A process-oriented approach to computing multi-field problems in porous media

Olaf Kolditz and Sebastian Bauer

ABSTRACT

Object-oriented methods are becoming more and more important in order to meet the challenges in scientific computing, such as the treatment of coupled multi-field problems with high spatial resolution. This paper introduces an object-oriented concept for modelling multi-process systems in porous media. The basic idea is the direct representation of a physical process by an object of the numerical code.

Because a physical process can be represented by a field variable and an equation system, the process object contains the equation system for the field variable as well as all methods needed for building and solving the equation system. The process object is thus self-configuring and well encapsulated, which are the main advantages of this new approach and thus allows for an easy extension to more complex applications with a higher number of processes.

The design and implementation of the object-oriented concept for the presented process objects is described in detail and illustrated by a set of application examples.

Key words | multi-field problems, object-orientation, porous media, process-orientation

Olaf Kolditz (corresponding author)
Sebastian Bauer
Center for Applied Geoscience,
Geohydrology/Hydroinformatics,
University of Tübingen,
Sigwartstrasse 10,
72076 Tübingen,
Germany
Tel: +49 7071 2973170;
Fax: +49 7071 5059;
E-mail: *kolditz@uni-tuebingen.de*

1 INTRODUCTION

Although the fundamentals of object-orientated programming (OOP) were established in the 1960s, it remains a very important concept to face challenges in scientific computation, such as the solution of multi-field problems with a large number of field variables. One of the reasons for its popularity is that software of increasing complexity has to be developed by large programmer teams. OOP has significant advantages by allowing rapid software development through component sharing and code reuse. Programming languages such as Smalltalk, C++ (Stroustroup 1991) and Java were designed in order to meet the requirements of object-oriented programming techniques.

The earliest use of OOP was connected with graphical applications and data visualization. These developments were instigated by natural analogues. Thus geometric objects like circles, triangles, spheres, etc, are treated as graphic objects. Another trigger for OOP was the requirement of graphical user interfaces (GUI) in order to handle the increasingly complex software. Utilization of OOP in numerical analysis came up in the 1990s (Mackie 1992; Scholz 1992; Zimmerman *et al*. 1992), who presented OOP versions of finite-element codes. OOP was also successfully introduced in several branches of computational mechanics, e.g. stress analysis (Dubois-Pelerin & Zimmermann 1993), structural analysis (Pidaparti & Hudli 1993), shell structures (Ohtsubo *et al*. 1993), inelastic strain analysis (Mentrey & Zimmermann 1993), contact problems (Feng 1995), fluid dynamics (Peskin & Hardin 1996; Kolditz 1999) and heat transfer and solidification (Cross *et al*. 1998). A comprehensive review of OPP in finite element analysis can be found in Masters *et al*. (1997). Akin (1999) points to OOP possiblities using Fortran90. Samapth & Zabaras (2000) use OOP to implement adjoint techniques in the field of continuum mechanics. Desitter *et al*. (2000) use OOP techniques for developing finite element groundwater flow models and demonstrate the possibilities of fast code extension.

Vonhooren *et al*. (2003) use OOP techniques for the modelling of biological wastewater treatment.

After Budd *et al*. (2002), the general principles of object-oriented programming include:

- extraction of data units (objects) as well as relationships between them (data model),
- object encapsulation (properties and methods) with private as well as public data and functions (classes),
- user-defined data constructs and abstract data types,
- virtual functions (prototypes), with defined interfaces but to be implemented (polymorphism),
- class hierarchy and heredity (basic and derived classes),
- communication by message passing (method execution on request),
- software reuse.

In the context of geosystem modelling, we use OOP to face a large variety of problems in environmental fluid mechanics. These include single-phase groundwater flow in porous and fractured media, multiphase flow in soils and rocks (Thorenz *et al*. 2002) and heat transport in fractured rock as well as deformation processes in geotechnical systems (Kolditz 2002). Furthermore, mass transport processes and chemical reactions between the components have to be calculated when dealing with contaminated aquifers (Habbar 2001).

In this paper we present the design and implementation of an object-oriented approach to the computation of multi-field systems. In this approach, each physical process considered, e.g. the movement of groundwater or the transport of heat, is associated with an object of the modelling software. Thus a multi-field system is translated into a multi-process algorithm by the use of OOP techniques. As a result, an arbitrary number of processes, which may interact with each other, can be generated, configured and solved. We term this approach a 'process-oriented approach', as the physical processes governing reality are also the objects of the program structure. Processes are implemented as objects (`PCS`) organized in a class (`CRFProcess`) containing all information and data required to treat the connected physical problem according to the finite element method.

Table 1 | PDE types of processes

Acronym	Process	PDE type
T	Thermal	Mixed parabolic–hyperbolic
H	Hydraulic	Parabolic
M	Mechanical	Elliptic
C	Componental	Mixed parabolic–hyperbolic

After a general outline of the processes in multi-field problems and the requirements for the object implementation in section 2, section 3 describes the implementation of the new process object and the multi-process algorithm. A dialogue-based user interface is described in section 4. Section 5 presents three application examples of increasing complexity, which demonstrate the use of the process orientation. For each application, the governing equations, the process implementation and the simulation results are given in detail.

2 OBJECT-ORIENTED DESIGN

2.1 Processes of multi-field problems

In multi-field problems we have to deal with several processes, which are described by a corresponding set of partial differential equations (PDEs). We may have different types of processes such as flow (H process), heat transport (T process) and deformation (M process). Those processes are described by PDEs of different types (compare Table 1), which are derived from the basic conservation principles of mass and energy together with the corresponding material laws (e.g. Kolditz 2002). Associated with each PDE are the corresponding initial and boundary conditions and material parameters as well as the appropriate numerical and solution techniques.

Processes can have complex relationships and interactions. Figure 1 shows the interactions possible for two-phase fluid flow (Processes H1 and H2) with heat

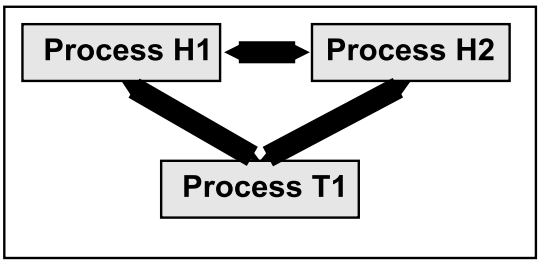

Figure 1 | Dependences of H^2T processes.

transport (Process T1). The two flow processes are coupled and interact by their respecitve saturations, while the heat transport process is partly based on the flow field of the two H processes. Additionally, the temperature distribution may determine material properties which influence the fluid movement.

Traditionally, one computer code was developed to simulate one physical process. The code could therefore deal with one PDE and the associated sets of boundary and initial conditions and provided a set of numerical techniques to solve the equation system. These codes were then extended to deal with coupled processes or to handle more than one PDE. This was often done without changing the data and program structure used and thus led to a complicated and inflexible code. As both research and applications move towards more complex and thus coupled systems, the need for a new design of the multi-process approach became apparent.

The solution of an equation system EQS as the representative of a physical process is a universal procedure which can be applied to each process independent of their specific type. This approach allows us to generalize the code and to treat processes in an object-oriented way.

The computation scheme must be very flexible concerning the number of processes, i.e. the number of H and T processes. Additionally, there must be a flexible way of specifying the relation between the flow processes as well as a very general way to define the interactions to the T process. The new process object therefore corresponds to all numerical and physical properties of a 'conventional' program and the multi-process concept presented here may be seen as a merging of many programs, each for

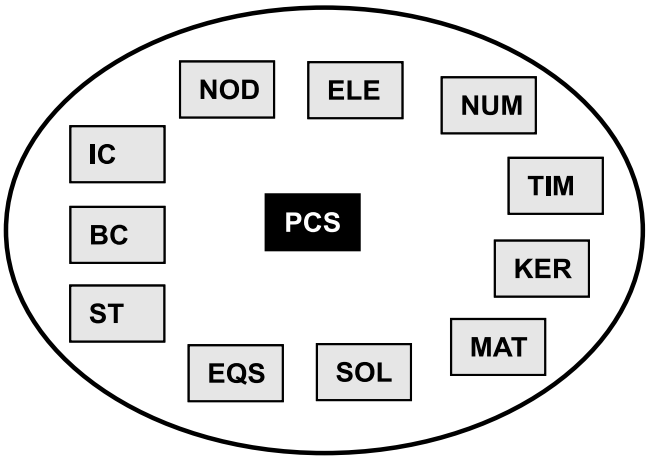

Figure 2 | The PCS object, its members and connected data.

a specific physical problem. Data structures, as well as the grid and the time stepping, are common for all processes, whereas the parameters, boundary conditions and the type of PDE and thus also the solving technique used may vary from process to process.

