A. Schiela, H. Olsson:
Mixed-mode Integration for Real-time Simulation.
Modelica Workshop 2000 Proceedings, pp. 69-75.

Paper presented at the Modelica Workshop 2000, Oct. 23.-24., 2000, Lund, Sweden.

All papers of this workshop can be downloaded from
http://www.Modelica.org/modelica2000/proceedings.html

Mixed-mode Integration for Real-time Simulation

Anton Schiela, DLR, Oberpfaffenhofen, Germany
Hans Olsson, Dynasim AB, Lund, Sweden

ABSTRACT

The new possibilities of multi-domain hierarchical modelling often lead to models with both fast and slow parts. In this paper a new approach to simulate such systems is discussed that is especially useful in real-time applications. Mixed-mode integration represents a middle course between implicit and explicit integration. The main idea is to split up the system into a fast and a slow part and to apply implicit discretization only to the fast part. The partitioning of the system can be performed offline using a newly developed automatic selection routine, before real-time simulation starts. Mixed-mode integration was applied to several Modelica models from different fields, e.g. models of a diesel engine and an industrial robot and tested using Dymola. Speedup factors from about 4-16 were recorded. In this paper, mixed-mode integration is introduced, the selection routine is described and numerical results are presented.

Introduction

Real-time simulation is a growing field of applications for simulation software. One goal is to be able to simulate more and more complex models in real-time with fast sampling rates. Many of those models are multi-domain models, which means, that they contain components from more than one physical domain. Mechanic, electric, hydraulic or thermodynamic components are often coupled together in one model. This leads to a large span of time-constants in the model.

In many classical integration methods the fastest time-constant determines the computational effort for the simulation, which is too high in many cases. Mixed-mode integration is one way to simulate such systems efficiently.

Problems in real-time simulation

The task of real-time simulation is different from the classical offline simulation and therefore poses different problems.

Special requirements

Compared to the classical problem of offline simulation, real-time simulation demands fundamentally different properties of the underlying integration routines, to solve an ordinary differential equation:

$$\dot{x} = f(x)$$

Offline integrators try to minimise the overall integration time at a given (high) accuracy, using highly sophisticated step size- and order control mechanisms and high-order methods.

In real-time simulation the computer typically communicates with peripheral hardware components during the simulation. This communication takes place using a fixed small sampling interval e.g. 1ms. The simulation must perform the time-step and provide its results before this interval ends. Exceeding this “deadline” would be an error. So the aim of real-time simulation is not to reduce the average computational cost, but to make sure, that the calculation time for one step never exceeds this time limit. This leads to a different choice of integration methods. To minimise the computational cost for one time-step usually a very simple discretization scheme is chosen as for example the explicit Euler-method.

Stiffness

Special problems are caused by stiff systems. These are systems with dynamically very fast and highly damped components. If an explicit method is used to integrate such systems, step size is limited due to stability problems of the integration method. If step size is too large, then the computed trajectory starts to oscillate and diverges. The simplest example for an explicit method is explicit Euler:

$$x_{n+1} = x_n + hf(x_n)$$

The standard cure for unstable behaviour is to use implicit methods. Implicit methods evaluate the right-hand side at future time points. This leads to a non-linear equation system that has to be solved at each step. The simplest example for an implicit method is implicit Euler:

$$x_{n+1} = x_n + hf(x_{n+1})$$

At each time-step this equation has to be solved for x_{n+1}.

For real-time integration both methods have their own severe drawbacks, which limit drastically the size of stiff systems that can be simulated in real-time. Explicit methods used for stiff systems usually demand step sizes that are much lower than the given real-time step size. As stiffness increases, the step size tends to zero. The basic problem of implicit methods is the non-linear equation system to solve at each time step. In conventional methods, the dimension of this system is at least the number of states. The solution process is iterative and requires the costly evaluation of the Jacobian of the right hand side.

