An introduction to WESS

Dr. Yonglin Lei
National University of Defense Technology, Changsha, China
2015.6

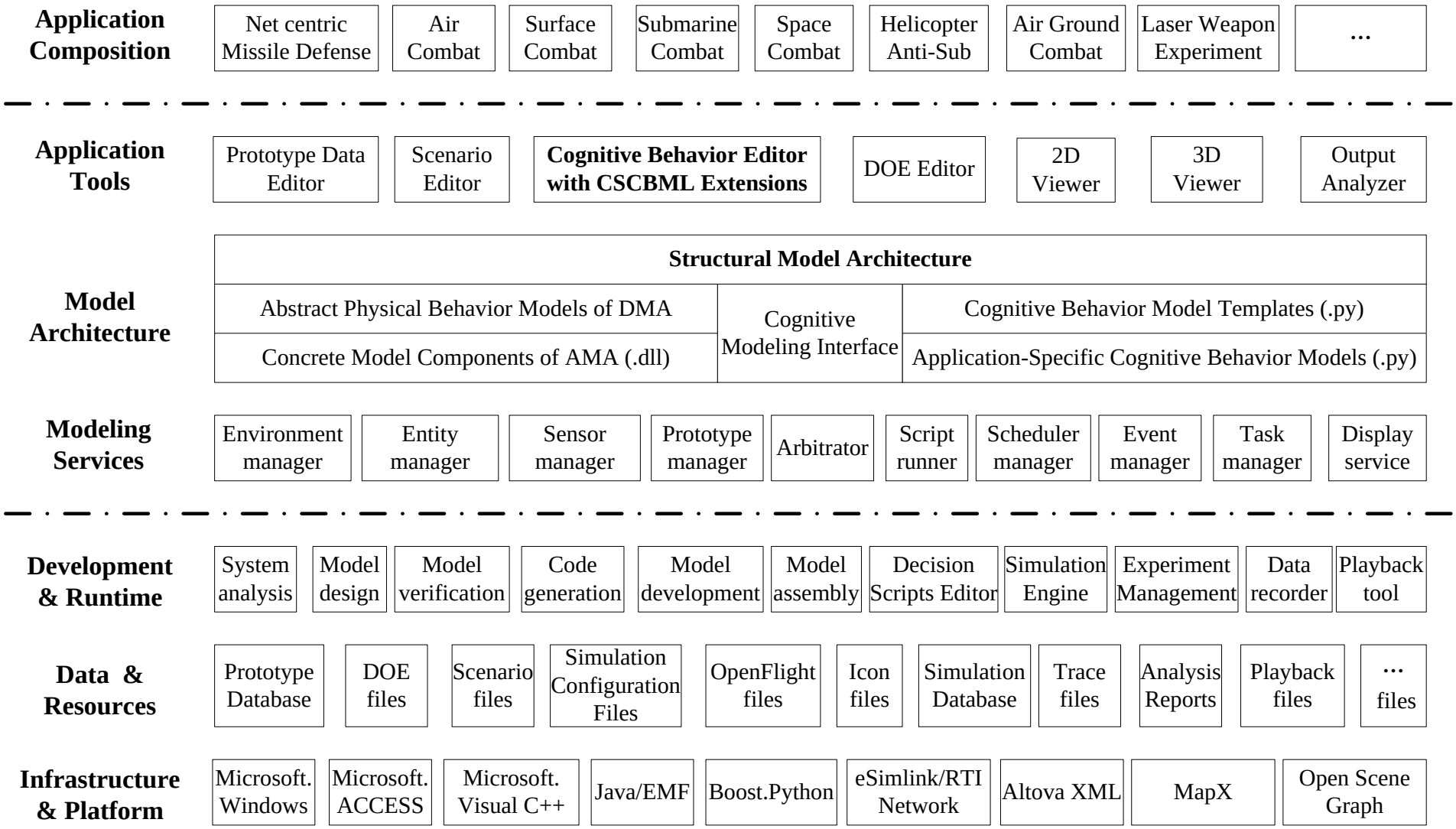
Outline

- Positioning WESS
- Software architecture
- Application flowcharts
- Main components

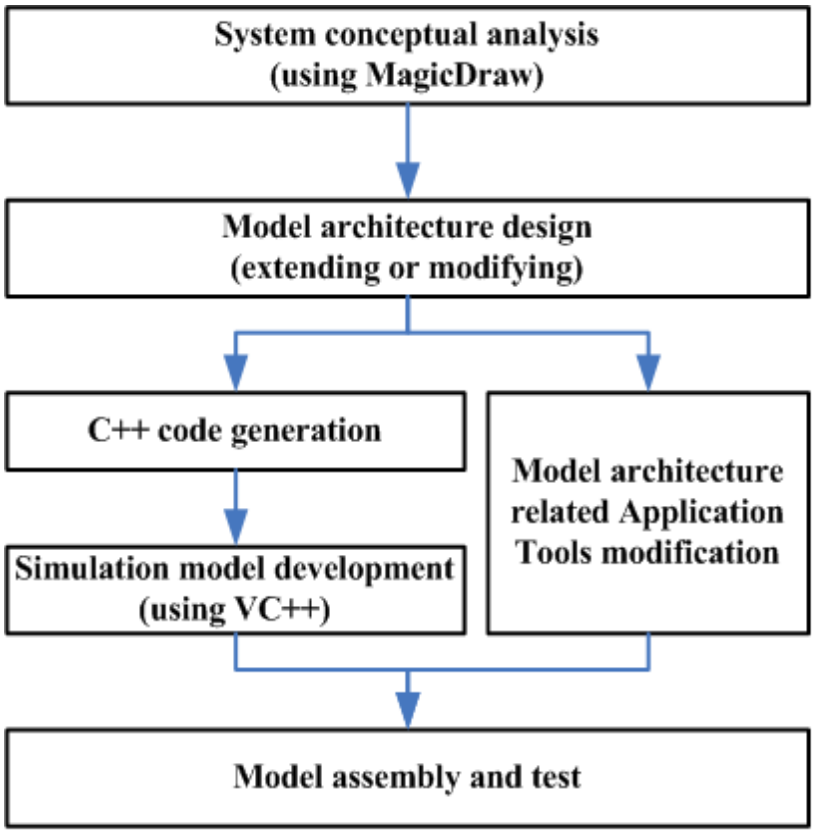
Positioning WESS

- WESS(Weapon Effectiveness Simulation System) is a generic and extensible combat system effectiveness simulation system.
- It is an engagement-level simulation system applied to acquisition and overall design of complex combat systems.
- It provides a platform-centric model architecture with net-centric modeling support based on SMP and other MDE technologies.

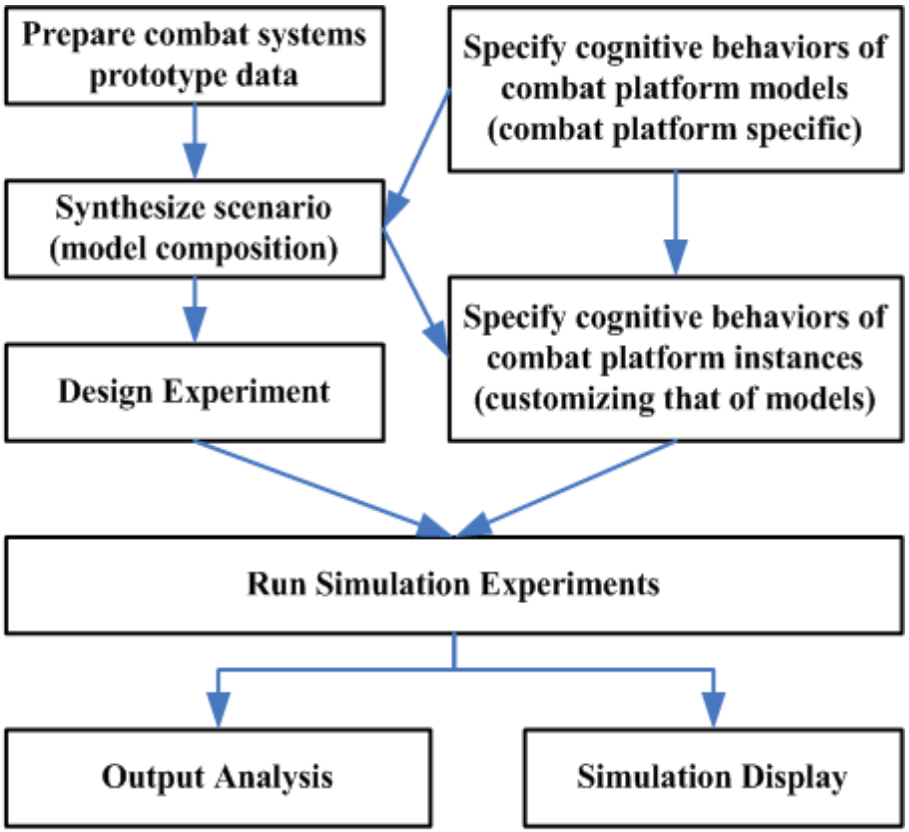
WESS software architecture



WESS application flowcharts



Workflow of Model Architecture Maintaining



Workflow of Composition and Application

Main components

- 1. Composable simulation development toolset
- 2. Simulation application and analysis toolset
- 3. Generic model architecture
- 4. Model component library
- 5. A set of cognitive decision model scripts

1. Composable simulation development toolset

- Functions
  - Based on Simulation Modeling Platform(SMP)
  - Modeling the model architecture of WESS and its components
  - Enable incorporation, extension, and evolvement of WESS simulation models.
  - Provide capabilities for simulation model design, verification, assembly, and code generation.

模型开发集成环境

文件 编辑 模型设计 帮助

CESS.catalogue

- file:/C:/WESS/CESS.catalogue
  - Document Root
    - <catalogue> Catalogue CESS
      - Namespace BaseModel
      - Namespace CommonDataType
      - Namespace CounterMeasure
      - Namespace Group
      - Namespace Gun
      - Namespace Interaction
      - Namespace Platform
        - Interface IAirbase
        - Model tmPlatform
          - Container CounterMeasureList
          - Container GunList
          - Container SensorList
          - Container WeaponList
          - Event Sink DamageResultSink
          - Event Sink LockedByRadarSink
          - Event Sink ReleaseByRadarSink
          - Event Source CloseJammerSource
          - Event Source FireBallisticGunSour
          - Event Source FireLaserGunSource
          - Event Source LaunchSBMSource
          - Event Source OpenJammerSource
          - Event Source SensorOffSource
          - Event Source SensorOnSource
          - Field CruiseSpeed
          - Field DHeight
          - Field DLength
          - Field DWidth
          - Field MaxSpeed
          - Field MissionName
          - Field rBallisticCountFired
          - Field rCMLaunchCount
          - Field rDamageLevel
          - Field rFirstCMLaunchTime
          - Field rFirstWeaponLaunchTime
          - Field rFoundTargetDistance
          - Field rFoundTargetTime
          - Field rHitTargetCount
          - Field rHitWeaponCount

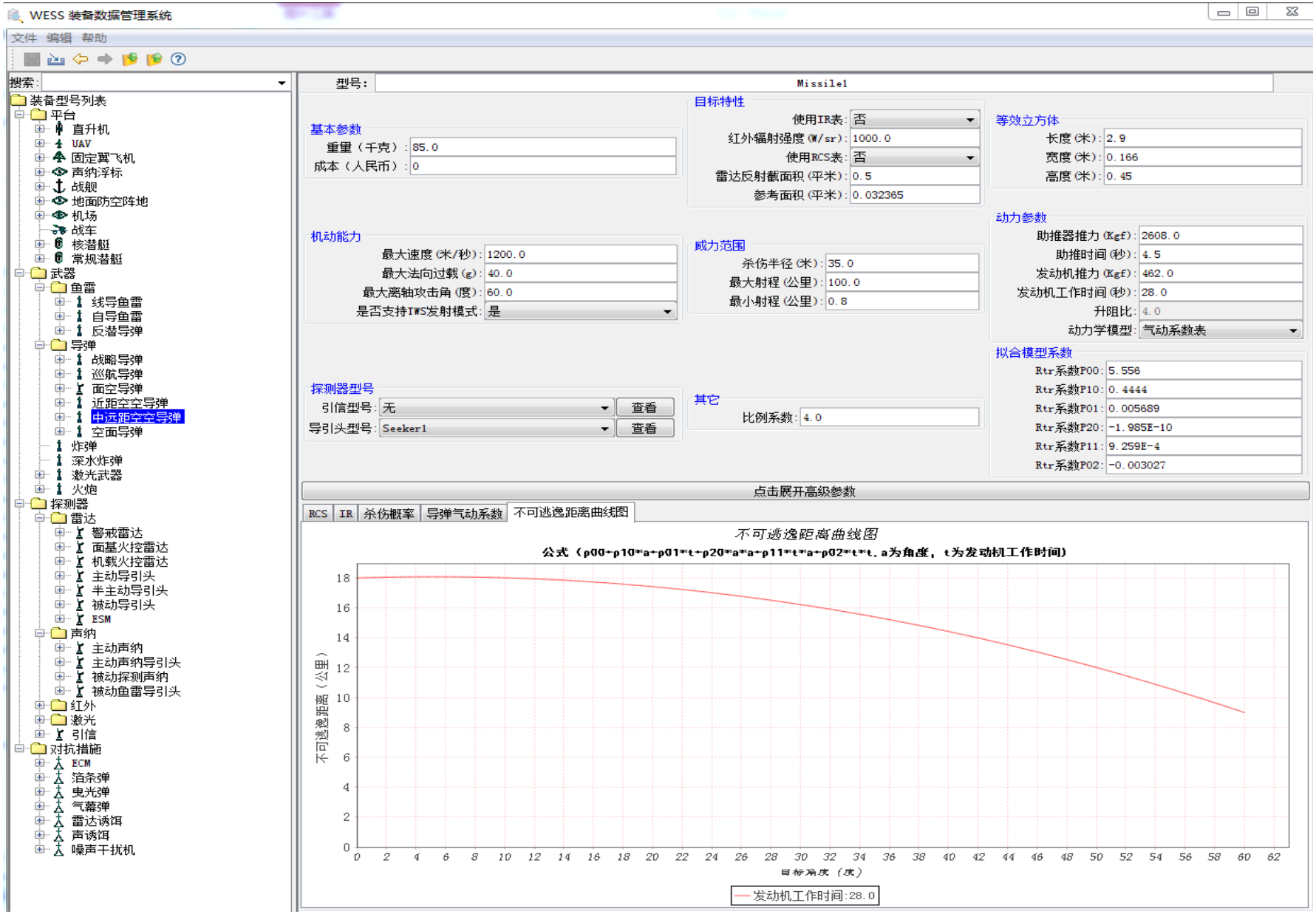
Pr... Pl... Pl... Pl... Pl... Pl... \t... \t...

```
#include "BaseModel/IWeaponLockSubscriber.h"
#include "Sensor/tmSensor.h"
#include "Weapon/tmWeapon.h"
#include "CounterMeasure/tmCounterMeasure.h"
#include "Gun/tmGun.h"
#include "Sensor/tmSensorForward.h"
#include "Interaction/DataType/DataTypeNamespace.h"
#include "Smp/SimpleTypes.h"
#include "CommonDataType/CommonDataTypeNamespace.h"

namespace Platform
{

	class tmPlatform:
		virtual public ::Smp::IDynamicInvocation,
		virtual public ::Smp::Mdk::Management::ManagedModel,
		virtual public ::Smp::Mdk::Composite,
		virtual public ::Smp::Mdk::Management::EventProvider,
		virtual public ::Smp::Mdk::Management::EventConsumer,
		virtual public ::BaseModel::tmSimEntity,
		virtual public ::BaseModel::IDecisionable,
		virtual public ::Group::IGroupNode,
		virtual public ::BaseModel::IWeaponLockSubscriber
	{

		// ------------------------- Constructors/Destructor ---------------------

		public:
			tmPlatform(void);
			tmPlatform( Smp::String8 name, Smp::String8 description, Smp::ICompos
			virtual ~tmPlatform(void);


		// ---------------------------------- Fields ----------------------------

	private:
			// 巡航速度(m/s)
		::Smp::Float32 CruiseSpeed;
	public:
		::Smp::Float32 GetCruiseSpeed() const {return CruiseSpeed;}
		void SetCruiseSpeed(const ::Smp::Float32& newValue) {CruiseSpeed = newValue

	private:
			// 最大航速(m/s)
		::Smp::Float32 MaxSpeed;
	public:
		::Smp::Float32 GetMaxSpeed() const {return MaxSpeed;}
		void SetMaxSpeed(const ::Smp::Float32& newValue) {MaxSpeed = newValue;}

	private:
			// 使命名字(从想定中解析。动态创建的子平台的同父平台的。)
		::CommonDataType::NameString MissionName;
	public:
		::CommonDataType::NameString GetMissionName() const {return MissionName;}
		void SetMissionName(const ::CommonDataType::NameString& newValue) {Mission

	private:
			// 是否完成任务
		::Smp::Bool rMissionSuccess;
```

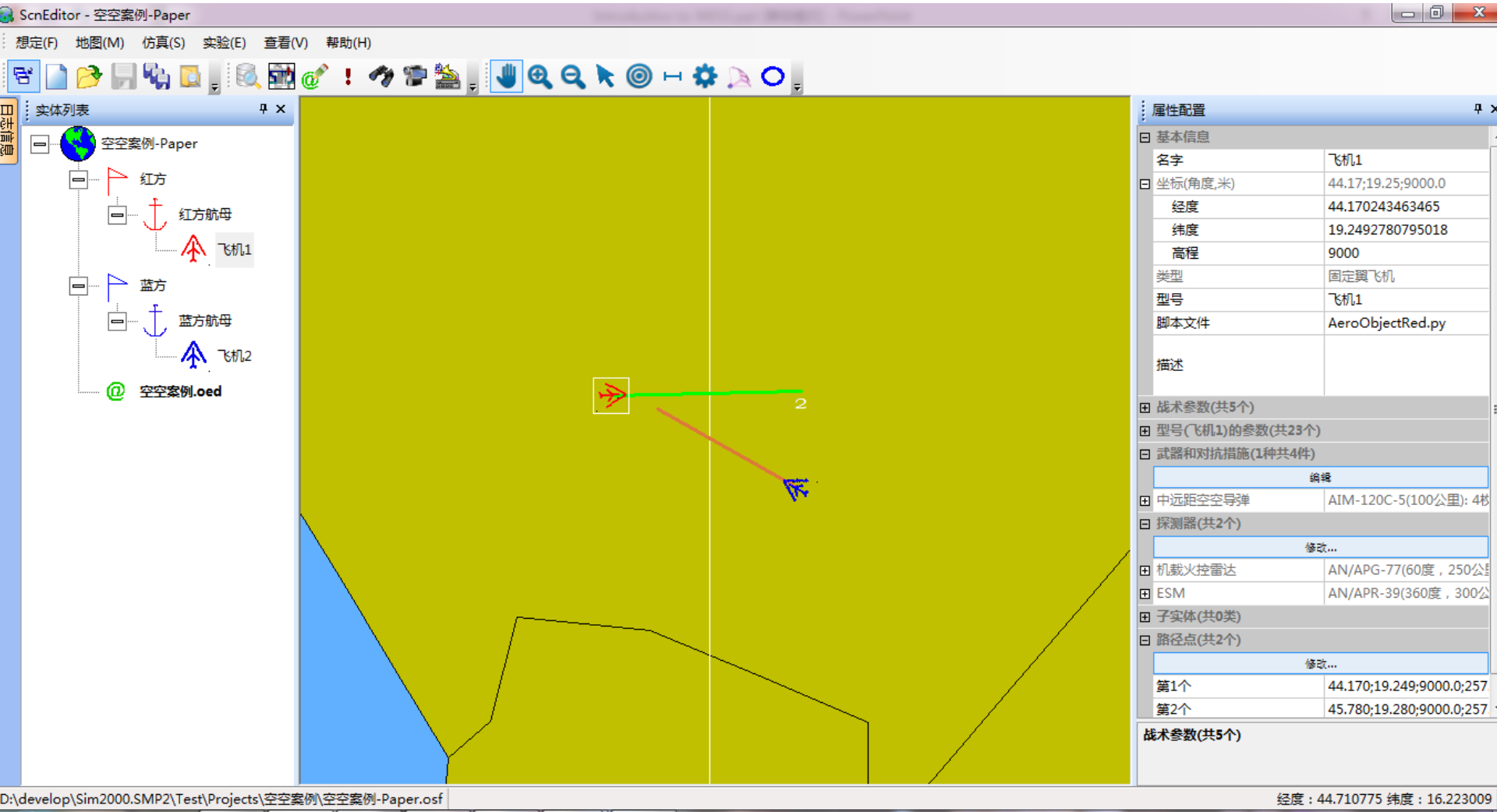
2. Simulation application and analysis toolset

- Component tools
  - DataManager: manage combat system parameters
  - ScnEditor: scenario editor
  - ScriptEditor: cognitive behavior model editor
  - DoeEditor: simulation experimental design
  - Simulator: batch runs with Monte-Caro sampling
  - SimDisplay: simulation display in 2D and 3D
  - SimLogger: log and playback simulation data
  - OutputAnalyzer: script-based output analyzer

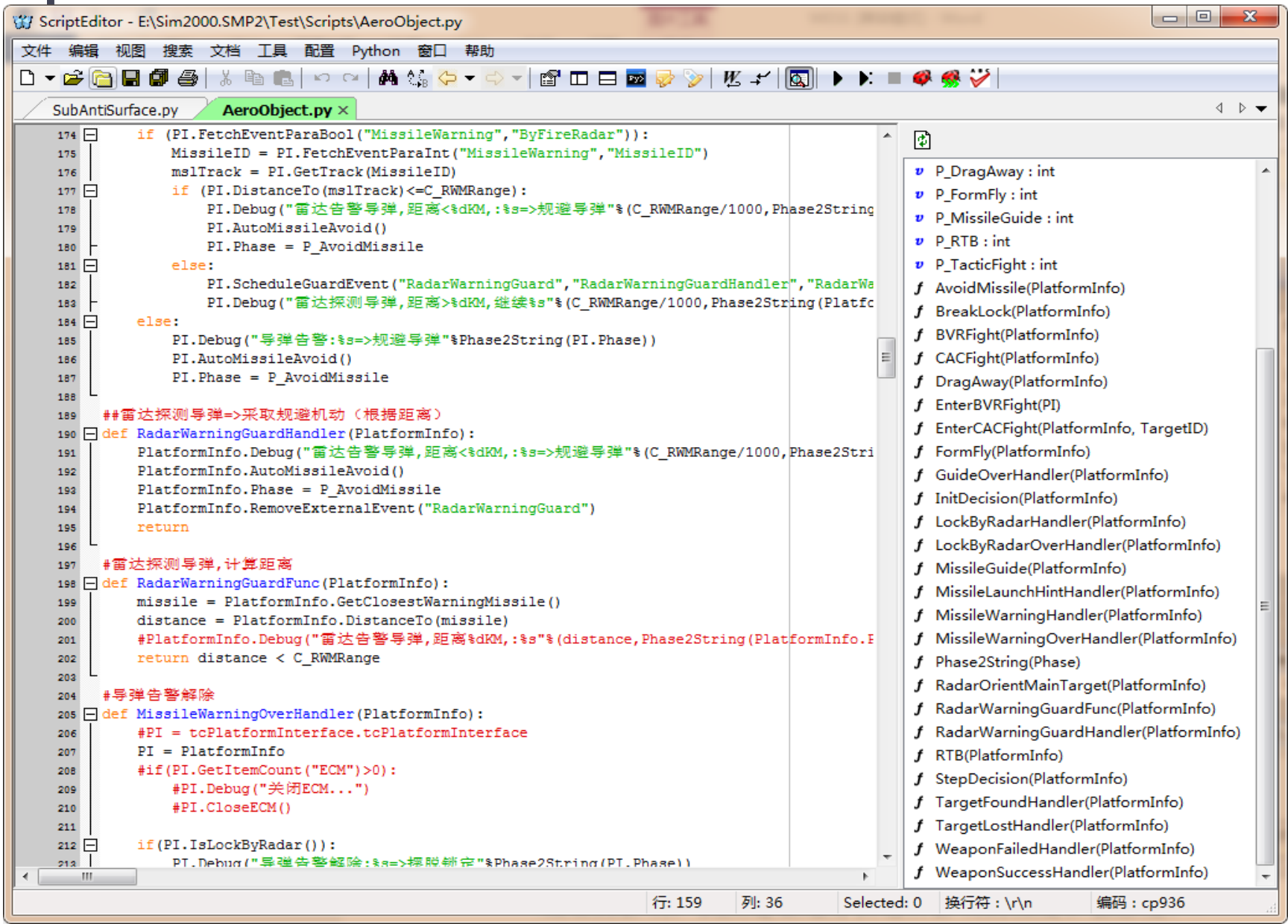
DataManager



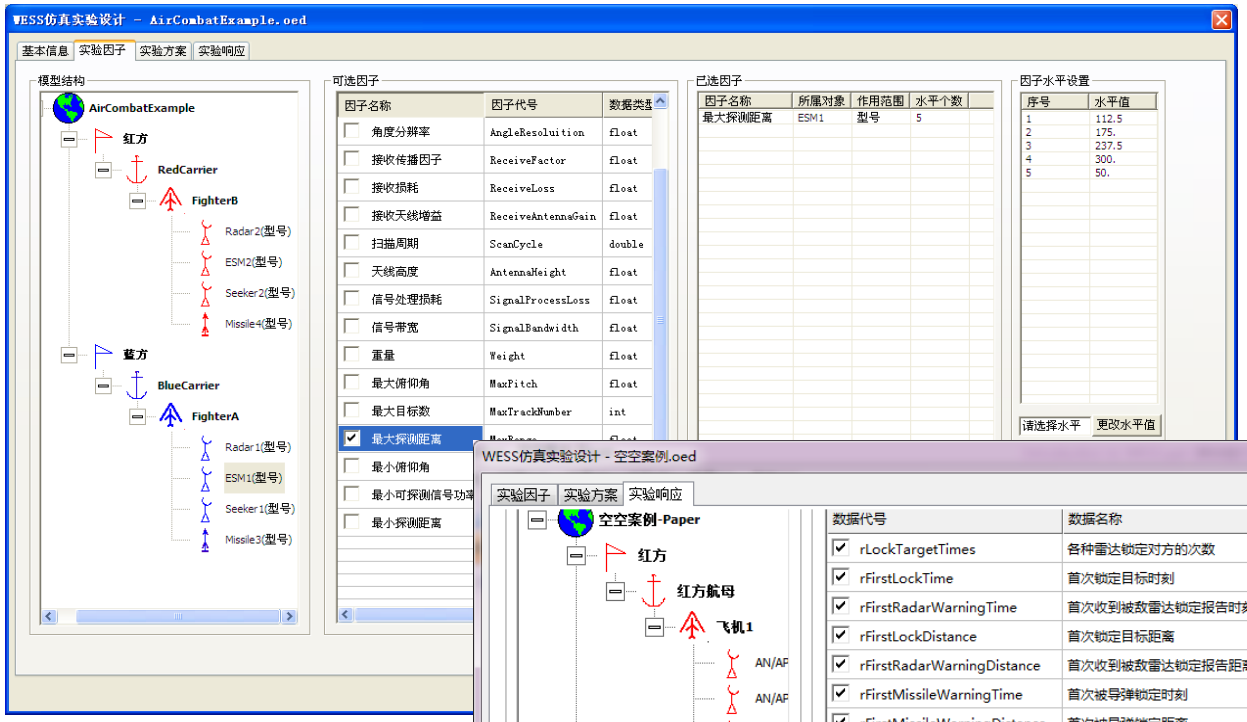
ScnEditor


ScnEditor - 空空案例-Paper
想定(F) 地图(M) 仿真(S) 实验(E) 查看(V) 帮助(H)
实体列表
空空案例-Paper
红方
红方航母
飞机1
蓝方
蓝方航母
飞机2
空空案例.oed
属性配置
基本信息
名字 飞机1
坐标(角度,米) 44.17;19.25;9000.0
经度 44.170243463465
纬度 19.2492780795018
高程 9000
类型 固定翼飞机
型号 飞机1
脚本文件 AeroObjectRed.py
描述
战术参数(共5个)
型号(飞机1)的参数(共23个)
武器和对抗措施(1种共4件)
编辑
中远距空空导弹 AIM-120C-5(100公里): 4枚
探测器(共2个)
修改...
机载火控雷达 AN/APG-77(60度，250公里
ESM AN/APR-39(360度，300公
子实体(共0类)
路径点(共2个)
修改...
第1个 44.170;19.249;9000.0;257
第2个 45.780;19.280;9000.0;257
战术参数(共5个)
D:\develop\Sim2000.SMP2\Test\Projects\空空案例\空空案例-Paper.osf
经度：44.710775 纬度：16.223009

