RSS Documentation

Cornelius Buerkle, Bernd Gassmann, Fabian Oboril

Version 0.2, 2018-08-29

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. Lateral conflicts . . . 7
2.1.2. Longitudinal conflicts . . . 7
2.1.3. Handling of Intersections . . . 9
2.1.4. Response Time and Other Parameters . . . 9
2.2. Situation-Based Coordinate System . . . 9
2.2.1. Comparing movements in lane-based coordinate systems . . . 10
2.2.2. Chosen Design: Individual lane-based coordinate system with properly scaled acceleration values 12
2.2.3. Design alternative: Iterative Approach [optional] . . . 15
2.3. Summary . . . 16
2.3.1. Key decisions . . . 16
2.3.2. Proposed changes / extensions to definitions in the RSS paper . . . 16
3. RSS System Architecture Overview . . . 17
3.1. Sense . . . 18
3.1.1. SensorSubsystem . . . 18
3.2. Plan . . . 19
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 . . . 25
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 . . . 57
5. Design for Security . . . 60
6. Design for Safety . . . 61
Appendix . . . 62
7. Parameter Discussion . . . 63
7.1. Decision on Selected Parameter Values . . . 64
7.1.1. Response time . . . 64
7.1.2. Longitudinal Acceleration . . . 64
7.1.3. Lateral Acceleration . . . 66
8. Terminology . . . 68
9. References . . . 69
10. Build RSS core library and documentation . . . 70
10.1. Build instructions . . . 70
10.2. Generate PDF document . . . 70
10.3. Generate HTML document . . . 70

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.

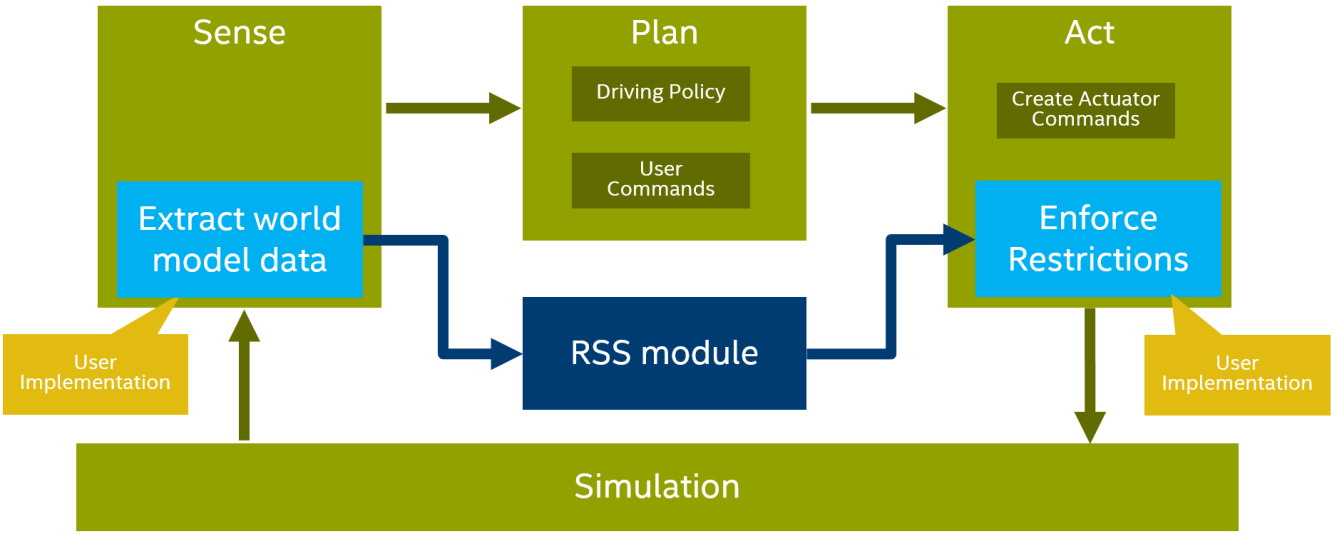

Figure 1. RSS module (implemented in the library) and its interfaces to the outside world

In summary, the RSS module receives an object list, with information about all object 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 obtained by the RSS module, and the corresponding actuator command restrictions (i.e. lateral and longitudinal acceleration restrictions) are sent out (see Figure 1).

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 scope of the planned release for end of Q3'2018 is:

- Standalone C++-library containing the implementation of the RSS module
- The RSS module covers multi-lane roads and intersections.
- Code quality will be ensured by automated testing with test coverage 80/50 and static code analysis
- Set of CarMaker scenarios to demonstrate functionality of library

The initial design does not yet cover:

- 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, and discusses potential issues.
- 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.

2.1.1. Lateral conflicts

The lateral checks and the proper response follow the definitions 8 and 9 of paper 1 (see Table 3).

The initial design does not yet cover the case of a cut-in (Definition 9.3.(2))

2.1.1.1. Lateral evasive maneuvers

If the car finds itself in a dangerous situation one possible action is always to brake. According to the RSS papers, this should always result in a safe state, but might lead in worst case to a complete stop of the vehicle. To improve the response, the RSS paper proposes that the car may perform also lateral maneuvers as a response to certain dangerous situations. 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. However, such a prediction for all objects comes with a high uncertainty, and therefore, would not be reliable. In addition to that, RSS just has only visibility of the dynamic and static objects in the environment. Hence, it cannot determine that a lateral movement initiated by RSS will cause an accident with an obstacle, or forces the vehicle to leave the road (as road boundary checks are currently not part of RSS).

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*. Instead, as formulated in the RSS papers, 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 given by RSS.

2.1.2. 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 and 4 of paper 1 (see Table 3). For the case of vehicles driving in opposite directions, the implementation follows the definitions 2 and 4 of the paper, but uses an extension, as explained next.

2.1.2.1. Response for vehicles driving in opposite directions

If two vehicles are approaching each other in opposite direction and the lateral distance between them is not safe, the RSS paper defines that the car that is on its own lane can brake with lower force than the car on the opposite direction. This definition leads to a correct behavior, if it is obvious that one car is on its own lane and the other car is driving on the lane belonging to the other car.

However, there are situations where this cannot be determined. One situation e.g. is that there is a bidirectional single lane road. For sure on such a road each car should drive on the right side, but it cannot be easily detected which car is no longer on its side of the road. Another situation is depicted in Figure 2, where there is a two lane road with one lane for each direction. If each car drives on its own lane but is very close to the left border, it could lead to a lateral conflict, as the safe lateral distance requires a safety margin. In this situation both cars are on the correct lane, but still there is a lateral conflict.

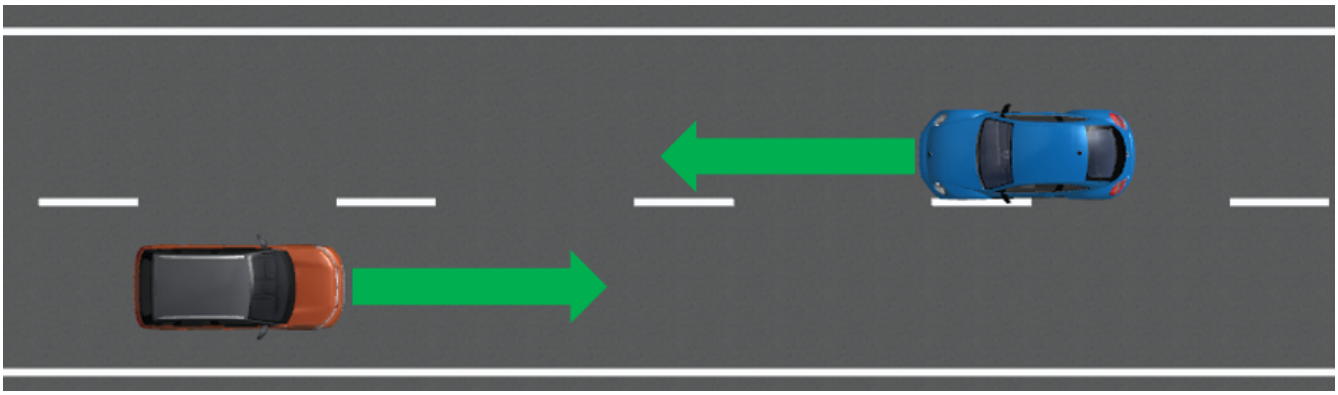

Figure 2. Two cars are driving towards each other, both on the correct lane

As a consequence, in all these cases, both cars will brake with low force and expect the other car to brake stronger. Consequently, this dangerous situation will never be solved and the cars will crash.

To avoid such a catastrophic situation, and to avoid that the ego vehicle is blamed, the RSS module will extend the rule given in definition 4.2 of the paper to the following.

Given a situation that two cars approach each other with a lateral conflict, the following rules apply:

1. Both cars must brake according to the "stated braking pattern", if the longitudinal distance is not safe given that both cars accelerate at maximum during their response time and brake with $\alpha_{min,brake}$.
2. If case 1 does not apply, and the longitudinal distance is not safe according to the definition in "Safe Longitudinal Distance for Opposite Directions", the vehicle on the correct lane has to brake with $\alpha_{min,brake,correct}$, and the vehicle on the wrong lane has to conduct the "stated braking pattern" (This is the situation described in the RSS paper).

As a result of these rules, two cars that approach each other, which are both on the correct lane (e.g. a bidirectional single lane road), will brake with $\alpha_{min,brake}$. If this is not desired, a special handling for this situation has to be introduced.

2.1.3. Handling of Intersections

The behavior for intersection conflicts (checks and response) for vehicles is implemented as described in the definitions 14 of paper 1 (see Table 3).

In detail, the current realization looks as follows:

1. It is checked, if the non-prioritized vehicle *was* able to stop in front of the intersection. If this is the case, the non-proritized 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 13.2 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 14.3 of paper 1 applies, i.e. both cars have to brake laterally and longitudinally with at least $\alpha_{min,brake}$.

Case 1. is the direct realization of Definition 13.1, 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. Therefore, it might be advisable to modify this check.

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

2.1.4. 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 $\alpha_{accel,max}$, maximum deceleration $\alpha_{brake,max}$, 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 < 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 (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 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. To illustrate this problems, let us consider the following examples:

2.2.1.1. Discontinuity Problem: Two parallel lanes, different width

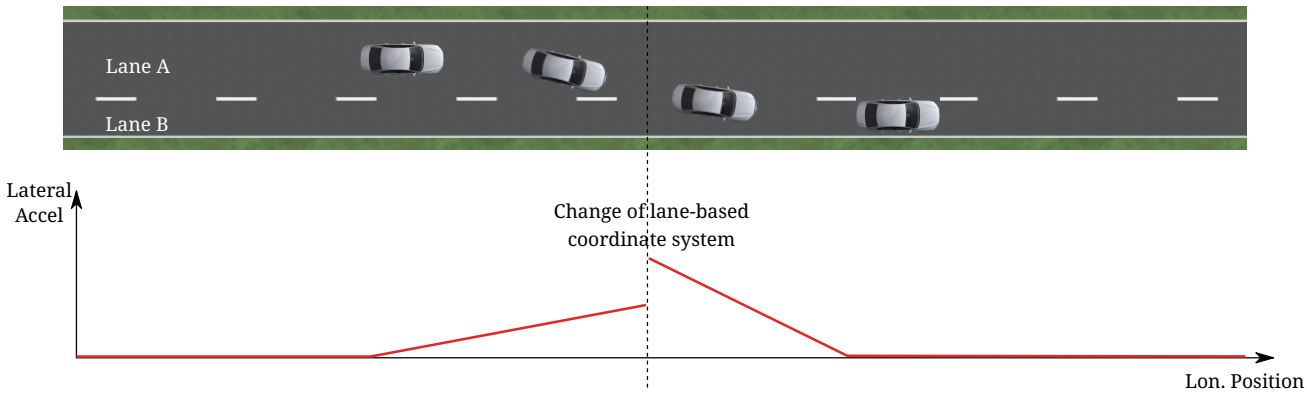

Figure 3. Two parallel lanes with different width causing a discontinuity in lateral acceleration

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

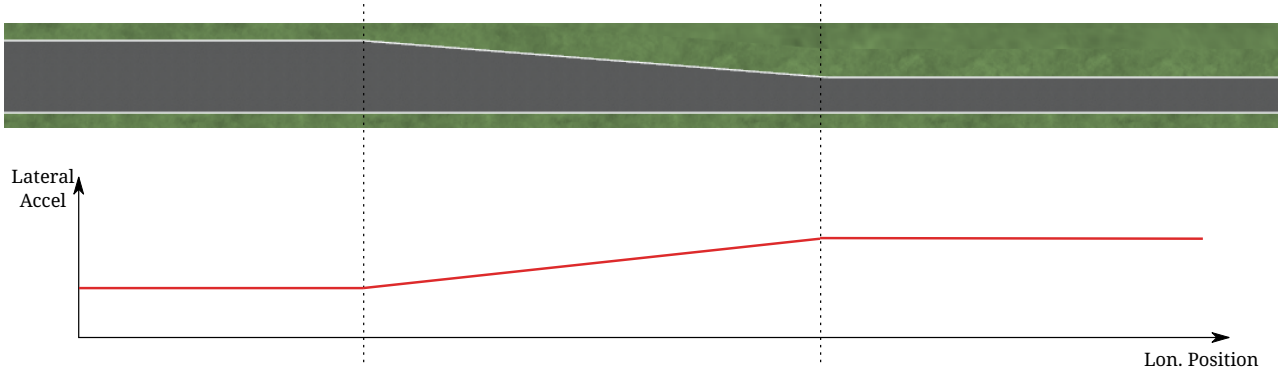

Figure 4. Changing lane width and its impact on the lateral acceleration

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/s²* is changing from *0.25 lat/s²* at the beginning towards *0.5 lat/s²* while advancing within the lane.

2.2.1.3. Changing Distances Problem: Lane with a narrow curve

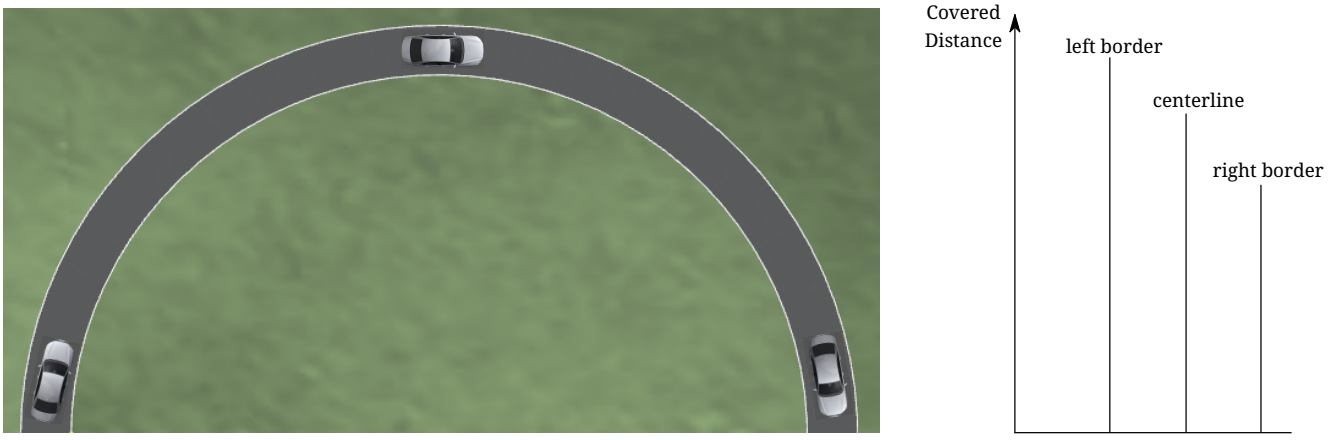

Figure 5. Lane describing a narrow 180° curve and its impact on driven distances

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/s²* for the center line, *0.96 lon/s²* for the outer border and *1.04 lon/s²* 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 the this situation.