2.2 PCS object design

Figure 2 gives a graphical illustration of the process (PCS) relationships to other objects. From these objects, information is required to solve a process. Because this information is universal, we take account of it as a set of data by constructing the new PCS object for handling the processes. This object is implemented as a class (see section 3).

The PDE is solved by using a computation grid for the spatial resolution and by applying a time-stepping scheme for the time discretization. Discrete approximation methods for PDEs on grids result in one equation system for each system variable. The general task of a process is, therefore, to construct and solve the corresponding equation system (EQS). The number of equation systems needed for a multi-field problem is defined by the corresponding balance equations that have to be solved to determine the required field quantities. For flow and heat transport processes in porous media, the number of equation systems is equivalent to the number of fluid

phases and the number of transported components involved, whereby temperature is treated as a component. Then the set of PDEs consists of fluid mass and energy conservation equations. In the case of deformable porous medium, a stress equilibrium equation has also to be solved.

2.2.1 PCS member objects

The functionality of the PCS object members, which are illustrated in Figure 2, is as given below.

- EQS: $\boldsymbol{Au} = \boldsymbol{b}$
  Processes must have a related equation system consisting of a system matrix $\boldsymbol{A}$, a solution vector $\boldsymbol{u}$ and the right-hand side (RHS) $\boldsymbol{b}$. For the solution procedure of an algebraic equation system we also need parameters to control the equation solver (SOL). During the solution procedure, the system matrix $\boldsymbol{A}$, solution vector $\boldsymbol{u}$ and RHS vector $\boldsymbol{b}$ will be manipulated according to the specified initial conditions (IC), boundary conditions (BC) and source terms (ST). All this data is related to nodes of the numerical grid. Having several equation systems to deal with, we correlate the data with the corresponding equation system by using names. The EQS name is selected according to the corresponding field quantity, e.g. PRESSURE for fluid flow and TEMPERATURE for heat transport. In the case of multi-phase or multi-component systems, we have to specify these names by adding the related phase or component number, i.e. PRESSURE1 and PRESSURE2 for the first and second fluid phase, respectively.
- NUM:
  The numerics object contains parameters of the numerical method, such as time collocation factors, the number of Gaussian points, upwind parameters, etc. Again, a NUM object will be related to the PCS by name.
- SOL:
  Solver objects enclose parameters controlling the equation solver algorithm, such as the solver method, tolerance criterion, maximum iteration number, etc.
- IC:
  Initial conditions are necessary for any initial value problem. They prescribe values for a certain time point $\boldsymbol{u}(t_0)$. Initial conditions are related to PCS by the process name. IC values correspond to grid nodes.
- BC:
  Boundary conditions are necessary for any boundary value problem. They can be of different kinds, prescribing a value $\boldsymbol{u}(x_0)$, a flux $\nabla\boldsymbol{u}(x_0)$ or a combination of both. Boundary conditions are related to PCS by the process name. BC values correspond to grid nodes.
- ST:
  Source terms give nodal loads to a system. They are directly applied to the RHS vector $\boldsymbol{b}$. Source terms are related to PCS by a name. ST values correspond to grid nodes.
- MAT:
  In general, the system matrix is built by material properties MAT, by shape functions SF and by their derivatives ∇SF. For flow and transport processes in porous media we need fluid, component and medium properties. The number of instances of the fluid, component and medium properties corresponds to the number of specified fluid phases, chemical components and material groups.
- TIM:
  For transient problems we have to apply appropriate time discretization schemes. Time stepping schemes are governed by numerical stability criteria, which depend on the specific type of PDE.
- KER:
  As we use the finite element method for the spatial discretization of the multi-field system, we need special data constructs for finite element matrices, which are linked to the element list (see section 3.2.2).

In the framework of discrete approximation methods, such as the finite element or the finite difference method, processes must be able to access node and element data, in order to save the solution vector $\boldsymbol{u}$ or resultants (e.g. flux vectors). Moreover, for coupled multi-field problems,

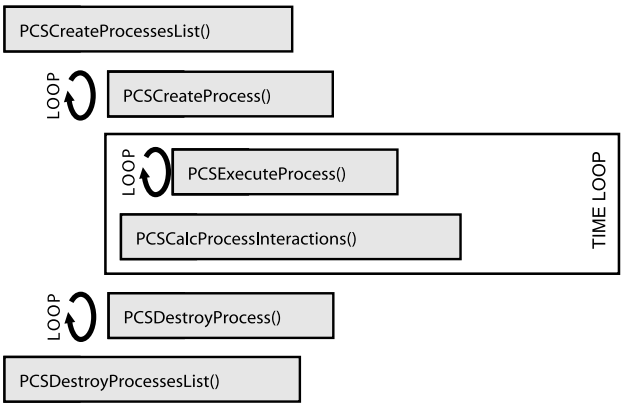

Figure 3 | PCS methods.

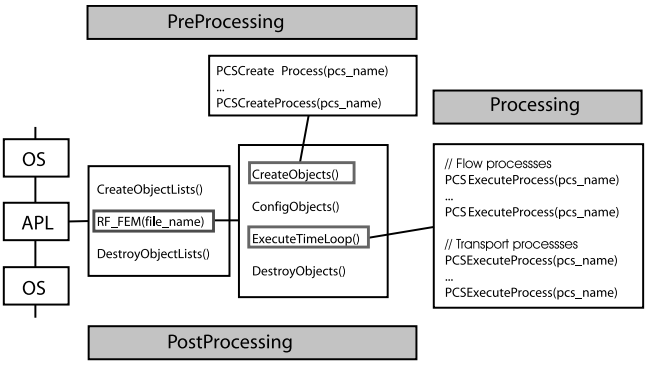

Figure 4 | Implementation scheme of the multi-process algorithm.

processes must have the right to use solution vectors or resultants of the other processes. This has to be implemented by data access functions to nodes (NOD) and elements (ELE) via the node and element lists.

- NOD:
  Each process has to define its related node values (e.g. the field quantity at the old and new time step, etc). Nodal values of all field variables are stored in a vector, which is connected with the node list. This nodal value vector must be flexible in order to account for a variable number of field variables.
- ELE:
  Each process has to define its related element values (e.g. element resultants (fluxes, stresses, strains), characteristic numbers, material properties, etc). Element values of all processes are stored in a vector, which is connected with the element list. This element value vector must also be flexible in order to account for a variable number of values.

2.2.2 PCS methods

After outlining the requirements and the object structure for PCS objects, we turn to the methods associated with them, i.e. the functions of PCS objects. This functionality is determined by the algorithm of the multi-process scheme, which is depicted in Figure 3.

As is visible in Figure 3, first the process list is created, which contains all the processes to be handled during a program run. These processes then have to be created and configured and the connections between all objects have to be set. Then the process can be executed within the time loop by solving the corresponding PDE and obtaining the field variables as well as any additionally wanted values. Within the time loop interactions of processes are also calculated. Functions are needed to create the PCS list as well as the PCS instances themselves. In order to destruct the list and its instances, corresponding destroy functions are required. The symmetry of data construction–destruction is very important for working with dynamic data and is reflected in the structure of Figure 3. This way the general methods for process evaluation become clear:

- Construction and destruction of the process list and the corresponding list operations.
- Process construction and destruction.
- Process execution of the individual process.

3 OBJECT-ORIENTED IMPLEMENTATION

In this section we discuss the object-oriented implementation of the multi-process algorithm.

Figure 4 provides an overview of the implementation scheme. Program execution proceeds from top to bottom, while the level of detail increases from left to right.

In general we distinguish between pre-processing, processing and post-processing steps. OS means the interface to the operational system, which can be DOS, Windows, etc. APL stands for application. Here we deal with a semi-discrete finite element application. This means the finite element procedure is embedded in a time loop, representing the time discretization. During the pre-processing step, any number of processes can be created in CreateObjects, which are organized in the process list. The process list itself is generated in CreateObjectLists. PCS instances have to be specified corresponding to their type (pcs_type), where the type indicates the type of PDE associated with the physical process (see Table 1). Processes can be identified and accessed through their name variable (pcs_name). The configuration of processes is done in ConfigObjects. In the processing step, which is performed within the time loop ExecuteTimeLoop, processes are executed. The corresponding equation systems are assembled and solved and the results are stored in the solution vectors for the output of results and as starting vectors for the next time step.