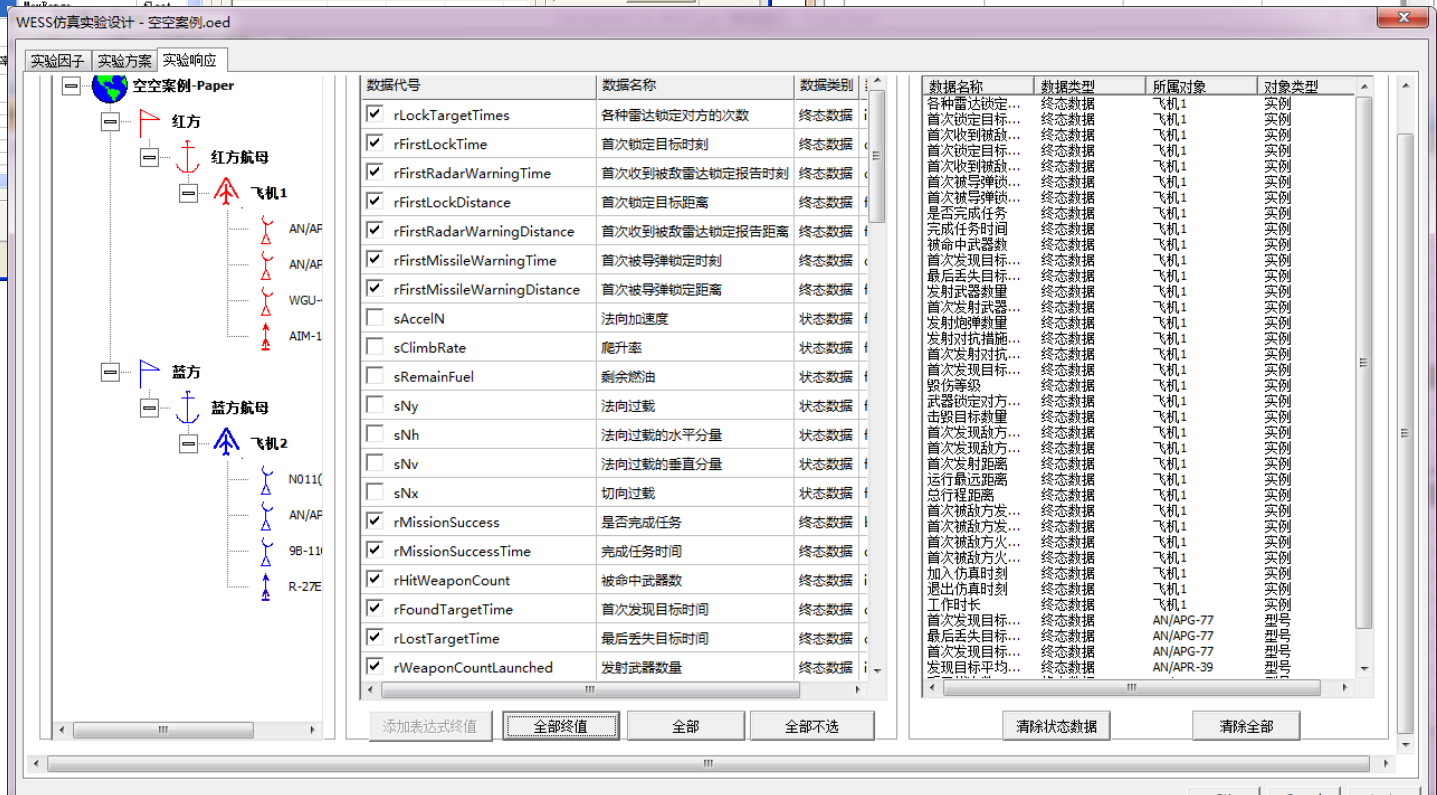
ScriptEditor



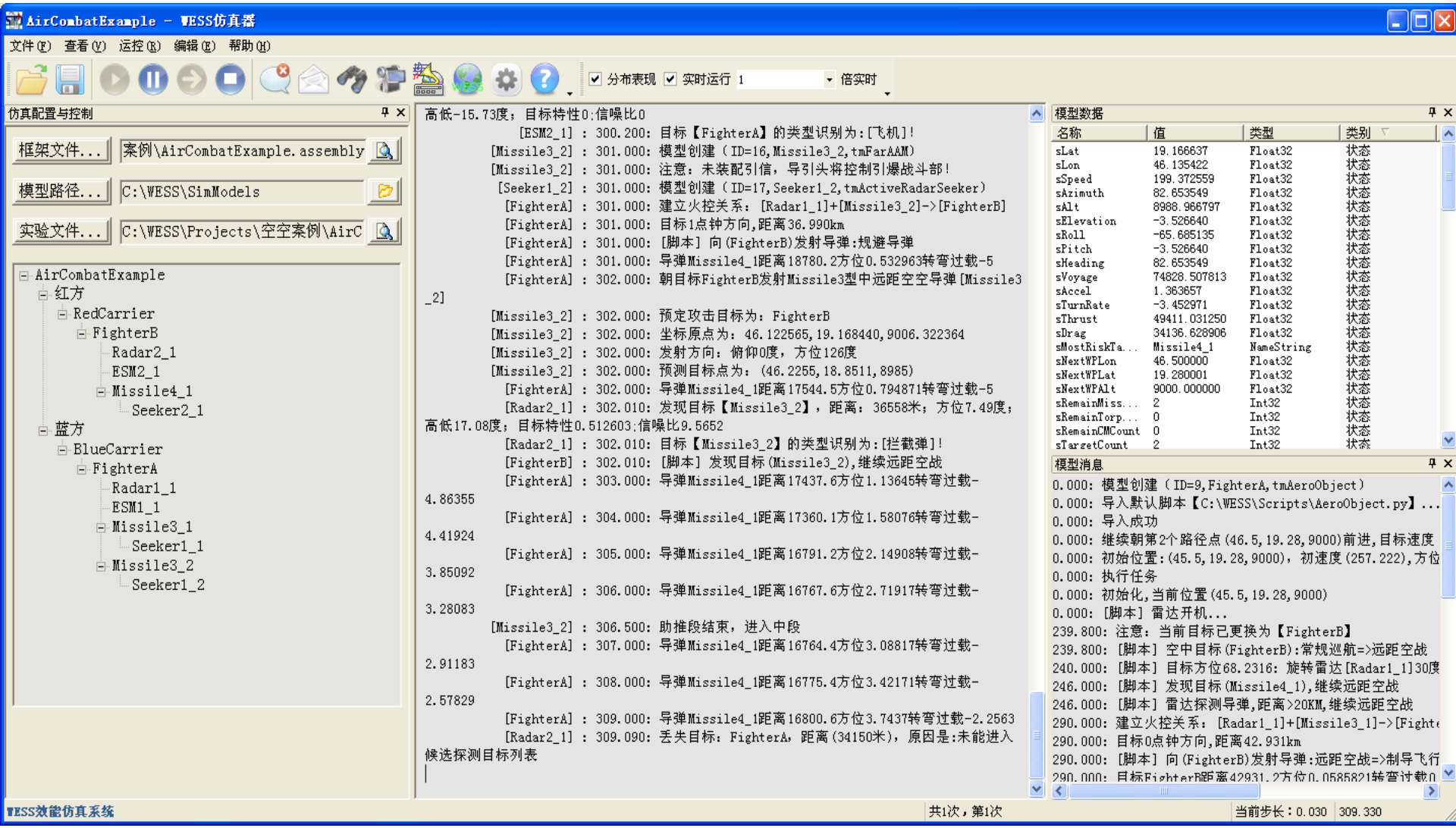
DoeEditor


WESS仿真实验设计 - AirCombatExample.oed
基本信息
实验因子
实验方案
实验响应
模型结构
AirCombatExample
红方
RedCarrier
FighterB
Radar2(型号)
ESM2(型号)
Seeker2(型号)
Missile4(型号)
蓝方
BlueCarrier
FighterA
Radar1(型号)
ESM1(型号)
Seeker1(型号)
Missile3(型号)
可选因子
因子名称
因子代号
数据类型
角度分辨率
AngleResolution
float
接收传播因子
ReceiveFactor
float
接收损耗
ReceiveLoss
float
接收天线增益
ReceiveAntennaGain
float
扫描周期
ScanCycle
double
天线高度
AntennaHeight
float
信号处理损耗
SignalProcessLoss
float
信号带宽
SignalBandwidth
float
重量
Weight
float
最大俯仰角
MaxPitch
float
最大目标数
MaxTrackNumber
int
最大探测距离
最小俯仰角
最小可探测信号功率
最小探测距离
已选因子
因子名称
所属对象
作用范围
水平个数
最大探测距离
ESM1
型号
5
因子水平设置
序号
水平值
1
112.5
2
175.
3
237.5
4
300.
5
50.
请选择水平
更改水平值

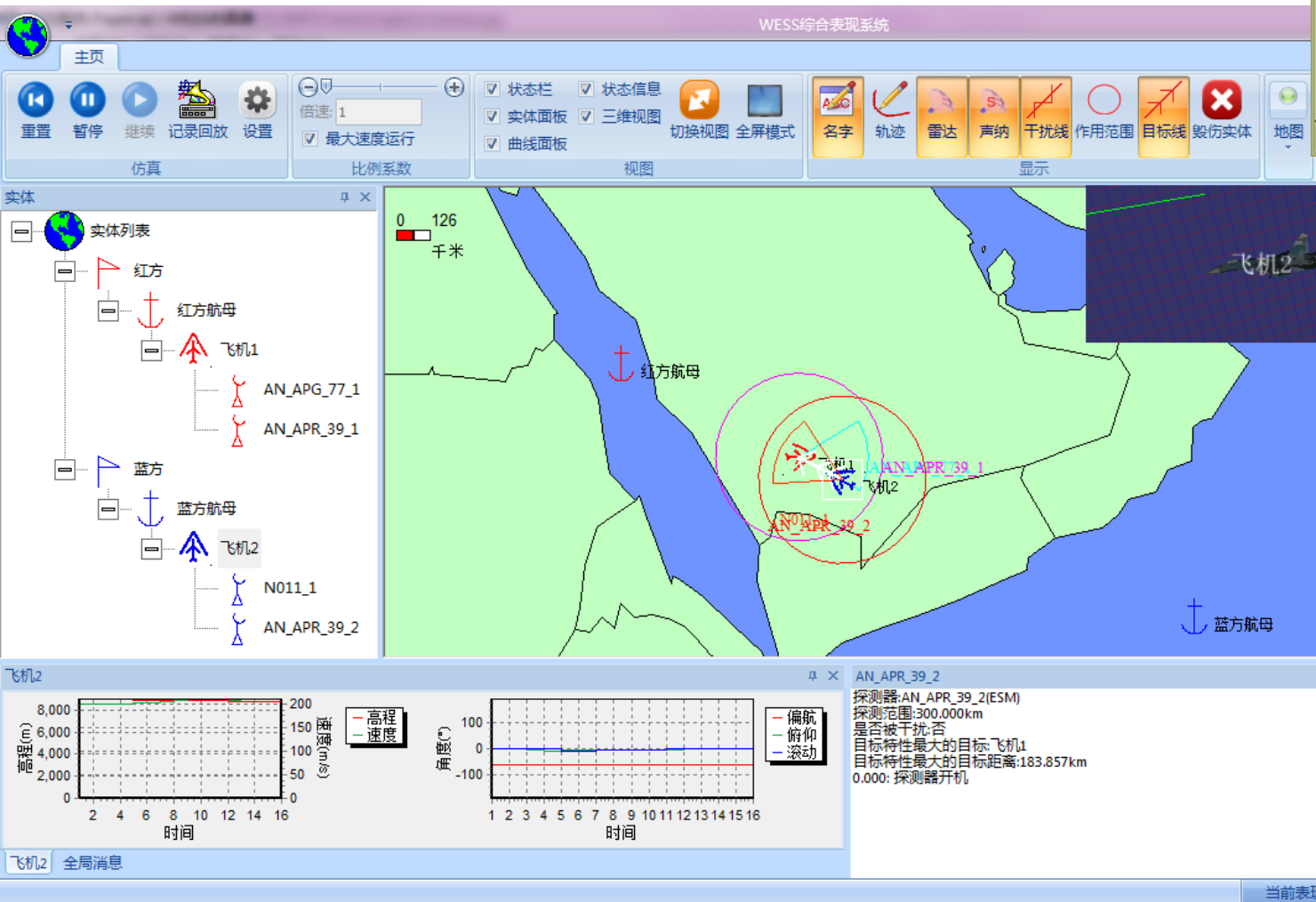

WESS仿真实验设计 - 想定1.oed
基本信息
实验因子
实验方案
实验响应
实验方案列表
全面设计表
雷达反射截面积_想定1.红方.苏27长机
最大探测距离_R...
实验1 2. 200. 100.
实验2 2. 200. 150.
实验3 2. 200. 50.
实验4 2. 300. 100.
实验5 2. 300. 150.
实验6 2. 300. 50.
实验7 3.5 200. 100.
实验8 3.5 200. 150.
实验9 3.5 200. 50.
实验10 3.5 300. 100.
实验11 3.5 300. 150.
实验12 3.5 300. 50.
实验13 5. 200. 100.
实验14 5. 200. 150.


WESS仿真实验设计 - 空空案例.oed
实验因子
实验方案
实验响应
空空案例-Paper
红方
红方航母
飞机1
AN/AP
AN/AP
WGU-
AIM-1
蓝方
蓝方航母
飞机2
N011(
AN/AP
9B-11
R-27E
数据代号
数据名称
数据类别
rLockTargetTimes 各种雷达锁定对方的次数 终态数据
rFirstLockTime 首次锁定目标时刻 终态数据
rFirstRadarWarningTime 首次收到被敌雷达锁定报告时刻 终态数据
rFirstLockDistance 首次锁定目标距离 终态数据
rFirstRadarWarningDistance 首次收到被敌雷达锁定报告距离 终态数据
rFirstMissileWarningTime 首次被导弹锁定时刻 终态数据
rFirstMissileWarningDistance 首次被导弹锁定距离 终态数据
sAccelN 法向加速度 状态数据
sClimbRate 爬升率 状态数据
sRemainFuel 剩余燃油 状态数据
sNy 法向过载 状态数据
sNh 法向过载的水平分量 状态数据
sNv 法向过载的垂直分量 状态数据
sNx 切向过载 状态数据
rMissionSuccess 是否完成任务 终态数据
rMissionSuccessTime 完成任务时间 终态数据
rHitWeaponCount 被命中武器数 终态数据
rFoundTargetTime 首次发现目标时间 终态数据
rLostTargetTime 最后丢失目标时间 终态数据
rWeaponCountLaunched 发射武器数量 终态数据
添加表达式终值
全部终值
全部
全部不选
数据名称
数据类型
所属对象
对象类型
清除状态数据
清除全部

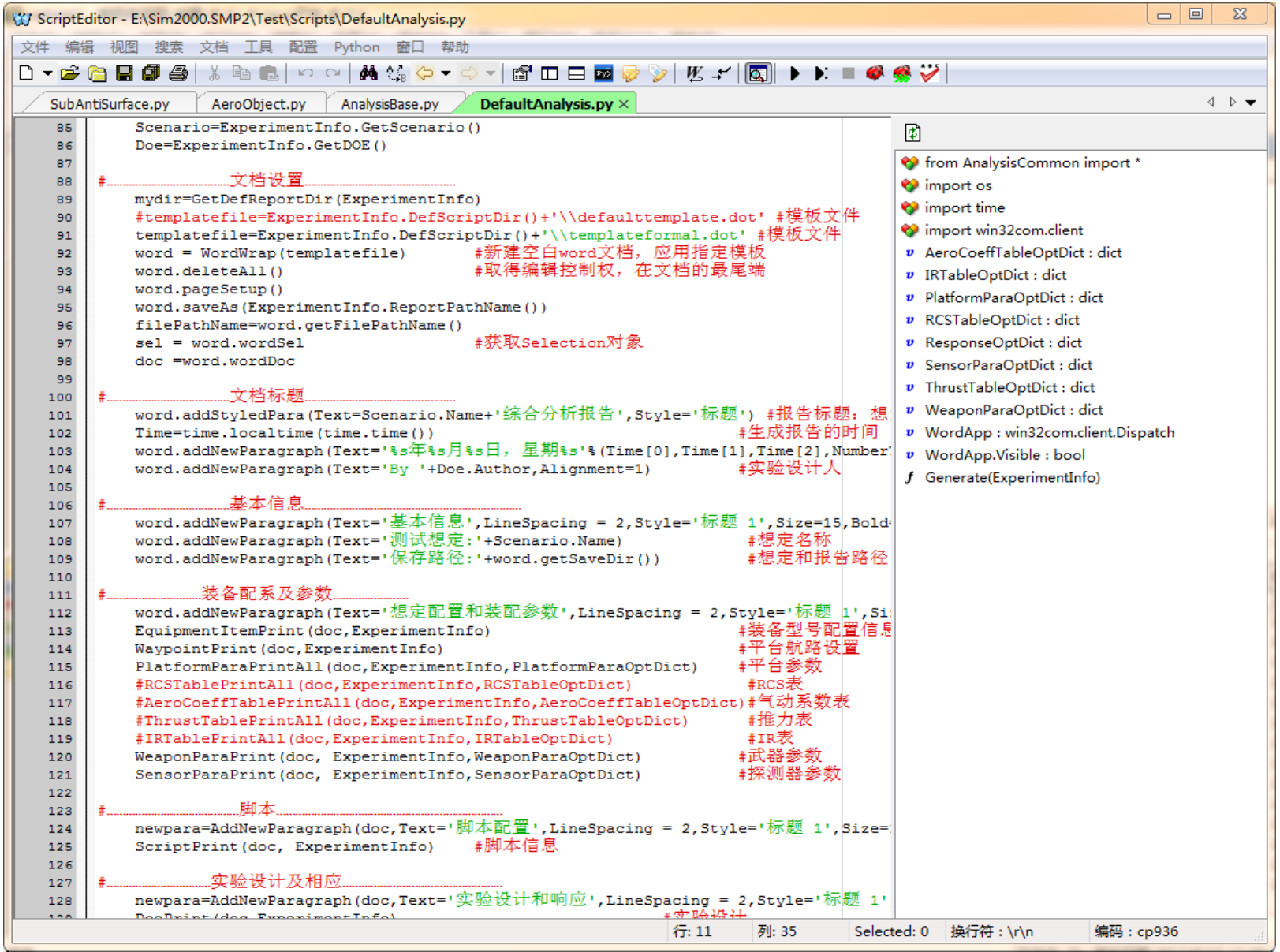
Simulator


AirCombatExample - WESS仿真器
文件(F) 查看(V) 运控(R) 编辑(E) 帮助(H)
分布表现 实时运行 1 倍实时
仿真配置与控制
框架文件... 案例\AirCombatExample.assembly
模型路径... C:\WESS\SimModels
实验文件... C:\WESS\Projects\空空案例\AirC
AirCombatExample
红方
RedCarrier
FighterB
Radar2_1
ESM2_1
Missile4_1
Seeker2_1
蓝方
BlueCarrier
FighterA
Radar1_1
ESM1_1
Missile3_1
Seeker1_1
Missile3_2
Seeker1_2
高低-15.73度；目标特性0;信噪比0
[ESM2_1] : 300.200: 目标【FighterA】的类型识别为:[飞机]!
[Missile3_2] : 301.000: 模型创建（ID=16,Missile3_2,tmFarAAM）
[Missile3_2] : 301.000: 注意：未装配引信，导引头将控制引爆战斗部!
[Seeker1_2] : 301.000: 模型创建（ID=17,Seeker1_2,tmActiveRadarSeeker）
[FighterA] : 301.000: 建立火控关系：[Radar1_1]+[Missile3_2]->[FighterB]
[FighterA] : 301.000: 目标1点钟方向,距离36.990km
[FighterA] : 301.000: [脚本] 向(FighterB)发射导弹:规避导弹
[FighterA] : 301.000: 导弹Missile4_1距离18780.2方位0.532963转弯过载-5
[FighterA] : 302.000: 朝目标FighterB发射Missile3型中远距空空导弹[Missile3_2]
[Missile3_2] : 302.000: 预定攻击目标为：FighterB
[Missile3_2] : 302.000: 坐标原点为：46.122565,19.168440,9006.322364
[Missile3_2] : 302.000: 发射方向：俯仰0度，方位126度
[Missile3_2] : 302.000: 预测目标点为：(46.2255,18.8511,8985)
[FighterA] : 302.000: 导弹Missile4_1距离17544.5方位0.794871转弯过载-5
[Radar2_1] : 302.010: 发现目标【Missile3_2】，距离：36558米；方位7.49度；高低17.08度；目标特性0.512603;信噪比9.5652
[Radar2_1] : 302.010: 目标【Missile3_2】的类型识别为:[拦截弹]!
[FighterB] : 302.010: [脚本] 发现目标(Missile3_2),继续远距空战
[FighterA] : 303.000: 导弹Missile4_1距离17437.6方位1.13645转弯过载-4.86355
[FighterA] : 304.000: 导弹Missile4_1距离17360.1方位1.58076转弯过载-4.41924
[FighterA] : 305.000: 导弹Missile4_1距离16791.2方位2.14908转弯过载-3.85092
[FighterA] : 306.000: 导弹Missile4_1距离16767.6方位2.71917转弯过载-3.28083
[Missile3_2] : 306.500: 助推段结束，进入中段
[FighterA] : 307.000: 导弹Missile4_1距离16764.4方位3.08817转弯过载-2.91183
[FighterA] : 308.000: 导弹Missile4_1距离16775.4方位3.42171转弯过载-2.57829
[FighterA] : 309.000: 导弹Missile4_1距离16800.6方位3.7437转弯过载-2.2563
[Radar2_1] : 309.090: 丢失目标：FighterA，距离(34150米)，原因是:未能进入候选探测目标列表
模型数据
名称 值 类型 类别
sLat 19.166637 Float32 状态
sLon 46.135422 Float32 状态
sSpeed 199.372559 Float32 状态
sAzimuth 82.653549 Float32 状态
sAlt 8988.966797 Float32 状态
sElevation -3.526640 Float32 状态
sRoll -65.685135 Float32 状态
sPitch -3.526640 Float32 状态
sHeading 82.653549 Float32 状态
sVoyage 74828.507813 Float32 状态
sAccel 1.363657 Float32 状态
sTurnRate -3.452971 Float32 状态
sThrust 49411.031250 Float32 状态
sDrag 34136.628906 Float32 状态
sMostRiskTa... Missile4_1 NameString 状态
sNextWPLon 46.500000 Float32 状态
sNextWPLat 19.280001 Float32 状态
sNextWPAlt 9000.000000 Float32 状态
sRemainMiss... 2 Int32 状态
sRemainTorp... 0 Int32 状态
sRemainCMCount 0 Int32 状态
sTargetCount 2 Int32 状态
模型消息
0.000: 模型创建（ID=9,FighterA,tmAeroObject）
0.000: 导入默认脚本【C:\WESS\Scripts\AeroObject.py】...
0.000: 导入成功
0.000: 继续朝第2个路径点(46.5,19.28,9000)前进,目标速度
0.000: 初始位置:(45.5,19.28,9000)，初速度(257.222),方位
0.000: 执行任务
0.000: 初始化,当前位置(45.5,19.28,9000)
0.000: [脚本] 雷达开机...
239.800: 注意：当前目标已更换为【FighterB】
239.800: [脚本] 空中目标(FighterB):常规巡航=>远距空战
240.000: [脚本] 目标方位68.2316：旋转雷达[Radar1_1]30度
246.000: [脚本] 发现目标(Missile4_1),继续远距空战
246.000: [脚本] 雷达探测导弹,距离>20KM,继续远距空战
290.000: 建立火控关系：[Radar1_1]+[Missile3_1]->[Fighte
290.000: 目标0点钟方向,距离42.931km
290.000: [脚本] 向(FighterB)发射导弹:远距空战=>制导飞行
WESS效能仿真系统
共1次，第1次
当前步长：0.030
309.330