Offline methods can decrease the average computation time for one implicit step by not evaluating the Jacobian at each time step and by allowing slower convergence slowly for the non-linear solver once in a while [1]. All these considerations are not possible dealing with real-time simulation, as each single step must meet the real-time requirements. In this context the use of implicit methods is very critical.

Fast and slow components

The situation is especially unsatisfactory when both fast and slow components are present in the model to be simulated. Especially multi-domain modelling often results in such systems. For example in mechatronical systems a slow mechanical part is often controlled by fast electric circuits or by a hydraulic drive. Unfortunately the fastest time-constant governs the stability of the whole system for explicit methods and if only one component becomes unstable the results are worthless. In the classical sense, the whole model has to be treated as stiff.

To choose between an explicit and an implicit method means that either the system is integrated at a too small step size or a non-linear equation system of huge size has to be solved at each step, although the fast part is very small. Here, the choice of an implicit method leads to similarly unsatisfactory results as the choice of an explicit one.

Mixed-mode integration

As we have seen, both implicit and explicit methods have performance problems simulating a model with fast and slow components. This situation motivates the idea of finding a middle course between implicit and explicit. One idea is to cut the system into two pieces: a (hopefully) small fast system that can be treated easily with implicit methods and a slow system where cheap explicit methods can be applied on: Mixed-mode integration.

Principles

The basic idea is simple. We perform a row-wise partitioning of the right hand side, which results in a partitioning of the states into fast and slow ones:

$$\dot{x}^S = f^S(x^S, x^F)$$
$$\dot{x}^F = f^F(x^S, x^F)$$

Here the superscript "*F*" denotes the fast, and the superscript "*S*" the slow part of the system.

Then, an implicit discretization formula for step size *h* is applied to the fast and an explicit one to the slow system, for example the Euler-formulas:

$$x^S_{n+1} = x^S_n + hf^S\left(x^S_n, x^F_n\right)$$
$$x^F_{n+1} = x^F_n + hf^F\left(x^S_{n+1}, x^F_{n+1}\right)$$

The first of these equations can be evaluated explicitly and the result is inserted into the second one. This reduces the size of the non-linear equation system from the number of states to the number of fast variables.

Note that the resulting method still has convergence order 1. For higher order Runge-Kutta methods certain coupling conditions have to be fulfilled which is described e.g. in [2, pp. 302-311]. These conditions trivially hold for the combination of implicit and explicit Euler.

The linearized model

In the following, the case of a linear differential equation system is regarded, which makes a closer analysis possible.

The linear differential equation of dimension *d*

$$\dot{x} = Ax$$

is partitioned by multiplying *A* with a diagonal projection matrix $P = diag(\delta_1,..\delta_n); \delta_i \in \{0,1\}$ from the left to select the slow part and with *I-P* to select the fast part.

$$A = PA + (I - P)A$$

P selects rows of *A*. *PA* is the slow part and $(I-P)A$ the fast part.

The partitioned differential equation is:

$$\dot{x}^S = P\dot{x} = PAx$$
$$\dot{x}^F = (I - P)\dot{x} = (I - P)Ax$$

The two equations are discretized as:

$$x^S_{n+1} = Px_{n+1} = Px_n + hPAx_n$$
$$x^F_{n+1} = (I - P)x_{n+1} = (I - P)x_n + h(I - P)Ax_{n+1}$$

Using explicit Euler for the slow part and implicit Euler for the fast part.

Adding those two equations yields:

$$x_{n+1} = x_n + hPAx_n + h(I - P)Ax_{n+1}$$

That can be solved for x_{n+1}:

$$x_{n+1} = (I - h(I - P)A)^{-1}(I + hPA)x_n$$

which is again a linear discrete system with system matrix:

$$U_h = (I - h(I - P)A)^{-1}(I + hPA)$$
$$x_{n+1} = U_h x_n$$

This discrete linear system has to be numerically stable for a desired step size *h*. This means for the eigenvalues λ_i of U_h that:

$$|\lambda_i| \le 1 + O(h) \qquad \forall 1 \le i \le d$$

So after all, a partitioning is needed, that guarantees this relation, but treats only a minimal number of states implicitly.

