2Simulate Modeling Language - Wiederbelebung und Modernisierung von alter Simulationssoftware

Jürgen Gotschlich
DLR – Institut für Flugsystemtechnik, Braunschweig

2Simulate Modeling Language - Wiederbelebung und Modernisierung von alter Simulationssoftware

Agenda

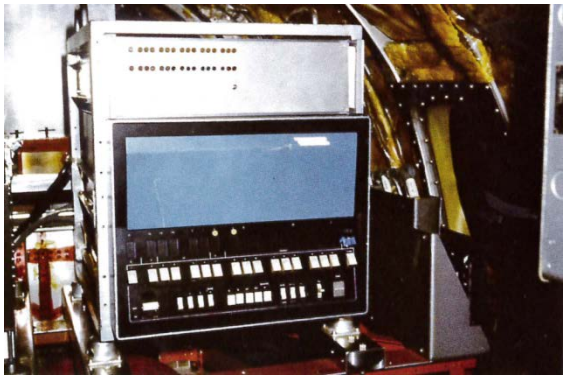
1. Historie der Flugsimulatoren im DLR Institut für Flugsystemtechnik
2. AVES - Air VEhicle Simulator
3. Intention für Wiederbelebung und Modernisierung von alter Simulationssoftware
4. Programmiersprache ADSIM
5. 2Simulate Modeling Language
6. Beispiel1: Feder-Masse-System
7. Beispiel2: Nichtlineares Flugzeugmodell
8. Zusammenfassung und Ausblick

DLR – Historie der Flugsimulatoren

HFB320 InFlight-Simulator (1975-1983)

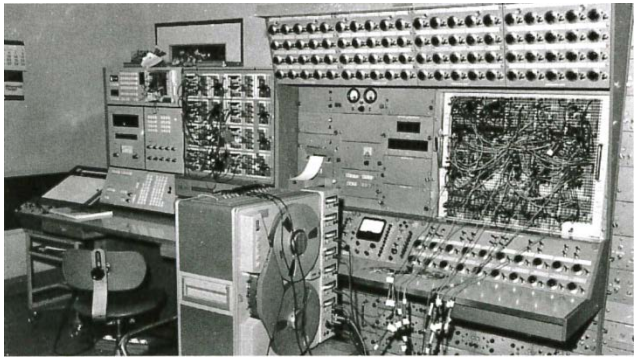
Bordrechner:

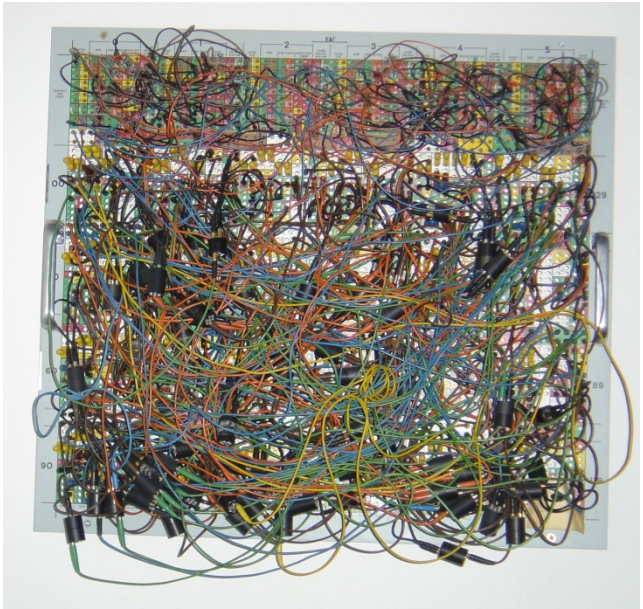
- Honeywell H316
- 16bit, 8K Kernsp.
- FORTRAN, Assem.



