Integration of FEM Analysis with CAD

Bauhaus-Universität Weimar, Informatik im Bauwesen

2009

Contents

Integration of FEM Analysis with CAD

1 Introduction

The integration between applications of Computer-Aided Design (CAD) and applications of the Finite Element Method (FEM) for the analysis of physical behaviour has been an issue of great interest for a number of years. Integration concepts most often utilized include:

- an integration via data exchange, i.e. relevant data are extracted from a CAD data file and imported via a data exchange definition into a FEM application. This generally means that two separate applications exist one which is used to export data and another one to import that data. The data format is either a specific format defined as a translator between two systems or a standardized data definition for data exchange.
- an integration via a common application development interface (API) between CAD and FEM functionality. This most often means that the integration is only accomplished within a closed, proprietary environment, i.e. concepts and implementations are only usable within the environment of a specific product which is not openly accessible to others.

Recently it has become apparent that an open, publically available approach has matured to a degree that it can serve as a basis for developing engineering applications and as a basis for practical applications as well. This has become well-known under the term “Open Source Initiative (OSI)“[1].

The technology used for these purposes is the same for a broad range of engineering applications (analysis, design, project management, etc.). The technology used in the context of this work is made publically available under the terms of the OSI and the name “Java” by the company SUN Microsystems [2].

At Bauhaus-Universität Weimar several applications were developed on the basis of these concepts. The purpose of these applications is primarily focused towards education and research. A general framework for FEM analysis with applications to heat transfer problems and structural analysis problems has been developed based upon this technology and is used for the above mentioned purposes. Furthermore a design application for CAD has been developed under the name *cademia* [3].

The purpose of this script is the exploration of FEM/CAD integration based upon a common, open and publically available development platform. The implementation includes the following steps:

- generate a new user command in CAD
- implement the functionality associated with the new command

The approach adopted for the purposes of a simple integration concept provides that the CAD application is started, the new command added and executed, input data for FEM heat transfer analysis is read, results are computed and mapped in the CAD environment onto standard CAD functionality (general path).

2 New User Command in CAD

In order to define a new user command in *cademia*, a class will be defined for starting the general CAD application and development platform *cademia*. This class will be named ***startCademia***. It will rely on *package cademia* to be available and it will basically invoke the main method of *cib\cad\Instance*.

```
package cademia;

import java.io.IOException;

public class StartCademia {

    public static void main(String[ ] args) {
        try {
            cib.cad.Instance.main(args);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

In the next step, a new user command with the name *fem* will be defined and added. This is accomplished via a **command script**, containing a sequence of valid user commands, stored in file *input\cademia\fem.cibs*. The commands defines a new scale to be set, a command *fem* to be added, the command *fem* to be executed and a complete zoom on all windows. The functionality associated with the command *fem* is provided by class *cademia\heat2d\cmds\ParseFile*.

```
setscale 10;
addCmd cademia.heat2d.cmds.ParseFile fem;
fem;
winfit -1;
```

Class *ParseFile* is stored in package *cademia\heat2d\cmds*. It relies on the availability of predefined FEM functionality (*Model*, Analysis, *FileParser* and *Exceptions*) and on the availability of predefined CAD functionality (*Kernel*, *UserInterface*, *CmdAbortedException* and *CmdAdapter*). The new, user defined functionality is implemented in class *VisualFemModel*.

```
package cademia.heat2d.cmds;

import java.io.File;
import java.io.IOException;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.filechooser.FileFilter;

import model.Model;
import analysis.Analysis;
import application.heat2d.parser.FileParse;
import model.ParseException;
import model.FemException;
import analysis.AlgebraicException;

// integration into cademia
import cib.cad.kernel.Kernel;
import cib.cad.kernel.UserInterface;
import cib.util.cmd.CmdAbortedException;
import cib.util.cmd.CmdAdapter;

import cademia.heat2d.comp.VisualFemModel;
```

The implementation of ***class ParseFile*** needs to be derived from a commmand adapter (*class CmdAdapter*). This is required for any command that is to be called from within *cademia*.

The adapter requires implementation of the method *doCmd*, which will start execution of the command. The CAD kernel software needs to create a context for the application to run in and to acquire a user interface needed for user interaction. Next, a new CAD model object is generated.

A file chooser object will present input files residing in the directory *input\heat2d*. It will return without execution if the file is not a valid input file for the heat analysis with the file extension .inp.

Next, the model object is populated with input from parsing the given input file, the model is passed to an analysis object and the analysis is performed.

The most vital implementation includes the evaluation of a new *VisualFemModel* object on the basis of the model data including input and result information.

Finally, the identifier of the model object is stored in the CAD kernel. The result from the *VisualFemModel* is mapped to a single CAD component and also stored in the kernel.

