文本

AI 生成的内容可能不正确。

**自动紧急刹车系统系统**

**Autonomous Emergency Braking**

**姓名：高桥辰宜 王策明 蔡文鼎**

**学号：12235101403 11240408484 10235101574**

**学院：软件工程**

**2025年 7月**

# 一．AEB（自动紧急制动）系统SysML模型

### 第一部分：系统需求分析 (Requirement Diagram)

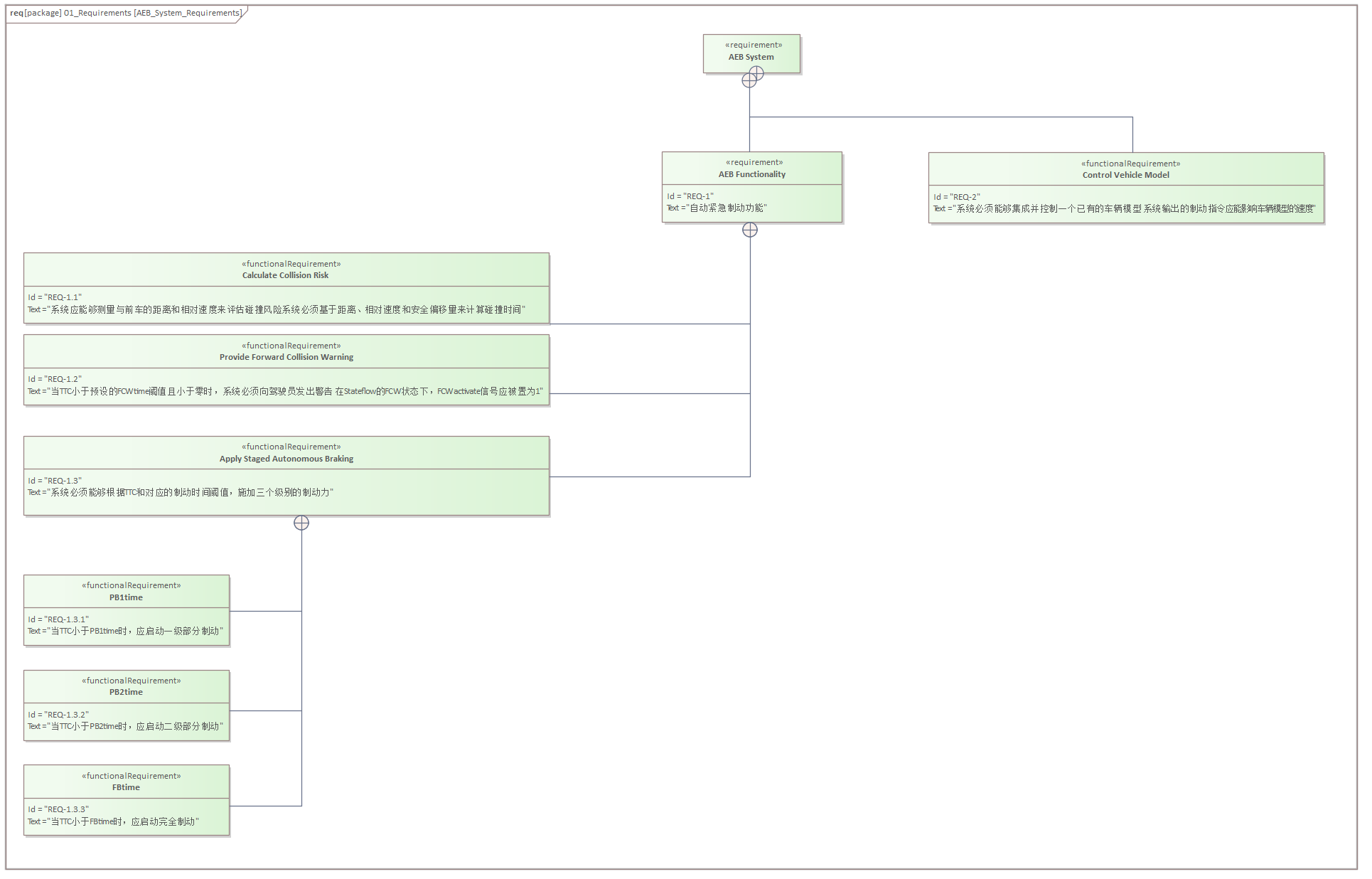
**目标**: “我们要做一个什么样的系统？”

我们利用需求图全面梳理并结构化了AEB系统的功能性与非功能性需求，从而建立了清晰的层次体系。

#### 1.1 需求层次结构

我们的需求模型从最顶层的系统目标AEB System开始，逐级分解：

* **REQ-1: AEB Functionality (AEB功能性)**: 这是核心的功能总览，规定系统必须具备自动紧急制动的核心能力。
* **REQ-2: Control Vehicle Model (控制车辆模型)**: 这是一条关键的约束性需求，规定我们的AEB系统必须能够与一个现有的车辆仿真模型进行集成和控制。



#### 1.2 功能性细化

AEB Functionality被进一步分解为三个关键的子功能，这直接对应了系统的工作流程：

1. **REQ-1.1: Calculate Collision Risk (计算碰撞风险)**:
   * **核心内容**: 系统必须能感知环境，并基于感知数据计算出核心的风险指标——**碰撞时间（Time-to-Collision, TTC）**。这是所有后续决策的基础。
2. **REQ-1.2: Provide Forward Collision Warning (提供前向碰撞预警)**:
   * **核心内容**: 在碰撞风险达到一定程度时（TTC小于FCWtime阈值），系统必须首先向驾驶员发出警告，而不是立即干预。
3. **REQ-1.3: Apply Staged Autonomous Braking (施加分阶段自动制动)**:
   * **核心内容**: 如果驾驶员未响应且风险持续升级，系统必须能够主动进行制动干预。
   * **设计亮点**: 我们将制动行为进一步细化为三个子需求（PB1time, PB2time, FBtime），明确了制动是**分阶段、逐步升级**的，这符合现实世界AEB系统的安全设计原则，避免了突兀的、不舒适的制动体验。