Bodensimulation:

- Hybridrechner EAI PACER600
- u.a. digitale Aerodynamik in FORTRAN
- 6-DOF Bewegungsgl. am Analogrechner



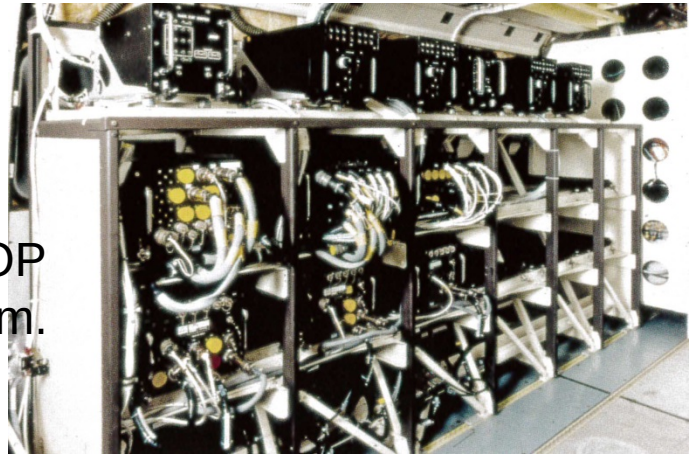


DLR – Historie der Flugsimulatoren

ATTAS InFlight-Simulator (1985-2012)

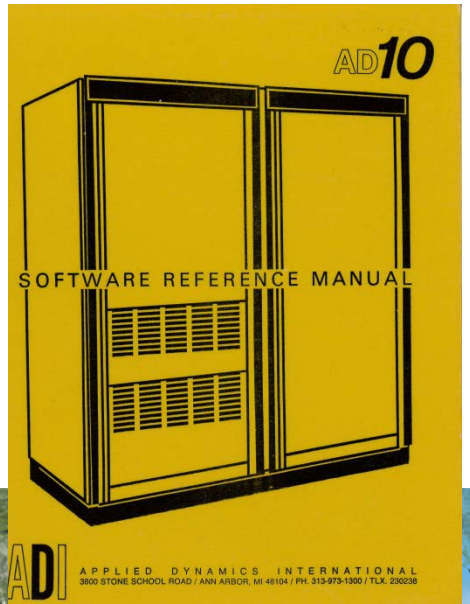
Bordrechner:

- ROLM
- MSE14, 16bit
- HAWK/32, 32bit
  - 2MB, 1.3MFLOP
- FORTRAN, Assem.



Bodensimulation:

- ADI AD10 (1983-1993),
  - Multiprozessor, ECL, 50 MIPS
  - MPS10
- ADI RTS (1993-2012) ,
  - MC88110, RISC, 70 MFLOPS
  - Upgrade: MVME1604, PowerPC, 200 MFLOPS
  - ADSIM

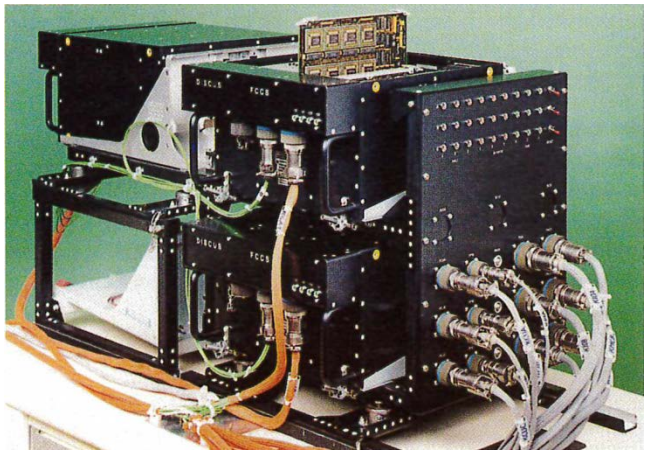


DLR – Historie der Flugsimulatoren

ATTHeS InFlight-Simulator (1986-1995)

Bordrechner:

- DISCUS
- Multiprozessor
- Triplex
- C



Bodensimulation:

- ADI AD100 (1986-1995),
  - Multiprozessor, ECL, 20 MFLOPS
  - ADSIM

DLR – Historie der Flugsimulatoren

FHS InFlight-Simulator (seit 1995)

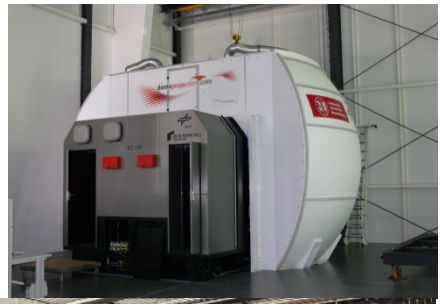
Experimental-Bordrechner:

- 3x Radstone PPC7A, latest PowerPC
- VMEbus
- VxWorks
- C++, Simulink

Bodensimulation:

- ADI RTS(1995-2012),
  - MVME1604, PowerPC, 133MHz
  - Upgrade: MVME2604, PowerPC, 400MHz
  - ADSIM