The program implementation is described below in the order PCS List, PCS Object and PCS Loop. This corresponds to the order of code development.

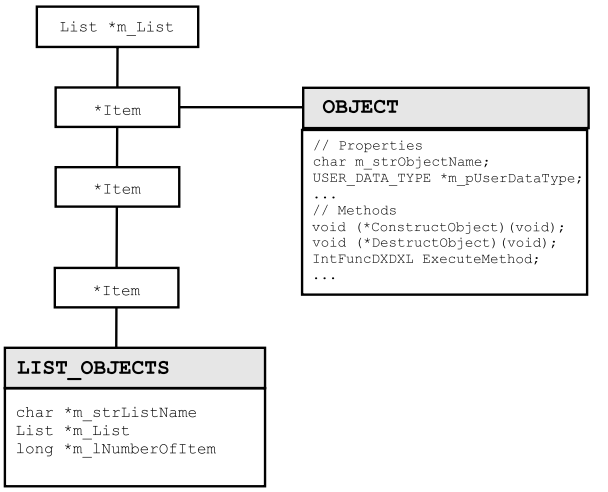

Figure 5 | Object list.

3.1 PCS list

The process list (PCS list) is created in the pre-processing step ('PreProcessing', Figure 4). We use a typical doubly-linked list based on a list template (Figure 5). Properties and methods of the list items have to be specified as described in the following subsections.

3.3.1 PCS list properties

Beside the list itself (pcs_list), general information on the multi-process system has to be stored, which is common to all individual processes: number of processes, number of fluid phases, number of solid phases, number of components and number of material groups. If a new process is created the number of processes is automatically increased. Additionally, the numbers of node and element values of the multi-process system belong to the list properties. Node and element value data are dynamic vectors, depending on the number of processes. The process list construct is given below:

```
typedef struct {
   char *name;
   List *pcs_list;
   long pcs_number_of_processes;
   long pcs_number_of_fluid_phases;
   long pcs_number_of_solid_phases;
   long pcs_number_of_components;
   long pcs_number_of_material_groups;
   int pcs_list_nvals;
   int pcs_list_evals;
}LIST_PROCESSES;
```

3.1.2 PCS list methods

The most important list methods are construction (1) and destruction (2) of the data construct. After processes are created, they have to be inserted into the process list (3). During the processing step, the required PCS instances are obtained from the process list by using a get-function (4). Processes are identified by their name:

```
// (1) - Construction of list -------------
PCSCreateProcessesList()
// (2) - Destruction of list -----------------
PCSDestroyProcessesList()
// (3} - Insert PCS instance into the list ----
PCSInsertProcess(pcs)
// (4) - Get PCS instance by name ------------
PCSGetProcess(pcs->name)
```

3.2 PCS object

The design of processes with their relationships to other objects was discussed in section 2.2. The PCS object must be able to administrate all required data for the complete solution algorithm of a PDE.

3.2.1 PCS object properties

The PCS has private and public data. The most important private property of a process is the connected equation system EQS, while public data reflect the relationship to other objects (Figure 1).

```
class CRFProcess
   {
   . . .
   private:
   \\ EQS - equation system for this process
   public:
   \\ Relationships to other objects
   . . .
}
```

3.2.2 PCS object methods

Typical methods of process instances are PCS creation, configuration and execution. The corresponding functions will be explained in the following.

(i) PCS creation

```
void CRFProcess::PCSCreateProcesses
(CRFProcess *m_pcs)
```

This function creates a PCS instance m_pcs where process name and type have to be specified. Additionally, an equation system EQS is created consisting of a system matrix $\boldsymbol{A}$, a solution vector $\boldsymbol{u}$ and a RHS vector $\boldsymbol{b}$.

(ii) PCS configuration

```
void CRFProcess::PCSConfigProcess(int type)
```

In the first part of the configuration, the connections of the PCS instance to other (existing) objects are established. Connections are needed to numerical properties NUM, solver properties SOL, boundary conditions BC, initial conditions IC, source terms ST and material properties for fluids and components MAT. These connections are established by assigning names to PCS properties, as is shown in Table 2.

It is thus clear that name conventions are very important for the establishment of these connections. As an example, the string variable 'FLOW1' is used for all objects related to the flow process of the first fluid phase. The PCS instance called FLOW1 is looking for all objects with this name (see section 2.2). Therefore, the boundary conditions having this name will be applied to the equation system of the process with the same name, the initial conditions having this name will be used as starting values, and so on for all other objects.

The second part of the PCS configuration concerns the node (i.e. field quantities of processes) and element data (i.e. fluxes of field quantities), which are stored in the node list and element list, respectively. As a first step, vectors for those node and element values have to be configured and created:

```
PCSConfigNODValues();
PCSConfigELEValues();
PCSCreateNODValues();
PCSCreateELEValues();
```

The length of the node and element value vectors must be flexible as it depends on the number of processes.

Processes must have access to node and element data in order to save the results of node values and of element values. Access to node values is also important for nonlinear problems, where material properties (MAT) depend on field quantities. These methods are introduced by

Table 2 | PCS name allocations and function names needed for the execution of a process

```
m_pcs->name                          = name                  // process name (PCS)
m_pcs->primary_function              = pf_name               // primary_variable (PCS)
m_pcs->eqs                           = eq_name               // equation system (PCS)
m_pcs->pcs_num_name                  = num_name              // numerical properties (NUM)
m_pcs->eqs->pcs_sol_name             = sol_name              // solver properties (SOL)
m_pcs->pcs_bc_name                   = bc_name               // boundary conditions (BC)
m_pcs->pcs_ic_name                   = ic_name               // initial conditions (IC)
m_pcs->pcs_st_name                   = st_name               // source terms (ST)
m_pcs->pcs_mat_fp_name               = fp_name               // fluid properties (MAT)
m_pcs->pcs_mat_cp_name               = cp_name               // component properties(MAT)
m_pcs->pcs_mat_soil_name             = sp_name               // soil properties (MAT)
PCSCalcElementMatrices()             = ker_name              // calculate element matrices (KER)
PCSAssembleFunction()                = ker_name              // assemble system matrix (KER)
```

```
PCSConfigNODMethods();
PCSConfigELEMethods();
```

In the context of the finite element method, processes are associated with element matrices (i.e. kernel data). These are configured and created by the following functions:

```
PCSConfigELEMatrices();
PCSCreateELEMatricesPointer();
```

(iii) PCS execution

Finally, after having created and configured the process object PCS instances, they can be executed to set up and solve the equation system. This is done by

```
void CRFProcess::PCSExecuteProcess
(EQUATION_SYSTEM *eqs)
```

The algorithm for process execution consists of the following steps:

1. initialize system matrix $\boldsymbol{A}$, RHS vector $\boldsymbol{b}$ and solution vector $\boldsymbol{u}$,
2. calculate finite element matrices,
3. assemble equation system,
4. solve equation system, and
5. store time step results.

The source code can be written now in a very compact way.

```
void CRFProcess::PCSExecuteProcess
(EQUATION_SYSTEM *eqs)
{
   DisplayMsgLn(``Process: ``, eqs->name);
   // 1 - Initializations --------------------
   // system matrix, RHS vector
   EQSSetZeroLinearSolver(eqs);
   // Get solution vector (initial values)
   NODTransferValues2EQS(PCSGetNODValueIndex
   (this->name),eqs);
```

```
   // 2 - Calc element matrices --------------
   PCSCalcElementMatrices();
   // 3 - Assemble equation system -----------
   PCSAssembleSystemMatrix(eqs->b,eqs->x);
   // 4 - Solve EQS --------------------------
   EQSExecuteLinearSolver(eqs);
   // 5 - Store solution in node values vector
   EQSTransferNODValues(eqs,nidx);
}
```

The function parameter is the equation system connected to the current process: `pcs->eqs`. Note that the above function sets up and solves a PDE, which can have different types (Table 1).

3.3 PCS loop

The application of both the processes according to the process list is done in the `PCS` loop, which is part of the processing step (compare Figure 4).

