Object-Oriented Framework for Large Scale Air Pollution Modeling

Anton Antonov

National Environmental Research Institute,
Frederiksborgvej 399, DK-4000 Roskilde, Denmark
Anton.Antonov@uni-c.dk

Abstract. The Object-Oriented Paradigm (OOP) provides methodologies how to build flexible and reusable software. The OOP methodology of patterns and pattern languages was applied to construct the object oriented version of the large scale air pollution model known as the Danish Eulerian Model (DEM). The obtained framework is amenable to resolve new computational tasks (e.g. parallel local refinement simulations over Europe), and the design and analysis of new (for the framework) numerical methods. In the paper will be described the general design of the object-oriented DEM, the design of the different layers, and the documentation organization. It will be also discussed the advantages the embedding a computer algebra system in the framework.

1 Introduction

Simulations based on a Large Scale Air Pollution Model (LSAPM) involve heavy computations because of the large modeling domain and the number of the considered chemical components. These computations result from the rather abstract and subtle notions on which their mathematical algorithms are based on. A program that perform the simulation is, as a mater of fact, a model of these mathematical algorithms. Object-Oriented(OO) languages provide a paradigm, in which the entities, the invariants, and the relations within the subject being modeled, can be reflected in the programming code. If we want to provide an environment, where area specific investigations are made, it is natural then to prepare some preliminary code that reflects the principles common for any activity in that area. That preliminary code is called framework. A framework should be very easily tuned, completed, extended to a concrete program that meets the user needs.

The Object-Oriented Danish Eulerian Model (OODEM), based on the existing Danish Eulerian Model (DEM, [13]), is a framework for large scale air pollution model. In the article are discussed the approaches and the paradigms used to build OODEM – it was built with the framework pattern language described in [4], completed with the design patterns language described in [5].

The features and structure of OODEM, and the patterns used in it, are presented Section 2. In Section 3 is discussed how the computer algebra system *Mathematica* is used. The different task specifications and framework issues are

S. Margenov, J. Wasniewski, and P. Yalamov (Eds.): ICLSSC 2001, LNCS 2179, pp. 247–254, 2001.

described with a special pattern-style language, which is presented in Section 4. A heavier accent than normally is given on the documentation, since its description completes the patterns discussion in Section 2, and the documentation is often neglected in the large scientific codes.

2 The Object-Oriented Danish Eulerian Model

2.1 Design of OODEM

Temporal and spatial variations of the concentrations and/or the depositions of various harmful air pollutants can be studied [13] by solving the system (1) of partial differential equations (PDE's):

$$\begin{aligned}\frac{\partial c_s}{\partial t} = &-\frac{\partial(uc_s)}{\partial x} - \frac{\partial(vc_s)}{\partial y} - \frac{\partial(wc_s)}{\partial z} \\ &+\frac{\partial}{\partial x}(K_x\frac{\partial c_s}{\partial x}) + \frac{\partial}{\partial y}(K_y\frac{\partial c_s}{\partial y}) + \frac{\partial}{\partial z}(K_z\frac{\partial c_s}{\partial z}) \\ &+E_s + Q_s(c_1, c_2, ..., c_q) - (k_{1s} + k_{2s})c_s, \\ &s = 1, 2, ..., q.\end{aligned} \tag{1}$$

The different quantities that are involved in the mathematical model have the following meaning: (i) the concentrations are denoted by c_s; (ii) u, v and w are wind velocities; (iii) K_x, K_y and K_z are diffusion coefficients; (iv) the emission sources in the space domain are described by the functions E_s; (v) κ_{1s} and κ_{2s} are deposition coefficients; (vi) the chemical reactions used in the model are described by the non-linear functions $Q_s(c_1, c_2, \ldots, c_q)$. The number of equations q is equal to the number of species that are included in the model.

It is difficult to treat the system of PDE's (1) directly. This is the reason for using different kinds of splitting. A simple splitting procedure, based on ideas discussed in Marchuk[7] and McRae et al. [8], can be defined, for $s = 1, 2, \ldots, q$, by the following sub-models:

$$\frac{\partial c_s^{(1)}}{\partial t} = -\frac{\partial(uc_s^{(1)})}{\partial x} - \frac{\partial(vc_s^{(1)})}{\partial y} \tag{2}$$

$$\frac{\partial c_s^{(2)}}{\partial t} = \frac{\partial}{\partial x}\left(K_x\frac{\partial c_s^{(2)}}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y\frac{\partial c_s^{(2)}}{\partial y}\right) \tag{3}$$

$$\frac{dc_s^{(3)}}{dt} = E_s + Q_s(c_1^{(3)}, c_2^{(3)}, \ldots, c_q^{(3)}) \tag{4}$$

$$\frac{dc_s^{(4)}}{dt} = -(\kappa_{1s} + \kappa_{2s})c_s^{(4)} \tag{5}$$

$$\frac{\partial c_s^{(5)}}{\partial t} = -\frac{\partial(wc_s^{(5)})}{\partial z} + \frac{\partial}{\partial z}\left(K_z\frac{\partial c_s^{(5)}}{\partial z}\right) \tag{6}$$