### 第二部分：系统“语言”定义 (Data Types & Interfaces)

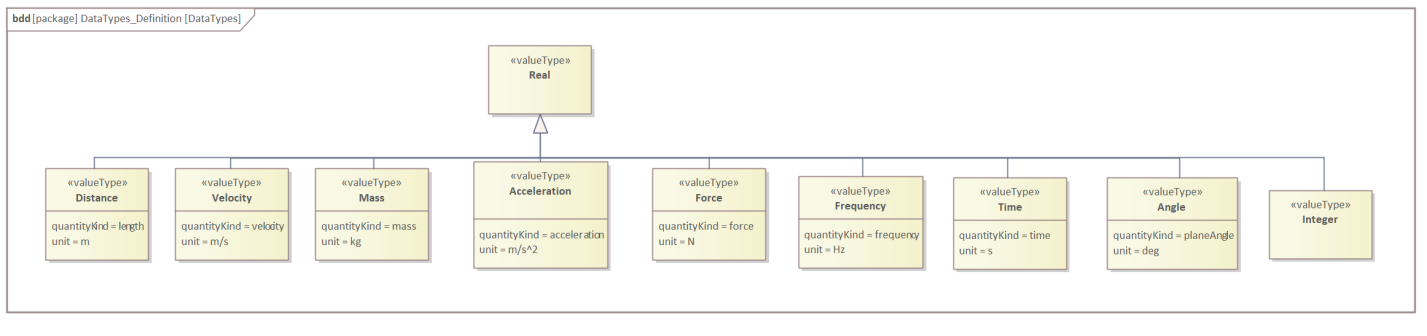
**目标**: 为系统定义一套无歧义的、统一的工程语言。

为了避免混淆（例如，一个工程师用“米”，另一个用“英尺”），我们首先定义了模型中所有数据的基础类型和交互的“契约”。

#### 2.1 数据类型定义 (DataTypes BDD)

我们没有直接使用Real或Integer这样的原始类型，而是创建了一系列带有明确物理单位的**值类型 (ValueType)**。

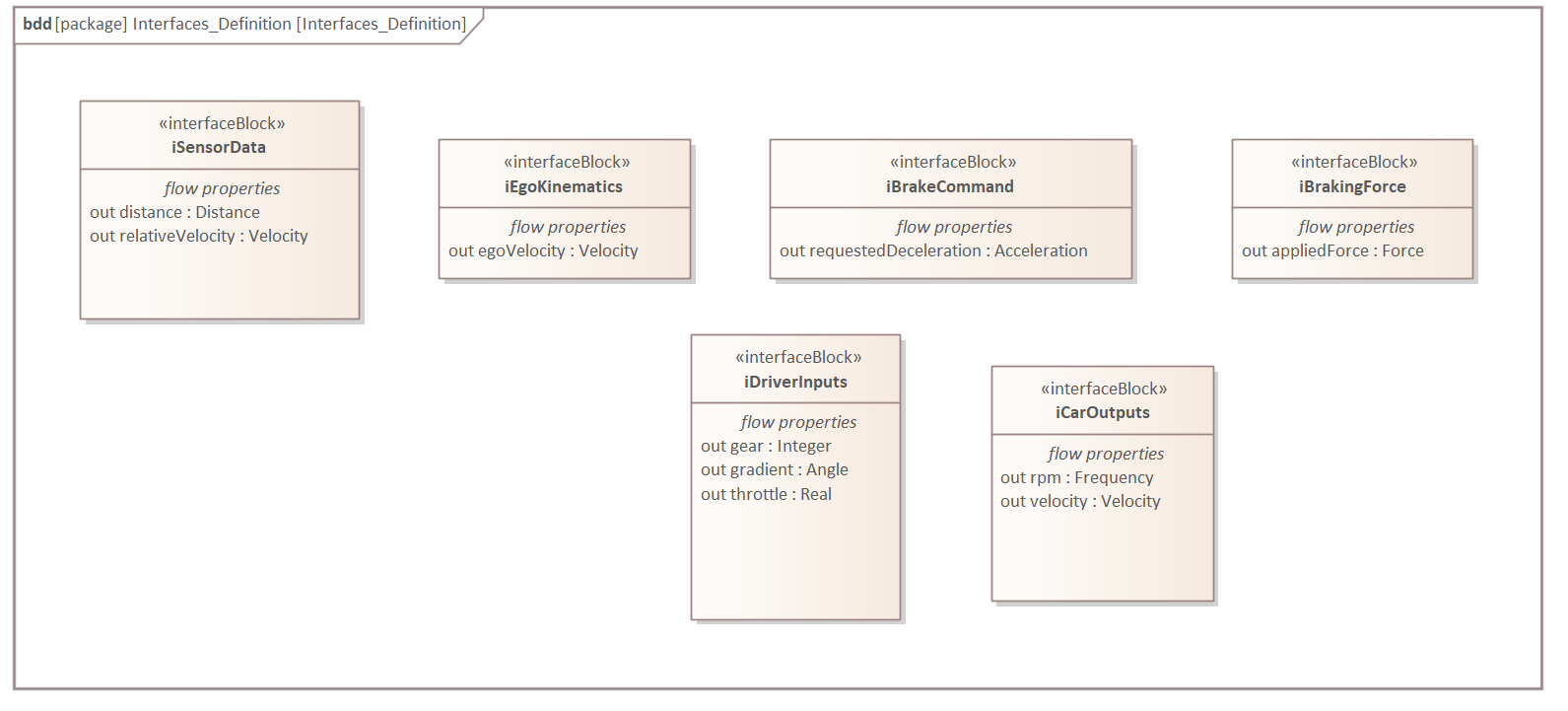
* **核心思想**: Distance这个类型本身就包含了“它的单位是米（m）”这个信息。这使得我们的模型在工程上是严谨和安全的。
* **定义的类型**: 我们定义了Distance (m), Velocity (m/s), Mass (kg), Acceleration (m/s^2)等所有工程计算需要用到的物理量。