In the non-linear case the differential equation can be linearized at several time-points along the trajectory and each linearization can be analysed. Then the union of all fast states will be selected as fast and the corresponding rows of the differential equation will be selected for implicit integration.

Mixed-mode and inline integration

The concept of mixed-mode integration for itself can already lead to a considerable reduction of the size of equation systems to solve. Especially, when there is only a small number of fast states and a large number of slow states.

Another common case is that there are *several* fast subsystems in a model coupled together by slow ones like in complex mechatronical systems. Then pure mixed-mode integration leads to a big implicit part, which contains all the fast components. This system has to be solved as a whole.

A big improvement in this situation is to combine mixed-mode integration with a technique called inline integration [3], which was designed to reduce the computational effort when an implicit discretization formula is used. The discretization formula is inserted (inlined) into the ODE and the size of the non-linear equation system can now be reduced by symbolic manipulations like Block-Lower-Triangular-transformation and tearing before the integration starts. That technique makes it possible to split large equation systems into smaller ones. These small systems are then much faster to solve. However, improvements achieved by this technique alone are relatively low in the case discussed here. The dimension of the remaining non-linear equation system is still too high, as the slow coupling between the fast components prevents the symbolic manipulation to split up the system.

The situation is different when mixed-mode integration and inline integration are used in combination. Then the slow part is discretized explicitly. It breaks up the coupling between the implicit components in the equation system. The decoupled systems can now be treated more easily by the symbolic manipulation. This results in several small instead of one large equation system. Sometimes the small systems can even be solved symbolically at translation time. All this leads to a drastic increase of computational speed. Obviously both methods fit together very well.

Alternative approaches

One alternative to mixed-mode integration is a technique called multi-rate integration. The main idea is to split the system into a slow and a fast part (like in the mixed-mode case) and to apply an explicit method to both parts using two different step sizes: a small one to the fast part and a big step size to the slow part.

Mixed-mode and multi-rate integration both have their own advantages and drawbacks. The most important advantage of multi-rate integration is that it doesn't contain any iterative parts during one (large) step, which means that it might even be better suited for real-time applications. The price for this is that one has to choose two step sizes and the smaller step size depends on the fastest component. This has several consequences:

- Very fast components lead to a very small step size and spoil the performance
- Integrating non-linear models, the small step size has to guarantee stability for the whole simulation, but the time-constants can vary during integration.

Another important reason for us to prefer mixed-mode integration is that it benefits largely from symbolic manipulation routines and especially from inlining. Both features are incorporated in Dymola [4].

There is also a different approach for systems with fast and slow components. It utilises the structure to solve the linear equation systems that occur at each time step quicker, using for example Krylov-space methods. This approach is described e.g. in [5, pp. 171-176]. However this class of methods is not that well suited for real-time integration, because they introduce another iterative loop (for solving the *linear* equation system) and the computational cost can vary considerably between steps.

Partitioning

Once a partitioning is made, the mixed method can be implemented quite easily in an environment like Dymola. The task is now to get a good partitioning that guarantees stability without choosing too many fast variables. We designed an automatic algorithm for this purpose which will be described briefly in this section.

Basic ideas

For a given linear differential equation system

$$\dot{x} = Ax$$

and a given step size *h* the partitioning algorithm will return a projection matrix *P* as described above such that the discrete system:

$$x_{n+1} = U_h x_n$$
$$U_h = \left(I - h(I - P)A\right)^{-1}\left(I + hPA\right)$$

is numerically stable and the implicit part is as small as possible.

This could be seen as a combinatorial problem, which could be solved by complete search. However this is not possible because of complexity reasons. Instead, heuristic criteria are used to classify variables as fast or as slow.

The algorithm is divided into two phases:

- Partitioning of the continuous system $\dot{x} = Ax$ into a fast and a slow part with the help of a user supplied threshold for the magnitude of the eigenvalues.
- Stability check of the resulting discrete system $x_{n+1} = U_h x_n$ and - if necessary - stabilisation of the discrete system.