DLR – AVES – Air VEhicle Simulator (2013+)



- 2 Simulatoren
  - Bewegungssimulator (14t Hexapod)
  - Festsitzsimulator

- Cockpit-Wechselsystem
  - Schienensystem
  - Kranunterstützung

- 2 Cockpits
  - A320 ATRA
  - EC135 FHS

- Projektionssystem
  - 15 Kanal Front Projektion, LED
  - Auflösung 1920X1200 (WUXGA)

- Zugangsbrücke
  - 1 t / 5 t Belastung

DLR – AVES - Air VEhicle Simulator (2013+)

EC135 FHS Cockpit

- FHS Replikat (RDE)
- 8 Kanal aktives Steuerkraftsystem (Wittenstein.)
- Standard PC Komponenten (COTS)
- Ansteuerung via CAN Bus
- Interface Computer (QNX, 2Simulate)
- Alle Software vom DLR
- Modell-Programmiersprache: ADSIM

A320 ATRA Cockpit

- ATRA Replikat (RDE)
- 2 aktive Sidesticks (Wittenstein.)
- Standard PC Komponenten (COTS)
- Ansteuerung via CAN Bus
- Interface Computer (QNX, 2Simulate)
- Alle Software vom DLR
- Modell: Simulink → Coder → 2Simulate

Absicht

- Mehr als **drei Jahrzehnte Know-how** über Modellierung und Simulation von hochkomplexen Flugzeugsystemen unter Echtzeitbedingungen

- Als **unschätzbarer Wert** des DLR, soll der Simulations-Code **wiederbelebt und migriert werden** , u.a. zur Nutzung auf dem neuen Air Vehicle Simulator (AVES)

*„Wiederverwendung von Legacy-Code ist effektiver, als zu versuchen, mit Re-Engineering den Code in eine andere Sprache zu portieren“ **

Stimmt das ?

→ *„Wiederverwendung ist gut, aber Re-Engineering ist besser ! „*

→ *„Entwicklung einer Modellierungssprache ähnlich ADSIM auf Basis einer C++ API“*

*J. Pullen and E. White, "Adapting Legacy Computational Software for XMSF," in *Fall 2003 Simulation Interoperability Workshop*, Orlando, FL, 2003

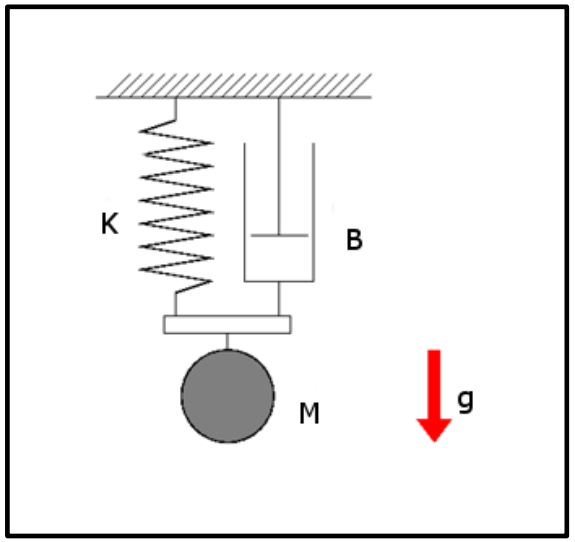
Programmiersprache ADSIM

Domain-Specific-Language zur Modellierung von dynamischen Systemen

- mathematische Notation
- Syntax für Zustandsvariablen
  - Deklaration : ' - Operator (x' = -w2*y)
  - Anfangswert : @ - Operator (@x = 0.0)
- Integrierte Ablaufsteuerung
  - Start / Reset / Halt
  - Berechnung des aktuellen Zustands (Rechte Seite)
  - Integration und Update der Zustandsvariablen
- Block-Struktur (REGION: INITIAL, DYNAMIC, TERMINAL)
- Auswahl von Echtzeit-Integrationsalgorithmen (Adams-Bashforth,…)

Programmiersprache ADSIM

Beispiel:



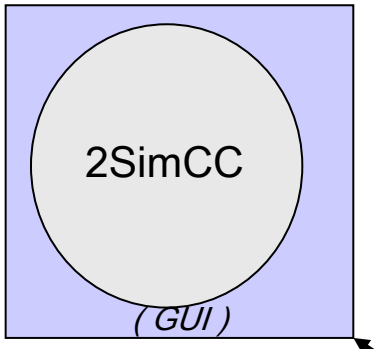

```
TITLE „Mass-Spring-Damper system in ADSIM"

REGION initial
      x@ = 0,      y@ = 0.3
END REGION

DYNAMIC continuous
      y' = x                                  ! velocity
      x' = - (K/M * y) – (B/M * x) + g        ! acceleration
END DYNAMIC

DATA   M = 1.0, K = 10.0, B = 9.81
```

Programmiersprache ADSIM

ADSIM Sprachumfang:

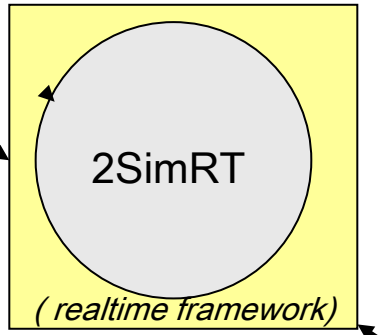
- Vektor- und Matrix-Arithmetik (y = C*x + D*u)
- Block Struktur
  - Prozedurale Blöcke: DYNAMIC, MODEL
    - Deklarationen
    - Algebraische Zuweisungen
    - Zustandsvariablen Zuweisungen
  - Nicht-Prozedurale Blöcke: REGION, FUNCTION, SIMEXEC
    - Deklarationen
    - Algebraische Zuweisungen
    - Kontrollanweisungen
- Sortierung der Zustandsgleichungen zwecks Optimierung
- Umfangreiche Funktionsbibliothek
  - Interpolation, Extrapolation, Limiter