```
// if to be called by a command, it needs to be derived from Command Adapter
public class ParseFile extends CmdAdapter {
    public final String EXT = "inp";

    // execution of command
    public void doCmd(Object context) throws CmdAbortedException {
        System.out.println("Loading model...");
        // kernel needs to create a context and to get a user interface
        Kernel krnl = (Kernel)context;
        UserInterface ui = krnl.getUserInterface();
        Model model;
        try {
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setCurrentDirectory(new File("input/heat2d"));
            fileChooser.setFileFilter(new FileFilter() {
                public boolean accept(File f) {
                    return f.getName().toLowerCase().endsWith("."+EXT) ||
                                                                f.isDirectory();
                }
                public String getDescription() { return EXT+"-Dateien (*."+EXT+")"; }
            });

            File file = null;
            int result = fileChooser.showOpenDialog((JFrame)ui);
            if (result == JFileChooser.APPROVE_OPTION)
                                    { file = fileChooser.getSelectedFile(); }
            else return;
            if(file == null) return;
            model = new FileParser(file).getModel();
            Analysis analysis = new Analysis(model);
            analysis.perform();

            VisualFemModel vfem = new VisualFemModel(model);

            // kernel to store model id and add visual fe-model as a single component
            krnl.getDatabase().getNameSpace().put(vfem,model.getId());
            krnl.getDatabase().getComponentSet().add(vfem);

        } catch (ParseException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (FemException e) {
            e.printStackTrace();
        } catch (AlgebraicException e) {
            e.printStackTrace();
}   }    }
```

3 Implementation of Functionality for FEM Integration

The implementation of functionality for FEM integration requires to import Java functionality for various purposes:

- standard functionality,
- functionality for Graphical User Interfaces (GUI) in Java Swing technology,
- FEM functionality and
- CAD technology.

Class VisualFemModel is required to implement the complete interface definitions for components in *cademia*, i.e.

clone(), transformBy(AffineTransform mat), getShape(int name), setShape(AttributedShape shape, int name), shapeIterator(), getText(int name), setText(AttributedText text, int name), textIterator(), getControlPoint(int name), setControlPoint(Point2D pnt, int name), controlPointIterator(), setAttributes(Attributes attribute), getAttributes(), hasFeature(String name), getFeature(String name), setFeature(Feature feature), addFeature(Feature feature), featureIterator() and update(Object context)

Mehod *shapeIterator()* is the essential method for interfacing FEM with CAD technology in this context. It maps the components of FEM technology onto a corresponding component in CAD, which is a generalPath object of the Java 2D API.

Further user defined methods are given for generating the isolines and isoareas of a steady state heat problem. The evaluation of this information is quite elaborate and the implementation is beyond the scope of this script. The user defined methods include:

getStroke(float mm), calcColor(int step), getInterpolatedPoint(double x0, double x1, double vd, double v0, double v1), checkRightTurn(Point2D p0, Point2D p1, Point2D p2), computeConvexHull(Set set), createGeneralPathFromSetOfLines(Set line2D), calculateIsoLines() and *calculateIsoAreas()*.

3.1 Predefined Functionality

Standard functionality includes Java awt (Abstract Windowing Toolkit) [4], input and output, text and general utility functionality for Java data structures.

FEM functionality includes application objects, model and node functionality. Interfaces for nodes and elements need to be available and the specific elements for heat transfer need to be defined.

Finally, the functionality for integration into the CAD environment *cademia* needs to be imported.

```
package cademia.heat2d.comp;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Stroke;
import java.awt.Shape;
import java.awt.geom.GeneralPath;
import java.awt.geom.Point2D;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.NoSuchElementException;

// import FEM functionality
import model.Model;
import model.interfaces.IElement;
import model.interfaces.INode;
import application.heat2d.model.HeatElement;
import application.heat2d.output.VisualIso;
import application.heat2d.output.IsoLine;
import application.heat2d.output.IsoArea;

// import cademia functionality
import cib.cad.db.comp.ComponentAdapter;
import cib.util.AttributedShape;
import cib.util.CoordSpace;
import cib.util.coll.IteratorFilter;
import cib.util.coll.NamedListIterator;
import cib.util.coll._FilterableNamedListIteratorAdapter;
import cib.util.coll._MultiFilterableNamedListIteratorAdapter;
```

3.2 Application Specific Functionality

Class *VisualFemModel* contains the implementation of the basic integration between FEM and CAD for generating elements, isolines and isoareas. The constructor of class *VisualFemModel* will compute this information on the basis of a given model object.

