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

### Application Composition
- Net centric Missile Defense
- Air Combat
- Surface Combat
- Submarine Combat
- Space Combat
- Helicopter Anti-Sub
- Air Ground Combat
- Laser Weapon Experiment
- ...

### Application Tools
- Prototype Data Editor
- Scenario Editor
- Cognitive Behavior Editor with CSCBML Extensions
- DOE Editor
- 2D Viewer
- 3D Viewer
- Output Analyzer

### Model Architecture
- Abstract Physical Behavior Models of DMA
- Cognitive Behavior Model Templates (.py)
- Concrete Model Components of AMA (.dll)
- Application-Specific Cognitive Behavior Models (.py)

### Modeling Services
- Environment manager
- Entity manager
- Sensor manager
- Prototype manager
- Arbitrator
- Script runner
- Scheduler manager
- Event manager
- Task manager
- Display service

### Development & Runtime
- System analysis
- Model design
- Model verification
- Code generation
- Model development
- Model assembly
- Decision Scripts Editor
- Simulation Engine
- Experiment Management
- Data recorder
- Playback tool

### Data & Resources
- Prototype Database
- DOE files
- Scenario files
- Simulation Configuration Files
- OpenFlight files
- Icon files
- Simulation Database
- Trace files
- Analysis Reports
- Playback files
- ...

### Infrastructure & Platform
- Microsoft Windows
- Microsoft ACCESS
- Microsoft Visual C++
- Java/EMF
- Boost.Python
- eSimlink/RTI Network
- Altova XML
- MapX
- Open Scene Graph
WESS application flowcharts

Workflow of Model Architecture Maintaining

1. System conceptual analysis (using MagicDraw)
2. Model architecture design (extending or modifying)
3. C++ code generation
4. Simulation model development (using VC++)
5. Model assembly and test

Workflow of Composition and Application

1. Prepare combat systems prototype data
2. Synthesize scenario (model composition)
3. Design Experiment
4. Specify cognitive behaviors of combat platform models (combat platform specific)
5. Specify cognitive behaviors of combat platform instances (customizing that of models)
6. Run Simulation Experiments
7. Output Analysis
8. Simulation Display
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.
namespace Platform {

  class tmPlatform {
  public:
    tmPlatform();
    tmPlatform(const Sm::String8& name, const Sm::String8& description);
    virtual ~tmPlatform();

  public:
    virtual void SetCruiseSpeed(const Sm::Float32& newValue) {
      CruiseSpeed = newValue;
    }
    virtual const Sm::Float32& GetCruiseSpeed() const {
      return CruiseSpeed;
    }
    virtual void SetMaxSpeed(const Sm::Float32& newValue) {
      MaxSpeed = newValue;
    }
    virtual const Sm::Float32& GetMaxSpeed() const {
      return MaxSpeed;
    }

  private:
    // 造山速度 (m/s)
    Sm::Float32 CruiseSpeed;
    Sm::Float32 MaxSpeed;

    // 造山器名 (从指定中解析。动态创建的子平台的同父平台的。)
    CommonDataType::NameString MissionName;
  }

  virtual ~tmPlatform();
}

// ---------------------------- Constructor/Destructor ----------------------------

public:
  tmPlatform();
  tmPlatform(const Sm::String8& name, const Sm::String8& description);
  virtual ~tmPlatform();

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

private:
  // 造山速度 (m/s)
  Sm::Float32 CruiseSpeed;
  Sm::Float32 MaxSpeed;

  // 造山器名 (从指定中解析。动态创建的子平台的同父平台的。)
  CommonDataType::NameString MissionName;

public:
  CommonDataType::NameString GetMissionName() const;
  void SetMissionName(const CommonDataType::NameString& newValue);
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-Carlo sampling
  ▫ SimDisplay: simulation display in 2D and 3D
  ▫ SimLogger: log and playback simulation data
  ▫ OutputAnalyzer: script-based output analyzer
if (PI.FetchEventParaBool("MissileWarning","ByFireRadar")):
    MissileID = PI.FetchEventParaInt("MissileWarning","MissileID")
    mslTrack = PI.GetTrack(MissileID)
    if (PI.DistanceTo(mslTrack)<=C_RVMRange):
        PI.Debug("雷达警告导弹,距离<%dKM,+%s>规避导弹\%(C_RVMRange/1000,Phase2String(PI.Phase))")
        PI.AutoMissileAvoid()
        PI.Phase = P_AvoidMissile
    else:
        PI.ScheduleGuardEvent("RadarWarningGuard","RadarWarningGuardHandler","RadarWarning")
        PI.Debug("雷达探测导弹,距离>10KM,继续+%s\%(C_RVMRange/1000,Phase2String(PI.Phase))")
        PI.Phase = P_AvoidMissile
    else:
        PI.Debug("导弹警告 +%s\%(Phase2String(PI.Phase))")
        PI.AutoMissileAvoid()
        PI.Phase = P_AvoidMissile

