ORCAN: A platform for complex parallel simulation software

Jan Treibig, Silke Bergler, Ulrich Rüde

Lehrstuhl für Systemsimulation, Institut für Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract: The Open Reflective Component Architecture (ORCAN) is a component based software platform for Simulation software. It enables to build applications out of runtime exchangeable components. One purpose of this framework is to introduce advanced software design techniques in simulation software for a longterm distributed development process. The other benefit of this framework is a predefined set of components for standard simulation purposes, which can be seen as a suggestion for a simulation middleware platform. In this paper we will present the basic design of ORCAN. As an example application we have implemented a parallel heat equation solver.

1 Introduction

Simulation software is a multi-disciplinary effort. Developing simulation software therefore is expensive and time consuming. Besides the physical and numerical issues, computer science is of great importance. Parallel simulation software on large HPC systems is mainly driven by public research projects. The people who write this software are physicists, engineers or mathematicians. Computer science aspects are often neglected in this field. As a consequence research codes often are large monolithic programs, which are very hard to maintain or extend. The painful decision to stop development of a code and dump the know how and work of many man years can be the end of the story in many cases. Software design techniques, which ease the maintainability and extensibility, are available. These techniques speed up the development process and force a better code quality. These points are especially true if many developers implement an application. This is one motivation to use software design techniques in simulation codes. Two other issues get more and more important today: First the reuse of existing code for different parts of the simulation chain and second coupling of codes for multiphysics simulations. Many parts of a simulation program are unspecific, e.g. geometry treatment, grid generation, linear equation solvers, visualization, grid management and file IO. Its inefficient to program all parts on your own. Often external programs are used for common tasks. This has the disadvantage, that you depend on the functionality and restrictions of a external program. Moreover such a solution is not as user friendly as a well integrated solution with a single GUI, where everything is tailored to the application. Especially the accep-
tance in industry often depends on the existence of an easy to use graphical user interface. Another possibility is to use external libraries, enabling an integrated solution. But often these libraries are very complex to use and require large skills in software development. Another disadvantage of external libraries is, that you make your code dependent on a single library for a specific task. The computational power of today's HPC systems makes it possible to simulate global geometries with multiple physical phenomena involved. Often you don't have the know how or time to develop things like an fluid or radiation solver, although your field of interest would require that. Of course there are a lot of commercial and open source solutions available, but the coupling and usage is often complex and limited to specific cases. While there are efforts to address these problems we see a growing demand for a powerful integrated software platform tailored to the specific needs of simulation. We propose an attempt to solve many of the above issues with the so called ORCAN (Open Reflective component architecture) framework. The main benefit apart other things is its role as a middleware platform for simulation enabling a professional development process.

2 The Software Architecture

We formulate the following requirements for the framework:

- The framework should enable to split up complex software systems in small maintainable parts, so called modules
- The modules should be exchangeable even at run time
- Different incarnations of a module should be available to the application
- The framework should enforce and provide clear definitions of module interfaces
- The module handling is hided from the end user
- The framework should be easy to use and no detailed knowledge about the implementation should be necessary
- It should be portable to different operating systems
- The rapid development of graphical user interfaces should be supported by the framework

To meet these requirements we decided to choose a component based software architecture [Szy02]. Components are elementary building units of an application. They represent objects with a specific functionality, e.g. volume mesh, linear equation solver or visualization. In distinction to an object oriented paradigm it provides its functionality not through a list of methods, but with a number of indirect interfaces. An Interface itself is made up of a list of semantically connected methods. All interfaces together provide the actual functionality of a component. A component includes no implementation, but only specifies
through its interfaces the access methods and expected behavior. A realization provides the actual implementation of a component. There can be an arbitrary number of realizations for a single component. Each realization implements a subset of the interfaces, but not necessarily all interfaces of a component. Figure 1 illustrates the component realization relationship.

![Diagram](image)

Figure 1: Component X has 4 Interface. The realization derives polymorphic from the component and three interfaces. Interface X3 is not implemented and therefore not available to the application.