3.3.1 PCS creation

Processes are generated in the `CreateObjects` function. As an example we depict the source code for creation of a fluid flow process according to the number of fluid components.

```
int CreateObjects(void)
{ . . .
   // Create PCS instances and insert
   // into to PCS list -----
   for(fluid=0;fluid<number_of_fluid_phases;
   fluid++) {
     CRFProcess* pcs=new CRFProcess;
     pcs->type=1; //type=1: flow process
     sprintf(pcs->name,''%s%ld'',''FLOW_'',
     pcs->this_object);
     // Check if PCS exists ------------------
     if(!(pcs->GetProcess(pcs->name))){
       pcs->PCSCreateProcesses(pcs);
       PCSInsertProcess(pcs);
     }
   }
. . .
}
```

For the creation of a process we have to define the type and name of the process. The type (`pcs->type`) is important for process configuration, as it determines whether the process is a flow process or a heat transport process. The name (`pcs->name`) is necessary for process identification and execution. In order to use create functions in dialogue-based applications (see section 4) we need to check whether this process already exists or not. After the process is generated it is inserted into the process list.

3.3.2 PCS execution

In time-dependent problems, such as flow or heat transport problems, the process execution is embedded within a time loop. The following steps are required for the execution of flow and heat transport processes:

1. calculate new time step based on numerical stability criteria,
2. adjust time-dependent boundary conditions and source terms,
3. execute the flow processes according to the number of fluid phases,
4. execute the heat transport processes according to the number of components,
5. calculate resultants such as fluxes and secondary variables, and
6. save time step results for data output and as start vector for the next time step.

The source code illustrating the implementation is given below:

```
int LOPTimeLoop_PCS(void) {
   // (1) - Calculate new time step -----------
   dt=TIMGetDt();
   time_current += dt;
   // (2) - Adjust BCs and STs for current
   //       time step --------
   BCExecuteAllBoundaryConditions();
   STExecuteAllSourceSink();
```

```
    // (3) - Flow processes -------------------
    for(i=0;i<number_of_phases;i++)
      CRFProcess *m_pcs=NULL;
      if(m_pcs=m_pcs->GetProcess(''FLOW''))
        m_pcs->PCSExecuteProcess(m_pcs->eqs);
    }
    // (4) - Heat transport processes ---------
    for(i=0;i<number_of_phases;i++)
      if(m_pcs=
        m_pcs->GetProcess(''HEAT_TRANSPORT''))
      {
        m_pcs->PCSExecuteProcess(m_pcs->eqs);
    }
    // (5) - Calculation of resultants --------
    LOPCalcElementResultants(m_pcs);
    // (6) - Set time step results ------------
    LOPTimeStepResults(m_pcs);
}
```

This procedure is universal for the simulation of multi-field problems and can easily be extended. The two process loops for the execution of the flow processes and the transport processes reflect the physical fact that the results of the flow process execution are needed for calculation of the transport processes. As an example we show the possible extension to include mass transport processes.

```
int LOPTimeLoop_PCS(void) {
. . .
    // 4b - Mass transport processes ----------
    for(i=0;i<number_of_components;i++)
      if(m_pcs=m_pcs->GetProcess
      (''COMPONENT_TRANSPORT'')) {
        m_pcs->PCSExecuteProcess(m_pcs->eqs);
    }
. . .
}
```

This requires adequate creation and configuration of mass transport processes.

4 DIALOGUE-BASED USER INTERFACE

Figure 6 shows the dialogue menu for creating processes. Thermal (T), hydraulic (H), mechanic (M) and chemical (C) processes including mass transport can be selected. Then a specific process type is selected, which corresponds to a specific PDE type (Table 1).

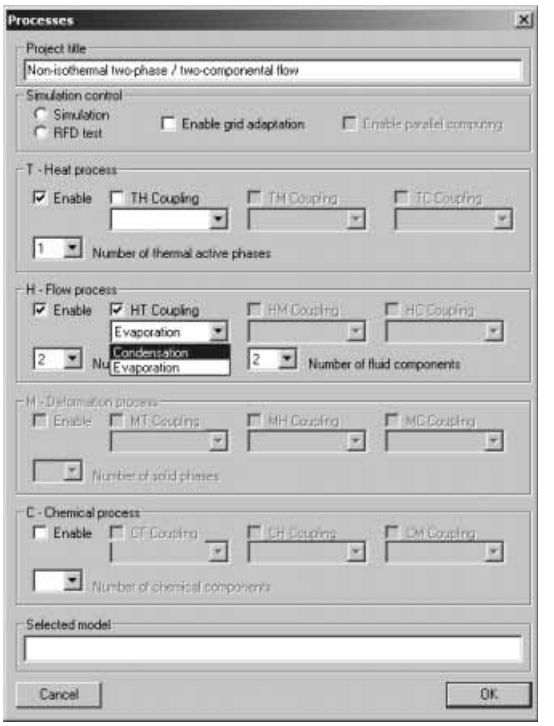

Figure 6 | Dialogue menu of process creation.

The number of fluid phases or thermal active phases chosen results in the generation of a corresponding number of `PCS` instances of the specific process type. Process creation is completed by the `void CWinProcess::OnOK()` function given below.

```
void CWinProcess::OnOK()
{ . . .
   CRFProcess *m_process=new CRFProcess;
   m_process->type=1; // PDE type 1: flow
   sprintf(m_process->name,''%s%ld'',
   ''FLOW_'',m_process->this_object);
   // Check if PCS exists
      if(!(m_process->GetProcess
   (m_process->name))){
     m_process->PCSCreateProcesses
     (m_process);
```

```
      PCSInsertProcess(m_process);
    }
    . . .
}
```

This function is executed if the OK button of the process dialogue is clicked. Then a corresponding message is sent to the operation system OS. In fact, we showed two different implementations for process generation. The dialogue-based implementation of `void CWinProcess::OnOK()` is needed for the Windows interface, whereas the implementation of `int CreateObjects()` given in section 3.3 is for any batch-mode operation OS. The applicability of these methods for different operating systems and platforms is a key demand for object-oriented programming.

5 EXAMPLES

In this section we provide physically different application examples, which demonstrate the usefulness of the presented approach by employing varying numbers of processes and different interactions. The examples include varying numbers of fluid phases and flow as well as heat transport processes. The examples are on very different length scales and include a wide range of material properties. For each example we will provide the governing equations, the initial and boundary conditions as well as the material properties. Furthermore, the representation of the PCS object structure and the process execution is explained in detail.

5.1 Water flow in a porous aquifer—the H process

5.1.1 Problem description, governing equations

As a first example we consider water flow in a porous aquifer, i.e. one H process. Calculations are performed in a strongly heterogeneous aquifer based on real data from the Jordan Valley. Details on the aquifer properties and parameters for this application are given by Beinhorn & Kolditz (2003).

The governing equation for water flow in an aquifer can be derived by using the mass conservation of a single fluid phase (Bear 1972):

$$\left(S_0 + n\frac{\partial \rho^l}{\partial p^l}\right)\frac{\partial p^l}{\partial t} - \nabla \cdot \left(\frac{\mathbf{k}}{\mu^l}\nabla(p^l - \rho^l \mathbf{g})\right) = Q_{\rho^l} \tag{1}$$

where S_0 is specific storativity and n is the porosity of the porous medium, ρ^l is the liquid phase density, $p^l = p^l(\boldsymbol{x},t)$ is the liquid phase pressure, t is time, $\boldsymbol{k}$ is the permeability tensor, μ^l is the liquid viscosity, $\mathbf{g}$ is the gravity vector and Q_{ρ^l} is the liquid phase source term. The equation is of parabolic type (see Table 1).

For each individual application, boundary and initial conditions as well as the specific material parameters have to be specified according to the particular application. This can be done in the input file or by employing the Graphical User Interface (see section 4). Boundary conditions are either constant pressure (Dirichlet) boundary conditions or fixed flow (Neumann) boundary conditions and can be specified as functions of time:

$$\begin{aligned} p^l(\mathbf{x},t) &= f_1(\mathbf{x},t) \quad \text{(Dirichlet)} \\ \nabla p^l(\mathbf{x},t) &= f_2(\mathbf{x},t) \quad \text{(Neumann)}. \end{aligned} \tag{2}$$

Furthermore, initial values of the pressure field are needed to solve the flow problem:

$$p^l(\mathbf{x}, t=0) = p^l_{\text{initial}}(\mathbf{x}). \tag{3}$$

As material MAT parameters for equation (1) the hydraulic permeability tensor $\boldsymbol{k}$, the porosity n and the storativity parameter S_0 need to be specified for the aquifer material. For the fluid phase (liquid), the density ρ^l and the viscosity μ^l have to be given as material parameters.