#### 2.2 接口契约定义 (Interfaces BDD)

接口块（Interface Block）定义了系统组件之间信息交换的“插座标准”。它规定了通过某个端口可以流动哪些数据。

* **iSensorData**: 定义了传感器向外提供的数据包，包含distance和relativeVelocity。
* **iCarOutputs**: 定义了任何“车辆”类型的模块向外输出的核心状态，包含velocity和rpm。这是一个**可重用**的设计，无论是本车（Ego）还是前车（Lead）都可以使用这个接口。
* **iDriverInputs**: 定义了从驾驶员或仿真环境输入到车辆的控制信号，如油门、档位和坡度。
* **iBrakeCommand & iBrakingForce**: 分别定义了从控制器发出的逻辑指令和最终施加到车身的物理制动力。



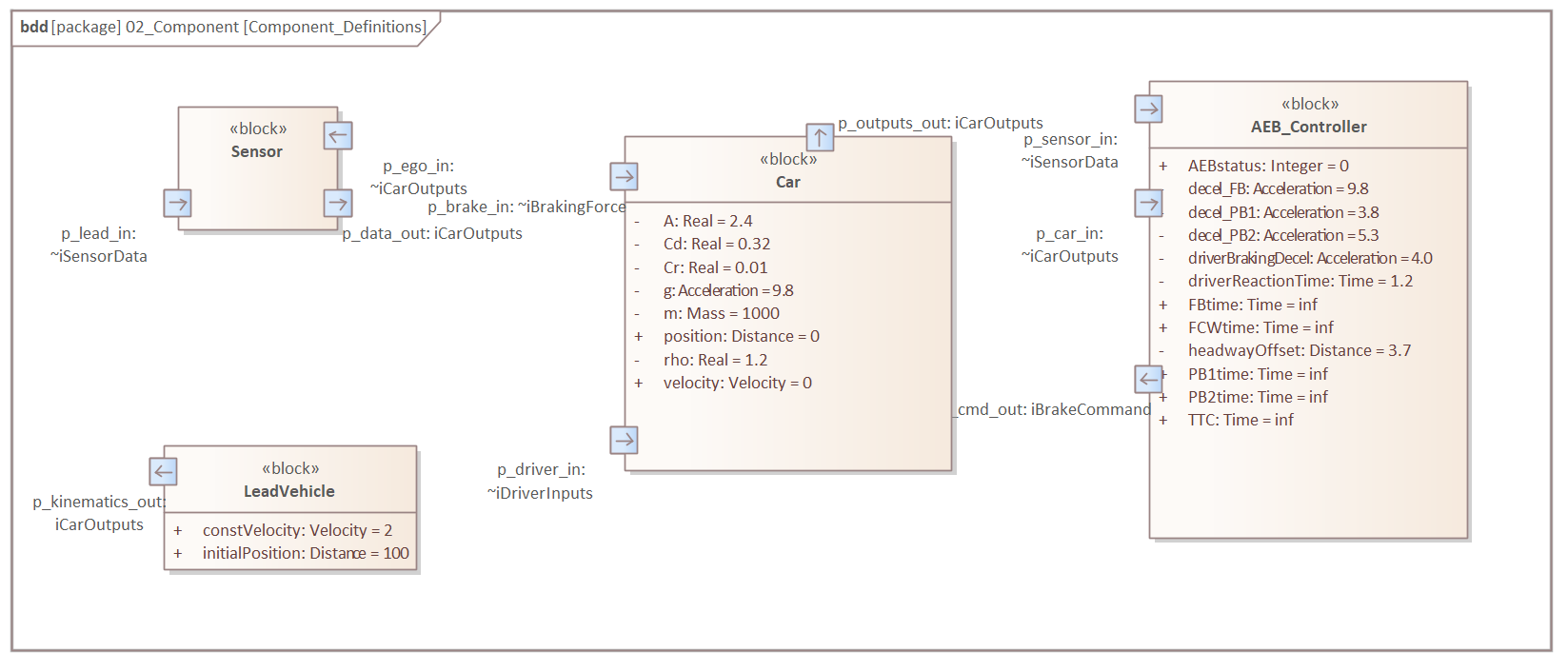
### 第三部分：系统静态架构 (System Architecture)

