XVIII International Conference on Water Resources
CMWR 2010
J. Carrera (Ed)
©CIMNE, Barcelona, 2010

# AN XML BASED PROBLEM SOLVING ENVIRONMENT FOR HYDROLOGICAL PROBLEMS.

**Luit J. Slooten**[*], **Fransisco Batlle**[†] **and Jesus Carrera**[††]

[*]KWR Watercycle Research Institute
Groningenhaven 7
3433 PE Nieuwegein, the Netherlands

[†] Geomodels
Parc cientific de Barcelona
Edifici Florensa,C/ Adolf Florensa s/n.
08028-Barcelona, Spain

[††] Jaume Almera Institute
Lluís Solé Sabarí
08028 Barcelona,Spain

**Key words:** simulation, optimization, software design, automatic derivation, finite elements

**Summary.**

## INTRODUCTION

Modern hydrogeological simulation software needs to be able to deal with a variety of issues such as coupled phenomena, stochastic formulations, sensitivity analysis and parameter estimation. This creates a challenge for software developers. Modelling software must be sufficiently flexible to allow solving many different equations, sufficiently robust to deal with sometimes strong coupling, and sufficiently efficient to allow parameter estimation, sensitivity analysis or both.

In response to these demands, the way simulation software is built has evolved over the years. The once-dominating procedural programming paradigm has been complemented with newer paradigms such as object-oriented programming (OOP)and related paradigms. A result of this paradigm shift, powerful new types of simulation software have emerged. The best example is the multiphysics framework, which allow solving arbitrary sets of coupled partial differential equations[10,11,6] or coupling arbitrary models[2,3]. While there are many frameworks focussed on mechanics or chemistry, there is none for general purpose groundwater hydrology.

However, there is a clear need for such a tool to allow the modeller to explore the cross-effects of hydrologically relevant issues such as large heterogeneity ; strong nonlinearities in systems (e.g. multiphase flow); coupling between phenomena (e.g. variable density flow and reactive transport) and simultaneous coupling with other systems (e.g. river-aquifer interaction, soil-water-atmosphere transfer schemes, or solute exchanges between mobile and immobile regions).

The program presented in this paper (Proost- Process Oriented Optimization and Simulation Tool) is designed to be a step toward such a general purpose hydrological modelling tool. The scope is limited to modelling and optimization (no decision support or graphical interface is provided), but allows the end user to define a variety of hydrological and mathematical "objects" (such as transmissivity fields,transport equations or spatial discretizations). These objects can be combined within a set of rules to construct numerical models of hydrological systems. This reflects a considerable deviation from the traditional, switch-board approach to user input. An XML input format was defined for the purpose. Internally, it has some of the abstractions and relationships that are present in many multiphysics frameworks (e.g. mesh, solver, field, ...) and some more specific to hydrology.

## OUTLINE OF DESIGN GOALS

### Flexible problem definition

One of the design goals of Proost is to allow the end-user to define the equations that need to be solved, and to allow spatial variability in this definition, rather than offering a limited number of choices. This flexibility should not be limited to the equations being solved, but also to the solution method, boundary and initial conditions etcetera.

The approach Proost takes to allow this flexibility, is letting the end user "declare" hydrological objects such as e.g. fields, time functions, processes and equations in input files in much the same way as a programmer would declare variables in a program. These objects are represented by XML elements in the input files. The amount of information contained in each object varies from a few attributes to a deep tree structure of sub-elements. The model emerges from these objects by establishing links and dependencies between them, for example by declaring which of the existing field is to be used as the transmissivity field of a flow process. In this example, both the field and the process are objects.

### Dealing with coupled problems

Hydrological models often need to include coupling between different equations or subsystems. Example include variable density flow, multiphase flow and coupling with the atmosphere. As coupling is essentially a source of nonlinearity, it comes a no surprise that the methods to deal with coupling stem from the family of methods to deal with nonlinearity. While there are many methods to deal with nonlinearity, the most used ones can be divided in two families of methods: the family of Newton's method, and the family of Picard's method[7,9].