2Simulate – Das AVES Simulationswerkzeug



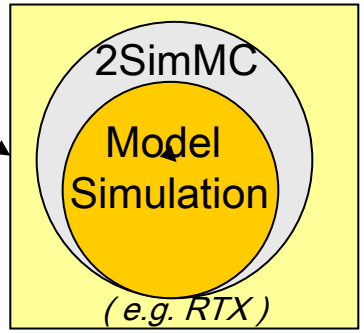
2Simulate Control Center

- Graphische Bedienoberfläche



2Simulate Realtime Framework

- Realtime Task Scheduler
- API



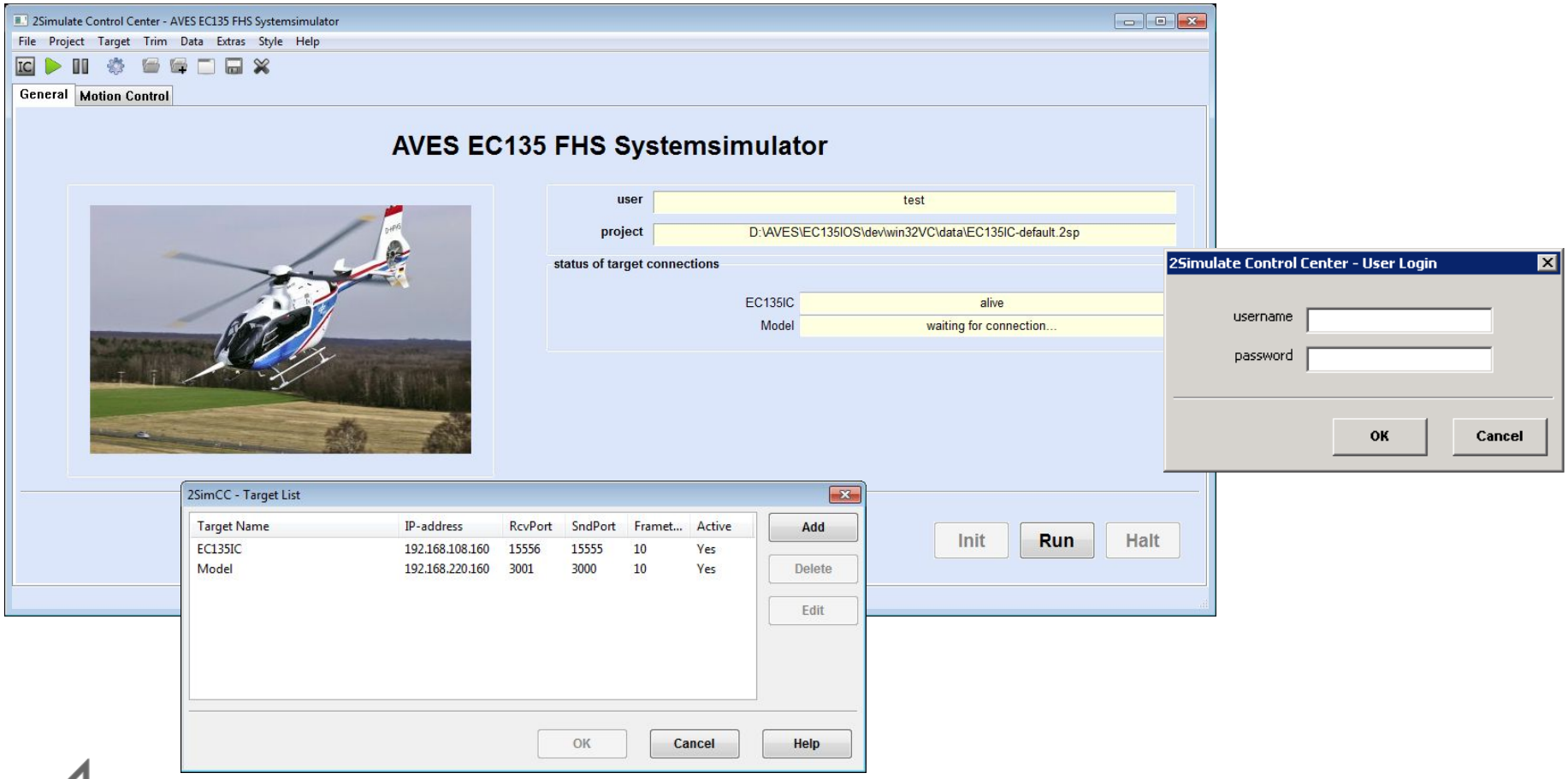
2Simulate Model Control