```
public class VisualFemModel extends ComponentsAdapter implements
                                                            Serializable {
    private static final long serialVersionUID = 1L;
    private Model m_model;
    private VisualIso iso;
    private List m_components = new LinkedList();

    private static final int steps = 10;

    public boolean SHOW_ELEMENTS = true;
    public boolean SHOW_ISOLINES = true;
    public boolean SHOW_ISOAREAS = true;

    // color matrix, contains the rgb values at specific intervals, used to interpolate
    //                                                          step values
    private static final int[ ][ ] cmat = new int[ ][ ]
    { {78,0,178}, {0,125,255}, {0,255,0}, {255,200,0}, {255,0,0}, {178,50,0} };

    // constructor
    public VisualFemModel(Model _model){
        this.m_model = _model;
        iso = new VisualIso(m_model, steps);
        iso.calculateIsoLines();
        iso.calculateIsoAreas();
        System.err.println("# nodes = "+m_model.size(INode.class));
        System.err.println("# elements = "+m_model.size(IElement.class));
    }
```

3.2.1 Methods required by the CAD Component Interface

Class *VisualFemModel* is defined to implement *interface Component* of all CAD components. This requires implementation of the following methods. The essence of this particular FEM/CAD integration is defined in **method *shapeIterator().*** It utilizes objects of classes *IsoArea*, *HeatElement* and *IsoLine*.

```
public NamedListIterator shapeIterator() {
    List filters = new ArrayList();
    if(SHOW_ISOAREAS) {
        filters.add(new IteratorFilter() {
            public boolean matches(Object o) { return o instanceof IsoArea; }
        });
    }
    if(SHOW_ELEMENTS){
        filters.add(new IteratorFilter(){
            public boolean matches(Object o) { return o instanceof HeatElement; }
        });
    }
    if(SHOW_ISOLINES){
        filters.add(new IteratorFilter(){
            public boolean matches(Object o) { return o instanceof IsoLine; }
        });
    }

    IteratorFilter[ ] _filters = new IteratorFilter[filters.size()];
    for(int i = 0; i < _filters.length; i++) _filters[i] = (IteratorFilter)filters.get(i);
    return new _MultiFilterableNamedListIteratorAdapter(
                                            m_components.listIterator(),_filters){
        public Object next(){ return createShape(super.next()); }
        public Object previous(){ return createShape(super.previous()); }
        private Object createShape(Object obj){
            if( obj instanceof IsoArea ){
                IsoArea _isoArea = (IsoArea)obj;
                AttributedShape as = new AttributedShape(_isoArea.gp);
                as.setStroke(getStroke(.1f));
                as.setDrawPaint(Color.BLACK);
                as.setFillPaint(calcColor(_isoArea.step));
                return as;
            }
```

```
                else if( obj instanceof IsoLine ){
                    IsoLine _isoLine = (IsoLine)obj;
                    AttributedShape as = new AttributedShape(_isoLine.gp);
                    as.setStroke(getStroke(.25f));
                    as.setDrawPaint(Color.BLACK);
                    return as;
                }
                else if(obj instanceof HeatElement){
                    HeatElement _elem = (HeatElement)obj;
                    GeneralPath _elemShape = new GeneralPath();
                    _elemShape.moveTo((float)_elem.getNodes()[0].getCoordinates()[0],

    (float)_elem.getNodes()[0].getCoordinates()[1]);
                    _elemShape.lineTo((float)_elem.getNodes()[1].getCoordinates()[0],

    (float)_elem.getNodes()[1].getCoordinates()[1]);
                    _elemShape.lineTo((float)_elem.getNodes()[2].getCoordinates()[0],

    (float)_elem.getNodes()[2].getCoordinates()[1]);
                    _elemShape.closePath();
                    AttributedShape as = new AttributedShape(_elemShape);
                    as.setStroke(getStroke(.1f));
                    as.setDrawPaint(Color.LIGHT_GRAY);
                    as.setFillPaint(null);
                    return as;
                }
                else { return "nothing"; }
            }
    };
}
```

The terminology for general geometrical components in Java is *shape*. A *shape* is an arbitrarily “shaped” geometric entity that can be very complex. In the context of this integration concept, the FEM components *elements*, *isolines* and *isoareas* are iterated and all mapped to shape components as general geometrical CAD components. In fact, they are all collected into many simple shape components.

The final CAD components generated are so-called *control points* at each *node* component of the FEM model. Control points are utilized to handle and manipulate the *shape* component, i.e. control points can be used for dimensioning purposes or for performing general CAD functionality like for example transformations of geometry. In this case, all element nodes have been defined as control points.