**目标**: “系统由哪些组件构成？”以及“它们之间是如何连接的？”

#### 3.1 组件蓝图定义 (Component Definitions BDD)

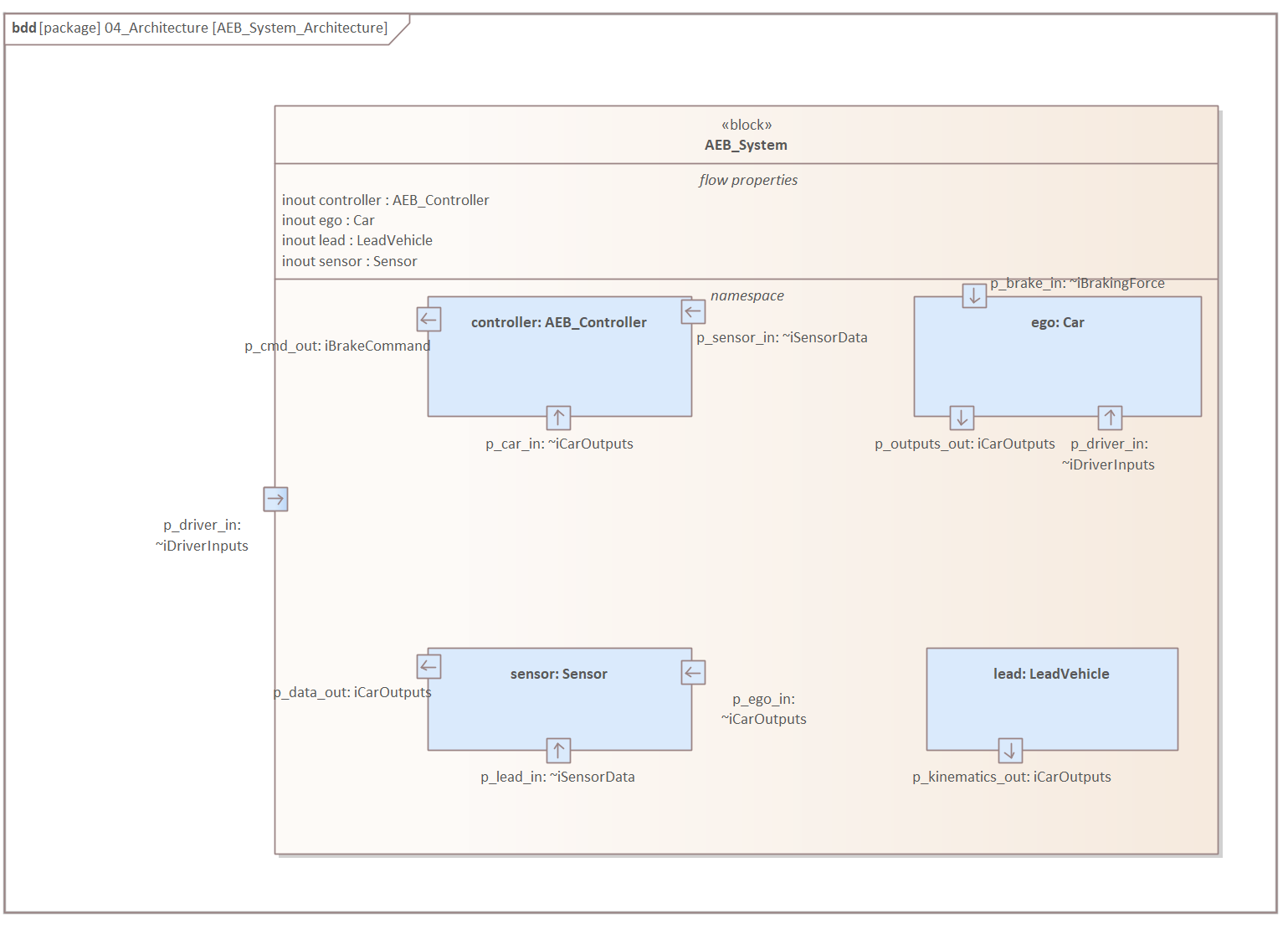
这张图是我们系统的“零件清单”，详细定义了每一个核心功能块的“蓝图”。

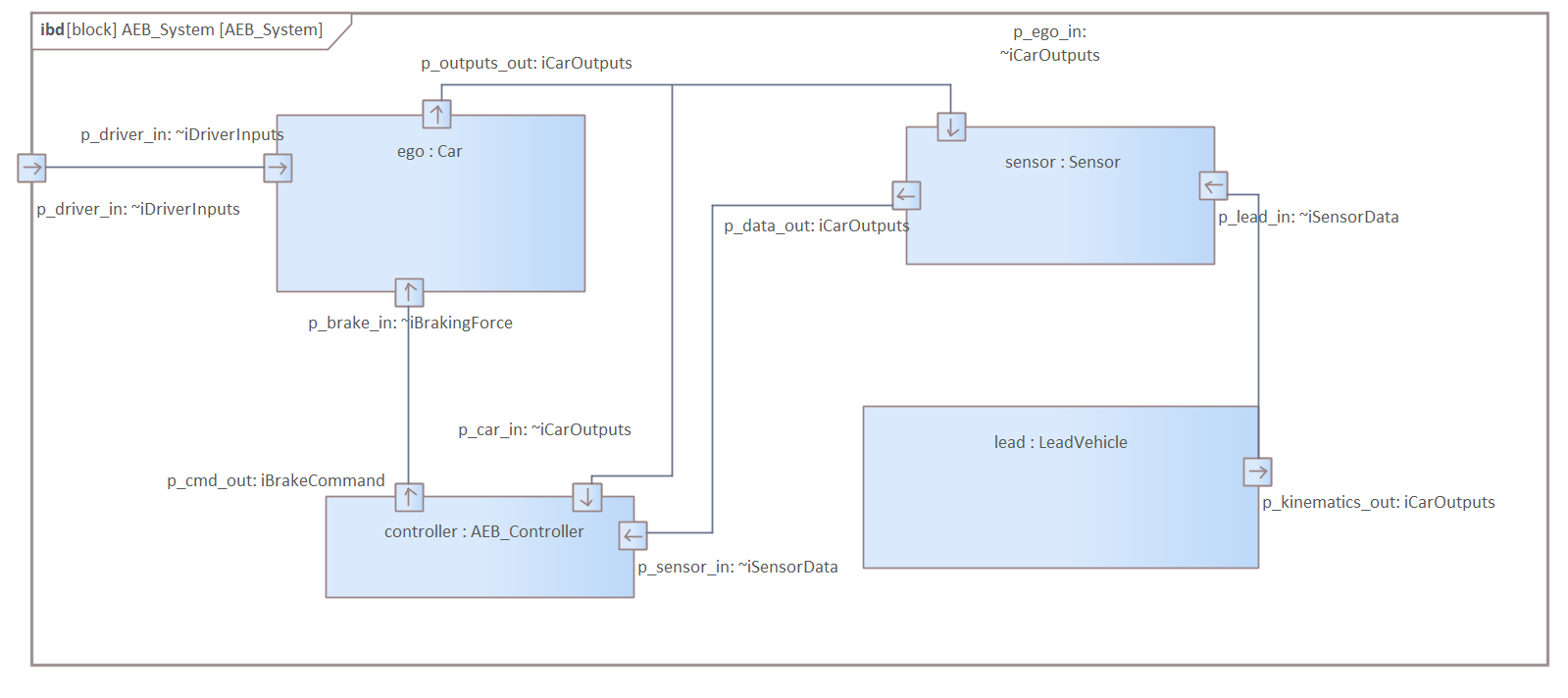
* **Car**: 车辆物理模型。它包含了质量、风阻、滚阻等一系列物理参数（Attributes），并定义了接收驾驶指令和制动力的输入端口，以及输出自身状态的端口。
* **LeadVehicle**: 一个简化的前车仿真模型，它以固定的速度和初始位置行驶。
* **Sensor**: 传感器仿真模型。它接收本车和前车的状态，计算出相对距离和速度。
* **AEB\_Controller**: **系统的“大脑”**。它包含了所有AEB决策所需的阈值参数（如decel\_FB, headwayOffset）和内部状态变量（如TTC）。它的端口定义了它如何从传感器和车辆获取信息，并向外发出制动指令。