#雷达探测导弹，采取规避机动（根据距离）
def RadarWarningGuardHandler(PlatformInfo):
    PlatformInfo.Debug("雷达警告导弹,距离<%dKM,+%s>规避导弹\%(C_RVMRange/1000,Phase2String(PI.Phase))")
    PlatformInfo.AutoMissileAvoid()
    PlatformInfo.Phase = P_AvoidMissile
    PlatformInfo.RemoveExternalEvent("RadarWarningGuard")
    return

#雷达探测导弹，计算距离
def RadarWarningGuardFunc(PlatformInfo):
    missile = PlatformInfo.GetClosestWarningMissile()
    distance = PlatformInfo.DistanceTo(missile)
    #PlatformInfo.Debug("雷达警告导弹,距离+%s\%(distance,Phase2String(PlatformInfo.Phase))")
    return distance < C_RVMRange

#导弹警示解除
def MissileWarningOverHandler(PlatformInfo):
    #PI = tcPlatformInterface.tcPlatformInterface
    PI = PlatformInfo
    if(PI.GetItemCount("ECM")>0):
        #PI.Debug("关闭ECM...")
        PI.CloseECM()
        if(PI.isLockByRadar()):
            PI.Debug("雷达警告解除 +%s\%(Phase2String(PI.Phase))")
        else:
            PI.Debug("雷达警告解除")
            return 159 36 0 174
DoeEditor
SimDisplay and SimLogger
OutputAnalyzer-analysis script
4.2.1 平台：航母类

<table>
<thead>
<tr>
<th>红方航母 (共 1 个)</th>
<th>蓝方航母 (共 1 个)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.3, 38.0, 0.0, 0.0</td>
<td>14.0, 58.0, 0.0, 0.0</td>
</tr>
</tbody>
</table>

4.2.2 平台：固定翼飞机类

<table>
<thead>
<tr>
<th>飞机 1 (共 2 个)</th>
<th>飞机 2 (共 2 个)</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.2492780785, 44.1702434635, 9000.0, 257.2224</td>
<td>18.5187684255, 45.7366948765, 9000.0, 257.2224</td>
</tr>
<tr>
<td>19.28, 45.76, 9000.0, 257.2224</td>
<td>19.1449195576, 44.5678790653, 9000.0, 257.2224</td>
</tr>
</tbody>
</table>

4.3 平台参数设置

4.3.1 平台配置：航母类

4.3.2 平台配置：固定翼飞机类

参数名称 | 飞机 1 | 飞机 2 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>长度</td>
<td>18.92</td>
<td>16.82</td>
</tr>
<tr>
<td>宽度</td>
<td>2500000000</td>
<td>2500000000</td>
</tr>
<tr>
<td>高度</td>
<td>5.08</td>
<td>6.06</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Domain modeling</th>
<th>DIK representation</th>
<th>Application modeling</th>
<th>AVK representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical behavior modeling</td>
<td>Structural architecture modeling</td>
<td>Cognitive behavior modeling</td>
<td></td>
</tr>
<tr>
<td>behavior modeling formalisms (SC, DE, AD, SD, …)</td>
<td>domain-specific extension</td>
<td>metamodeling framework (EMF, GME, …)</td>
<td>behavior modeling formalisms (SC, PN, …)</td>
</tr>
<tr>
<td>DMA physical behavior (physical behavior model of each abstract model in DMA)</td>
<td>DMA structure (domain abstract models and their relationships)</td>
<td>base cognitive behavior metamodel (BCBM)</td>
<td>combat-platform specific cognitive behavior metamodel (CSCBM)</td>
</tr>
<tr>
<td>AMA physical behavior (physical behavior model of each concrete model in AMA)</td>
<td>AMA structure (concrete models and their relationships)</td>
<td>AMA cognitive behavior (cognitive behavior models of each combat platform model in AMA)</td>
<td></td>
</tr>
<tr>
<td>C++ based representation (physical behavior)</td>
<td>executable simulation model (.dll)</td>
<td>C++ based representation (structure)</td>
<td>Dynamic composition</td>
</tr>
</tbody>
</table>

Legend: InstanceOf, InheritFrom, ContainedBy, GenerateFrom, MetamodelOf
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

```
Event based time advancement

Basic event
- All tmSimEntity instances moving
  - Updating all target state by extrapolation
    - All tmPlatform instances decision monitoring

Decision event
- All tmSimEntity instances moving
  - Relevant tmSensor instances detecting
- All tmSimEntity instances moving
  - Updating all target state by extrapolation
- All tmSimEntity instances moving
  - Relevant tmPlatform instances decision making
```
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

BackToWaypointMode

WaypointCmd

SuperManeuverCmd

FollowCmd

ManualManeuverCmd

AutoManeuverCmd

ControlThrottle

Platform model architecture
Sensor model architecture
Template Method pattern based detection model

```cpp
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();
}
```