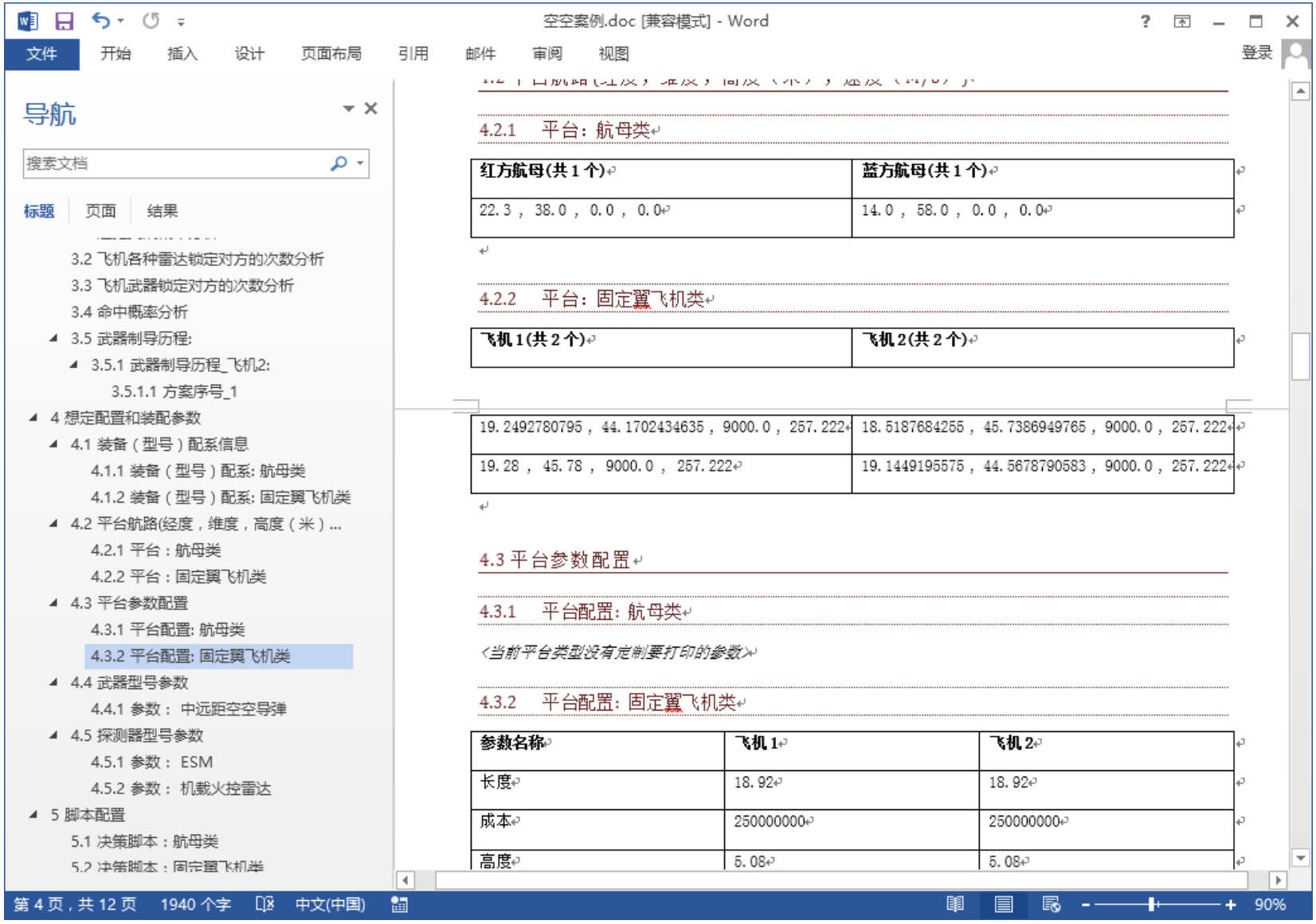
SimDisplay and SimLogger



OutputAnalyzer-analysis script



OutputAnalyzer-generated report

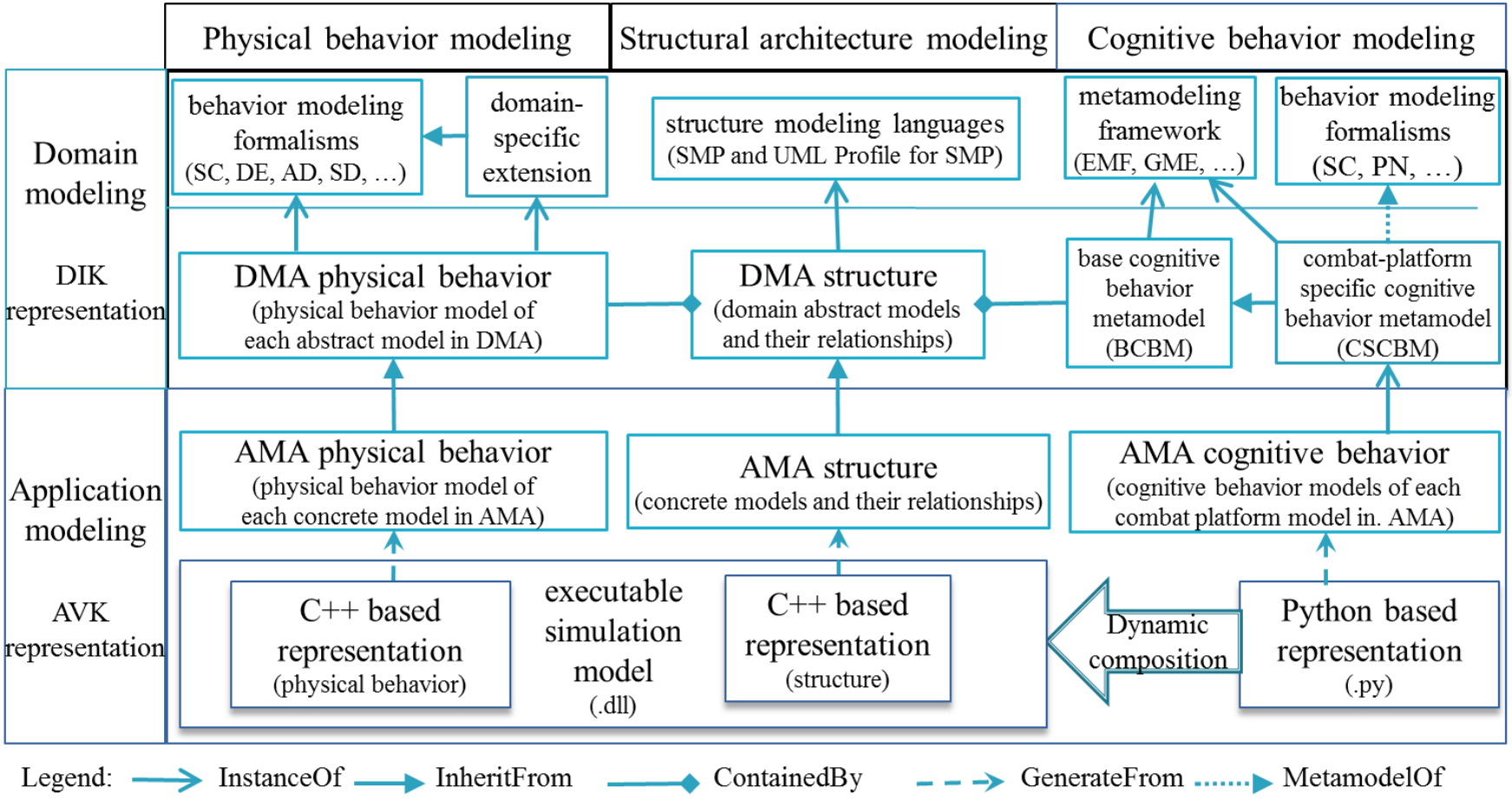


3. Generic model architecture

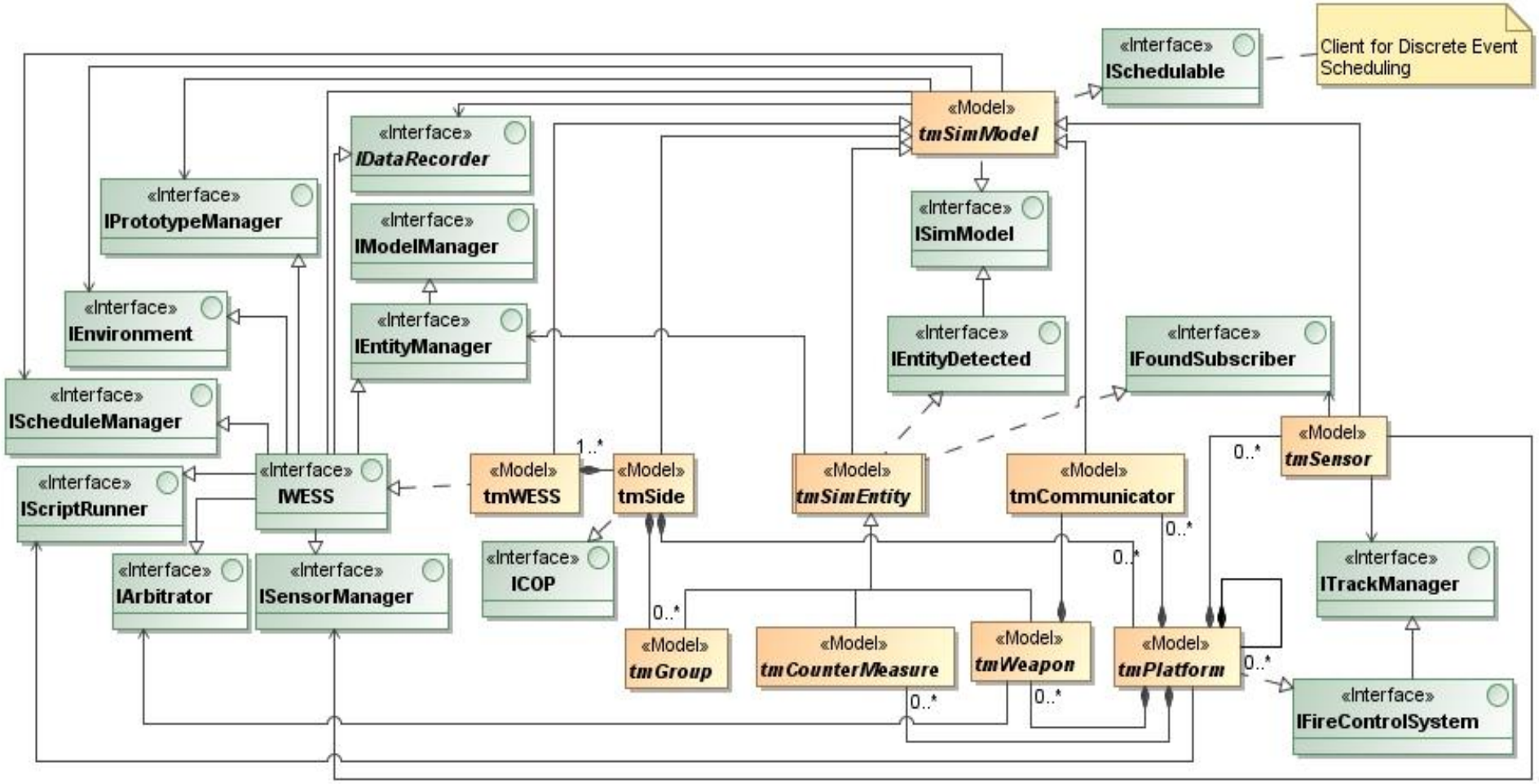
- Rational
  - Targeted to engagement-level simulation, few to few
  - Platform-centric combat with net-centric support
  - Separation Domain MA from Application MA
    - All models within DMA are abstract, i.e. non-instantiable
    - All models within AMA are concrete, i.e. instantiable
    - AMAs inherit the DMA
  - Separation cognitive behaviors from physical behaviors
    - Stable physical behaviors represented in C++
    - Variable cognitive behaviors represented in Python
    - Cognitive modeling interface
  - Formalized structure representation based on SMP
  - Quasi-formalized behavior representation

3. Generic model architecture

- Modeling framework of WESS model architecture



Top view of the model architecture

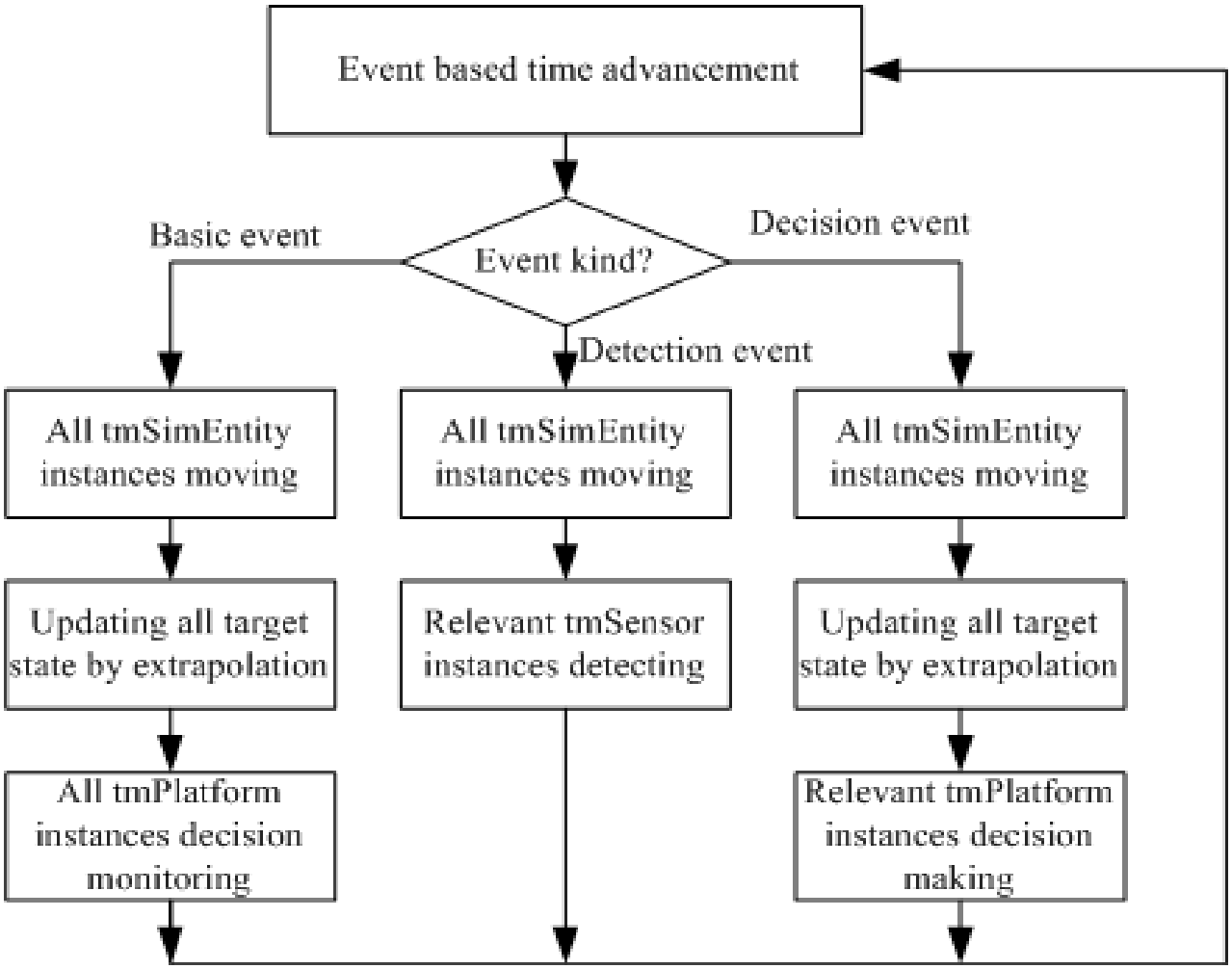


Top view of the model architecture

- Abstract event-based time advancement algorithm
  - Basically, the motion model of every entity, i.e. subclass of *tmSimEntity* model, is continuous. To advance time, the simulation time should be discretized with a basic step. At the functional modeling level, the detection model of a sensor is discrete time, probably with time-varying steps. For the platform model excluding motion, another important functionality is to invoke its cognitive behavior model for tactical decision-making over all simulated time when the platform is "alive" in the mission. To represent these cognitive behaviors, both discrete time and discrete event time advance mechanisms are demanded. The former is applied generally for regular situation monitoring; whereas the latter is useful to schedule a decision point ahead of a certain time already known in the moment or given the quantified condition under which a decision would be made.
  - To precisely and efficiently model these different time advance requirements, a discrete event time advance mechanism is chosen by the simulator. Each model will register its time advance requirements to simulator in terms of simulator events for discrete event or cycle events for discrete time. For this sake, two kinds of events are available for registration. One is **decision event**, and the other is **detection event**. For the continuous time advance requirement, a basic step specified by the user is used to register the third event kind, i.e., **basic event**. When a simulator event is triggered, depending on its kind, different execution sequences happen as shown in next slide.