#### 3.2 系统组成与连接 (System Architecture BDD/IBD)

这张图展示了我们的最终产品——AEB\_System——是由哪些**部件（Parts）**构成的。

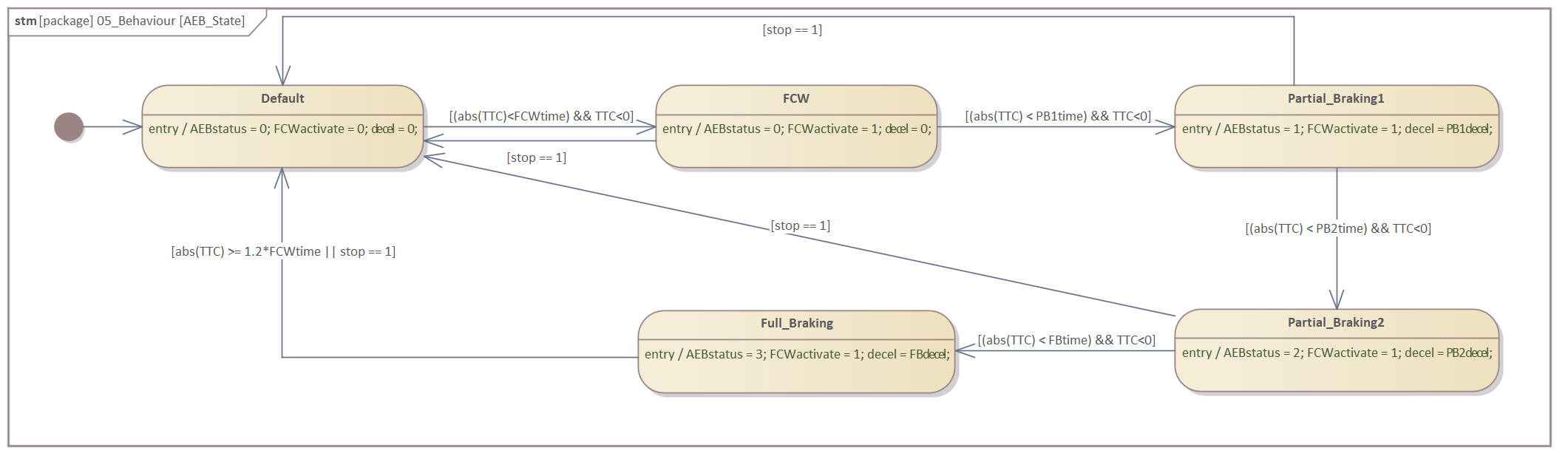
* **核心思想**: 我们将之前定义的块（蓝图）实例化为系统中的具体角色。例如，ego是Car蓝图的一个实例，扮演“本车”的角色。
* **内部连接**: 图中清晰地展示了部件之间的“接线关系”。例如：
  1. sensor从ego和lead获取状态。
  2. sensor处理后的数据被发送到controller。
  3. controller也直接从ego获取速度信息。
  4. controller做出决策后，将制动指令发送给ego。
  5. 外部的driverInputs信号被送入ego。



### 第四部分：系统动态行为 (State Machine Diagram)

**目标**: “系统的大脑（控制器）是如何思考和决策的？”

这是我们整个模型中最核心的图之一，它精确地描述了AEB\_Controller的内部逻辑。



* **状态 (States)**:
  + **Default**: 系统的安全/默认状态，不采取任何动作。
  + **FCW (Forward Collision Warning)**: 风险出现，激活警告但未制动。
  + **Partial\_Braking1 & Partial\_Braking2**: 风险升级，启动两级渐进式制动。
  + **Full\_Braking**: 最高风险等级，执行全力制动。
* **转换 (Transitions) & 守卫 (Guards)**:
  + 连接状态的箭头代表逻辑的流转。
  + 箭头上的[...]条件（守卫）是触发转换的关键。例如，从Default到FCW的转换，只有在TTC小于FCWtime阈值时才会发生。这精确地实现了我们需求REQ-1.2。
* **进入动作 (Entry Actions)**:
  + 每个状态框内的entry: ...定义了当系统进入该状态时必须执行的操作。例如，当进入Partial\_Braking1状态时，系统会立刻将decel（目标减速度）设置为PB1decel的值。

# 二．AEB系统Simulink模型：现状分析与优化策略

### 1. 系统概述 (System Overview)

我们当前的Simulink项目旨在构建一个完整的自动紧急制动（AEB）系统仿真环境。该环境基于两个核心模型：

* **car\_model**: 一个相对成熟和详细的**车辆动力学模型**。它扮演了“被控对象”（Plant）的角色，能够模拟车辆在接收到驾驶员指令（油门、档位）和外部作用力（阻力、制动力）后的物理行为。
* **AEB\_Controller**: 一个集成了AEB控制逻辑的**顶层系统模型**。它尝试将一个AEB控制器与car\_model结合，以实现自动制动的功能。

