

RSS Documentation

Cornelius Buerkle, Bernd Gassmann, Fabian Oboril

Version 0.29, 2019-02-04

Table of Contents

Overview.....	2
1. Introduction	3
1.1. RSS Summary	3
1.2. Purpose and Scope of this Library.....	4
1.3. Overview of the Document	5
High Level Design.....	6
2. RSS Module Realization	7
2.1. RSS checks and response	7
2.1.1. Longitudinal conflicts	7
2.1.2. Lateral conflicts.....	8
2.1.3. Combining longitudinal and lateral response	8
2.1.4. Combining all situations of a given point in time.....	8
2.1.5. Evasive maneuvers	8
2.1.6. Handling of Intersections	9
2.1.7. Response Time and Other Parameters	9
2.2. Situation-Based Coordinate System.....	10
2.2.1. Comparing movements in lane-based coordinate systems	10
2.2.2. Chosen Design: Individual Situation-Based Coordinate System	12
2.2.3. Design alternative: Iterative Approach [optional]	16
2.3. Summary	16
2.3.1. Key decisions	16
3. RSS System Architecture Overview.....	17
3.1. Sense	18
3.1.1. SensorSubsystem	18
3.2. Plan	20
3.2.1. PlanningSubsystem	20
3.3. Act	20
3.3.1. ActuatorSubsystem	20
3.4. RSS	22
3.4.1. RssSubsystem	22
4. RssModule: Software Architecture	25
4.1. Static View	25
4.1.1. Artifact Deployment	26
4.1.2. Interfaces	26
4.1.3. DataTypes	30
4.2. Dynamic View	52
4.2.1. RssSituationExtractionImpl	52
4.2.2. RssSituationCheckingImpl	54

4.2.3. RssResponseResolvingImpl	55
4.2.4. RssResponseTransformationImpl	56
5. Design for Security	59
5.1. Architecture and dataflow overview	59
5.2. RSS input data (WorldModel)	61
5.2.1. Input data source	61
5.2.2. Error handling and logging	61
5.2.3. Non-functional requirements	61
5.3. RSS output data (AccelerationRestriction)	62
5.3.1. Output data destination	62
5.3.2. Bounds on the output data values	63
5.3.3. Bounds on the output data size	63
5.4. RSS library (RssCheck and supporting classess)	64
5.4.1. Dependencies	64
5.4.2. Coding standards	64
5.4.3. Compiler and compiler security flags	64
5.4.4. Code analysis	64
5.4.5. Critical data path and data structures	64
6. Design for Safety	65
Appendix	66
7. Parameter Discussion	67
7.1. Decision on Selected Parameter Values	68
7.1.1. Response time	68
7.1.2. Longitudinal Acceleration	68
7.1.3. Lateral Acceleration	70
8. Terminology	72
9. References	73
10. Revision History	74
11. Build RSS core library and documentation	75
11.1. Build instructions	75
11.2. Generate PDF document	75
11.3. Generate HTML document	75

Copyright (c) 2018-2019 Intel Corporation

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND

ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.

IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT,

INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,

BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA,

OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY,

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

Overview

Chapter 1. Introduction

The Responsibility-Sensitive Safety (RSS) model is designed to formalize and contextualize human judgment regarding all multi-agent driving situations and dilemmas. RSS formalizes terms like dangerous situations, proper response and notion of blame in a mathematical way. From planning and decision-making perspective, RSS ensures that the AD system will not issue a command that would lead to an accident.

1.1. RSS Summary

The available RSS publications (see [Table 2](#)), can be summarized as follows:

1. RSS continuously monitors the *current* state of the environment, in order to determine if the ego vehicle is *currently* in a safe state. If the ego vehicle is not in a safe state, RSS will provide a response action that will bring the car back into a safe state. As a result, in case of an accident, the ego vehicle cannot be blamed responsible, as it was not causing the collision.
2. A state is regarded as *safe*, if the ego vehicle is not causing a collision with another object, under the worst case assumption that the ego vehicle will accelerate (depending on the situation this can be also a deceleration) at maximum possible speed during its response time. Hence, RSS does not take the output of the driving policy into account. However, as RSS uses worst case assumption on reaction time, acceleration etc., it is guaranteed that, no valid action of the driving policy can bring the vehicle into an unsafe state, if RSS regards the current situation as safe.
3. If the ego vehicle is in a *dangerous* situation, RSS will assure a proper reaction, that will bring the car back into a safe state. Therefore, it will impose proper restrictions for the longitudinal and lateral accelerations of the driving command, that is send to the ego vehicle. Note that this will assure that the vehicle reacts correctly, but the driving policy still has the chance to solve the dangerous situation on a more elaborate way, as long as the desired action is within the RSS limits. The reason is that RSS has only basic information about the environment, whereas the driving policy can use much more information, and is able to perform much more sophisticated path finding strategies.
4. RSS differentiates between *longitudinal* and *lateral* conflicts. A longitudinal conflict means that the distance between the ego vehicle and an object in front or in the back of the ego vehicle is smaller than the longitudinal safety distance. Similarly, a lateral conflict arises, if the distance to the left or right of the ego vehicle to another object is less than the required lateral safety margin. Depending on the type of conflict, RSS requires a different response.
5. In addition, RSS differentiates between normal (single- or multi-lane) roads, intersections and unstructured roads (e.g. parking areas). Depending on the type of road, the required response for a conflict is different.
6. Objects are classified into *Vulnerable Road Users* (e.g. pedestrians) and other (dynamic) traffic objects. The reason for this separation is that the first object group requires special safety considerations, as for example pedestrians may have unknown routes, compared to vehicles.
7. RSS is not about:
 - How to get "good enough" sensor data. It is about the usage, which may impose some sensor

requirements.

- Avoiding collisions, if other traffic participants show a completely erratic driving behavior. Instead, it is about ensuring that the ego vehicle cannot be blamed for the accident.

1.2. Purpose and Scope of this Library

The design of the library at hand is based on the academic paper "*On a Formal Model of Safe and Scalable Self-driving Cars*". The library provides a C++ implementation of RSS according to the aforementioned summary. The key component of this implementation is called "*RSS module*". This module receives (post-processed) sensor information and provides actuator command restrictions as output.

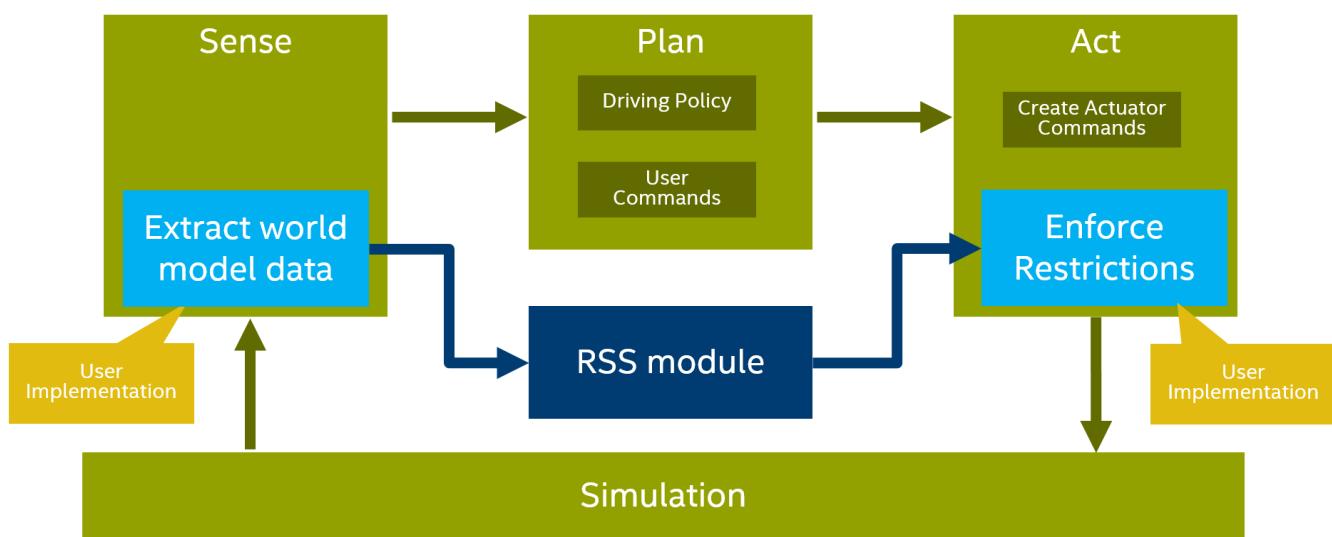


Figure 1. RSS module (implemented in the library) and its interfaces to the outside world

In summary, the RSS module receives an object list, with information about all objects in the surrounding environment of the ego vehicle. Then, the RSS module creates an object - ego vehicle pair, for each object. This pair is usually referred to as "*Situation*". For all situations, the aforementioned RSS checks are performed and a proper response is calculated. Finally, one overall response is computed by the RSS module, and the corresponding actuator command restrictions (i.e. lateral and longitudinal acceleration restrictions) are sent out (see [Figure 1](#)).

i The conversion from sensor data to the object list required by the RSS module, as well as the conversion of the actuator command restrictions to real driving commands, have to be implemented by the user of this library, as these parts heavily depend on the actual vehicle setup.

The current scope of the provided implementation is:

- i**
- Standalone C++-library containing the implementation of the RSS module
 - The RSS module covers multi-lane roads and intersections
 - Code quality is assured by automated testing with test coverage of 100% on methods and 80% on branches as well as static code analysis

The initial design does not yet cover:



- Compensating for improper behavior of others
- Lateral conflicts in case of intersections
- Respecting occlusions
- Checks with Vulnerable Road Users
- Checks for unstructured roads, e.g. parking spaces

1.3. Overview of the Document

The remainder of this document is structured as follows:

- [Chapter 2](#) provides an overview of the RSS implementation provided within the library. It discusses how each situation is handled.
- [Chapter 3](#) introduces the system architecture for RSS and the interfaces of the RSS module to other components.
- [Chapter 4](#) provides a description of the software architecture, included the interface and data type definitions. If you would like to integrate the library into your system, this section, as well as [Chapter 3](#) are of particular importance.

High Level Design

Chapter 2. RSS Module Realization

2.1. RSS checks and response

To check whether the ego vehicle is in a safe state, all the objects in the surrounding must be respected. To do so the RSS module will perform an analysis against all the objects in the environment individually. Meaning, for each object in the environment the RSS module will check whether the ego vehicle conflicts with this object. Therefore, longitudinal and lateral checks are performed. As mentioned earlier, these checks are performed separately for each object - ego vehicle pair, i.e. for each situation.

At this, the type of situation has to be evaluated upfront outside of this library implementation. Otherwise, this RSS implementation would enforce a concrete representation of the environment and i.e. the map data with lanes, intersections and priority rules. The situations types that have to be identified are:

1. both vehicles drive on the same road
 - in the same direction
 - in opposite direction
2. both vehicles drive on different roads that are connected by an intersection
 - the ego vehicle has priority over the other vehicle
 - the other vehicle has priority over the ego vehicle
 - both vehicles have same priority (no vehicle has priority over the other vehicle)

2.1.1. Longitudinal conflicts

The behavior for longitudinal conflicts (checks and response) for vehicles driving in the same direction are implemented as described in the definitions 1, 3 and 4 of paper 1 (see [Table 3](#)). For the case of vehicles driving in opposite directions, the implementation follows the definitions 2, 3 and 4 of the paper.

In detail, the current realization looks as follows:

2.1.1.1. Same direction

If the longitudinal distance is not safe and

1. if the ego vehicle is in front, the other vehicle has to break longitudinally. There is no longitudinal response for the ego vehicle issued.
2. otherwise, the ego vehicle has to break longitudinally with at least $a_{min,brake}$.

2.1.1.2. Opposite direction

If the longitudinal distance is not safe and

1. if the ego vehicle is driving in the correct direction, the ego vehicle has to break longitudinally

2.1. RSS checks and response

with at least $\alpha_{min,brake,correct}$.

2. otherwise, the ego vehicle has to break longitudinally with at least $\alpha_{min,brake}$.

2.1.2. Lateral conflicts

The lateral checks and the proper response follow the definitions 5, 6, 7 and 8 of paper 1 (see [Table 3](#)).

In detail, the current realization looks as follows: If the lateral distance is not safe and

1. if the ego vehicle is on the left side, the ego vehicle has to break laterally with at least $\alpha_{brake,min}^{lat}$ on the right side
2. if the ego vehicle is on the right side, the ego vehicle has to break laterally with at least $\alpha_{brake,min}^{lat}$ on the left side
3. otherwise, the ego vehicle has to break laterally with at least $\alpha_{brake,min}^{lat}$ on both sides

2.1.3. Combining longitudinal and lateral response

The combination of longitudinal and lateral response of a single situation (object - ego vehicle pair) is implemented as described in definitions 9 and 10 of paper 1 (see [Table 3](#)).

In detail, the current realization looks as follows:

If the situation is dangerous (i.e. there exists a longitudinal and a lateral conflict), the stored last non-dangerous state of the of the same situation (ego-vehicle - object pair) is checked:

1. if there was no lateral conflict, the combined response breaks laterally
2. if there was no longitudinal conflict, the combined response breaks longitudinally

2.1.4. Combining all situations of a given point in time

Since this RSS implementation performs the above mentioned check separately for each situation, the overall response of the ego vehicle has to consider all individual situations of the current scene.

The current realization loops over all situations and combines the lateral left, the lateral right and the longitudinal response states of these by selecting the most severe response of each component respectively.

2.1.5. Evasive maneuvers



This initial implementation does not yet cover evasive maneuvers to compensate for improper behavior of others according to definitions 11, 12 and 13 of the paper 1.

If the car finds itself in a dangerous situation one possible action is always to brake. This should always result in a safe state, if both vehicles respond properly according to RSS. Improper behavior of others might still lead to an accident. This is where evasive maneuver could prevent from this. To do so, it must be assured that this lateral movement brings the vehicle into a safe state and does not

conflict with another vehicle. In order to determine whether a lateral movement solves the conflict a prediction of the state would be necessary.

Such a naive predication following definition 11 of the paper is not available yet in this initial implementation. In addition, the RSS module currently has no notion of drivable freespace area. Hence, it cannot determine that an evasive maneuver initiated by RSS will cause an accident with an obstacle or forces the vehicle to leave the road.

Therefore, it is impossible for the RSS module in its current form to detect whether a lateral evasion is really feasible. Hence, *the RSS module will not initiate a lateral evasive maneuver* according to definitions 12 and 13 of the paper. Instead, it will only restrict the movement in the dangerous direction.

Please note that this restriction does not hinder the driving policy to find a better escape for the current situation. If this is the case, for example by braking harder, or changing lanes quicker, RSS will not forbid this maneuver, as long as it does not create another conflict, and is compliant with the restrictions calculated by RSS.

2.1.6. Handling of Intersections

The behavior for intersection conflicts (checks and response) for vehicles is implemented as described in the definitions 16, 17 and 18 of paper 1 (see [Table 3](#)).

In detail, the current realization looks as follows:

1. It is checked, if a non-prioritized vehicle *was* able to stop in front of the intersection. If this is the case, the non-prioritized vehicle is supposed to brake, whereas the prioritized vehicle can continue driving as before.
2. If 1. does not hold, it is checked, if there is a safe longitudinal distance between the two vehicles according to Definition 17.2 of paper 1. In this case, the leading vehicle can continue driving, whereas the following vehicle has to respect the "stated braking pattern".
3. If 1. and 2. do not hold, there is a time period in which both vehicles may be crossing the intersection. In this case Definition 18.3 of paper 1 applies, i.e. both cars have to brake laterally and longitudinally with at least $a_{min,brake}$.



Case 1. is the direct realization of Definition 17.1, where it is mentioned that the vehicle *was* able to stop safely. However, as a consequence, the prioritized vehicle is not forced to brake, if the non-prioritized vehicle does not respect RSS. Since the evasive maneuvers are not yet implemented [Section 2.1.5](#), such kind of improper behavior cannot be handled by the current RSS module.



In the current realization of the RSS module, it is assumed that there is always a lateral conflict in case of intersections. This will be addressed in future.

2.1.7. Response Time and Other Parameters

According to the papers each traffic participant has a response time, and is objected to respect certain acceleration limits (e.g. maximum acceleration $a_{accel,max}$, maximum deceleration $a_{brake,max}$, etc.).

Within this response time the participants (including the ego vehicle) are allowed to accelerate with at most $\alpha_{accel,max}$, and thus increase their velocity. The distance covered during the response time is part of the safe distance, as defined by RSS. Hence, upon entering a dangerous situation, it would be possible to accelerate with up to $\alpha_{accel,max}$ for at most $t < \text{response time}$, as this acceleration is already considered.



It is important to note that the implementation of the RSS module in the library only uses parameters, but not the exact value. By this means, the library is independent to changes of the parameter values. Instead, the user defines a feasible parameter set, which is provided as input to the RSS module.

A discussion on the parameter selection can be found in [Chapter 7](#).

2.2. Situation-Based Coordinate System

As described in paper 1 section 3.2 "Preliminaries — A Lane-Based Coordinate System" (see [Table 3](#)), all RSS calculations are based on a lane-centric coordinate system. This system uses adjacent, straight lanes of constant width, and thus requires a transformation of the object states from Cartesian into the lane space. This transformation into a lane-based coordinate system is described by a bijective function, as pointed out by paper 1. Therein, the lateral position of a vehicle within the lane is mapped to a parametric interval $[-0.5; 0.5]$, where the lane boundaries are fixed at the borders of the interval. The advantage of such a coordinate system over the Cartesian system is that it allows the direct calculation of longitudinal and lateral distances of objects.

However, when transforming the Cartesian space into a lane-based coordinate system, several challenges have to be taken into consideration.

2.2.1. Comparing movements in lane-based coordinate systems

During the transformation process to a lane-based coordinate system, not only the position but also the velocities and accelerations have to be transformed. As a matter of fact, the resulting values depend on the actual lane geometry, and thus, velocities and accelerations of different lane-based coordinate systems cannot be compared to each other anymore (ego vehicle - object pair). To illustrate this problems, let us consider the following examples:

2.2.1.1. Discontinuity Problem: Two parallel lanes, different width

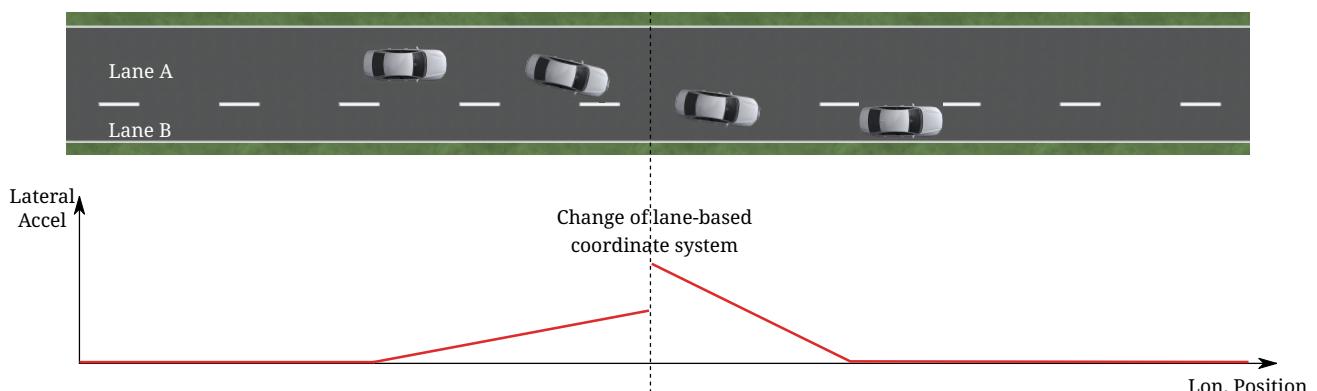


Figure 2. Two parallel lanes with different width causing a discontinuity in lateral acceleration

Let us illustrate this on a simple example with two parallel lanes of different width. Let the left lane A have a constant width of 4 m where the right lane B only has a constant width of 2 m . If both lanes define their own lane-based coordinate system LCS_A and LCS_B , a Cartesian lateral acceleration value of 1 m/s^2 becomes 0.25 lat/s^2 in LCS_A and 0.5 lat/s^2 in LCS_B . Therefore, the formula for constant accelerated movement has to use different acceleration constants in different lanes. This situation is getting even worse, if a car is changing the lane from lane A to lane B: then the closed formula for constant accelerated movement to calculate the lateral distance over time cannot be applied anymore directly.

2.2.1.2. Changing Acceleration Problem: Lane is widening/narrowing

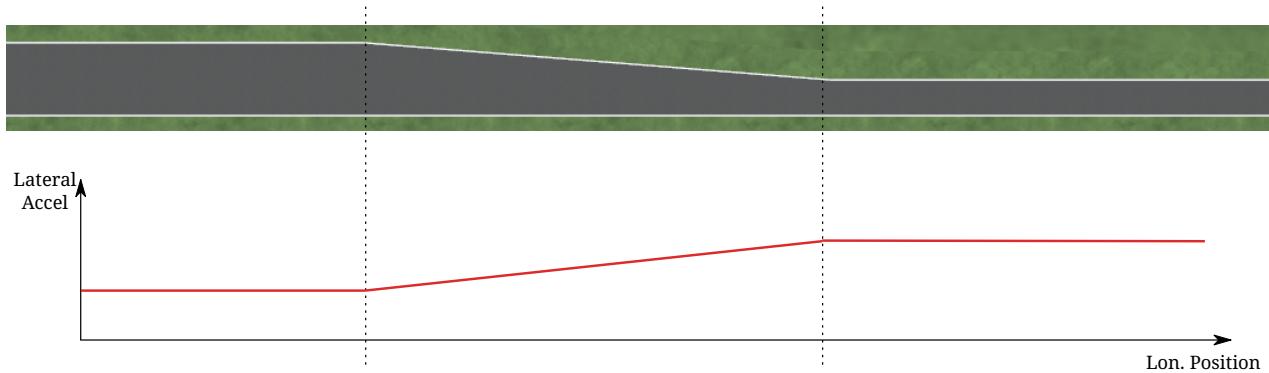


Figure 3. Changing lane width and its impact on the lateral acceleration

Let us consider a lane with changing width in another example. If the lane's width at the beginning is 4 m and 100 m away the lane is narrowing to 2 m . In such a case the Cartesian lateral acceleration value of 1 m/s^2 is changing from 0.25 lat/s^2 at the beginning towards 0.5 lat/s^2 while advancing within the lane.