The horizontal advection, the horizontal diffusion, the chemistry, the deposition and the vertical exchange are described with the system (2)-(6). This is not the only way to split the model defined by (1), but the particular splitting procedure (2)-(6) has three advantages: (i) the physical processes involved in the big model can be studied separately; (ii) it is easier to find optimal (or, at least, good) methods for the simpler systems (2)-(6) than for the big system (1); (iii) if the model is to be considered as a two-dimensional model (which often happens in practice), then one should just skip system (6).

The Chemical Sub-model (CS) reduces to large number of relatively small ODE systems, one such system per grid-point; therefore its parallelization is easy: the number of grid points (96×96, 480×480 in the two-dimensional DEM) is much bigger that the number of processors. From another hand, the two-dimensional Advection-Diffusion Sub-model (ADS) that combines (2) and (3), poses the non-trivial question how it should be implemented for parallel computations, especially when higher resolution is required on specified regions i.e. locally uniform grids are used.

The development task considered is the implementation of an object-oriented framework for DEM. The framework should be amenable for simulations with local refinements, 3D simulations, and inclusion of new chemical schemes. The way this task is approached is to adopt the splitting procedure (2)-(6) and to build first a framework for the ADS. It was assumed that the ADS framework should be flexible on what parallel execution model is used. After scanning different books, articles, and opinions was decided to start a Conceptual Layering framework [4] with conceptual layer for Galerkin Finite Element Methods (GFEM), and building blocks layer comprising a mesh generator package, and a package of parallel solvers for linear systems. It was decided to develop a mesh generator for locally uniform grids, and to employ PETSc ([3]) for the parallel solution of the linear systems. The initial Conceptual Layering framework mutated to a Multi-level (several conceptual layers) framework, because of the mesh generator. One more layer was added the Data Handlers layer. It responsible for the data approximation over the grids obtained by the mesh generator. The OO construction of the framework employs design patterns [5]. The design of the GFEM layer is based on the Template Method Design Pattern (DP), which is combined with the Abstract Factory DP that provide consistency of the usage of a number of Strategies that provide different behavior flexibilities. The design of the Data Handlers layer uses the design patterns (i) Essence to define a class that represent the fields over the used grids, (ii) Chain of Responsibility to handle the data requests over the locally refined grids, and (iii) Strategy for the different data readers and writers, (iv) and Decorator (see bellow).

Since PETSc is based on the Message Passing Interface (MPI), and because the MPI model runs on all parallel architectures, MPI is reflected in the GFEM layer. The idea behind the way the parallelism is facilitated is similar to the ideas used in OpenMP, HPF and PETSc: the user achieves parallelism via domain decomposition, designing sequential code that is made parallel with minimal changes (comments in OpenMP and HPF, and name suffixes in PETSc). The

Decorator DP for the data feeding was applied to the Data Handlers layer; the Strategy DP for the parallel/sequential GFEM computations was added to the GFEM layer. Using these two patterns the sequential and the parallel code look in the same way: in order to add new parallel behavior, new classes are added, the existing code is not changed. Also, the use of the Strategy DP makes the GFEM classes independent of the linear solvers package.

The mesh generator provides description of the designed by the user grid. The grid nodes are ordered linearly according to their spatial coordinates. They are divided into classes: each class contains nodes with equal patches. The description is read by the GFEM layer and the nodes are distributed in equal portions among the parallel processes. GFEM classes know the lowest and the highest number of the nodes their processes are responsible for. For the machine presentation of the GFEM operators are used sparse matrices distributed over the processors; their handling is provided by PETSc.

The design of the top CS layer resembles the following view: a group of chemists develop a reaction mechanism, a physicist works out the formulae for the photochemical coefficients, another physicist derives the deposition model, and finally a numerical analyst finds a suitable numerical method for the stiff system of ODE's derived from the chemical reactions. (Even if one person is going through all of the stages, he/she acts as the mentioned scientists.)

From numerical point of view, the hardest question is how to treat numerically the stiff ODE system. It is suitable to tackle this problem separated from the others mentioned above: they are somewhat easier. If we have a solver for this system we should (just) attach to it routines for the rest of the computations: the photochemical coefficients, and the deposition. It is natural to apply the Decorator DP to coat the ODE system solver with the other routines, with which it should be included in OODEM.

