The CKEMSim Integrated Flight Simulation (IFS)

Mac McCrum

Lockheed Martin Missiles and Fire Control

mac.mccrum@lmco.com
(972)603-1286

Developing an IFS - Issues to Address

"What is an IFS?" An IFS is any high fidelity non-real-time all digital simulation that contains the missile system's flight software.

"Does that mean we need to (or want to) rewrite our engineering simulation to match the language of our tactical embedded code?"

"If we mix languages, won't there be compatibility issues? What kinds of compilers work together?"

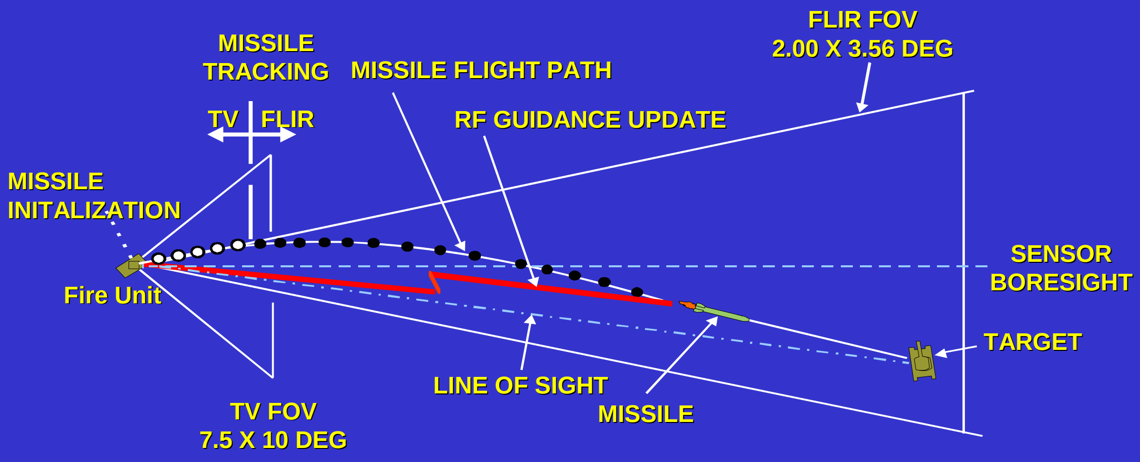
"How do we pass data across the interfaces between components compiled under different languages?"

"For sensor-driven embedded software, the data is much different from the data passed to the old processor emulation. How do we simulate the sensor output?"

"What pitfalls and strategies should I know about before I begin?"

"What's in it for me? Should I argue about this requirement?"

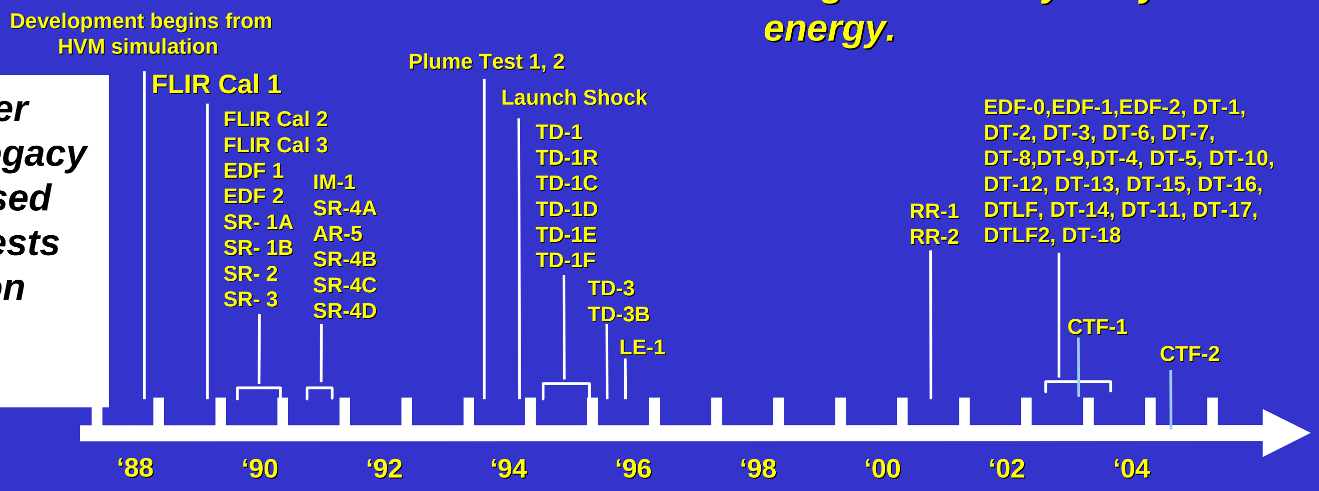
The CKEM Guidance Methods and Simulation Are Mature