2.2.1.3. Changing Distances Problem: Lane with a narrow curve

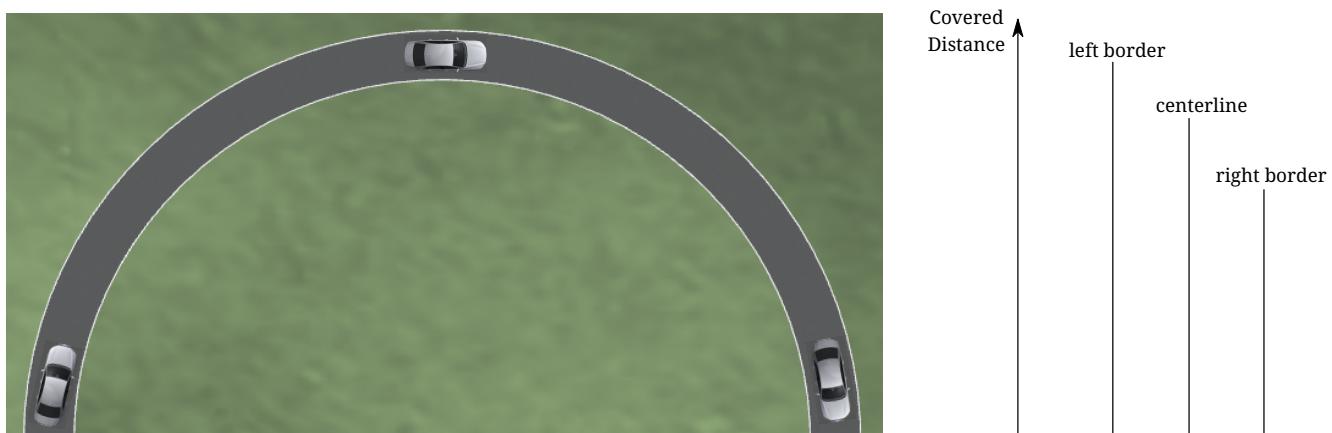


Figure 4. Lane describing a narrow 180° curve and its impact on driven distances

This section illustrates a longitudinal situation similar to the lane widening example. Let us assume the lane has a constant width of 4 m describing a curve with inner radius of 50 m covering 180° . The inner border of the lane has a length of about 157.1 m , the center line 163.4 m the outer border 169.7 m . In that situation a longitudinal acceleration value will evaluate to 1.0 lon/s^2 for the center line, 0.96 lon/s^2 for the outer border and 1.04 lon/s^2 for the inner border. Therefore, the longitudinal acceleration changes over time, if the vehicle changes its lateral position within the lane.

2.2.1.4. Summary

As sketched in the previous sections both the longitudinal as well as the lateral acceleration values, as well as velocities within the lane-based coordinate system cannot be considered as constant anymore. Moreover, these values do not only change within one coordinate system, but also when changing from one lane-based system to another one. To overcome this issue, we use a "Situation-Based Coordinate System", that is described in detail in the next section. This system is unique for each situation (ego vehicle - object pair) and comprises *all* lanes required to describe this situation.

2.2.2. Chosen Design: Individual Situation-Based Coordinate System

As RSS performs a worst case assessment, the idea followed by the RSS module implementation is to calculate the min/max position values of the vehicles within the situation specific coordinate system. According to the constellation of the vehicles within the situation, the respective worst case lateral and longitudinal border values are selected and processed by the RSS formulas. Like this, it is assured that the calculations are sound, nevertheless this might lead to a more cautious behavior of the vehicle. The following subsections describe the selected approach in more detail.

2.2.2.1. Two parallel lanes, different width

As described in [Section 2.2.1](#), the border between neighboring lanes of different width introduces discontinuities of the lateral acceleration values (see [Figure 2](#)).

As the RSS module judges the relative situation between the ego vehicle and the other objects one by one individually, it is not required to distinguish between the actual lanes within the individual distance calculations. Combining all lanes relevant for the individual situation s_i between ego vehicle and object o_i into one single situation-based coordinate system SCS_i resolves all discontinuities, as depicted in [Figure 5](#)

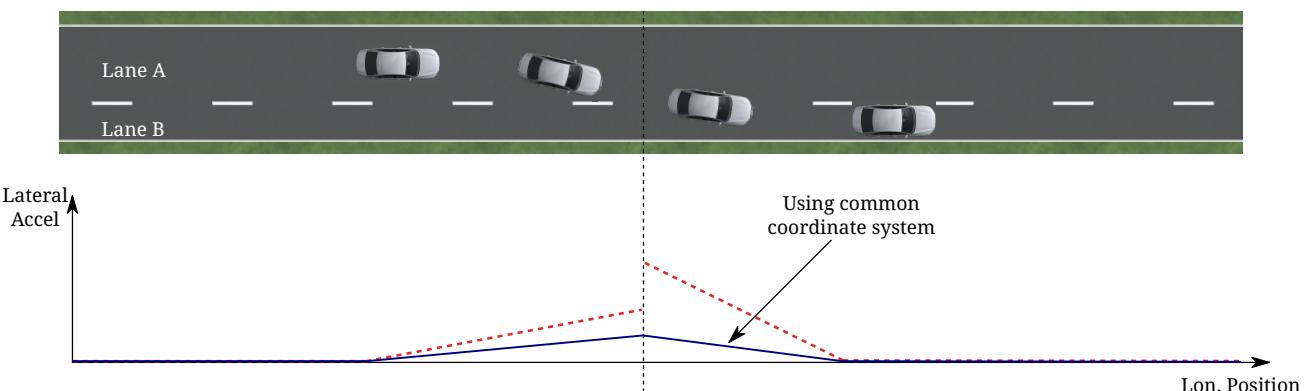


Figure 5. Avoid discontinuities by using one single situation-based coordinate system

Coming back to the concrete example from [Figure 2](#), left lane A having a constant width of 4 m and right lane B having a constant width of 2 m, both lanes together have a resulting width of 6 m and form an area with continuous lateral acceleration (see also illustration in [Figure 5](#)).

The check of the ego vehicle with another object o_j which is two lanes at the right of the ego vehicle in a lane C having a constant width of 3 m, has to take all three lanes into account with resulting width of 9 m. Therefore, a different situation-based coordinate system SCS_j is required, when checking another object.

2.2.2.2. Lane is widening or has a narrow curve

The individual situation specific coordinate system *SCS* does not yet cover the situations of widening lanes or narrow curves. To take the variation of the lane width and length into account, it is required to apply the extrema within the respective *SCS* accordingly.

Again, coming back to the examples from above, let us have a lane with non constant width between 2 m and 4 m. Then the transformation of the maximal possible lateral position value of the vehicle into the situation coordinate system *SCS* has to take the maximum width of 4 m into account, while the transformation of the minimal possible lateral position has to be transformed with the minimum width of the lane of 2 m. Like this it is guaranteed that we don't underestimate the distances of the vehicles towards each other. As a result, it is ensured that under all conditions, the safety distances are calculated in a conservative manner.

In a similar way, it is possible to transform the longitudinal position values into the situation-based coordinate system SCS_k .



The performed operations can be interpreted as enlarging the vehicles bounding boxes to ensure the worst case is covered.

2.2.2.3. Road area

To overcome the problems of discontinuities, changing lateral and longitudinal distances resulting in not comparable velocities and accelerations the situation based coordinate system merges in a first step all lanes segments relevant to the situation (ego vehicle - object pair) into one situation specific metric road area. One can imagine this step as the creation of a bounding box around the two vehicles that is large enough to cover the (relevant) worst case movements of those within the reaction time while ignoring actual markings on the road between the lanes. By this, especially the physical lateral velocities and accelerations within the Cartesian 2D space of the road area can be measured and calculated straight forward without any discontinuities (see also the illustrations in [Figure 6](#) and [Figure 7](#) below).

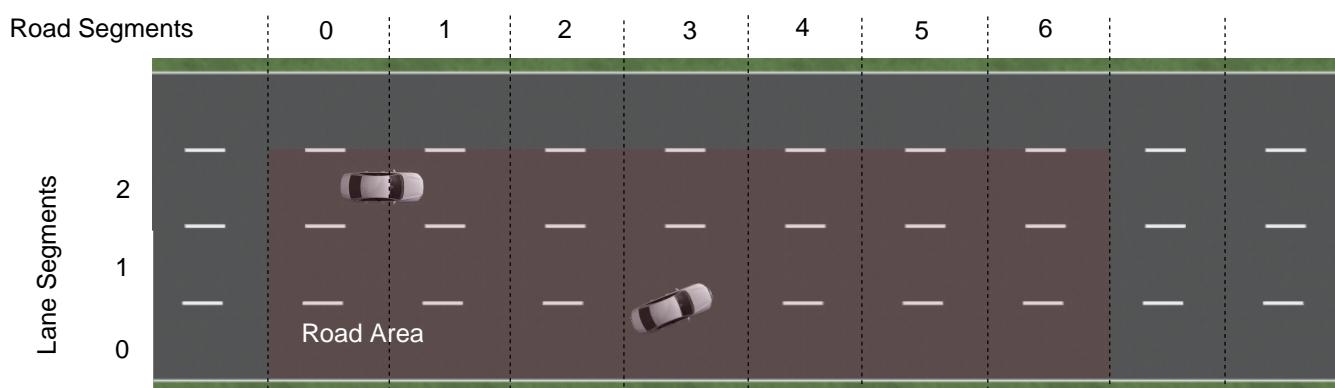


Figure 6. Creation of the situation-based coordinate system: The road area (red) consists of all lane segments along the road relevant for the situation between the two vehicles

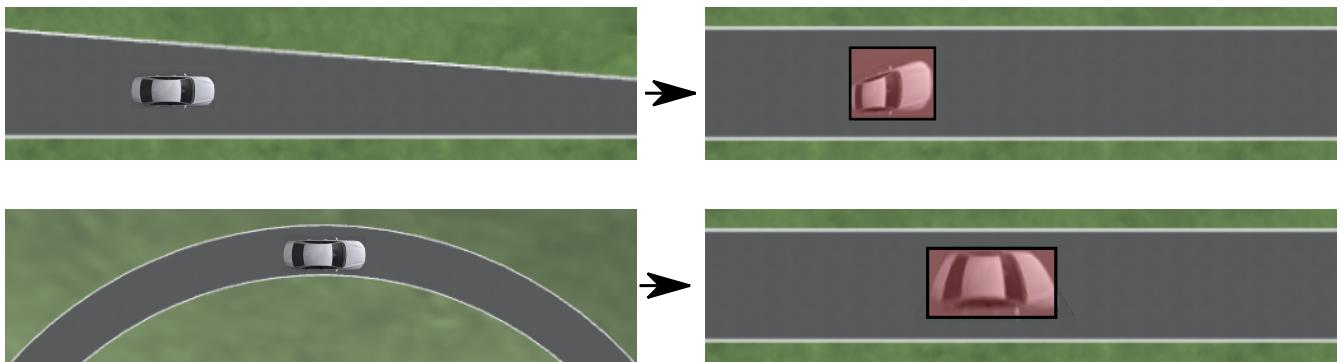


Figure 7. Creation of the situation-based coordinate system: Worst-case transformation of the vehicle bounding box. The metric road on the left leads to transformed vehicles and their bounding boxes (red) on the right, sketched for a narrowing road area at the top and a curve at the bottom.

It is worth to mention, that in these calculations the actual shape of the lane is not used. Therefore, detailed knowledge of the actual lane geometry is not required. The absolute maximum and minimum width and length values of the lane segments is sufficient to calculate a proper transformation into the space of the situation specific coordinate systems.



In case of intersections both vehicles define their own road areas including the intersecting parts. The case that a lane is ending or two lanes are merged into one single lane have to be treated like intersections since a lateral conflict is unavoidable.

2.2.2.4. Considerations on reverse transformation of the proper response

As the proper response is referring to the situation-based coordinate systems, the response has to be transformed back considering the actual lane geometry. Therefore, first the transformation into the vehicle-specific lane-based coordinate system is required, and then the transformation into the Cartesian space is performed.

A simple example illustrates this: a vehicle driving in a curve will for sure have to perform a lateral acceleration in Cartesian space otherwise it will leave the lane because of the centripetal force, as illustrated in [Figure 8](#). However, in the vehicle specific lane-based system the lateral acceleration will be 0.



Figure 8. Constant drive around a curve will result in a zero lateral acceleration in a lane-based coordinate system and in a non-zero acceleration in a cartesian system

Because the proper response of RSS is defined with respect to the actual lane the vehicle is driving in, it is required to assure that the reverse transformation of the proper response considers only the ego-lane and not the complete situation specific coordinate systems. For example, let us consider a scenario as depicted in [Figure 9](#), where one widening lane A and one narrowing lane B are neighbors in such a way that the overall width of the road is constantly 6 m. Lane A starts with 2 m and ends with 4 m width, whereas lane B starts with 4 m and ends with 2 m width. A lateral velocity of 0 in respect to the whole road differs from the definition of a lateral velocity of 0 in lane A/lane B in Cartesian space.

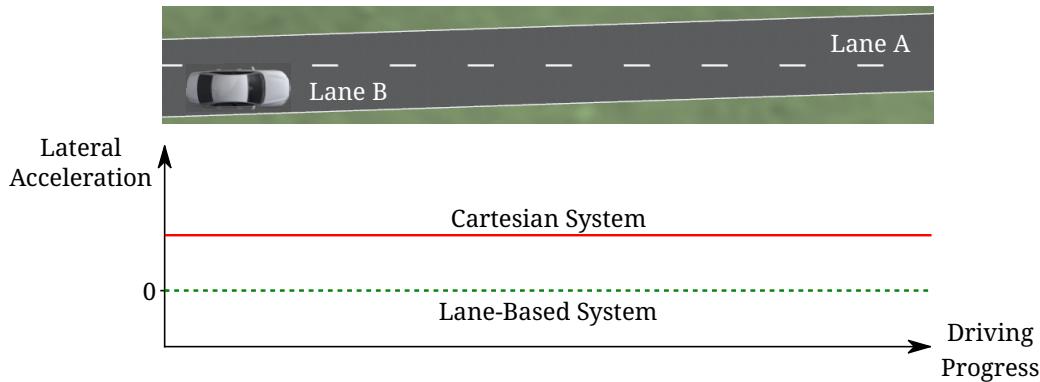


Figure 9. Different lateral accelerations in a lane-based system and Cartesian system for a vehicle following the centerline of lane B



It is worth to note that in the particular implementation of the RSS module in the library at hand, the reverse transformation from the situation-specific into a vehicle-centric lane coordinate system is not required, as the RSS response is defined such that it is independent of these two coordinate system.

2.2.2.5. Summary

The presented construction of a continuous situation-based coordinates system will allow the pairwise calculation of the safe distances between ego vehicle and objects with the assumption of constant acceleration. Still, the worst case assessment of RSS is not violated. This situation-based coordinate system in conjunction with the situation specific consideration of the position extrema allows the calculation of the safe distances, the decision on dangerous situations and deduction of a proper response.

The main benefits of the selected Situation Based Coordinate System definition are:

- * Velocity and acceleration values in longitudinal and lateral lane directions do not change when transforming into the situation based coordinate system
- * All formulas for constant accelerated movement can be applied in the situation based coordinate space.
- * Distance calculations in situation based coordinate systems are simple additions or subtractions
- * The transformation into the situation based coordinate system is simple and therefore can be implemented easily with the restricted computational resources of safety controllers

For safety considerations, it is crucial to do all calculations considering the worst case assumption. Therefore, it is required to choose the correct minimum or maximum value of the positions in the situation-based coordinate system to assure that always the worst case is considered.

2.2.3. Design alternative: Iterative Approach [optional]

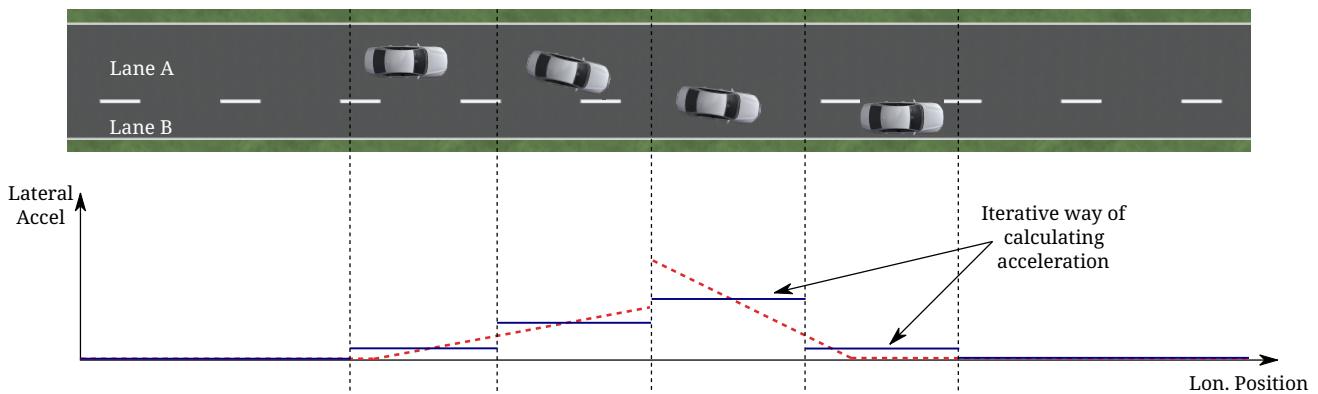


Figure 10. Illustration of an iterative approach to calculate non-constant acceleration, velocity etc.

Another possible way to handle the non-constant acceleration values would be an iterative approach: based on the position, the velocity and the acceleration values at the given position at time t_0 , the position at time t_1 is calculated. The smaller the time interval between the iteration steps is chosen, the smaller the calculation error gets (see [Figure 10](#)).

One drawback of the iterative approach is that the RSS implementation has to get to know the lane geometries in detail to be able to calculate the acceleration values to be used for every position within the situation-based coordinate systems. Therefore, this design approach is not selected by this RSS module implementation.

2.3. Summary

2.3.1. Key decisions

- RSS checks are performed on the current state on a ego vehicle - object pair basis
- In dangerous situations only braking maneuvers are issued. RSS does not initiate evasive maneuvers, but will not hinder the driving policy to execute lateral evasive maneuvers, as long as these are compliant with the required RSS response.
- To handle changing lateral/longitudinal lane geometries when transforming the Cartesian space into the situation based coordinate system, the vehicles position extrema are chosen in such a way that accelerations can still be treated as constant, but guarantee safe operation (see [Section 2.2.2](#)).

Chapter 3. RSS System Architecture Overview

This section sketches the integration of RSS into a existing system architecture.

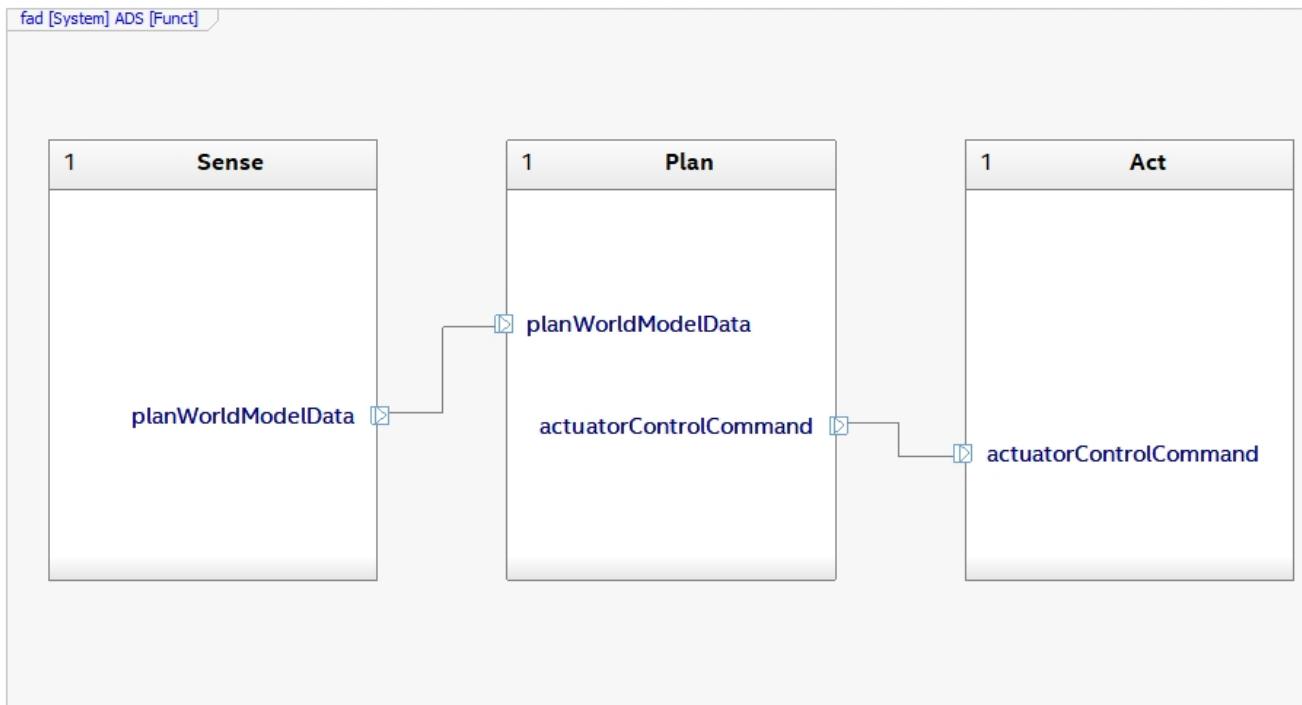


Figure 11. A generic system architecture for an ADS.

From high level view the architecture of an ADS consists of three main parts: Sense, Plan, Act. The world model data provided by the Sense part is used by the Plan part to create the near term actuator control commands executed by the Act part.