To satisfy the concept of exchangeable components, there was a special focus on the management of the components and their realizations. Because the ORCAN based application should be independent of a concrete realization of a component, the objects cannot be generated the standard C++ way. The instantiation is therefore done by the class Object Factory. This pure virtual class delegates the implementation to specific versions e.g. for loading dynamic libraries or getting an instance of an object from an CORBA server. Figure 2 shows, which role the object server has, in order to provide a realization of a component to the application. Well known design patterns introduced in [GHJJ97] are applied here. The configuration of the object server is realized through an XML file. The user or an frontend can specify which realization for a component is loaded from which object server. The syntax of the XML file is designed in such a way, that entries can be also specified operating system specific.

Often different realizations of a component have a number of specific parameters, which cannot be treated with the interface concept in a sensible and efficient way. For this purpose ORCAN components are reflective, which means that a realization can dynamically be asked which parameters with which type it has. For this purpose every component has an so called PropertyMap, illustrated in figure 3. Through this interface all realization specific parameters can be requested dynamically at runtime. The abilities of the property map allow to define all types of data and even methods as parameter. The access to the data is associative and typesafe. In addition it is possible to add attributes to parameters, e.g. to specify value ranges, and to define rules for changing the value of an parameter depending on another parameter.
3 Implementation

The framework is entirely implemented in C++ and does not depend on any external library. It consists of four base libraries:

- **ORCAN** The base library enabling the definition of components and providing the management of the module mechanism
- **ORCAN/Sim** A collection of predefined components specific for simulation applications
- **ORCAN/SimTools** A collection of ready to use solutions for a more efficient handling of ORCAN/Sim components
- **ORCAN/Wx** Automatic GUI generation controlled by XML specifications based on wxWidgets [WXH]

All four libraries are ported to the operating systems Linux, Windows and MacOSX. The code is open source and available under the GPL license [ORC]. Figure 4 shows the basic structure of a typical application developed on top of ORCAN.
While the framework is not specific for parallel software it is especially useful in this context. Beyond the advantages described above the flexibility of the framework enables it to easily adopt an application to new parallel architectures.

The critical and most important issue is to specify simple and still powerful and flexible interfaces for the modules. While we tried to specify sensible interfaces only the trial of many users can lead to a good solution. This can be seen as a middleware solution for simulation. As an example we want to explain the interface to an unstructured or structured volume mesh, which has proved to be very flexible and useful. Of course there can be additional interfaces for e.g. a more efficient access to a structured mesh.

The unstructured mesh is organized in an index based fashion, in which each element and vertex has an unique id. Elements are defined by its type and a list of their vertex ids. To store data on the mesh entities an attribute system is used. The attributes are distinguished with regard to where the data is stored topologically. Data can be stored for example per element, per vertex, per edge, per face or for the whole mesh. Even the coordinate of a vertex is an optional attribute resulting in a very lightweight base mesh. It is possible to create or delete attributes any time at program execution, enabling the mesh data to adjust to the specific needs. It is also possible to store data only on a subset of the mesh.
How attractive it is to use the framework depends on how many component realizations are available. At the moment there exist realizations for all parts necessary to write a full scale simulation application for a heat conduction radiation problem. There are different realizations for unstructured surface and volume meshes. Among those is a parallel Volume mesh based on Parmetis [KSK03] and PETSc [BKG+01] data structures. For preprocessing of the geometry there is a geometry module based on OpenCASCADE [OCC]. Generation of unstructured grids is enabled through realizations based on Gmsh [Gms], GRUMMP [GRU] and Tetgen [Tet]. Powerful interactive visualization is available through modules based on VTK [VTK] and OpenGL [Ope]. There are different serial and parallel realizations for linear equation solvers based on PETSc and Laspack [Las]. For the solution of the stationary and instationary heat equation discretization modules with a finite element discretization are available. Among the components is also a material database for a central representation of material parameters and a wide range of file writers and readers for various mesh formats. Figure 5 gives an example of an application based on ORCAN. You can see the GUI based on the automatic GUI generation with XML rules and the Visualization component. This application computes temperature distributions in high temperature furnaces for crystal growth. These furnaces incorporate large domains with complicated geometries. The solver considers heat transport through heat conduction and radiation. It is at the moment still serial, but in the next section we will present a pilot implementation of an parallel heat equation solver based on ORCAN, which is a first step towards a complete parallel implementation.

