# **AEB（自动紧急制动）系统SysML模型说明文档**

### **第一部分：系统需求分析 (Requirement Diagram)**

**目标**: 回答“我们要做一个什么样的系统？”

需求是所有设计的起点。我们使用需求图来捕获和组织AEB系统的所有功能性及非功能性要求，并构建了一个清晰的层次结构。

#### **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。

#### **1.3 优化建议**

* **量化性能需求**: 当前的需求主要描述了“做什么”，我们可以增加更具体的性能需求，例如“REQ-1.3.3.1: 在相对速度低于50km/h时，完全制动应能避免碰撞”，这将为后续的测试和验证提供明确的通过/失败标准。
* **增加可追溯性**: 在后续工作中，我们会将这些需求与其他模型元素（如AEB\_Controller块、状态机中的状态）建立<<trace>>关系，确保设计的每一个部分都是为了满足某个特定需求而存在的。

### **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。

#### **2.3 优化建议**

* 在Interfaces\_Definition图中，iEgoKinematics可以被iCarOutputs完全替代，以提高模型的可重用性和一致性。建议将所有依赖iEgoKinematics的端口类型都更新为iCarOutputs。

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

**目标**: 回答“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。
* **澄清**: 这张图在EA中虽然是BDD，但因为它展示了内部连接，所以它扮演了\*\*内部块图（Internal Block Diagram - IBD）\*\*的角色。这是一种常见的实践，清晰地描绘了系统的内部静态结构和数据流路径。

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

**目标**: 回答“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。

#### **4.1 优化建议**

* **增加退出动作 (Exit Actions)**: 我们可以为某些状态增加exit:动作。例如，在离开FCW状态时，可以确保警告灯被关闭。
* **考虑并发状态**: 对于更复杂的系统，可以考虑使用并发区域（Concurrent Regions）来同时处理不同的逻辑，例如，一个区域处理制动逻辑，另一个区域处理与HMI（人机界面）的交互。

### **第五部分：总结与后续步骤**

通过以上SysML模型，我们成功地：

1. **明确了系统目标** (需求图)。
2. **统一了工程语言** (数据类型与接口图)。
3. **设计了系统静态结构** (块定义图与组成图)。
4. **精确描述了核心决策逻辑** (状态机图)。

**后续步骤建议**:

1. **模型验证与仿真**: 以此SysML模型为蓝图，在MATLAB/Simulink中构建或完善仿真模型。确保Simulink模型（特别是Stateflow）的行为与我们的状态机图完全一致。
2. **形式化验证**: 将状态机模型导出或转换为UPPAAL等模型检查工具的格式，以形式化地证明系统的关键安全属性（例如，“系统永远不会在TTC小于危险阈值时还处于Default状态”）。
3. **完善追溯性**: 建立从需求到架构再到行为的完整<<trace>>链接，确保设计的完整性，并为未来的变更影响分析提供支持。