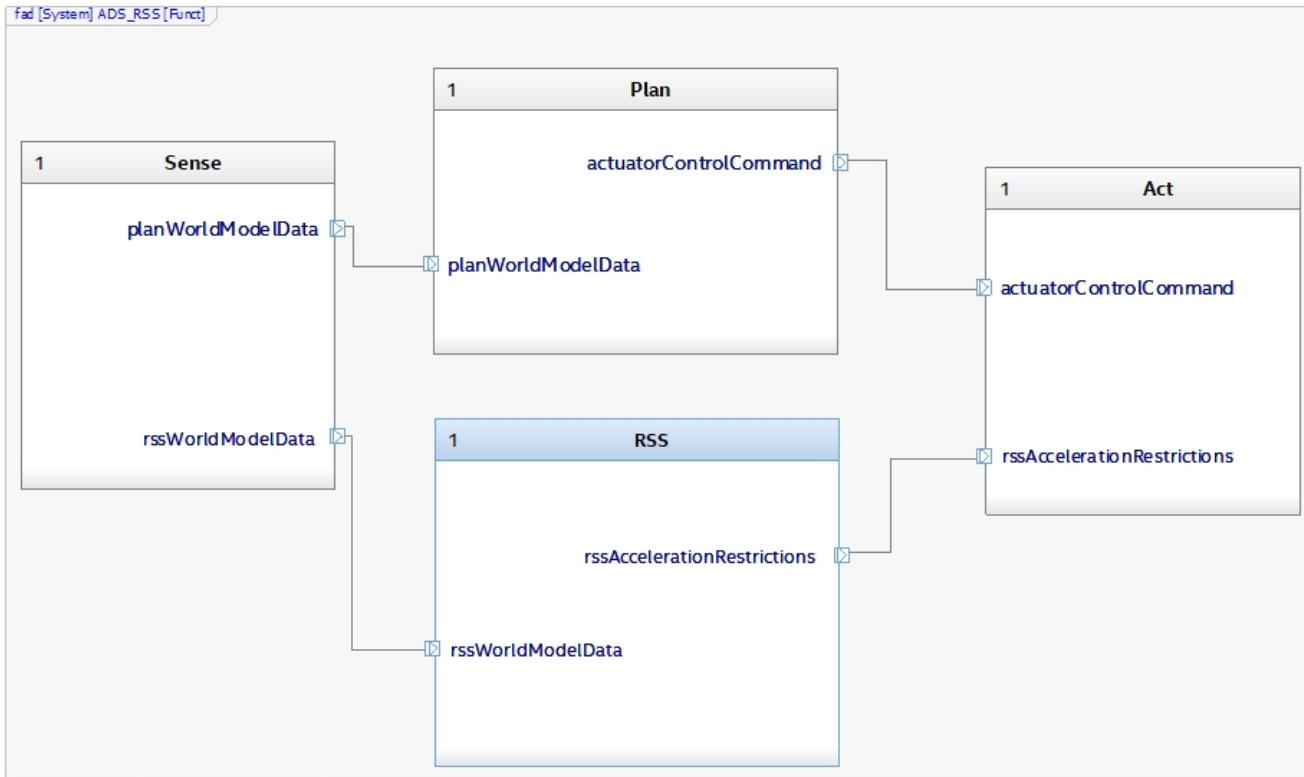


Figure 12. Integration of RSS into a system architecture

RSS is integrated into such an ADS architecture by placing it in parallel to the Plan. To provide a safety envelope around the planning output the integration of RSS into a ADS architecture spans into the Sense part to provide the required RSS world model data as well as into the Act part to limit the actuator control commands to the RSS restrictions.

3.1. Sense

The Sense part gathers information on the environment required to fulfill the ADS task. In general there exist several level of perception and fusion which enrich the world model to the extend required for the planning algorithms. In this high-level architecture overview the Sense part takes over this task.

3.1.1. SensorSubsystem

The SensorSubsystem realizes the Sense part functionality. It is responsible for the perception of the environment. It interfaces to the real world by receiving, processing and fusing sensor information. It provides all information in form of the world model to the other ADS subsystems. The exact content of the world model data is highly dependent on the concrete realization of the receiving subsystems, especially the degree of perception and fusion applied will differ. The provided world model data might include raw sensor data, high level object data, but also a-priori knowledge such as AD map data. To account for this the SensorSubsystem provides separate output ports for every connected subsystem.

3.1.1.1. Output Ports

planWorldModelData: The world model data required by the PlanningSubsystem

`rssWorldModelData`: The world model data required by the RssSubsystem.

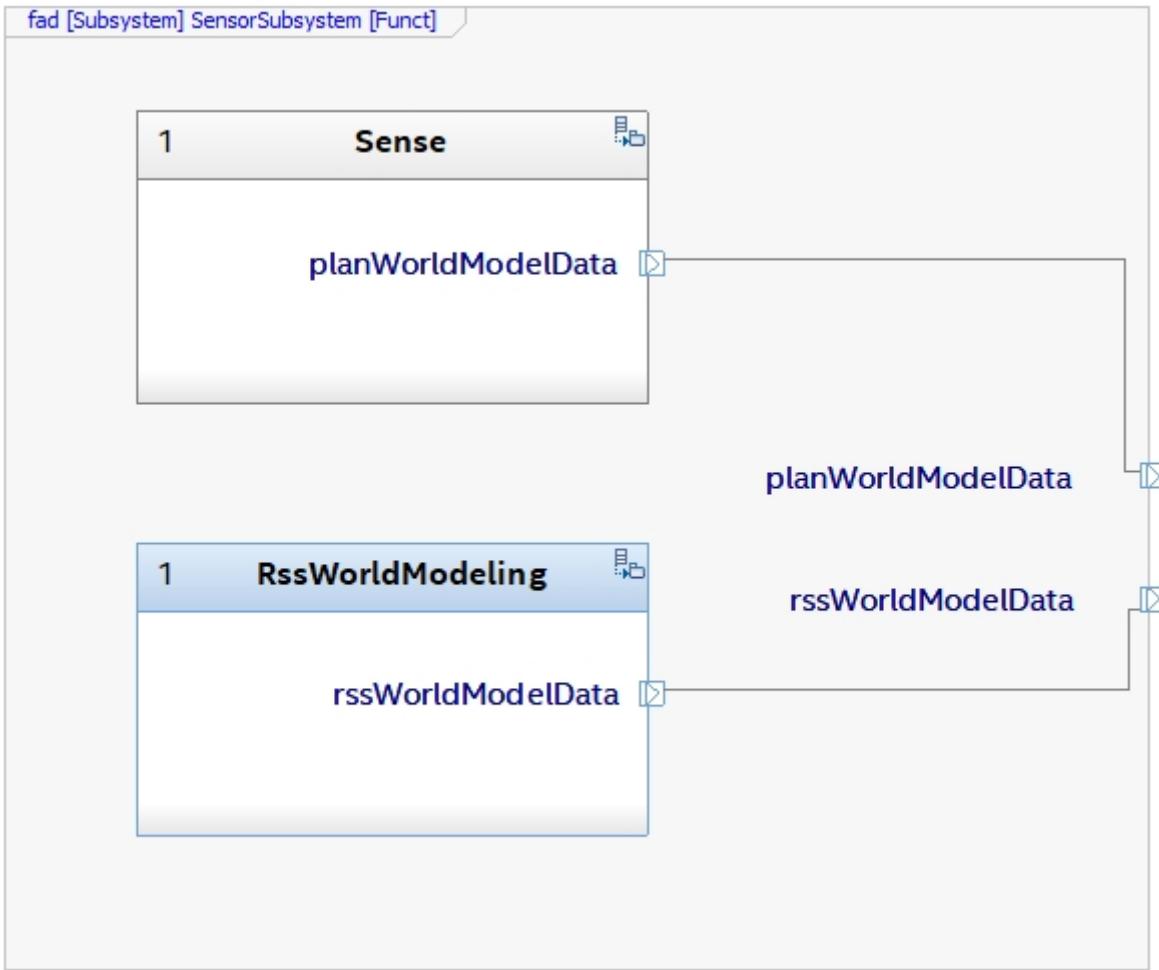


Figure 13. Extending the Sense part with world model data required by RSS.

3.1.1.2. Sense

The Sense entity is responsible to create the world model data required by the PlanningSubsystem. This contains the ADS Planning specific implementation of the Sense part which already exists.

3.1.1.2.1. Output Ports

`planWorldModelData`: The world model data required by the PlanningSubsystem

3.1.1.3. RssWorldModeling

The RssWorldModeling entity is responsible to create the world model data required by the RssSubsystem. The user has to implement this functionality as part of the RSS integration efforts into the ADS system. The `RssWorldModelData` has to be filled with meaningful and reliable data.

3.1.1.3.1. Output Ports

`rssWorldModelData`: The world model data required by the RssSubsystem.

3.2. Plan

The Plan part processes the information on the environment to create the internal world model required to plan the next moves of the ADS.

3.2.1. PlanningSubsystem

The PlanningSubsystem realizes the Plan part functionality. It performs the decision-making of the ADS. It analyses the provided PlanWorldModelData and decides what action to take. Finally, this leads to concrete control commands for the ADS ActuatorSubsystem defining the next move of the ADS.

3.2.1.1. Input Ports

planWorldModelData: The world model data required to perform the planning task

3.2.1.2. Output Ports

actuatorControlCommand: The control commands to realize the next moves within the current plan.

3.3. Act

The Act part executes the moves which the Plan part has calculated.

3.3.1. ActuatorSubsystem

The ActuatorSubsystem realizes the Act part functionality. It receives the ActuatorControlCommands controlling i.e. acceleration, braking and steering to execute the plan.

3.3.1.1. Input Ports

actuatorControlCommand: The control commands from the PlanningSubsystem to control the vehicle actuator system.

rssAccelerationRestrictions: The restrictions on the acceleration for the vehicle calculated by RSS.

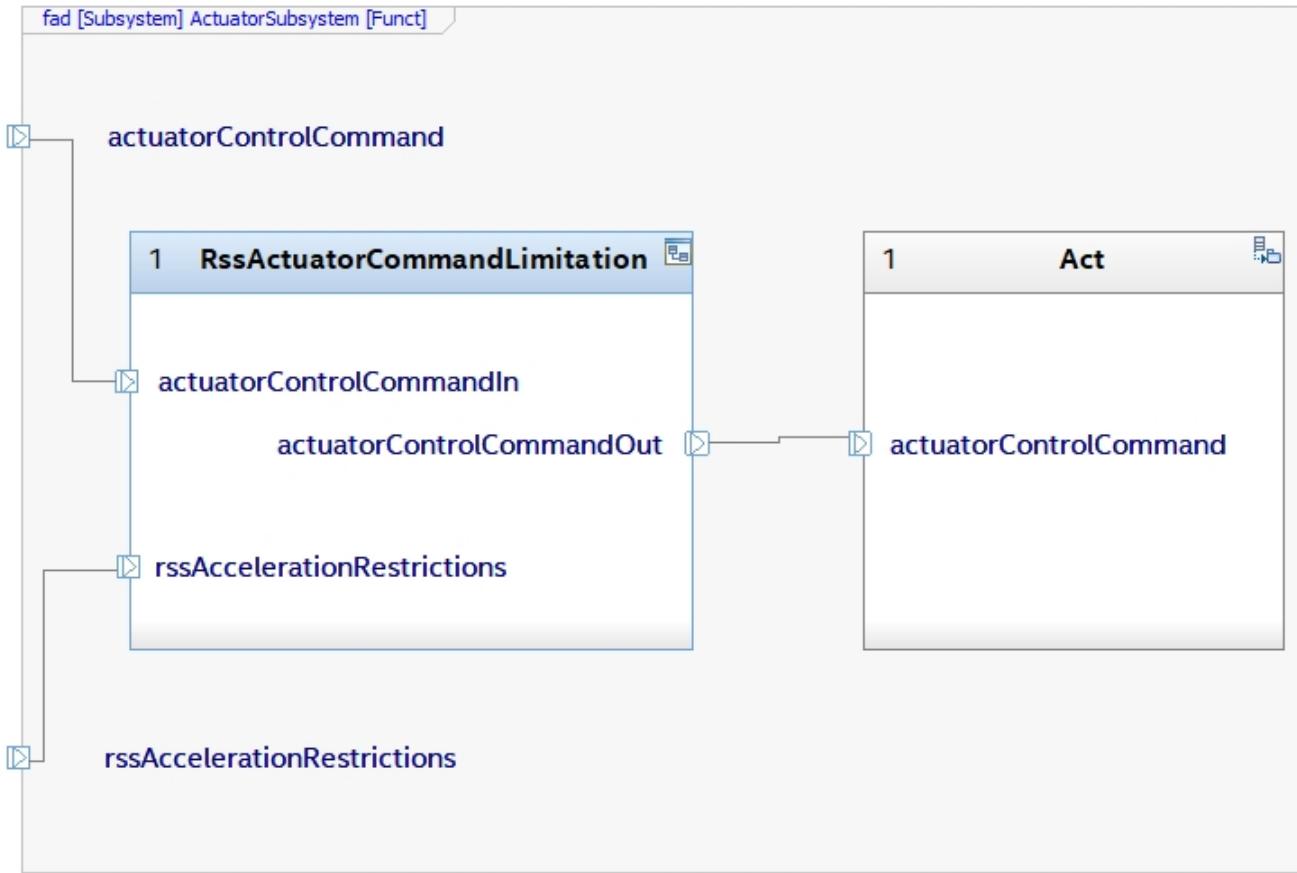


Figure 14. Extending the Act part by RSS command limitation.

3.3.1.2. RssActuatorCommandLimitation

The RssActuatorCommandLimitation entity receives the actuator control commands from the PlanningSubsystem and restricts the control values according to the restrictions calculated by RSS. The resulting actuator control commands are safe in respect to RSS rules. This calculation heavily depends on the actual representation of the ActuatorControlCommands data. Therefore, a generic implementation is not possible. Therefore, the user has to implement this functionality as part of the RSS integration efforts into the ADS system.

3.3.1.2.1. Input Ports

rssAccelerationRestrictions: The restrictions on the acceleration for the actuator control calculated by RSS.

actuatorControlCommandIn: The control commands from the PlanningSubsystem to control the vehicle actuator system.

3.3.1.2.2. Output Ports

actuatorControlCommandOut: The adapted control commands from the PlanningSubsystem to control the vehicle actuator system in a RSS safe manner.

3.3.1.3. Act

The Act entity is responsible to execute the moves which the PlanningSubsystem has calculated. This contains the ADS actuator specific implementation of the Act part which already exists.

3.3.1.3.1. Input Ports

actuatorControlCommand: The control commands to control the vehicle actuator system.

3.4. RSS

The RSS part restricts the moves the Plan part has calculated according to the RSS proper response

3.4.1. RssSubsystem

The RssSubsystem realizes the RSS part functionality. It implements the RSS checks based on the RssWorldModelData received from the SensorSubsystem:

1. Keep a safe distance from the car in front
2. Leave time and space for others in lateral maneuvers
3. Exhibit caution in occluded areas
4. Right-of-Way is given, not taken

In case a dangerous situation is detected a respective proper response is calculated and the actuator control commands received from the PlanningSubsystem are restricted accordingly to realize planning safety.

3.4.1.1. Input Ports

rssWorldModelData: The world model data required to calculate the RSS checks

3.4.1.2. Output Ports

rssAccelerationRestrictions: The restrictions on the acceleration for the actuator control calculated by RSS.

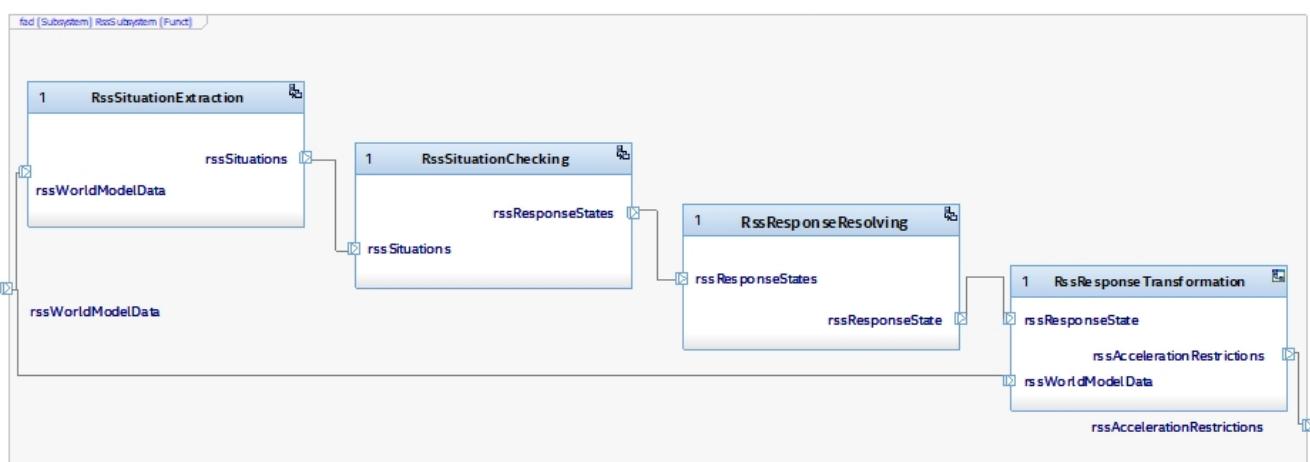


Figure 15. RSS internal processing steps to perform RSS checks and execute the RSS proper response

3.4.1.3. RssSituationExtraction

The RssSituationExtraction entity transforms the global Cartesian world model data into individual RssSituations between the ego vehicle and each of the objects. For every pair <ego-vehicle, object>

in the world model data the individual situation coordinate system transformation is performed.

3.4.1.3.1. Input Ports

rssWorldModelData: Global Cartesian world model data providing information on the local surrounding environment required to create the situation coordinate system pairs <ego-vehicle, object>. Requires local map data (i.e. lane segments and semantics on intersections and priority rules), ego vehicle and object (i.e. position, velocity and RSS dynamics) information.

3.4.1.3.2. Output Ports

rssSituations: A list of individual RSS Situations between the ego vehicle and each of the objects. Each situation is formulated within its own lane-based coordinate system. **EgoVehicle and Objects:** i.e. (relative) position, velocity, priority flag, situation specific RSS acceleration values.

3.4.1.4. RssSituationChecking

The RssSituationChecking entity performs the RSS check on all incoming individual RssSituations and creates the required RssResponseStates if dangerous situations are detected.

3.4.1.4.1. Input Ports

rssSituations: The RssSituations as provided by the RssSituationExtraction

3.4.1.4.2. Output Ports

rssResponseStates: A list of RSS response states in respect to the individual RSS Situations

3.4.1.5. RssResponseResolving

The RssResponseResolving entity handles conflicts of the incoming RssResponseStates. It combines the individual <ego-vehicle, object> situation specific response states into one single overall RssResponseState.

3.4.1.5.1. Input Ports

rssResponseStates: The list of RSS response states as provided by the RssSituationChecking

3.4.1.5.2. Output Ports

rssResponseState: Resulting combined overall RssResponseState.

3.4.1.6. RssResponseTransformation

The RssResponseTransformation entity transforms the overall RssResponseState back into the global Cartesian world. This results in RSS restrictions for the actuator commands.

3.4.1.6.1. Input Ports

rssWorldModelData: Global Cartesian world model data providing information on the local surrounding environment required to transform the RSS response state back into acceleration restrictions in the Catesian space. Requires local map data (i.e. lane segments of the ego vehicle)

and ego vehicle (position, velocity and RSS dynamics) information.

rssResponseState: Resulting combined overall RssResponseState provided by the RssResponseResolving.

3.4.1.6.2. Output Ports

rssAccelerationRestrictions: The resulting restrictions of the actuator control command

Chapter 4. RssModule: Software Architecture

This section describes the software architecture of the RSSModule in detail. The static view covers the deployed artifacts, the interfaces and data types of the module. The dynamic view describes the behaviour of the module in more detail.

4.1. Static View

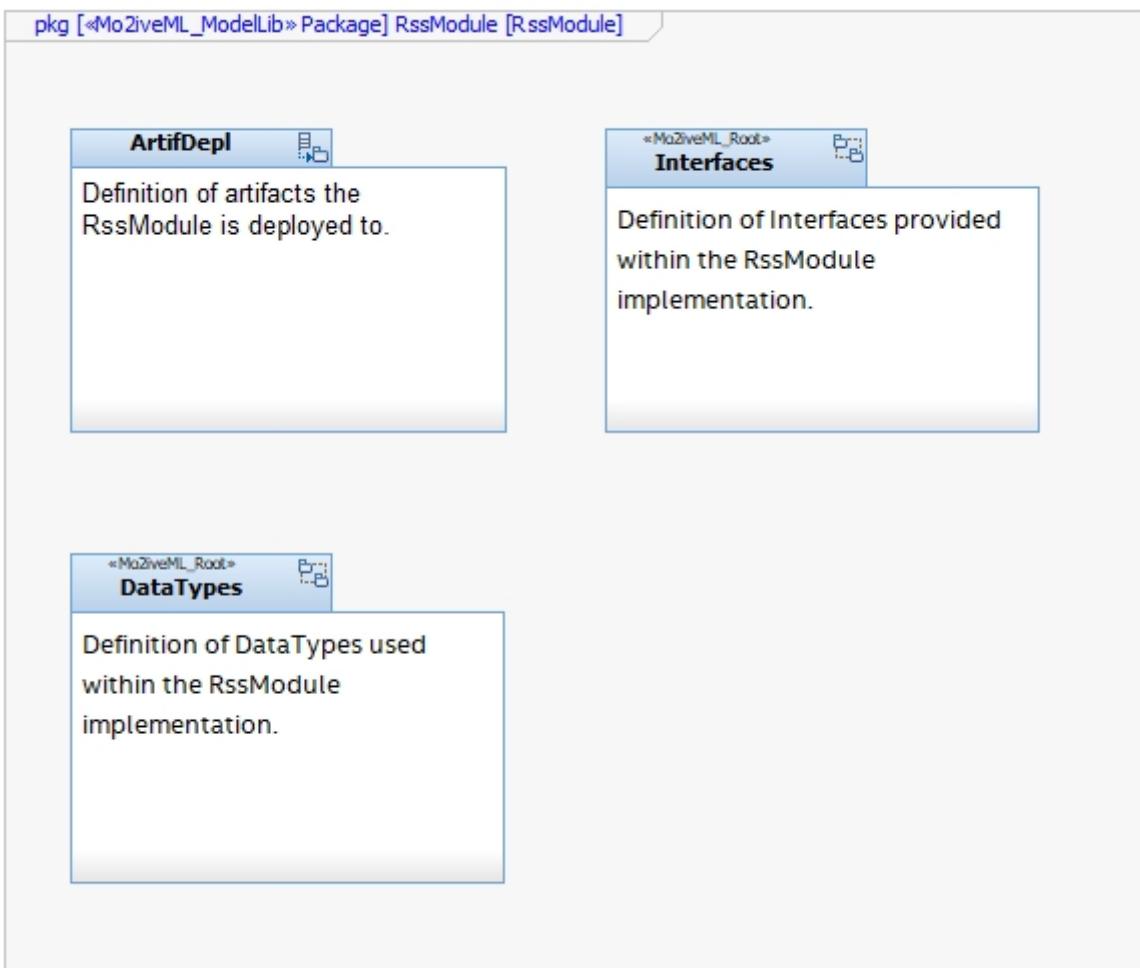


Figure 16. The packages within the RssModule.

The RssModule is implemented in C++. Based on a set of datatype definitions the interfaces are provided to:

1. prepare the RSS situations between individual pairs of ego vehicle and objects
2. trigger the situation specific RSS calculations
3. resolve the situation specific responses to one global response
4. translate the global response into concrete restrictions of the actuator control commands

Finally, the implementation of the RssModule is deployed within a set of artifacts.

4.1.1. Artifact Deployment

Definition of artifacts the RssModule is deployed to.



Figure 17. Artifacts to be deployed from the RssModule.

4.1.1.1. rss_module

This library contains the RssModule implementation.

4.1.1.2. rss_support

This library contains functionality supporting the integration of the RssModule into a system.

4.1.2. Interfaces

Definition of Interfaces provided within the RssModule implementation.

The RssModule is implemented as a whole within the namespace rss.