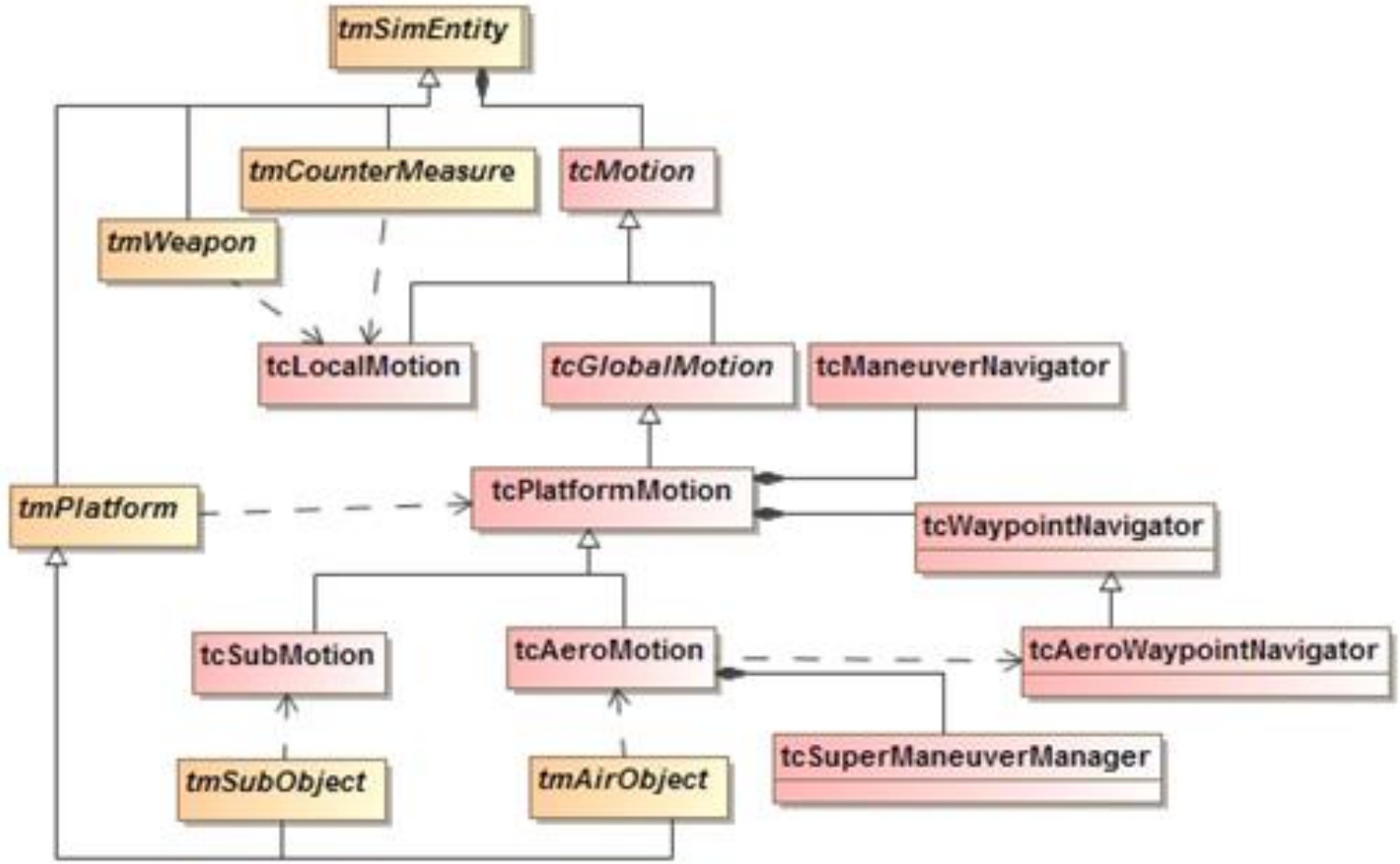
Top view of the model architecture

- Abstract event-based time advancement algorithm

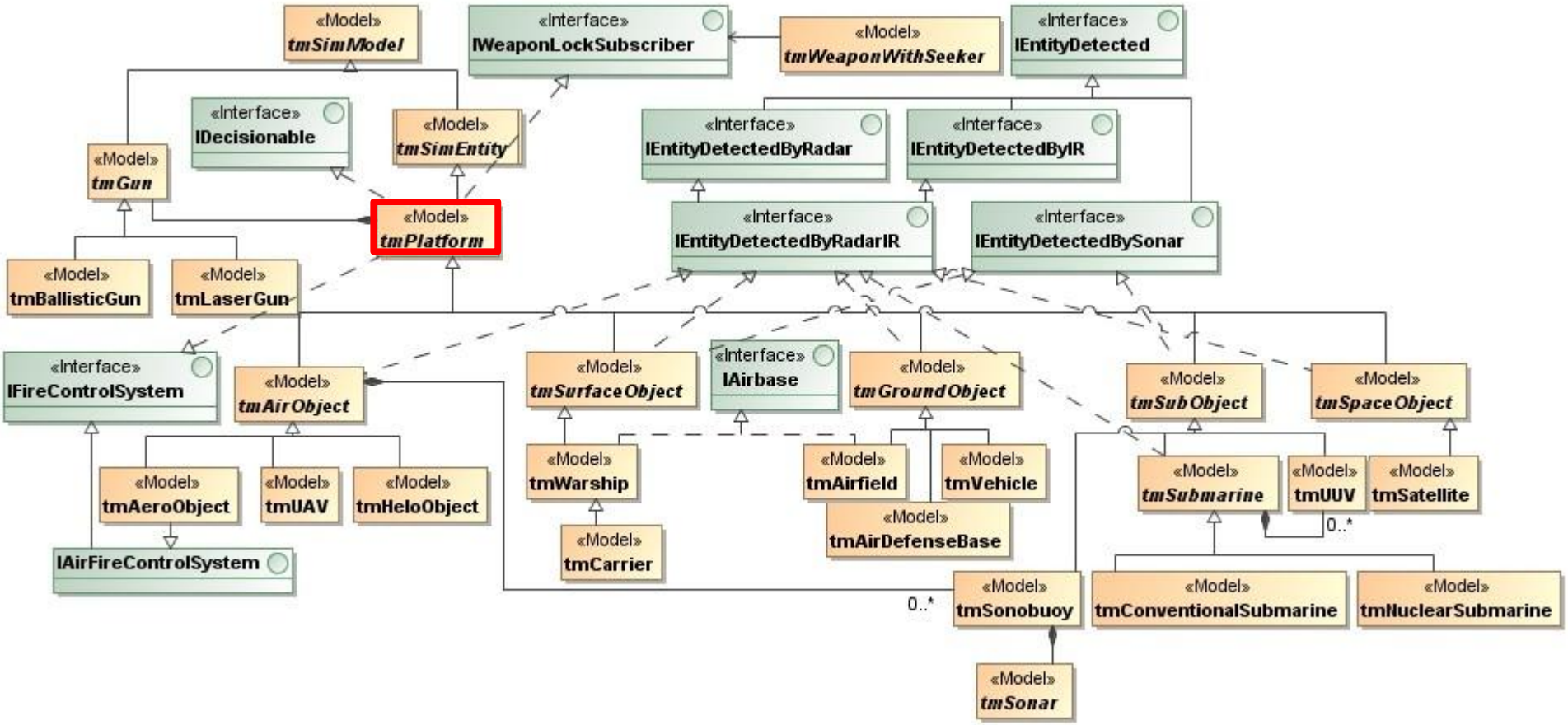


Top view of the model architecture

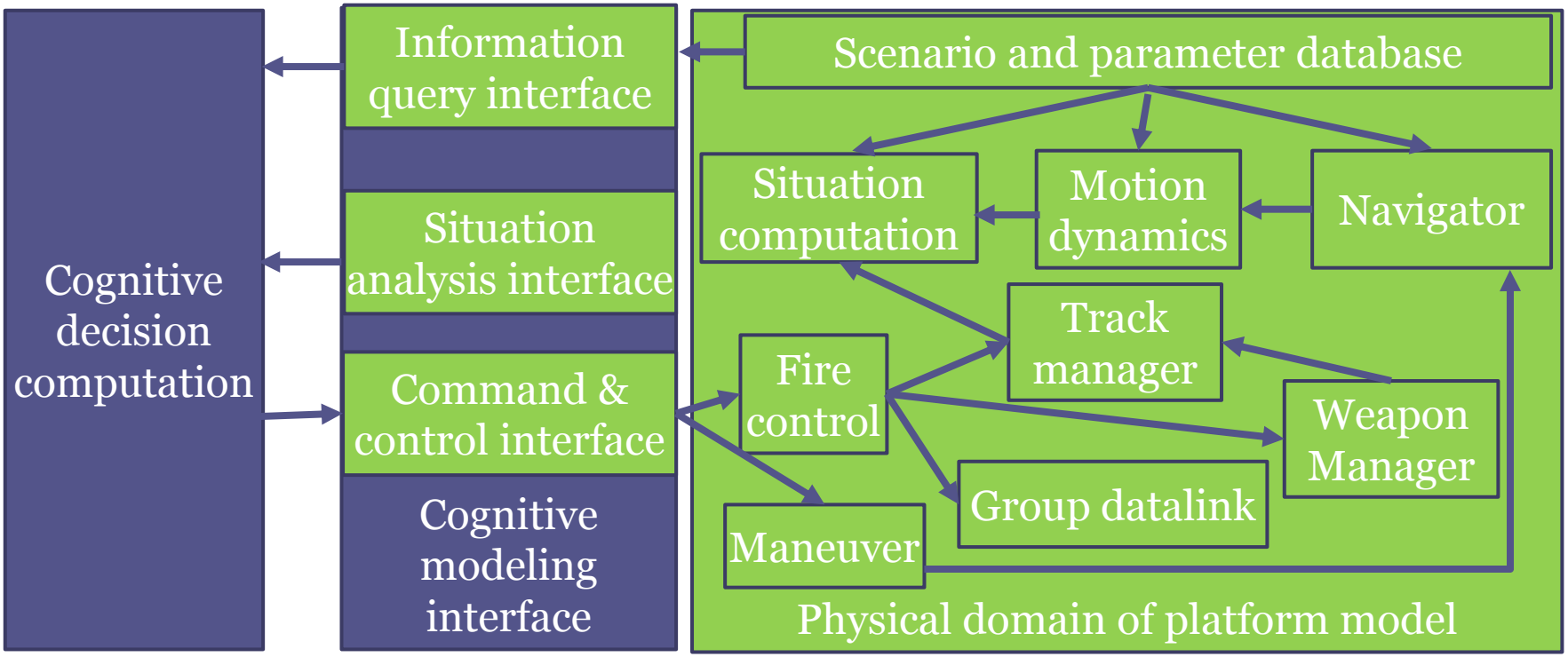
- Abstraction-oriented motion modeling



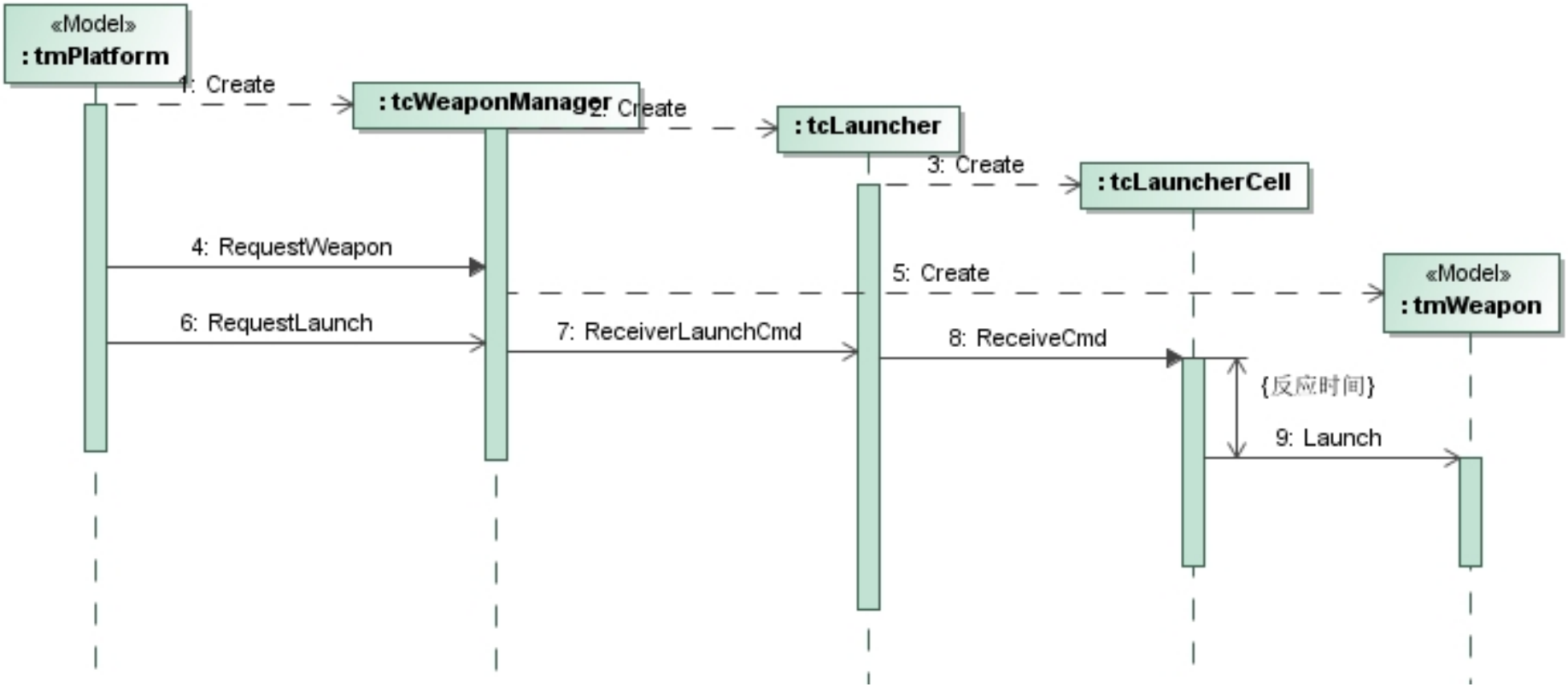
Platform model architecture



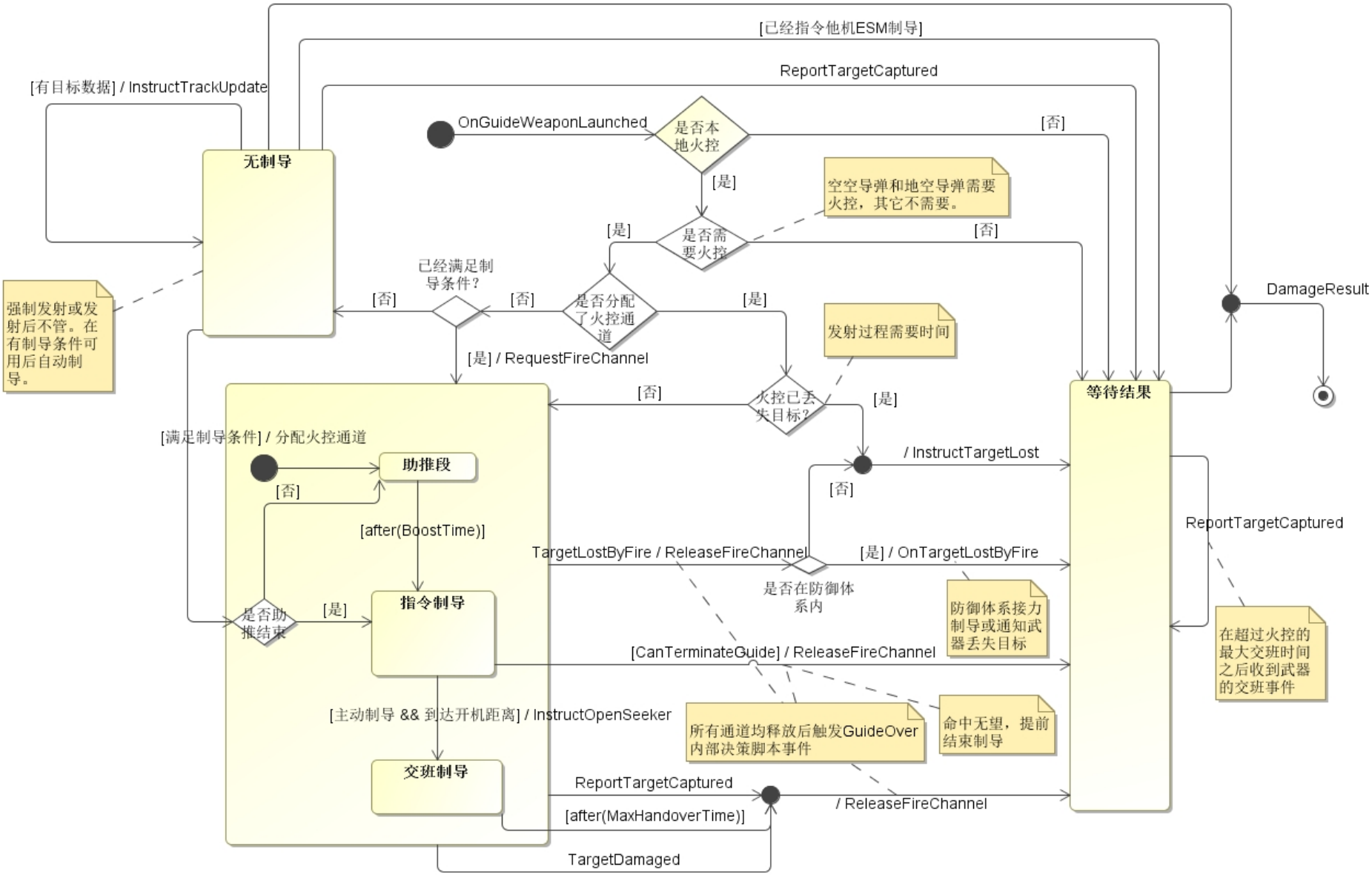
Modeling framework of combat platform models



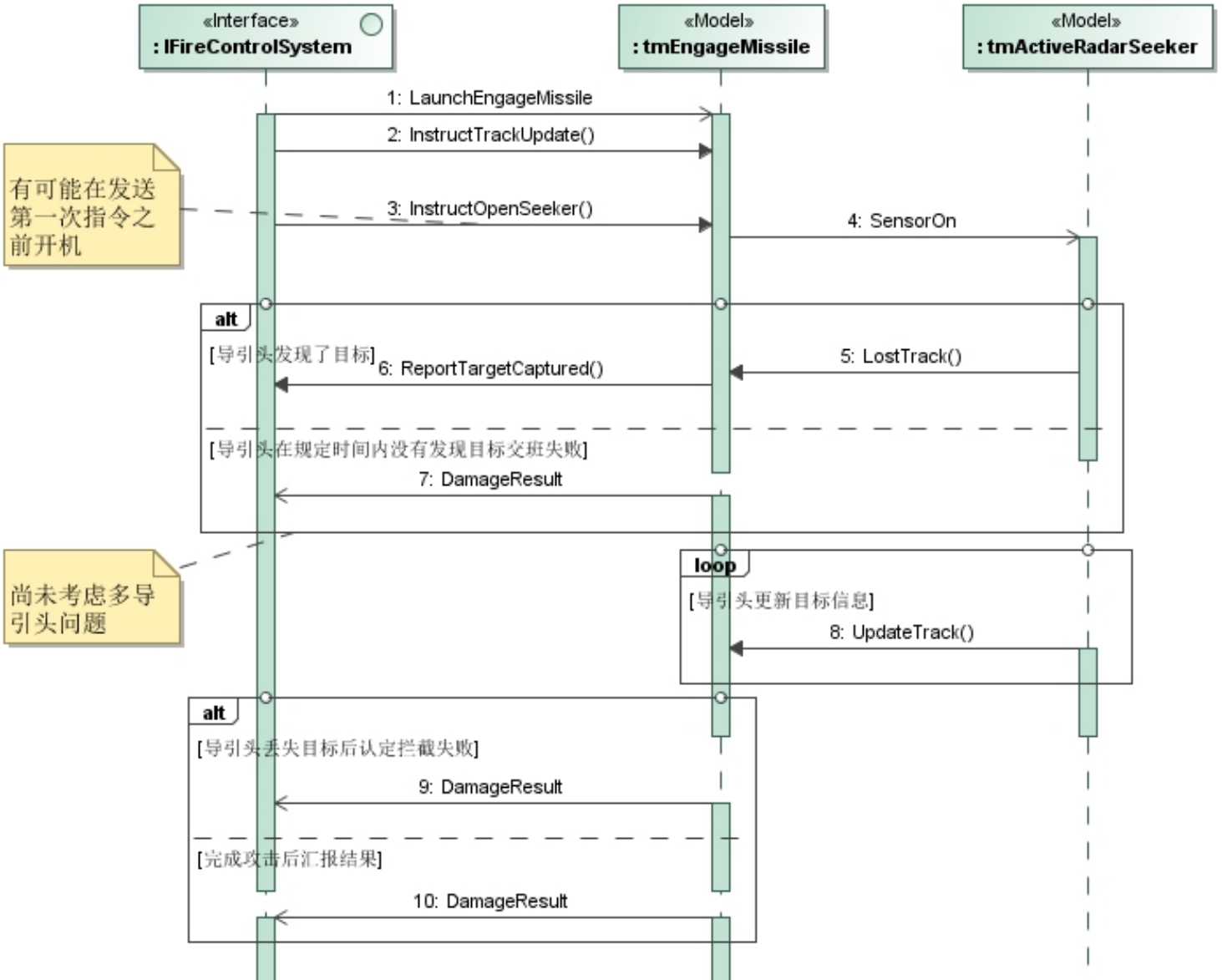
Weapon launch management



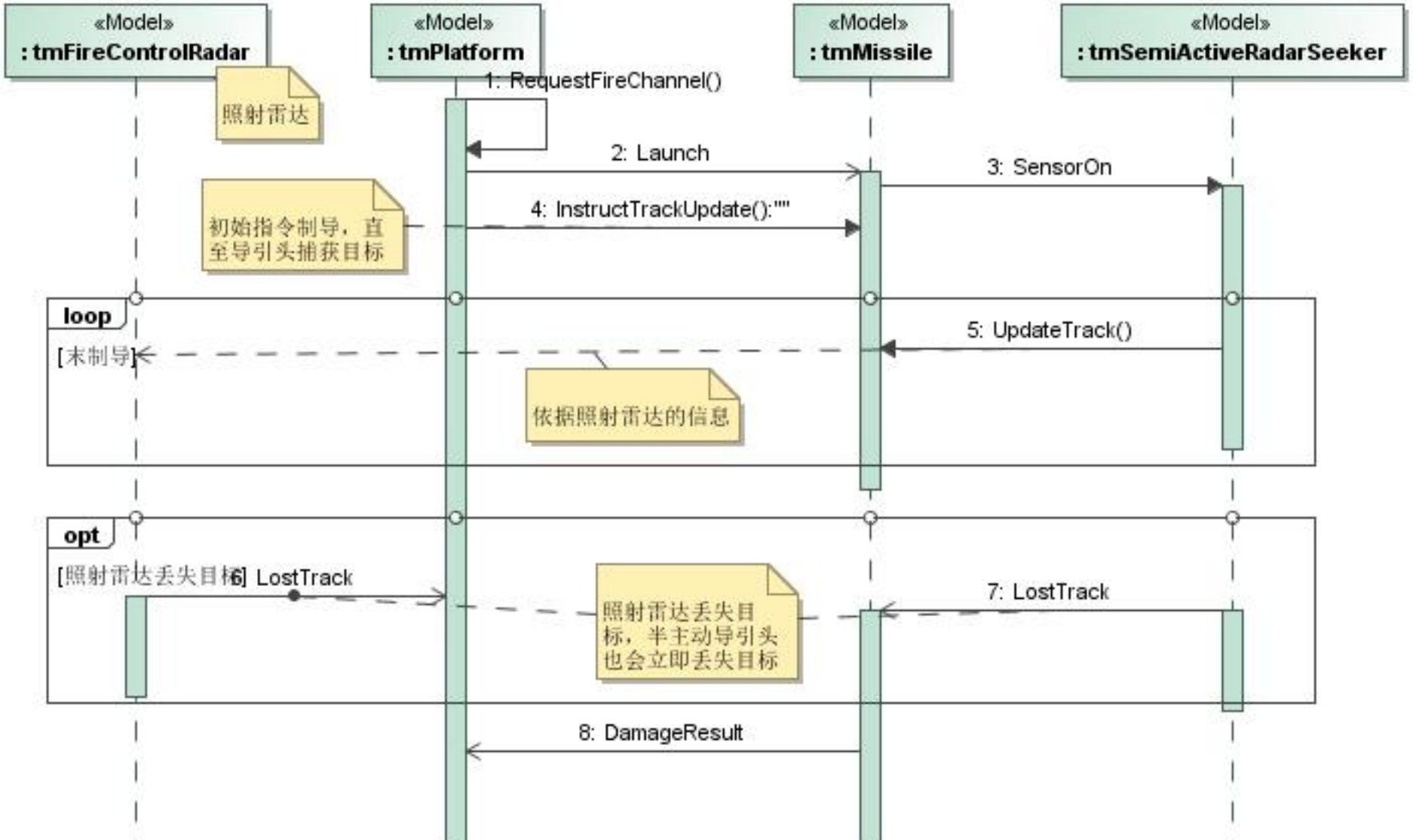
Fire control weapon management



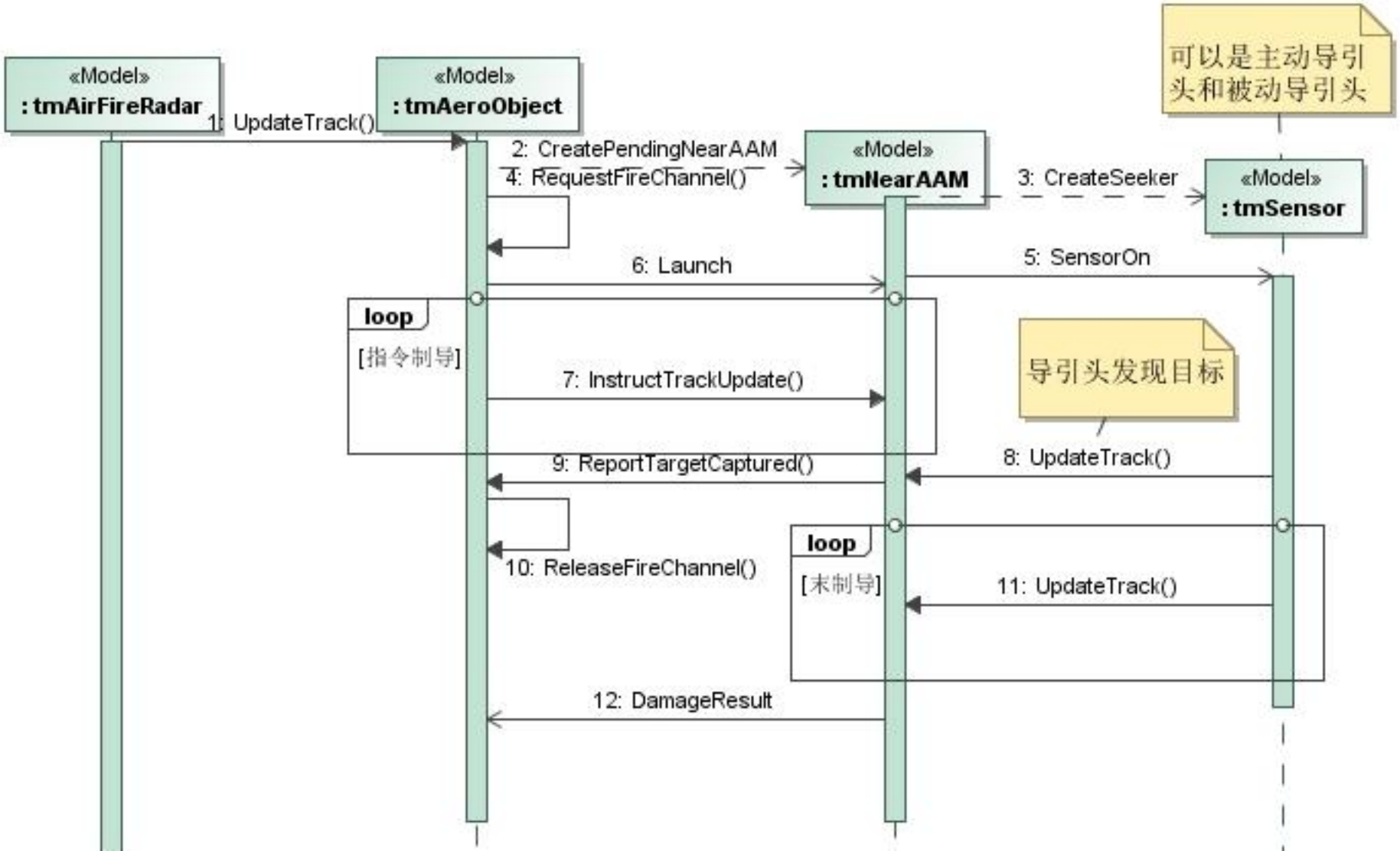
Missile guiding (active radar seeker)



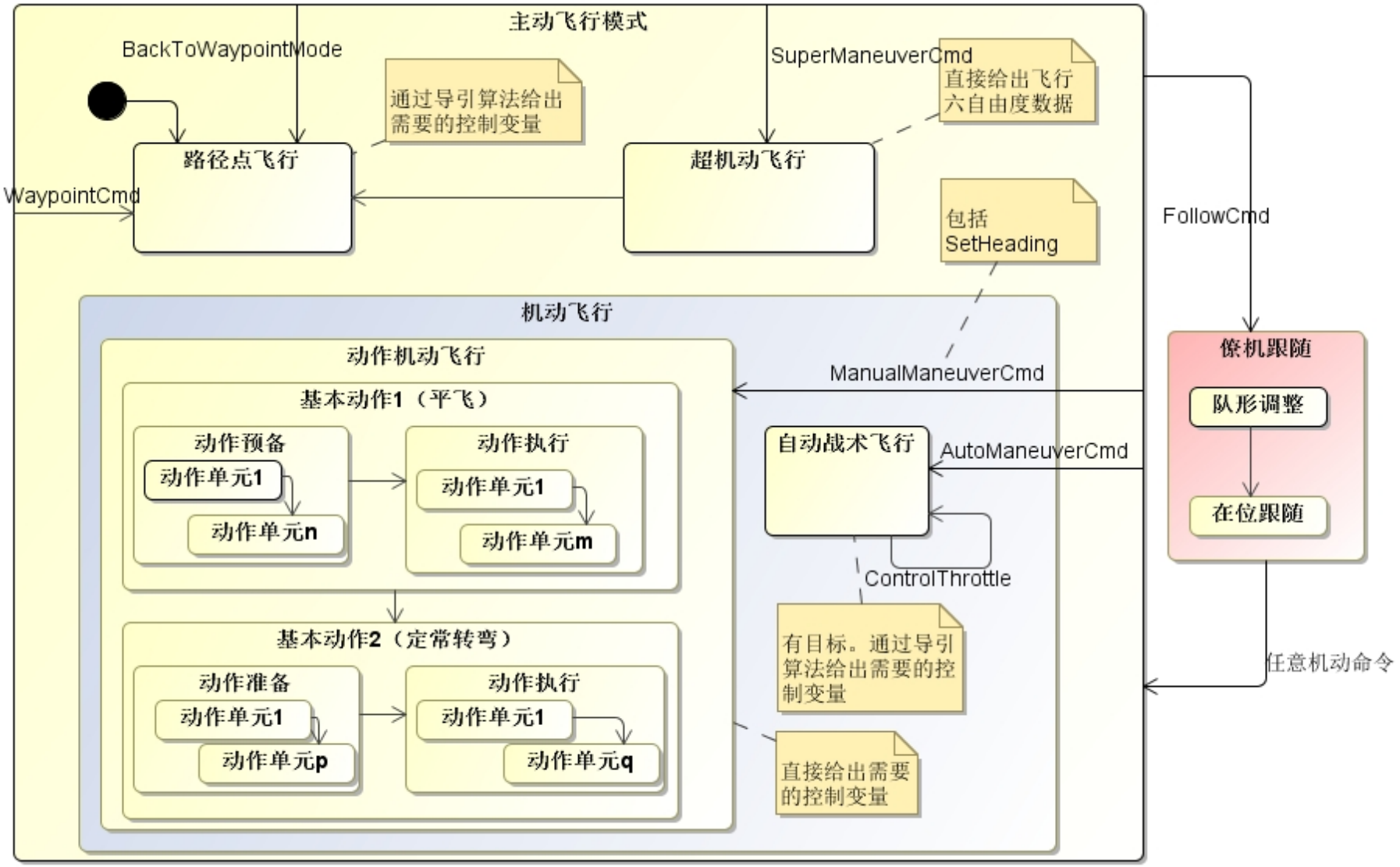
Missile guiding (semiactive radar seeker)



