Comparison 11 – ACSL

Hybrid Modelling Approach – Environment Level

ACSL is a widely used, compiler-based simulation language for continuous models with textual and graphical model description. It provides explicit and implicit integration algorithms and (beside others) event handling features. ACSL Math is a convenient experimentation environment for ACSL with numerous analysis and graphical tools. It is based on MATLAB syntax and can make use of MATLAB m-files.

Model description (Task a): ACSL allows the description of implicit models (and DAE models) by means of an `IMPLC` operator, which either breaks an algebraic loop before a numerical integration step or calls directly an implicit integration scheme (DASSL Code).

The following abbreviated `DERIVATIVE` Section shows the essentials of the implicit model description

```
DERIVATIVE ! Implicit Dynamic Model
ma11 = th1+2*th2*c2+th3; ma12 = ...
b1 = t1+th2*(2*dq1*dq2+dq2**2)*s, b2 = ...
residdq1 = ma11*ddq1 + ma12*ddq2 -b1
residdq2 = ma21*ddq1 + ma22*ddq2 -b2
residdq3 = ma33*ddq3  -b3
dq1, ddq1 = IMPLC(residdq1, dq1ic)
dq2, ddq2 = IMPLC(residdq2, dq2ic)
dq3, ddq3 = IMPLC(residdq3, dq3ic)
q1 = INTEG( dq1, q1ic); q2 = ...
END ! of Derivative
```

When using a standard integration algorithm (e.g. Runge Kutta 4th order) the algebraic loop for the derivatives `ddqx` within the `IMPLC` statement and the equations for the variables `residxx` is broken by a Newton-Raphson iteration. If the DASSL Code for direct integration of implicit systems is chosen the variables `residxx` represent the residuum for the algorithm. In order to compare the two implicit methods an explicit model was also programmed.

Employing ACSL Math allows to transfer all necessary parameter initializations and pre-calculations (outside of the integration loop) to an ACSL Math m-file that can be used for both the implicit and the explicit models. Different integration algorithms can be chosen by assigning appropriate values to the `IALG` parameter.

Point to Point Control (Task b): Servo motors and controllers can be easily implemented by standard modelling features of ACSL. Figure 1 shows the time history for the joint angles (results of implicit and explicit model look identical). The following table compares the normalized simulation times for a simulation over 2 sec. For the implicit model the DASSL code is faster than the Runge-Kutta algorithm. But as expected, execution of the explicit model is considerably faster compared to the implicit one.

Computation time is not affected by using ACSL Math as a runtime interpreter. However, models can be switched easily by loading the respective model into the ACSL Math workspace via the `LOAD` command. This facilitates the model comparison significantly.

```
load @file=scara_exp @format=model
 !!prepare t,q1,q2,q3
 tic(); start, toc()
load @file=scara_imp @format=model
 !!prepare t,q1,q2,q3
 tic(); start, toc()
```

Model Description	Implicit		Explicit
Integr. Algorithm (Stepsize 0.005 s)	RK-4 `IALG=5`	DASSL `IALG=10`	RK-4 `IALG=5`
Norm. CPU-time	1.0	0.86	0.12

Computation times on a HP715/100, ACSL Vers. 11

Obstacle avoidance (Task c): To detect a state event the `SCHEDULE` operator is used in ACSL which starts an iterative state event locating routine and finally executes a `DISCRETE` Section. For collision avoidance a generic `SCHEDULE` command is used in the ACSL model description and the actual state event to be checked for is selected via the index variable `ichk`.

```
chkvar(1)=d-dcr; chkvar(2)=h
SCHEDULE event .XZ. chkvar(ichk)
```

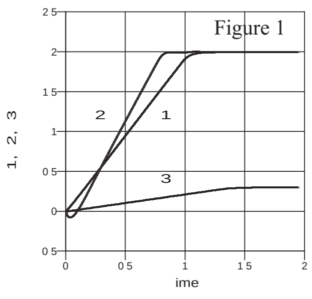
If the value of the checked variable `chkvar(ichk)` crosses zero the `DISCRETE` section `"event"` is triggered and the simulation run is terminated:

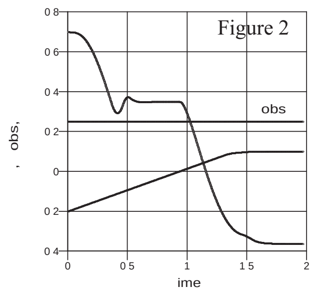
```
DISCRETE event ; TERM(.true.) ; END
```

Parameter changes, restart of simulation and sampling of output data is all done by an ACSL Math script file:

```
!! PREPAR t, x, xobs, h
Q1IC=0; Q2IC=0; Q3IC =0; … % set init. cond.
ichk = 1;  % Check obstacle distance
!! START
collect_data  % script to save prepared data
if (h < 0)
  set_par1 % script to set new parameters
  reinit  % script resets initial conditions
  ichk = 2  % Check h if obstacle cleared
  !! START
  collect_data  % script saves prepared data
end
set_par2 % script to set new parameters
reinit  % script to reset initial conditions
collect_data  % script to save prepared data
!! START
plot(time,d,time,x_obs,time,h)
```

Figure 2 was plotted from ACSL Math. It shows that the tool tip does not cross the obstacle border line until its height has reached a positive height above the obstacle.





Horst Ecker, Institute for Machine Dynamics and Measurement, TU-Vienna, Wiedner Hauptstr. 8-10, A-1040 Vienna/ Austria, email: hecker@email.tuwien. ac.at