4.1.2.1. Namespace rss::core

Namespace for RSS core interfaces and operations.

This contains the interfaces including the operations the RssModule provides to the user to perform the RSS functionality.

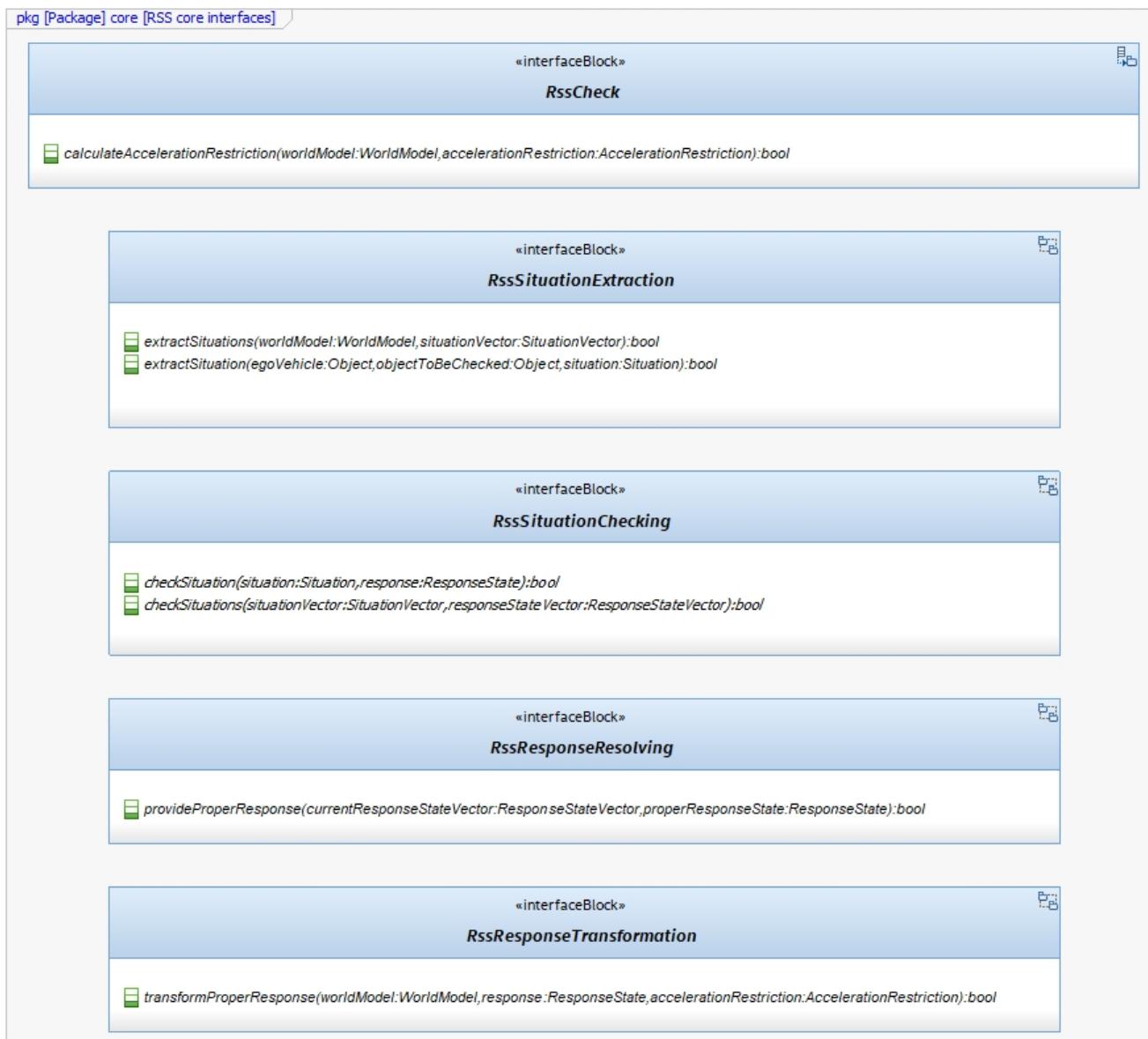


Figure 18. The interfaces provided by the RssModule.

4.1.2.1.1. RssCheck

Class providing the functionality of the RSS check sequence at once with the RSS world model as input and restrictions of the acceleration for the actuator control as output. This class internally makes use of the RssSituationExtraction, RssSituationChecking, RssResponseResolving and RssResponseTransformation functionality.

calculateAccelerationRestriction

Perform the calculation of the RSS checks on the current world model and provide the acceleration restriction for the actuator control.

Returns true if the checks could be calculated, false if there was an error during the operation.

returns	bool	Out	False on failure.
worldModel	WorldModel	In	The current world model information.

4.1. Static View

accelerationRestriction	AccelerationRestriction	Out	The restrictions on the vehicle acceleration to become RSS safe.
-------------------------	-------------------------	-----	--

4.1.2.1.2. RssSituationExtraction

Class providing functions required for the extraction of the RSS situations from the RSS world model.

extractSituations

Extract all RSS situation pairs pair <ego-vehicle, object> to be checked from the world model.

Returns true if the situations could be created, false if there was an error during the operation.

returns	bool	Out	False on failure.
worldModel	WorldModel	In	The current world model information.
situationVector	SituationVector	Out	The vector of situations to be analyzed with RSS.

extractSituation

Extract the RSS situation pair <ego-vehicle, object> of the ego vehicle and the object to be checked.

Returns true if the situation could be created, false if there was an error during the operation.

returns	bool	Out	False on failure.
egoVehicle	Object	In	The information on the ego vehicle object.
objectToBeChecked	Object	In	The information on the object to be checked.
situation	Situation	Out	The situation to be analyzed with RSS.

4.1.2.1.3. RssSituationChecking

Class providing functions required for the RSS checks of the situation.

checkSituations

Checks if the current situation pairs <ego-vehicle, object> are safe.

Returns true if the situations could be analyzed, false if an error occurred during evaluation.

returns	bool	Out	False on failure.
---------	------	-----	-------------------

situationVector	SituationVector	In	The vector of situations that should be analyzed.
responseStateVector	ResponseStateVector	Out	The vector of response states for the current situations.

checkSituation

Checks if the current situation pair <ego-vehicle, object> is safe.

Returns true if the situation could be analyzed, false if an error occurred during evaluation.

returns	bool	Out	False on failure.
situation	Situation	In	The situation that should be analyzed.
response	ResponseState	Out	The response state for the current situation.

4.1.2.1.4. RssResponseResolving

Class to resolve the responseStateVector of the different situation specific responses into a single responseState. This class tracks the RSS response state of every situation id over time and especially stores the respective response state before the danger threshold time. This requires that the id of a RSS situation remains constant over time in case it refers to the same object; otherwise tracking over time will fail.

provideProperResponse

Calculate the proper response from of the current situation pair <ego-vehicle, object> response states. It combines all response states into one single overall RssResponse.

Returns true if the proper response state could be calculated, false otherwise.

returns	bool	Out	False on failure.
currentResponseStateVector	ResponseStateVector	In	Vector with all the responses gather for the current individual situations
properResponseState	ResponseState	Out	The proper overall response state.

4.1.2.1.5. RssResponseTransformation

Class providing functions required to transform the proper response into restrictions of the acceleration for the actuator control.

transformProperResponse

Transform the proper response into restrictions of the acceleration for the actuator control. Since

the RssResponseResolving entity is acting within the situation coordinate system, it is not able to decide on the actual lateral movement of the ego-vehicle within its lane. Within this function the required world model data is available to decide if a desired lateral response can be resolved either by applying a restriction on the lateral acceleration or, in addition, requires a restriction of the longitudinal acceleration.

Returns true if the acceleration restrictions could be calculated, false otherwise.

returns	bool	Out	False on failure.
worldModel	WorldModel	In	The current world model information.
response	ResponseState	In	The proper overall response to be transformed.
accelerationRestriction	AccelerationRestriction	Out	The restrictions on the vehicle acceleration to become RSS safe.

4.1.3. DataTypes

Definition of DataTypes used within the RssModule implementation.

The RssModule is implemented as a whole within the namespace rss.

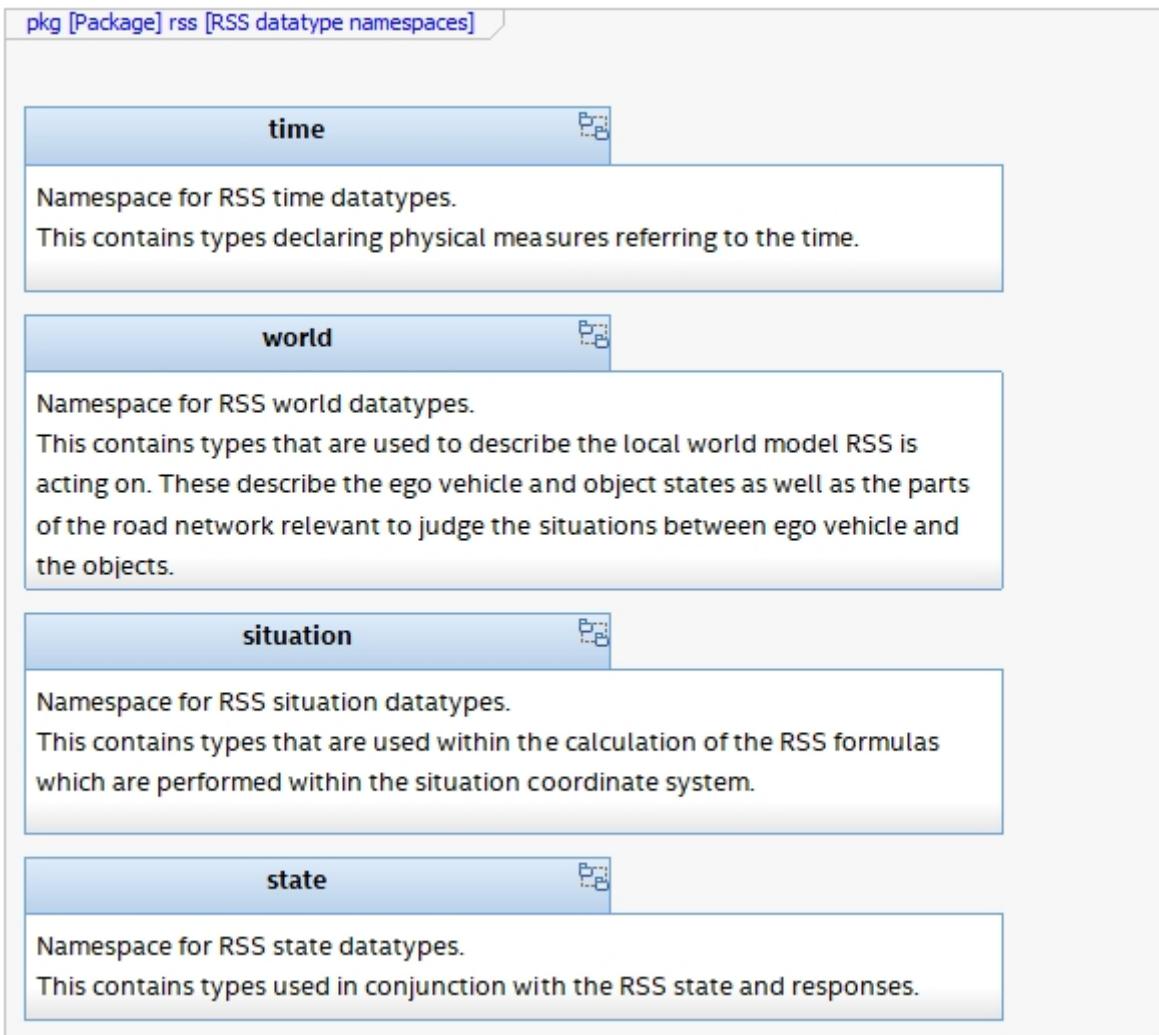


Figure 19. The RSS datatypes are organized within several sub-namespaces.

4.1.3.1. Namespace rss::time

Namespace for RSS time datatypes.

This contains types declaring physical measures referring to the time.

4.1.3.1.1. Duration (Typedef)

A duration represents a time interval

Unit: second

unit	Second
dimension	Time
float64_t	[1]

4.1.3.1.2. TimeIndex (Typedef)

Defines a certain point in time.

uint64_t	[1]
----------	-------

4.1.3.2. Namespace rss::world

Namespace for RSS world datatypes.

This contains types that are used to describe the local world model RSS is acting on. These describe the ego vehicle and object states as well as the parts of the road network relevant to judge the situations between ego vehicle and the objects.

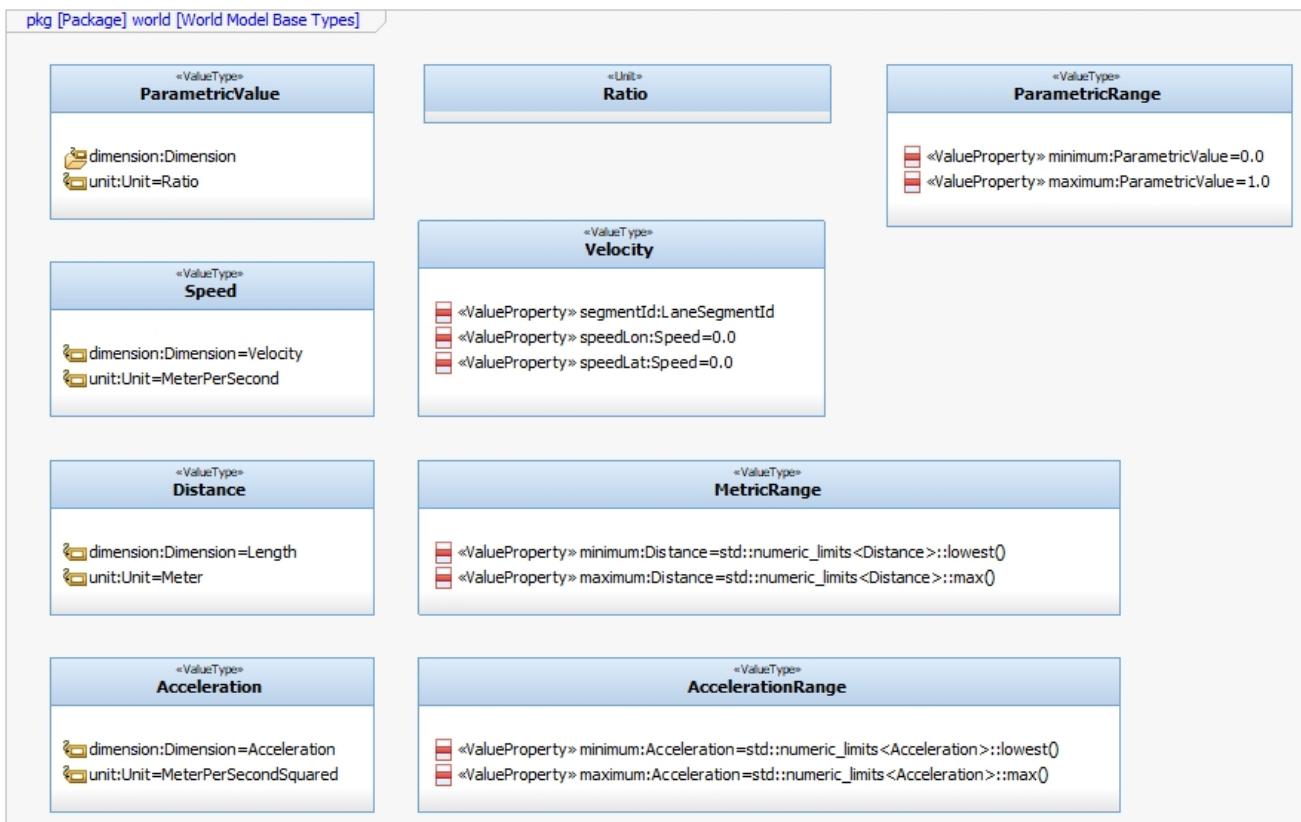


Figure 20. The base types used within the local RSS world model.

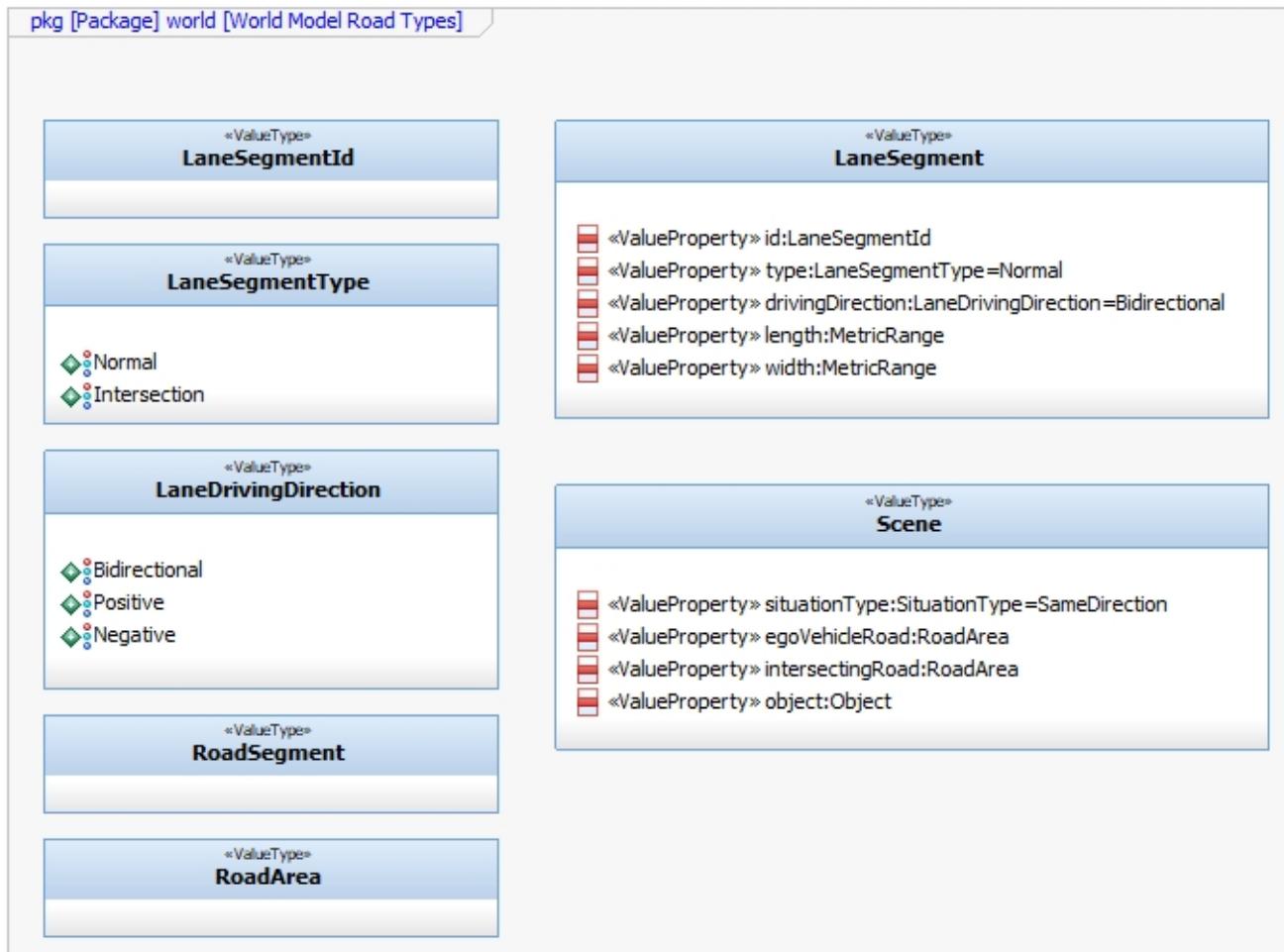


Figure 21. The types describing the road and lane geometries used within the local RSS world model.

4.1. Static View

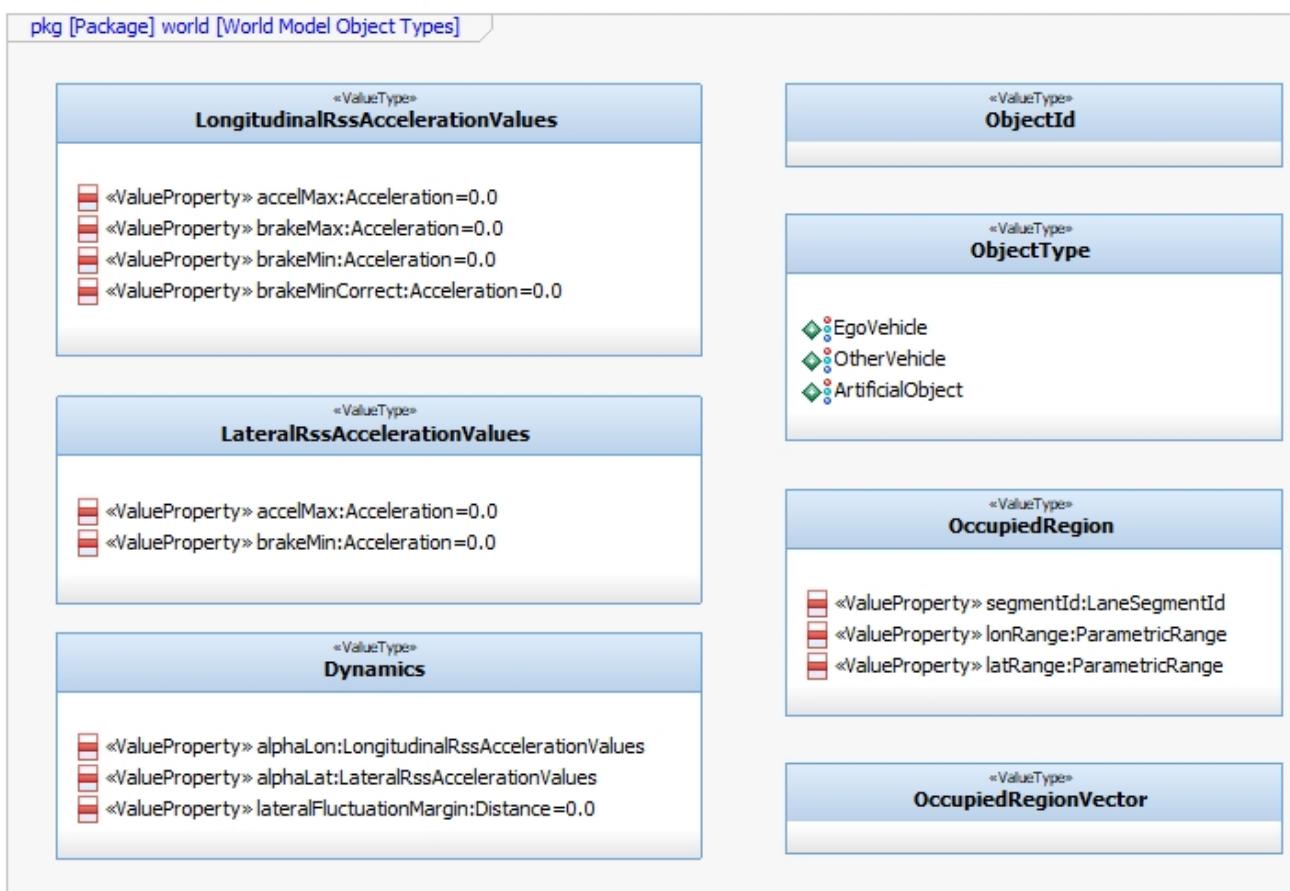


Figure 22. The types describing the objects used within the local RSS world model.

