Comparison 11 - ACSL
Implicit/DAE Modelling Approach

ACSL is a well known and widely used, compiler-based simulation language for continuous models. It provides explicit and implicit integration algorithms, event handling features and a powerful experimentation environment via ACSL Math. ACSL offers both textual and graphical model description.

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, dq10)
dq2, ddq2 = IMPLC(residdq2, dq20)
dq3, ddq3 = IMPLC(residdq3, dq30)
q1 = INTEG( dq1, q10); q2 = ...
END ! of Derivative
```

If a standard integration algorithm is chosen (via `IALG`-parameter), the algebraic loop for the second derivatives `ddqx` within the `IMPLC` statement and the equations for the variables `residxx` is broken by a Newton-Raphson iteration within each evaluation of the derivatives. Since version 10.2 ACSL offers also the DASSL-Code for direct integration of implicit equations. If this algorithm is chosen the `residxx` variables represent the residuum for the algorithm. In order to compare these two implicit methods also a "classical" approach was programmed.

```
DERIVATIVE ! Explicit Dynamic model
ma11 = th1+2*th2*c2+th3; ma12 = ...
b1 = t1+th2*(2*dq1*dq2+dq2**2)*s2; b2 = ...
det  = ma11*ma22 - ma12*ma21
ddq1 = (ma22*b1 - ma12*b2)/det
ddq2 = (ma11*b2 - ma21*b1)/det
ddq3 = b3/ma33
dq1  = INTEG(ddq1,dq10); dq2 = ...
q1   = INTEG( dq1, q10);  q2 = ...
END ! of Derivative
```

From the viewpoint of implementation, the implicit method is to be preferred. It is very simple to formulate whereas the symbolic inversion of the mass matrix would be practically impossible in case of a large system. A numerical inversion of the mass matrix could be implemented by means of external FORTRAN subroutines.

Point to Point Control (Task b): Servo motors and control can be easily implemented by standard modelling features of ACSL like the limiting integrator `LIMINT`. Figure 1 shows the results for the joint angles using the DASSL Code. The following table compares the normalised simulation times for a simulation over 2 sec (execution is very fast because ACSL is a compiling simulator). As expected the DASSL Code is faster than the iterative loop breaking method using a standard Runge-Kutta algorithm. However, in this case the explicit model is still significantly faster.

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): State events may be described in ACSL by `DISCRETE` Sections that are managed by `SCHEDULE` operators which start an iterative state event locating routine.

For collision avoidance two such sections are used: `Obs_Stop` is activated if the distance `d` to the obstacle borderline falls below the critical distance `dcr` and if the alarm switch `alon` is set. Within this first `DISCRETE` Section the target position for the joint angles is changed temporarily and set to the momentary position. To guarantee maximum deceleration the voltage limits `LmU` are changed to the maximum values `MxU`. The second `DISCRETE` Section `Obs_Clear` resets all modified parameters when the tool-tip position `q3` has reached a level above the obstacle height (q_3-h_{obs} = $h > 0$).

```
SCHEDULE Obs_Stop .XN.(alon*(d-dcr)+ aloff)
SCHEDULE Obs_Clear .XP. h
:
DISCRETE Obs_Stop
q1tp=q1; q2tp=q2;LmU1=MxU1; LmU2=MxU2
alon = 0; aloff = 1.d0
END    ! obs_stop
DISCRETE Obs_Clear
q1tp=q1fin; q2tpc=q2fin;
LmU1=OpU1;LmU2=OpU2; alon=0;aloff=1
END    ! obs_clear
```

Figure 2 shows that the *x*-position of the tool-tip does not cross the obstacle borderline (dashed line) until the tool-tip height has reached a positive height above the obstacle. Identical results were obtained with the implicit model description using standard integrators and the DASSL-code. However, a considerable increase in computational time was observed for the last-named, caused by the usage of state events.

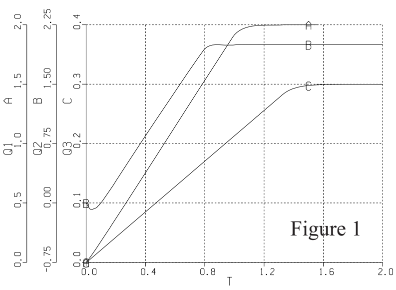

Figure 1

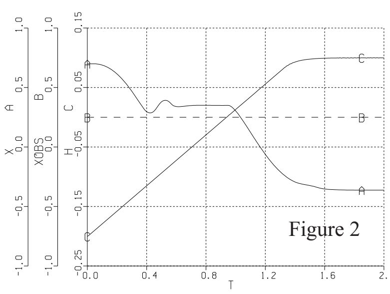

Figure 2

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