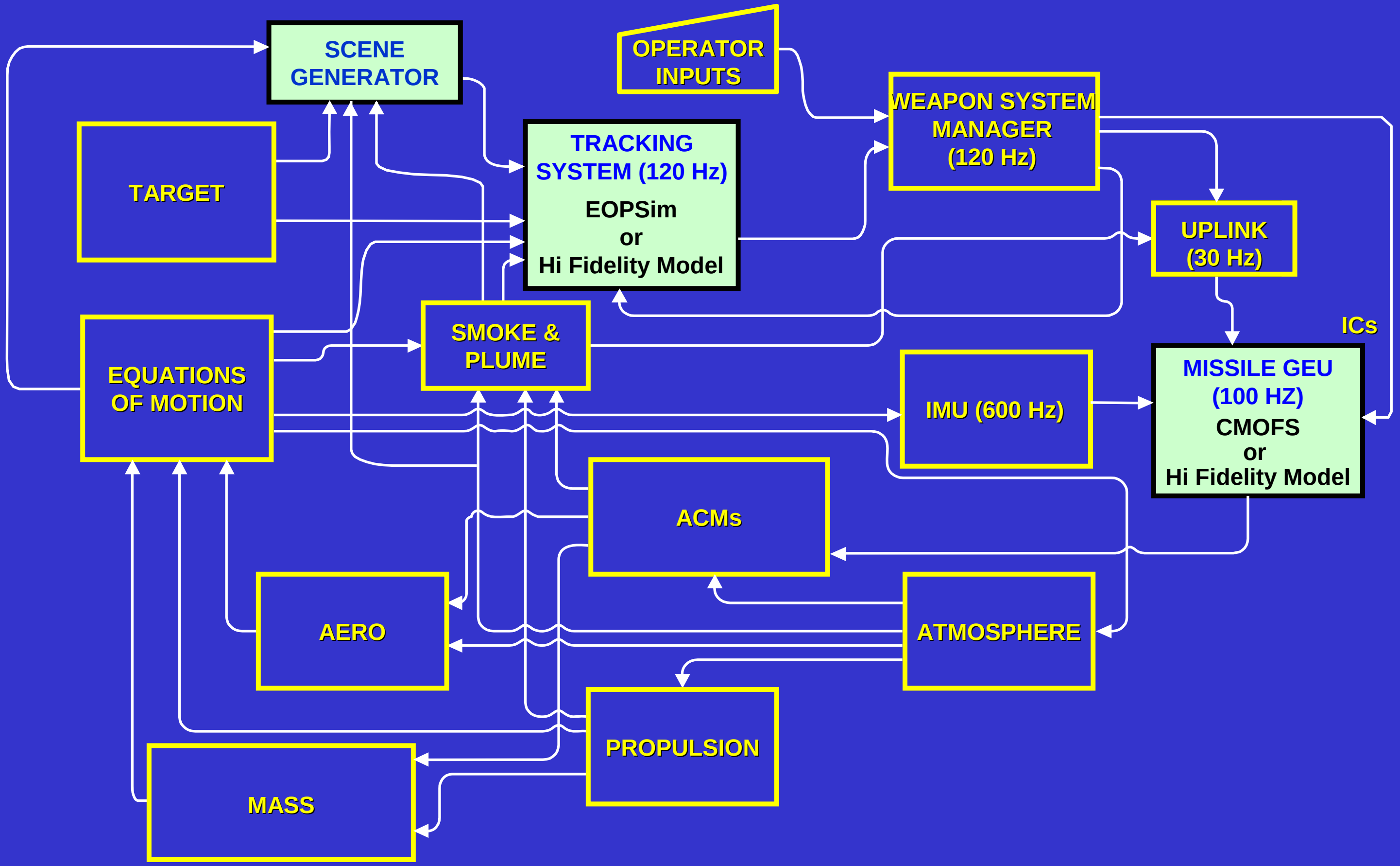
The Fire Unit:
- ***Acquires the target***
- ***Launches the missile***
- ***Tracks the missile and target via DayTV and FLIR***
- ***Sends guidance updates to the missile via the RF Uplink.***

The Missile:
- ***Corrects course by firing small steering motors (ACMs) in the nose of the missile***
- ***Target is destroyed by kinetic energy.***

The decision to reuse rather than rewrite CKEMSim's legacy FORTRAN models was based on 18 years and 54 flight tests of refinement and validation under the LOSAT® and CKEM programs.