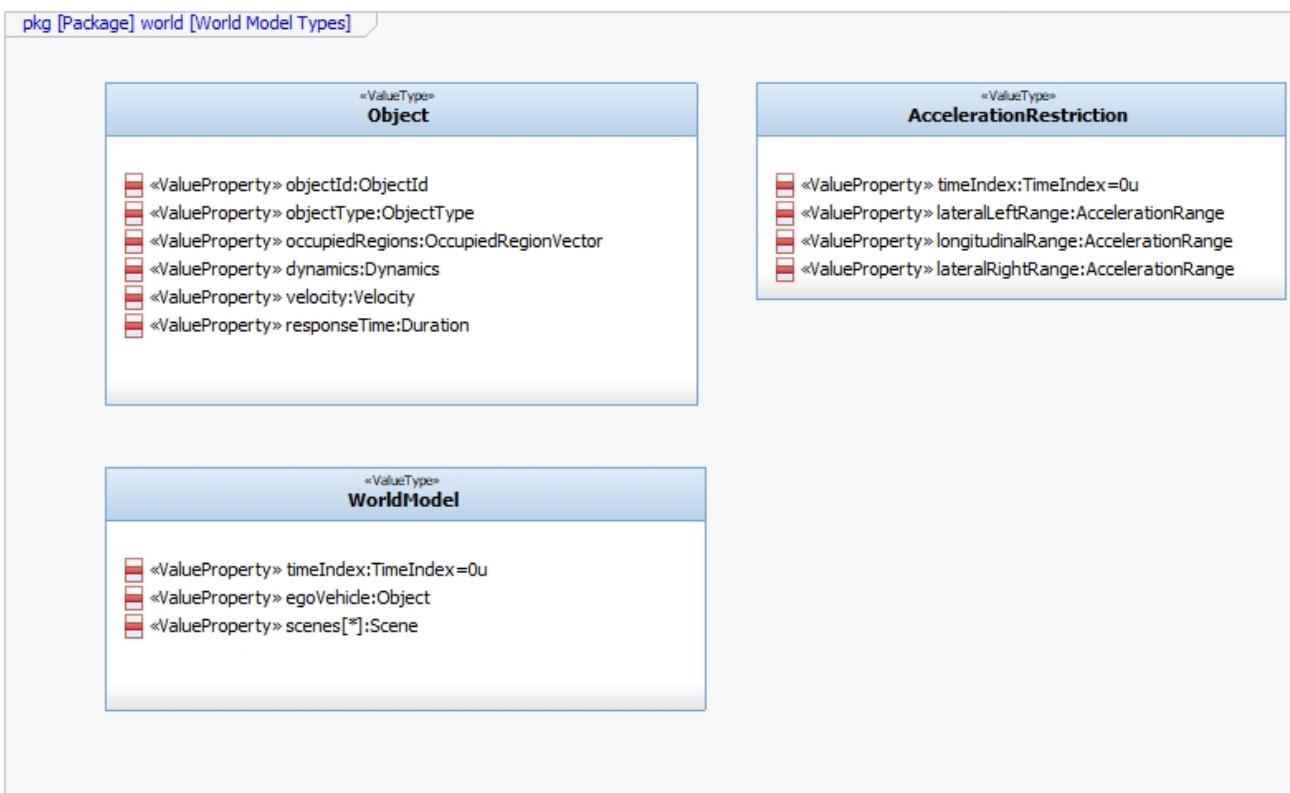


Figure 23. The types describing the high level world model and object used within the local RSS world model.

4.1.3.2.1. Units

Ratio	Defines a unit which is a dimensionless ratio.
-------	--

4.1.3.2.2. ParametricValue (Typedef)

A parametric value in the range of [0.0; 1.0] describing the relative progress.

unit	Ratio
float64_t	[1]

4.1.3.2.3. Distance (Typedef)

The length of a specific path traveled between two points.

Unit: meter

dimension	Length
unit	Meter
float64_t	[1]

4.1.3.2.4. Speed (Typedef)

The rate of change of an object's position with respect to time. The speed of an object is the magnitude of its velocity.

Unit: meter per second

dimension	Velocity
unit	MeterPerSecond
float64_t	[1]

4.1.3.2.5. Acceleration (Typedef)

The rate of change of Speed of an object with respect to time.

Unit: meter per second squared

dimension	Acceleration
unit	MeterPerSecondSquared
float64_t	[1]

4.1.3.2.6. ParametricRange (Structure)

A parametric range within a lane segment described by its borders: [minimum, maximum].

minimum	ParametricValue	The minimum value of the parametric range.
---------	-----------------	--

4.1. Static View

maximum	ParametricValue	The maximum value of the parametric range.
---------	-----------------	--

4.1.3.2.7. MetricRange (Structure)

A metric range described by its borders: [minimum, maximum].

minimum	Distance	The minimum value of the metric range.
maximum	Distance	The maximum value of the metric range.

4.1.3.2.8. AccelerationRange (Structure)

An acceleration range described by its borders: [minimum, maximum].

minimum	Acceleration	The minimum value of the acceleration range.
maximum	Acceleration	The maximum value of the acceleration range.

4.1.3.2.9. Velocity (Structure)

Defines the velocity of an object within its current lane. The velocity consists of a longitudinal and a lateral component.

segmentId	LaneSegmentId	The id of the lane segment this velocity refers to.
speedLon	Speed	The longitudinal speed component of the velocity vector. The longitudinal component of the velocity is always measured tangential to the center line of the current lane.
speedLat	Speed	The lateral speed component of the velocity vector. The lateral component of the velocity is always measured orthogonal to the center line of the current lane.

4.1.3.2.10. LaneSegmentId (Typedef)

Defines the unique id of a lane segment.

uint64_t	[1]
----------	-------

4.1.3.2.11. LaneSegmentType (Enumeration)

Normal	0	Normal lane segment. Nothing special to consider.
Intersection	1	Lane segment is intersecting with another lane segment of the intersecting road.

4.1.3.2.12. LaneDrivingDirection (Enumeration)

Bidirectional	0	Traffic flow in this lane segment is in both directions.
Positive	1	Nominal traffic flow in this lane segment is positive RoadArea direction.
Negative	2	Nominal traffic flow in this lane segment is negative RoadArea direction.

4.1.3.2.13. LaneSegment (Structure)

Defines a lane segment.

id	LaneSegmentId	The id of the lane segment.
type	LaneSegmentType	The type of this lane segment in context of the RssArea it belongs to.
drivingDirection	LaneDrivingDirection	The nominal direction of the traffic flow of this lane segment in context of the RssArea it belongs to.
length	MetricRange	The metric range of the lane segments length.
width	MetricRange	The metric range of the lane segments width.

4.1.3.2.14. RoadSegment (Typedef)

A RoadSegment is defined by lateral neighboring lane segments. The lane segments within a road segment have to be ordered from right to left in respect to the driving direction defined by the road area.

rss::world::LaneSegment	[*]
-------------------------	-------

4.1.3.2.15. RoadArea (Typedef)

A RoadArea is defined by longitudinal neighboring road segments. The road segments within a road area have to be ordered from start to end in respect to the driving direction.

rss::world::RoadSegment

[*]

4.1.3.2.16. Scene (Structure)

A Scene defines the relation between the ego vehicle and another object. It consists of the type of situation between these two and the corresponding road areas of interest. All lane segments on the route between ego vehicle and the object have to be part of this. The RssModule has to be able to calculate minimum and maximum distances between ego vehicle and object as well as accelerated movements within this area.

situationType	SituationType	The type of the current situation. Depending on this type the other fields of the RssArea might be left empty.
egoVehicleRoad	RoadArea	The RssRoadArea the ego vehicle is driving in. The driving direction of the ego vehicle define the ordering of the road segments. In non-intersection situations the object is also driving in this road area.
intersectingRoad	RoadArea	The RssRoadArea an intersecting vehicle is driving in. The driving direction of the intersecting vehicle define the ordering of the road segments. The road area should contain all neighboring lanes the other vehicle is able to drive in. In non-intersection situations this road area is empty.
object	Object	The object this scene refers to.

4.1.3.2.17. ObjectId (Typedef)

Defines the unique id of an object.

uint64_t

[1]

4.1.3.2.18. ObjectType (Enumeration)

Enumeration describing the types of object.

EgoVehicle	0	The object is the ego vehicle.
OtherVehicle	1	The object is some other real vehicle.
ArtificialObject	2	The object is an artificial one.

4.1.3.2.19. OccupiedRegion (Structure)

Describes the region that an object covers within a lane segment.

An object on a lane is described by the parametric range it spans in each of the two lane segment directions.

segmentId	LaneSegmentId	The id of the lane segment this region refers to.
lonRange	ParametricRange	The parametric range an object spans in longitudinal direction within a lane segment.
latRange	ParametricRange	The parametric range an object spans in lateral direction within a lane segment.

4.1.3.2.20. OccupiedRegionVector (Typedef)

A vector of occupied regions.

rss::world::OccupiedRegion	[*]
----------------------------	-------

4.1.3.2.21. LongitudinalRssAccelerationValues (Structure)

Collection of the RSS acceleration values in longitudinal direction.

accelMax	Acceleration	Absolute amount of the maximum allowed acceleration. This value has always to be positive, zero is allowed.
brakeMax	Acceleration	Absolute amount of the maximum allowed braking deceleration. This value has always to be positive and not smaller than brakeMin.
brakeMin	Acceleration	Absolute amount of the minimum allowed breaking deceleration. This value has always to be positive and not smaller than brakeMinCorrect.
brakeMinCorrect	Acceleration	Absolute amount of the minimum allowed breaking deceleration when driving on the correct lane. This value has always to be positive.

4.1.3.2.22. LateralRssAccelerationValues (Structure)

Collection of the RSS acceleration values in lateral direction.

accelMax	Acceleration	Absolute amount of the maximum allowed acceleration. This value has always to be positive, zero is allowed.
brakeMin	Acceleration	Absolute amount of the minimum allowed breaking deceleration. This value has always to be positive.

4.1.3.2.23. Dynamics (Structure)

Describes the RSS dynamics values to be applied for an object within the metric world frame. The dynamics consist of a longitudinal component, a lateral component and a lateral fluctuation margin to be taken into account to compensate for lateral fluctuations.

alphaLon	LongitudinalRssAccelerationValues	RSS dynamics values along longitudinal coordinate system axis.
alphaLat	LateralRssAccelerationValues	RSS dynamics values along lateral coordinate system axis.
lateralFluctuationMargin	Distance	Defines the lateral fluctuation margin to be taken into account.

4.1.3.2.24. Object (Structure)

An object is described by several aspects: the unique id of an object, the type of the object, the lane regions the object occupies, the objects velocity within its lane and finally the area of interaction of ego vehicle and the object.

objectId	ObjectId	Defines the unique id of an object. This id has to be constant over time for the same object.
objectType	ObjectType	Defines the type of the object.
occupiedRegions	OccupiedRegionVector	Defines the lane regions the object occupies.
dynamics	Dynamics	Defines the objects dynamics to be applied. This parameters are provided on a per object basis to be able to adapt these e.g. in respect to the weather conditions.
velocity	Velocity	Defines the objects velocity in respect to its current major lane.
responseTime	Duration	Defines the response time of the object within the current scene.

4.1.3.2.25. WorldModel (Structure)

The world model, RSS requires as input, consists of the egoVehicle and object description as well as the list of relevant lane segments.

timeIndex	TimeIndex	The time index is required to distinguish different points in time when tracking states or transforming responses back. Each world model referring to another point in time should get another time index. The time index of the world model must not be zero.
egoVehicle	Object	The ego vehicle.
scenes	Scene	All scenes

4.1.3.2.26. AccelerationRestriction (Structure)

Defines restrictions of the vehicle acceleration.

timeIndex	TimeIndex	The time index these acceleration restrictions are referring to.
lateralLeftRange	AccelerationRange	The range of the acceleration restriction in lateral left direction.
longitudinalRange	AccelerationRange	The range of the acceleration restriction in longitudinal direction.
lateralRightRange	AccelerationRange	The range of the acceleration restriction in lateral right direction.

4.1.3.3. Namespace rss::situation

Namespace for RSS situation datatypes.

This contains types that are used within the calculation of the RSS formulas which are performed within the situation coordinate system.

4.1. Static View

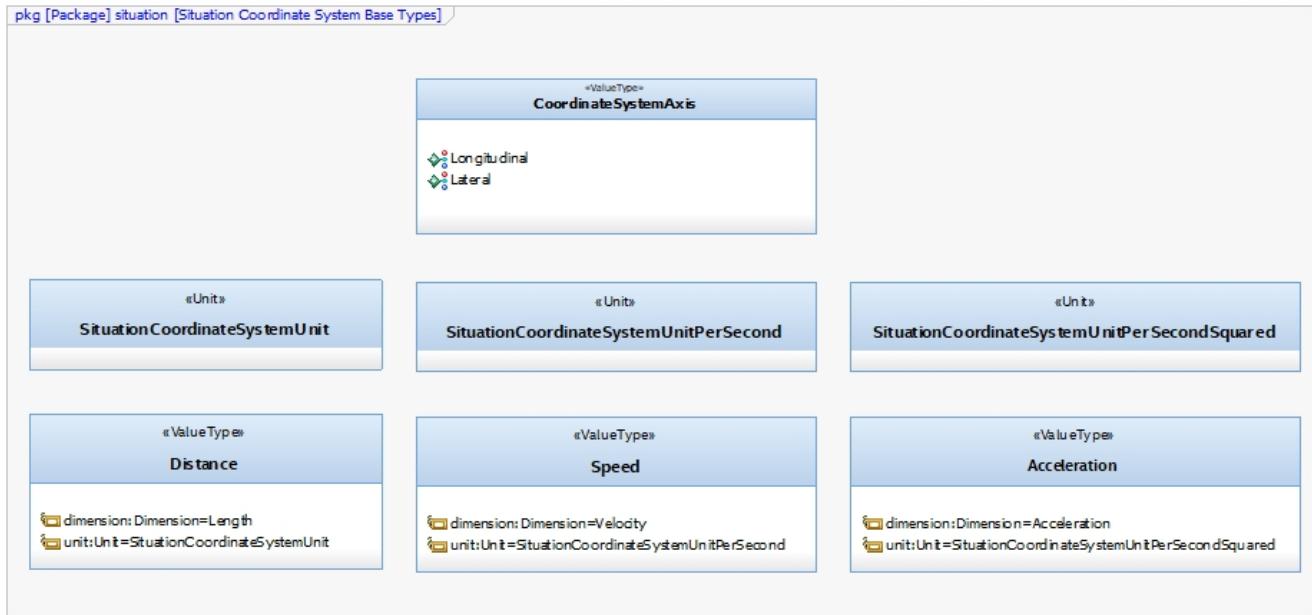


Figure 24. The base types used within the situation coordinate system.

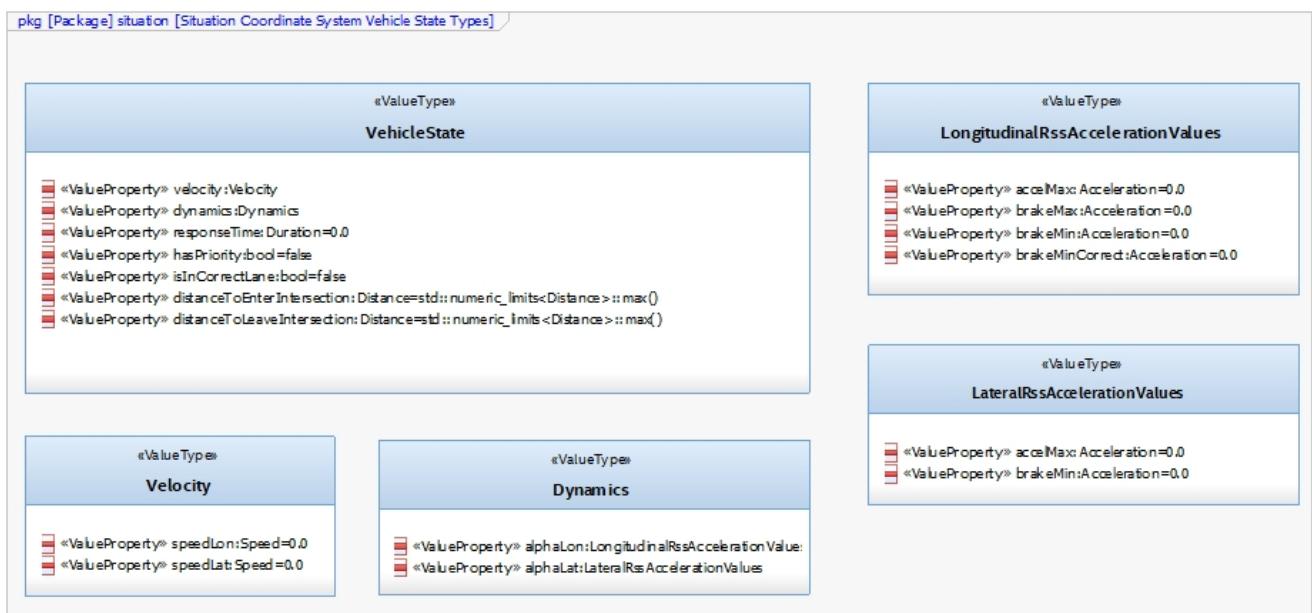


Figure 25. The types describing the vehicle state used within the situation coordinate system.

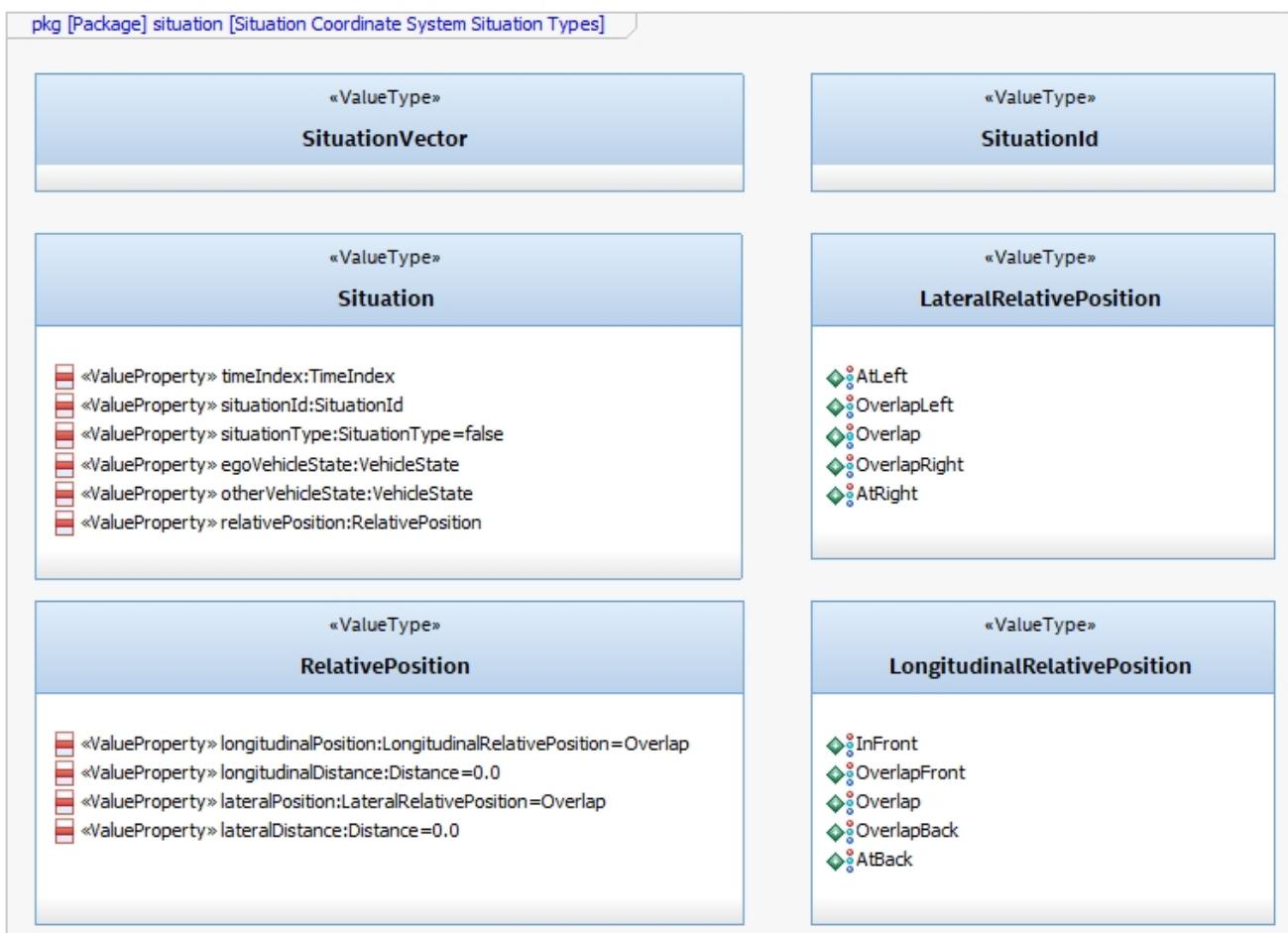


Figure 26. The types describing a situation used within the situation coordinate system.

4.1.3.3.1. Units

SituationCoordinateSystemUnitPerSecond	Defines a unit in the situation coordinate system per second.
SituationCoordinateSystemUnitPerSecondSquared	Defines a unit in the situation coordinate system per second squared.
SituationCoordinateSystemUnit	Defines a unit in the situation coordinate system. The situation coordinate system has two dimensions:- longitudinal axis- lateral axis

4.1.3.3.2. CoordinateSystemAxis (Enumeration)

Enumeration defining the axis of the situation coordinate system.

Longitudinal	0	longitudinal axis: ego vehicle is driving in positive direction
Lateral	1	lateral axis: ego vehicle left to right defines the positive direction

4.1.3.3.3. Acceleration (Typedef)

The rate of change of Speed of an object with respect to time.

Unit: situation coordinate system unit per second squared

dimension	Acceleration
unit	SituationCoordinateSystemUnitPerSecondSquared
float64_t	[1]

4.1.3.3.4. Distance (Typedef)

The length of a specific path traveled between two points.