Partitioning of the differential equation

The main task of this partitioning is to select a "fast" part so that the magnitudes of the eigenvalues in the remaining slow part of the system are below a user-supplied threshold.

The first step is quite straightforward: the fast components of the system are the eigenvectors corresponding to the fast eigenvalues. The cleanest solution would be to cut out the subspace spanned by these eigenvectors. However this is not possible in general because in the non-linear case it would involve a coordinate transformation in each step of the integration routine as the linearization *A* is time varying.

Our only freedom is to select states; that means rows of the matrix. One could select all rows that are influenced by the fast subspace. This would however lead to the choice of too many states. In most cases all states would be selected which means, that we would receive a fully implicit method again.

The best way to solve this problem is to choose only states, that are effected strongly by the fast dynamics and put them into the set of fast states. This means of course that the fast subspace is not completely removed from the slow system. Some eigenvalues of the slow system could still be larger in magnitude than the threshold. The slow system has to be checked again and the state selection is repeated, until the eigenvalues of the slow system are below the threshold.

Stability of the mixed discrete system

Obviously, there are many possible partitionings of a model, e.g. depending on the supplied threshold. All of them split up the continuous system into two parts of which the slower part has got eigenvalues below the threshold. But not all are equally well suited for a model. This has to do with the coupling between the two parts. It strongly affects the stability of the discretization, especially if the eigenvalues of the two systems are too close to each other.

We illustrate this in the case of a *2x2*-matrix:

$$A = \begin{bmatrix} S & C_S \\ C_F & F \end{bmatrix}$$

Here the components *S* and *F* are the eigenvalues of the uncoupled slow and fast system. *S* is assumed to be small and negative, *F* is large and negative. C_F and C_S are the couplings between them. They shall be assumed as small compared to *F* and *S*.

A partitioning is now performed such that the first row of *A* is classified as slow, the second as fast. The projection-matrix is therefore:

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

The matrix of the discrete system is then computed as:

$$U_h = \begin{bmatrix} 1 + hS & hC_S \\ hC_F \dfrac{1 + hS}{1 - hF} & \dfrac{1 + h^2 C_F C_S}{1 - hF} \end{bmatrix}$$

If one of the coupling elements is zero, then U_h is triangular and the eigenvalues μ_i of the combined discrete system are not different from those of the two separate discrete systems:

$$\mu_1 = 1 + hS$$
$$\mu_2 = 1/(1 - hF)$$

This system therefore inherits the stability properties of the two separate discrete systems.

If both of the coupling elements are non-zero, the eigenvalues λ_i of the discrete system change. The best insight is given for the linearization around μ_i for small $h^2 C_F C_S$:

$$\lambda_1 \doteq \mu_1\left(1 - \frac{\mu_2}{\mu_1 - \mu_2} h^2 C_F C_S\right) \qquad (1)$$

$$\lambda_2 \doteq \mu_2\left(1 + \frac{\mu_2}{\mu_1 - \mu_2} h^2 C_F C_S\right) \qquad (2)$$

Some interesting observations can easily be made regarding those formulas. First, for loose couplings the explicit part is stabilised, whereas the implicit part is destabilised. If the coupling between the two systems becomes tighter, then $C_F C_S$ gets larger,

and with it the deviation from μ_i. The other important factor is $\mu_2/(\mu_1 - \mu_2)$ which also can amplify the deviation, if μ_1 and μ_2 are too close together. On the other hand, if μ_2 is very small, that means if the fast system is very stiff, this effect is reduced.

The higher these two factors are, the worse can the stability properties of the combined system be. In some cases strong coupling can even result in a severe destabilization. Then the resulting system only allows much smaller step sizes than expected. On the other hand, if the coupling is only moderate, then the implicit system often stabilizes the combined system.

If coupling between the two parts is too strong, one can try to reduce it by selecting slow states that contribute most to it and move them into the set of fast states. This is performed iteratively in the second part of the algorithm.