2.2.2. Chosen Design: Individual lane-based coordinate system with properly scaled acceleration values

As RSS performs a worst case assessment, the idea followed by the RSS module implementation is to scale the min/max acceleration values for calculation of the safe distances in order to adapt to the observed situation individually. 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 3).

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 6

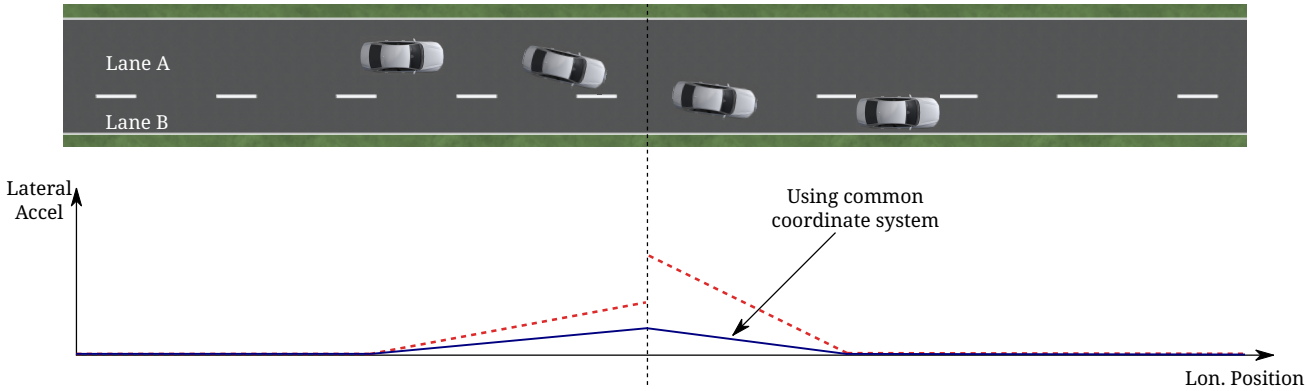

Figure 6. Avoid discontinuities by using one single situation-based coordinate system

Coming back to the concrete example from Figure 3, 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*. A Cartesian lateral acceleration value of $\alpha = 1\ m/s^2$ becomes an acceleration value of $\alpha_i = 1/6\ lat/s^2 = 0.167\ lat/s^2$ within the individual situation specific oordinate system SCS_i (see also illustration in Figure 6).

The check 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 using a different acceleration value of $\alpha_j = 1/9\ lat/s^2 = 0.111\ lat/s^2$.

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 scale the applied acceleration values 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 maximum possible acceleration into the situation coordinate system *SCS* has to take the minimum width of *2 m* into account, while the transformation of the deceleration values has to be transformed with the maximum width of the lane of *4 m* (see Figure 7). Like this it is guaranteed that we neither underestimate the acceleration of the vehicles towards each other nor overestimate the deceleration of the vehicles while braking. As a result, it is ensured that under all conditions, the safety distances are calculated in a conservative manner.

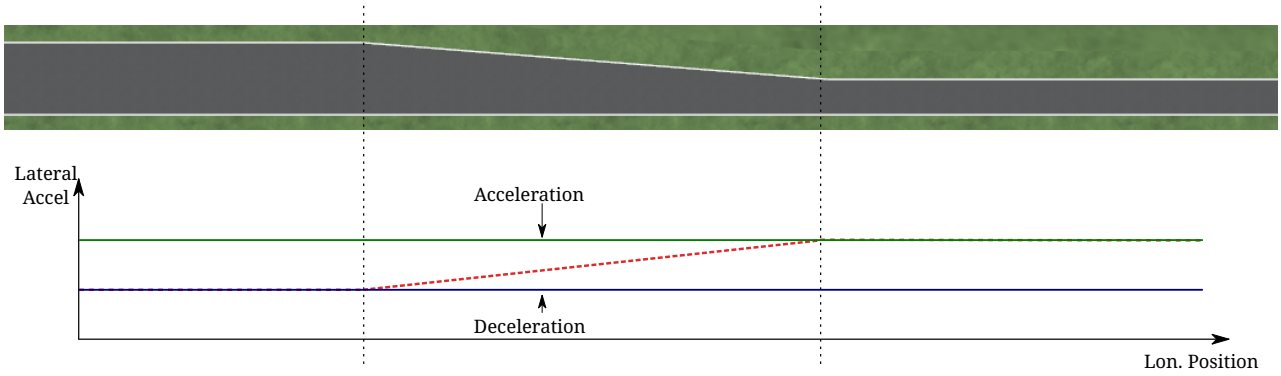

Figure 7. Changing lane width and the resulting deceleration and acceleration transformations

In a similar way, it is possible to transform the longitudinal acceleration values into the situation-based coordinate system SCS_k. Taking the nominal center line length (in the above example: *163.4 m*) as basis, we have to apply the factors $scale^{lon}_{k,min} = 0.96$ and $scale^{lon}_{k,max} = 1.04$ appropriately to consider the minimum and maximum lane length of *157.1 m* and *169.7 m*. The decision on which of the two factors has to be selected for which of the acceleration/deceleration values depends also on the situation between ego vehicle and the actual object.

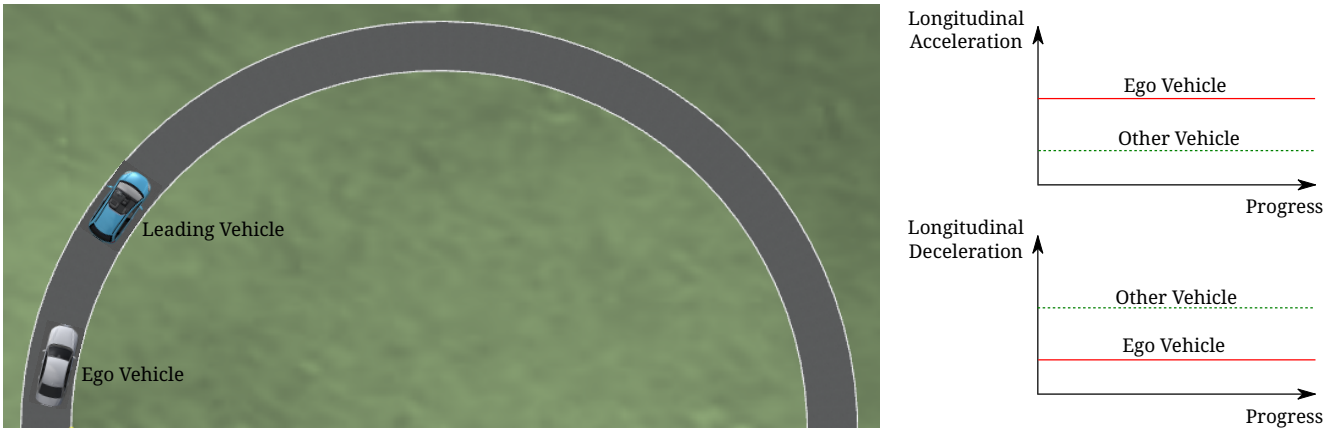

Figure 8. Lane describing a narrow 180° curve and the resulting deceleration/acceleration for a leading vehicle and the following ego vehicle

In case the ego vehicle is following object o_k within the same lane, the acceleration value of the ego vehicle ($\alpha^{ego}_{accel,k} = \alpha_{accel} * scale^{lon}_{k,max}$) as well as the deceleration values of the object o_k ($\alpha^{o}_{brake,k} = \alpha_{brake} * scale^{lon}_{k,max}$) have to be scaled with the maximum scale factor *1.04*, whereas the deceleration of the ego vehicle ($\alpha^{ego}_{brake,k} = \alpha_{brake} * scale^{lon}_{k,min}$) and the acceleration of the object ($\alpha^{o}_{accel,k} = \alpha_{accel} * scale^{lon}_{k,min}$) have to be scaled with the minimum scale factor *0.96*. This has to be adapted in case the ego vehicle is the vehicle in front or the object is approaching from the opposite direction. Nevertheless, there is always a selection possible that guarantees that the worst case is covered.

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.

2.2.2.3. 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 into Cartesian space. 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 9. However, in the vehicle specific lane-based system the lateral acceleration will be 0.

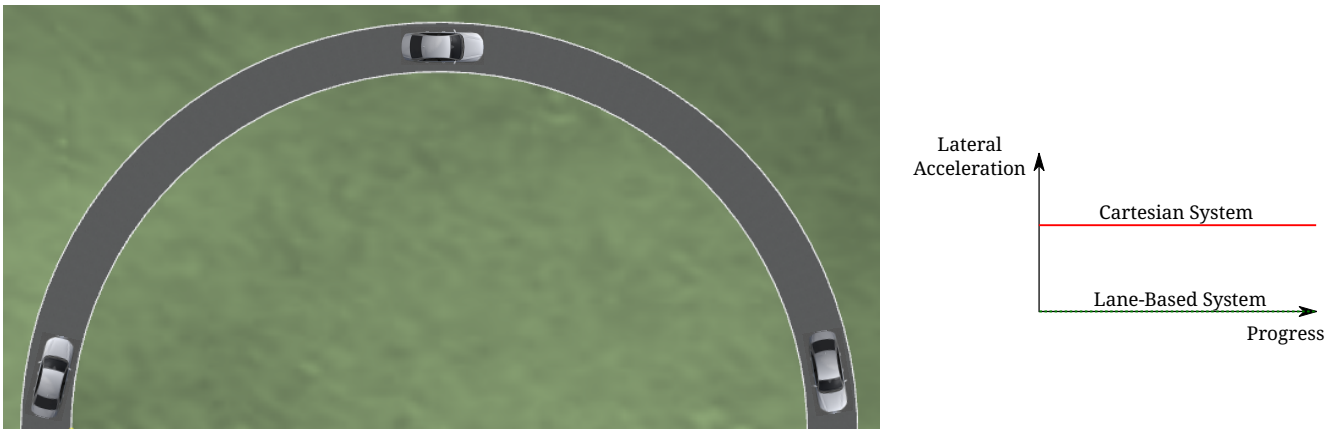

Figure 9. Constant drive around a curve will result in a zero lateral acceleration in a lane-based coordinate system and in a non-zero acceleration in a cartesian system

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 10, 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.

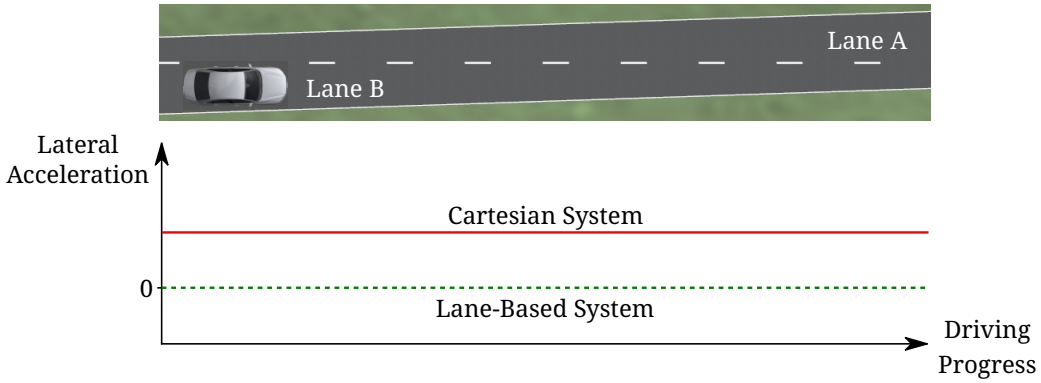

Figure 10. Different lateral accelerations in a lane-based system and Cartesian system for a vehicle following the centerline of lane B

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.

2.2.2.4. 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 scaling of the applied acceleration and braking values allows the calculation of the safe distances, the decision on dangerous situations and deduction of a proper response.

2.2.3. Design alternative: Iterative Approach [optional]

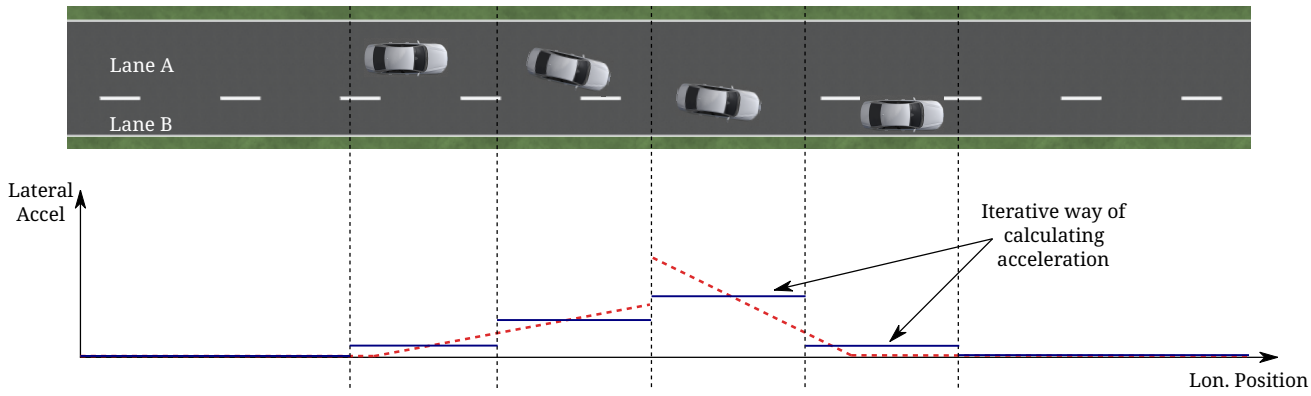

Figure 11. Illustration of an iterative approach to calculate non-constant acceleration, velocity etc.

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 11).

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 lateral evasive maneuvers.
- To handle changing lateral/longitudinal acceleration parameters when transforming the Cartesian space into the lane-based system, the acceleration and deceleration parameters are chosen in such a way that these are constant within the lane-based system, but guarantee safe operation (see Section 2.2.2).