Figure 5: Example application simulating a high temperature furnace

300
Example Parallel Application

In a first parallel pilot application we implemented a finite element based instationary heat equation solver operating on unstructured grids. Figure 6 shows the parallel components involved. The application is distributed memory parallel based on MPI [For95]. An unstructured mesh is loaded and partitioned fully parallel, the actual partitioning is realized using the external ParMETIS library. A parallel mesh management component cares about the different numberings and ghost nodes in a transparent way. Based on the parallel mesh a finite element discretization was realized, the resulting linear equation system is solved using a module based on PETSc. Again the solver could be exchanged against a different solver package without changing one line in the application code. At the moment the results need to be serialized for visualization, but there is on going work for parallel visualization using the capabilities of VTK. This new realization will enable a fully parallel simulation tool chain.

![Figure 6: Overview about the parallel components](image)

Parallel Results

We present parallel results, proving that the approach works and shows good parallel performance. All experiments were done on an AMD Quad Opteron Cluster with Infiniband interconnect. Figure 7 left shows results from the parallel reading of the mesh file. Each process reads only its part of the mesh. All scaleups were done with unit cube tetrahedron meshes with number of elements ranging from 4 326 783 to 104 607 744 and 695 280 to 17 841 624 unknowns.
In figure 8 the results for all parts of the simulation are shown. The code scales quite well up to 29 CPUs (8 left), tests with more CPUs have still to be done.

A few words to the often doubted performance of a sophisticated software approach. We haven’t done detailed benchmarks against a specialized monolithic solution. Of course you have to pay for the added functionality, the question is if this is negligible or not. The only penalty of our approach are the indirect function calls to abstract methods located in shared libraries. While this of course is costly, the actual penalty depends on how often you have to call these methods. For example for the visualization component you have no penalty at all, because it is one or two calls to initialize the canvas. On the other side if you go over a mesh and call 10 abstract methods for each element, of course you have to face a big penalty against a more efficient solution. The point is you can decide. If performance is not that critical or you need the added functionality you can implement it entirely in ORCAN module interfaces. If performance is an issue you could encapsulate
your code in a module, and only uses e.g. the pre- and postprocessing capabilities of the ORCAN framework. Because a component consists of a collection of interfaces, there can be specialized interfaces for efficient handling of e.g. structured meshes. The framework does not force you to anything. You can only use the parts you need or you can use different realizations, which are more efficient for your case. Of course comparisons against standard solutions have to be done to get an objective measure for the overhead.

4 Related work

There are a number of other projects which aim to solve the same problems. The following section tries to give an overview of two of them and to explain the differences to ORCAN. PETSc [BKG+01] stands for Portable, Extensible Toolkit for Scientific computation. It has a suite of data structures and routines for the (parallel) solution of large linear equation systems. One of its advantages is that it acts as a wrapper for many other solvers. Future plans are to extend PETSc to include also higher level functionality to solve partial differential equations. Its strengths are a very good collection of high performance parallel linear equation solvers and parallel data structures. It is not really comparable to ORCAN because its focus is not to act as an middleware for simulation, but to provide a integrated solution for the efficient solution of PDEs. In our opinion it does not address the software design aspects of large simulation software. It does not incorporate a powerful module mechanism as ORCAN. Many parts of the simulation tool chain are missing, e.g. Visualization, GUI support, geometric modeling and discretization models. Salome [sal] is in many aspects similar to ORCAN. It is also based on a component architecture. Its focus lies on providing a rich functionality for pre- and postprocessing of data for numerical computations. It is also suited for multi-physics coupling between simulation software.

5 Conclusion and Outlook

The ORCAN framework has already shown to be very useful as a platform for parallel simulation software. Especially for the case, where many people develop a single application the advantages are obvious. It is intended to be used as a community driven platform, where everybody can contribute its ideas. The reuse of existing and coupling of different codes is eased. The development process is accelerated, one can concentrate on its special competences while still maintaining a overall high quality of an application. The main motivation is to create a middleware platform for simulation. The often criticized performance penalty of sophisticated software design solutions can be avoided with a sensible use of the framework.

Many issues mentioned in the introduction are still not fulfilled in the parallel pilot application. It is a command line program producing an serialized mesh which is after that
loaded in a separate ORCAN based visualization program. Future work includes to have a central GUI for parameter adjustments, geometric preprocessing and (parallel) mesh generation. Also the parallel environment, at the moment MPI [For95], should be controlled by that GUI. After you started the parallel application, as a separate process, it should be possible to detach and again attach to the running parallel program, in order to control program execution. All this functionality is of course planned to be made available to other applications through an ORCAN component.

References


304