```
public Point2D getControlPoint(int name) throws IllegalArgumentException {
    INode _node = (INode)m_components.get(name);
    return new
              Point2D.Double(_node.getCoordinates()[0],_node.getCoordinates()[1]);
}
public NamedListIterator controlPointIterator() {
    IteratorFilter _nodeFilter = new IteratorFilter(){
        public boolean matches(Object o) {
            return o instanceof INode;
        }
    };
    return new
      _FilterableNamedListIteratorAdapter(m_components.listIterator(),_nodeFilter){
        public Object next(){
            INode _node = (INode)super.next();
            return new
              Point2D.Double(_node.getCoordinates()[0],_node.getCoordinates()[1]);
        }
        public Object previous(){
            INode _node = (INode)super.previous();
            return new
              Point2D.Double(_node.getCoordinates()[0],_node.getCoordinates()[1]);
        }
        public int previousName() throws NoSuchElementException {
            return previousIndex();
        }
        public int nextName() throws NoSuchElementException {
            return nextIndex();
        }
        public void set(Object o) {
            Point2D p2d = (Point2D)o;
            INode _node = (INode)m_components.get(previousIndex());
            _node.setCoordinates(new double[]{p2d.getX(),p2d.getY()});
            _notifyWasChanged();
        }
    };
}
```

3.2.2 Methods defined for application-specific Functionality

In addition to those methods required to be implemented by the Component interface, additional methods can be implemented to cover the functionality of a specific application.

Method *getStroke* will define the linetype definition for rendering geometry either on the screen or on paper.

Method *calcColor(int step)* will calculate the gradation of colors for isochromatic areas associated with different levels of temperature.

```
public Stroke getStroke(float mm) {
     // Paper size in [mm]
    float paperToModel = (float)(CoordSpace.getCoordSpace().getScale() / 1000.);
    return new BasicStroke(
         /* float width      */  mm * paperToModel,
         /* int cap          */  BasicStroke.CAP_ROUND,
         /* int join         */  BasicStroke.JOIN_ROUND
    );
}

// maps a step to a color
private Color calcColor(int step) {
    // determine the color boundary
    double x = (step*1.)/(steps*1.-1);
    if(x == 0)  return new Color(cmat[0][0],cmat[0][1],cmat[0][2]);
    double _increment = 1./(cmat.length-1);
    for(int i = 1; i < cmat.length; i++){
         if( x <= _increment*i){
              int r = (int)(getInterpolatedPoint
                        (cmat[i-1][0],cmat[i][0],x,_increment*(i-1),_increment*i));
              int g = (int)(getInterpolatedPoint
                        (cmat[i-1][1],cmat[i][1],x,_increment*(i-1),_increment*i));
              int b = (int)(getInterpolatedPoint
                        (cmat[i-1][2],cmat[i][2],x,_increment*(i-1),_increment*i));
              return new Color(r,g,b);
         }
    }
    return new Color(cmat[cmat.length-1][0],cmat[cmat.length-1][1],
                                                            cmat[cmat.length-1][2]);
}
```

4 Summary

Different concepts are available for the integration of FEM with CAD. These include

1) data interface definition with corresponding data exchange between separate applications
2) data visualization without CAD functionality in a single application
3) mapping FEM components to appropriate CAD components in a single application with separate object models
4) integrating FEM and CAD functionality in a single application with an integrated object model, i.e. FEM components are implemented as structural objects in CAD

While concept 1 and 2 have been used most often in the past, method 4 was only available in a proprietary, closed application programming environment. Open source code based upon a consistent development environment was generally not available for CAD and to a limited degree for FEM.

With Java technology as an open, common development environment concept 3 was shown in this script to be feasible with very limited effort. Concept 4 would still require considerably more effort but could be implemented without major problems or uncertainties. In this case, each FEM component needed would have to be defined as a CAD component with all requirements on a general CAD component to be utilized with all the corresponding CAD functionality.

literature

[1] http://www.opensource.org/

[2] http://java.sun.com/docs/books/tutorial/java/, Copyright 1995-2005 Sun Microsystems, Inc. All rights reserved

[3] http://www.cademia.org, © 2006, Bauhaus-Universität Weimar | Lombego Systems

[4] Java 2D API Graphics, Vincent J. Hardy, SUN Microsystems Press, 2000