Open Source, component based simulation software development using Orcan

Horst Hadler  
University Erlangen-Nuernberg,  
Department of Computer Science 9

Thomas Jung  
Fraunhofer Institute of Integrated Systems and Device Technology, Erlangen (IISB)

Michael Kellner  
University Erlangen-Nuernberg,  
Department of Materials Science 6

Crystal Growth Lab (Prof. G. Mueller):  
www.cgl-erlangen.com

Institute for Material Sciences VI,  
University Erlangen-Nuremberg

Fraunhofer-Institute IISB, Erlangen

- Development and analysis of crystal growth processes (Si, InP, GaAs, CaF2, oxide crystals, ...)
- Experiments + Simulation
- CrysVUn: TMT for the user support program of the MSL
Background of ORCAN development:

- need to be able to simulate radiative heat transfer in complex 3D geometries – including volume effects (absorption, scattering), and different surface effects (diffuse/ specular reflection, ...), e.g. for optical crystals, szintillators, ...

Options:

- **Use a commercial CFD-code?**
  - +? most is already done... (however, radiation models are weak)
  - - you don’t know what is going on behind the scenes
  - - difficult to extend
  - - expensive, take a lot of learning time: have to select one, and stick to it

- **Develop your own proprietary code?**
  - - beyond our possibilities: we cannot do everything needed

- **Use the wealth of existing open source tools: mesh generators, solvers, geometry handling, visualizations, ...**
  - + its free ..
  - + source code available -> possibility to check what is really done, and to extend the code
  - - no common interfaces: some common software infrastructure needed
  - ? how to finance yourself?
  - - no guaranteed support
Decision:
Create a framework, which hopefully can serve as basis for our future software development:

- separate the task into clearly distinguished components
- easily exchange these components, even on runtime
- allow independent development of components at different sites by clear interface specifications
- no intrinsic dependency on other packages (just C++), platform independent
- supports and simplifies creation of GUI’s
- Open Source, in the hope to initiate an exchange of components with other working groups

Open Reflective Component Architecture

- ORCAN: application-independent component management: ObjectServer handles component creation and deletion
- Orcan/Sim: a set of component and interface specifications specific for CFD-style simulation applications
- Orcan/Wx: a library based on WxWidgets, allows automatic creation of GUI elements based on XML-descriptions, exploiting the reflexivity of Orcan components
- Orcan/SimTools: frequently used tools, e.g. polygon class, parsers
**what is a "component"?**

![Diagram of Component X and Realization X](image)

- **e.g. Mesh component**
  - **mandatory interface:** "IMeshCreate" -> functions to build a mesh
  - **optional interface:** "IMeshChange" -> functions to modify (e.g. refine) an existing mesh

**Reflexivity**

via an associated "PropertyMap", each component realization can be queried for implemented interfaces and specific parameters

Each "Property" has associated "Resources" and "Rules", which are specified in a realization-specific XML file
GUI generation

some currently existing components and realizations:

**Geometry**
read, e.g. from CAD, perform shape healing, deliver a polygonal approximation

- **OCCGeometry**: a realization based on OpenCASCADE

**Surf/VolMesh**
surface and volume meshes, dynamic attributes

- **SimpleSurfMesh / SimpleVolMesh**: own implementations
- **VtkUGrid**: using the VTK mesh implementation

**VolMeshGen**
volume mesh generation from surface mesh as input

- **VolMeshGenTetgen**: using Tetgen
- **VolMeshGenGmsh**: using Gmsh

**Surf/VolMesh Reader/Writer**
several formats already available, list is constantly expanding

**Visualization**
Visualization of scalar or vector attributes on surface or volume meshes

- **VtkMeshVisualization**: using the Visualization Toolkit (VTK)
components and realizations, continued