Implementing both methods has been another design goal of Proost. While Picard's method requires little more than being able to solve the individual equations, Newton's method requires deriving all coupled equations to all unknowns. As a consequence, the computation of derivatives was explicitly considered in the design.

### Parameter sensitivity

When dealing with a parameterized model, it is often useful to compute the sensitivity of simulation results to the model parameters. These sensitivities can be used for the computation of prediction confidence intervals, for the quantification of parameter covariance or for experiment design. Furthermore, sensitivity calculation is needed to allow the use of parameter estimation methods (e.g. Marquardt-Levenberg's method and the steepest descent method). There are two common approaches to sensitivity calculation: parameter perturbation and direct derivation. When using parameter perturbation, a model run is perfomed with the true or current parameters, followed by one run for each model parameter with that parameter slightly perturbed. An estimate of the sensitivity is obtained by comparing the simulated results for each perturbed run with the first run.

An alternative method is the direct derivation approach. This method relies on computing the derivatives of the state variables to the parameters, by deriving model equations with respect to the parameters. As discussed in[4], this leads to solving an extra linear system per timestep and per parameter, once the nonlinear problem loop has converged. The system matrix is in fact the same for all parameters (which can be taken advantage of by some math libraries), and it is equal to the jacobian of Newton's method.

The potential for time saving with respect to the parameter perturbation method lies in the fact that the nonlinear problem loop needs not be repeated for every time step and every parameter.

One of the goals of Proost was to implement both methods. The parameter perturbation is necessary because it is robust and only demands being able to solve a model with different parameter sets. The direct derivation method is necessary because it can greatly speed up sensitivity calculation for nonlinear problems.

## DESIGN AND IMPLEMENTATION OF THE PROOST FRAMEWORK

### Design pillars

Proost implementation is based on four general pillars.

The first pillar is ensuring compatibility of different class extensions. This means that if class $A$ uses objects of base class $B$, and base class $B$ has several specializations, then class $A$ should not know and not care which specialization of $B$ is active. By being unaware of specializations, they become independent of them (provided that all specialization classes implement all the base class interfaces). However, in some cases, generic types and interfaces are not enough. Consider the example of a field which may have values associated to nodes or elements. If a new type of mesh would be implemented (using finite difference cells instead of nodes or elements), the compatibility with fields would be lost. In this case, ensuring compatibility requires adding more abstraction: the field's definition must be expressed in terms of generic mesh building blocks, without specifying whether they are elements, nodes or cells. In the ideal case, complete compatibility of "everything with everything" gives rise to a quadratic growth model of application power with application expansion.

The second pillar is automatic derivation. In general, any object containing values that directly or indirectly depend on those of the state variables or of the parameters, must be able to compute derivatives to both, which is feasible through repeated application of the chain rule.

The third pillar is performance related: a preference to perform bulk operations. The division of responsibilities over classes is done where possible in such a way that large amounts of data can be processed in a single subroutine. This design choice mostly affects low-level modules where actual computations are carried out(e.g. mesh or matrix); for high level classes it has little consequences.

The fourth pillar is also performance related: the need to avoid unnecessary computations. The approach taken in Proost to this issue is based on storing the results of expensive computations, and using class methods that evaluate whether re-computing these results is necessary, or whether those of the last evaluation are still valid. To this purpose, a global variable known as the "time stamp" was introduced. Every time an expensive computation is carried out, the current value of the time stamp is stored along with the results of the computation, and the time stamp itself increases one. In order to see whether computed results are up-to-date, the time stamp associated to these results needs to be compared with that of the objects they depend upon. Class methods were written for this purpose. Furthermore, the class methods that perform actual computations were made private. Instead, clients can call a "get" method, which first checks whether the stored values are still up-to-date, and calls the corresponding computation method if this is not the case.

## Global description of classes

The base classes of Proost often represent concepts that are of use in general, multiphysics simulation tools, while the specialization classes that extend these base classes are more hydrological in nature. General purpose classes are: Matrix (implementing storage of and operations with matrices), time function, mesh (implementing spatial discretization and numerical methods), meshfields (implementing storage of and operations with discretized fields) and territory elements (containing sub domains).