2.3.2. Proposed changes / extensions to definitions in the RSS paper

- When two vehicles are driving in opposite direction, but both cars are on the correct lane, both cars will brake with $\alpha_{brake,min,correct}$ assuming that the other car slows down with $\alpha_{brake,min}$. However, this may not clear the dangerous situation. Therefore, it is important to introduce a special treatment for the case of opposing cars that both are on the correct lane. This handling is explained in Section 2.1.2.1.
- It cannot be determined whether lateral evasive maneuvers are actually possible. Therefore, the RSS Module will not initiate such maneuvers, but will not hinder the driving policy to execute lateral evasive maneuvers, as long as these are compliant with the required RSS reponse.

Chapter 3. RSS System Architecture Overview

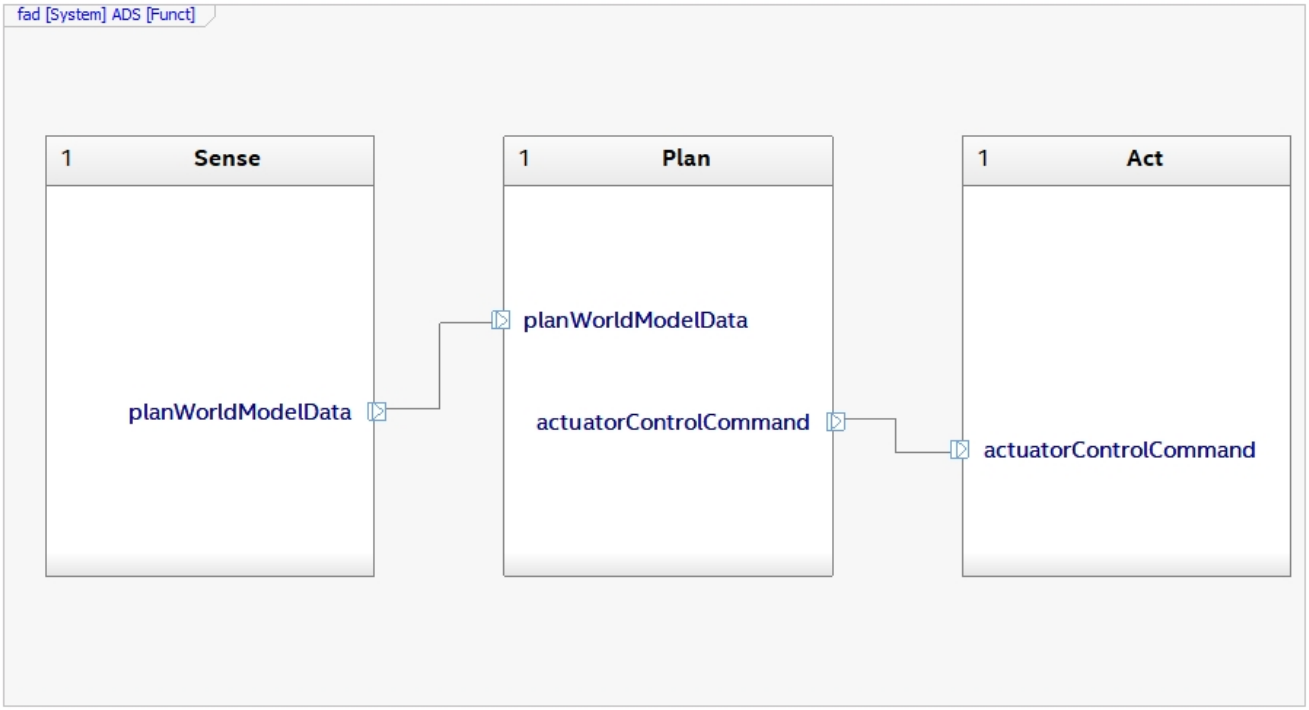

Figure 12. A generic system architecture for an ADS.

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.

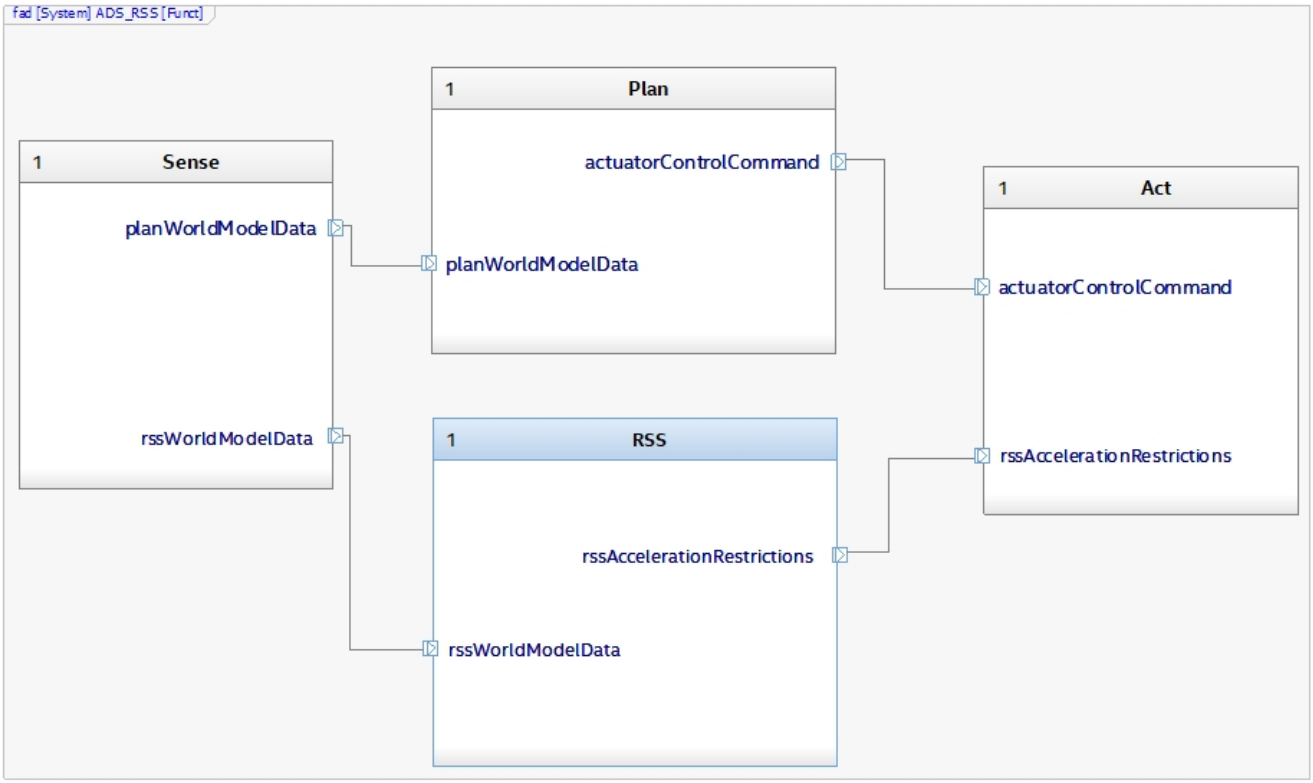

Figure 13. Integration of RSS into a system architecture

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.

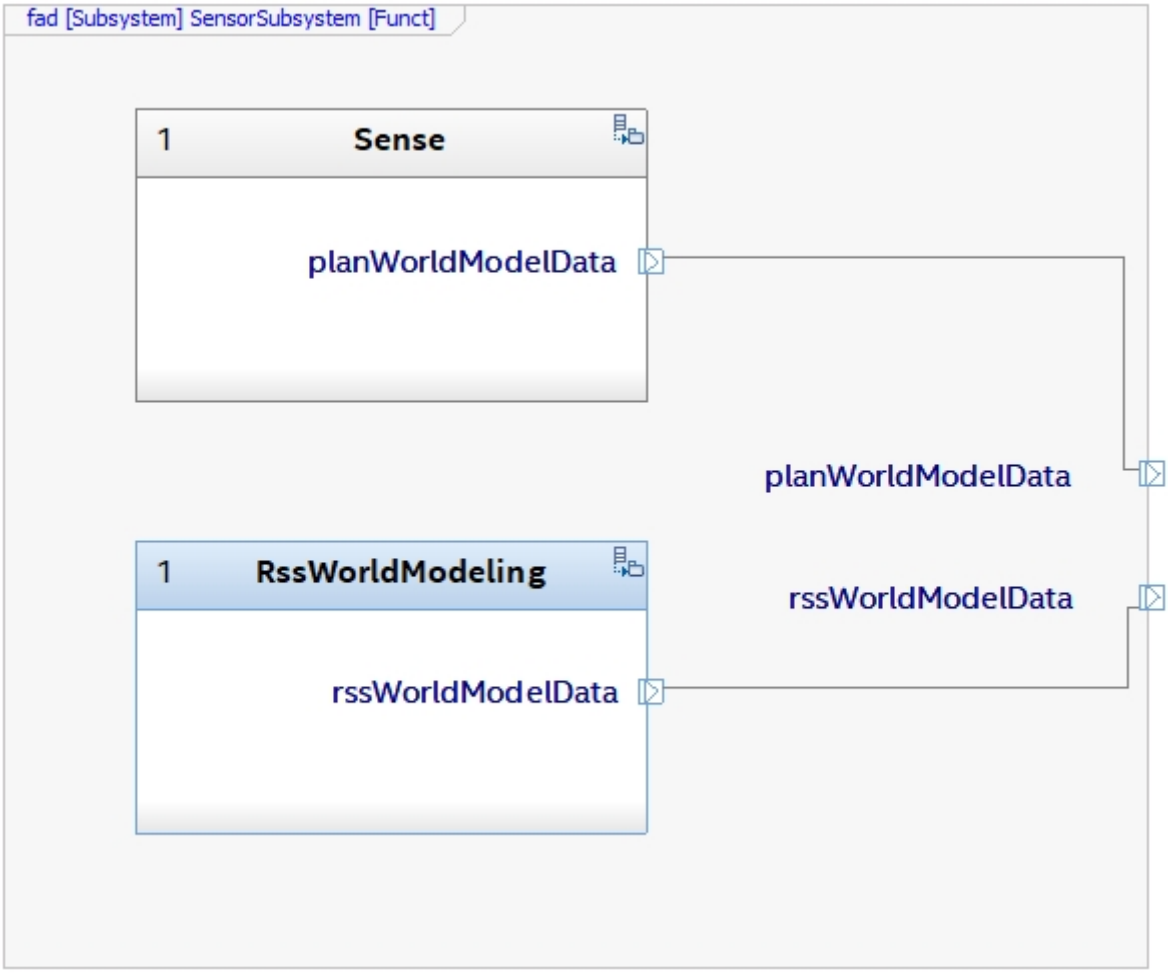

Figure 14. Extending the Sense part with world model data required by RSS.

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.

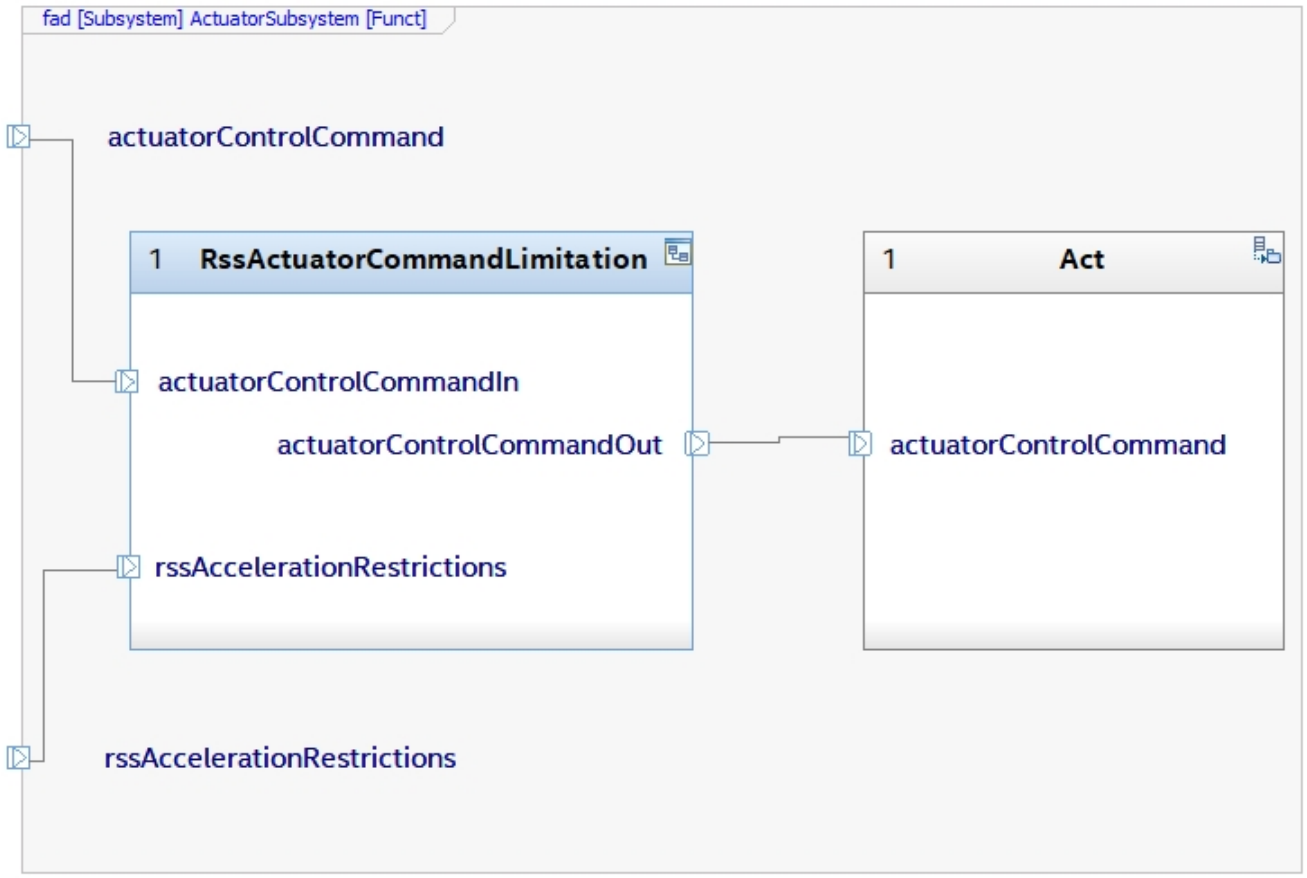

Figure 15. Extending the Act part by RSS command limitation.

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 PlanningSubsytem 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.