其核心思想是：

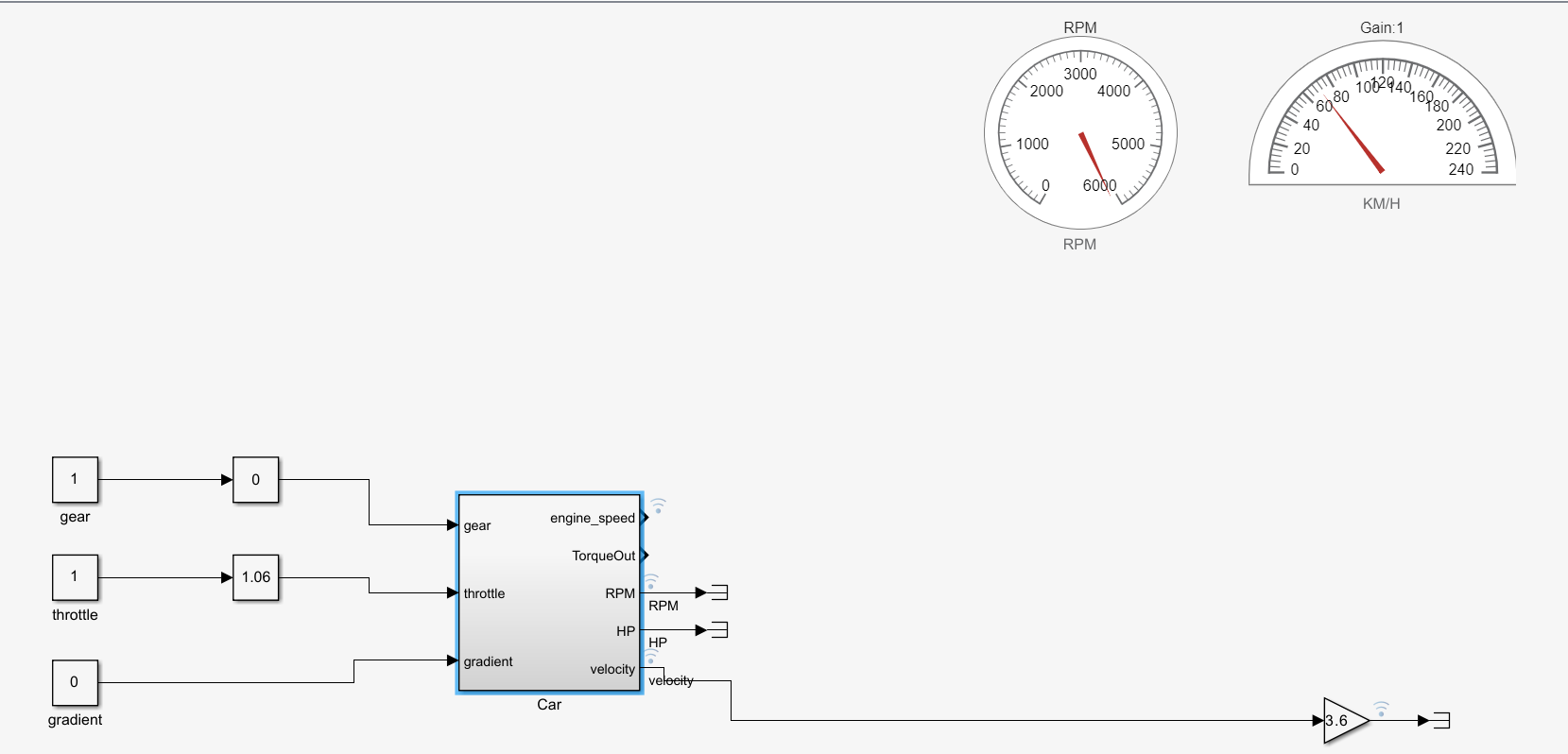
1. 通过一系列Constant模块**模拟**前车状态、本车状态以及AEB的控制参数。
2. 这些数据被送入一个名为 **AEB Controller** 的核心子系统。
3. AEB Controller经过计算，输出制动决策信号（如deceleration）。
4. 这些决策信号（理论上）应作用于 **Car** 子系统，以控制车辆速度。

### 2. 核心组件深度解析 (Component Deep Dive)

#### 2.1 车辆动力学模型 (Car Subsystem)

这是我们系统的物理基础，其内部结构清晰，符合基本的车辆动力学原理。

* **输入**:
  + gear (档位), throttle (油门), gradient (坡度): 模拟驾驶员和环境输入。
  + Braking: 一个关键的输入端口，用于接收外部施加的**制动力(Force)**。
* **输出**: velocity (速度), engine\_speed (引擎转速)等。
* **内部逻辑**:
  + **Tractive Force (牵引力)模块**: 根据油门和档位，通过查表和公式计算引擎产生的驱动力。
  + **Damping Force (阻力)模块**: 计算车辆行驶中受到的各种阻力，如空气阻力、滚动阻力和坡度阻力。
  + 核心运动方程: 模型的核心是一个积分器，它遵循牛顿第二定律：  
    速度 = ∫ ( (牵引力 - 阻力 - 制动力) / 车辆质量 ) dt