CKEMSim Data Flow with EOPSIM and CMOFS IFS



- *CKEM SOW required the development of an Integrated Flight Simulation (IFS).*
- *The existing FORTRAN simulation was adapted by replacing the GEU and Fire Unit FORTRAN emulations with embedded code.*
- *A high fidelity synthetic scene generator was needed to drive the tracker software.*

CKEMSim Hardware and Software

Component	Developed By	Compiler Used	Hardware Required	Comments
C++ Model Developer (CMD) Executive	AMRDEC/ DESE Research	Visual C++ Toolkit	PC Processor	Open source C++ kernel used for simulation and scientific computing development
Tracker Software	Raytheon	Visual Studio 6.0 (C/C++)	PC Processor	Operational software recompiled for PC
Synthetic Scene Generator	LMMFC, with data from AMRDEC	Visual Studio.net 2003 (C++), Nvidia CG v 1.3 with OpenGL version 2.0	nVidia Graphics Card with 6600 or 6800 series chips (with 12-bit blending + CG), PC Processor	Produces a data stream identical to the FLIR and DayTV output
CKEM Missile Operational Flight Software	LMMFC	Visual Studio 6.0 (C++)	PC Processor	Operational software recompiled for PC
Missile and Target Truth Models	LMMFC	Lahey Fortran 95 v5.6, Visual C++ Toolkit	PC Processor	Legacy models

CMOFS to FORTRAN/C++ Attitude Control Motor Model Interface

- *CMOFS issues commands to fire Attitude Control Motors to correct course.*
- *This interface is typical of the interfaces developed for scene generation and tracker software.*

gpa_subsys.f

#include acmgpa_common.inc

```
acmgpa_common.inc
common acmgpa nsel, idacm(4), tfire_gpa(4), dt_acm_fire(4)
```

call dtsim_mofs_interface(. . . nsel, idacm, tfire_gpa, dt_acm_fire)

dtsim_mofs_interface.cpp

ACMs_to_Fire_Next_CC_Type ACMs

```
Execute_CMOFS_Comp_Cycle( . . . ,ACMs)
```

#include cmofs_interface.h

```
cmofs_interface.h

  struct Acm_To_Fire_Record_Type
  {
    double time-to-Fire // since time of breakwire
    int ACM_To_Fire // Physical ACM number
  };

  struct ACMs_to_Fire_Next_CC_Type
  {
    ACM_To_Fire_Record_Type ACM_1;
    ACM_To_Fire_Record_Type ACM_2;
    ACM_To_Fire_Record_Type ACM_3;
    ACM_To_Fire_Record_Type ACM_4;
  }
```