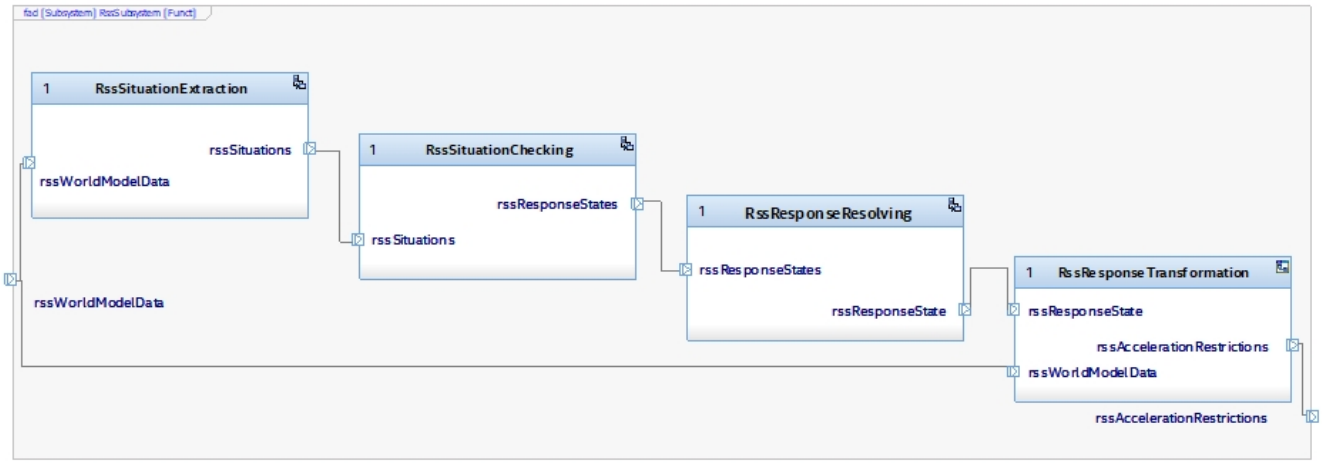

Figure 16. RSS internal processing steps to perform RSS checks and execute the RSS proper response

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

4.1. Static View

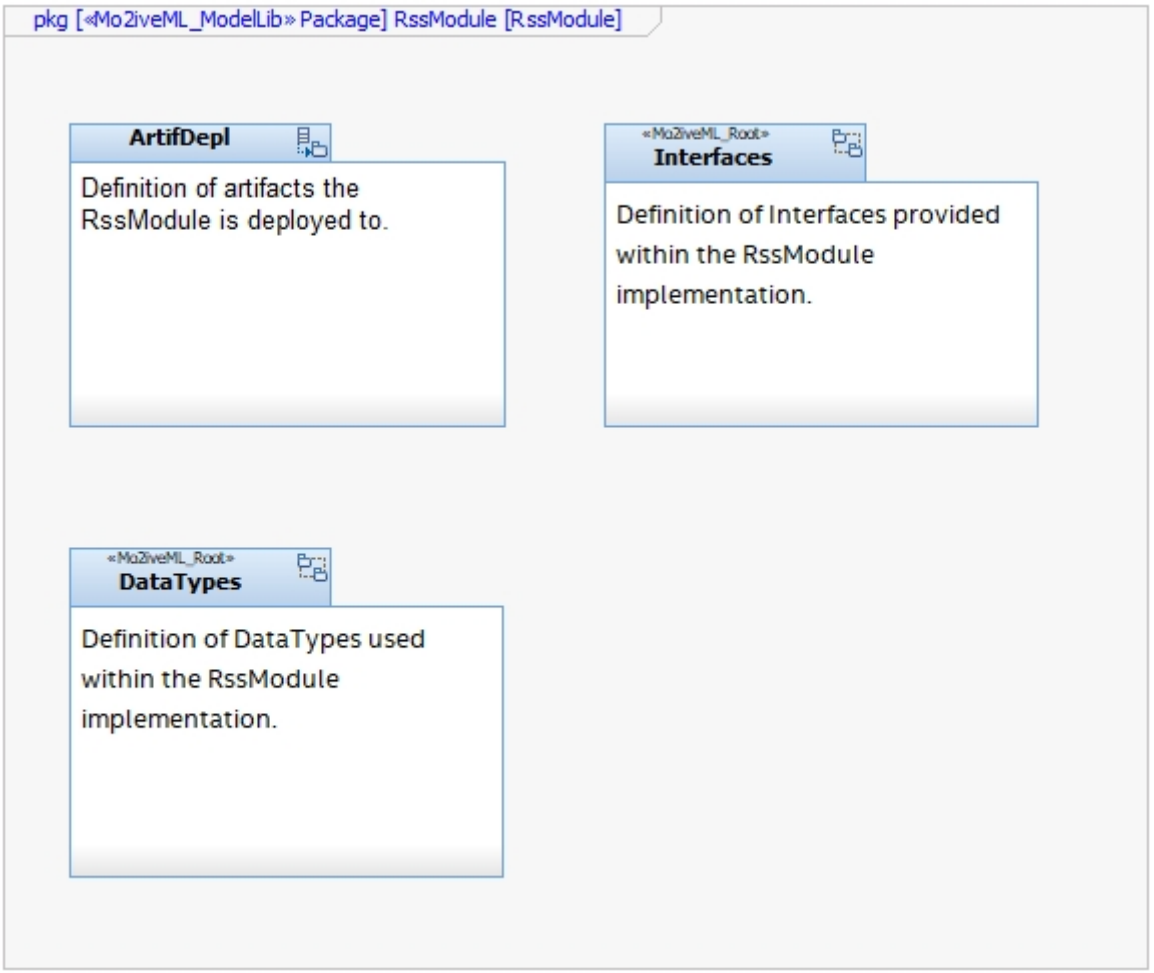

Figure 17. The packages within the RssModule.

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.

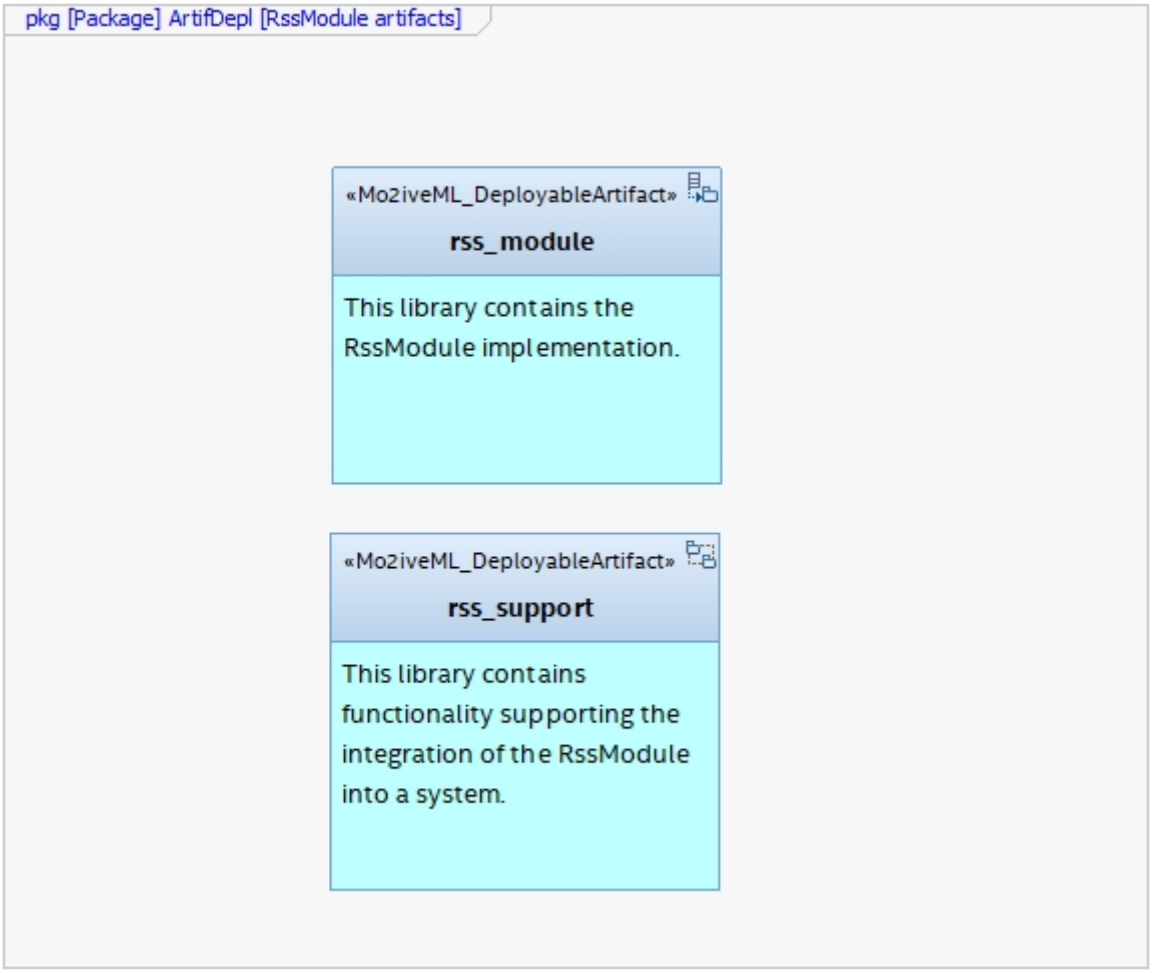

Figure 18. Artifacts to be deployed from the RssModule.

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.

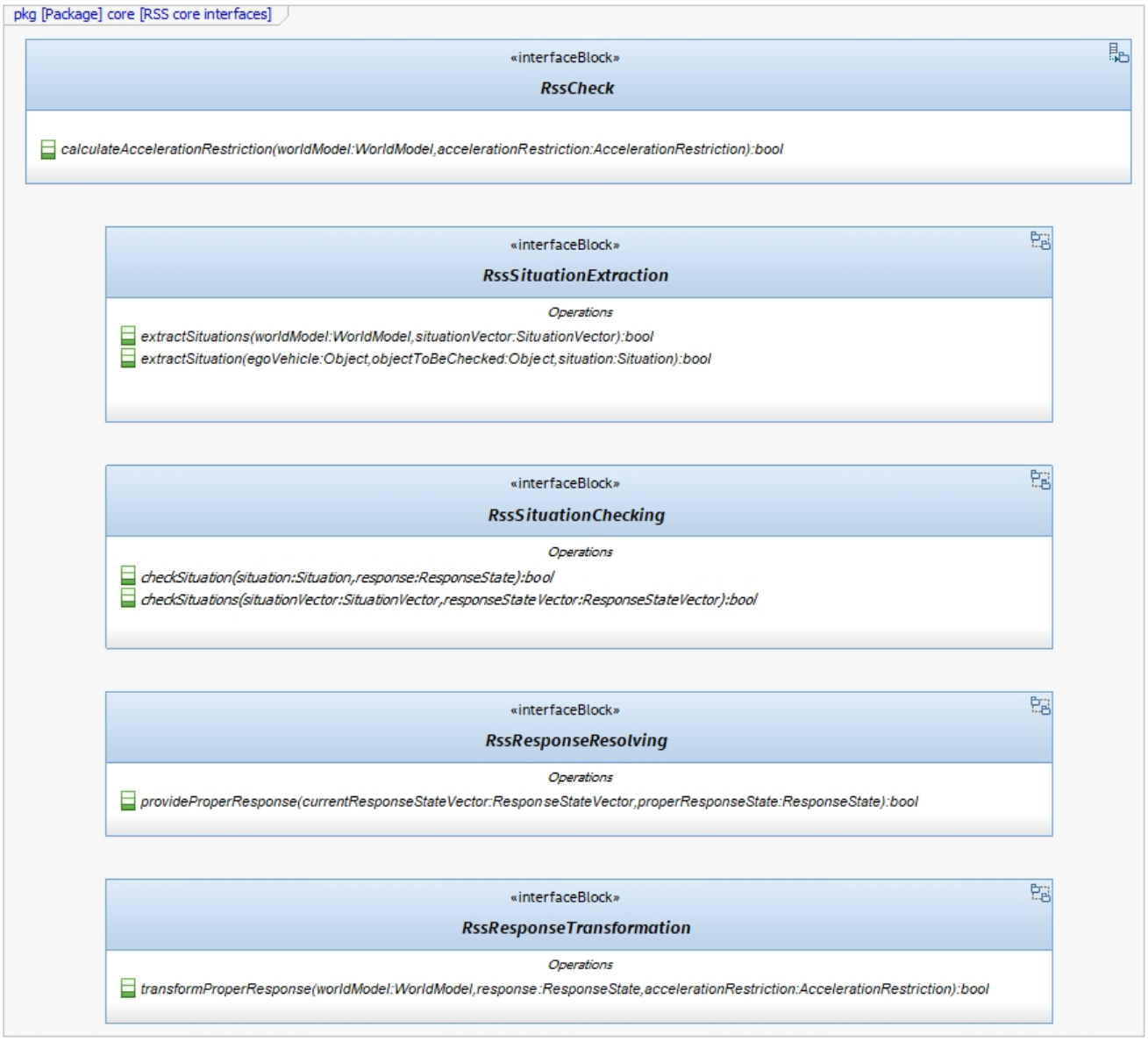

Figure 19. The interfaces provided by the RssModule.

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.

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 blame 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.