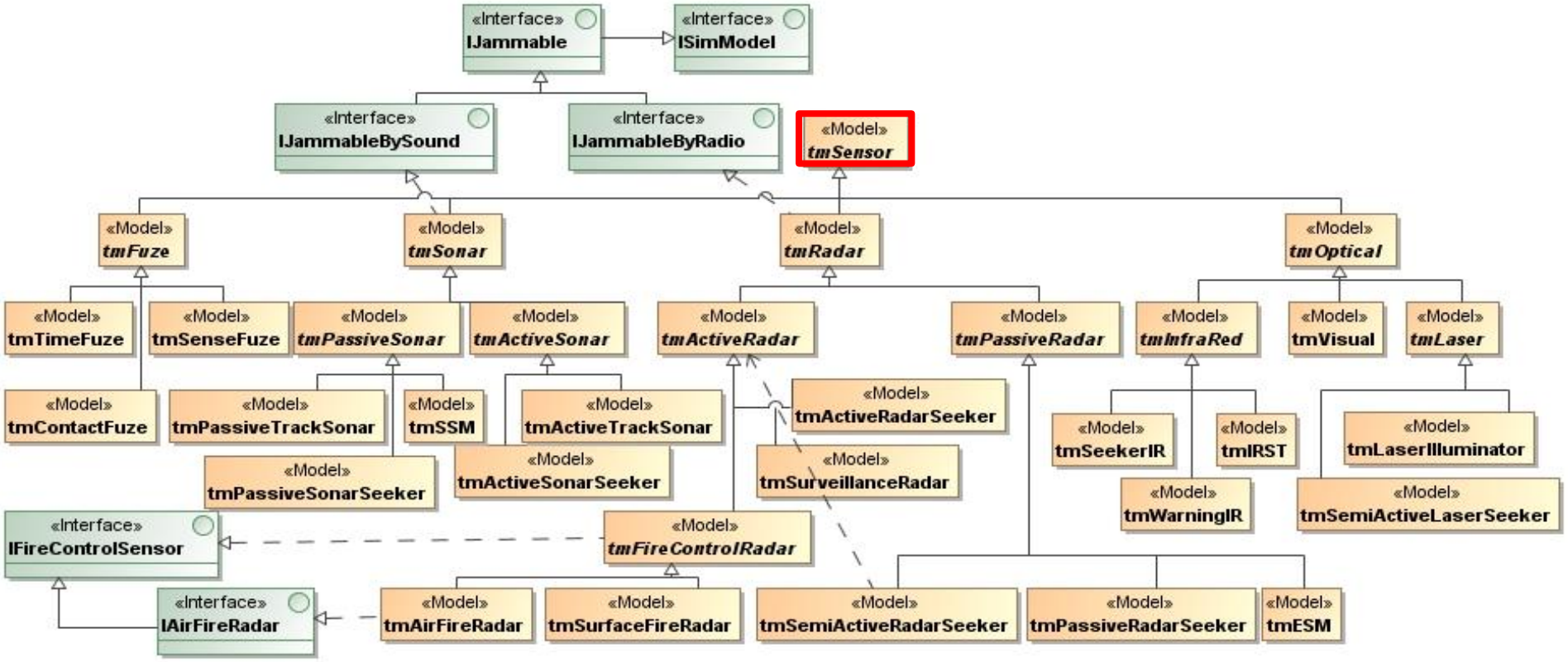
Missile guide (near air to air missile)



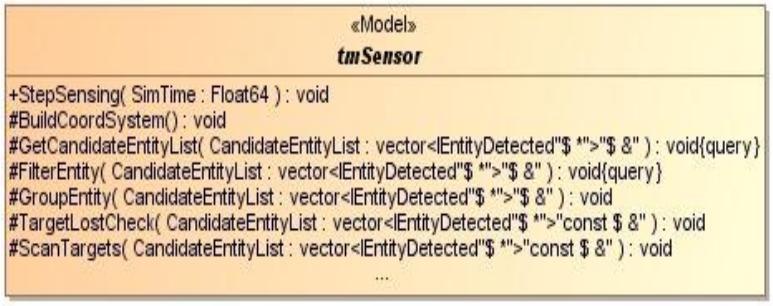
Platform motion statecharts



Sensor model architecture



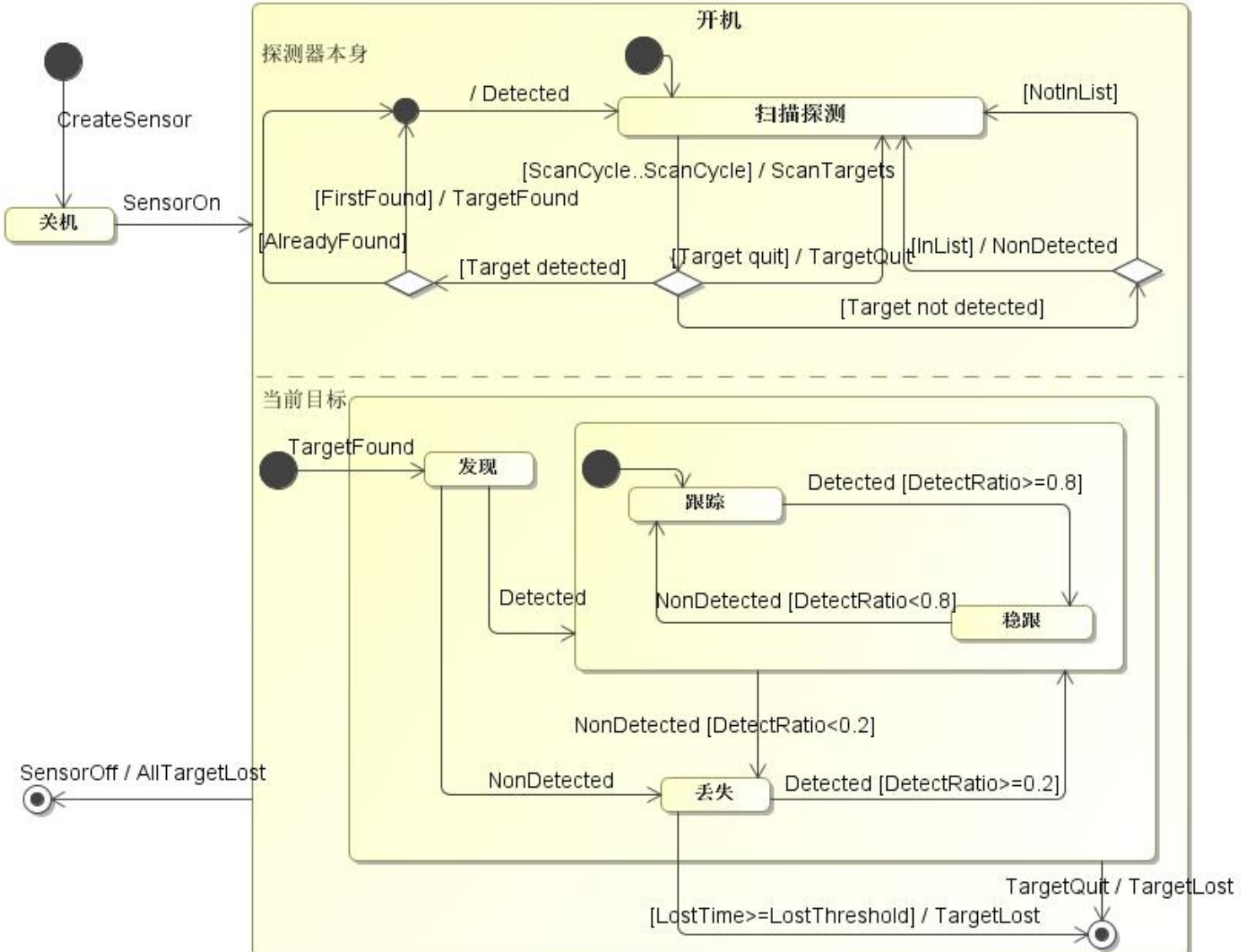
Template Method pattern based detection model



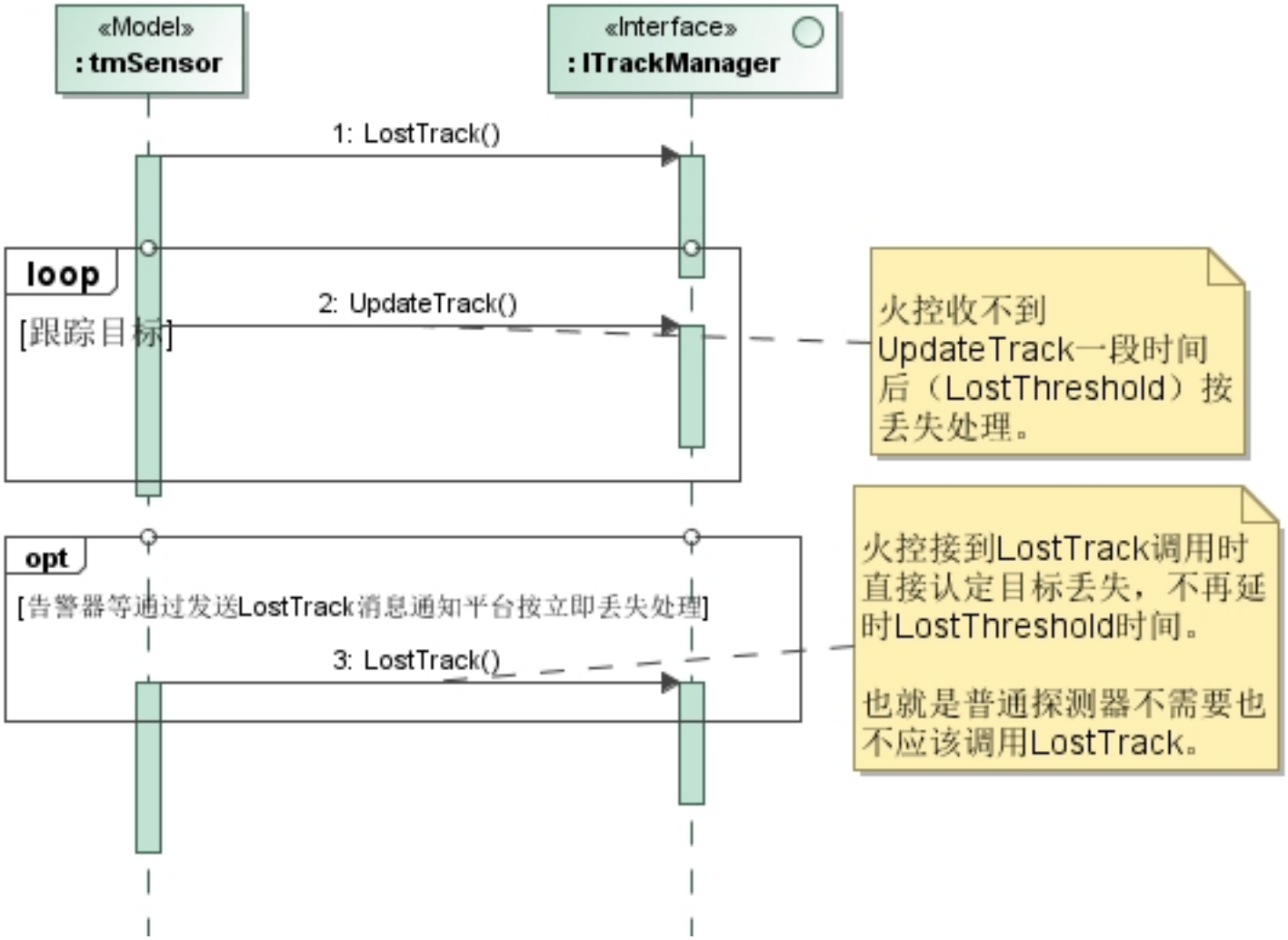

```
void tmSensor::StepSensing( Smp::Float64 SimTime )
{
    //Build local coordinate system for sensor
    BuildCoordSystem();

    //Get candidate entities within max scope
    vector<BaseModel::IEntityDetected*> CandidateEntityList;
    GetCandidateEntityList(CandidateEntityList);

    //Filter Entities according to angle limits
    FilterEntity(CandidateEntityList);

    //combine indistinguishable entities
    GroupEntity(CandidateEntityList);

    //Check local target list for possible out of sensor's scope
    TargetLostCheck(CandidateEntityList);

    //simulate detection
    ScanTargets(CandidateEntityList);

    //save states
    SaveState();
}
```

```
//处理目标的发现与否
for (unsigned i=0;i<CEL.size();i++){
    BaseModel::IEntityDetected* pTarget = CEL[i];
    if (DetectTarget(pTarget)){//探测到目标
        //不在目标列表中
        if (!AlreadyFound(pTarget->get_ID())){
            OnTargetFound(pTarget);
        }
        else{//在目标列表中
            OnTargetDetected(pTarget);
        }
    }
    else{//未探测到目标
        //之前已经发现了目标
        if (AlreadyFound(pTarget->get_ID())){
            OnTargetNonDetected(pTarget,"信号功率太弱");
        }
    }
}
```