5.1.2 PCS implementation

In Table 3, the allocations for the H process for this application are shown (compare to Table 2). Additionally, Figure 7 depicts how the PCS concept works for the H process. Object instances specified for the H process are printed in grey.

Table 3 | Name allocations for the FLOW (H) process

```
m_pcs->pcs_name                = ''FLOW1''                          // process name (PCS)
m_pcs->primary_function        = ''PRESSURE1''                      // primary_variable (PCS)
m_pcs->eqs                     = ''PRESSURE1''                      // equation system (PCS)
m_pcs->eqs->pcs_sol_name       = ''PRESSURE1''                      // solver properties (PCS)
m_pcs->pcs_num_name            = ''PRESSURE1''                      // numerical properties (NUM)
m_pcs->pcs_bc_name             = ''PRESSURE1''                      // Dirichlet boundary conditions (BC)
m_pcs->pcs_ic_name             = ''PRESSURE1''                      // initial conditions (IC)
m_pcs->pcs_st_name             = ''SOURCE_MASS_FLUID_PHASE1''       // Neumann boundary conditions (ST)
m_pcs->pcs_mat_fp_name         = ''FLUID_PROPERTIES1''              // fluid properties (MAT)
m_pcs->pcs_mat_cp_name         = ''COMPONENT_PROPERTIES1''          // component properties (MAT)
PCSCalcElementMatrices()       = FMCalcElementMatrices()            // calculate element matrices (KER)
PCSAssembleFunction()          = FMAssembleFunction()               // assemble system matrix (KER)
```

Creation of a PCS instance (PCSCreateProcess()) consists of two steps. First, memory for the equation system EQS (i.e. for the system matrix $\boldsymbol{A}$, the solution vector $\boldsymbol{x}$ and the RHS vector $\boldsymbol{b}$) is allocated. Secondly, the names of the related PCS member variables have to be set in order to link the BC, IC, MAT, KER, etc, objects to the process (see Table 3). The second step is performed by PCSConfigProcess() (see Figure 7). The equation system EQS inherits two names of the PCS object: the process name (FLOW1) and the process primary function PRESSURE1.

During execution of the processes loop, the PCS object with the name identifier FLOW1 is selected from the process list (see Figure 7, left). By two identifiers, i.e. process name and primary variable name, all data required to set up the equation system can be obtained from the related objects. The advantage of the PCS concept is that the very general function PCSExecuteProcess() can be used to execute an arbitrary process (i.e. solve the related equation system). During the initialization (step 1 of the function PCSExecuteProcess(), compare section 3.2.2), the results from the last time or iteration step are obtained via the name of the primary variable PRESSURE1 (Figure 7, column ICList). For calculation of the element matrices (step 2) as well as assembling the equation system (step 3) the corresponding (polymorphic) functions of the process FLOW1 are executed (see Table 3). For calculation of the element matrices the related kernel function FM (Flow Module) must be selected. Again, identification is by the name of the active process (FLOW1) (Figure 7, column KERList). To build the element matrices we need element and material data (Figure 7). Material data are passed via the element ELE object to the finite element kernel FM. To assemble the equation system, boundary conditions have to be incorporated (Figure 7, column BCList). Boundary conditions can be time-dependent: therefore they are also connected to the TIM object. Related data are, as before, identified by the primary variable name of this process. The equation system of this process is thus established and an equation solver is applied (step 4). Finally, the results of the solution vector for the primary variable PRESSURE1 are stored for further usage (step 5).

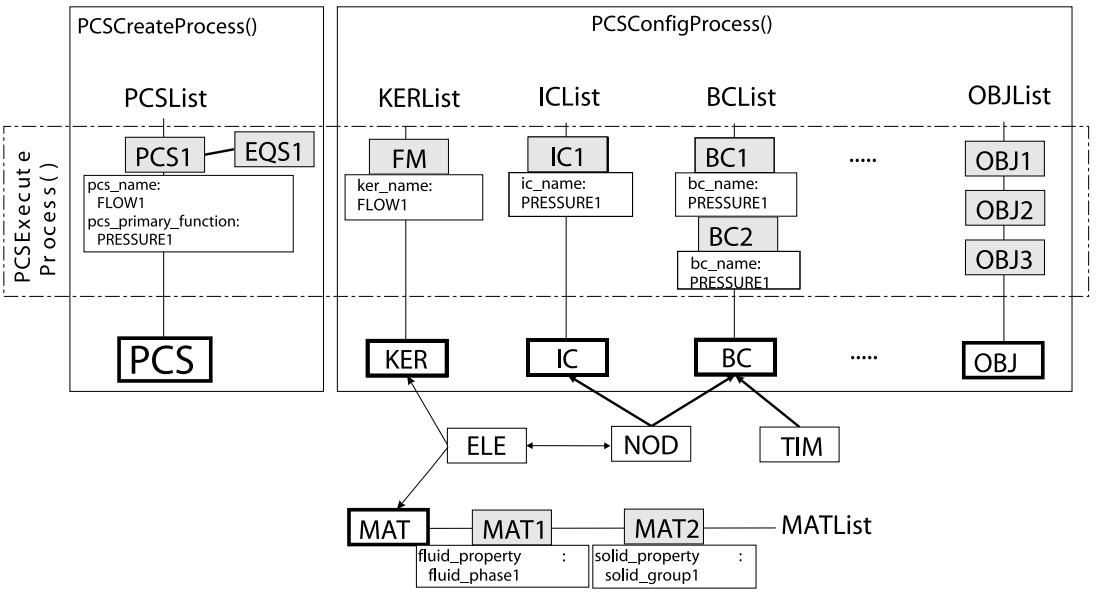

Figure 7 | PCS configuration for FLOW (H) processes.

The material properties of the aquifer are specified in a separate object (Figure 7), as the aquifer material is not process-dependent but rather common to all processes. Note that the process name is 'FLOW1', while the name of the corresponding field variable, i.e. 'PRESSURE1', is used for the names of initial and boundary conditions.

After the flow problem is set up in the manner described, it is solved by employing a standard finite-element technique. A variety of elements from 1D to 3D, as well as a selection of numerical solvers and numerical techniques for the resulting equation systems, are provided in the computer code (Kolditz *et al*. 2003).

5.1.3 Simulation results

Figure 8 shows the model domain in model coordinates and the final distribution of the primary variable PRESSURE1 in top view as a result of the flow process described above. The applied boundary conditions were impermeable boundaries to the North, West and South, representing flow lines and the water divide in the Judean mountains. A constant pressure boundary was set along the east model boundary representing the Jordan river and the Dead Sea. The bottom of the model domain is assumed impervious due to the geological setting, while inflow from the top is specified to represent the spatially varying rain and thus recharge to the aquifer. As a result of this set-up, Figure 8 depicts a pressure distribution which decreases

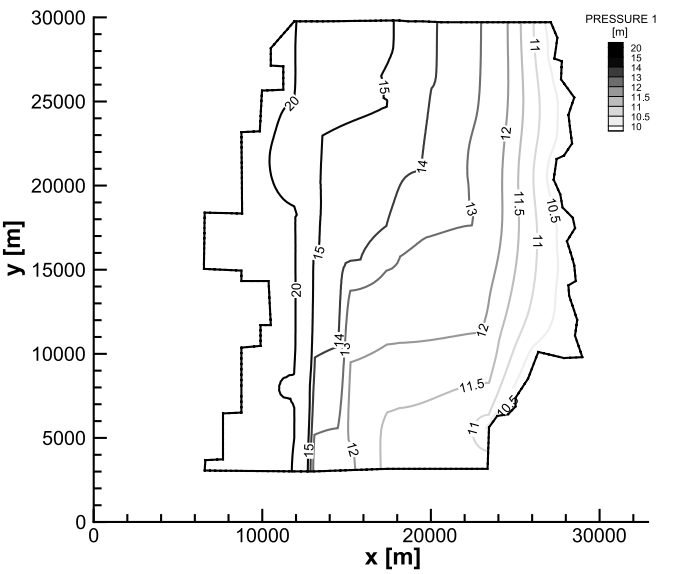

Figure 8 | Distribution of PRESSURE1 in the aquifer. Water flow is from left to right.

from West to East, indicating that groundwater flow is also along this direction.