Stability problems are often caused by a bad choice of the threshold; e.g. when there are two eigenvalues slightly above and below it. Then sometimes the partitioning algorithm is forced to tear subsystems in two pieces that should be kept undivided. This leads of course to a very strong coupling between the fast and the slow part. Users can avoid this problem by carefully choosing the threshold value e.g. after looking at the eigenvalues of the system.

The user's view

In this paragraph we are summing up the most important issues how to control the process of partitioning in our algorithm.

The algorithm partitions a linear system of ordinary differential equations into a fast and a slow part.

Users can supply two parameters:

- A step size, for which the integration process should remain stable. If the differential equation is linear, then stability can be guaranteed. If the system is non-linear, then in many cases stability is preserved, if the non-linearity doesn't affect the structure of the system severely.

- An optional parameter "threshold", which is an upper bound for the magnitude of the fastest eigenvalue of the explicit part. If the users don't supply "threshold", it will be calculated from the step size via a simple formula.

In general the threshold should be chosen low enough to avoid the need for stabilisation at a certain step size. The threshold is also a good method to give the algorithm knowledge about the distribution of the eigenvalues.

The output is a string array with the names of all the fast states. It can easily be inserted into the Modelica model at the top-level.

Computational results

Mixed-mode integration using the explicit and implicit Euler formulas was implemented in Dymola [4], where the method and the partitioning algorithm were tested on several Modelica [6] models. In the following, three of those test cases are described. The performance of the new method was measured and compared with the performance of the fully implicit and explicit Euler methods. All three methods were used together with inline integration, which means that symbolic manipulations could be applied to the discrete system. The models contained small algebraic loops.

Partitioning a robot model

The first model to describe is an industrial robot with six degrees of freedom. For each joint, besides the mechanical part, the electric circuit of the motor and a controller are modelled. This leads to an overall number of 78 states. The fastest eigenvalues of the linearization of the system at the starting point are about 7000 in magnitude. See Figure 1. The

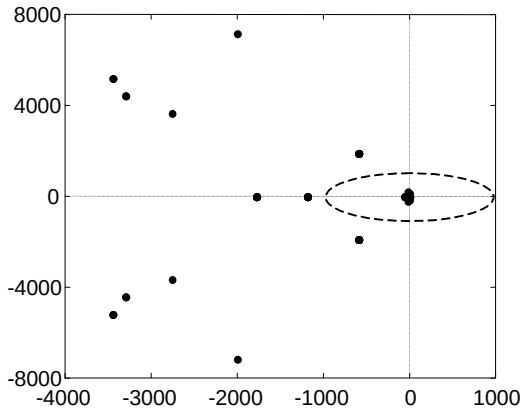

Figure 1: The eigenvalues of the robot

model was simulated for 1.5 s at a fixed given step size of 1 ms.

First the explicit Euler method was applied to the problem. To obtain stable behaviour the step size had to be reduced for about a factor ten to achieve stable behaviour. The largest (linear) equation system that had to be solved during the integration was of dimension six. This was the mass matrix of the robot. With a step size of 0.05 ms it took 16.4 sec to perform the simulation.

Implicit Euler performed the simulation run without stability problems. The execution time was shorter than the time explicit Euler needed to integrate with smaller step size but still too long. This was due to a large non-linear equation system of dimension 39 that couldn't be split up by the symbolic manipulation routines. This resulted in an overall execution time of 13 sec.