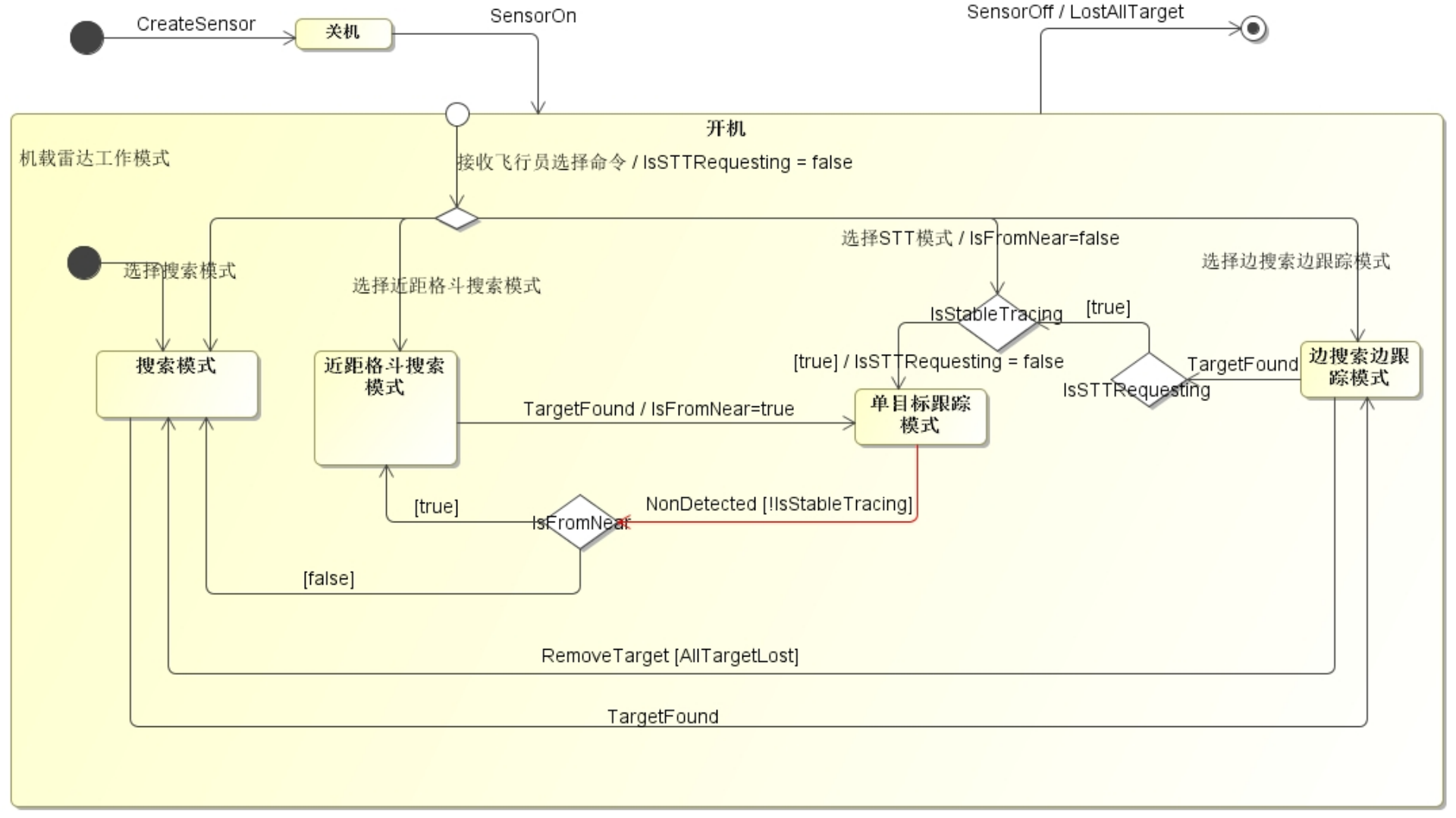
Abstract Statecharts of sensor models



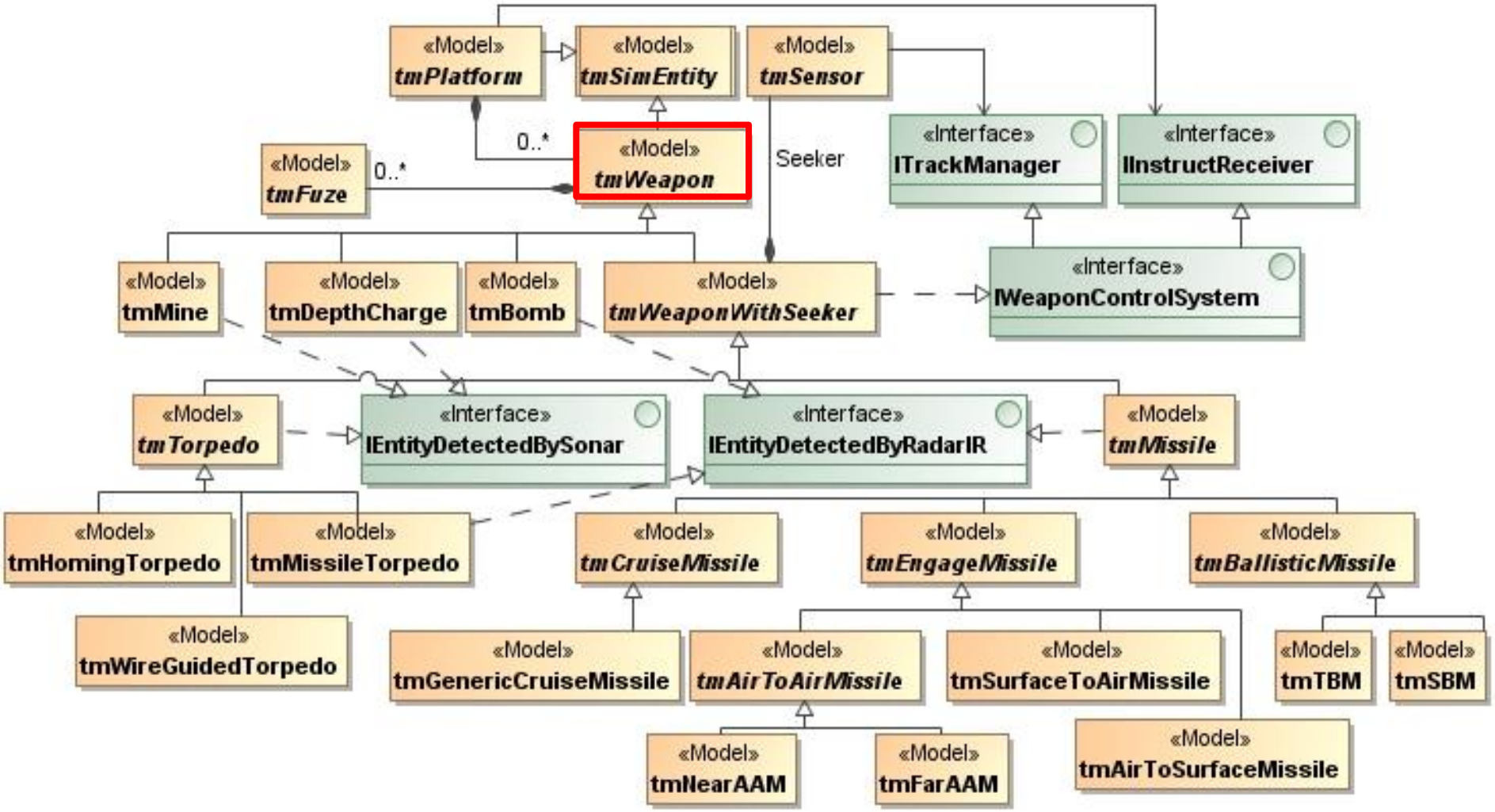
Track report



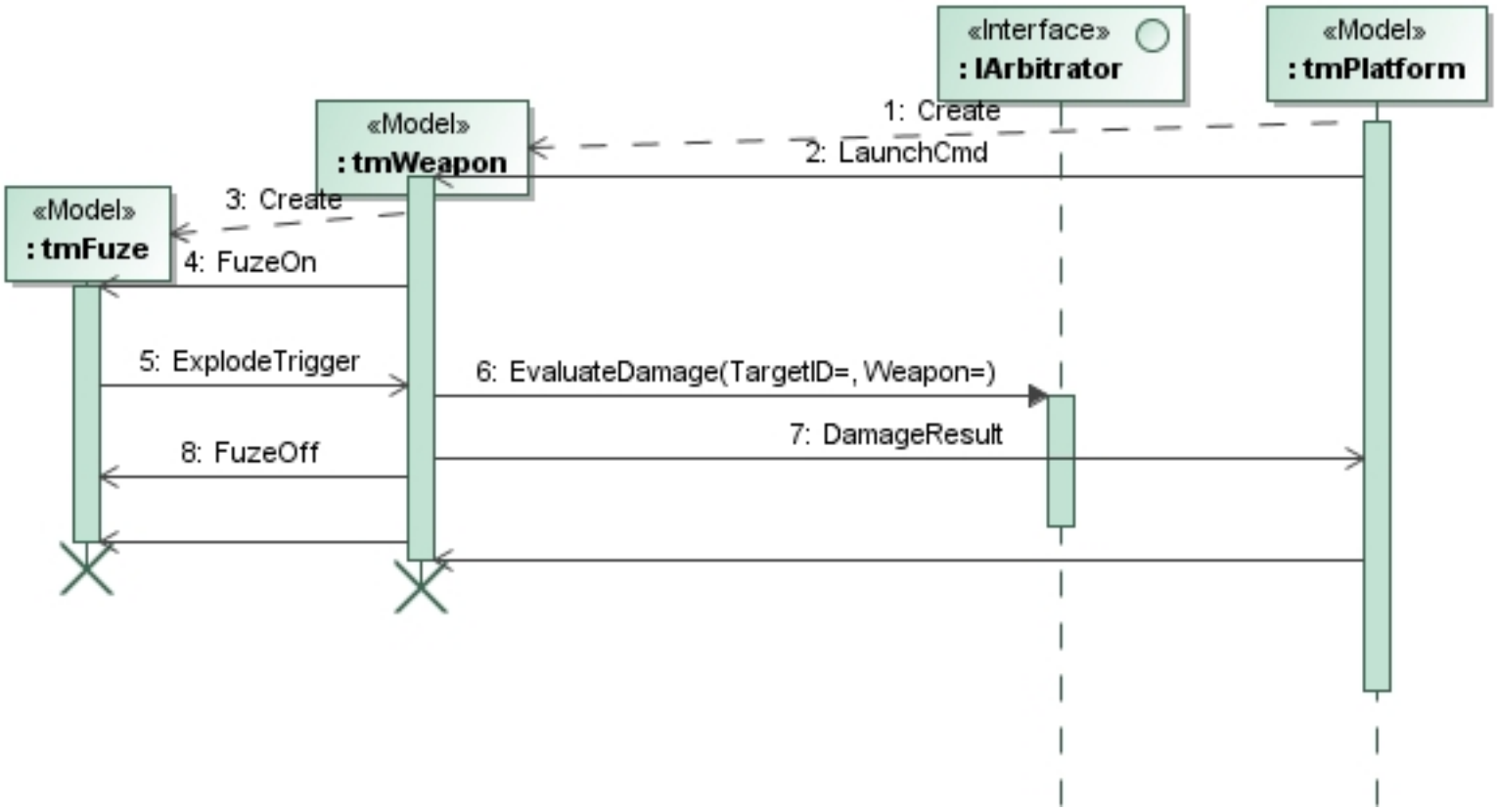
Air fire radar statecharts model



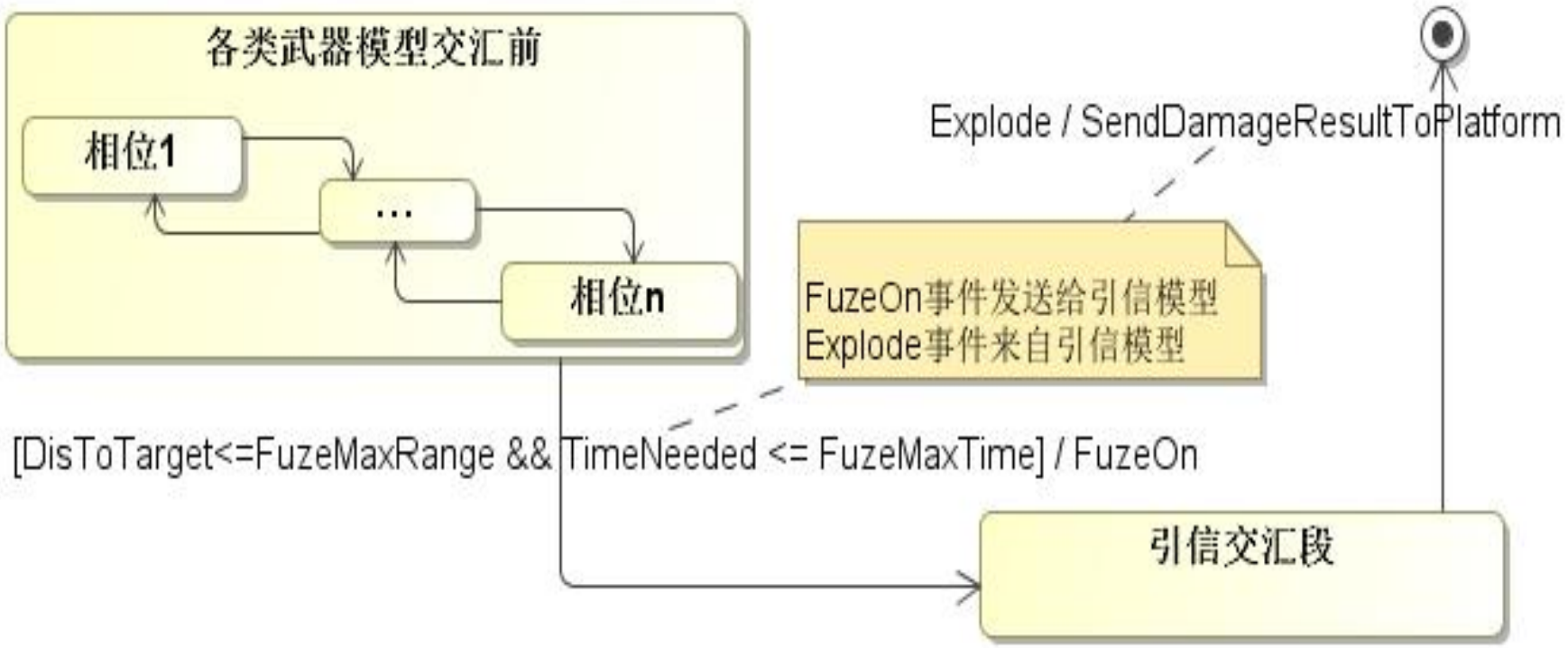
Weapon model architecture



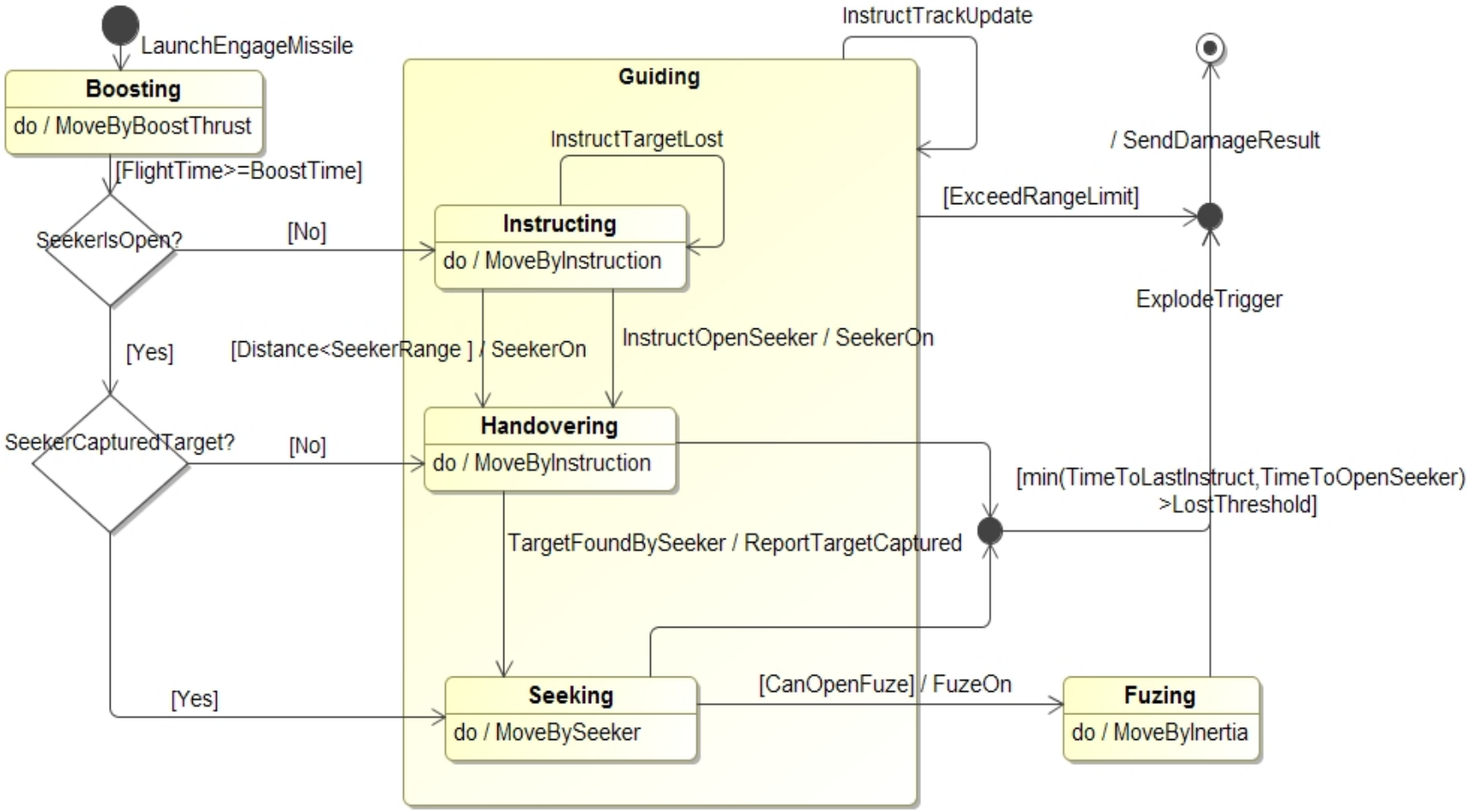
Damage evaluation and report



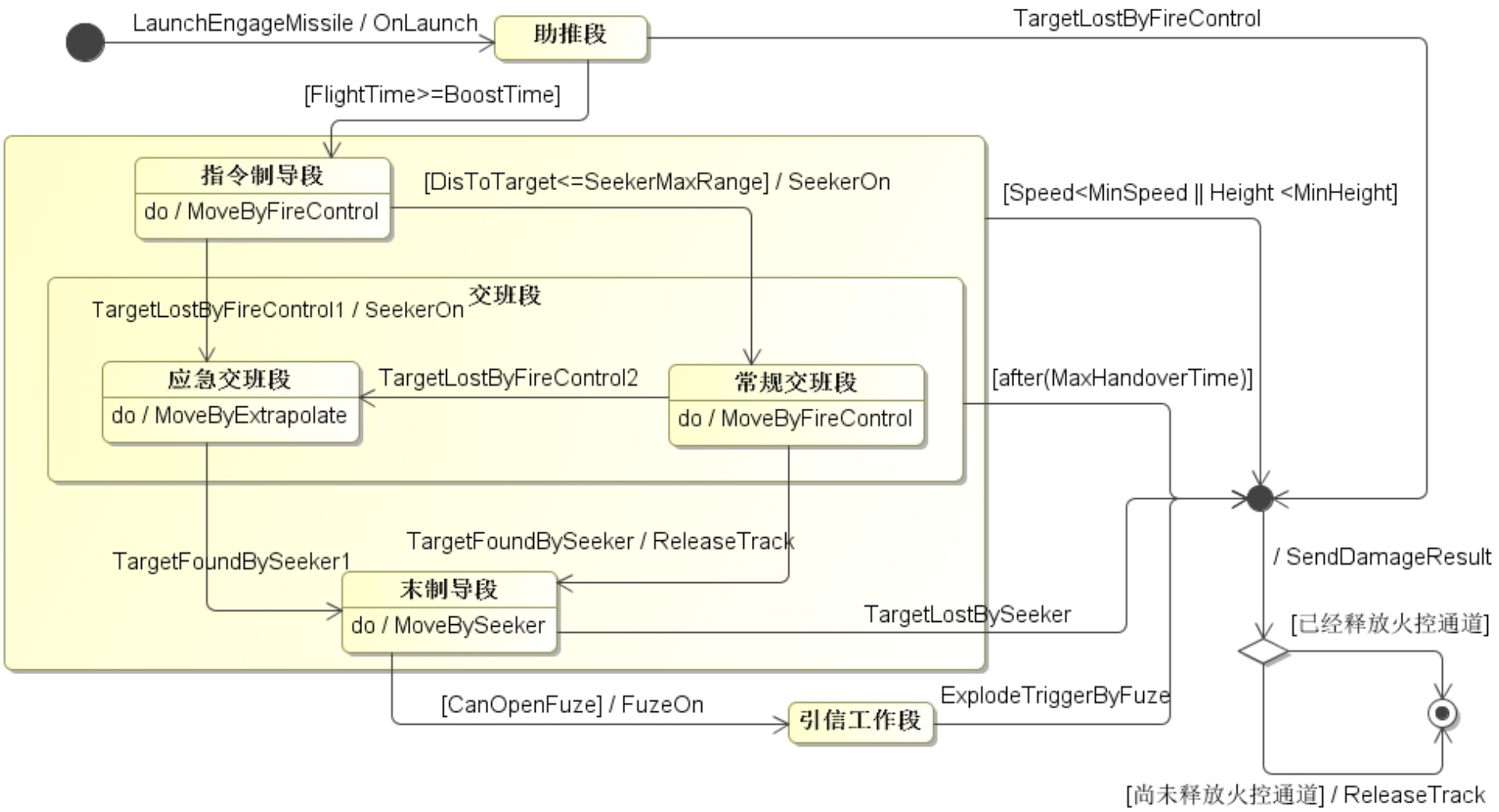
Statecharts-based guided weapon modeling framework



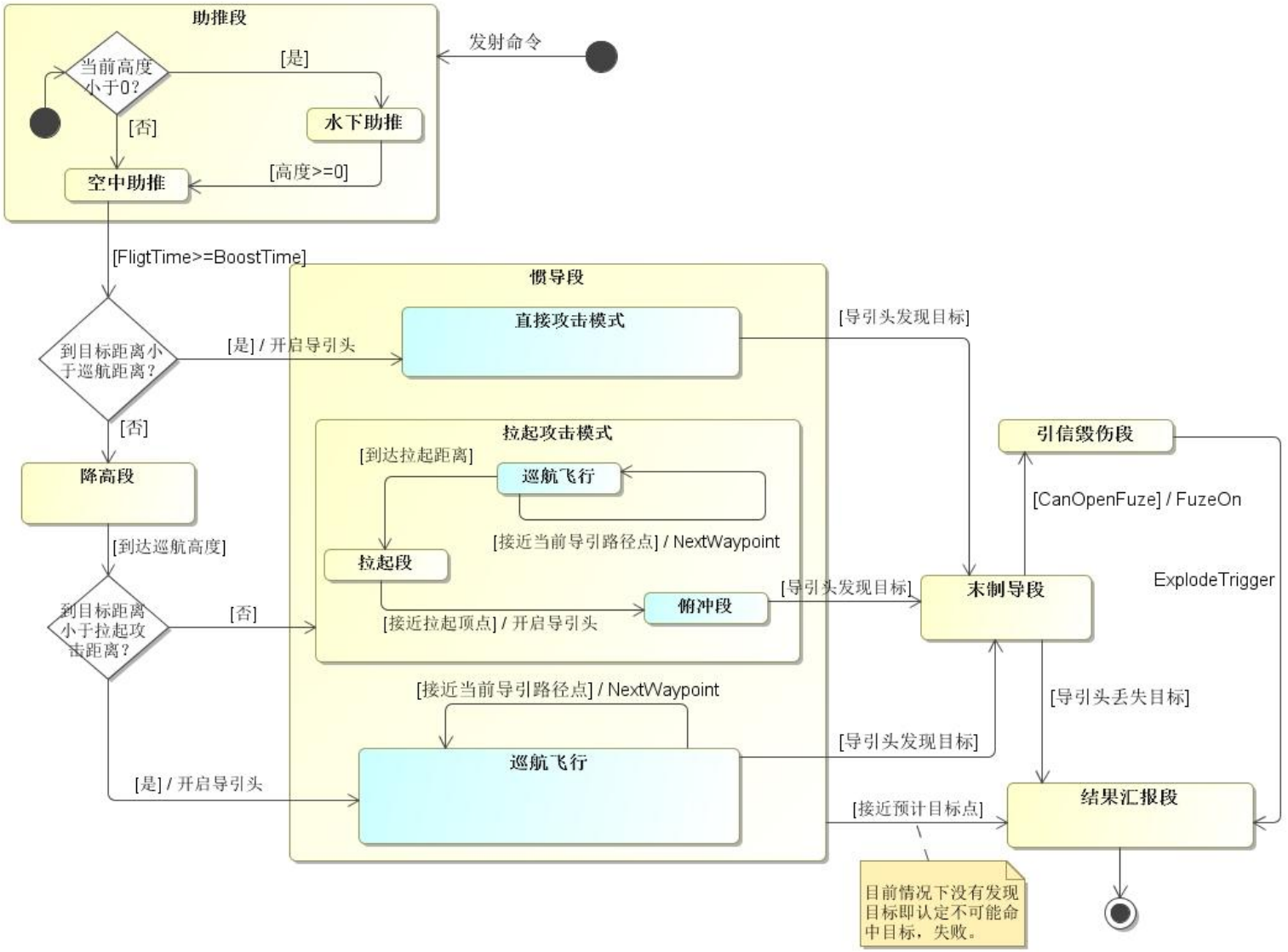
Surface to air missile physical behavior



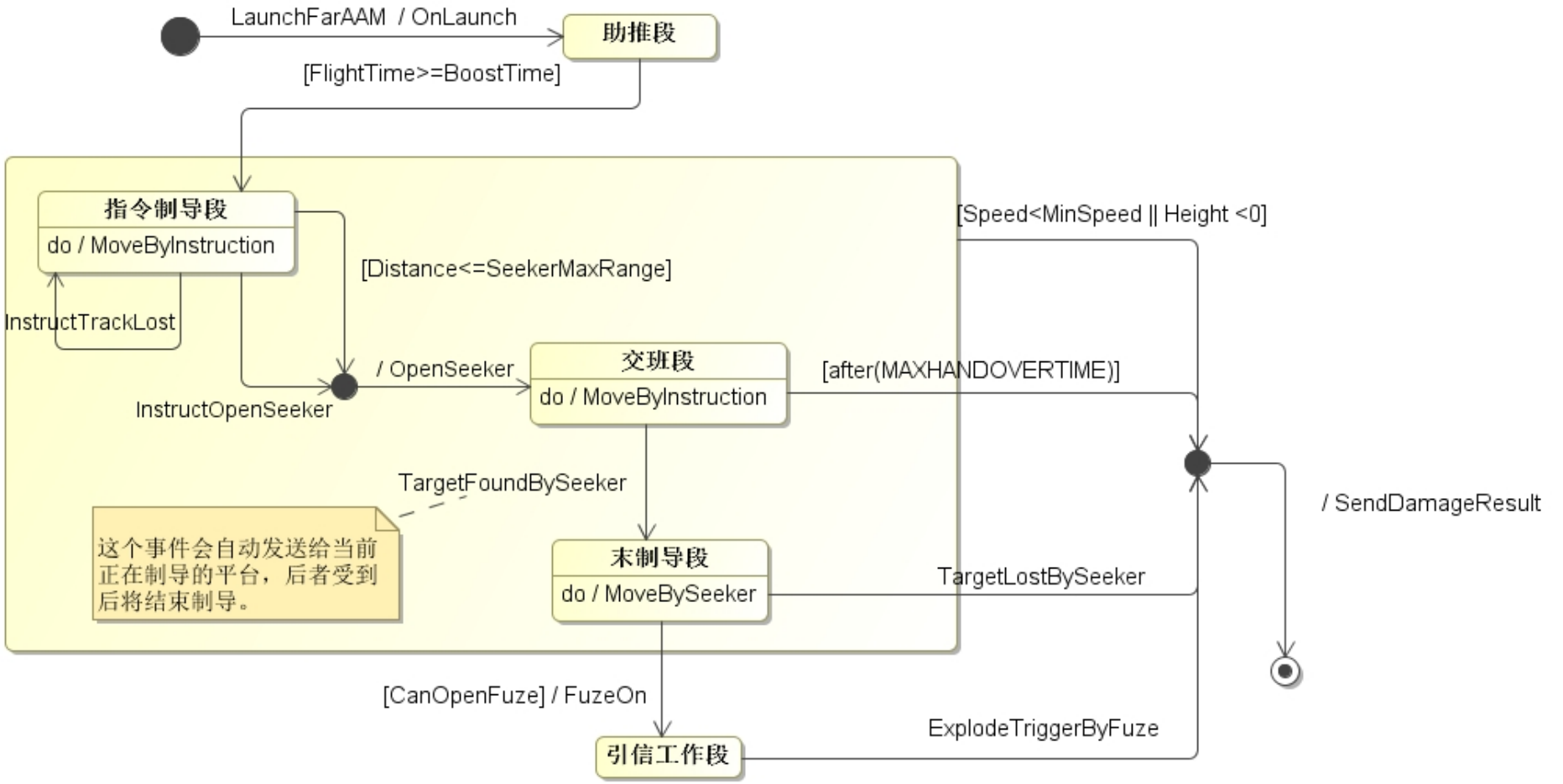
Air to surface missile physical behavior



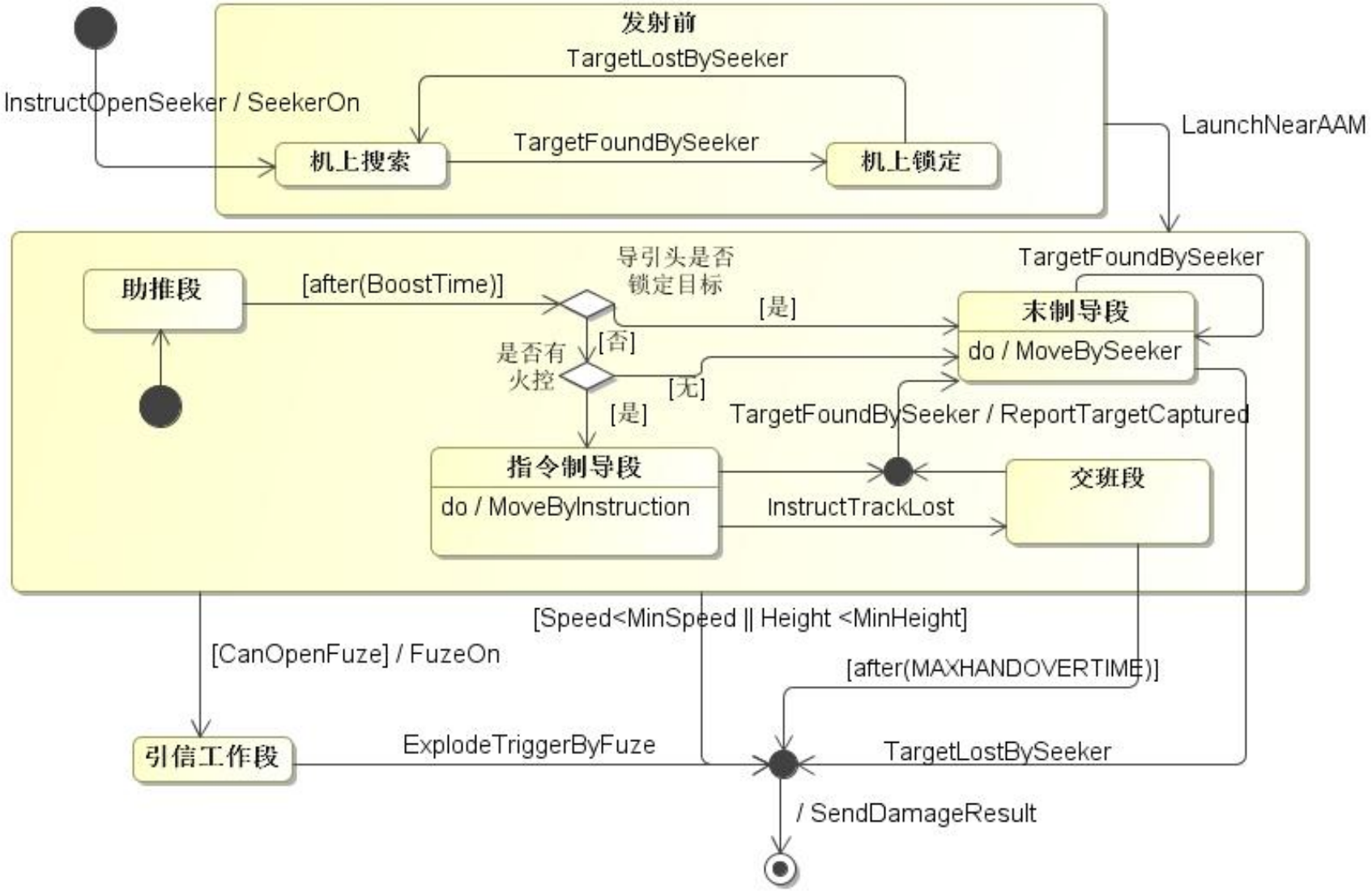
Cruise missile physical behavior



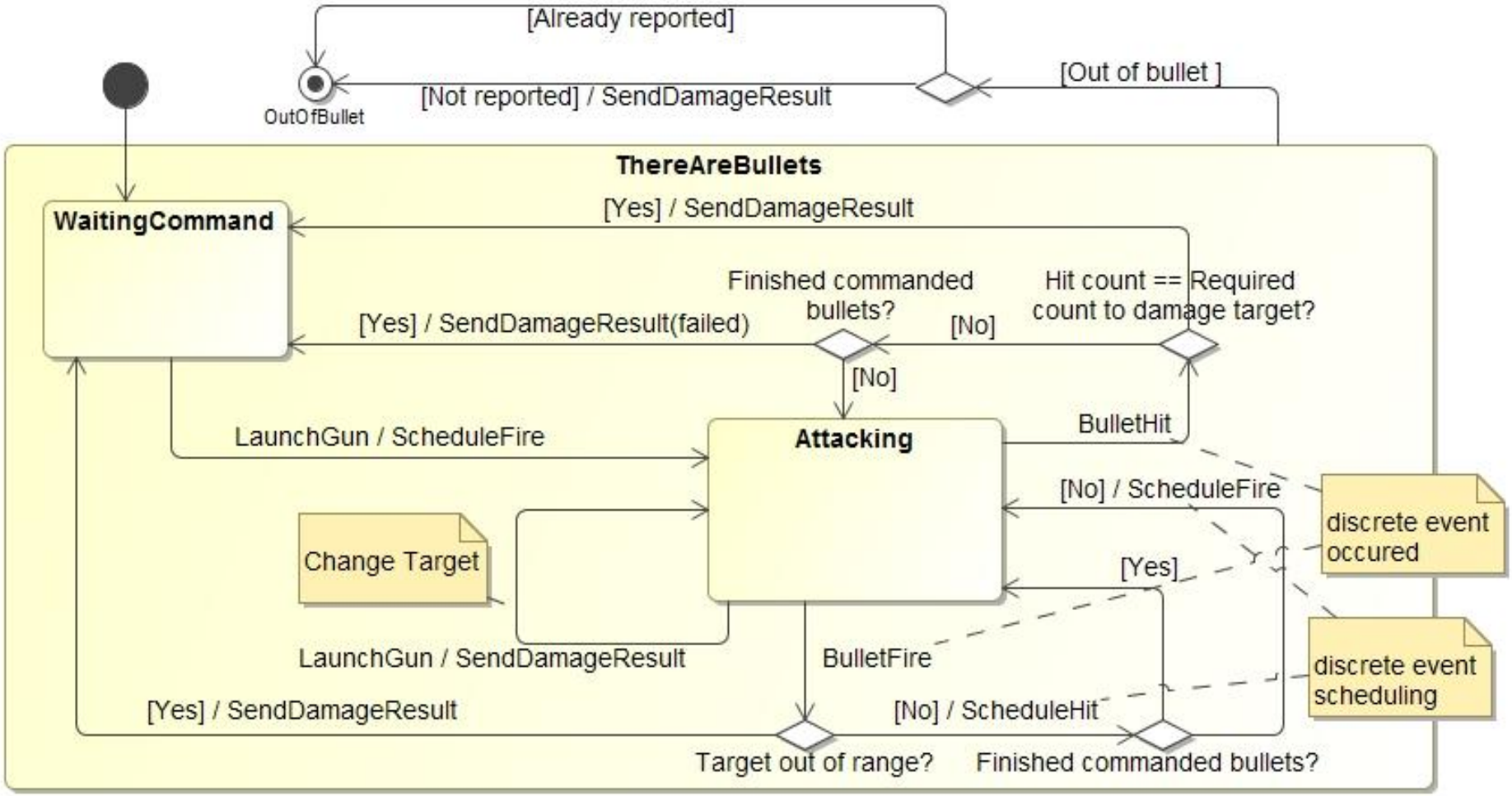
Far air to air missile physical behavior



