AMDiS - Adaptive multidimensional simulations: Towards component based adaptive finite element software

Simon Vey
vey@caesar.de

Axel Voigt
voigt@caesar.de

Research Center Caesar, Crystal Growth Group
Ludwig-Erhard-Allee 2, 53175 Bonn

Abstract

In this paper we describe the concepts of our finite element toolbox AMDiS which is a C++ library written in an object oriented manner. Furthermore we introduce the Shared Mesh Interface, that enables the distributed management of unstructured meshes and CaST, which is an approach for component based adaptive simulations.

1 Introduction

In the field of scientific computing continuesly new technologies are developed which enables the user to solve more and more complex problems. On the other hand these increasing possibilities often lead to a more complex software handling, which quickly can neutralize the achieved benefits. Therefore modern simulation software should provide a high level of abstraction, keeping numerical issues away from the user, without loosing generality, extensibility, maintainability or efficiency. To reach these often contradicting goals, well proven design patterns and software techniques are applied to our finite element toolbox AMDiS. As the name implies, adaptivity and multidimensionality are two of the most important properties. Other main features of AMDiS are the coupling of different problems of maybe different dimensions within one simulation code, the solution of systems of coupled PDEs, the implementation of parametric elements, which allows to compute on arbitrary manifolds, and the implementation of composite finite elements ([SVV05]), which can resolve complex domains without the need of an explicit triangulation of these domains.
In section 2 we describe some implementation aspects of AMDiS, including the adaption loop as highest abstraction level. To be able to exchange mesh data including the simulation results with other software tools, we developed the *Shared Mesh Interface* (SMI), which provides an abstract and application independent interface for unstructured meshes and a client-server environment which allows to exchange data even between applications running on different machines. SMI is described in section 3. Finally we describe in section 4 an approach for a component based simulation framework (CaST) supporting adaptivity as well as dynamic and distributed module instantiation.

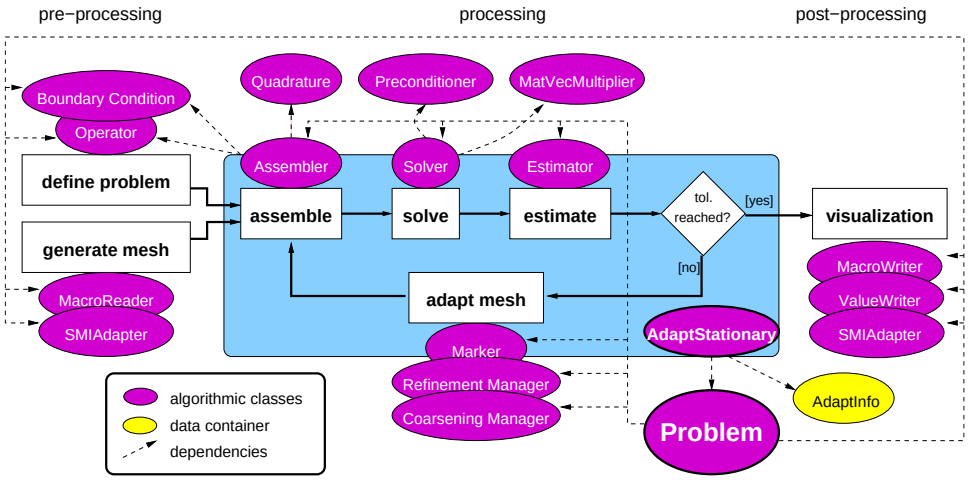

Figure 1: Adaption loop for a stationary problem and the needed software components

2 AMDiS implementation aspects

AMDiS is written as an object oriented library in the programming language C++. The main development goals was to implement a software which is able to solve a wide range of problem classes and on the other hand provides a high level of abstraction, keeping numerical issues away from the user. Furthermore modern simulation techniques should be usable in a flexible, extensible and efficient way. A modular design using modern programming techniques and well proven object oriented design patterns helps to achieve these often contradicting goals. In Figure 1 the main AMDiS components with the adaption loop as highest abstraction level are shown.

2.1 Adaption loop

The adaption loop builds the highest abstraction level in the simulation. Here the decisions are made, when a problem has to assemble its equation system, when to solve the system, when to estimate the error indicators and when to adapt the underlying mesh. In order to keep the implementation of these single steps transparent at adaption loop level, it is delegated to a *ProblemIterationInterface* and a *ProblemTimeInterface*. The implementation of the iteration interface knows what to do for one adapt iteration. In the standard case for one single problem, the equation system must be assembled and solved, local errors have to be estimated and the mesh has to be adapted. In the case of several coupled problems this iteration can become more complex. The time interface is responsible for all aspects concerning time dependency. The problems are known to the adaption loop only by this simple interfaces. This allows a very easy and straightforward formulation of the adaption loop by using the interface instances as black box components, what in turn leads to a high