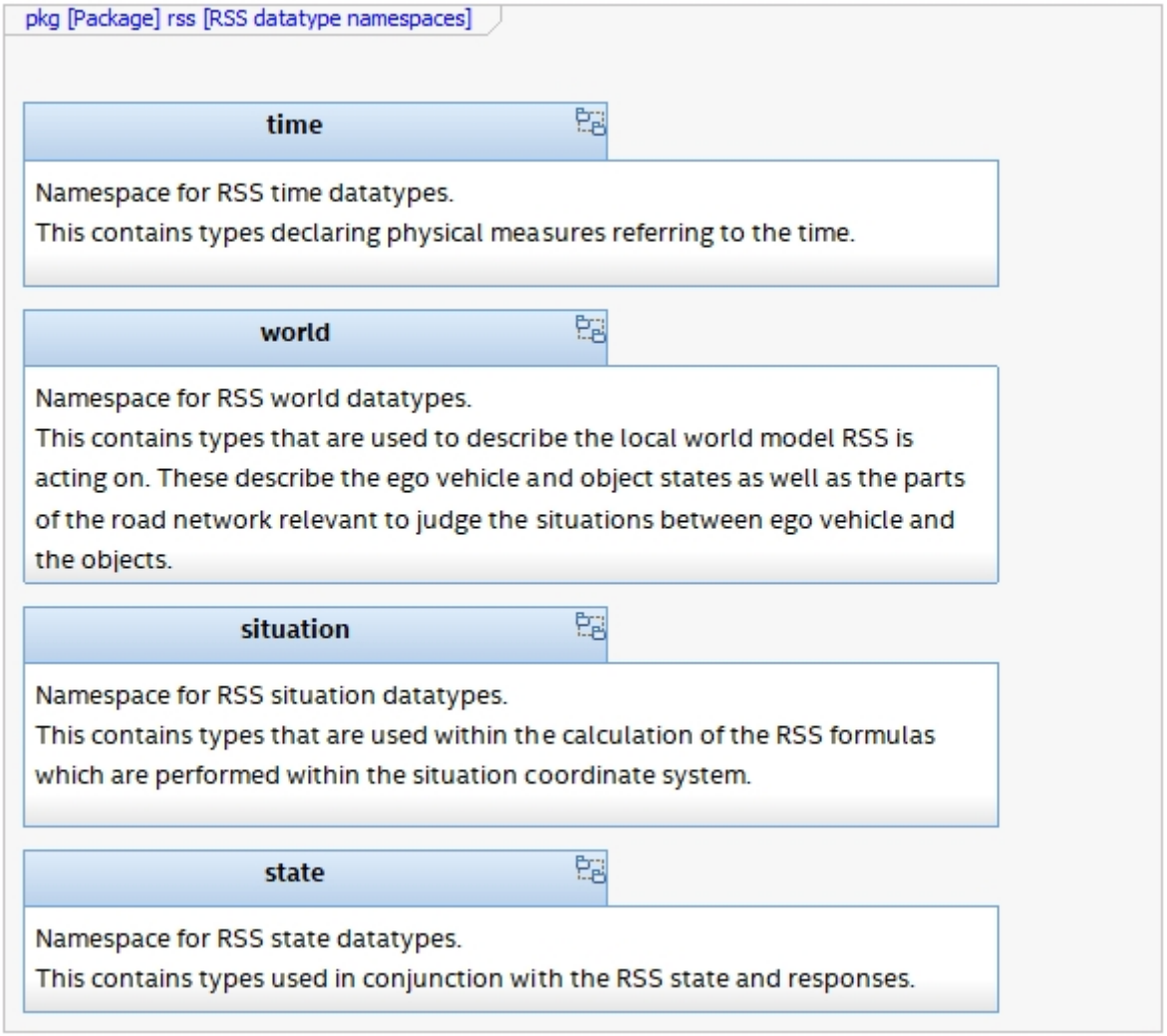

Figure 20. The RSS datatypes are organized within several sub-namespaces.

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.

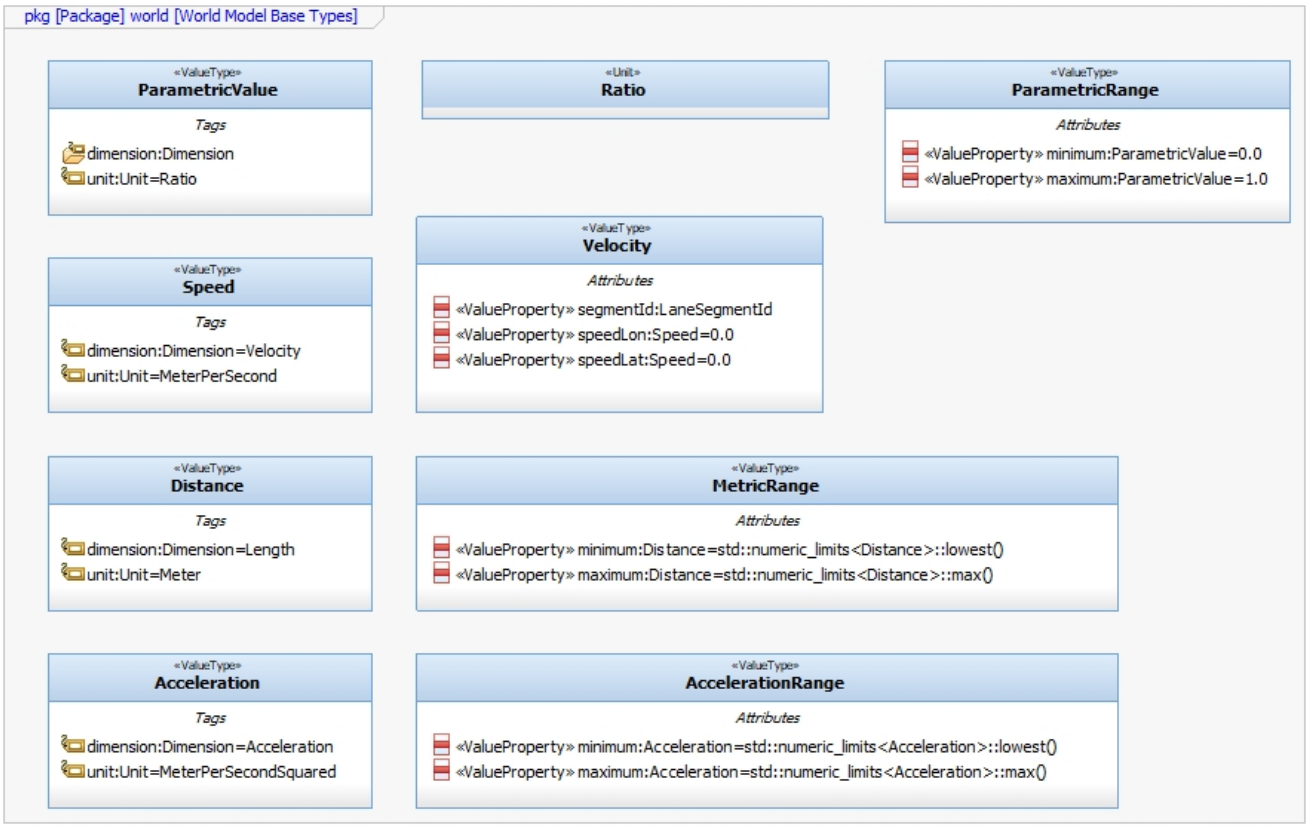

Figure 21. The base types used within the local RSS world model.

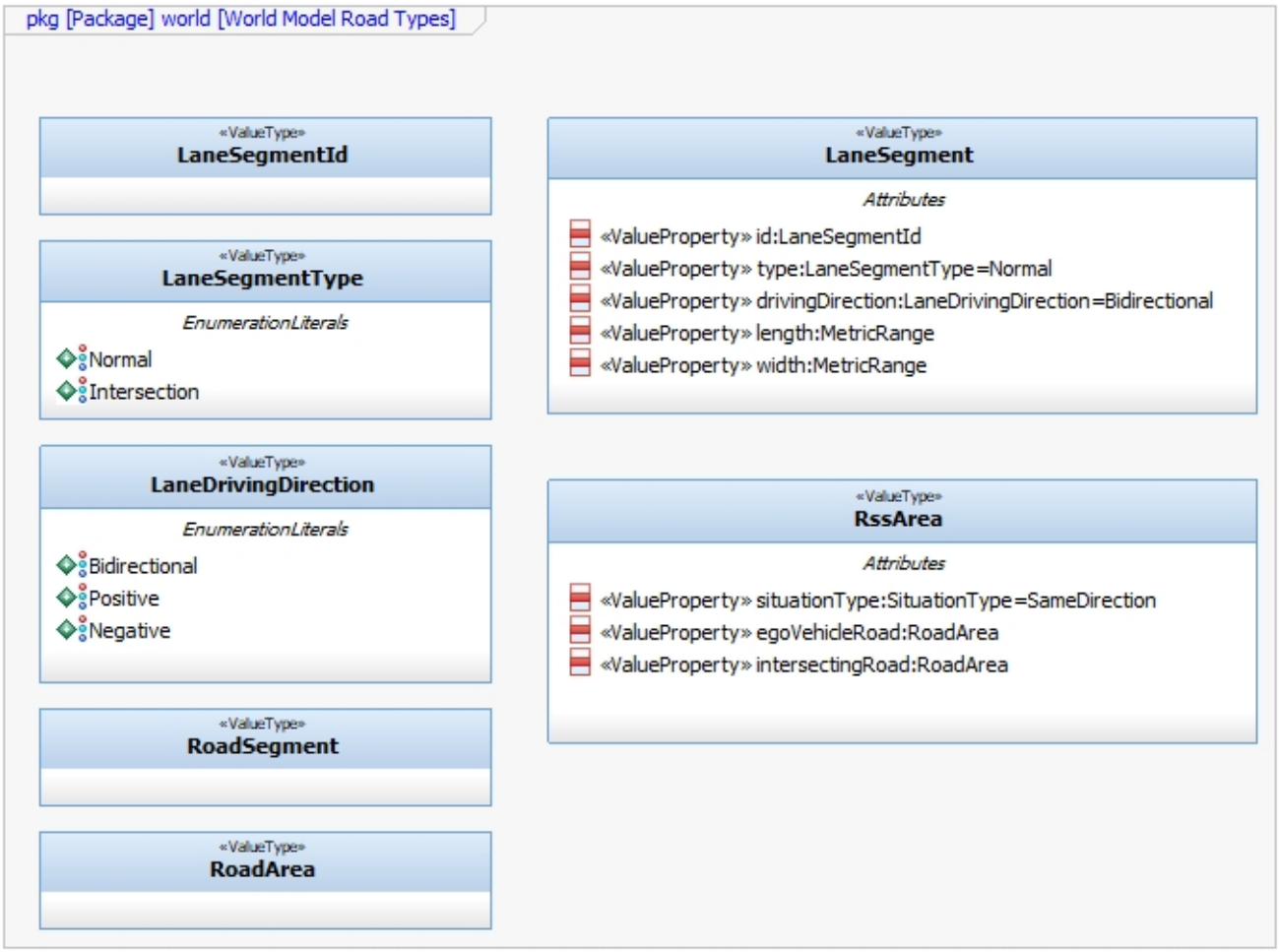

Figure 22. The types describing the road and lane geometries used within the local RSS world model.

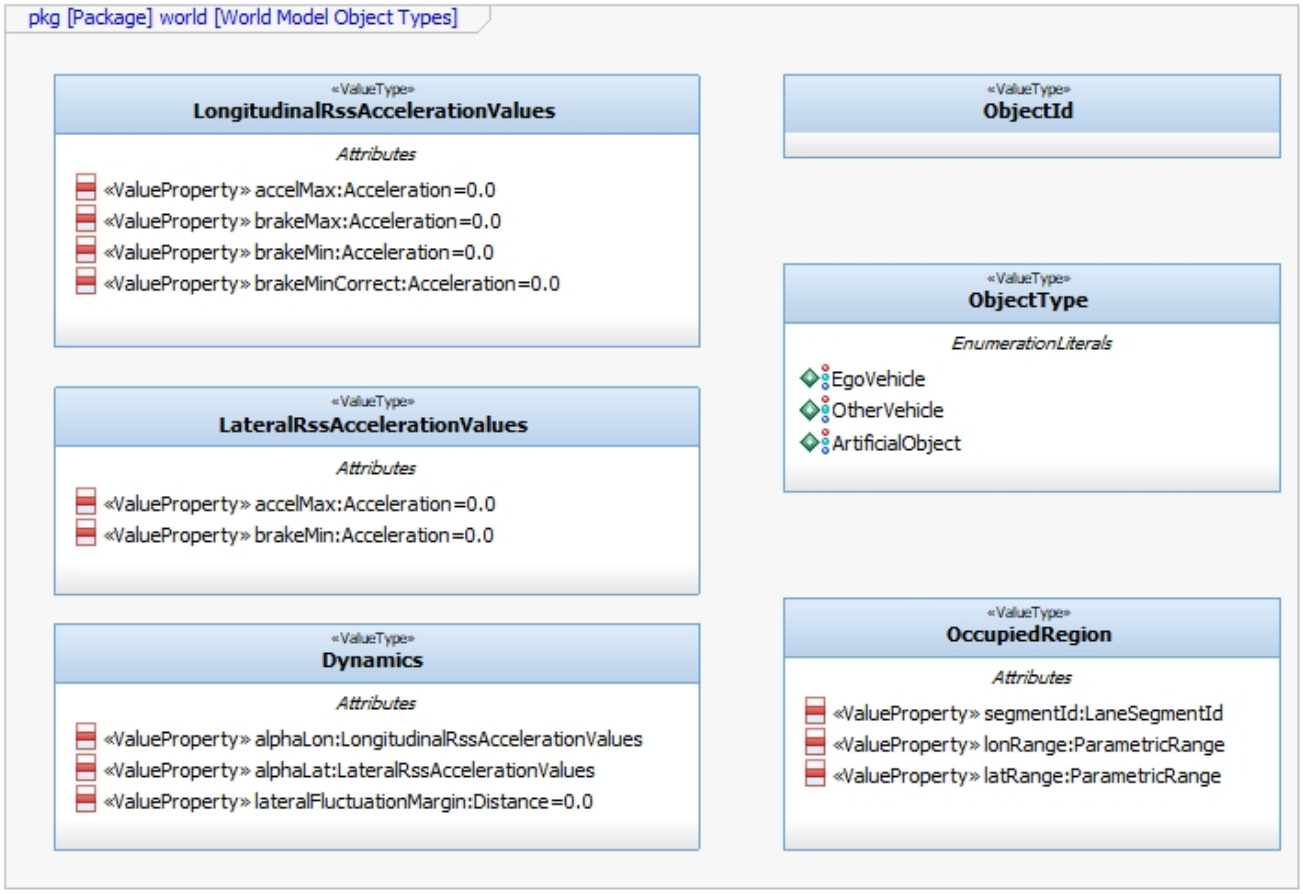

Figure 23. The types describing the objects used within the local RSS world model.