Unit: situation coordinate system unit

dimension	Length
unit	SituationCoordinateSystemUnit
float64_t	[1]

4.1.3.3.5. Speed (Typedef)

The rate of change of an object's Distance with respect to time. The speed of an object is the magnitude of its velocity.

Unit: situation coordinate system units per second

dimension	Velocity
unit	SituationCoordinateSystemUnitPerSecond
float64_t	[1]

4.1.3.3.6. Velocity (Structure)

Defines the velocity of an object in the respective situation coordinate system. The velocity consists of a longitudinal and a lateral component.

speedLon	Speed	Absolute amount of the longitudinal speed component of the velocity vector. This value has always to be positive.
speedLat	Speed	The lateral speed component of the velocity vector.

4.1.3.3.7. LongitudinalRssAccelerationValues (Structure)

Collection of the RSS acceleration values in longitudinal direction.

accelMax	Acceleration	Absolute amount of the maximum allowed acceleration. This value has always to be positive, zero is allowed.
----------	--------------	---

brakeMax	Acceleration	Absolute amount of the maximum allowed braking deceleration. This value has always to be positive and not smaller than brakeMin.
brakeMin	Acceleration	Absolute amount of the minimum allowed breaking deceleration. This value has always to be positive and not smaller than brakeMinCorrect.
brakeMinCorrect	Acceleration	Absolute amount of the minimum allowed breaking deceleration when driving on the correct lane. This value has always to be positive.

4.1.3.3.8. LateralRssAccelerationValues (Structure)

Collection of the RSS acceleration values in lateral direction.

accelMax	Acceleration	Absolute amount of the maximum allowed acceleration. This value has always to be positive, zero is allowed.
brakeMin	Acceleration	Absolute amount of the minimum allowed breaking deceleration. This value has always to be positive.

4.1.3.3.9. Dynamics (Structure)

Describes the RSS dynamics values to be applied for an object within the respective situation coordinate system. The dynamics consist of a longitudinal and a lateral component.

alphaLon	LongitudinalRssAccelerationValues	RSS dynamics values along longitudinal coordinate system axis
alphaLat	LateralRssAccelerationValues	RSS dynamics values along lateral coordinate system axis

4.1.3.3.10. VehicleState (Structure)

The state of an object in a RSS situation.

The state consists of the following components in respect to the situation coordinate system: the velocity, the distance to the intersection (if applicable), the dynamics, the response time, a Right-of-Way priority flag as well as a flag stating if the vehicle is driving in its correct lane.

velocity	Velocity	The situation specific velocity.
----------	----------	----------------------------------

dynamics	Dynamics	The situation specific dynamics.
responseTime	Duration	The situation specific response time.
hasPriority	bool	Flag indicating if the situation specific Right-of-Way relation.
isInCorrectLane	bool	Flag indicating if the vehicle driving in the correct lane
distanceToEnterIntersection	Distance	The minimum distance to be covered by the vehicle to enter the intersection.
distanceToLeaveIntersection	Distance	The maximum distance to cover by the vehicle to leave the intersection completely.

4.1.3.3.11. SituationId (Typedef)

The unique id of an situation over time.

uint64_t	[1]
----------	-------

4.1.3.3.12. SituationType (Enumeration)

Enumeration describing the type of situation.

NotRelevant	0	The other vehicle cannot conflict with the ego vehicle. This kind of situations are always considered to be safe. Use this situation state to make the object visible in the result vector to be a known object, but not relevant for RSS (e.g. object in opposite direction, but already passed by).
SameDirection	1	Both drive on the same road in the same direction.
OppositeDirection	2	Both drive on the same road in the opposite direction.
IntersectionEgoHasPriority	3	Both drive on individual roads which intersect at the end. Ego vehicle has priority over object.
IntersectionObjectHasPriority	4	Both drive on individual roads which intersect at the end. Object has priority over ego vehicle.

IntersectionSamePriority	5	Both drive on individual roads which intersect at the end. Object and ego vehicle have same priority.
--------------------------	---	---

4.1.3.3.13. LateralRelativePosition (Enumeration)

Enumeration describing the relative lateral position between two objects, a and b, within their situation coordinate system.

AtLeft	0	The object a is completely left of object b. This means there is an actual lateral space between them.
OverlapLeft	1	The objects overlap. The left border of object a is left of the left border of object b AND the right border of object a is left of the right border of object b.
Overlap	2	The objects overlap, but neither the conditions for OverlapLeft nor OverlapRight are applicable.
OverlapRight	3	The objects overlap. The left border of object a is right of the left border of object b AND the right border of object a is right of the right border of object b.
AtRight	4	The object a is completely right of object b. This means there is an actual lateral space between them.

4.1.3.3.14. LongitudinalRelativePosition (Enumeration)

Enumeration describing the relative longitudinal position between two objects, a and b, within their situation coordinate system.

InFront	0	The object a is completely in front of object b. This means there is an actual longitudinal space between them.
OverlapFront	1	The objects overlap. The front border of object a is in front of the front border of object b AND the back border of object a is in front of the back border of object b.

4.1. Static View

Overlap	2	The objects overlap, but neither the conditions for OverlapFront nor OverlapBack are applicable.
OverlapBack	3	The objects overlap. The front border of object a is at back of the front border of object b AND the back border of object a is at back of the back border of object b.
AtBack	4	The object a is completely at back of object b. This means there is an actual longitudinal space between them.

4.1.3.3.15. RelativePosition (Structure)

Describes the relative position between two objects within their situation coordinate system.

longitudinalPosition	LongitudinalRelativePosition	The longitudinal relative position between two objects within their situation coordinate system.
longitudinalDistance	Distance	The longitudinal distance between the two objects within their situation coordinate system.
lateralPosition	LateralRelativePosition	The lateral relative position between two objects within their situation coordinate system.
lateralDistance	Distance	The lateral distance between the two objects within their situation coordinate system.

4.1.3.3.16. Situation (Structure)

Describes a RSS situation.

A situation always considers the relative relation between two objects: the ego vehicle and one other vehicle. The situation coordinate system is unique for one specific situation. As a consequence the vehicle state of the ego vehicle in different RSS situations cannot be compared to each other. Consists of a situation id and type, the VehicleState of the ego vehicle, the VehicleState of the other vehicle and the RelativePosition between ego vehicle and other vehicle.

timeIndex	TimeIndex	The time index is required to distinguish different points in time when tracking states or transforming responses back.
-----------	-----------	---

situationId	SituationId	The unique id of the situation. The situation id has to be constant over time for a pair of ego vehicle and specific other vehicle. E.g. might be filled with an id identifying the other vehicle unambiguously.
situationType	SituationType	The type of the current situation.
egoVehicleState	VehicleState	The vehicle state of the ego vehicle
otherVehicleState	VehicleState	The vehicle state of the other vehicle within the situation.
relativePosition	RelativePosition	The relative position between the ego vehicle and the other vehicle within this situation.

4.1.3.3.17. SituationVector (Typedef)

A vector of situations.

rss::situation::Situation	[*]
---------------------------	-------

4.1.3.4. Namespace rss::state

Namespace for RSS state datatypes.

This contains types used in conjunction with the RSS state and responses.

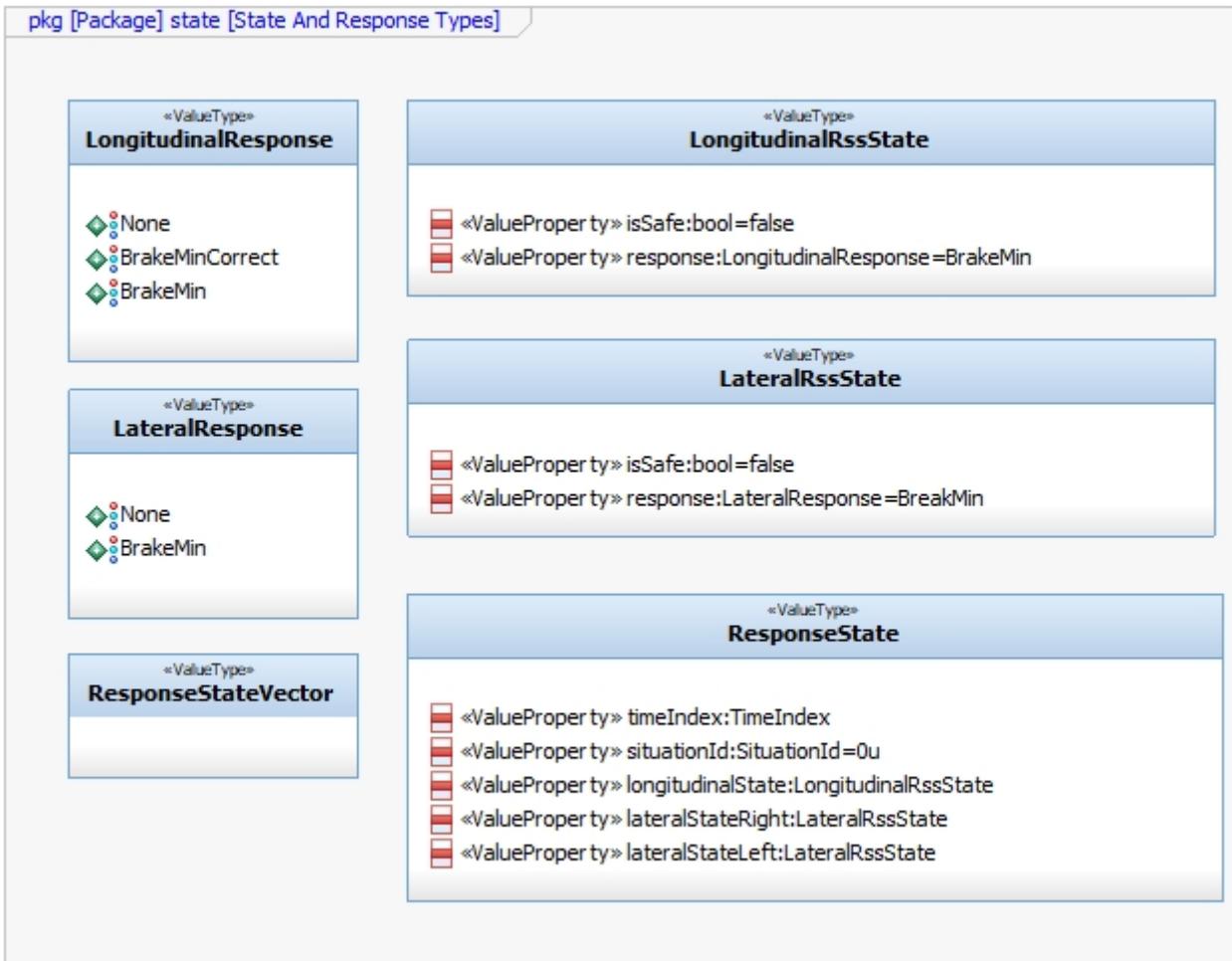


Figure 27. The types describing the RSS response and the RSS state.

4.1.3.4.1. LongitudinalResponse (Enumeration)

Enumeration defining the possible longitudinal responses.

Be aware: there has to be a strict order of the enumeration values according to the strictness of the response.

None	0	No action required.
BrakeMinCorrect	1	Vehicle has to decelerate at least with brake min correct longitudinally
BrakeMin	2	Vehicle has to decelerate at least with brake min longitudinally

4.1.3.4.2. LateralResponse (Enumeration)

Enumeration defining the possible lateral responses.

Be aware: there has to be a strict order of the enumeration values according to the strictness of the response.

None	0	No action required.
------	---	---------------------

BrakeMin	1	Vehicle has to decelerate at least with brake min laterally
----------	---	---

4.1.3.4.3. LongitudinalRssState (Structure)

Struct to store the longitudinal RSS state.

isSafe	bool	Flag to indicate if the state is longitudinal safe.
response	LongitudinalResponse	required response in longitudinal direction

4.1.3.4.4. LateralRssState (Structure)

Struct to store the lateral RSS state.

isSafe	bool	Flag to indicate if the state is lateral safe.
response	LateralResponse	required response in lateral direction

4.1.3.4.5. ResponseState (Structure)

Struct defining the RSS state of a single object.

timeIndex	TimeIndex	The time index is required to distinguish different points in time when tracking states or transforming responses back.
situationId	SituationId	Id of the situation this state refers to. The id has to remain unique over time representing the situation (ego-vehicle / object pair) under investigation. It is used to track the state of the ego-vehicle / object constellation i.e. at point of danger threshold time.
longitudinalState	LongitudinalRssState	The current longitudinal rss state.
lateralStateRight	LateralRssState	The current lateral rss state at right side in respect to ego-vehicle driving direction.
lateralStateLeft	LateralRssState	The current lateral rss state at left side in respect to ego-vehicle driving direction.

4.1.3.4.6. ResponseStateVector (Typedef)

A vector of response states.

rss::state::ResponseState	[*]
---------------------------	-------

4.2. Dynamic View

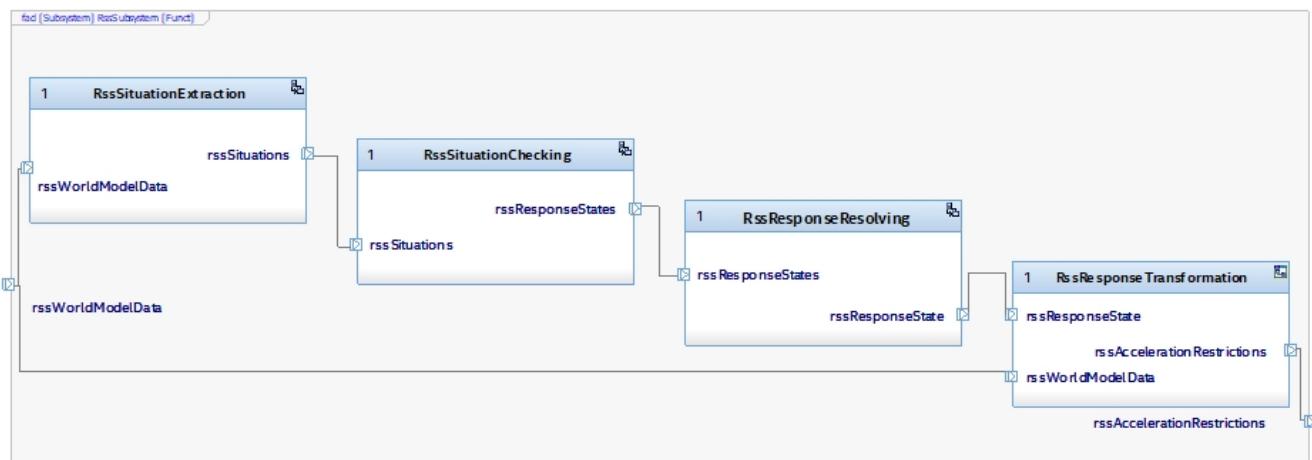


Figure 28. RSS internal processing steps to perform RSS checks and execute the RSS proper response

The RssSubsystem realizes the RSS part functionality. It implements the RSS checks based on the RssWorldModelData received from the SensorSubsystem:

1. Keep a safe distance from the car in front
2. Leave time and space for others in lateral maneuvers
3. Exhibit caution in occluded areas
4. Right-of-Way is given, not taken

In case a dangerous situation is detected a respective proper response is calculated and the actuator control commands received from the PlanningSubsystem are restricted accordingly to realize planning safety.

4.2.1. RssSituationExtractionImpl

RssSituationExtractionImpl describes the implementation of the RssSituationExtraction entity by defining a statechart.

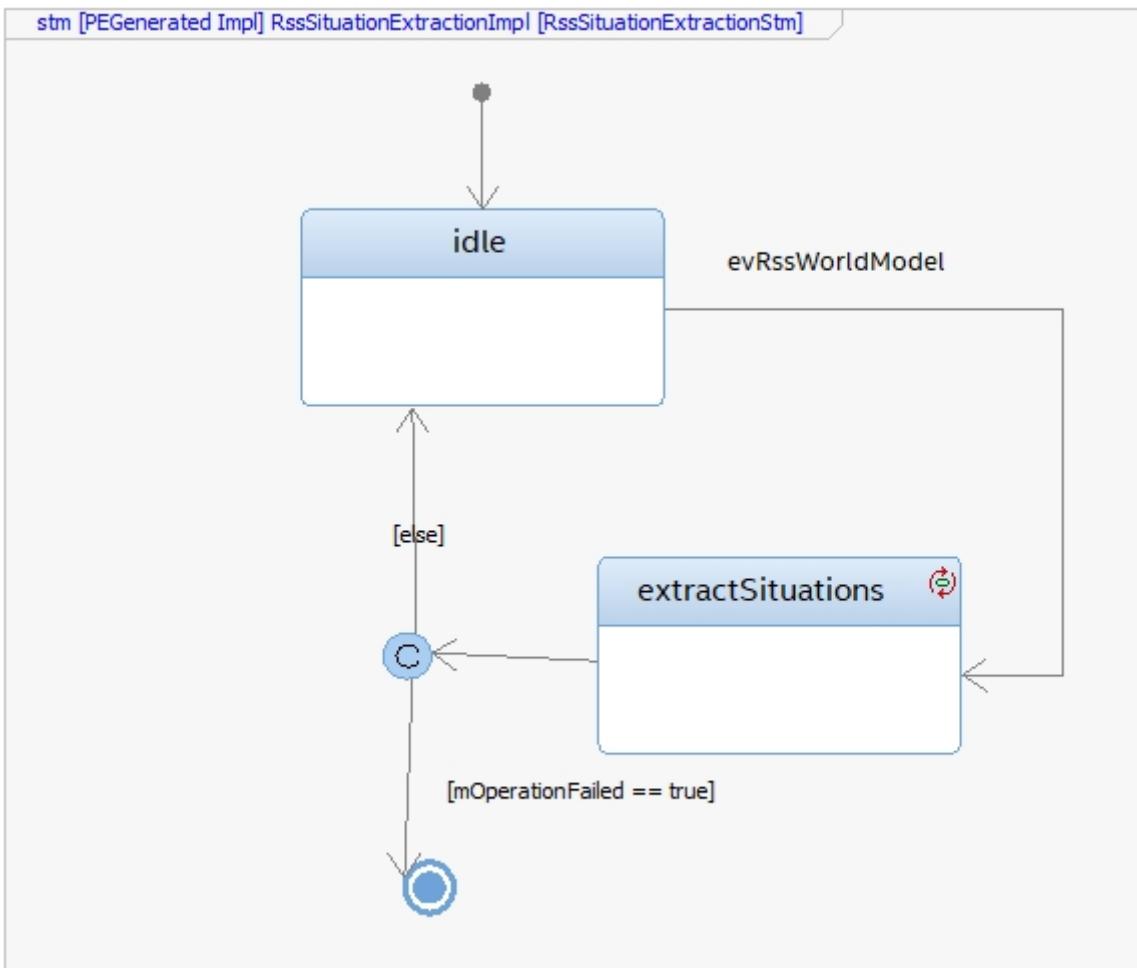


Figure 29. Statechart Diagram describing the dynamic behavior of the RssSituationExtraction entity.

The following table describes the events, triggers, states and their transitions of the statechart in detail.

Event/Trigger	Argument Type	Argument	Description
evRssWorldModel	WorldModel	worldModel	Event triggered when RssWorldModel data is received at the input.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.
→	evRssWorldModel	extractSituations	WorldModel data received.
extractSituations			All input data received. Perform operation to extract situations.
→		checkOperationResult	
error			Final error state.

4.2.2. RssSituationCheckingImpl

RssSituationCheckingImpl describes the implementation of the RssSituationChecking entity by defining a statechart.

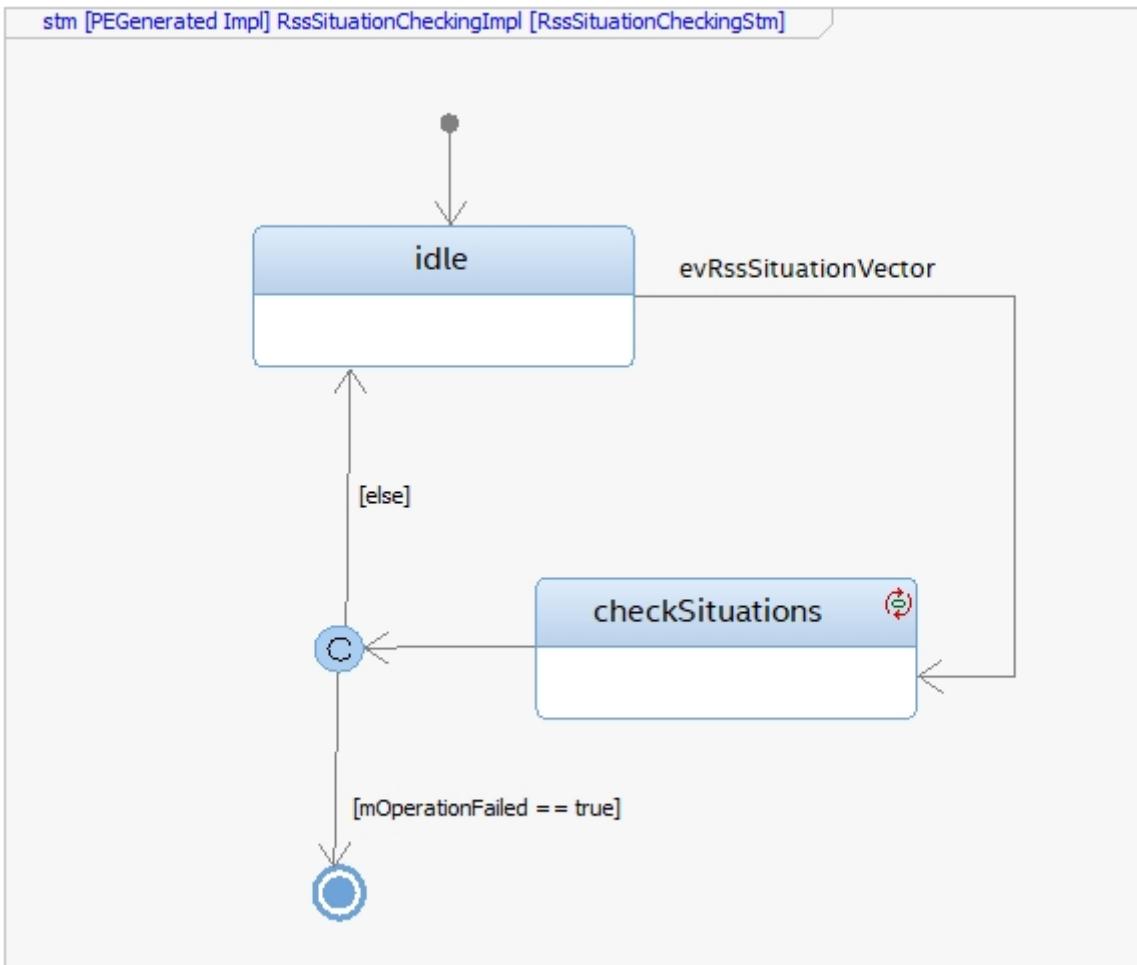


Figure 30. Statechart Diagram describing the dynamic behavior of the RssSituationChecking entity.