Figure 1 presents the current state of the OODEM development and documentation (see Section 4). On Table 1 are shown the design patterns used in the different OODEM layers. Below are listed the framework features.

Table 1. Patterns used in the OODEM layers. The pattern names are abbreviated as: Template Method (TM), Strategy (S), Decorator(D), Chain of Responsibility (CR), Abstract Factory (AF), Builder (B).

	TM	S	D	CR	AF	B
DEM Layer						
GFEM Layer	*	*			*	
Chemistry Sub-model Layer			*			
Data Handlers Layer		*	*	*		*
Mesh Generator Layer		*				
PETSc Layer	*	*			*?	*?

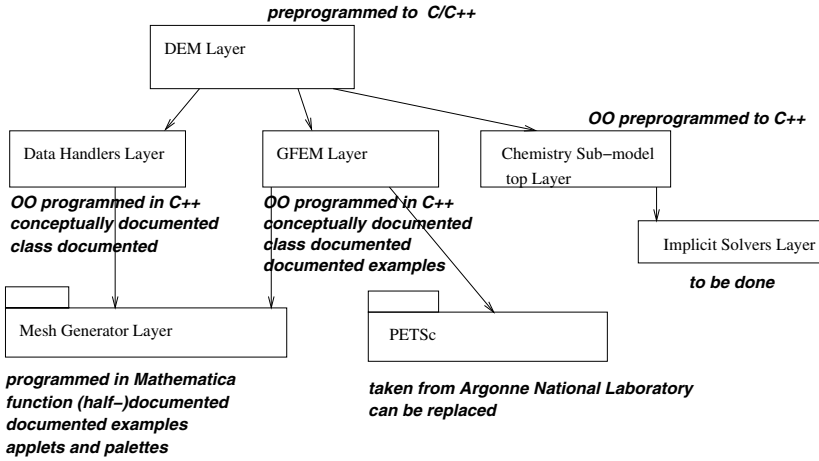

Fig. 1. Layers and their dependencies of OODEM framework. The expression "OO programmed in C++" means "C++ programmed in the object-oriented paradigm using design patterns". See Section 4 for the types of documentation in OODEM.

2.2 Features

OODEM has the following features

- Employs operator splitting for the different physical processes
- Genuine two-dimensional advection simulation
- Numerical zooming with static grids over several specified by the framework user regions
- MPI based; able to run on both shared and distributed memory machines
- Amenable for extension to three-dimensional simulations (1D×2D or 3D)
- Amenable for non-conforming and high-order finite element methods
- Has a mesh generator for locally uniform grids with triangular and rectangular element geometry
- Framework's mesh generator is independent from the rest of the framework; hence, can be replaced with another one
- Framework's mesh generator has routines for visualizing grids, their interpretation and their place over a given geographical map
- Most of the framework layers are with object-oriented design using design patterns
- Flexible data feeding
- Conceptually documented, class documented, documented examples; the documentation uses UML; the documentation is web available
- Employs PETSc for the solution of large sparse linear systems (other library can be used)

3 Use of *Mathematica*

Mathematica [12] was used in the initial stages of the development, in order to make a prototypes of Galerkin finite element methods. Then the whole mesh generator for locally uniform grids was developed in *Mathematica*. In a way, all the theory needed to construct a GFEM method is contained in the Mesh Generator layer: the other layers just manipulate the generated grid descriptions (GFEM layer) and approximation rules (Data Handlers layer).

The separation of "the theory" from the implementation, proved to be fruitful. It is easy to cope with finite element basis function and their properties within the computer algebra system. Grid description files with a simple format are provided to the simulation code, for which all finite element methods are in this way "the same". The consequence is that it is easy to introduce new GFEM methods in the framework.

Although the mesh generator is developed within the modular programming paradigm, it achieves, via overloading, a shallow form of OO polymorphism. This gives the ability to produce descriptions for different types of grids: rectangular, triangular, mixed rectangular and triangular, segmental (1D), non-conforming.

The developed mesh generator package was completed with a package for analysis of the GFEM, which was also used in the debugging phases.

The Generic ODE Solving System (GODESS, [10]) uses similar separation of the theory from the implementation: a method's Runge-Kutta coefficients are calculated with a program written in Maple (a computer algebra system), and with them a class for the method is generated.

4 Documentation

One of the most important framework features is the documentation: framework without a documentation is not a framework. Framework documentation is difficult to write, since frameworks have greater level of abstraction than most software: frameworks are also reusable designs. The documentation of OODEM is divided into three parts: conceptual documentation, class documentation, and documented examples.