5.2 Heat transport in a porous and fractured aquifer—the TH process

5.2.1 Problem description, governing equations

In the second example we consider a vertical cross section of the same aquifer in the Jordan area described above. In this area, hot spots are observed resulting from thermal water up-coming through vertical fractures. The flow problem has now to be solved in a 2D porous and fractured aquifer. The fractures are preferential pathways for water and are represented by 1D line elements, which are given a high permeability. Otherwise, equations (1)–(3) are valid in this case as well. In addition to water flow, heat transport (T process) has to be modelled in this application and has to be coupled to the flow process. This second application is thus an example for two coupled processes.

For the case of heat transport, the governing equation can be obtained based on the heat balance equation for the porous medium consisting of two phases, i.e. the solid and the liquid phase. The following thermal energy equation for the temperature T is obtained (Kolditz 2002):

$$\left((1-n)c^s\rho^s + nc^l\rho^l\right)\frac{\partial T}{\partial t} + \underbrace{c^l\rho^l n\,\mathbf{v}\nabla T}_{\text{HT coupling term}} - \left((1-n)\lambda^s + n\lambda^l\right)\nabla^2 T = Q_T \quad (4)$$

where c is the thermal capacity, λ is the thermal conductivity, $\boldsymbol{v}$ is the fluid velocity and Q_T is the heat source term. The superscripts l and s refer to the liquid and the solid phase, respectively.

Coupling between the two processes is twofold. The temperature distribution in the aquifer influences some material parameters, as, for example, fluid density and fluid viscosity. Warm water has a lower density and thus rises in comparison to colder water. Thus the temperature induces fluid movement. On the other hand, the pressure distribution calculated yields the fluid velocities $\boldsymbol{v}$, which in term influences the advective transport of heat in the fluid phase. This term is designated the 'HT coupling term' in equation (4) and designates the coupling to equation (1).

As for the H process, boundary and initial conditions have to be specified for the T process, analogous to equations (2) and (3). Additional material parameters needed are the thermal conductivities and heat capacities of the aquifer material and the fluid.

5.2.2 PCS implementation

The PCS allocations for the T process for this application are then as given in Table 4 (compare to Table 2). Additionally, Figure 9 depicts how the PCS concept works with the additional T process.

Again, as for the FLOW1 process, creation of a PCS instance (using PCSCreateProcess()) consists of two steps. First, memory for the equation system EQS for HEAT_TRANSPORT with the primary function TEMPERATURE is allocated. Second, the names of the related PCS member variables have to be set in order to link the BC, IC, MAT, KER, etc, objects to the HEAT_TRANSPORT process (see Table 4). The second step is performed by PCSConfigProcess(), as shown in Figure 9. Objects specified already for the H process are printed in white, objects added for the T process to the corresponding list for BC, IC or KER objects are printed in grey.

During execution of the process loop, the PCS objects with the name identifier FLOW1 as well as HEAT_TRANSPORT are now selected from the process list (see Figure 9, left). For the execution of the H process, all is as described in the previous section. Additionally, when the T process is executed, all data required to set up the equation system for HEAT_TRANSPORT is gathered by process and primary variable names.

For calculation of the element matrices as well as assembling the equation system the corresponding functions of the process HEAT_TRANSPORT are executed (see Table 4). For calculation of element matrices the related kernel function (HTM: Heat Transport Module) is selected (Figure 9). To build the element matrices we need additional material data (Figure 9). Therefore an instance of the MAT object is added to the MATList which contains

Table 4 | Name allocations for the HEAT_TRANSPORT (T) process

```
m_pcs->pcs_name                 = ''HEAT_TRANSPORT1''          // process name (PCS)
m_pcs->primary_function         = ''TEMPERATURE1''             // primary_variable (PCS)
m_pcs->eqs                      = ''TEMPERATURE1''             // equation system (PCS)
m_pcs->eqs->pcs_sol_name        = ''TEMPERATURE1''             // solver properties (PCS)
m_pcs->pcs_num_name             = ''TEMPERATURE1''             // numerical properties (NUM)
m_pcs->pcs_bc_name              = ''TEMPERATURE1''             // Dirichlet boundary conditions (BC)
m_pcs->pcs_ic_name              = ''TEMPERATURE1''             // initial conditions (IC)
m_pcs->pcs_st_name              = ''SOURCE_HEAT_PHASE1''       // Neumann boundary conditions (ST)
m_pcs->pcs_mat_fp_name          = ''FLUID_PROPERTIES1''        // fluid properties (MAT)
m_pcs->pcs_mat_cp_name          = ''COMPONENT_PROPERTIES1''    // component properties (MAT)
m_pcs->pcs_mat_soil_name        = ''SOIL_PROPERTIES1''         // heat parameters (MAT)
PCSCalcElementMatrices()        = HTMCalcElementMatrices()     // calculate element matrices (KER)
PCSAssembleFunction()           = HTMAssembleFunction()        // assemble system matrix (KER)
```

the parameters needed for the calculation of heat transport (compare equation (4)). The equation system of the HEAT_TRANSPORT process is thus established and an equation solver is called to solve it.

5.2.3 Simulation results

Figure 10 shows the vertical temperature distribution for such a hot spot. This graph depicts the preferential heat flow along a vertical fracture, which is located at $x = 255{,}500$ in Figure 10. Aquifer layers are represented by 2D rectangular elements and the fracture zone is represented by 1D line elements. The up-coning effect of warm water at the fracture can be clearly seen by the higher temperatures along the vertical fracture. Vertical heat transport in the model area without fractures is much less pronounced, which shows the relative importance of heat transport by water movement compared to heat conduction in the aquifer material and stagnant water. More details can again be obtained from Beinhorn & Kolditz (2003).

5.3 Non-isothermal two-phase flow in buffer materials—H^2T process

5.3.1 Problem description, governing equations

The third example of the process-oriented approach to multi-field problems is dealing with the non-isothermal two-phase flow of water and air through geotechnical buffer materials such as bentonite. In this case we use two H and one T process. The mathematical formulation of this multi-field problem consists of three partial differential equations derived from the mass balances of the fluid components (air, water) and the thermal energy balance. In addition to the above examples, phase transitions now have to be included in the equations.

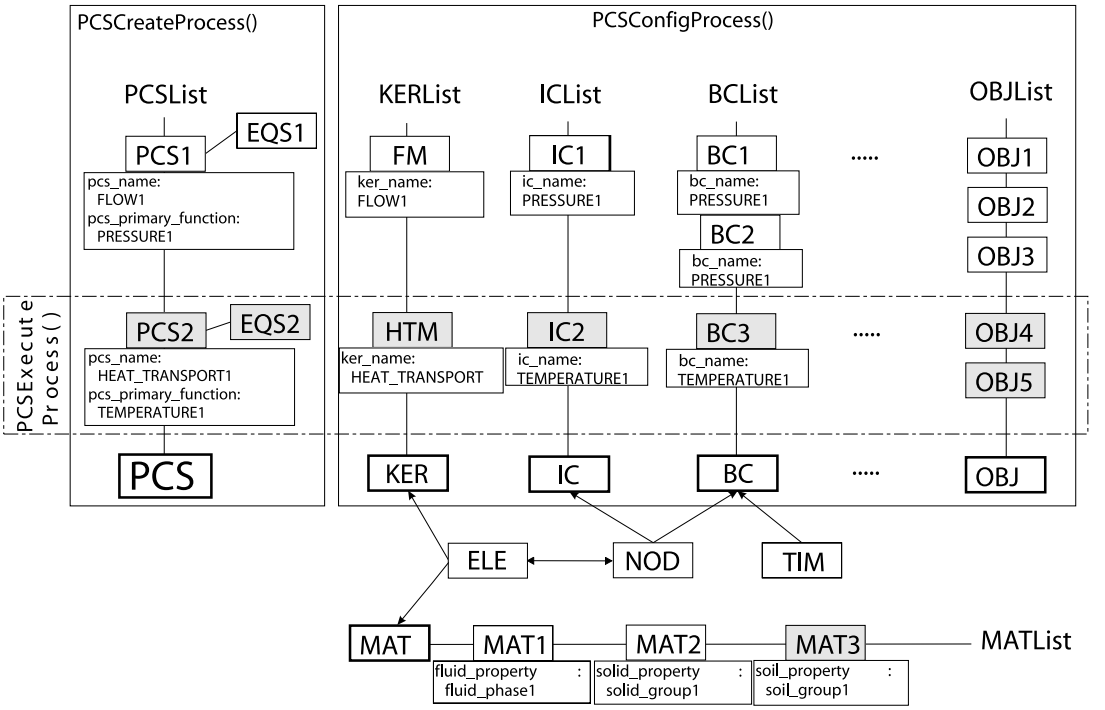

Figure 9 | PCS configuration for the FLOW1 (H) and the HEAT_TRANSPORT (T) processes.