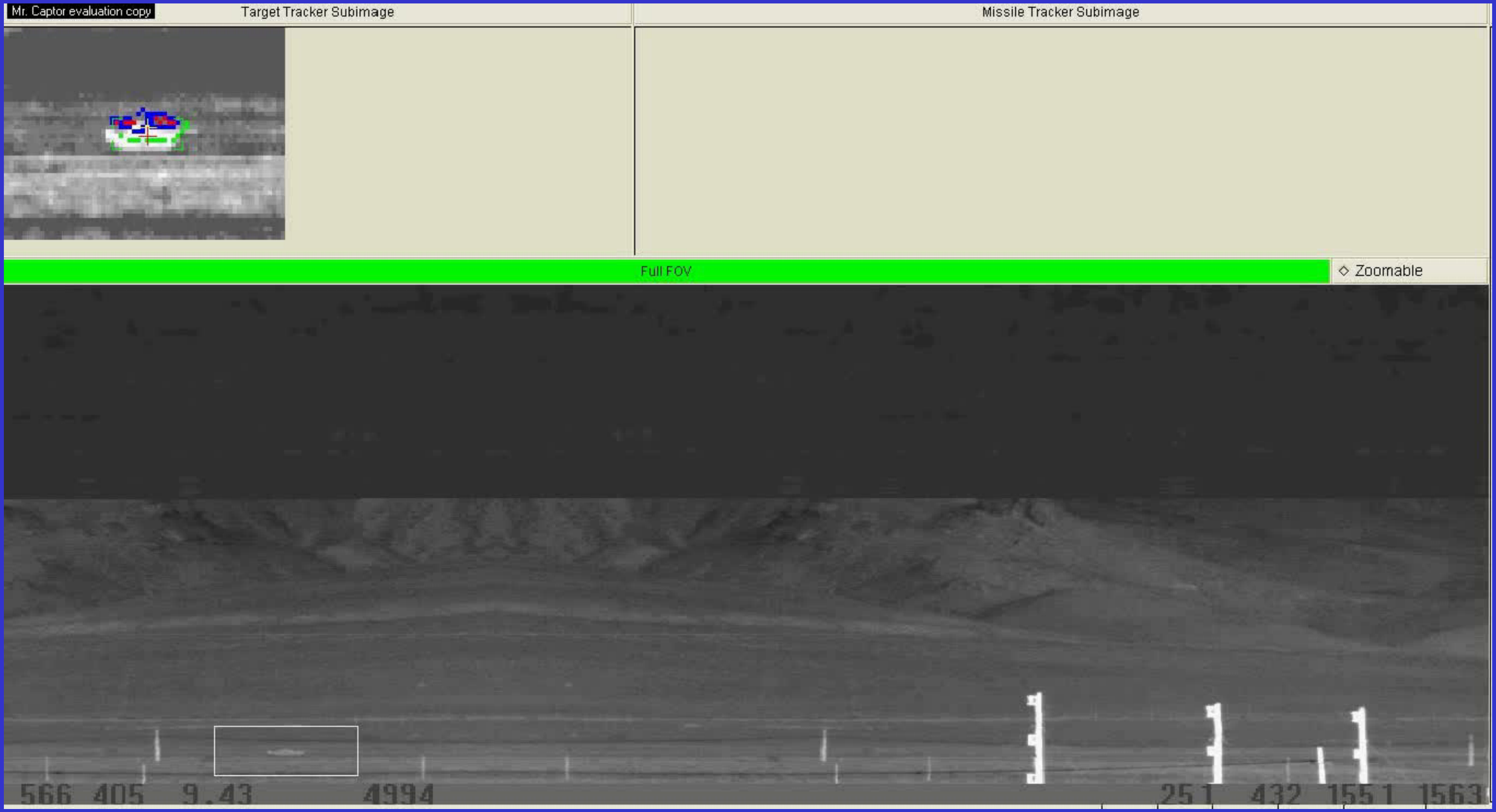
```
idacm[0] = ACMs.ACM_1.ACM_To_Fire;
idacm[1] = ACMs.ACM_2.ACM_To_Fire;
idacm[2] = ACMs.ACM_3.ACM_To_Fire;
idacm[3] = ACMs.ACM_4.ACM_To_Fire;
tfire_gpa[0] = ACMs.ACM_1.Time_To_Fire;
tfire_gpa[1] = ACMs.ACM_2.Time_To_Fire;
tfire_gpa[2] = ACMs.ACM_3.Time_To_Fire;
tfire_gpa[3] = ACMs.ACM_4.Time_To_Fire;
nsel=0
for (i=0;i<4;i++) {
  if (idacm[i] >0) {
    nsel=nsel++;  // count up acm selected
    dt_acm_fire[i] = tfire_gpa[i] – gpatm_tg;
  }
}
```

```
Extern "C"
{
include "dtsim_mofs_interface.h";
};
```

```
dtsim_mofs_interface.h

void dtsim_mofs_interface_(
  int &nsel,
  int idacm[4],
  double tfire_gpa[4],
  double dt_acm_fire[4]
)
```

Visualization of a Simulated Engagement



Lessons Learned

- *Intermixed languages work together reasonably well.*
- *Operational code needed to be revised to reinitialize between Monte Carlo samples.*
- *Graphics cards with the same chip, but different model numbers or different drivers, produced different results.*
- *Integration went smoothly because a clean interface was established up front.*
- *If you need a scene generator, find someone who's done it before successfully. Chances are even they will find it very difficult.*

Benefits

- *Prior capabilities of the simulation were retained.*
  - *Probability of hit analysis, ACM consumption rates, sensitivity analysis*
  - *HWIL driver, Raytheon SIL*
  - *Mission planning, range safety, algorithm development*
- *The synthetic scene generator can test the tracker software with a wider variety of flight conditions than just using flight test imagery and HWIL.*
  - *Can also use Monte Carlo methods*
  - *Specific Tracker algorithm problems exposed and fixed to date include inefficiencies in the search rate, and errant missile track rejection due to sporadic smoke obscuration*
  - *Synthetic imagery is the only way to test multigrain motor plume imagery before flight*
- *Allows HWIL testing to focus on timing and host processor issues.*
- *Closer match between simulation and embedded code = increased credibility.*