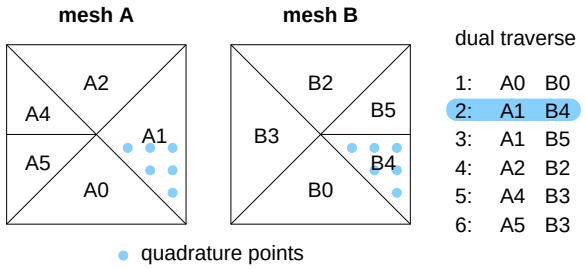

Figure 2: Dual traverse of two independently refined meshes

reusability of the adaption loop for different problem types. Beside the problem the current adaption state, which is stored in an *AdaptInfo* object, must be known to the adaption loop. Here informations about the current simulation time, the current timestep size and the current iteration number are stored and whether the desired tolerances or the maximal iteration numbers are reached. In Algorithm 1 we show an adaption loop of a time dependent problem solved with an explicit time strategy. Before the time loop starts an initial solution must be calculated for the start time.

Algorithm 1 Explicit time strategy

```
adaptInfo→time = adaptInfo→startTime;
timeInterface→solveInitialProblem();
iterationInterface→oneIteration(adaptInfo, ESTIMATE);
while adaptInfo→time < adaptInfo→endTime do
  adaptInfo→time += adaptInfo→timestep;
  timeInterface→initTimestep();
  timeInterface→setTime(adaptInfo);
  iterationInterface→beginIteration(adaptInfo);
  iterationInterface→oneIteration(adaptInfo, FULL_ITERATION);
  iterationInterface→endIteration(adaptInfo);
  timeInterface→closeTimestep();
end while
```

Note that no assumptions about the interface implementations are made in the adaption loop. The calculation of the initial solution is delegated to the time interface, whose implementation also is transparent for the calling adaption loop.

2.2 Systems of coupled PDEs

In AMDiS it is very easy to solve systems of coupled PDEs in one equation system. To illustrate this possibility we explain it on the easy example problem

$$-\Delta u \quad = \quad f$$

$$u - v \quad = \quad 0$$

To define this problem we formulate a matrix of operators for the left hand side and a vector of operators for the right hand side of the equation system which results in the following equation system after assemblage:

$$\begin{pmatrix} L_{-\Delta u} & 0 \\ L_u & L_{-v} \end{pmatrix} \begin{pmatrix} u_h \\ v_h \end{pmatrix} = \begin{pmatrix} f_h \\ 0 \end{pmatrix}$$

Let n be the dimension of the finite element space of u_h and m that of v_h. Then $L_{-\Delta u} \in \mathbb{R}^{n \times n}$, $L_{-v} \in \mathbb{R}^{m \times m}$, $L_u \in \mathbb{R}^{m \times n}$, $u_h \in \mathbb{R}^n$, $v_h \in \mathbb{R}^m$ and $f_h \in \mathbb{R}^n$. So we have an equation system consisting of a matrix of DOF matrices and two vectors of DOF vectors. In the general case of l coupled PDEs we have a matrix of $l \times l$ operators where the entry (i, j) contains the operator which couples the i-th equation of the system with the j-th variable.
If equation i and equation j live on the same finite element space the assemblage can be done in a standard way. If the mesh is the same but the basis functions differ, the finite element space of equation i must be considered as *row space* and that of equation j as *column space* in the assembling routines. If even different meshes are used for the different components, for one element S of the mesh of component i all elements of the j-th components mesh must be considered, that have an overlap with element S.
Similar to the method described in [Sch03] in AMDiS meshes of different components share one initial triangulation but can be adapted independently of each other. The assemblage then is done within a parallel traverse of the two involved meshes (*dual traverse*), where each traverse step returns one element of each mesh. If a leaf element of one mesh has further refinements in the other mesh, it is returned in several dual traverse steps, until all corresponding leaf elements of the other mesh was traversed as well. The integration now is done over the smaller element using a parameterized quadrature for the basis functions of the bigger one. Figure 2 shows an example of a dual traverse for two triangular meshes.
The solution of the resulting equation system, error estimation and element marking can be deduced from the scalar case in a general way.

3 Shared Mesh Interface (SMI)

The goal of SMI is to provide an unified and distributed management for arbitrary meshes. It is not part of AMDiS but can be used by any set of applications which want to share one or more meshes. In AMDiS SMI is used to couple the simulation with a specialized visualization software in the post-processing step and to allow the integration of external mesh generators in the pre-processing step. Furthermore SMI can be used to turn the hierarchical mesh structure of AMDiS into a flat representation of the mesh, which allows a flexible iteration over elements and nodes, what is useful for some algorithms.
On the one hand SMI provides an abstract interface which can handle with any kind of unstructured meshes consisting of arbitrary and even mixed element types, on the other hand the client-server architecture of SMI allows to share meshes between different application running at even different computers.