The following table describes the events, triggers, states and their transitions of the statechart in detail.

Event/Trigger	Argument Type	Argument	Description
evRssSituationVector	SituationVector	situationVector	Event triggered when RssSituationVector data is received at the input.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.
→	evRssSituationVector	checkSituations	SituationVector data received.
checkSituations			All input data received. Perform operation to check situations.

State	Transition	Target State	Description
→		checkOperationResult	
error			Final error state.

4.2.3. RssResponseResolvingImpl

RssResponseResolvingImpl describes the implementation of the RssResponseResolving entity by defining a statechart.

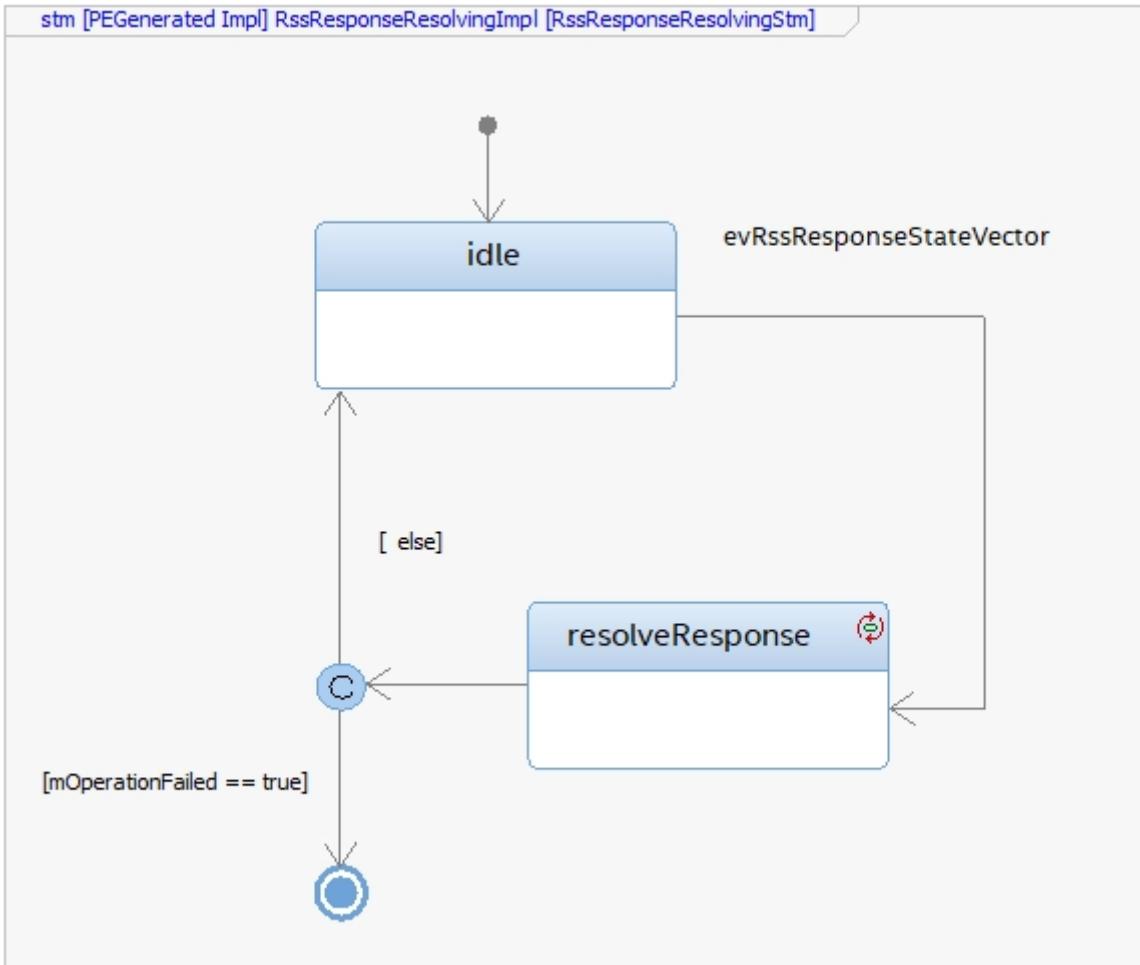


Figure 31. Statechart Diagram describing the dynamic behavior of the RssResponseResolving entity.

The following table describes the events, triggers, states and their transitions of the statechart in detail.

Event/Trigger	Argument Type	Argument	Description
evRssResponseStateVector	ResponseTypeVector	responseStateVector	Event triggered when RssResponseStateVector data is received at the input.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.

State	Transition	Target State	Description
→	evRssResponseStateVector	resolveResponse	ResponseStatusVector data received.
resolveResponse			All input data received. Perform operation to resolve responses.
→		checkOperationResult	
error			Final error state.

4.2.4. RssResponseTransformationImpl

RssResponseTransformationImpl describes the implementation of the RssResponseTransformation entity by defining a statechart.

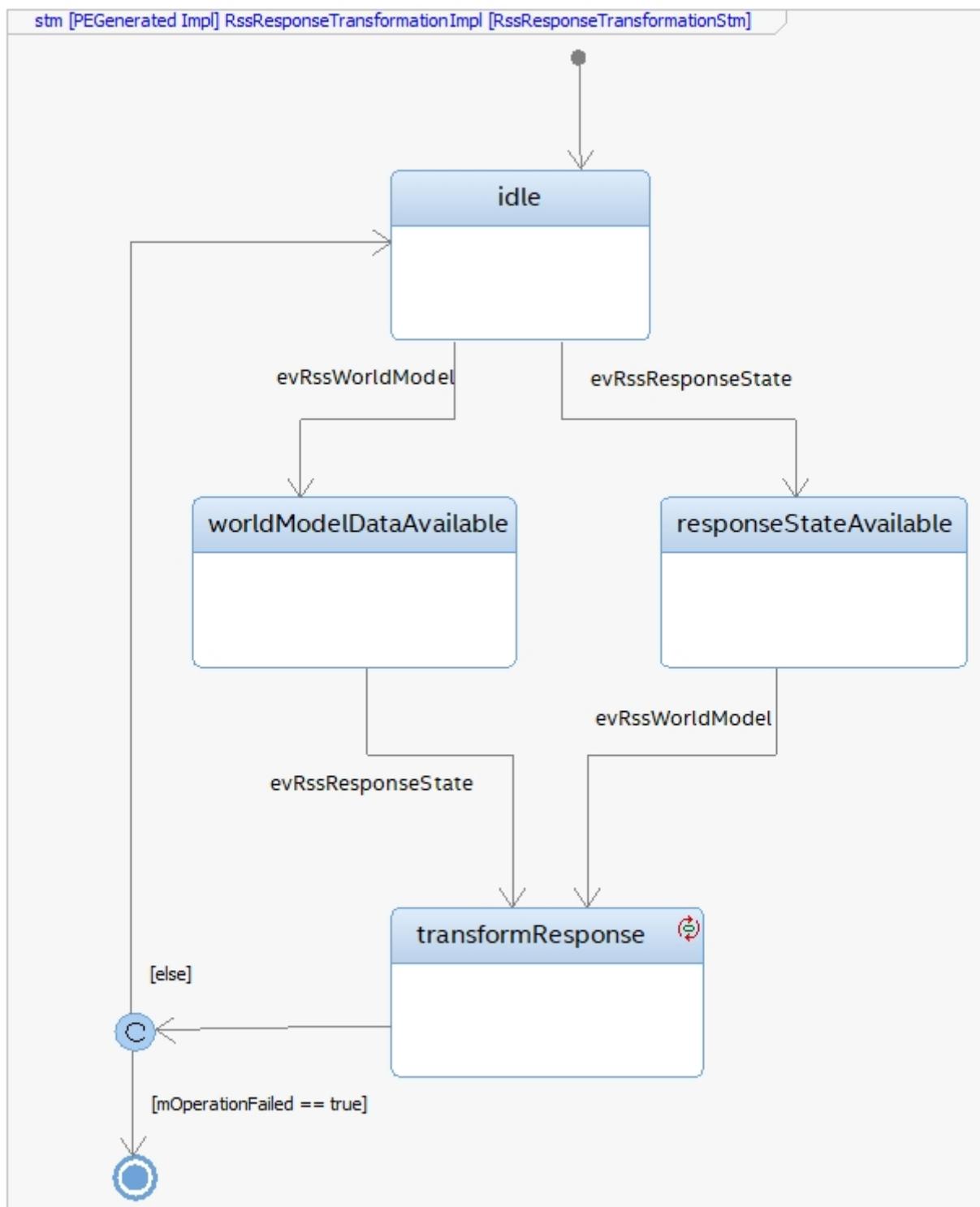


Figure 32. Statechart Diagram describing the dynamic behavior of the RssResponseTransformation entity.

The following table describes the events, triggers, states and their transitions of the statechart in detail.

Event/Trigger	Argument Type	Argument	Description
evRssResponseState	ResponseType	responseState	Event triggered when RssResponseState data is received at the input.

Event/Trigger	Argument Type	Argument	Description
evRssWorldModel	WorldModel	worldModel	Event triggered when RssWorldModel data is received at the input.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.
→	evRssWorldModel	worldModelDataAvailable	WorldModel data received.
→	evRssResponseState	responseStateAvailable	ResponseState data received.
worldModelDataAvailable			World model data received. Still waiting for response states.
→	evRssResponseState	transformResponse	ResponseState data received.
responseStateAvailable			Response state data received. Still waiting for world model.
→	evRssWorldModel	transformResponse	WorldModel data received.
transformResponse			All input data received. Perform operation to transform response.
→		checkOperationResult	
error			Final error state.

Chapter 5. Design for Security

5.1. Architecture and dataflow overview

This chapter discusses security aspects. As a reminder of the high-level architecture, we repeat the following two figures from the chapter [Chapter 3](#). The first figure shows the RSS function within the autonomous driving (AD) system. The RSS function has a single input called `rssWorldModelData`, and a single output called `rssAccelerationRestrictions`.

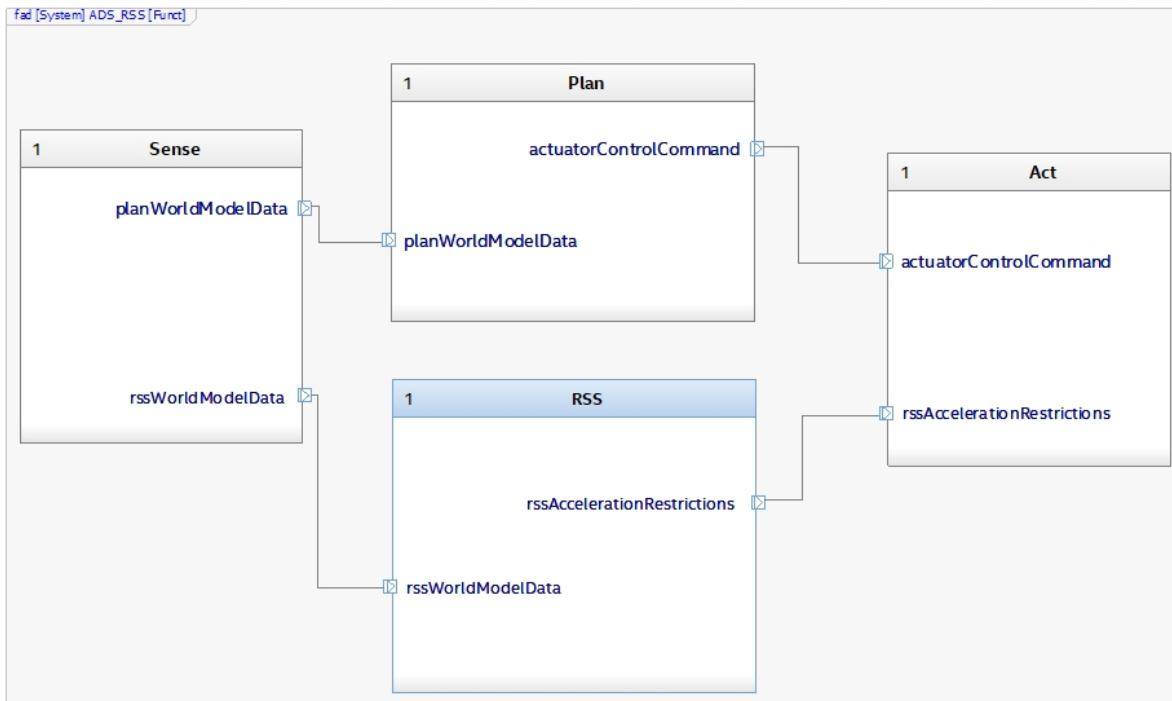


Figure 33. System architecture including RSS function

The second figure shows further detail of the RSS function, with the same external inputs and outputs (`rssWorldModelData` and `rssAccelerationRestrictions`), as well as the internal calculation steps and internal dataflow.

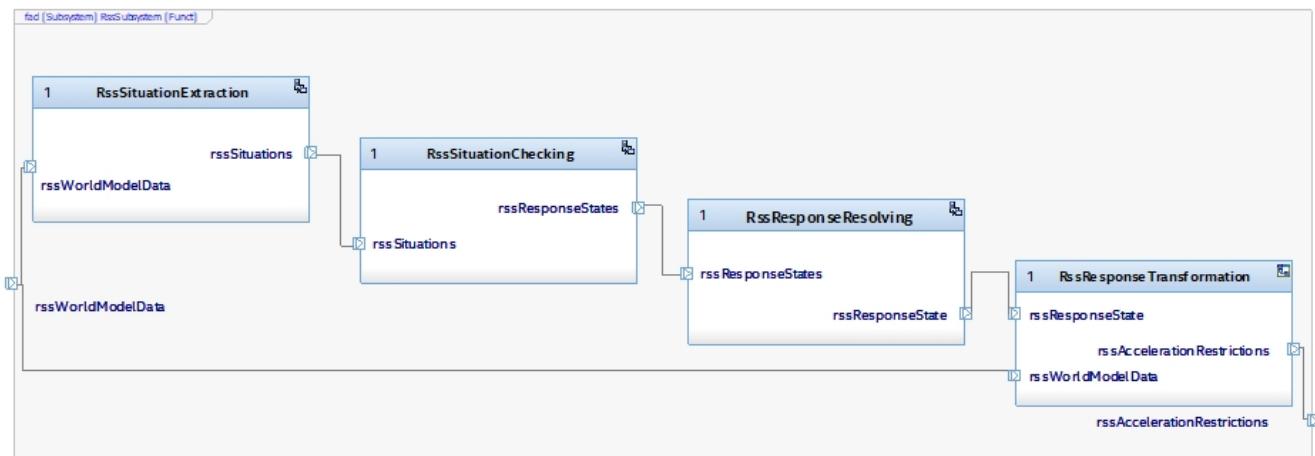


Figure 34. RSS calculation steps and internal dataflow

In addition, we repeat the following figure from the chapter [Chapter 4](#). This figure shows the actual C++ software class that implements the external RSS interface, called `RssCheck`. The top-level

external interface is implemented using a single routine, called `calculateAccelerationRestriction()`, which is a member function of the `RssCheck` class. This routine has a single input, called `worldModel` (of type `WorldModel`), and a single output, called `accelerationRestriction` (of type `AccelerationRestriction`), as consistent with the architecture diagram shown above.

This figure also shows the subroutines called from `RssCheck::calculateAccelerationRestriction()`, as consistent with the architecture diagram shown above. These subroutines, which are member functions of supporting classes, are:

- `RssSituationExtraction::extractSituations()`,
- `RssSituationChecking::checkSituations()`,
- `RssResponseResolving::provideProperResponse()`, and
- `RssResponseTransformation::transformProperResponse()`.

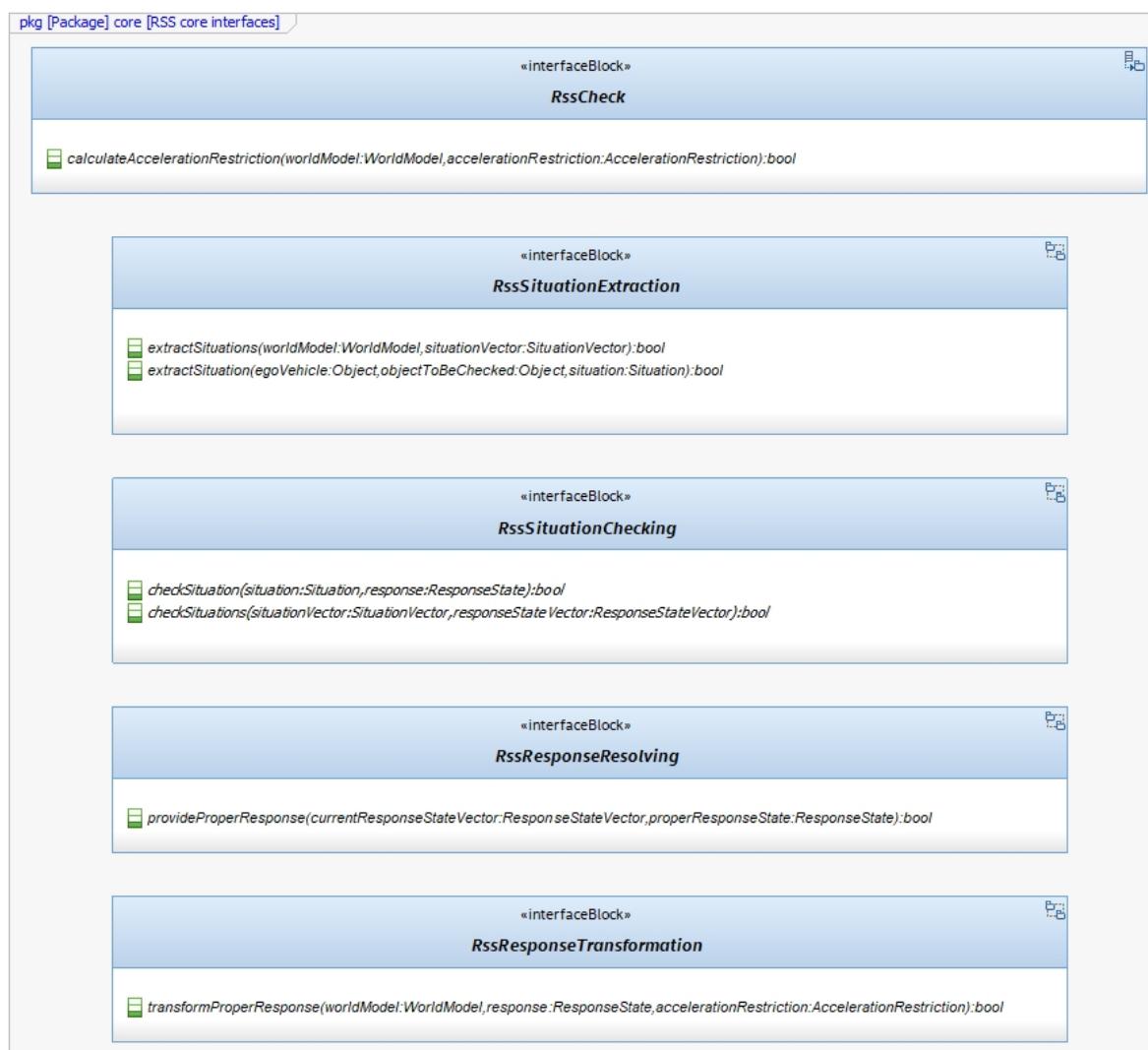


Figure 35. Classes and external interface provided by the RSS software implementation.

In the following sections, we discuss security aspects related to: input data (implemented by `WorldModel`), output data (implemented by `AccelerationRestrictions`), and the RSS function itself (implemented by `RssCheck` and supporting library classes).

5.2. RSS input data ([WorldModel](#))

5.2.1. Input data source

The input data is contained in [WorldModel](#). This data includes distances to other road agents (vehicles and other) and velocities of other road agents. This data also includes description of the geometry of the road (and road lanes). All data types are described in detail in the chapter [Chapter 4](#).

The data contained in [WorldModel](#) is provided directly by a calling routine external to the RSS library. Typically, this data will ultimately originate from the vehicle sensing subsystem, but it is the responsibility of the calling routine to potentially receive, convert, and provide this sensing-derived data to the RSS function in the correct format. The calling routine must ensure the integrity of the input data provided to the RSS library functions.

5.2.2. Error handling and logging

Error handling: Each routine in the library that invokes subroutines checks a success/fail flag returned by the subroutine. In addition, many routines perform custom checking of the internal logic against known constraints of RSS. A failure of a logic check or of a subroutine results in interruption of the routine and a fail flag being passed to the calling routine.

Specific checks that are performed include the following.

- Check that the data structure describing the local road and lane geometries is internally consistent. The description of a local road follows basic constraints, and if the description is inconsistent with these constraints, the routine that analyzes the situation from the input data returns, with the success/fail flag set to 'fail'.
- Check that the data structures describing the ego-vehicle or other objects (other vehicles) are internally consistent. The routine that analyzes the situation checks against basic constraints. If the constraints are violated, the routine returns with the success/fail flag set to 'fail'.

Exception handling: The code-blocks in high-level routines are enclosed inside C++ try-catch blocks. Any exception thrown at lower-level software routines results in a fail flag being returned to the caller of the publicly available high-level routines.

When any of the above errors occur, the top-level RSS routine, [RssCheck::calculateAccelerationRestriction\(\)](#), will return with the success/fail flag set to 'fail'.

Logging: The top-level routine in the library returns a success/fail flag that can be recorded by logging functions that should be available in the overall AD system (external to the library). Beyond this, the library does not perform error logging internally. In future versions of the library, if logging of specific internal errors is required, this may be implemented by returning more specific error codes.

5.2.3. Non-functional requirements

The top-level routine that is invoked to perform the RSS calculations, [RssCheck::calculateAccelerationRestriction\(\)](#), should be called at a rate that is lower or equal to

the rate that the actuation subsystem will accept actuation/control commands. For example, if the rate that the AD system accepts actuation/control commands is 10 Hz (once every 100 ms), the average rate that this RSS routine can be invoked should be less than or equal to 10 Hz.

The top-level RSS routine should be called at a rate that is lower or equal to the rate that the sensor subsystem will provide new sensing data. For example, if a camera-based perception system uses cameras with a frame rate of 30 Hz, the sensor subsystem will not provide its sensing data faster than 30 Hz. In that case, the average rate that [RssCheck::calculateAccelerationRestriction\(\)](#) can be invoked should be less than or equal to 30 Hz.