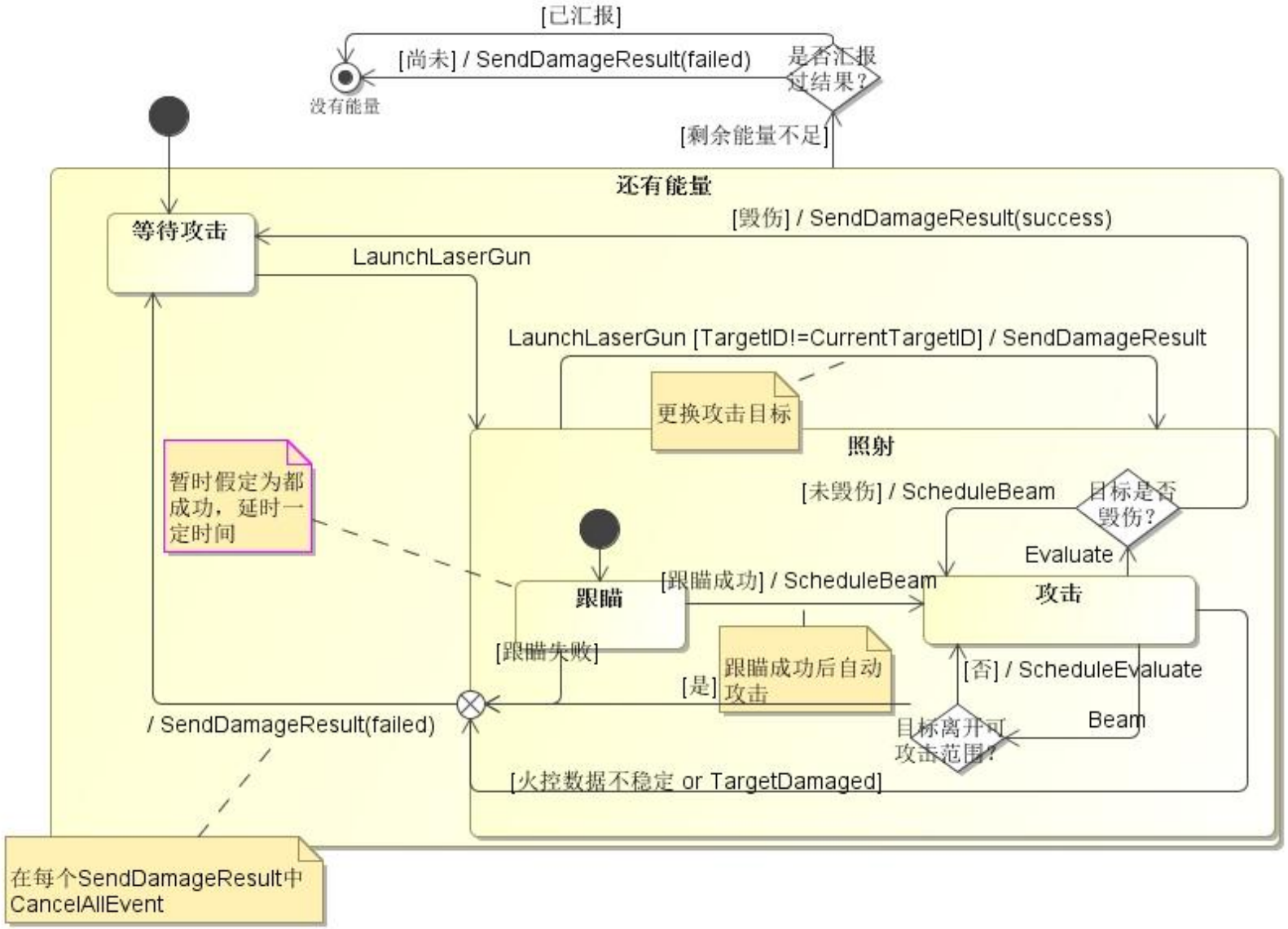
Near air to air missile physical behavior



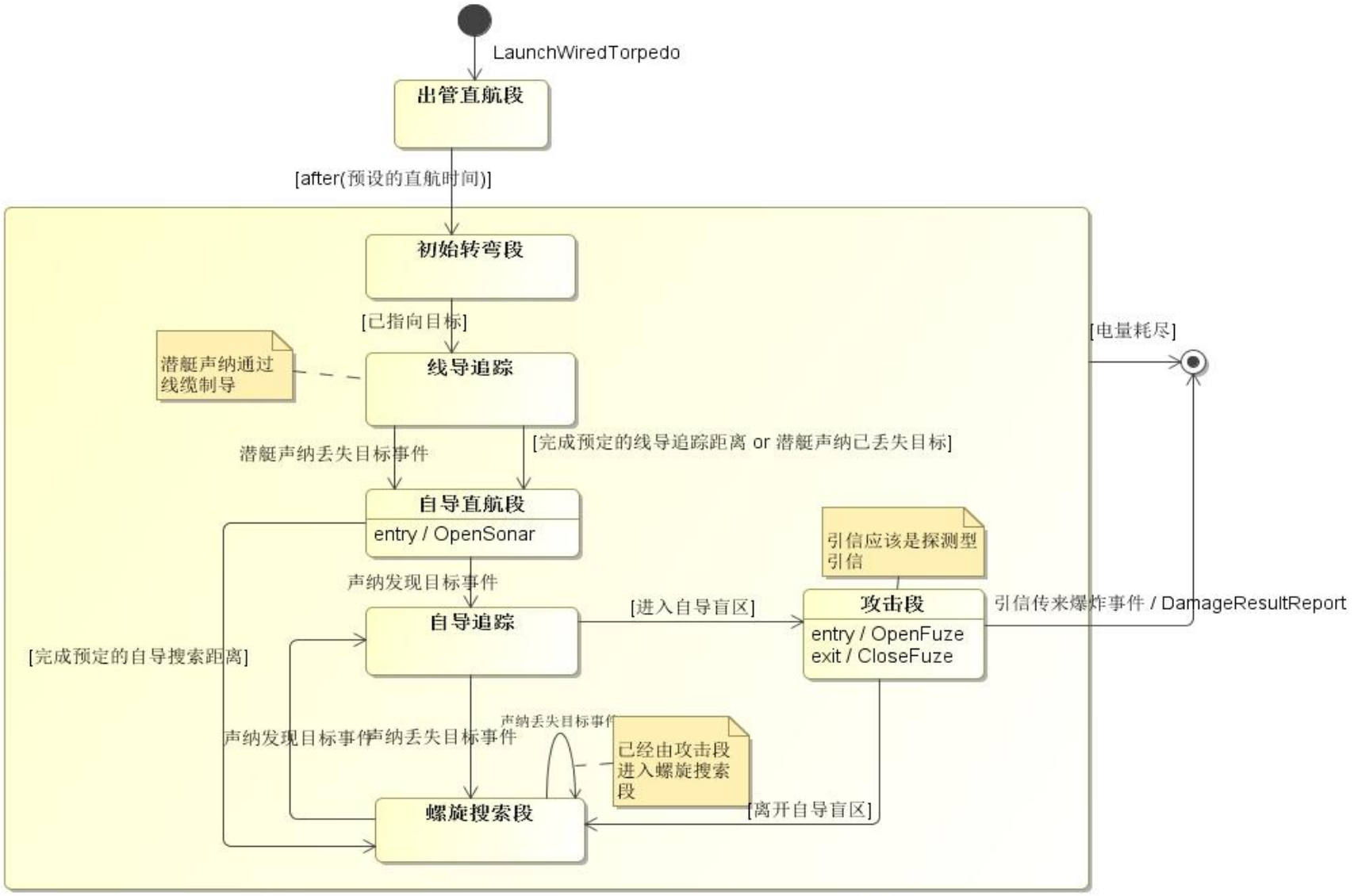
Ballistic gun physical behavior



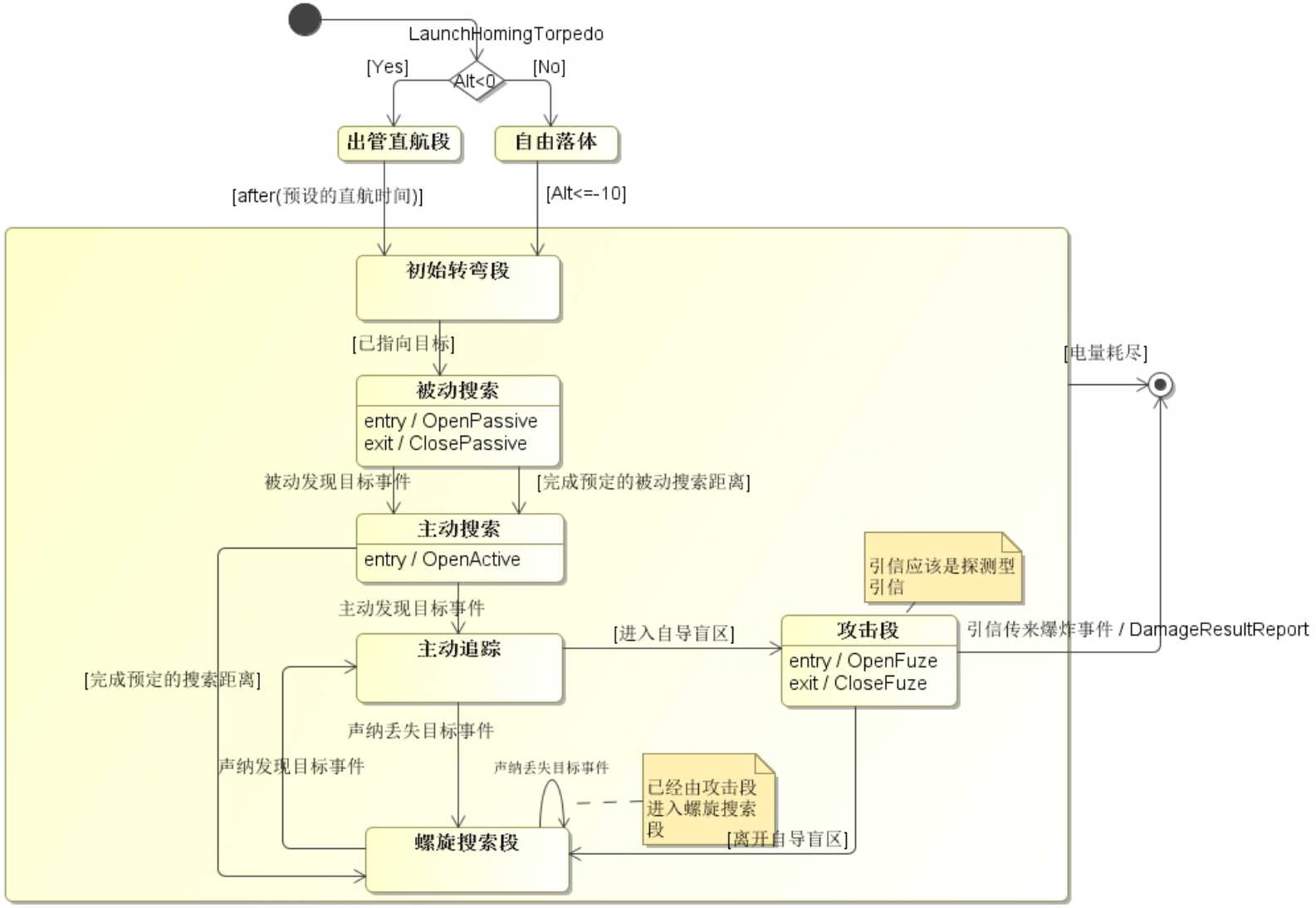
Laser gun physical behavior



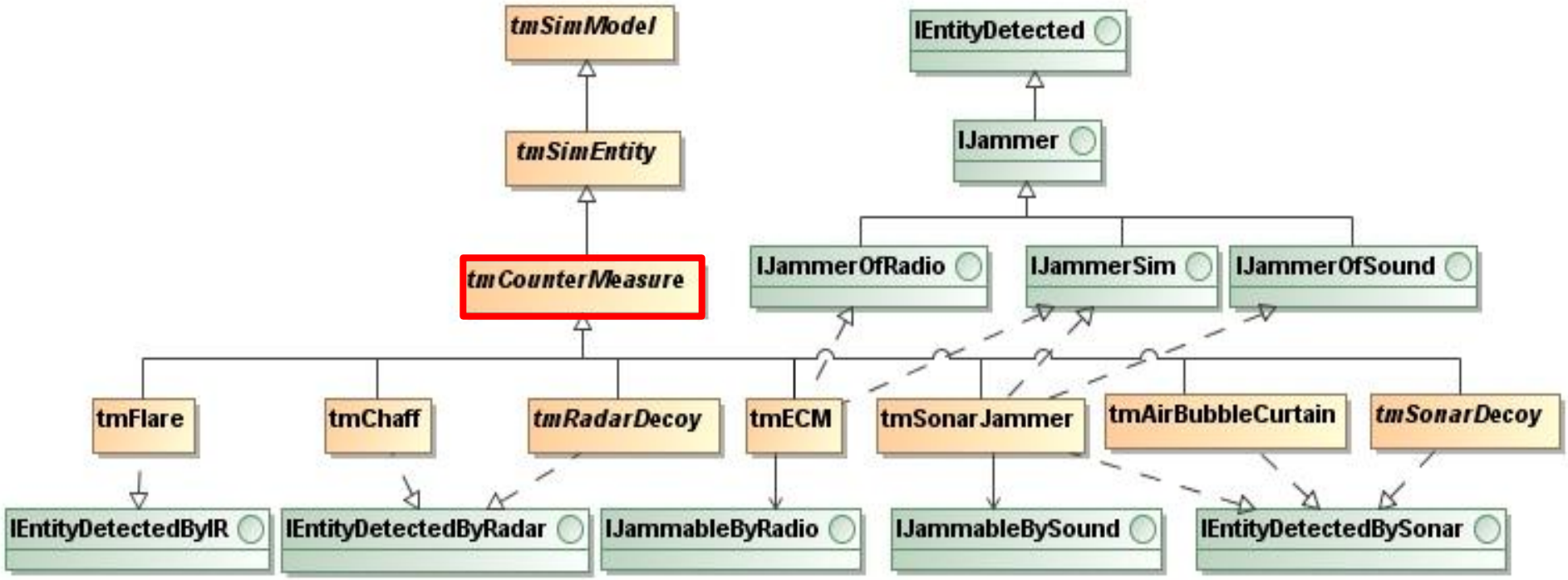
Wired torpedo physical behavior



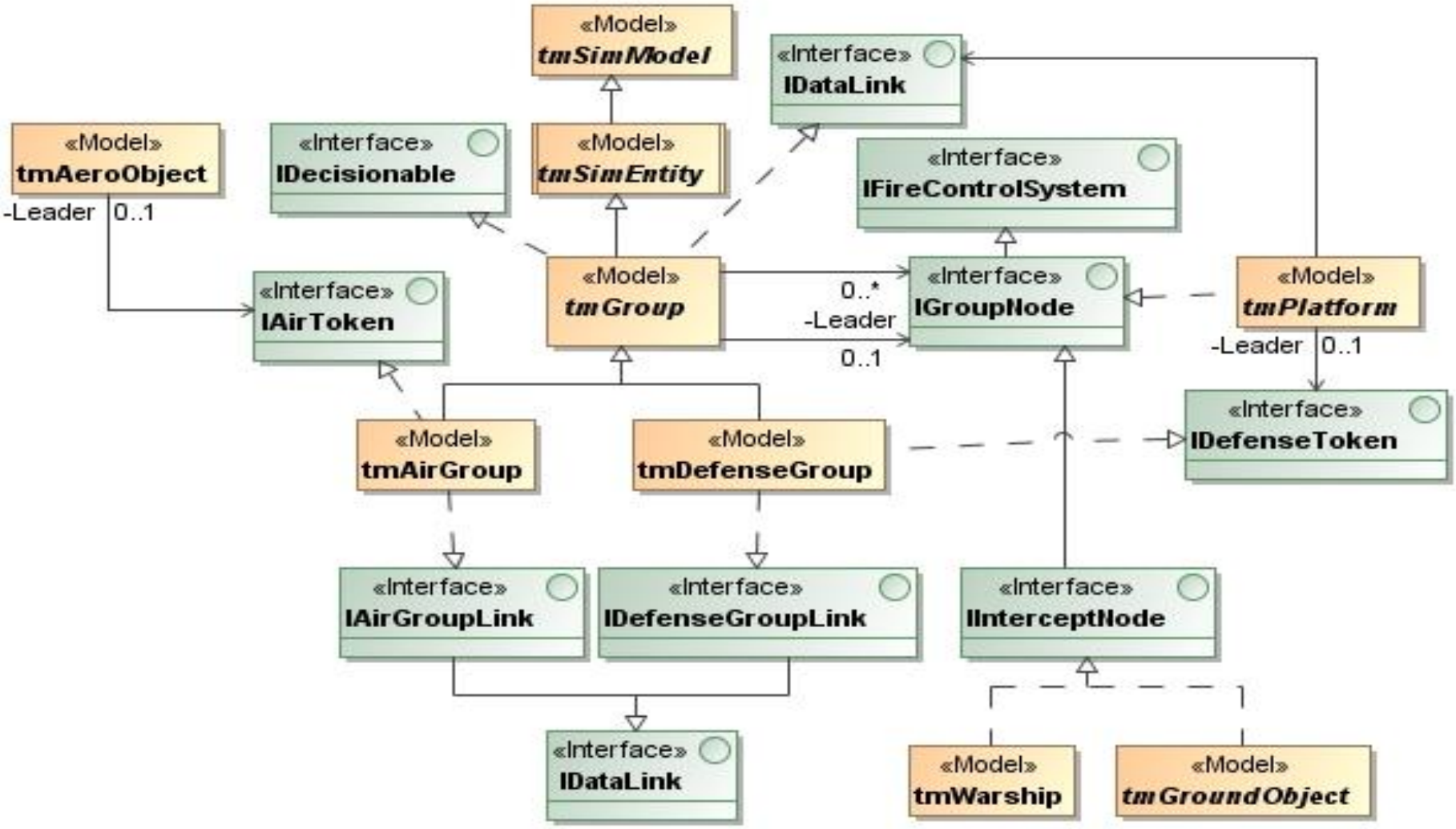
Homing torpedo physical behavior



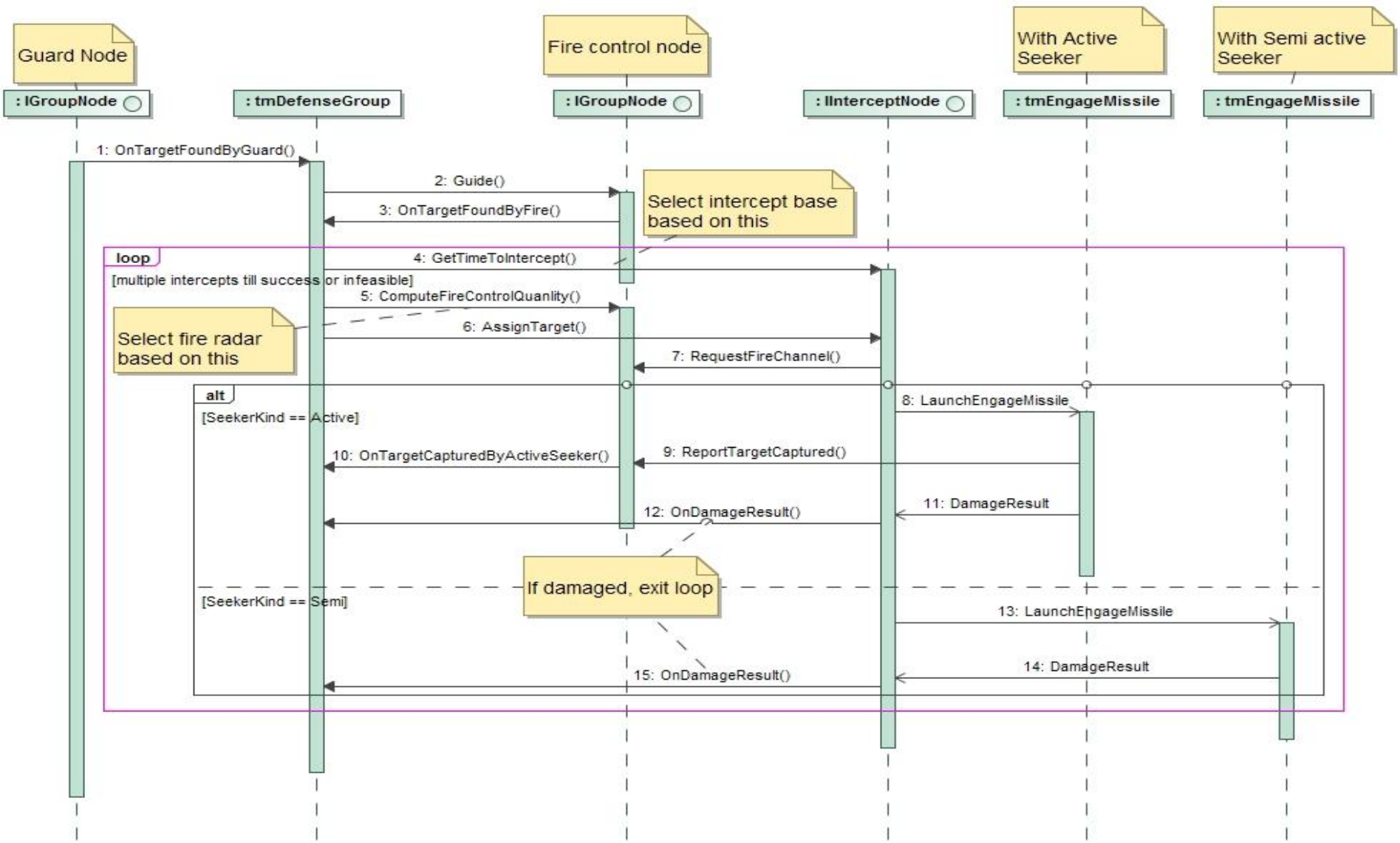
Countermeasure model architecture



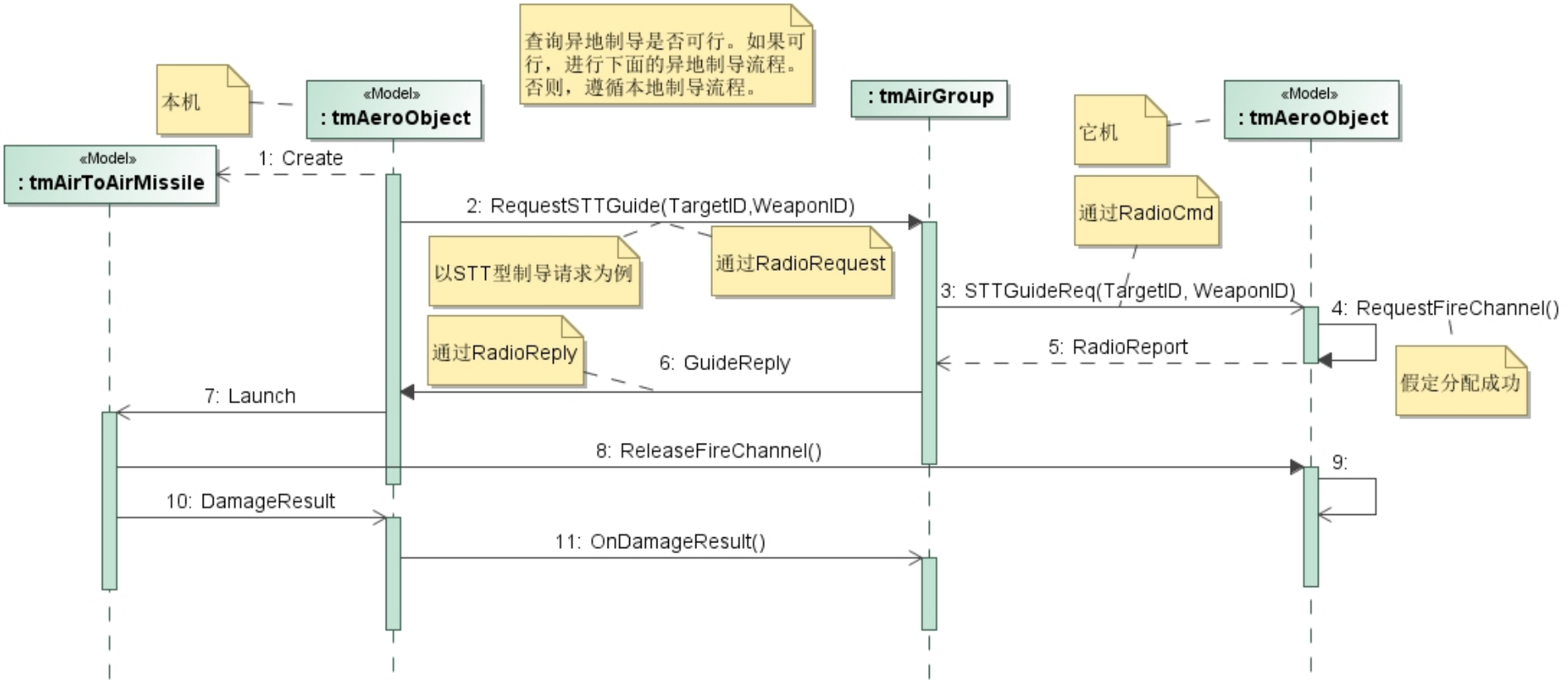
Group model architecture



Sequential diagram-based defense group collaborative behavior



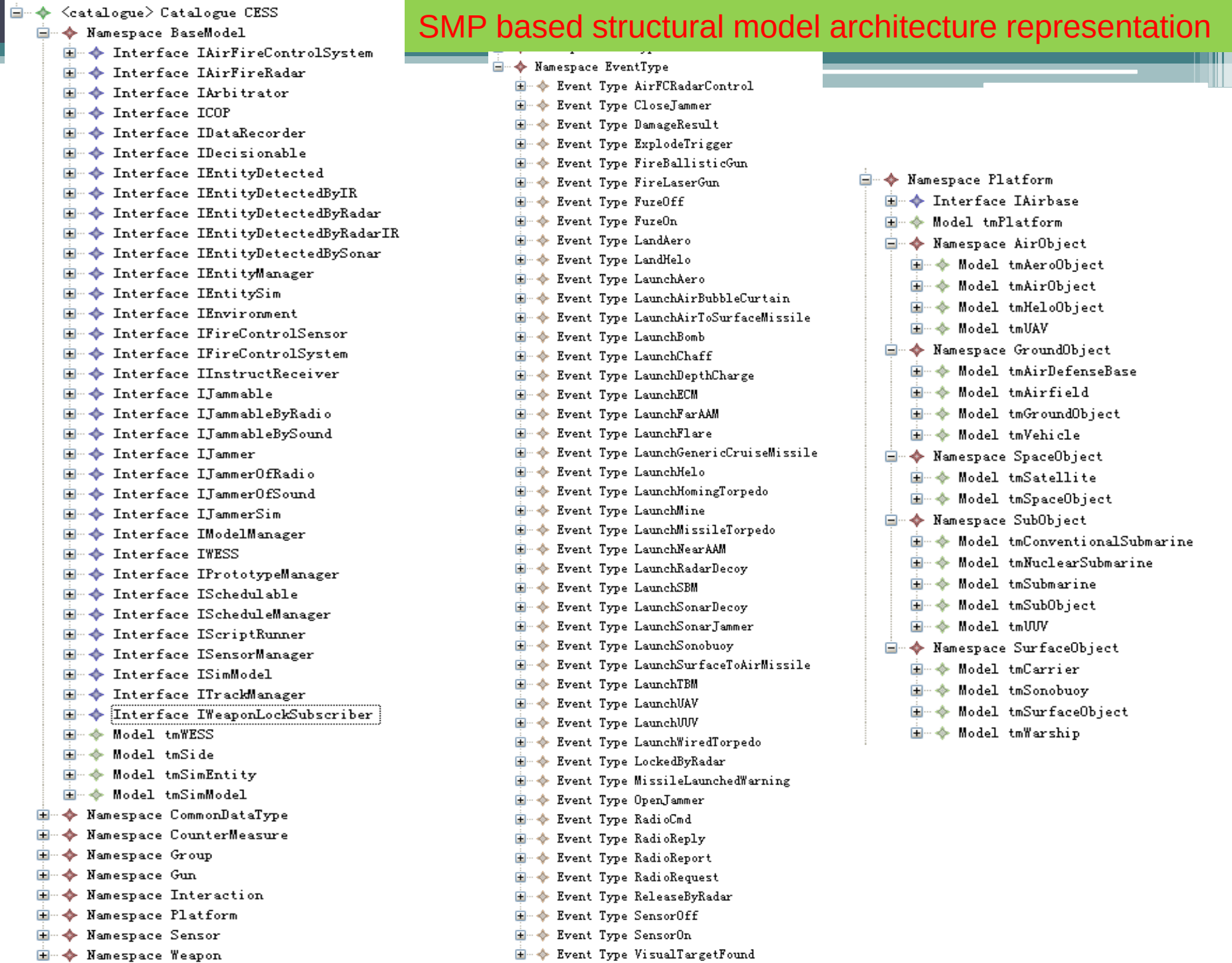
Air group collaborative guiding behavior



SMP based structural model architecture representation

- Represented using simulation model definition language (SMDL) of SMP standard
- Generated from the UML representation partly shown in previous slides by way of the UML profile for SMP.