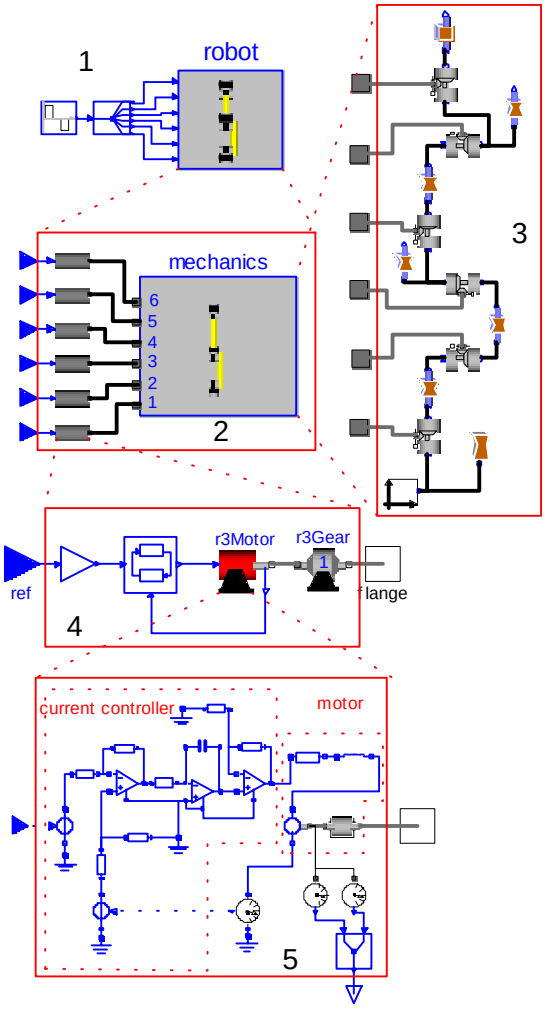

Figure 2: Model of the robot

Now the algorithm we described in the last section was used to partition the robot. The maximum magnitude of eigenvalues allowed for the slow system was chosen as 100. The partitioning algorithm chose 45 fast states, which were representing the electric circuits and the fast parts of the controller of each joint. Slow variables were mainly in the mechanical part or pure integrators. Although a relatively large number of states had to be treated implicitly, the execution time was much shorter than in the fully implicit case. This was because the symbolic manipulation routines could now split up the large non-linear equation system that had to be solved using fully implicit discretization. It was split into a couple of small linear systems - two for each joint drive - that could be solved symbolically at compile time and one linear system of dimension six to be solved during integration. This was the mass matrix again: the same as in the explicit case. Mixed-mode integration managed to simulate the model in 1 sec.

Obviously the weak mechanical coupling between the fast components of the robot was responsible for the large equation system of implicit Euler. The partitioning algorithm together with the symbolic manipulation detected and broke up this weak coupling. This led to speedup factors of 16 compared to explicit Euler and 13 compared to implicit Euler.

To judge the quality of the computed solution a reference trajectory was created using a high accuracy method (DASSL; tol =1e-6) and compared to the mixed-mode solution. As one can expect the accuracy of the mixed-mode solution wasn't very high but fairly good. The errors at the position level were around 3mm (that is about 2e-3 x the dimensions of the robot) and on velocity level around 0.2 m/s (that is about 8e-3 x the maximum speeds). For the robot model mixed-mode integration performed better than DASSL even when the overall integration time was considered. DASSL needed 10.5 s to simulate the model using a low tolerance of 1e-2. This suggests the possibility of using the mixed-mode approach also for offline-integration, if a partitioning can be reused for several simulation runs.

Partitioning an engine model

Runtime comparisons have also been made simulating a model of a drive train. It consists of a diesel engine, its air-supply and a modelled load.

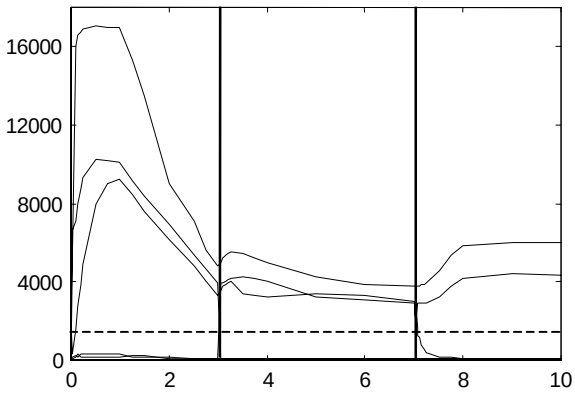

Figure 3: Magnitudes of eigenvalues changing during execution time

Together there were 26 states. Most of them were used to model the air-supply. An eigenvalue analysis showed that three eigenvalues were much faster than the others were, namely about factor 20 (see Figure 3). The model was simulated for 10 seconds and a step size of at least 1 ms should be used.