Car\_model

图示

AI 生成的内容可能不正确。

Subsytem:Car

图示, 示意图

AI 生成的内容可能不正确。

Subsytem: Tractive Force

图示

AI 生成的内容可能不正确。

Subsystem: Damping Force

#### 2.2 AEB控制器 (AEB Controller Subsystem)

这是系统的控制器，负责所有的决策逻辑。其内部结构主要由三部分组成：

1. **TTCCalculation (碰撞时间计算)模块**:
   * **功能**: 根据输入的距离 (mioDistance)、相对速度 (mioVelocity) 和安全偏移量 (AEB Headway Offset)，计算核心风险指标——**碰撞时间 (TTC)**。
   * **公式**: TTC = (距离 - 安全偏移) / abs(相对速度)
2. **StoppingTimeCalculation (停车时间计算)模块**:
   * **功能**: 计算在不同制动级别下（如FCW预警、一级制动等），车辆从当前速度到停止所需的时间。
   * **公式**: 停车时间 = (本车速度 / 目标减速度) + 时间余量
3. **AEB\_Logic (Stateflow决策逻辑)模块**:
   * **功能**: 这是控制器的心脏。它是一个**状态机**，根据输入的TTC和各种时间阈值，在不同状态间切换，并输出最终的控制指令。
   * **状态**: 包含Default (安全), FCW (预警), Partial\_Braking1 (一级制动), Partial\_Braking2 (二级制动), Full\_Braking (全力制动)五个状态。
   * **决策逻辑**: 状态之间的切换由TTC和预设时间阈值的比较结果严格控制。例如，当abs(TTC)小于FCWtime时，系统从Default切换到FCW状态。
   * **输出**: 在每个状态下，通过entry动作，状态机会设置不同的decel（目标减速度）值和AEBstatus（系统状态码）。

图示, 示意图

AI 生成的内容可能不正确。

Main\_assignment

图示, 示意图

AI 生成的内容可能不正确。

Subsytem: AEB Controller

图示

AI 生成的内容可能不正确。

Subsytem: TTC Calculation

图示, 示意图

AI 生成的内容可能不正确。

Subsytem: StoppingTimeCalculation

图示

AI 生成的内容可能不正确。

Subsytem: AEB\_Logic

### 3. 关键问题诊断 (Critical Issue Analysis)

尽管两个核心组件的设计都相对完善，但在系统集成层面，设计后发现模型存在几个**致命问题**，导致其无法按预期工作。

#### 问题一：开环控制 (Open-Loop Control) - 系统的“断头路”

这是最严重的问题。AEB Controller计算出的deceleration（减速度）输出信号，**并未连接**到Car子系统的Braking输入端口。

* **影响**: 控制器在“自言自语”。无论它做出多么紧急的制动决策，这个决策都无法传递给车辆模型，车辆的速度因此完全不受AEB系统的影响。整个系统是一个**开环系统**，无法形成有效的反馈控制。

#### 问题二：代数环 (Algebraic Loop) - “先有鸡还是先有蛋”的困境

原始报告提到，当作者尝试将deceleration反馈给车辆时，Simulink报错“algebraic loop”。

* **什么是代数环**: 简单来说，就是一个信号的输出值，在**同一仿真时刻**，又依赖于这个信号本身作为输入。A -> B -> C -> A。Simulink无法求解这种瞬时的循环依赖。
* **产生原因**: AEB\_Logic (Stateflow)的输出在同一时刻被用于计算车辆状态，而车辆状态又在同一时刻作为输入影响AEB\_Logic的决策。
* **影响**: 这是导致作者最终放弃连接反馈回路的直接技术原因。

#### 问题三：接口不匹配 (Interface Mismatch) - “牛头不对马嘴”

* AEB Controller输出的是一个**逻辑信号**：deceleration (目标减速度, 单位 m/s²)。
* Car模型需要的是一个**物理输入**：Braking (制动力, 单位 N)。
* 直接将减速度连接到力的输入端口，在物理上是**错误**的，不符合牛顿第二定律 (F=m\*a)。

### 4. 优化策略与后续步骤 (Optimization & Next Steps)

#### 步骤一：实现制动执行器 (Implement Braking Actuator)

**目标**: 解决接口不匹配问题。

1. 我们将创建一个新的子系统，名为 **Braking Actuator**。
2. **输入**: 接收来自AEB\_Controller的requestedDeceleration (m/s²)。
3. **输出**: brakingForce (N)。
4. **核心逻辑**: brakingForce = vehicleMass \* requestedDeceleration。这里，vehicleMass是车辆模型的质量参数。
5. 将这个brakingForce输出连接到Car模型的Braking输入端口。

#### 步骤二：打破代数环 (Break the Algebraic Loop)

**目标**: 解决仿真报错问题，实现闭环。

1. 最直接有效的方法是在反馈路径中插入一个 **Unit Delay** 或 **Memory** 模块。
2. **放置位置**: 建议将延迟模块放置在反馈给AEB\_Controller的egoVelocity信号路径上。
3. **合理解释**: 这在物理上是完全合理的。任何传感器、控制器和执行器都有延迟。增加一个采样周期的延迟，意味着控制器是基于**上一个时刻**的车辆状态来计算**当前时刻**的指令，这完美地模拟了数字控制系统的真实工作方式，并从根本上打破了代数环。

图示

AI 生成的内容可能不正确。

function floorVelocity = fcn(velocity)

if velocity<0

floorVelocity = 0;

else

floorVelocity = velocity;

end

#### 步骤三：构建闭环系统与测试场景

**目标**: 整合所有修改，并进行测试。

1. **连接闭环**: 将AEB\_Controller的输出通过我们新建的Braking Actuator连接到Car的制动输入。将Car的速度输出通过Unit Delay模块反馈给AEB\_Controller。

图示, 示意图

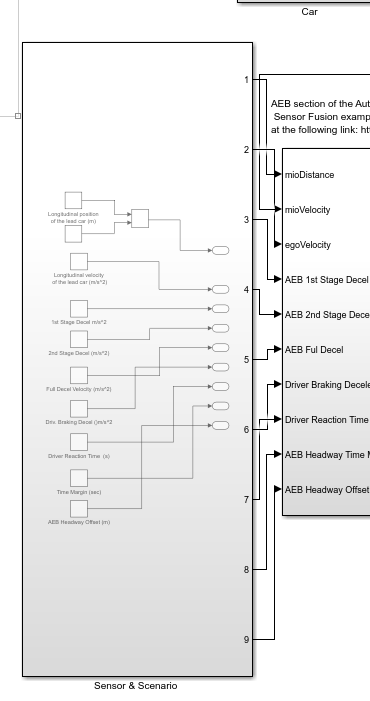
AI 生成的内容可能不正确。

形状, 箭头

AI 生成的内容可能不正确。

Subsystem:Braking Actuator

1. **建立清晰的传感器模型**: 整合所有Constant模块，创建一个专门的Sensor & Scenario子系统，负责提供本车和前车的初始状态，并计算相对距离和速度。