SMP based structural model architecture representation



SMP based structural model architecture representation

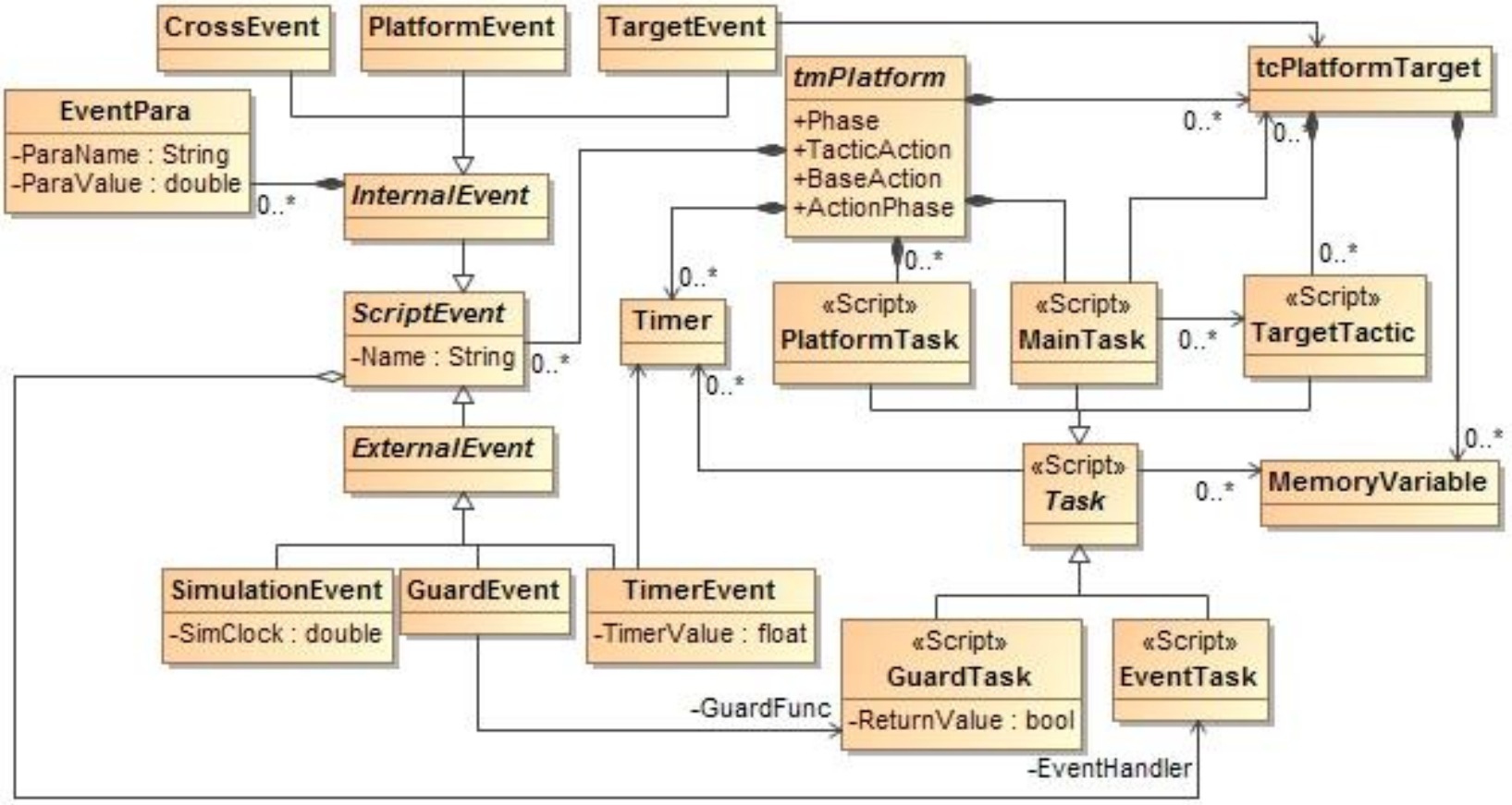
- Namespace Sensor
  - Model tmSensor
  - Namespace Fuze
    - Model tmContactFuze
    - Model tmFuze
    - Model tmSenseFuze
    - Model tmTimeFuze
  - Namespace Optical
    - Model tmInfraRed
    - Model tmIRST
    - Model tmLaser
    - Model tmLaserIlluminator
    - Model tmOptical
    - Model tmSeekerIR
    - Model tmSemiActiveLaserSeeker
    - Model tmVisual
    - Model tmWarningIR
  - Namespace Radar
    - Model tmRadar
    - Namespace ActiveRadar
      - Model tmActiveRadar
      - Model tmActiveRadarSeeker
      - Model tmAirFireRadar
      - Model tmFireControlRadar
      - Model tmSurfaceFireRadar
      - Model tmSurveillanceRadar
    - Namespace PassiveRadar
      - Model tmESM
      - Model tmPassiveRadar
      - Model tmPassiveRadarSeeker
      - Model tmSemiActiveRadarSeeker
  - Namespace Sonar
    - Model tmActiveSonar
    - Model tmActiveSonarSeeker
    - Model tmActiveTrackSonar
    - Model tmPassiveSonar
    - Model tmPassiveSonarSeeker
    - Model tmPassiveTrackSonar
    - Model tmSonar
    - Model tmSSM

- Namespace Weapon
  - Interface IWeaponControlSystem
  - Model tmBomb
  - Model tmDepthCharge
  - Model tmMine
  - Model tmWeapon
  - Model tmWeaponWithSeeker
  - Namespace Missile
    - Model tmMissile
    - Namespace BallisticMissile
      - Model tmBallisticMissile
      - Model tmSBM
      - Model tmTBM
    - Namespace CruiseMissile
      - Model tmCruiseMissile
      - Model tmGenericCruiseMissile
    - Namespace EngageMissile
      - Model tmAirToAirMissile
      - Model tmAirToSurfaceMissile
      - Model tmEngageMissile
      - Model tmFarAAM
      - Model tmNearAAM
      - Model tmSurfaceToAirMissile
  - Namespace Torpedo
    - Model tmHomingTorpedo
    - Model tmMissileTorpedo
    - Model tmTorpedo
    - Model tmWireGuidedTorpedo

- Namespace CounterMeasure
  - Model tmAirBubbleCurtain
  - Model tmChaff
  - Model tmCounterMeasure
  - Model tmECM
  - Model tmFlare
  - Model tmRadarDecoy
  - Model tmSonarDecoy
  - Model tmSonarJammer
- Namespace Group
  - Interface IAirGroupLink
  - Interface IAirToken
  - Interface IDataLink
  - Interface IDefenseGroupLink
  - Interface IDefenseToken
  - Interface IGroupNode
  - Interface IInterceptNode
  - Model tmAirGroup
  - Model tmDefenseGroup
  - Model tmGroup
- Namespace Gun
  - Model tmBallisticGun
  - Model tmGun
  - Model tmLaserGun

Cognitive modeling interface

- Base cognitive behavior metamodel (BCBM)

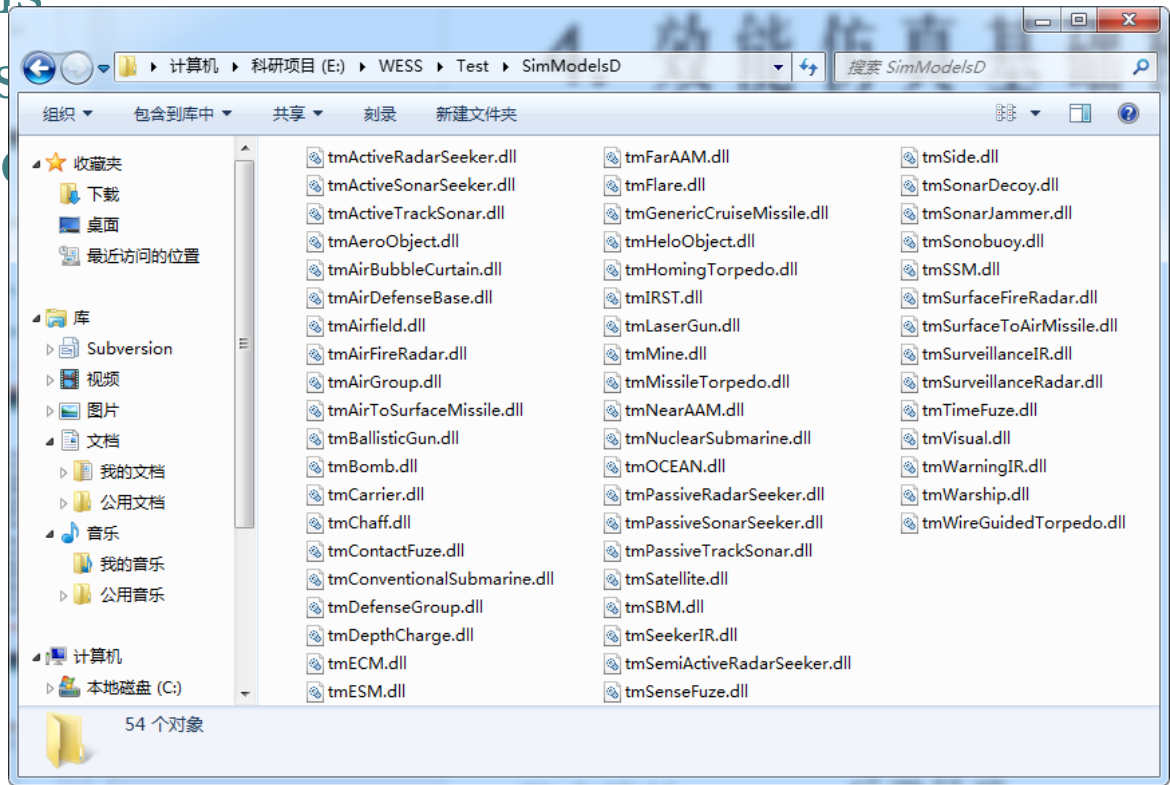


165 API functions for cognitive behavior modeling

- Parameters query(23)
- Mission & task query(12)
- Situation analysis (16)
- Platform maneuver(8)
- Aircraft close combat maneuver(13)
- Waypoint management(13)
- Fire control(26)
- Sensor control(16)
- Group control(10)
- Simulation control(5)
- State based modeling(8)
- Task based modeling(4)
- Event based modeling(11)

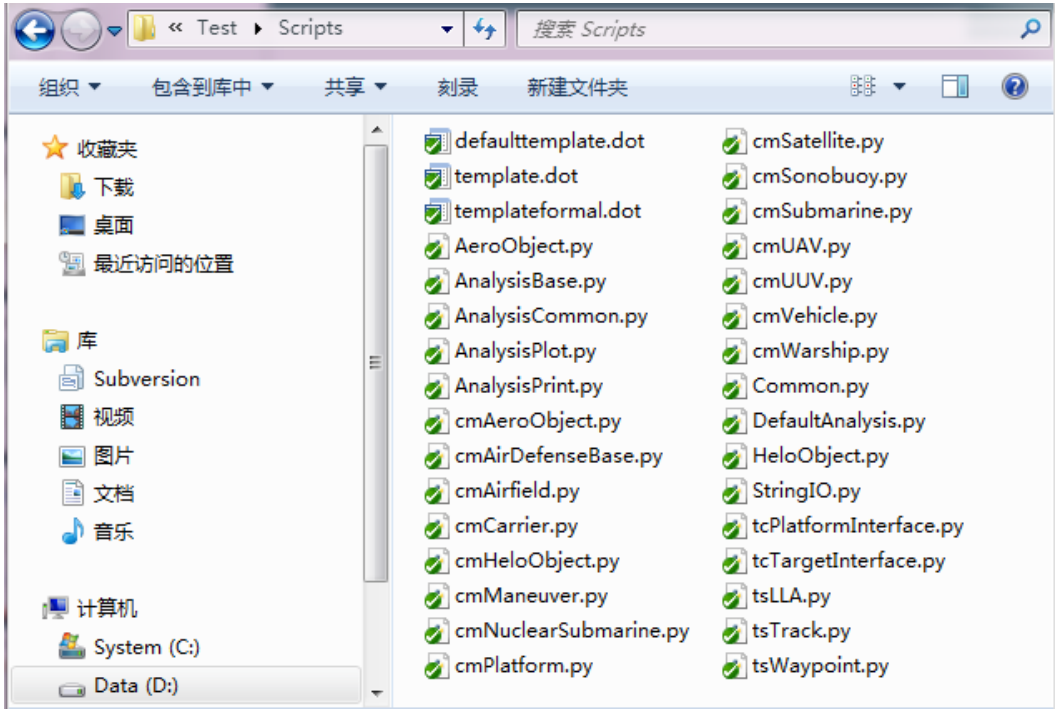
4. a model component library

- One DLL for each concrete model within AMA
- Total 54
  - 10 platform models
  - 14 weapon models
  - 20 sensor models
  - 7 countermeasure
  - 3 group models.



5. a set of cognitive decision model scripts

- One default script for each combat platform model
- Model-instance separation of cognitive behavior scrip
- Each script corresponds to a graphical conceptual mo



Model-instance separation of cognitive behavior scripts

- Instance specific scripts inherit from the default

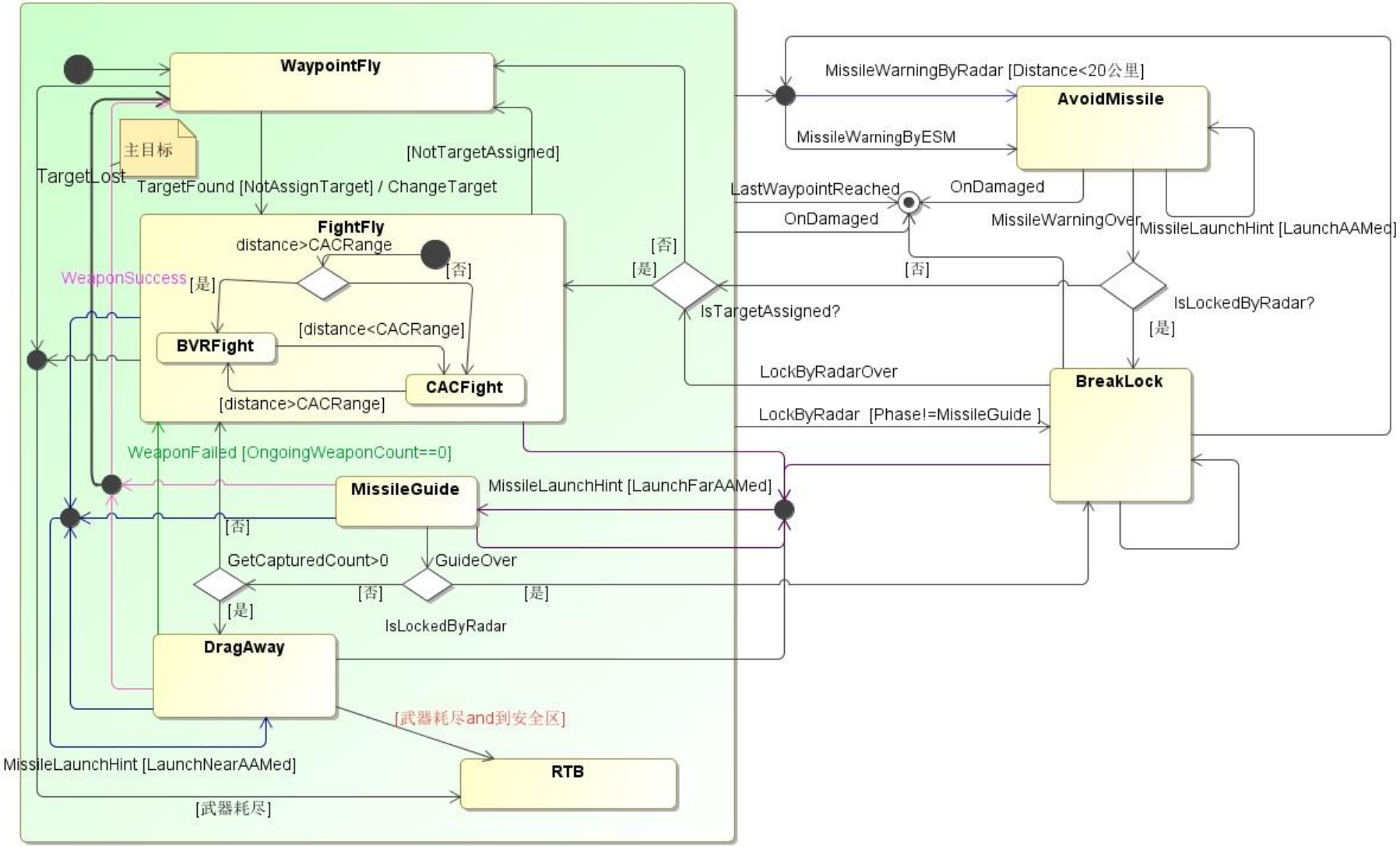
cmAeroObject.py | cmPlatform.py | **cmHeloObject.py** | Helo1new.py

```
# -*- coding: cp936 -*-
from cmPlatform import *

C_DippingSonarWorkingDepth = 30#the actual workign depth of dipping sc

#statemachine
P_Flying                = 1 #fly to next waypoint
P_Hovering_Releasing    = 2 #hover and release dipping sonar
P_Hovering_Waiting      = 3 #hover and wait target found event
P_Hovering_Withdrawing  = 4 #hover and withdraw dipping sonar

class cmHeloObject(cmPlatform):
    #entry point for initialization
    def InitDecision(self,PI):
        PI.SubscribeEvent("WaypointReached","WaypointReachedHandler")
        PI.SubscribeEvent("TargetFound","TargetFoundHandler")
        PI.AircraftTakeOff()
        PI.Phase = P_Flying

    #entry point for decision
    def StepDecision(self,PI):
```

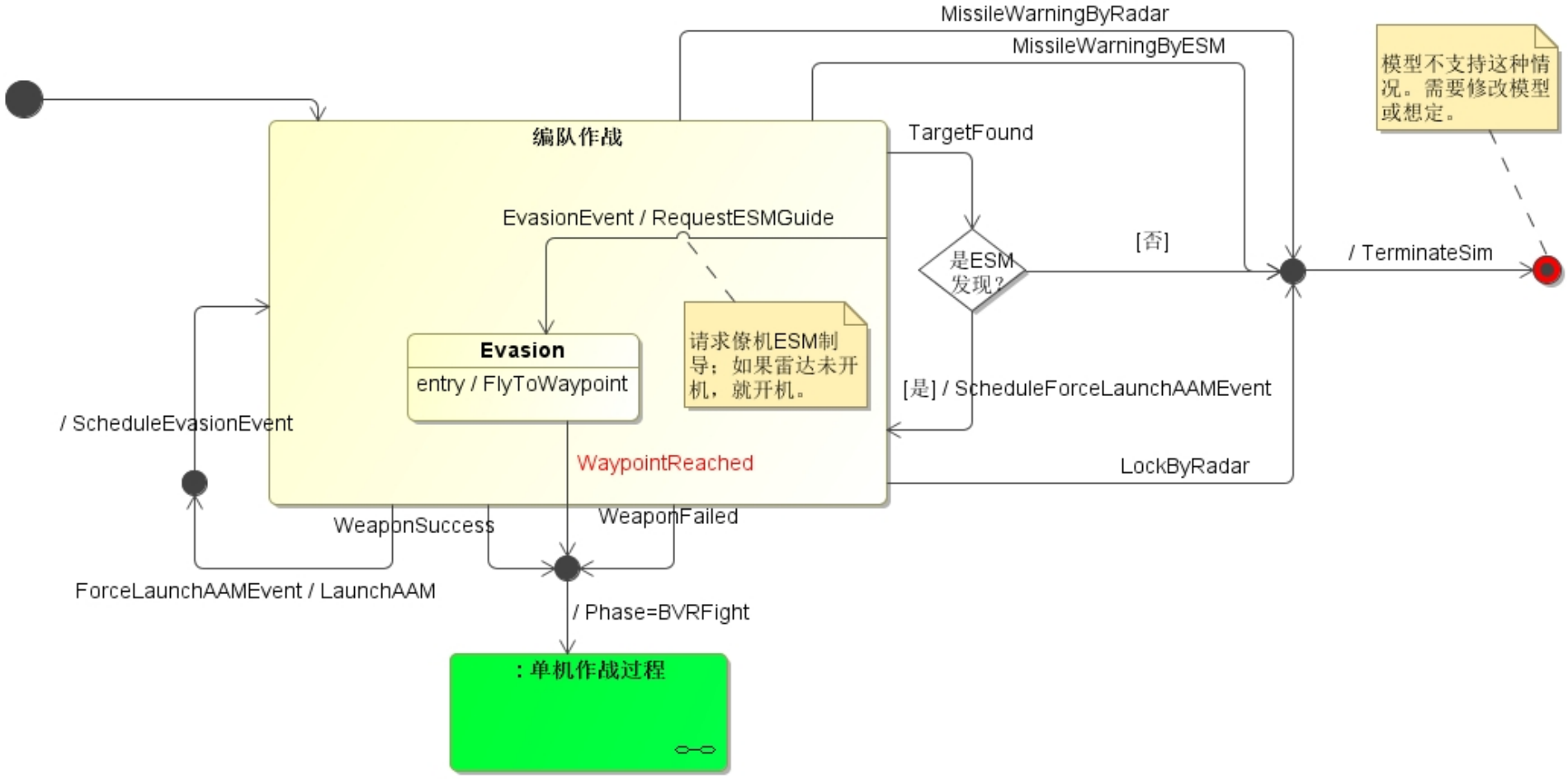
cmAeroObject.py | cmPlatform.py | cmHeloObject.py | **Helo1new.py**

```
# -*- coding: cp936 -*-
from cmHeloObject import *

class Helo1new(cmHeloObject):
    def InitDecision(self,PI):
        PI.SubscribeEvent("WaypointReached","WaypointReachedHandler")
        PI.SubscribeEvent("TargetFound","TargetFoundHandler")
        PI.Phase = P_Flying

def InitDecision(PlatformInfo):
    Helo1new(PlatformInfo.Name).InitDecision(PlatformInfo)
```

Each script corresponds to a graphical conceptual model - default aero object



conceptual model of an instance aero object by reuse of the default



欢迎使用WESS效能仿真系统！

Dr. Yonglin Lei(雷永林)
0731-84574538
13874992600
yllei@nudt.edu.cn