4 Component based adaptive simulation

Numerical simulation software naturally decomposes into several building blocks. In the pre-processing step the problem domain must be defined, a discretization of this domain must be created, and problem operators and functions must be given. In the post-processing step the simulation result may be given to a visualization software or other post-processing tools. The processing step in the adaptive case typically consists from the assemblage of a equation system, the solution of this system, a global and local error estimation and a mesh adaption based on the local estimates.
The goal of a component based simulation approach is to define uniform interfaces for each of these tasks and to provide a framework which allows an easy and flexible interchange of the single modules.
ORCAN (Open Reflective Component Architecture, [ORC]) is such a framework including its own component management which allows component instantiation dynamically at runtime. Each interface method invocation is done by a virtual function call which causes a certain time overhead. This is unproblematic if such interface calls are done at a high abstraction level outside the inner simulation loops. But adaptive mesh refinement, which is not supported by the ORCAN interfaces, needs such an inner loop interface.
DUNE (Distributed and Unified Numerics Environment, [BDE+05]) focuses on interfaces of fine granularity supporting adaptivity. One of the main aspects is the unified access of hierarchical meshes. Here the interfaces are realized by static polymorphism using the template mechanisms of C++. The main drawback of this approach is the fact, that all needed modules have to be linked together in one user application, so no dynamic component exchange is possible. Every time a new module should be included, the source code of this module has to be compiled together with all other modules of the application.
So our approach is the attempt to combine the two ideas of ORCAN and DUNE using the CORBA component model (CCM [CCM02]) as component management, which is a widespread platform and language independent standard supporting even distributed components. To avoid slow mesh accesses in inner computation loops, local mesh representations are stored where necessary. The local mesh access then is done through a DUNE like template based interface. Using this interface the components also can be statically linked together, if no component management is needed. To avoid a complete mesh transfer after each mesh change, SMI is used as shared mesh management. So only local mesh changes must be communicated, which drastically reduces the needed communication overhead. So far CaST (Component based Adaptive Simulation Technology) is just a coarse concept which is illustrated in Figure 3.

5 Conclusion

In materials science as well as in other areas different problems occur with different requirements to the simulation software. On the other hand from an abstract point of view often the same building blocks are needed to implement the application. Therfore the strict separation of abstraction levels in AMDiS helps to produce reusable and maintainable code. In

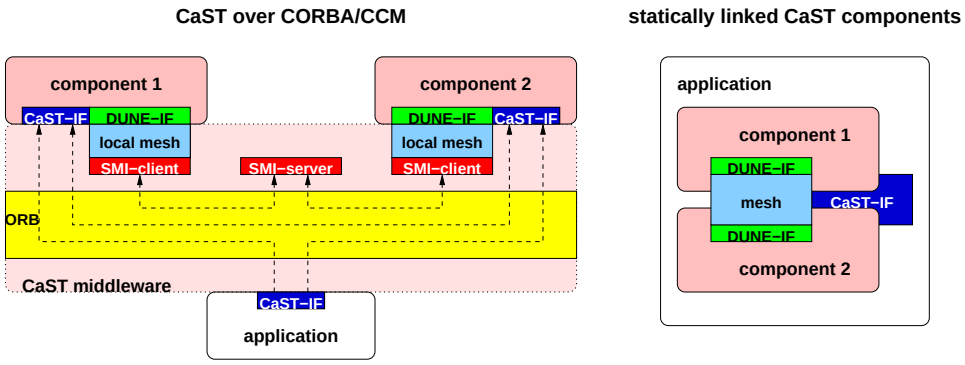

Figure 3: CaST components used with and without the CaST middleware

the future component based frameworks including unified module interfaces can improve the productivity of simulation software development drastically. With CaST we introduced a concept for such a framework supporting adaptivity as well as dynamic and distributed module access.

References

[BDE+05] P. Bastian, M. Droske, C. Engwer, R. Klöfkorn, T. Neubauer, M. Ohlberger, and M. Rumpf. Towards a unified framework for scientific computing. In R. Kornhuber, R.H.W. Hoppe, D.E. Keyes, J. Periaux, O. Pironneau, and J. Xu, editors, *Proceedings of the 15th Conference on Domain Decomposition Methods*, LNCSE. Springer-Verlag, 2005. accepted for publication.

[CCM02] Corba component model. Technical report, Object Management Group, http://www.omg.org/technology/documents/formal/components.htm, 2002.

[ORC] Orcan - open reflective component architecture, http://www.cgl-erlangen.com.

[Sch03] A. Schmidt. A multi-mesh finite element method for phase-field simulations. volume 32 of *LNCSE*, pages 209–217, 2003.

[SVV05] C. Stöcker, S. Vey, and A. Voigt. AMDiS-adaptive multidimensional simulation: composite finite elements and signed distance functions. *WSEAS Trans. Circ. Syst.*, 4:111–116, 2005.