Proost derives its name from the process class. This class represents physical processes (its specializations include advection, dispersion, sinks and sources, etc. as terms in a conservation equation . In this sense, processes are similar to the "brick" class of Getfem[8]. Responsibilities of the class include evaluating the process' terms to compute the process' contribution to either (mass) balance or to solution matrices, as well as the derivatives of

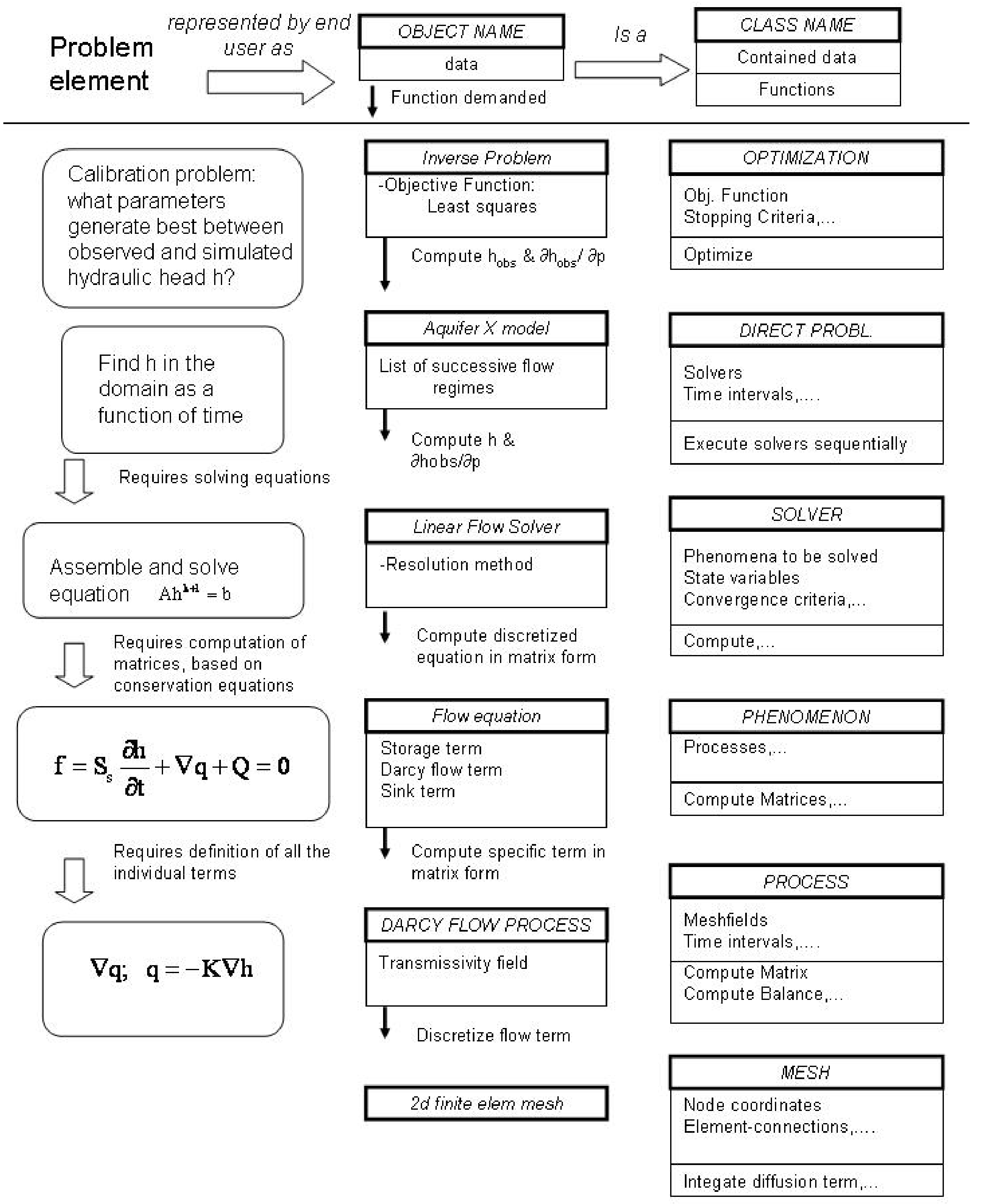
Problem element
represented by end user as
OBJECT NAME
data
Function demanded
Is a
CLASS NAME
Contained data
Functions
Calibration problem: what parameters generate best between observed and simulated hydraulic head h?
Inverse Problem
-Objective Function: Least squares
Compute $h_{obs}$ & $\partial h_{obs}/\partial p$
OPTIMIZATION
Obj. Function
Stopping Criteria,...
Optimize
Find h in the domain as a function of time
Aquifer X model
List of successive flow regimes
Compute h & $\partial hobs/\partial p$
DIRECT PROBL.
Solvers
Time intervals,....
Execute solvers sequentially
Requires solving equations
Assemble and solve equation $Ah^{k+1} = b$
Linear Flow Solver
-Resolution method
Compute discretized equation in matrix form
SOLVER
Phenomena to be solved
State variables
Convergence criteria,...
Compute,...
Requires computation of matrices, based on conservation equations
$f = S_s \frac{\partial h}{\partial t} + \nabla q + Q = 0$
Flow equation
Storage term
Darcy flow term
Sink term
Compute specific term in matrix form
PHENOMENON
Processes,...
Compute Matrices,...
Requires definition of all the individual terms
$\nabla q; \quad q = -K\nabla h$
DARCY FLOW PROCESS
Transmissivity field
Discretize flow term
PROCESS
Meshfields
Time intervals,....
Compute Matrix
Compute Balance,...
2d finite elem mesh
MESH
Node coordinates
Element-connections,....
Integate diffusion term,...