// Process target detection and removal
for (unsigned i=0; i<CEL.size(); i++){
    BaseModel::IEntityDetected* pTarget = CEL[i];
    if (DetectTarget(pTarget)){ // Detect target
        // Not in target list
        if (!AlreadyFound(pTarget->get_ID())){
            OnTargetFound(pTarget);
        }
        else{ // In target list
            OnTargetDetected(pTarget);
        }
    }
    else{ // Not detected
        // Already found target
        if (AlreadyFound(pTarget->get_ID())){
            OnTargetNonDetected(pTarget, "信号功率太弱");
        }
    }
}
Abstract Statecharts of sensor models
Track report

Sensor model architecture
Air fire radar statecharts model

Sensor model architecture
Weapon model architecture
Damage evaluation and report
Statecharts-based guided weapon modeling framework
Surface to air missile physical behavior

**Weapon model architecture**

- **Boosting**
  - do / MoveByBoostThrust
  - [FlightTime] >= [BoostTime]
  - SeekerIsOpen? [No]
  - SeekerCapturedTarget? [Yes]

- **Guiding**
  - InstructTargetLost
  - InstructTrackUpdate

- **Instructing**
  - do / MoveByInstruction
  - InstructOpenSeeker / SeekerOn

- **Handovering**
  - do / MoveByInstruction
  - TargetFoundBySeeker / ReportTargetCaptured

- **Seeking**
  - do / MoveBySeeker
  - [CanOpenFuze] / FuzeOn

- **Fuzing**
  - do / MoveByInertia

- **End**
  - [ExceedRangeLimit]
  - [min(TimeToLastInstruct, TimeToOpenSeeker) > LostThreshold]
  - ExplodeTrigger
  - SendDamageResult
Air to surface missile physical behavior
Cruise missile physical behavior

Weapon model architecture
Far air to air missile physical behavior

Weapon model architecture

LaunchFarAAM / OnLaunch
[FlightTime >= BoostTime]

InstructTrackLost
InstructOpenSeeker /
/ OpenSeeker

[Distance <= SeekerMaxRange]

TargetFoundBySeeker
TargetLostBySeeker

[CanOpenFuze] / FuzeOn
ExplodeTriggerByFuze

/ SendDamageResult

[Speed < MinSpeed || Height < 0]

/ SendDamageResult

[after(MAXHANDOVERTIME)]

do / MoveByInstruction

do / MoveByInstruction

这个事件会自动发送给当前正在制导的平台，后者受到后将结束制导。
Near air to air missile physical behavior
Ballistic gun physical behavior

Weapon model architecture

[Diagram of ballistic gun physical behavior with states and decision paths]

- WaitingCommand
  - [Yes] / SendDamageResult
  - OutOfBullet
    - [Already reported]
    - [Not reported] / SendDamageResult
    - [Out of bullet]

- ThereAreBullets
  - [Yes] / SendDamageResult
  - Finished commanded bullets?
    - [Yes] / SendDamageResult(failed)
    - [No]
      - Hit count == Required count to damage target?
        - [No] / ScheduleFire
        - [Yes]
          - Change Target
          - LaunchGun / SendDamageResult
          - [Yes] / SendDamageResult
          - BulletFire
          - [No] / ScheduleHit
          - Target out of range?
          - Finished commanded bullets?

- Attacking
  - BulletHit
  - [No] / ScheduleFire
  - discrete event occurred
  - discrete event scheduling
Laser gun physical behavior

[Diagram of Laser gun physical behavior process]

- Wait for attack
- LaunchLaserGun
  - LaunchLaserGun [TargetID! = CurrentTargetID] / SendDamageResult
  - SendDamageResult(failed)

- Has energy
  - SendDamageResult(success)
  - SendDamageResult(failed)

- No energy
  - Report
  - SendDamageResult(failed)

- Evaluate
  - ScheduleBeam
  - Beam
  - ScheduleEvaluate

- Target damaged
  - ScheduleEvaluate
  - Beam
  - CancelAllEvent

- Target not damaged
  - ScheduleBeam
  - Beam
  - ScheduleEvaluate

In each SendDamageResult, cancel all events.
Wired torpedo physical behavior

LaunchWiredTorpedo

[after(预设的直航时间)]

初始转弯段

[已指向目标]

线导追踪

潜艇声纳通过线缆制导

潜艇声纳丢失目标事件

[完成预定的线导追踪距离 or 潜艇声纳已丢失目标]

自导直线段

entry / OpenSonar

声纳发现目标事件

[完成预定的自导搜索距离]

自导追踪

声纳发现目标事件声纳丢失目标事件

螺旋搜索段

[离开自导盲区]

已经由攻击段进入螺旋搜索段

攻击段

entry / OpenFuze exit / CloseFuze

引信应该是探测型引信

引信传来爆炸事件 / DamageResultReport

Weapon model architecture
Homing torpedo physical behavior

Weapon model architecture
Countermeasure model architecture
Group model architecture
Sequential diagram-based defense group collaborative behavior
Air group collaborative guiding behavior

Group model architecture
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
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 models
  - 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 scripts
- Each script corresponds to a graphical conceptual model
Model-instance separation of cognitive behavior scripts

- Instance specific scripts inherit from the default...
Each script corresponds to a graphical conceptual model - default aero object

5. a set of cognitive decision model scripts
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