- Modellschnittstelle

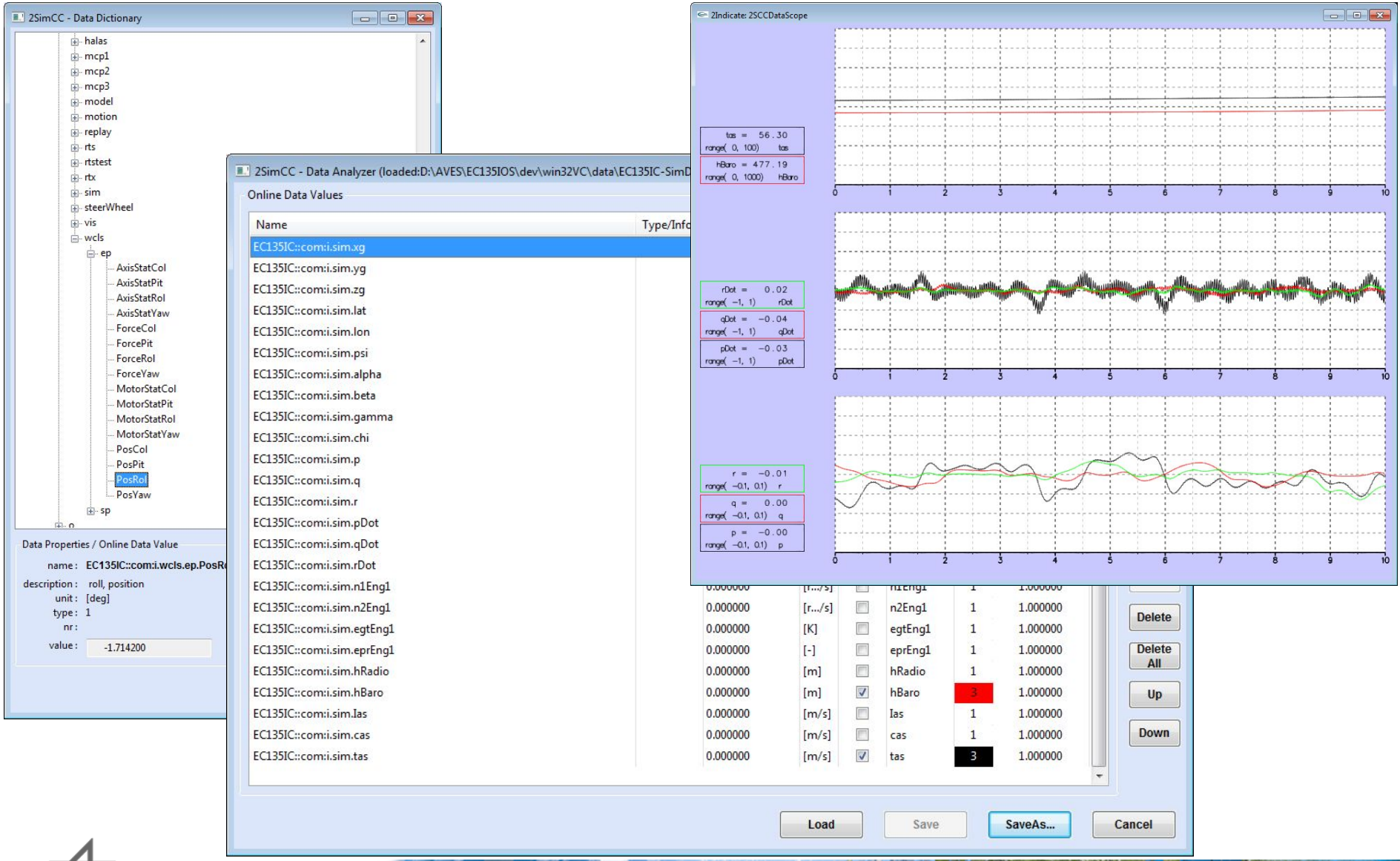
2SimCC – Steuerung von Simulatoren

GUI zur generischen Konfiguration und Steuerung von Simulatoren

- Beispiel: EC135 Instructor Operator Station
  - Nutzer-, Projekt- und Targetverwaltung