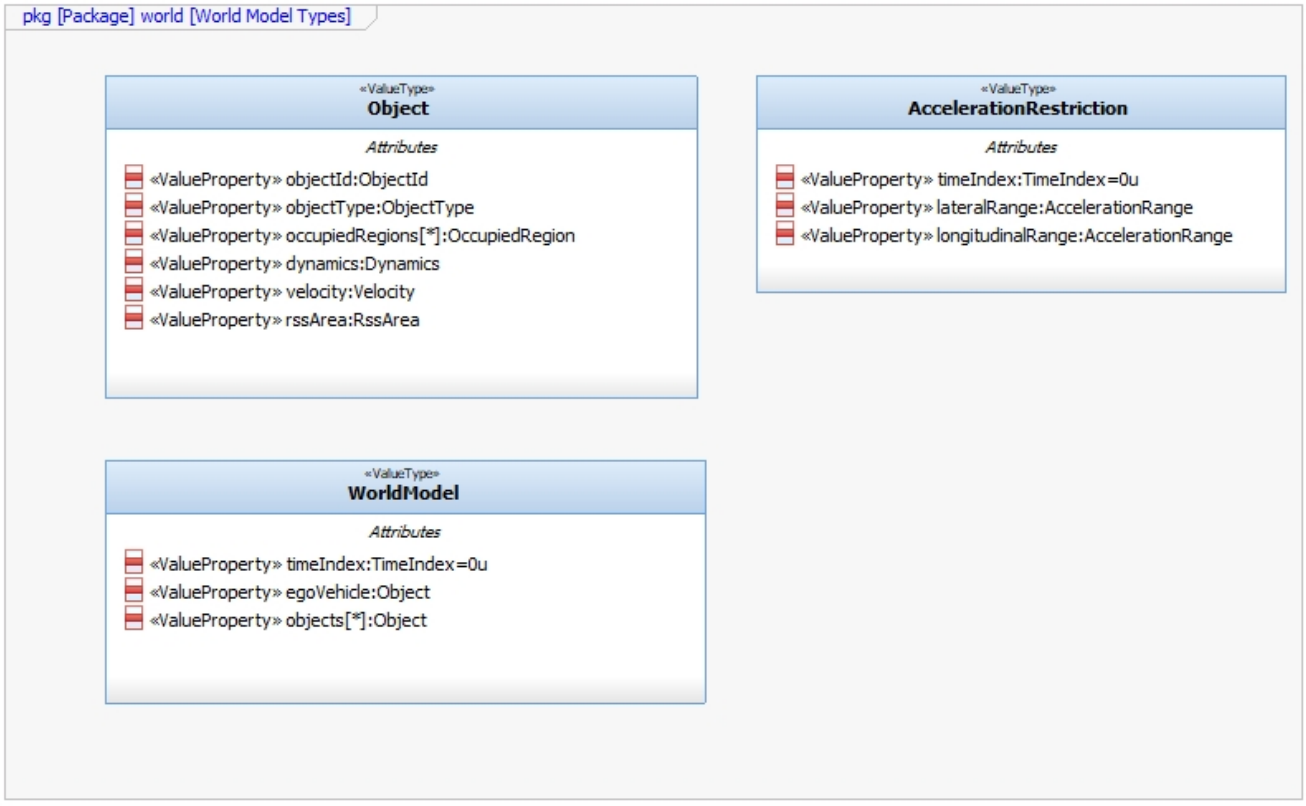

Figure 24. The types describing the high level world model and object used within the local RSS world model.

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.

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 segement. 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. RssArea (Structure)

A RssArea defines the area of interaction 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 neigboring lanes the other vehcile is able to drive in. In non-intersection situations this road area is empty.

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. 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.

4.1.3.2.21. 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.

4.1.3.2.22. 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.

4.1.3.2.23. 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	OccupiedRegion	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. Furthermore this allows to introduce artificial objects for different purposes e.g. to respect occluded regions or to create artificial repulsive objects at the outer road borders to prevent the ego vehilce from leaving the road.
velocity	Velocity	Defines the objects velocity in respect to its current major lane.

rssArea	RssArea	Defines the area of interaction between ego vehicle and the object.

4.1.3.2.24. 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.
objects	Object	The objects within the scene.

4.1.3.2.25. AccelerationRestriction (Structure)

Defines restrictions of the vehicle acceleration.

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

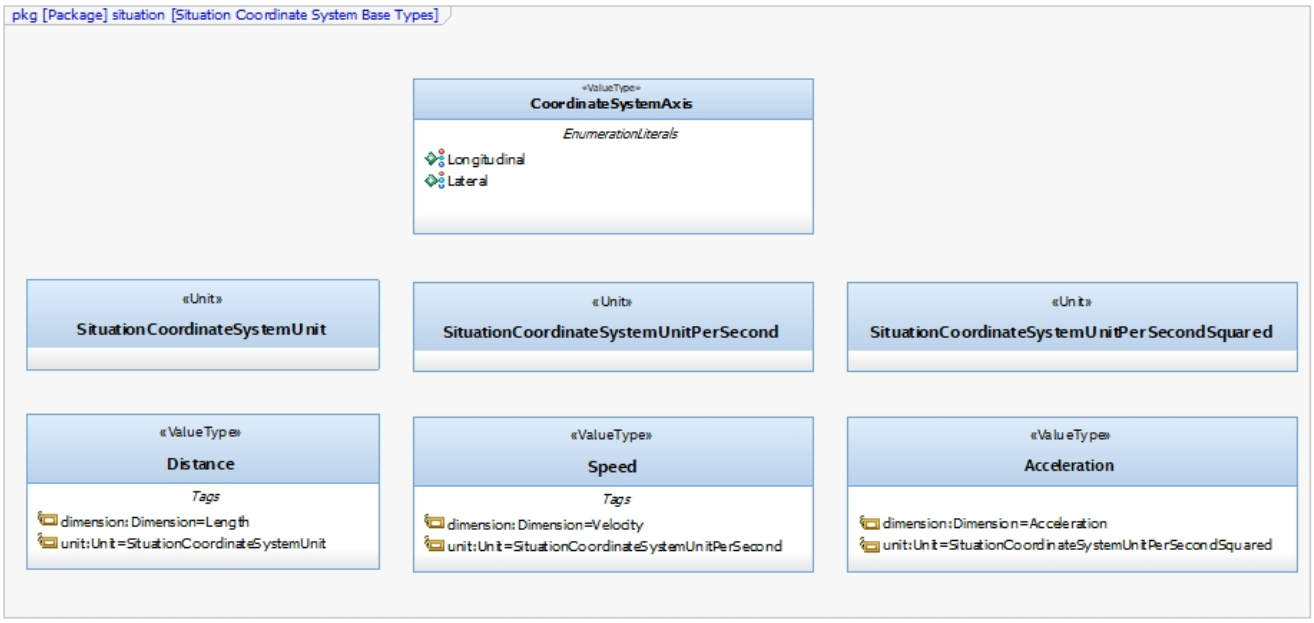

Figure 25. The base types used within the situation coordinate system.

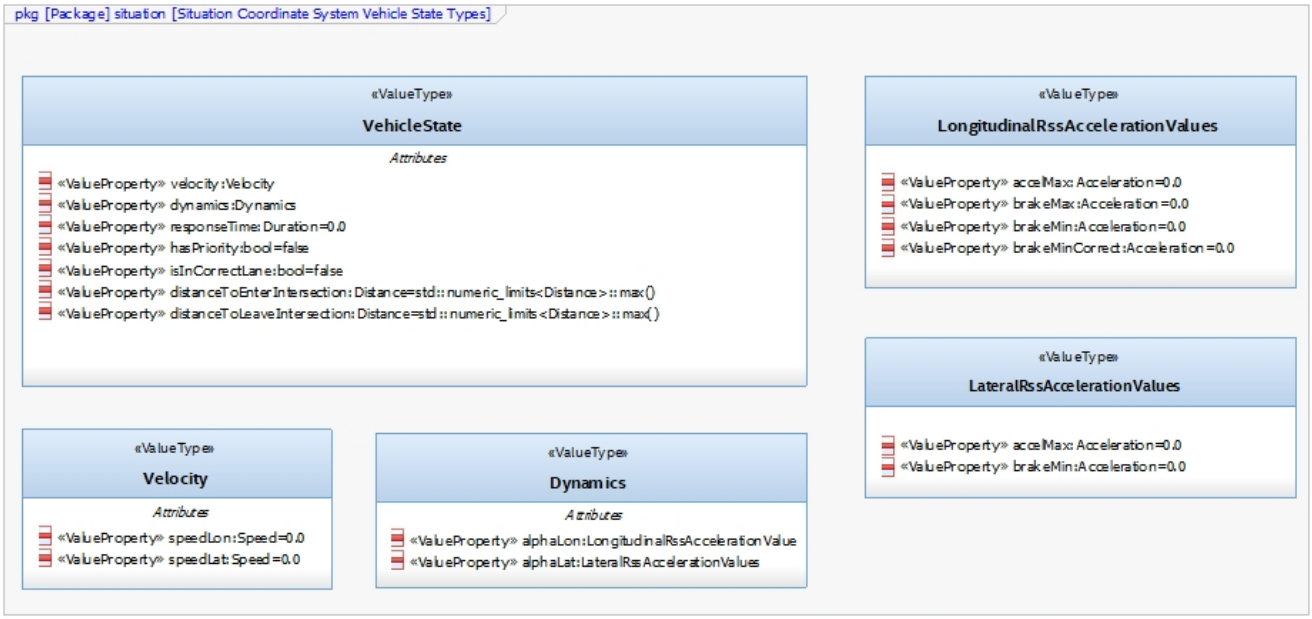

Figure 26. The types describing the vehicle state used within the situation coordinate system.

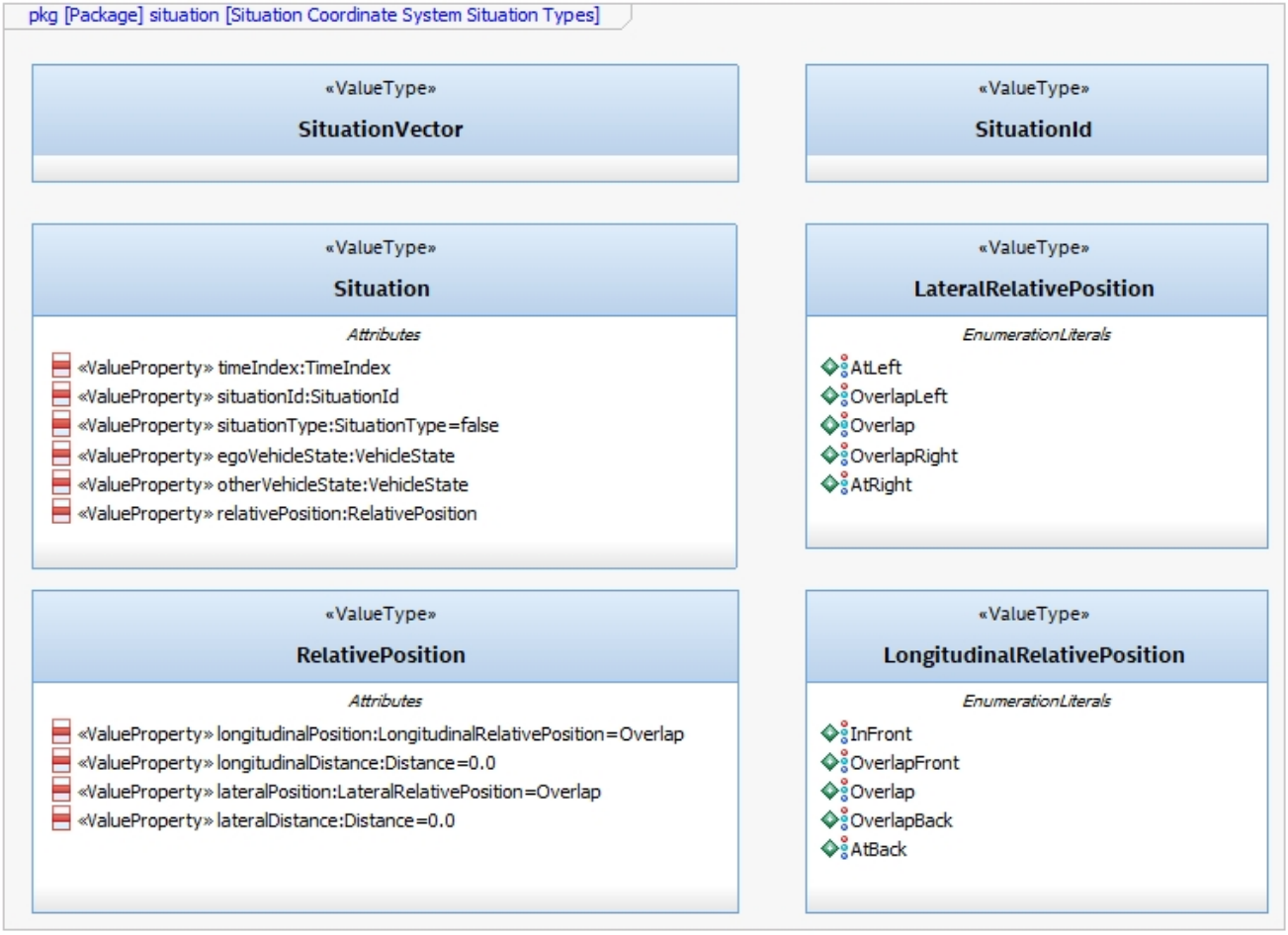

Figure 27. The types describing a situation used within the situation coordinate system.

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.

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 to 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 to 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.

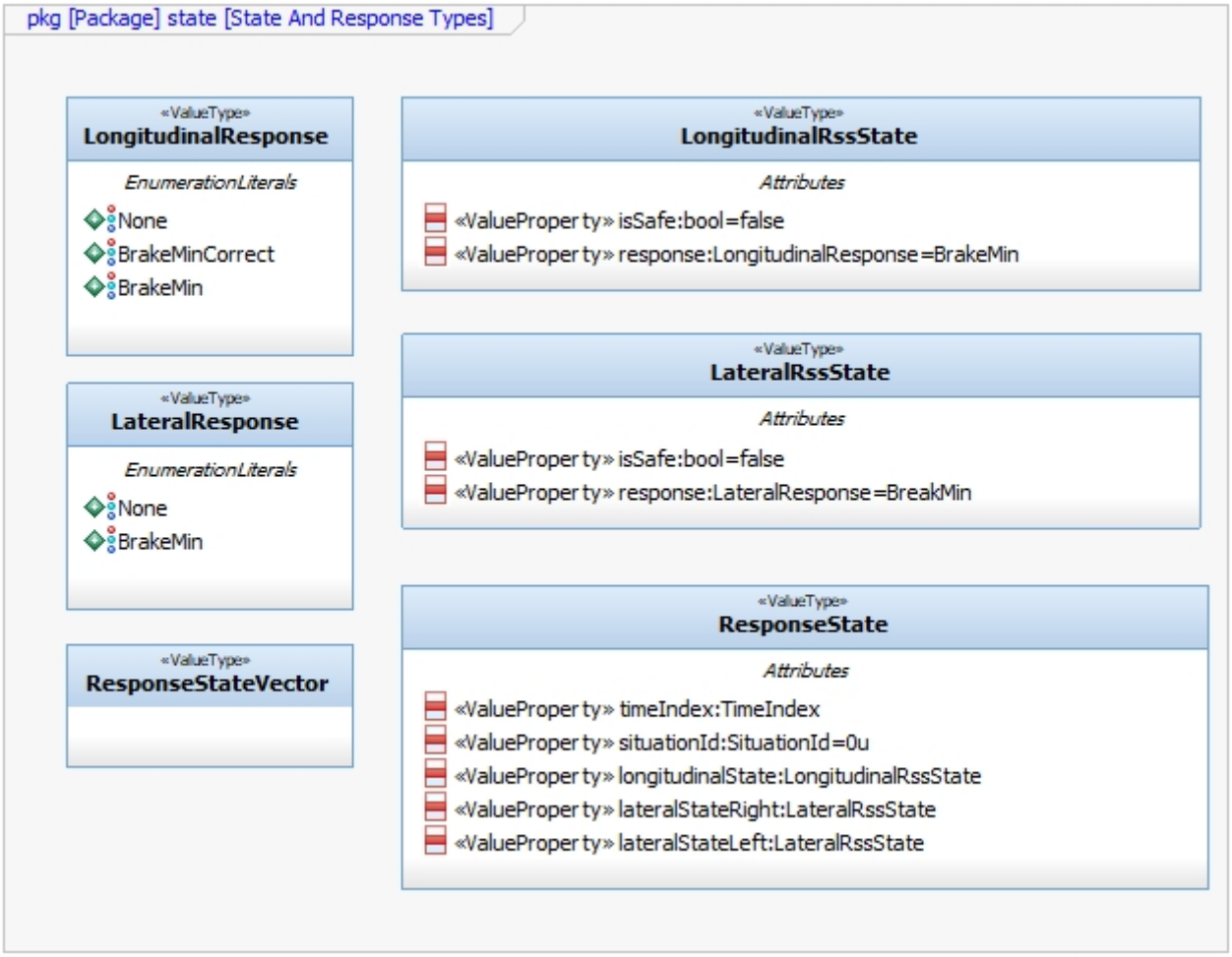

Figure 28. The types describing the RSS response and the RSS state.

