reached *VFusion* or all corresponding outputs have been produced.

2.3.3.4 SituationAnalysis

The task of the component *SituationAnalysis* is to decide whether the sensors shall be switched from *DistanceScan* mode to *CVMeasurement* mode. This decision is based on the information available from *EnvDescription*. If the mode has to be changed, *SituationAnalysis* will send a *CVMReq* command to *MeasurementControl*. This is done when an object on collision course needs less than 30 ms before it will reach the first range gate RG1.

Initiating this mode switch is the only purpose of *SituationAnalysis*. It is therefore only present in CPS systems with Precrash functionality.

2.3.3.5 PreSet

The component *PreSet* provides the information needed by the airbag control unit in case of an imminent collision to adjust its firing thresholds. This information consists of the four-tuple (Δy, v_{crash}, Δy, α) and is sent via the COI (see Sec. 2.2.2). *PreSet* calculates this information based on the information available from *EnvDescription*.

The data has to be delivered to the airbag control unit 10ms before the predicted crash time. If the expected collision will not take place, the airbag control unit will reset the firing thresholds to the default values automatically. *PreSet* must be able to operate with data from both sensor modes.

2.3.3.6 PreFire

The component *PreFire* uses the output information from *EnvDescription* to generate the command for activating the belt tensioner in case of an imminent collision. The belt tensioner system must receive this command at least 110ms before the predicted collision time. If the collision will not take place, it is the task of *PreFire* to de-activate the belt tensioner after a certain period of time. The same is true for the case that the collision takes place.

2.3.3.7 Parking Aid

The component *ParkingAid* determines the distance between the car and objects during parking and indicates to the HMI when the distance decreases below a lower threshold. *ParkingAid* can only process data which is produced in the mode *DistanceScan*. In the mode *CVMeasurement*, it is not possible to obtain reliable and accurate measurements in the close vincinity of the car. Therefore, *ParkingAid* has to notify the HMI that it is not operational when the sensors are switched to *CVMeasurement* mode. *ParkingAid* will only become active below a velocity of 15 km/h.

3. Timing

The ECU is running the operating system OSEK [2]. This means, the components *MeasurementControl*, *CVEstimation*, *EnvDescription*, *SituationAnalysis*, *PreSet*, *PreFire* and *ParkingAid* (see Figure 4) must be assigned to OSEK tasks. In the normal case, OSEK tasks are executed at regular intervals (e.g., each millisecond, each 10 ms, etc.) but it is also possible to define asynchronuous tasks.

The computation time consumption of the ECU components depends on the measurement mode and the volume of data that must be processed. The latter varies, for instance, with the number of objects detected in *DistanceScan* mode. The following figures can be used: *MeasurementControl*: 0.4 ms, *CVEstimation*: 1 – 2 ms, *EnvDescription*: 8 – 10 ms (varying with the number of objects), *SituationAnalysis*: 1 – 2 ms, *PreSet* + *PreFire*: 2 – 3 ms, and *ParkingAid*: 0.1 ms.

4. Problems

The CPS system is still under development and a complete timing analysis has not yet been done. Obviously there are many parameters and assumptions which influence the fulfillment of timing requirements, e.g., the hardware platform, maximal and minimal relative speed of objects, minimal distance between objects, lead time of pre-crash detection, maximal number of applications, the scheduling concept (task assignment, *DScan* frequency) etc. Some of them are fixed, for some of them working assumptions are made. In this situation, formulating one single timing problem is not helpful. Instead, any systematic timing analysis is valuable which provides solid results about the relation between these parameters. The following questions are examples:

- Is the 15 ms rate realistic for the *DistanceScan* mode? How big is the margin left for longer runtimes of the components or if additional components have to be added? Can it still be realized when the components are assigned to synchronous OSEK tasks? How much overhead for task switching is acceptable?
- Will the lead time requirements of *PreSet* and *PreFire* be fulfilled for a fast (200 km/h) relative velocity?
- What assumptions have to be made about the collision objects (number, velocities, follow-up distance, curvature) such that the lead time requirements of *PreSet* and *PreFire* can be guaranteed. (In particular, which implications has the intermediate *DistanceScan* over 2 meters?)
- What is the optimal scheduling of the components in the sense of robustness (against computation delays) and flexibility (addition of further applications on the ECU) while still guaranteeing, for instance, the pre-crash lead time requirements?

5. References

[1] R. Moritz. Pre-crash Sensing – Functional Evaluation based on Short Range Radar Sensor Platform. SAE Conference, Detroit, 2000, paper no. 00IBECD-11. Available from AMETIST web site: http://ametist.cs.utwente.nl/RESEARCH/Moritz_precrash.pdf.