2SimCC – Daten Management und Visualisierung

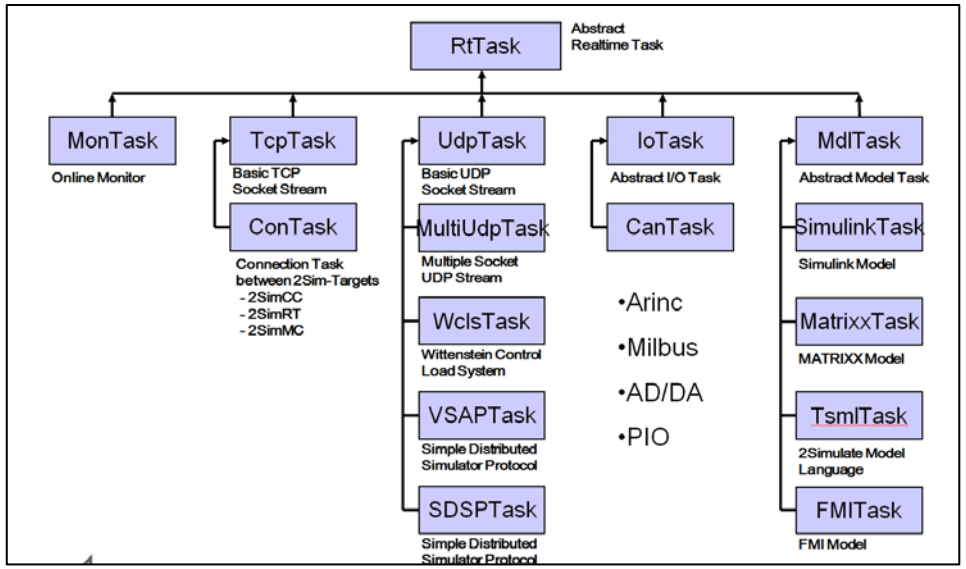


2SimRT – 2Simulate Realtime Framework API

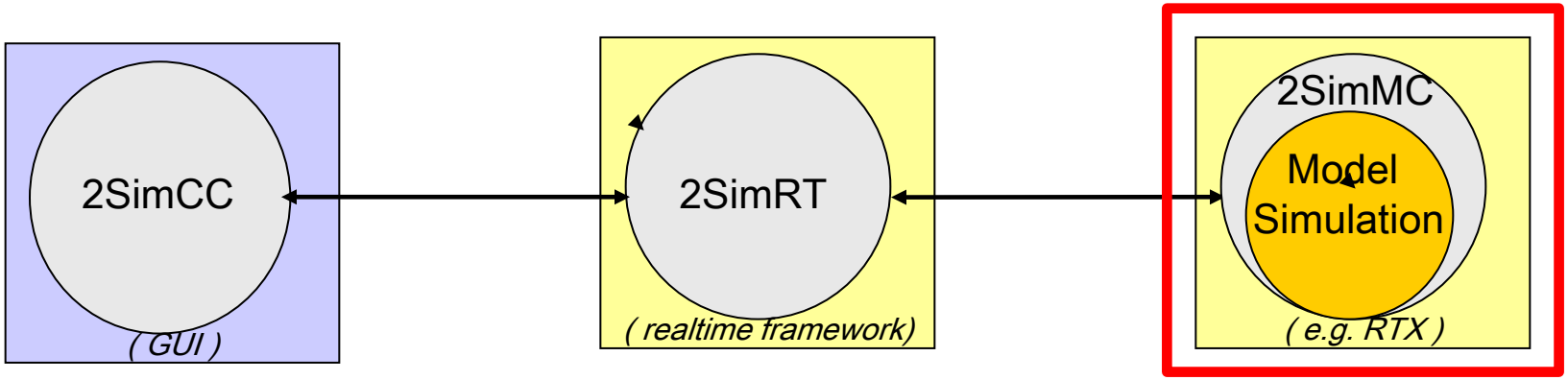
- Interface Bibliothek für kontinuierliche Echtzeit Prozessierung
  - Basis für 2Simulate Target (z.B. Interface Computer)
  - Plattformunabhängig: Windows, QNX
  - C++
  - Generische, modulare Task Objekte

Beispiel zur Integration eines Model-Tasks:
Feder-Masse-System

```
#include „FMS.h“

TsmlTask *pTask = new TsmlTask( pTSim, „FMS", 10*iMS);

FMS* pFMS = new FMS( "FMS" );
pTask ->addModel( pFMS );
```



2SimMC- 2Simulate Modellsteuerung



- Einbettung des Modells auf dem Target System
- Instanzen für Simulink, MatrixX, ADSIM, 2SimML, …
- Modellsteuerung durch Steuerkommandos: IC, RUN, HALT, TRIM
- Bereitstellung einer Modelldatenbank
- Zugriff auf alle Modelldaten (states, derivatives, input, output, parameter)
- Integrierte Trimmrechnung zur Herstellung eines beschleunigungsfreien Modellzustands möglich

2SimML - 2Simulate Modeling Language

- Basiert auf C++
  - Für Ingenieure: keine speziellen Kenntnisse notwendig !!!
    - Für C++-Kenner: abstrakte Basisklasse
  - Integrierte Vektor- und Matrix-Arithmetik (y = C*x + D*u)
  - Ein vorgegebener Satz von Methoden kann/muss zur Implementierung eines Modelles benutzt werden
  - Methoden werden vom 2Simulate Framework aufgerufen
    - Kontinuierliche Modellsteuerung: IC, RUN, HALT
    - Berechnung der rechten Seiten
    - Integration und Update der Zustandsvariablen
    - Datenaustausch

2SimML - 2Simulate Modeling Language

- Notwendige Methoden:
  - doOneFrame() : Berechnung der RHS für einen Zeitschritt
  - setIC() : Setzen von Anfangsbedingungen
  - initDataDict() : Initialisierung eines Data-Dictionaries

- Mögliche Methoden:
  - defineStates() : Deklaration von Zustandsvariablen
  - initData() : Initialisierung von Daten beim Programmstart
  - initSubModels() : Initialisierung von Submodellen
  - initFunctions() : Initialisierung von Funktionen
  - getInputData() : Eingangsdaten lesen
  - putOutputData() : Ausgangsdaten schreiben
  - …

- Z.Zt. implementierte Echtzeit-Integrationsalgorithmen: Adams-Bashforth2,3,4

2SimML - 2Simulate Modeling Language

2SimML	ADSIM
setIC()	(interne Steuerung)
doOneFrame()	DYNAMIC continuous
initDataDict()	(nicht vorhanden)
defineStates()	(interne Steuerung)
initData()	REGION initial
initSubModels()	MODEL xxx
initFunctions()	INTERPOLATION_FUNCTION xxx
getInputData()	routine_interface with C : getInputData()
putOutputData()	routine_interface with C : putOutputData()