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 decerate at least with brake min correct longitudinally
BrakeMin	2	Vehicle has to decerate 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 decerate 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 blame 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

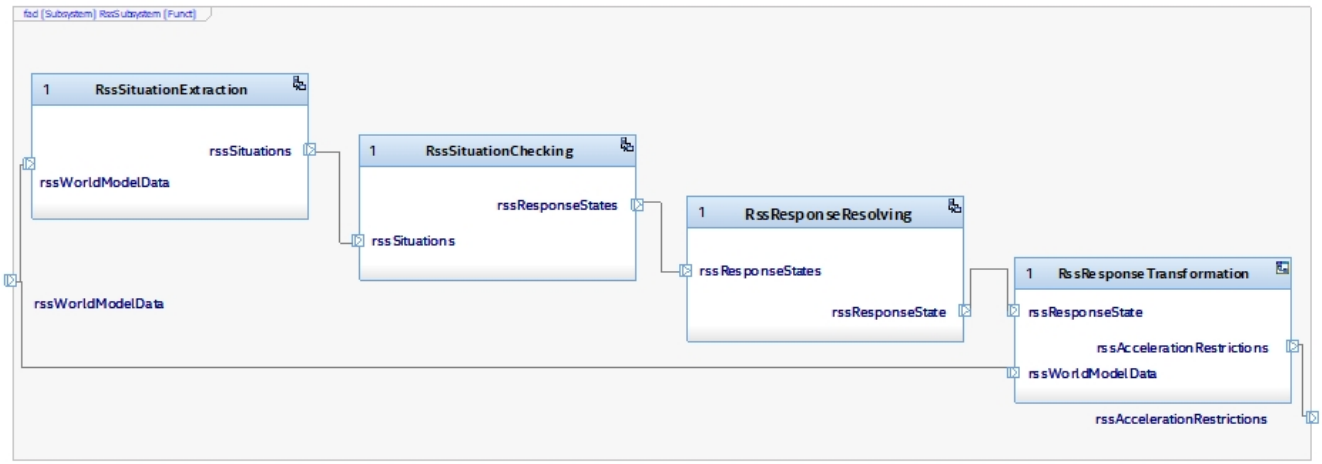

Figure 29. RSS internal processing steps to perform RSS checks and execute the RSS proper response

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.

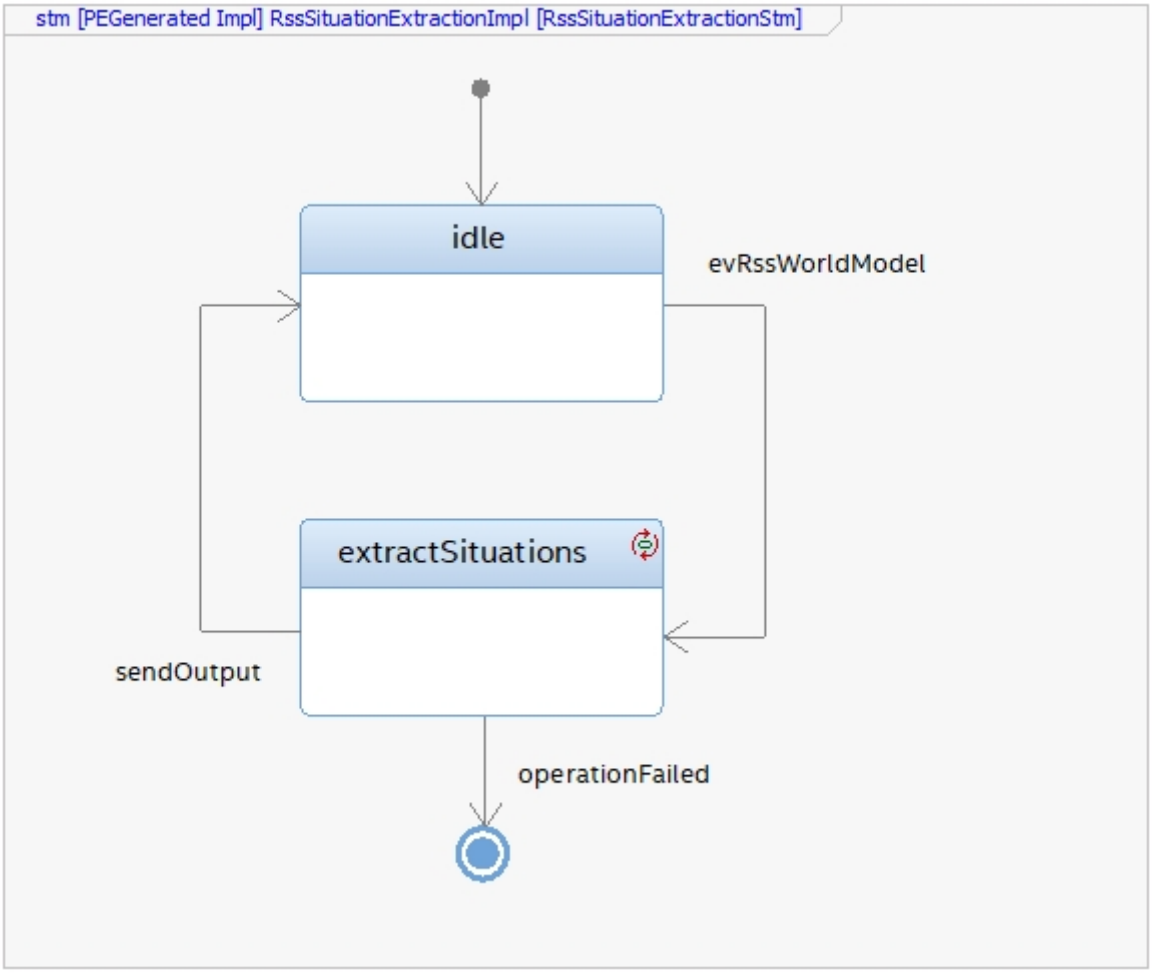

Figure 30. Statechart Diagram describing the dynamic behavior of the RssSituationExtraction entity.

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 trigged when RssWorldModel data is received at the input.
operationFailed	string	errorString	Triggered if the operation failed.
sendOuput	SituationVector	situationVector	Send out resulting situationVector after situations have been extracted.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.
→	evRssWorldModel	extractSituations	WorldModel data received.

State	Transition	Target State	Description
extractSituations			All input data received. Extract situations and trigger send situation vector.
→	sendOutput	idle	All data is processed; send output data.
→	operationFailed	error	The operation failed.
error			Final error state.

4.2.2. RssSituationCheckingImpl

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

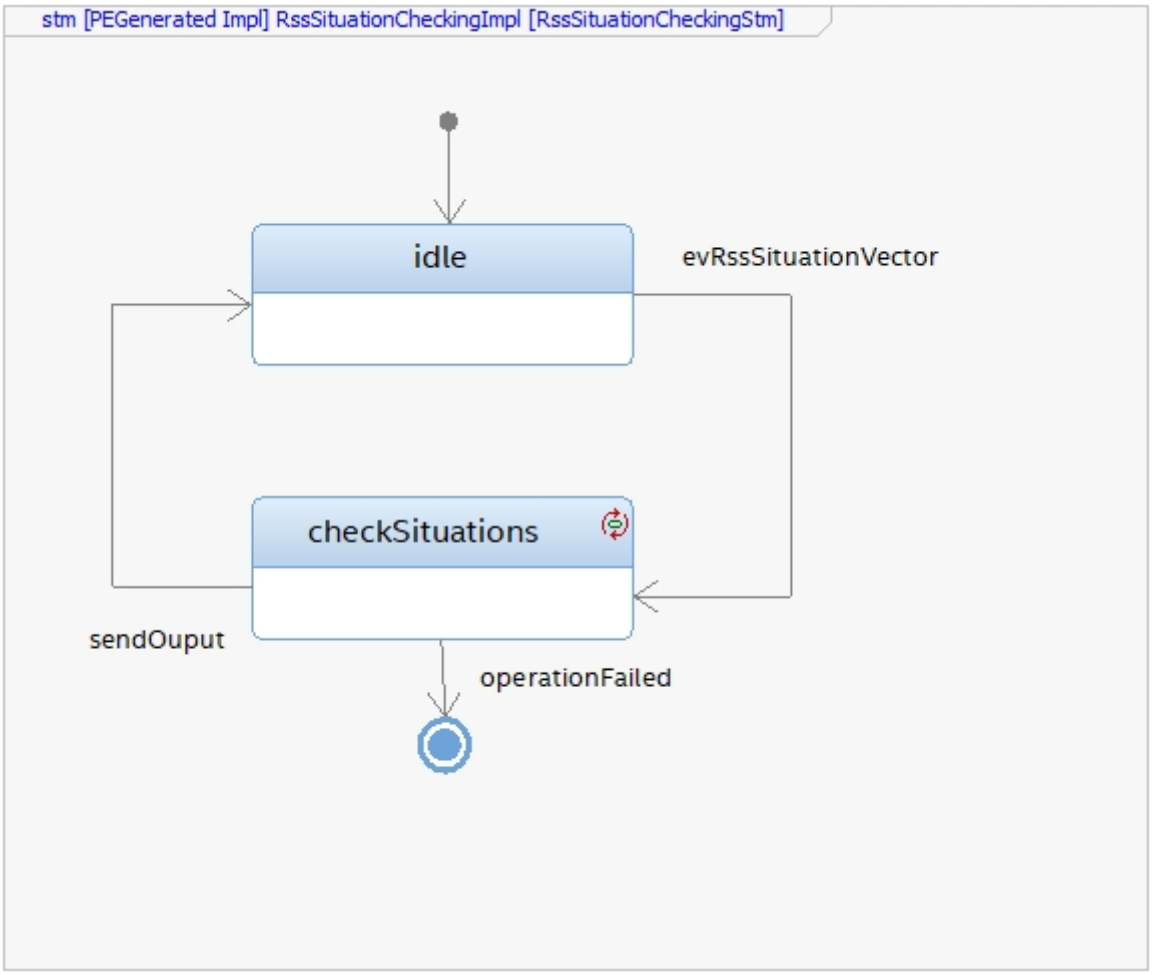

Figure 31. Statechart Diagram describing the dynamic behavior of the RssSituationChecking entity.

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 trigged when RssSituationVector data is received at the input.
operationFailed	string	errorMessage	Triggered if the operation failed.
sendOuput	ResponseStateVector	responseStateVector	Send out resulting responseStateVector after situations have been checked.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.
→	evRssSituationVector	checkSituations	SituationVector data received.
checkSituations			All input data received. Check situations and trigger send RSS state vector.
→	sendOuput	idle	All data is processed; send output data.
→	operationFailed	error	The operation failed.
error			Final error state.

4.2.3. RssResponseResolvingImpl

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

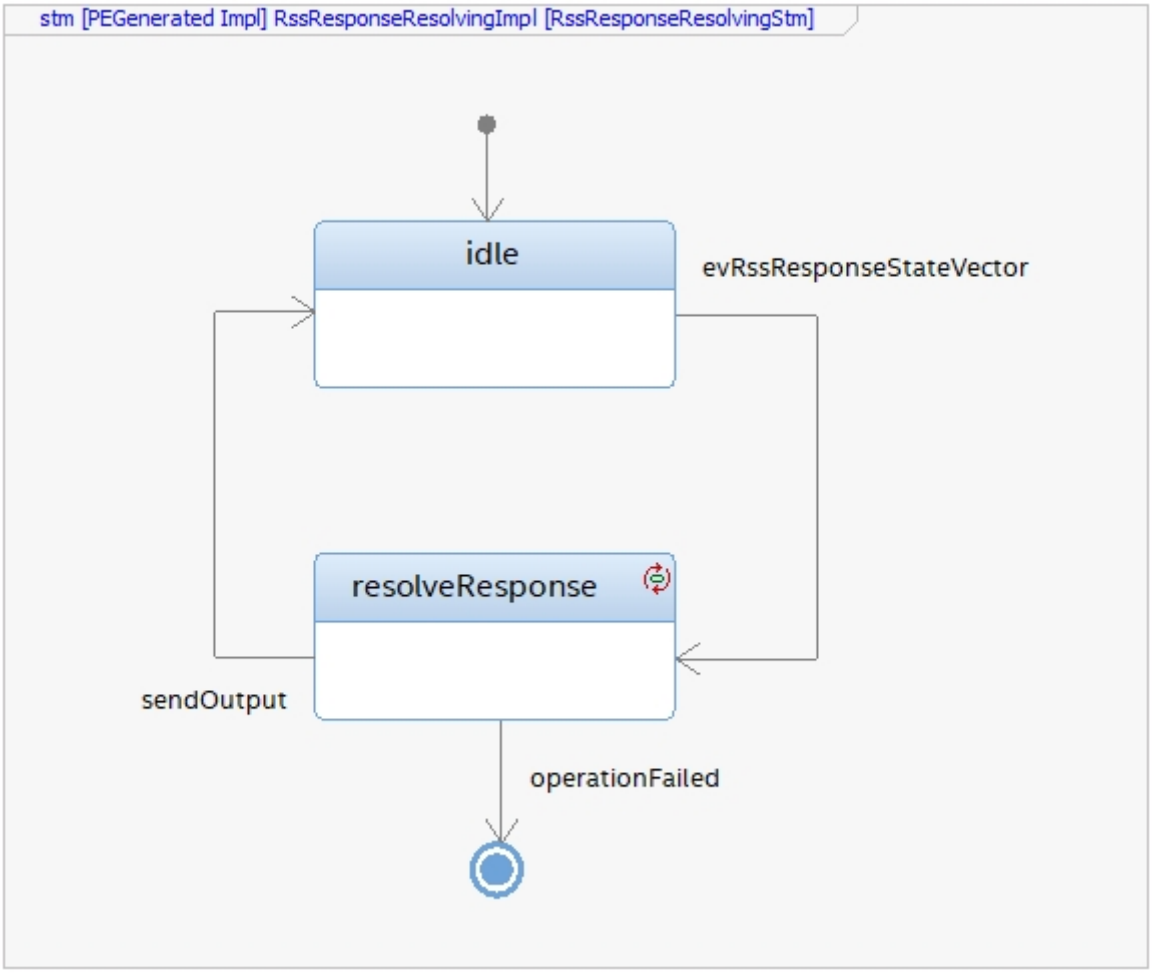

Figure 32. Statechart Diagram describing the dynamic behavior of the RssResponseResolving entity.