[2] OSEK/VDX Operating System Specification, Version 2.2, September 2001. Available at http://www.osek-vdx.org/mirror/os22.pdf.

6. Abbreviations

- to be added -

7. Appendix

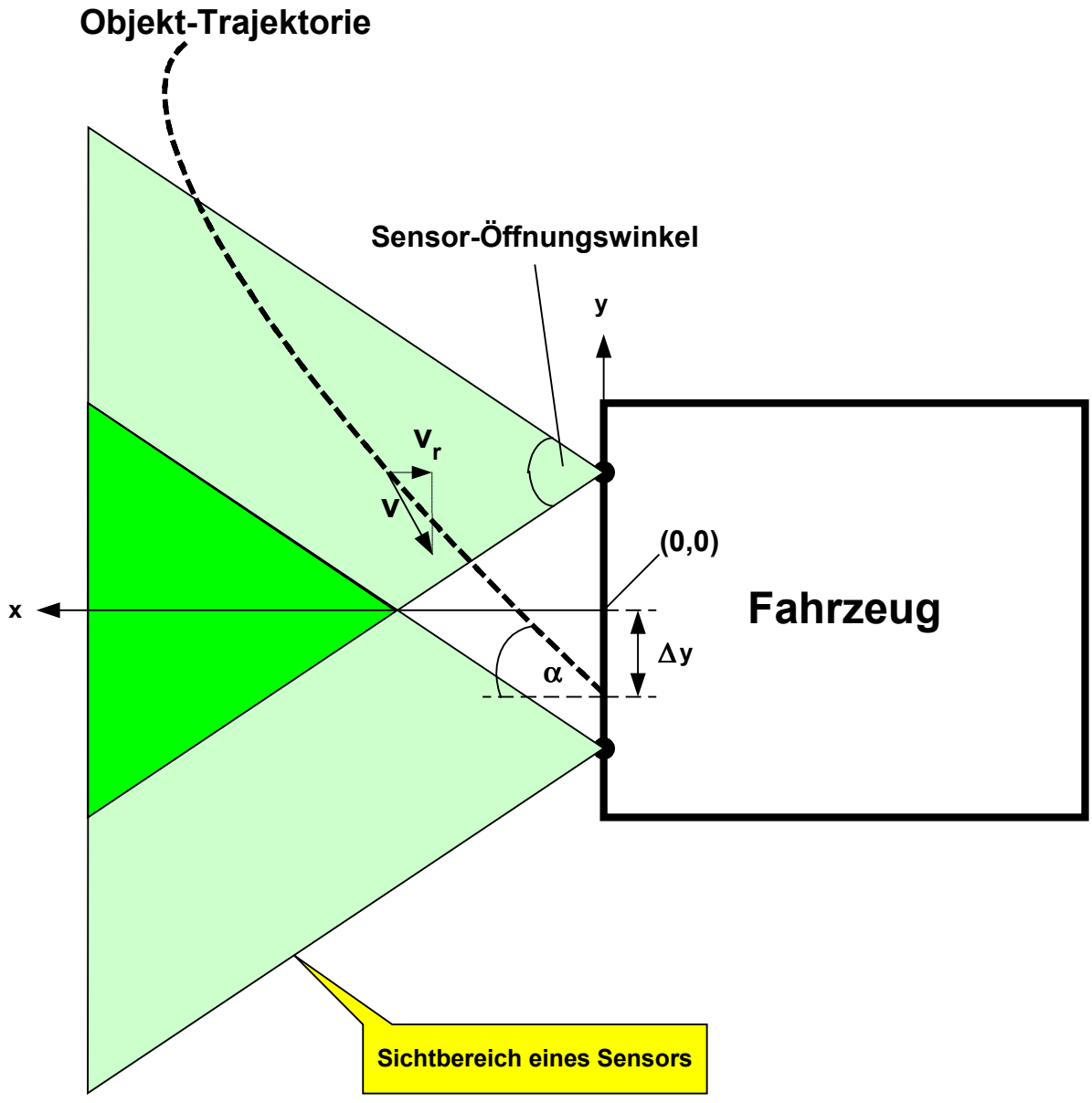

Figure 8: Basic geometry of the CPS system

Translations for Figure 8: "Sensor-Öffnungswinkel" = aperture angle of a sensor, "Fahrzeug" = vehicle, "Sichtbereich eines Sensors" = visibility region of a sensor.