For this example we consider two-phase/two-component flow. The fluid phases are gas (g) and liquid (l). The fluid phases consist of air ($k=a$) and water ($k=w$) components. As a result of phase changes (i.e. evaporation and gas solution) both components can be present in both phases. Balance equations for the fluid components (air and water) are used in order to avoid explicit phase change terms. Phase changes appear in the mass fraction terms and are considered based on the Clausius–Clapeyron equation for vapour pressure and the Henry equation for gas dissolved in the liquid phase. A derivation of the following equations can be found in Kolditz & de Jonge (2003):

$$
\begin{aligned}
&nX_k^g\left(S^g\frac{\partial\rho^g}{\partial p^g}+S^l\frac{\partial\rho^l}{\partial p^l}\right)\frac{\partial p^g}{\partial t}-\nabla\cdot\left(\rho^g X_k^g\frac{k_{\mathrm{rel}}^g\mathbf{k}}{\mu^g}\nabla p^g\right)\\
&-\nabla\cdot\left(\rho^l X_k^l\frac{k_{\mathrm{rel}}^l\mathbf{k}}{\mu^l}\nabla p^g\right)+n\left(-\rho^g X_k^g+\rho^l X_k^l\right)\frac{\partial S^l}{\partial t}\\
&=Q_k+\left(nS^l X_k^l\frac{\partial\rho^l}{\partial p^l}\right)\frac{\partial p_c}{\partial t}-\nabla\cdot\left(\rho^l X_k^l\frac{k_{\mathrm{rel}}^l\mathbf{k}}{\mu^l}\nabla p_c\right)\\
&-\nabla\cdot\left(\rho^g X_k^g\frac{k_{\mathrm{rel}}^g\mathbf{k}}{\mu^g}\rho^g\mathbf{g}\right)+\nabla\cdot\left(\rho^l X_k^l\frac{k_{\mathrm{rel}}^l\mathbf{k}}{\mu^l}\rho^l\mathbf{g}\right)\\
&+\underbrace{\nabla\cdot\left(nS^g\rho^g D_k^g\nabla X_k^g\right)}_{\text{TH coupling term}}
\end{aligned}
\tag{5}
$$

where X_k^γ is the mass fraction of component k in fluid phase γ, S^γ is the saturation of fluid phase γ, ρ^γ is the density of fluid phase γ, k_{rel}^γ is the relative permeability of fluid phase γ, D_k^g is the diffusion coefficient of component k in the gas phase and p^c is the capillary pressure.

The last term in equation (5) represents the vapour diffusion and constitutes the coupling to heat transport (TH coupling). Vapour is mainly transported due to thermal gradients. Coupling is also due to the dependences of material and state functions such as density, viscosity, capillary pressure, vapour pressure and mass fractions on the primary variables: gas pressure p^g, liquid saturation S^l and temperature T.

Based on the heat balance equation for the porous medium consisting of three phases (solid, gas, liquid), we

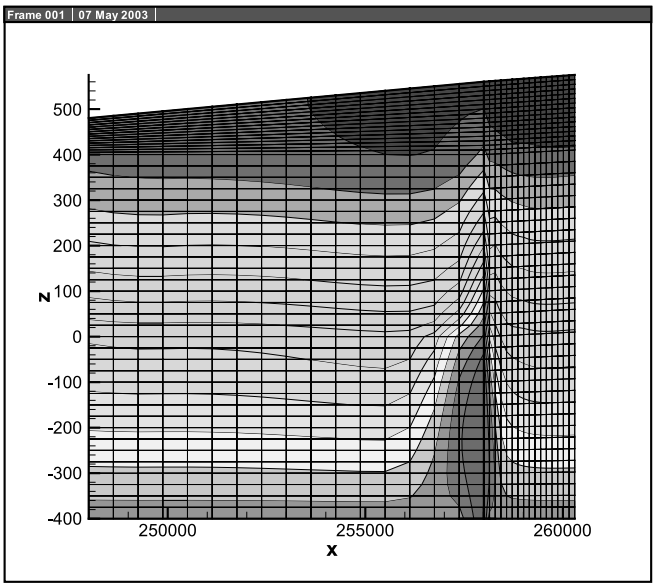

Figure 10 | Vertical temperature distribution in the aquifer.

obtain the following thermal energy equation (Kolditz & de Jonge 2003):

$$\left((1-n)\rho^s c^s + nS^g \rho^g c^g + nS^l \rho^l c^l\right)\frac{\partial T}{\partial t} + \underbrace{n\left(\rho^g c^g \mathbf{v}^g + \rho^l c^l \mathbf{v}^l\right)\nabla T}_{\text{HT coupling term}} - \nabla \cdot \left(\left((1-n)\lambda^s + nS^g \lambda^g + nS^l \lambda^l\right)\nabla T\right) = \rho Q_T + \underbrace{\nabla \cdot \left(nS^g \rho^g D_a^g h^g \nabla X_a^g\right) + \nabla \cdot \left(nS^g \rho^g D_w^g h^g \nabla X_w^g\right)}_{\text{TH coupling term}} \quad (6)$$

where c^γ is the specific heat capacity of phase γ, T is the temperature, λ^γ is the thermal conductivity of phase γ, h^γ is the enthalpy of fluid phase γ and Q_T is the heat source term.

5.3.2 PCS implementation

While the PCS allocations for the first H process and the T process are as specified in the previous examples, an

Table 5 | Name allocations for the FLOW2 (H) process

```
m_pcs->pcs_name                = ''FLOW2''                          // process name (PCS)
m_pcs->primary_function        = ''SATURATION2''                    // primary_variable (PCS)
m_pcs->eqs                     = ''SATURATION2''                    // equation system (PCS)
m_pcs->eqs->pcs_sol_name       = ''SATURATION2''                    // solver properties (PCS)
m_pcs->pcs_num_name            = ''SATURATION2''                    // numerical properties (NUM)
m_pcs->pcs_bc_name             = ''SATURATION2''                    // Dirichlet boundary conditions (BC)
m_pcs->pcs_ic_name             = ''SATURATION2''                    // initial conditions (IC)
m_pcs->pcs_st_name             = ''SOURCE_MASS_FLUID_PHASE2''       // Neumann boundary conditions (ST)
m_pcs->pcs_mat_fp_name         = ''FLUID_PROPERTIES2''              // fluid properties (MAT)
m_pcs->pcs_mat_cp_name         = ''COMPONENT_PROPERTIES2''          // component properties (MAT)
m_pcs->pcs_mat_soil_name       = ''SOIL_PROPERTIES1''               // heat parameters (MAT)
PCSCalcElementMatrices()       = MPCCalcElementMatrices()           // calculate element matrices (KER)
PCSAssembleFunction()          = MPCAssembleFunction()              // assemble system matrix (KER)
```

additional PCS allocation for the second H process for this application is needed (compare to Table 2), as shown in Table 5. In analogy to Figures 7 and 9, a third line has to be added for the third process (second H process) to link the corresponding BC, IC, MAT, KER, etc, objects to the FLOW2 process.

These two-component/two-phase flow equations are highly non-linear and strongly coupled. Details of the continuum–mechanical model and the numerical solution procedure are described in de Jonge *et al*. (2003) and Kolditz & de Jonge (2004).

5.3.3 Simulation results

The example considers a one-dimensional probe of bentonite buffer material of length 0.5 m. Initial conditions are a temperature of 283 K, full fluid saturation and pressure equilibrium. Boundary conditions are a fixed pressure of 100,000 Pa at $x = 0.5$, a constant temperature of 373 K at $x = 0$ and otherwise free outflow boundaries. Further details of the material parameters are given by de Jonge *et al*. (2003). The temporal evolution of the primary variables gas pressure of fluid phase one (PRESSURE1), liquid saturation of the second fluid phase (SATURATION2) and temperature (TEMPERATURE1) is depicted in Figure 11 in response to the heating at $x = 0$. Near the heating boundary, water is evaporating, which causes a decrease in saturation as well as an increase in gas pressure. Due to the high water retention capacity of the buffer material, the fluid conductivity of the bentonite is small and thus the saturation and pressure fronts advance only slowly along the buffer material. Heat transport is also by conduction through the bentonite and thus faster, as compared to fluid movement.