Figure 1: The main classes of Proost

both to unknowns and parameters. Each process is associated to a user defined domain that may overlap with that of other processes. This way, complex boundary conditions may be simulated (simultaneous injection of various fluids with different concentrations at the same location for example) without simplifications, and correct mass balances may be computed.

Conservation equations are implemented by the phenomenon class (similar to the "physics" class of[1] or the "process" class of[5]). Each phenomenon is defined as a sum of processes. Hence, the phenomenon' precise form may change in space, and properties such as nonlinearities of coupling may be present locally or in all the domain. In the current implementation, the phenomenon may be expressed as

$$f^j = \sum_{i=1}^{n\_eq} A_i^j u_i + D_i^j \frac{\Delta u_i}{\Delta t} + b_j = 0 \tag{1}$$

where $f^j$ is the j-th phenomenon,$u_i$ is the set of unknowns associated to phenomenon $i$, $A$, $D$ and $b$ are matrices and an independent term computed according to a numerical method. The phenomenon object is responsible for computing the coefficient matrices and the independent term $b_j$. Other responsibilities include the computation of secondary results such as velocity or balance. Derivatives to parameters and unknowns may also be computed.

One level higher are the solvers. A solver has one or more phenomena associated to it. The solver implements a time-stepping scheme and is responsible for assembling the coefficient matrices and derivative matrices computed by the phenomena into solvable systems of equations, and solving these, as well as performing nonlinear problem iterations when necessary.

The Model class contains one or more solvers. Each solver has an associated time interval, allowing the simulation of a system in different steps (e.g. by simulating a system first in a steady state, and then in a transient regime).

Finally, there are several classes that allow running optimization problems: Observations (containing actual measurements of the system), Simulated Observations (responsible for computing simulated equivalents to the observations), Parameters, Objective Function and Optimization. Of these, the Objective Function class is responsible for computing the function to be optimized (e.g. a measure of model fit to desired output), as well as its derivatives. The Optimization class is responsible for finding extreme values of the Objective Function by iteratively changing the values of the model parameters. The Parameter class contains estimable parameters. Several building blocks of the Model (specifically Time Functions and Fields) can be expressed in function of parameters.

## SUMMARY AND CONCLUSIONS

We presented Proost, an object-oriented code for mathematical modelling of hydrological systems, designed with 4 objectives in mind: to allow the end-user to define the problem to be solved with relative freedom; to be able to deal with coupled equations in an efficient way; to implement parameter estimation and sensitivity computation in an efficient way, and to have a maintainable and expandable software product. To meet the first design goal, allowing the user to define hydrological models freely, an XML format was designed. The XML elements represent hydrological and mathematical concepts (eg a field, a conservation principle or a diffusion process) that can be specified and combined by the end user to build a model.