图示, 示意图

AI 生成的内容可能不正确。

Subsytem:Sensor&Scenario

1. **定义测试场景**: 设立一个明确的基准测试工况。例如：
   * **本车 (Ego)**: 初始速度 25 m/s (90 km/h)。
   * **前车 (Lead)**: 初始速度 10 m/s (36 km/h)，位于本车前方 100 米处。
   * **仿真时间**: 15 秒。
2. **观察指标**: 运行仿真，重点观察velocity, TTC, AEBstatus和deceleration等信号的波形，以验证系统行为是否符合预期。

# 三．AEB控制器核心逻辑的形式化验证报告 (Z3)

目标：为AEB这个安全关键系统，提供最高等级的质量保证。

### 1. 我们要验证什么？—— 关键安全属性

对于一个AEB系统，最可怕的两种失效模式是：

1. **该刹车时不刹车 (False Negative)**
2. **不该刹车时乱刹车 (False Positive)**

后者，即“幽灵刹车”，会极大地影响驾乘体验和用户信任。因此，我们选择验证以下这个至关重要的**安全属性 (Safety Property)**：

**“系统绝不应该在情况安全时，却错误地执行了全力制动。”**

这个属性保证了系统的行为是可预测且合理的，避免了最危险的误触发场景。

### 2. 验证策略：用Z3寻找“反例”

我们的核心策略是**通过矛盾来证明 (Proof by Contradiction)**。我们不直接让Z3去证明我们的属性是“对的”，而是反过来，要求Z3去努力寻找一个能推翻我们属性的**反例 (Counterexample)**。

我们向Z3提出的问题是：

**“请你（Z3）寻找一组输入（TTC, FBtime等），使得‘系统执行了全力制动’和‘当时情况其实是安全的’这两个条件能够同时成立。”**

这个问题被我们编码在Python脚本中。

### 3. Python代码解析：将系统逻辑翻译为数学约束

我们的veri.py脚本精确地执行了上述策略，它主要包含以下几个步骤：

#### 3.1 定义变量

我们将AEB控制器的所有相关输入、输出和参数，都定义为Z3可以理解的数学变量（在这里是Real实数）。

TTC      = Real('TTC')# 碰撞时间

FCWtime  = Real('FCWtime')  # 预警时间阈值

FBtime   = Real('FBtime')   # 全力制动时间阈值

decel    = Real('decel')    # 最终输出的减速度

FBdecel  = Real('FBdecel')  # 全力制动的目标减速度值  
文本

AI 生成的内容可能不正确。

#### 3.2 建模控制器逻辑

我们将SysML状态机图中的决策逻辑，使用Z3的If(condition, then\_value, else\_value)函数，翻译成了一个精确的数学表达式。这就像一个嵌套的决策树，完美复现了Stateflow的行为。

# 定义进入全力制动区的条件  
is\_full\_braking\_zone = And(Abs(TTC) < FBtime, TTC < 0)  
# ... (其他区域的定义) ...  
# 核心决策逻辑：如果满足全力制动条件，则decel等于FBdecel；否则，进入下一层判断...  
controller\_logic = If(is\_full\_braking\_zone, decel == FBdecel,  
 # ... (其他制动级别的逻辑) ...  
图片包含 文本

AI 生成的内容可能不正确。

#### 3.3 定义“反例”

这是最关键的一步，我们用Z3的语言来描述我们想要寻找的那个“坏”情况。

**“系统执行了全力制动”**:  
full\_brakes\_applied = (decel == FBdecel)

**“当时情况其实是安全的”:**  
这在逻辑上等同于“不满足进入全力制动状态的条件”。我们使用Not()来实现。  
situation\_is\_safe = Not(And(Abs(TTC) < FBtime, TTC < 0))

**最终要检查的“坏”情况**:  
property\_to\_check = And(full\_brakes\_applied, situation\_is\_safe)  
文本

AI 生成的内容可能不正确。

#### 3.4 请求Z3求解

最后，我们将所有这些规则（控制器逻辑、参数值、以及我们想找的“坏”情况）全部交给Z3求解器，然后命令它开始计算。

solver.add(controller\_logic)

solver.add(param\_constraints)

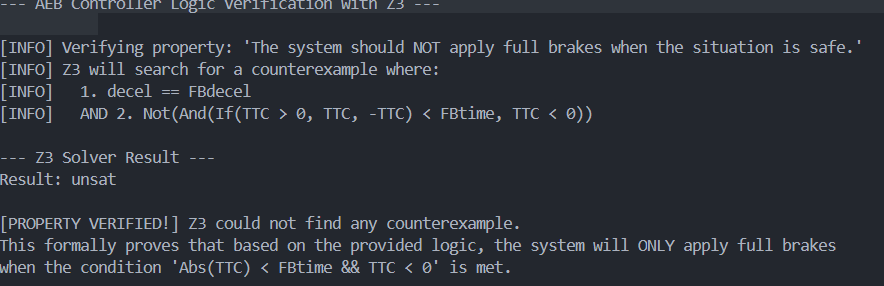
solver.add(property\_to\_check)

result = solver.check()

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

### 4.结果解读：



--- Z3 Solver Result ---  
Result: unsat

**unsat是Unsatisfiable。这个结果的含义是：**

**Z3在穷尽了所有可能的输入组合后，得出结论：我们要求它寻找的那个“坏”情况（即property\_to\_check）在逻辑上是永远无法成立的。描述“全力制动”和“情况安全”的这两个条件，存在根本性的数学矛盾，它们永远不可能同时为真。**

换句话说，Z3没能找到任何一个反例。由于Z3的搜索是完备的，这就从数学上**证明**了我们的安全属性是成立的。