The size of the input data ([WorldModel](#)) provided to [RssCheck](#) is determined by the number of other objects (other vehicles or road agents) that the ego-vehicle is interacting with, and the size of the description of each object and associated road areas. The size of the descriptions of objects and road areas is strictly bounded, by the number and type of individual properties of these objects. We can limit the number of other objects (other road agents) the ego-vehicle is interacting with to a reasonable number N . Then it is straightforward to bound the size of data that must be consumed by the [RssCheck](#) in a single call, based on N and the sizes of individual datatypes. It is responsibility of the system integrator to determine N and to ensure the input data does not exceed this maximum.

Based on limiting the calling rate and data size for each call, one can bound the rate of the data that must be consumed by [RssCheck](#). It is the responsibility of the system integrator to determine the desired maximum data rate and to ensure the system does not exceed this maximum.

The latency resulting from the processing time of [RssCheck::calculateAccelerationRestriction\(\)](#) should be strictly limited, due to the real-time nature of the AD system and closed-loop interaction with the real world environment. This latency should be less than the time period (1/rate) of the ActuatorSubSystem. It is the responsibility of the system integrator to ensure that this is achieved, or to achieve a latency below an upper bound that is tighter (lower) than the one mentioned here.

5.3. RSS output data ([AccelerationRestriction](#))

[RssCheck::calculateAccelerationRestriction\(\)](#) outputs a structure of type [AccelerationRestriction](#). This is a very small structure that contains three ranges of acceleration allowed by RSS:

- allowable range of longitudinal acceleration,
- allowable range of lateral acceleration to the left of the vehicle, and
- allowable range of lateral acceleration to the right of the vehicle.

Each acceleration range consists of two values: a lower bound and an upper bound. The lower bound and upper bound simply specify an interval of allowable acceleration. Note that a negative value of acceleration implies deceleration, i.e. braking.

5.3.1. Output data destination

These three ranges (in [AccelerationRestriction](#)) are provided to the system that calls the RSS top-level routine, and indicate the range of acceleration that the AD system must ultimately achieve in order to comply to RSS. In other words, if the AD system seeks to comply to the RSS model, it must

achieve an acceleration that is within these bounds provided by the RSS function.

Typically, the system would provide the RSS acceleration restriction data to the actuation and control subsystem of the autonomous driving system. This subsystem, external to the RSS library, should enforce the lower and upper bounds of lateral and longitudinal acceleration provided by the RSS function. Implementation of this process is external to the RSS library, and the system integrator must ensure that this is implemented properly and that RSS output values are used correctly.

5.3.2. Bounds on the output data values

As discussed above, the output values of the RSS function are lower and upper bounds on acceleration. The lower and upper bounds calculated by the RSS function depend on the input and ultimately on the situation of the ego-vehicle and other surrounding agents. For example, in some situations (such as when the ego-vehicle does not encounter any other agents in its surroundings), the upper bound on acceleration is some finite positive value, indicating that the ego-vehicle can accelerate up to that value, in compliance with RSS rules. In other situations (such as when the ego-vehicle is closing in on another vehicle in front), the upper bound on acceleration is some negative value, indicating the the ego-vehicle must brake at least as much as sufficient to achieve that negative upper bound, in order to comply with RSS rules.

These output values are themselves bounded by absolute lower and upper limits that are specified by the RSS model. For example, a vehicle acceleration should never be higher than an RSS-specified absolute upper limit in any situation. Likewise, the vehicle should never brake harder so as the result in a negative acceleration that is lower than an RSS-specified absolute lower limit. These limits are specified in the RSS paper (see [Table 3](#)). The system that receives output values from the RSS function can always check these values against these RSS-specified absolute lower and upper limits. Furthermore, it is the responsibility of the system external to the RSS library to ensure that the acceleration and deceleration actually achieved by the vehicle always remain within these RSS limits and remain within other limits unrelated to RSS (e.g. physical limits imposed by the vehicle dynamics and the environment).

5.3.3. Bounds on the output data size

The output data of the RSS calculations consists of 6 floating point values (3 ranges, each with a lower and upper limit).

As discussed in the previous sections, the RSS function should be called at a rate that is lower or equal to the rate that the actuation subsystem will accept actuation/control commands. The calling rate should also be lower or equal to the rate that the sensor subsystem will provide new sensing data. Based on limiting the calling rate and data size for each call, one can bound the rate of the data that is produced by the RSS function and must be consumed by the calling system. It is the responsibility of the system integrator to determine the desired maximum data rate and to ensure the system can handle this maximum data rate.

5.4. RSS library ([RssCheck](#) and supporting classess)

5.4.1. Dependencies

The RSS library has no dependency on any external library (except for the C++ Standard Library).

5.4.2. Coding standards

Code quality has been ensured through unit testing.

Unit testing achieved code coverage of: 95% lines, 100% functions, 83% branches.

5.4.3. Compiler and compiler security flags

The development platform is Ubuntu Linux 16.04 Xenial.

A standard cmake toolchain has been used to compile the library. The compiler is g++ 5.4.0.

The following, strict, compilation flags are used: C++14, -Werror, -Wall, -Wextra, -pedantic, -Wconversion, -Wsign-conversion.

- -Werror turns all warnings into errors
- -Wall enables many warnings about code constructs that are questionable
- -Wextra enables additional warnings not enabled by -Wall
- -Wconversion warns for implicit conversions (e.g. between integer and real types)
- -Wsign-conversion warn for implicit conversions that reduce
- -pedantic issues all warnings demanded by strict ISO C and C++

Compilation with these flags completes without any error or warning.

5.4.4. Code analysis

The code analysis tool [cppcheck](#) was run on the library code, and no relevant issues found.

5.4.5. Critical data path and data structures

The data flow through the steps of the RSS calculations are shown in the introduction of this chapter. The input data, contained in [WorldModel](#) is a critical data structure. If the input data is invalid, either the RSS function will return with the 'success/fail' flag set to 'fail', or the output values will likely be invalid. Other than that, there is a single direct dataflow through the RSS calculations internal to the library. Failure at any point in the calculations leads to the calculations being stopped and the RSS function returning with a flag set to 'fail'.

Chapter 6. Design for Safety

In principle, RSS is implementing safety of the intended functionality (SOTIF). One can apply RSS without general functional safety (FuSa) aspects to safeguard the planning functionality. If RSS is implemented in an ADS with FuSa constraints on system level, RSS contributes to the FuSa goals of the whole ADS.

Appendix

Chapter 7. Parameter Discussion

The RSS papers uses a few constants required for the safety calculations. The values for these constants are not yet defined and open for discussion/regulation. Nevertheless, the implementation of the RSS module needs to define initial values for these functions.



To be as flexible as possible, the parameters are implemented as configuration values so these can be easily adjusted during evaluation or after the release. In other words, the values are not hard coded in the library, but can be changed via the provided inputs.

In the following, the key parameters and the decision for possible initial values are discussed. The used parameters are (see paper 1 in [Table 3](#)):

- Response time ρ . It is assumed that an AV vehicle has a shorter response time than a human driver. Therefore, there is a need to have two different parameters. As it might not be possible to determine whether another object is an AV vehicle or has a human driver, the RSS module will safely assume that all other objects are driven by humans. Hence, two parameters for the response time are used.
 - ρ_{ego} for the ego vehicle
 - ρ_{other} for all other objects
- Acceleration α . RSS proposes several different acceleration/deceleration values. One could argue that acceleration/deceleration differs with the type of vehicle. Also at least the acceleration is dependent on the current vehicle speed. As it cannot be assured that the individual acceleration of each and every car can be known and the specific car can be reliably detected, the RSS module will assume fixed constants for those values. These could be either the maximum physically possible values or restrictions that are imposed by regulation. Also there will not be different values for the ego vehicle and the other vehicles. It could be argued that for the ego vehicle e.g. desired acceleration might be known. Therefore, a shorter safety distance would be sufficient. But as all other vehicles do not know about the intention of the ego vehicle this would lead to a violation of their safe space. So the RSS module will need to calculate its checks with the globally defined accelerations values even if the vehicle does not intend to utilize them to its limits. The parameters used for acceleration are:
 - $\alpha_{accel,max}$ maximum possible acceleration
 - $\alpha_{brake,min}$ minimum allowed braking deceleration in longitudinal direction for most scenarios
 - $\alpha_{brake,max}$ maximum allowed deceleration in longitudinal direction
 - $\alpha_{brake,min,correct}$ minimum allowed deceleration in longitudinal direction for a car on its lane with another car approaching on the same lane in wrong driving direction
 - $\alpha_{brake,min}^{lat}$ minimum allowed braking deceleration in lateral direction
 - $\alpha_{accel,max}^{lat}$ maximum allowed acceleration in lateral direction
 - δ_{min}^{lat} fluctuation margin for that needs to be respected when calculating lateral safe distance

7.1. Decision on Selected Parameter Values



The following parameter values are only suggestions and open for discussion. These can be changed at anytime, if it is required.

7.1.1. Response time

For the response times a common sense value for human drivers is about 2 seconds. For an AV vehicle the response time could be way lower. In order to be not too restrictive the initial value for the ego vehicle response time will be assumed as 1 second. Hence, $\rho_{other} = 2\text{ s}$ and $\rho_{ego} = 1\text{ s}$. If we assume a case of AD vehicles only, the response time may be reduced.

7.1.2. Longitudinal Acceleration

Finding meaningful acceleration values is more complicated. At the one hand the values should be as close as possible or even exceed the maximum physically possible values. The minimum deceleration values must also not exceed normal human driving behavior. So assuming a too high deceleration for other cars may lead to a false interpretation of the situation.

On the other hand a too big difference between the minimum and maximum acceleration values will lead to a very defensive driving style. As a result, participating in dense traffic, will not be possible (see [Figure 36](#)). A rule of thumb for deceleration in German driving schools is: $a_{brake,min} = 4\text{ m/s}^2$ and $a_{brake,max} = 8\text{ m/s}^2$

But on the other hand, modern cars are able to decelerate with up to 12 m/s^2 . Especially for deceleration, it is questionable whether it is possible and tolerable to restrict maximum braking below physically possible braking force.

For the maximum acceleration at low speeds a standard car will be in the range of 3.4 m/s^2 to 7 m/s^2 . But there are also sport cars that can go faster than that. But for acceleration a regulation to a maximum value seems to be more likely than for deceleration.

7.1.2.1. Restricting velocity to the current speed limit

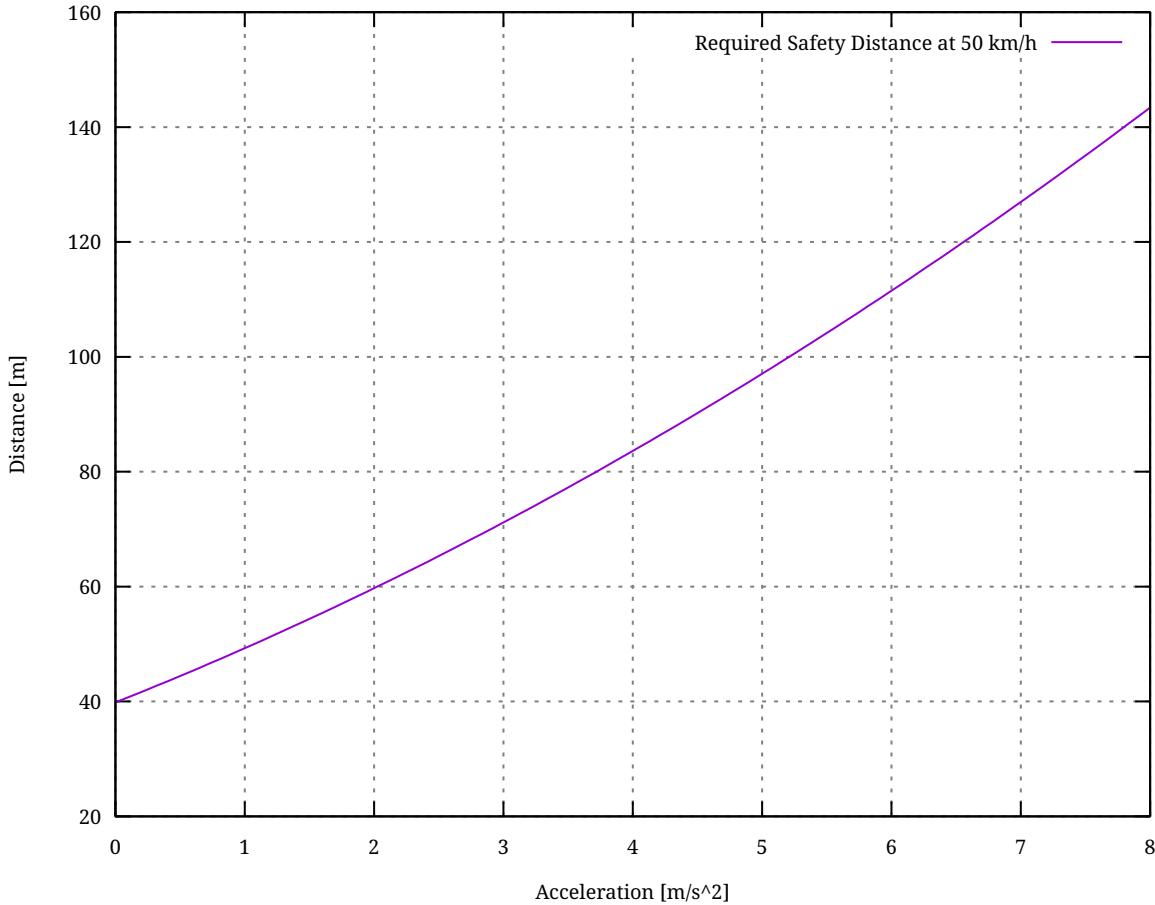


Figure 36. Required safety distance for cars driving at 50 km/h (city speed) in same direction with $a_{brake,min} = 4 \text{ m/s}^2$ and $a_{brake,max} = 8 \text{ m/s}^2$ and $\rho_{ego} = \rho_{other} = 2 \text{ s}$

The assumption that a car can always accelerate at $a_{accel,max}$ during the response time, leads to a significant increase of the required safety distance. Figure 36 shows the required safety distance for different acceleration values. So acceleration about 4 m/s^2 doubles the required safety distance from 40 m to about 80 m at city speeds.

Therefore, it might be advisable to add a restriction that a car is only allowed to accelerate up to the allowed speed limit. In addition, the common behavior (in Germany) is to respect a safety distance of speed/2, e.g. in a city with a speed limit of 50 km/h the safety distance shall be 25 m . Hence, it is obvious that the parameters may require some adjustments to allow reasonable driving.

7.1.2.2. Further possible restrictions

Another possibility to decrease the required safety distance to the leading vehicle would be to take the intention of the ego vehicle into account. E.g. if the ego vehicle is following another vehicle and is not intending to accelerate, then there is no need to assume that the ego vehicle is accelerating during its response time. Nevertheless, there are several issues with that approach:

1. It needs to be assured that all intended and unintended accelerations (e.g. driving down a slope) are known to RSS.
2. If RSS formulas are regarded as regulations, the safety distance must be kept regardless to the intent of the vehicle.

Therefore, in the current implementation this approach will not be applied.

7.1.3. Lateral Acceleration

When defining the parameters for lateral acceleration and deceleration, it is important to keep in mind that the definition must allow bypassing of vehicles. Physically high lateral accelerations are possible, especially as the transform to the lane coordinate system could add additional lateral acceleration. In order to be able to bypass a vehicle that is driving on a parallel lane, the safe lateral distance needs to be safe during the complete response time of the other vehicle.

Let us consider two identical vehicles driving on the centerline of two adjacent lanes with zero lateral velocity. There is no lateral conflict, if the distance between the border of the car and the adjacent lane is bigger than the distance that the vehicle will cover when accelerating laterally at maximum during its response time and then decelerating to zero lateral velocity.

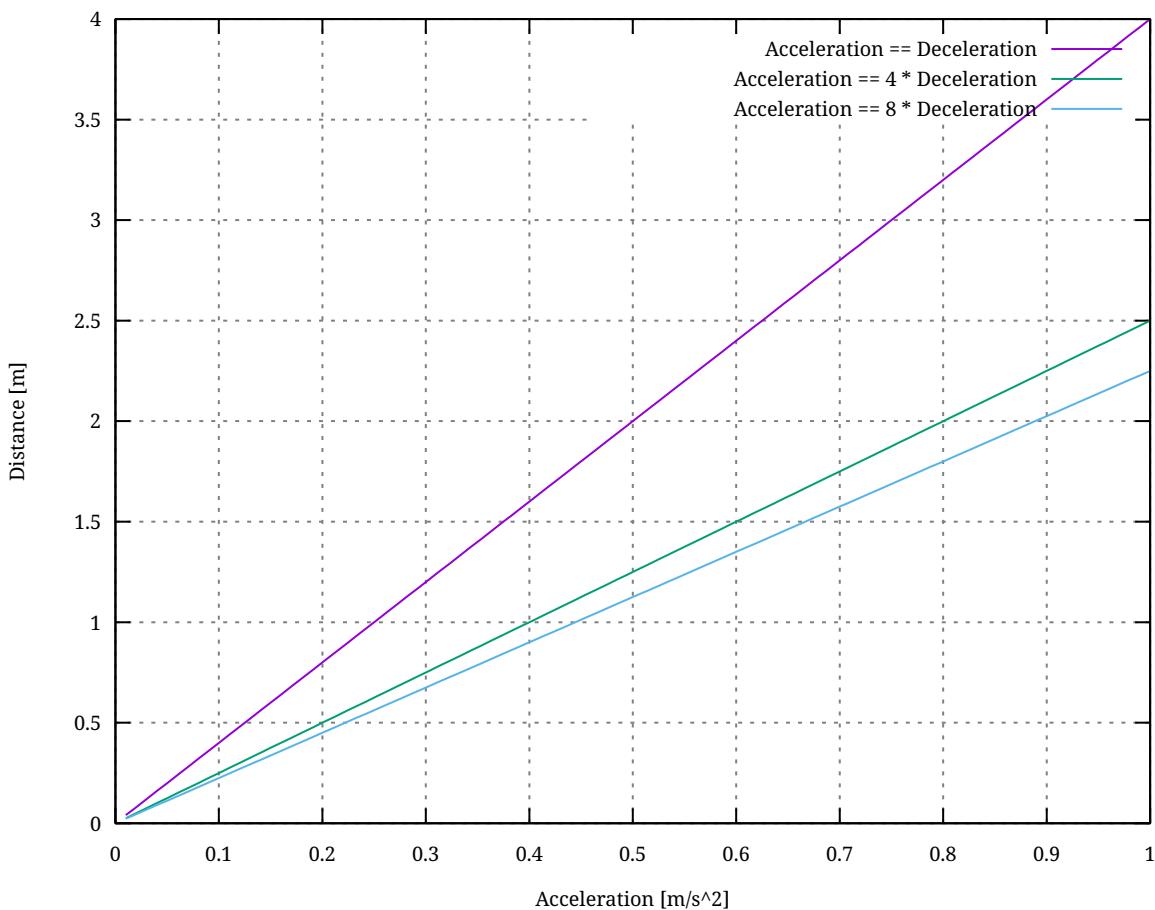


Figure 37. Distance a vehicle will cover when applying the "Stated Braking Pattern" with $\rho_{vehicle} = 2 s$

Figure 37 shows the required safety distance, without considering the fluctuation margin, each car needs to keep to the lane border so the vehicles can pass without lateral conflict. With an assumed minimal lane width of 3 meters and an assumed vehicle width of 2 meters, the distance from vehicle edge to lane border is 0.5 meter if the car is driving exactly in the middle of the lane.

Hence, the required safety distance must be at most 0.5 meter. When using the same values for acceleration and deceleration this will lead to $a_{accel,max}^{lat} < 0.1 m/s^2$. But when restricting the acceleration to that value a lane change will take almost 8 seconds.

As a result it is advisable, to use a higher deceleration than acceleration to keep the required safety margin and allow for faster lane changes. E.g. $a_{brake,min}^{lat} = 0.8 m/s^2$ and $a_{accel,max}^{lat} = 0.2 m/s^2$ will fulfill

the given safety distance requirement. An increase to higher acceleration values is for the given constraints not possible, as the distance covered during response time is already 0.4 meters.

It is obvious that given the lateral safety definition a lane change will at least have a duration of two times the response time.

The lateral distance requirement is very strict, therefore it is required to also come up with a desirably small value for the required lateral safety margin δ_{min}^{lat} . As this should only cover for fluctuations, there is also no need for a huge margin. Thus initially this value will be set to $\delta_{min}^{lat} = 10\text{ cm}$. This value should be able to cover small fluctuations, but will not have a big impact on the safety distance.

As a starting point the values are set to:

Table 1. Chosen Default Parameters



Parameter	Value
ρ_{ego}	1 s
ρ_{other}	2 s
$a_{accel,max}$	3.5 m/s^2
$a_{brake,min}$	4 m/s^2
$a_{brake,max}$	8 m/s^2
$a_{brake,min,correct}$	3 m/s^2
$a_{brake,min}^{lat}$	0.8 m/s^2
$a_{accel,max}^{lat}$	0.2 m/s^2
δ_{min}^{lat}	10 cm

Chapter 8. Terminology

Table 2. Terminology

Term	Description
AD	Automated Driving
ADS	Automated Driving System
HLD	High Level Design
Long	Longitudinal
Lat	Lateral
RSS	Responsibility-Sensitive Safety
RSS Module	This is the entity performing all RSS operations

Chapter 9. References

Table 3. References

Ref	Document Name	Version	Location
1	On a Formal Model of Safe and Scalable Self-driving Cars	v6	https://arxiv.org/abs/1708.06374
2	Implementing the RSS Model on NHTSA Pre-Crash Scenarios	July 2018	https://www.mobileye.com/responsibility-sensitive-safety/rss_on_nhtsa.pdf

Chapter 10. Revision History

Table 4. Documentation Revision History

Revision	Date	Author(s)	Description
0.1	2018	Cornelius Buerkle, Bernd Gassmann, Fabian Oboril	Created
0.2	2018-08-29	Cornelius Buerkle, Bernd Gassmann, Fabian Oboril	Updated
0.29	2018-11-14	Peter van Beek	Added content to Design for Security chapter (in progress)
0.3	2019-01-31	Bernd Gassmann	Adapt to RSS paper v6

Chapter 11. Build RSS core library and documentation

11.1. Build instructions

The RSS library is built with a standard cmake toolchain. Therefore, run the following commands, to build the library and documentation:

```
mkdir build  
cd build  
cmake ..  
make  
make apidoc # optional, if API documentation is desired
```

11.2. Generate PDF document

To generate a PDF for this document, it is recommended to use asciidoctor-pdf ([Asciidoctor](#)). Therefore, the following commands have to be executed:

```
sudo apt-get install asciidoctor  
sudo -E gem install asciidoctor-pdf --pre  
sudo -E gem install coderay  
asciidoctor-pdf ./doc/Main.adoc -n
```

11.3. Generate HTML document

To generate a HTML for this document, the following commands have to be executed:

```
sudo apt-get install asciidoctor  
asciidoctor doc/Main.adoc -d book -b html5 -n
```