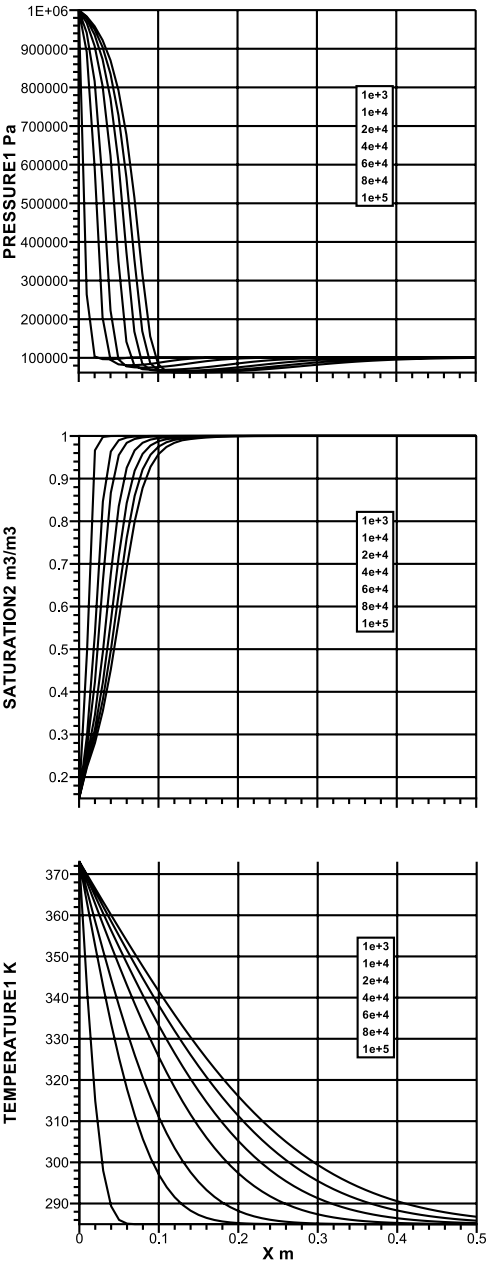

Figure 11 | Profiles of primary variables along the buffer material: (a) gas pressure, (b) liquid saturation, (c) temperature (from top to bottom) for time points: 10^3, 10^4, 2×10^4, 4×10^4, 6×10^4, 8×10^4 and 10^5 seconds (curves from left to right).

6 CONCLUSIONS

This paper describes the concept of an object-oriented approach for modelling multi-field problems in porous media. The basic idea is the direct correlation of a physical process, corresponding to one field variable and one differential equation, and the process object (PCS) of the numerical code. Therefore we termed this a 'process-oriented approach'. The design and the implementation into an object-oriented C++ code with the respective data structures and functions needed is given in detail. The short and concise way in which the execution of a process

and thus the solution of the corresponding field equation can be implemented allows for an easy extension of the program to further multi-field problems. This could be demonstrated by the examples given. We believe that this concept is a valuable contribution to the object-oriented development and will help in adapting the code to more complex, and thus more realistic, tasks, typically represented by a large number of field variables and equations. Future program development will include the extension of the presented concept to multi-component reactive mass transport as well as deformation processes.

ACKNOWLEDGEMENTS

The authors would like to thank the Geohydrology/Hydroinformatics (GHI) team in Tübingen and Hannover for the enthusiastic work on the RockFlow/GeoSys project (www.rockflow.net). We appreciate very much the helpful comments of Chris McDermott and Wenqing Wang on the manuscript. We are grateful to the German Federal Ministry of Education and (BMBF) for funding this work.

REFERENCES

Akin, J. E. 1999 Object oriented programming via Fortran 90. *Int. J. Computer-Aided Engng.* **16**, 26–48.

Bear, J. 1972 *Dynamics of Fluids in Porous Media*. Elsevier, New York.

Beinhorn, M. & Kolditz, O. 2003 Density dependent flow in unconfined aquifers – application to the Jordan Valley. *RockFlow Report* 2003-13, Center for Applied Geosciences, University of Tübingen.

Budd, T. 2002 *Object-oriented Programming* 3rd edn. Addison Wesley, Boston.

Cross, J. T., Masters, I. & Lewis, R. W. 1998 Why you should consider object-oriented programming techniques for finite element techniques. *Int. J. Numer. Meth. Heat Fluid Flow* **9**, 333–347.

de Jonge, J., Xie, M. & Kolditz, O. 2003 Numerical implementation of thermally and hydraulically coupled processes in non-isothermal porous media. *Rockflow Report* 2003-11, Centre for Applied Geoscience, University of Tübingen.

Desitter, A., Bates, P. D., Anderson, M. G. & Hervouet, J. M. 2000 Development of one, two and three-dimensional finite element groundwater models within a generalized object-oriented framework. *Hydrol. Process*. **14**, 2245–2259.

Dubois-Pelerin, Y. & Zimmermann, T. 1993 Object-oriented finite element programming 3. An efficient implementation in C++. *Comp. Meth. Appl. Math. Engng.* **108**, 165–183.

Feng, Z. Q. 1995 2d or 3d frictional contact algorithms and applications in a large deformation context. *Commun. Numer. Meth. Engng.* **11**, 409–416.

Habbar, A. 2001 Direkte und inverse geochemische Mehrkomponentenmodelle. *PhD thesis*, Universität Hannover.

Kolditz, O. 1999 Hydroinformatics concepts in subsurface modeling – object-oriented methods. In *International Conference on Water in the Environment* (ed. Taniguchi, T. *et al*.), pp. 11–25. Okayama University Press, Okayama, Japan.

Kolditz, O. 2002 *Computational Methods in Environmental Fluid Mechanics*. Springer, Berlin.

Kolditz, O., de Jonge, J., Beinhorn, M., Xie, M., Kalbacher, T., Wang, W., Bauer, S., McDermott, C., Kaiser, R. & Kohlmeier, M. 2003 *ROCKFLOW – Theory and User Manual* release 3.9 (in preparation). Technical report, Groundwater Modelling Group, Center for Applied Geosciences, University of Tübingen; Institute of Fluid Mechanics, University of Hannover.

Kolditz, O. & de Jonge, J. 2004 Non-isothermal two-phase flow in porous media. *Comput. Mech*. **33**(5), 345–364.

Mackie, R. I. 1992 Object-oriented programming of the finite element method. *Int. J. Numer. Meth. Engng.* **35**, 425–436.

Masters, I., Cross, J. T. & Lewis, R. W. 1997 A review of object oriented programming techniques in finite element methods. In *Proc. of the 10th International Conference for Numerical Methods in Thermal Problems* (ed. Lewis, R. W. & Cross, J. T.), pp. 766–776. Pineridge Press Swansea, UK.

Mentrey, P. & Zimmmermann, T. 1993 Object-oriented non-linear finite element analysis – application to J2 plasticity. *Comput. Struc.* **49**, 767–777.

Ohtsubo, H., Kawamura, Y. & Kubota, A. 1993 Development of the object-oriented finite element modelling system Modify. *Engng. Comput.* **9**, 187–197.

Peskin, A. P. & Hardin, G. R. 1996 An object-oriented approach to general-purpose fluid-dynamics software. *Comput. Chem. Engng.* **20**, 1043–1058.

Pidaparti, R. V. M. & Hudli, A. V. 1996 Dynamic analysis of structures using object-oriented techniques. *Comput. Struc*. **49**, 149–156.

Sampath, R. & Zabaras, N. 2000 An object-oriented framework for the implementation of adjoint techniques in the design and control of complex continuum systems. *Int. J. Numer. Meth. Engng.* **48**, 239–266.

Scholz, S. P. 1992 Elements of an object-oriented FEM++ program in C++. *Comput. Struc.* **43**, 517–529.

Stroustrup, B. 1991 *The Programming Language C++*. Addison-Wesley, Reading, MA.

Thorenz, C., Kosakowski, G., Kolditz, O. & Berkowitz, B. 2002 An experimental and numerical investigation of saltwater movement in partially saturated systems. *Wat. Res. Res.* **38** (6), 1029.

Vanhooren, H., Meirlaen, J., Amerlinck, Y., Claeys, F., Vangheluwe, H. & Vanrolleghem, P. 2003 West: Modelling biological wastewater treatment. *J. Hydroinf.* **5**(1), 27–50.

Zimmermann, T., Dubois-Pelerin, Y. & Bomme, P. 1992 Object-oriented finite element programming 1: Governing principles. *Comput. Meth. Appl. Mech. Engng.* **98**, 291–303.