4.1 Conceptual Documentation

The conceptual documentation of OODEM is based on ideas presented in [6]. OODEM was build with three pattern languages: (i) a framework pattern language described in [4], (ii) a software micro-design pattern language, the *design patterns* language, described in [5], and (iii) a special pattern language to specify the addressed problems and the adopted assumptions (see [1]). The last pattern language – we will call it *task definition* language – leads into patterns from the other two. It has two formats of pattern descriptions:

– *Problem specification* format of a module or programming task that has the topics

- Intent,
- Explanation of the context ,
- Objectives,
- Approach.

The problem specifications are further unfolded with the design patterns language in a separate sections.

– *Framework issue* format for describing the macro-problems that cannot have detailed specification e.g. the building of the framework, the structure and functionality of its layers or sub frameworks. It has the topics

- The addressed problem
- The patterns used

A framework issue is eventually further refined with either problem specifications or another framework issues(s). It can also directly lead to design patterns – so there is no more refinements of it.

The conceptual documentation was made for pedagogical purposes. The task definition language is made to clarify how the framework issues are resolved to concrete-programming patterns – the design patterns. It does not provide random access to the patterns in it. The access is hierarchical: one can restrict himself just to the levels he wants to be aware of. The access to the descriptions of the design patterns usage can be random. In general, patterns provide random access to a documentation made with them.

The explanations in the conceptual documentation are accompanied with Unified Modeling Language (UML) class and sequence diagrams. The UML class diagrams express the static structure of a system in terms of classes and relationships. The UML sequence diagrams illustrate interactions between objects using a temporal structure that represents the order of communication [9]. Our experience shows that the UML sequence diagrams are easier to comprehend by newcomers who are not familiar with OO programming.

4.2 Class Documentation

The class documentation describes the interfaces of the classes in the framework and the roles they play in the applied in OODEM design patterns. The OODEM class documentation can be found on [2]. It is important the class documentation to be kept up to date. We use Doxygen [11] to generate the on-line HTML and LaTeX documentation from comments included in the C++ source code.

Our experience with the class documentation is that the class creators should be especially well documented: besides the parameter list, it should be described in what context they are supposed to be used, and what context they will produce. Ideally, the re-users of the class should not be concerned about the obtained encapsulated context, but they and the framework constructors will benefit from these descriptions when the framework's code (and may be design) is not completely stabilized.

4.3 Documented Examples

This is the first thing a newcomer will look for/look at. Our approach is to make extensive comments in the example programs, and provide an examples guide. See, again, [2].

5 Concluding Remarks

The usefulness, easiness to use, and versatility of OODEM, would reveal themself in a longer term. Nevertheless, the employment of the object-oriented paradigm, and especially the employment of the design patterns, give us confidence that the framework has these properties. (See [1] where is *proven* that the design patterns Template Method, Strategy, Abstract Factory, Builder and Decorator, provide code reusability – with them the existing code is not changed, just new code is added.) The easiness to use is tightly connected with the documentation: good documentation is crucial.

Possible future extensions of OODEM are described in [1, Ch. 8].

References

1. A. Antonov. Object-Oriented Framework for Large Scale Air Pollution Models, PhD thesis, Danish Technical University, April 2001.
2. A. Antonov. The Object-Oriented Danish Eulerian Model Homepage. http://www.imm.dtu.dk/~uniaaa/OODEM/, 2001.
3. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. PETSc home page. http://www.mcs.anl.gov/petsc, 2000.
4. S. Ben-Yehuda. Pattern language for framework construction, in R. Hanmer, (ed.), *The 4th Pattern Languages of Programming Conference 1997*, 97-34, Technical Report. Washington University, Technischer Bericht, 1997. http://jerry.cs.uiuc.edu/~plop/plop97/Workshops.html.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.
6. R. Johnson. Documenting frameworks using patterns, in *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) '92 Proceedings*, ACM Press, 1992.
7. G. I. Marchuk. *Methods for Numerical Mathematics*, Springer-Verlag, 2 edition, 1982.
8. G. McRae, W. R. Goodin, and J. H. Seinfield. Numerical solution of the atnospheric diffusion equation for chemically reacting flows, *Journal of Computational Physics*, 45(1), 356–396, 1982.
9. P.-A. Muller. *Instant UML*, Wrox Press, 1997.
10. H. Olsson. Runge-Kutta Solution of Iinitial Value Problems, PhD thesis, Lund University, Sweden, November 1998.
11. D. van Heesh. Doxygen. http://www.doxygen.org, 2001.
12. S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*, Wolfram Media, Cambridge University Press, 4 edition, 1999.
13. Z. Zlatev. *Computer Treatment of Large Air Pollution Models*, Kluwer, 1995.