In order to be able to deal with coupling, both the Newton-Rhapson and Picard linearization schemes were implemented. The computation of the jacobian matrix of Newton-Rhapson's method requires derivation functionality in many classes; however, the low-level actual derivative computations are always carried out in the meshfield class and the mesh class. To be able to compute parameter sensitivities, the direct derivation approach to sensitivity calculation was implemented, as well as the parameter perturbation.

To increase computational efficiency a timestamp-based structure was implemented to keep track of which computations are necessary and which ones are not. This automated approach provides efficiency to the end user and takes a burden off of the developer. Finally, to favour maintainability and expandability, our design was made as much as possible in terms of hydrological or mathematical concepts.

Our experience in the making and using of the Proost problem solving environment has shown us positive and negative aspects of the design and of XML. To start with the design, the most positive aspect we found was the flexibility in combining classes to make models. The most negative aspect is the coupling that this implies in the class' implementations. This makes it difficult to re-use these classes outside the scope of the Proost application. Our experience with XML for the end user was positive. Text-based input is inevitably more tedious than graphical user interface input. Still, choosing clear element and attribute names made the input files relatively readable for the human eye.

## Acknowledgements

One of the authors was financially supported with an FI scholarship of the Department of Universities, Research and Information Society of the Generalitat de Catalunya. The work was carried out within the PIM project of ENRESA. Financial support also came from the Institute of Geology and Mining of Spain, IGME.

## References

[1] C. Boivin and C. Ollivier-Gooch. A toolkit for numerical simulation of pdes. ii. solving generic multiphysics problems. *COMPUTER METHODS IN APPLIED MECHANICS AND ENGINEERING*, 193(36-38):3891–3918, 2004.

[2] Tom Bulatewicz, J. Cuny, and M. Warman. The potential coupling interface: Metadata for model coupling. In *Proceedings of the 2004 Winter Simulation Conference*, 2004.

[3] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. The cactus framework and toolkit: Design and applicationsinvited talk. *HIGH PERFORMANCE COMPUTING FOR COMPUTATIONAL SCIENCEVECPAR 2002*, 2565:197–227, 2003.

[4] J. J. Hidalgo, L. J. Slooten, A. Medina, and J. Carrera. *Groundwater And Saline Intrusion Selected : Papers From The 18th Salt Water Intrusion Meeting. 18th SWIM, Cartagena 2004*, chapter A Newton-Raphson based code for seawater intrusion modelling and parameter estimation, pages 111–120. Number 15 in Hidrogeologa y Aguas Subterraneas. IGME, Madrid, 2005.

[5] O. Kolditz and S. Bauer. A process-oriented approach to computing multi-field problems in porous media. *JOURNAL OF HYDROINFORMATICS*, 6(3):225–244, July 2004.

[6] H. P. Langtangen and O. Munthe. Solving systems of partial differential equations using object-oriented programming techniques with coupled heat and fluid flow as example. *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE*, 27(1):1–26, March 2001.

[7] M. Putti and Paniconi. Picard and newton linearization for the coupled model of saltwater intrusion in aquifers. *Adv. Water Res.*, 18:159–170, 1995.

[8] Y. Renard and Pommier J. *Getfem++, a Generic Finite Element library in C++. Short User Documentation.*

[9] M. W. Saaltink, J. Carrera, and C. Ayora. On the behavior of approaches to simulate reactive transport. *JOURNAL OF CONTAMINANT HYDROLOGY*, 48(3-4):213–235, April 2001.

[10] R. Sahu, M. J. Panthaki, and W. H. Gerstle. An object-oriented framework for multidisciplinary, multi-physics, computational mechanics. *ENGINEERING WITH COMPUTERS*, 15(1):105–125, 1999.

[11] A.V. Smirnov. Multi-physics modeling environment for continuum and discrete dynamics. *International Journal of Modeling and Simulation*, 24(3):190–197, 2004.