First explicit Euler was used. The largest non-linear equation system for it to solve had dimension 1. Due to the stiffness of the system the integration failed at a step size of 1ms. Reasonable results were obtained at a step size of 0.1 ms, which corresponds to the observation, that the fastest Eigenvalues were 2e5 in magnitude. In this case the execution time was 23 sec.

Implicit Euler had no problems with stability running at a step size of 1 ms. The drawback was, that it had to solve a non-linear equation system of size 17, which increased the computational cost per step considerably. The overall execution time was now 21 sec.

Using mixed-mode integration with a threshold of 1000, 4 fast variables were selected. The fastest Eigenvalues of the slow system therefore had a size of 500 in magnitude and furthermore the discrete system was stabilised by the implicit part (which is not always the case). This enabled the simulation to run stable at a step size of 1 ms and in each step a non-linear equation system of dimension 3 had to be solved. The overall execution time could be reduced to 5.4 sec, which is a speedup factor of 4 compared to implicit Euler and factor 4.5 compared to explicit Euler.

	Robot	**Engine**
No. of states	78	26
Simulated Time	1.5 s	10 s
Sample Time	1 ms	1 ms
Explicit Euler		
Overall Time	16.4 s	23 s
Stable Step size	0.05 ms	0.1 ms
Size of Eq. Syst.	6x6	1x1
Implicit Euler		
Overall time	13 s	21 s
Size of Eq. Syst.	39x39	17x17
Mixed-mode Integration		
Overall Time	1 s	5.4 s
Size of Eq. Syst.	6x6	3x3

Table 1: Performance of the three methods

Conclusions and Outlook

Mixed-mode combined with inline integration provides a good alternative to the classical explicit and implicit methods. Especially real-time integration of multi-domain models can benefit considerably. The main strength of mixed-mode integration lies in its increased flexibility and in the additional information the integration routine uses. Stability is preserved, but the equation systems to be solved are much smaller compared to an implicit method.

The designed selection routine automatically chooses states, so that the stability of the linear system for a given step size is guaranteed. However it is still important for users to know about the properties of their model if they want to achieve the best results.

One future extension to this concept could be the introduction of a third class of states: oscillating states, which could be treated using the trapezoidal rule. In many cases mixed-mode integration can also accelerate off-line simulation especially when large models with few fast components are considered. In this case mixed-mode integration should be performed using partitioned methods of higher order.

Acknowledgements

This work was in parts supported by the European Commission under contract IST-199-11979 with DLR and Dynasim AB under the Information Societies Technology as the project entitled "Real-time simulation for design of multi-physics systems".

The handling of mixed-mode integration was added to Dymola by Sven Erik Mattsson.

Anton Schiela would like to thank Martin Otter, DLR and Hilding Elmqvist, Dynasim for support and encouragement.

References

[1] H. Olsson, G. Söderlind. *Stage value predictors and efficient Newton iterations in implicit Runge-Kutta methods.* SIAM Journal on Scientific Computing 20(1):185-202, 1999.

[2] E. Hairer, S. P. Nørsett, G. Wanner *Solving Ordinary Differential Equations I. Nonstiff Problems.* Springer Series in Computational Mathematics, 2nd edition, 1992.

[3] H. Elmqvist, F. Cellier, M. Otter *Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems*. Proceedings: European Simulation Multiconference June 1995 Prague, pp: XXIII-XXXIV.

[4] Dymola. Dynasim AB, Lund, Sweden. Homepage: http://www.dynasim.se.

[5] E. Hairer G. Wanner *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.* Springer Series in Computational Mathematics, 1st edition, 1991.

[6] Modelica. *A unified object-oriented language for physical systems modelling.* Modelica homepage: http://www.Modelica.org, 2000.