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

Event/Trigger	Argument Type	Argument	Description
evRssResponseStateVector	ResponseStateVector	responseStateVector	Event trigged when RssResponseStateVector data is received at the input.
sendOutput	ResponseState	properResponse	Send out resulting properResonse after response states have been resolved.
operationFailed	string	errorMessage	Triggered if the operation failed.

State	Transition	Target State	Description
idle			Idle state. Waiting for input data.
→	evRssResponseStateVector	resolveResponse	ResponseStateVector data received.

State	Transition	Target State	Description
resolveResponse			All input data received. Resolve responses and trigger send proper response.
→	sendOutput	idle	All data is processed; send output data.
→	operationFailed	error	The operation failed.
error			Final error state.

4.2.4. RssResponseTransformationImpl

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

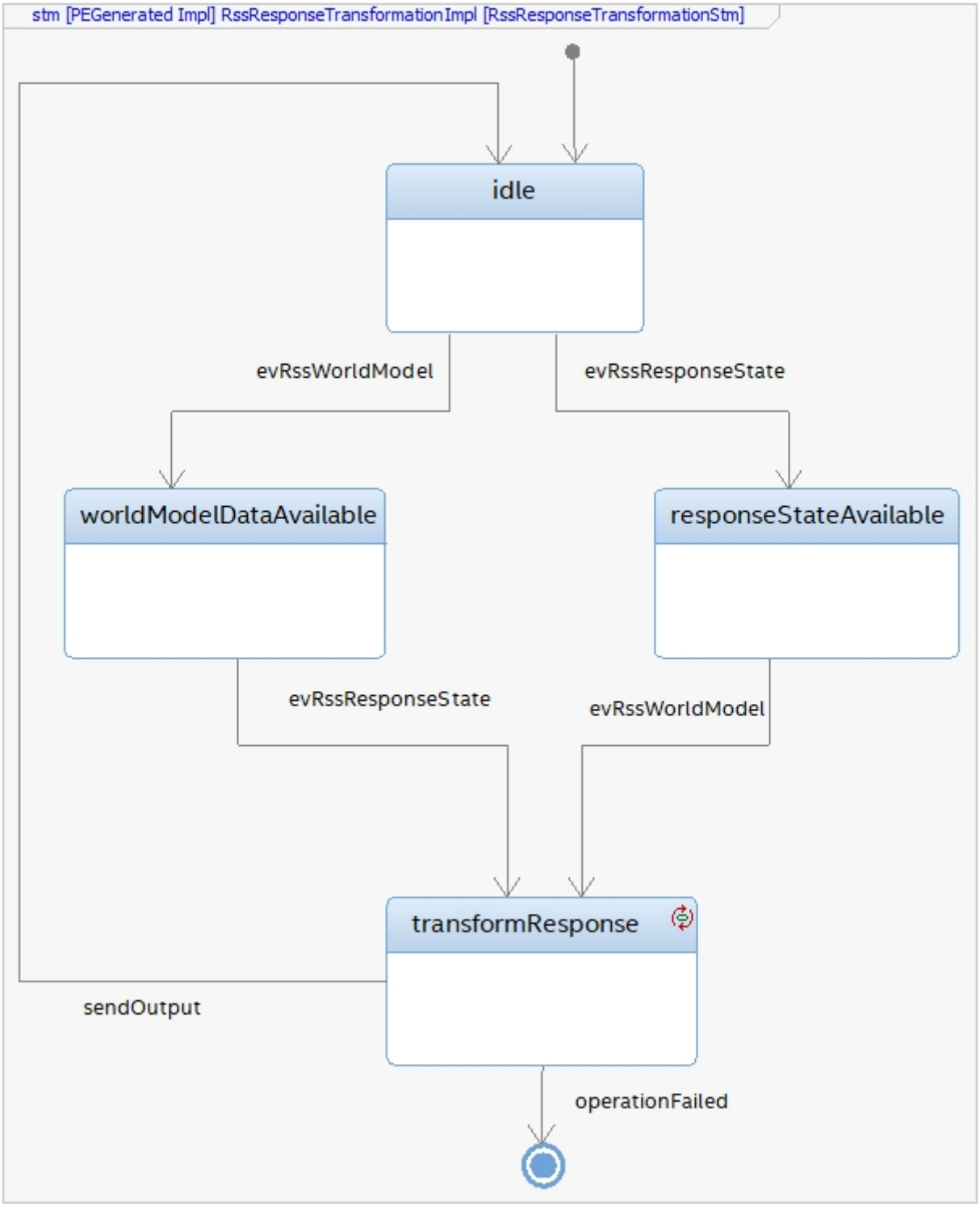

Figure 33. Statechart Diagram describing the dynamic behavior of the RssResponseTransformation entity.

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

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

Event/Trigger	Argument Type	Argument	Description
evRssWorldModel	WorldModel	worldModel	Event trigged when RssWorldModel data is received at the input.
operationFailed	string	errorMessage	Triggered if the operation failed.
sendOutput	AccelerationRestriction	acellerationRestriction	Send out resulting accelerationRestriction after response has been transformed.

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. Transform response and trigger send accelerator restrictions.
→	sendOutput	idle	All data is processed; send output data.
→	operationFailed	error	The operation failed.
error			Final error state.

Chapter 5. Design for Security

To be discussed

Chapter 6. Design for Safety

TODO Maybe we should explain here the differentiation between Functional Safety (FuSa) in general and the planning safety which RSS is covering.

Basic idea would be: In principle, RSS is implementing safety of the intended functionality. One can apply RSS without FuSa to safeguard the planning functionality. If one implements RSS 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
  - $\alpha^{lat}_{brake,min}$ minimum allowed braking deceleration in lateral direction
  - $\alpha^{lat}_{accel,max}$ maximum allowed acceleration in lateral direction
  - δ^{lat}_{min} 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, $\rho_{other} = 2\ s$ and $\rho_{ego} = 1\ s$. 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 34). A rule of thumb for deceleration in German driving schools is: $\alpha_{brake,min} = 4\ m/s^2$ and $\alpha_{brake,max} = 8\ m/s^2$

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

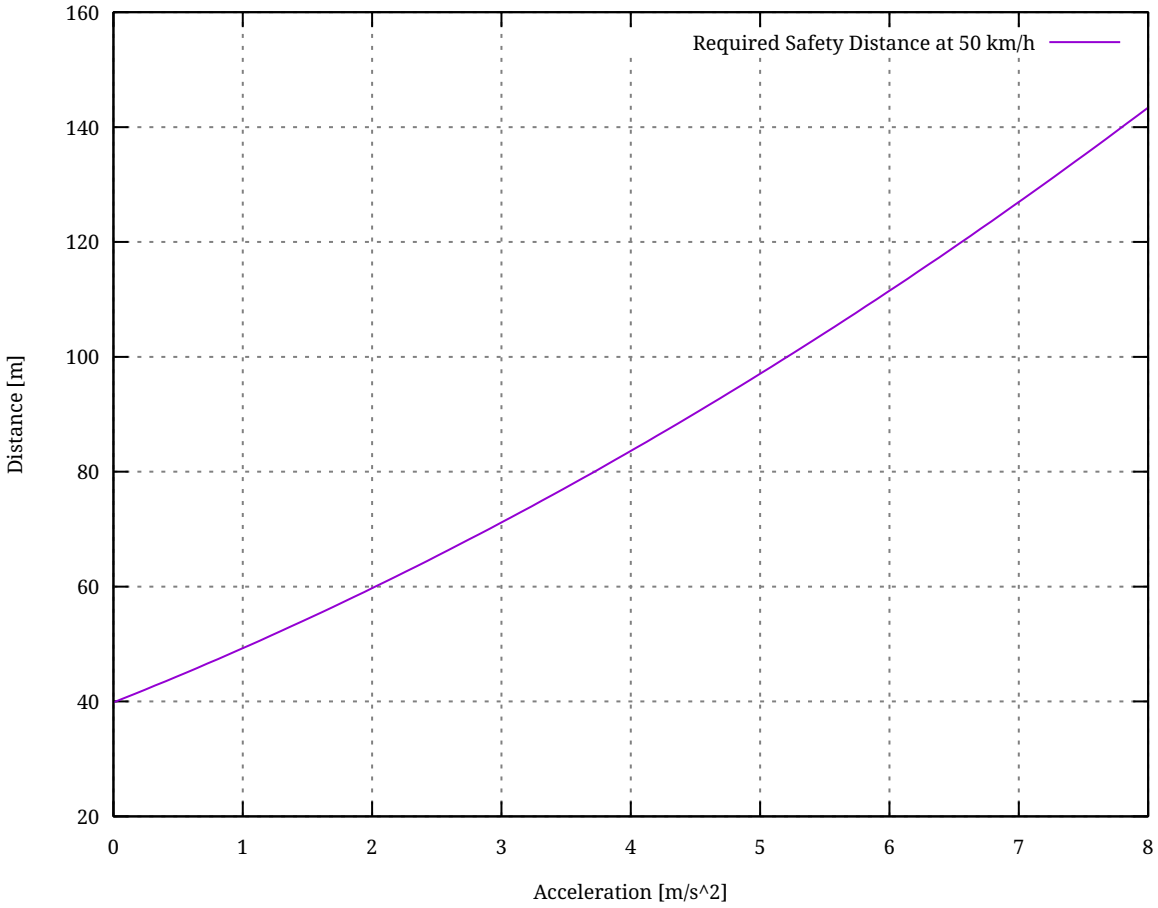

Figure 34. Required safety distance for cars driving at 50 km/h (city speed) in same direction with $\alpha_{brake,min}$ = 4 m/s^2 and $\alpha_{brake,max}$ = 8 m/s^2 and $\rho_{ego} = \rho_{other}$ = 2 s

The assumption that a car can always accelerate at $\alpha_{accel,max}$ during the reponse time, leads to a significant increase of the required safety distance. Figure 34 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 import 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.

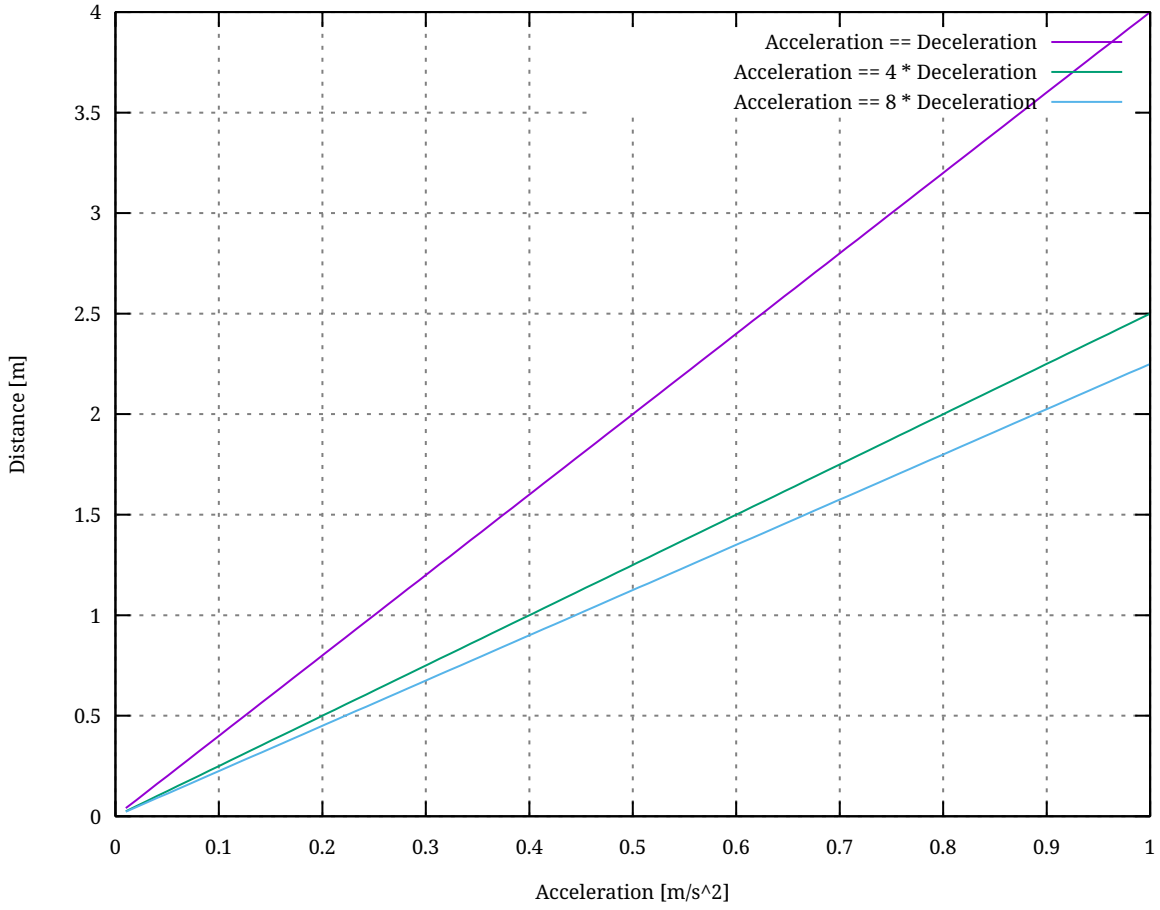

Figure 35. Distance a vehicle will cover when applying the "Stated Braking Pattern" with $\rho_{vehicle}$ = 2 s

Figure 35 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 ${a^{lat}}_{accel,max} < 0.1\ m/s^2$. 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. ${a^{lat}}_{brake,min} = 0.8\ m/s^2$ and ${a^{lat}}_{accel,max} = 0.2\ m/s^2$ 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^{lat}_{min}. As this should only cover for fluctuations, there is also no need for a huge margin. Thus initially this value will be set to $\delta^{lat}_{min} = 10\ cm$. 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}	$2\ s$
$\alpha_{accel,max}$	$3.5\ m/s^2$
$\alpha_{brake,min}$	$4\ m/s^2$
$\alpha_{brake,max}$	$8\ m/s^2$
$\alpha_{brake,min,correct}$	$3\ m/s^2$
$\alpha^{lat}_{brake,min}$	$0.8\ m/s^2$
$\alpha^{lat}_{accel,max}$	$0.2\ m/s^2$
δ^{lat}_{min}	$10\ cm$

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	v5	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. Build RSS core library and documentation

10.1. 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
```

10.2. 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
```

10.3. 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
```