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PDEDiscretizer</strong></td>
<td>Discretization of partial differential equations, input: VolMesh and LESSolver</td>
</tr>
<tr>
<td><strong>FEMLaplace</strong></td>
<td>discretization of temperature equation using finite element method by J. Haerdtlein, Computer Science 10, University Erlangen/Nuernberg</td>
</tr>
<tr>
<td><strong>LESSolver</strong></td>
<td>linear equation solver, offers an interface to build and fill a matrix, and to solve the system <strong>LaspackSparseSolver</strong>: uses the Laspack library</td>
</tr>
<tr>
<td><strong>MeshCoupling</strong></td>
<td>functionality to extract sub-meshes, interpolate/transfer data between different meshes <strong>SimpleMeshInterpolator</strong> (own implementation) <strong>SimpleMeshCoupling</strong> (own implementation)</td>
</tr>
</tbody>
</table>

components and realizations, continued

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PhotonMapper</strong></td>
<td>An efficient ray/tracing based Monte Carlo Method for thermal radiation, capable to take into account basically any effect (different emission/reflection models, absorption, scattering, ....)</td>
</tr>
<tr>
<td><strong>GPURad</strong></td>
<td>Radiosity computation with hierarchical clustering and hardware-assisted (GPU) viewfactor computation</td>
</tr>
</tbody>
</table>
Getting component realizations:

```cpp
c::VolMeshRef mesh = c::VolMesh::New()
c::VolMeshRef mesh = c::VolMesh::New("SpecificImplementation")
```

or: query available implementations, and select one which implements a required interface

Using interfaces:

```cpp
if(mesh.I.TopologyPtr) {
    mesh.I.TopologyPtr->GetNeighbourElements(...)
}
```

accessing parameters:

```cpp
c::PropertyMap::iterator r = mesh.GetProperties().begin()
-> iterate over properties, get names and types, modify
```

a minimalistic, but working, application (showing Commands):

```cpp
#include <ocs/SurfMesh.hh>
#include <ocs/SurfMeshWriter.hh>

int main(int argc, char** argv) {
    c::SurfMeshRef surfmesh= c::SurfMesh::New();
    c::File infile("inputfile");
    c::SurfMeshReaderRef reader = c::SurfMeshReader::New("ocsf::SurfMeshBinaryReader");
    reader.SetInput(infile);
    reader.SetOutput(surfmesh);
    reader.Execute();
    
    c::SurfMeshWriterRef writer = c::SurfMeshWriter::New("ocsf::SurfMeshUnvWriter");
    writer.SetInput(surfmesh);
    writer.SetOutput("outputfile");
    writer.Execute();

    return 0;
```
SimTest

an example- and component testing application

colorful pictures: coupled conduction + radiation
currently used external components/tools/libraries:

- **OpenCASCADE** ([www.opencascade.com](http://www.opencascade.com)) : Geometry handling, shape healing, Cad import
- **Vtk (Visualization Toolkit, [www.kitware.com](http://www.kitware.com))** : Visualization of mesh data
- **Laspack** ([www.tu-dresden.de/mwism/skalicky/laspack/laspack.html](http://www.tu-dresden.de/mwism/skalicky/laspack/laspack.html)) : sparse matrices, LES solver
- **Finite Element discretization from LSS 10**
- **WxWidgets** ([www.wxwidgets.com](http://www.wxwidgets.com)) : platform-independent GUI-toolkit

grid generators:

- **Gmsh** ([www.geuz.org/gmsh](http://www.geuz.org/gmsh))
- **Tetgen** ([tetgen.berlios.de](http://tetgen.berlios.de))
- **Netgen** ([www.hpfem.jku.at/netgen/](http://www.hpfem.jku.at/netgen/))
Work in progress:

- generic coupling with other solvers

Block-Newton coupling:

\[
\begin{pmatrix}
D_x f(x, y) & D_y f(x, y) \\
D_x g(x, y) & D_y g(x, y)
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta y
\end{pmatrix}
= \begin{pmatrix}
f(x, y) \\
g(x, y)
\end{pmatrix}
\]

\[
f(x, y) = x - F(x, y) = 0
\]

\[
g(x, y) = y - G(x, y) = 0
\]

still work in progress:

- integration of OpenFOAM ([www.opencfd.co.uk/openfoam/](http://www.opencfd.co.uk/openfoam/)) via coupling interface

OpenFOAM (Open Field Operation and Manipulation) is a VERY impressive set of C++ libraries for very general CFD and multiphysics simulations

- Extension of the Photon Mapping module to participating media (absorption, refraction, scattering, ..)
Conclusions

- Framework itself is set up and in a quite stable state
- Possibility to build useful applications has been demonstrated

Outlook

- load modules across network – GRID integration ???
- convince some more people to use it, and exchange components
- Orcan will not be the last say ... but we believe that the future in numerical simulation will belong to modular and open systems

This work was supported by the German Federal Ministry of Education and Research (BMBF)
Grant Number 0327324A