Beispiel1: Feder-Masse-System

```
void FMS::initData( void ) {
  M =  1.0;                    // [Kg] mass
  K = 10.0;                    // [N/m] spring constant
  B =  0.2;                    // [Kg/s] damping
}

void FMS::setIC( void ) {
  x = 0.0;                     // x initial value
  y = 0.3;                     // y initial value
}

void FMS::defineStates( void ) {
  addState( x, xDot, "x", "velocity", "m/s"  );
  addState( y, yDot, "y", "position", "m" );
}

int FMS::doOneFrame( void ) {
  xDot = -(K/M * y) - (B/M * x) + g;
  yDot = x;
}
```

Initialisierung

Mathematisches Modell

Beispiel2: Nichtlineares Flugzeugmodell (1)

```
void NonlinAC::getInputData( void ) {
  etaCmd  = aInDataRef.acCtrlData->etaCmd;
  …
}

void NonlinAC::putOutputData( void ) {
  aOutData.ukg   = ukg;  // [m/s]      longitudinal velocity (north-south-veloctity)
  aOutData.vkg   = vkg;  // [m/s]      lateral velocity (east-west-velocity)
  aOutData.wkg  = wkg;  // [m/s]       vertical velocity (inertial vertical speed)
  …
  aOutData.tas    = TAS;  // [m/s]      true airspeed
  …
}

void FMS::initData( void ) {
  …
  Mass_ic          = 19224.849609;     // [kg]
  fuel_init        =   3630.399902;    // [kg]
  xCG_p_actual   =      24.922340;     // [%]
 …
}
```

Eingang vom Joystick

Ausgangsvektor

Initialisierung von Daten beim Programmstart

Beispiel2: Nichtlineares Flugzeugmodell (2)

```
void FMS::initFunctions( void ) {
  …
  el_alpha_aFct = new Linint( "ehs_el_alpha_a.fcn", bpt_el_etak_deg_9 , 1 );
 …
}
```

Initialisierung der Funkt. für Lineare Interpolation

```
void FMS::initSubModels( void ) {
  …
 m_pQuat = new Quat( "Quaternion" );
 m_pGear = new Gear( „Gear" );
…
}
```

Initialisierung der Submodelle

```
void FMS::setIC( void ) {
  …
  xg    = xg_ic;
  yg    = yg_ic;
  zg    = -H_ic;
  H     = - zg;
  H_ground  = max ( 0.0, ( H - ground_level ) );
 …
}
```

Setzen von Anfangswerten beim Modellstart

Beispiel2: Nichtlineares Flugzeugmodell (3)

```
void FMS::defineStates( void ) {
  …
  addState( ukf, ukf$DOT, "ukf" );
  addState( vkf, vkf$DOT, "vkf" );
  addState( wkf, wkf$DOT, "wkf" );
  …
}

int FMS::doOneFrame( void ) {
  …
  //---- aerodynamics in body fixed frame ----
  cX  =   cA_WB*sin_alpha - cW*cos_alpha + cX_HT*SH_by_S;
  cY  =   cQ;
  cZ  = - cA_WB*cos_alpha - cW*sin_alpha + cZ_HT*SH_by_S;
  cl   =   cl_25*cos_alpha - cn_25*sin_alpha + ( cY*delta_zCG_m - cZ*delta_yCG_m )*sl_inv;
  cm =   cm_25 - cX*delta_zCG_m*lmy_inv + cZ*delta_xCG_p;
  cn  =   cn_25*cos_alpha + cl_25*sin_alpha + ( cX*delta_yCG_m - cY*delta_xCG_m )*sl_inv;
  …
  //---- accelerations in body fixed frame ----
  ukf$DOT = -qr*wkf + rr*vkf - GN*sin_THETA + a_CG_x;
  wkf$DOT = -pr*vkf + qr*ukf + GN*cos_THETA*cos_PHI + a_CG_z;
  vkf$DOT  = -rr * ukf + pr * wkf + GN * cos_THETA * sin_PHI + a_CG_y;
 …
}
```

Definition der Zustandsvariablen

Mathematisches Modell

Zusammenfassung und Ausblick

- Weiterverwendung von altem Simulationscode in ADSIM ist möglich

- Portierungsaufwand stark abhängig von Komplexitätsgrad
  - Einfache Modelle innerhalb sehr kurzer Zeit
  - Komplexe Modelle beliebig länger, aber möglich
  - Auch spezielle Anforderungen sollten durch die offene C++-Programmierung erfüllt werden können

- Ausblick: Erweiterung der Sprache durch Ableitungs-Operator (y') und automatische Verwaltung der Zustandsgrößen/Derivative

Vielen Dank !

DLR