WORLD HEAVY OIL CONGRESS 2015
Edmonton, Alberta, Canada

WHOC15 – 167

NEW TOOLS FOR NEW TECHNOLOGY – OASIS: MULTIPHYSICS ENGINEERING SIMULATION SOFTWARE FOR "WHAT-IF PHYSICS"

Carolina Diaz-Goano (Laricina Energy Ltd.), Achara Tiong (Laricina Energy Ltd.), Heather Herring (Laricina Energy Ltd.)

ABSTRACT

Reservoir simulation models are used extensively for comparative studies and forecasting performance. As more complex reservoirs are developed, the understanding of the physical phenomena occurring *in situ* needs to be reconsidered. Incorporating new physics in commercial simulators is very time consuming and software companies cannot provide the average user flexibility on the model formulation at a reasonable cost. Therefore, Laricina developed OASIS, a multiphysics engineering simulation software that enables physics equations to be used as plug and play components. It is a living system that evolves incrementally, using interchangeable and extensible models to develop and modify complex formulations with minimal effort, making the studies of "what-if physics" possible. The OASIS Modeling Language (OML) and compiler translate equations into a working model and provide analytic tools to check the consistency of the system and examine the problem formulation. The model formulation, numerical engine, user interfaces and other components are each defined by an abstract programming interface. Incorporated features ensure that only physically plausible answers are calculated, boundary conditions can be set and any condition can be defined to stop the simulation. Different linear and non-linear solvers are integrated including open source code developed at the University of Florida and the French Institute for Research in Computer Science and Automation (INRIA) with the flexibility to incorporate others. The choice of equations is transparent and changes are auditable. Input data can be loaded through the OML script or an external ASCII file. The ability to selectively record any or all computational output is incredibly powerful for debugging. All results are saved into a database thus enabling cross plotting and analysis during or after the simulation. Database queries are done through a filtering mechanism. The cloud based capability allows scalability without additional infrastructure investment, unlimited storage, system integrity backup and recovery.

KEY WORDS

Software, Simulation, Multiphysics Modeling, Reservoir Engineering

INTRODUCTION

Simulation is a powerful way to analyze, design and evaluate complex systems. It is a cost effective way to explore new processes and technology and to de-risk decision making; it is a necessary companion to any laboratory, field pilot and production process. Simulation increases the understanding of a process through a stepwise approach. OASIS (**O**pen **A**ccess **S**ystem for **I**ntegrated **S**imulation) is Laricina's tool for performing analytical calculations and engineering modeling. OASIS was motivated by the limitations of commercial software and inspired by the necessity to modify and extend reservoir models to test new ideas, develop new technology and to reduce computing cost by adhering to the philosophy of computing on demand. OASIS consists of a language (OML), a generic numerical engine, and user interfaces for development, execution and result visualization. OASIS handles linear and non-linear systems of equations for static and time-dependent problems along with constrained and unconstrained optimization problems. The equations can be associated with or without a grid. Numerical models consist of user-defined equations expressed in OML syntax. The main difference with other engineering simulation software is that OASIS users have full access and control of the modeling equations that the software provides as a set of stackable "drivers" that can be used in an interchangeable fashion to solve the system in question. Since the main target users are engineers and scientists, parameters are defined with a type (for example temperature, pressure, viscosity, etc.) and the

value and units are provided. In OASIS the units correspond to the declared type and are internally converted to SI.

OASIS follows the object-oriented paradigm to enable the reusability and extension of modeling components. In a plug and play fashion, modeling components can be used by themselves or as part of a larger model. For example, the Flash Calculation Model can be used by itself or as part of a SAGD model.

Cells are the fundamental building blocks in OASIS. A cell has members that can be parameters, variables or equations. Cells can have instances of other cells and are designed following a hierarchy. OASIS permits encapsulation. The system is divided into modular cells with their own data and behavior. Cell inheritance was implemented making the cell reusable and its structure extensible. Subtype polymorphism was also implemented, in this way a child cell can override the definition of a parent's member. For example, a "phase" can be the higher level of "oil phase," "gas phase" and "water phase." Figure 1 shows an extraction of an OASIS SAGD model illustrating some of these concepts.

```
cell Gas extends Phase {
    @PhaseFlux     fluxType = GasFlux;
    Density        phsDen   = sum( pure[].x * pure[].pRo), unit kg*m^-3;
    Viscosity      vis      = sum( pure[].x * (pure[].comp.molWt * 1 mol/kg )^0.5*pure[].vis )
                            / sum( pure[].x * (pure[].comp.molWt * 1 mol/kg )^0.5), unit cp;
    TConductivity  kt       = con_kg, unit kJ*m^-1*day^-1*K^-1;
}

cell Oil extends Phase {
    @PhaseFlux     fluxType = OilFlux;
    Density        phsDen   = 1 / sum( pure[].x / pure[].pRo), unit kg*m^-3;
    Viscosity      vis      = exp(sum( pure[].x * ln(pure[].vis / 1Pa*s ) ) )*1Pa*s, unit cp;
    TConductivity  kt       = con_ko, unit kJ*m^-1*day^-1*K^-1;
}

cell Water extends Phase {
    @PhaseFlux     fluxType = WaterFlux;
    Density        phsDen   = 1 / sum( pure[].x / pure[].pRo), unit kg*m^-3;
    Viscosity      vis      = sum( pure[].x * pure[].vis), unit cp;
    TConductivity  kt       = con_kw, unit kJ*m^-1*day^-1*K^-1;
}
```

Figure 1. Example of extension and polymorphism.

OASIS has its own approach to sharing data between objects and the model as a whole. Each cell can have a link to another cell making its information accessible to each other.

Since OASIS is a collaborative tool to aid new technology development, models, scripts, tables and files can be shared through a virtual file system. This file system is part of the OASIS developing environment, shown in Figure 2. Results from an executed job are stored in a database. Those results can be accessed during or after a simulation run. A version control system tracks the job status, original script, expanded script (all files invoked by the original scripts are automatically appended to the expanded script) and visualization of any output requested for that job, thus providing enhanced auditing capabilities. Also, the developing environment has search capabilities to look for a specific job descriptor or a text within a script.

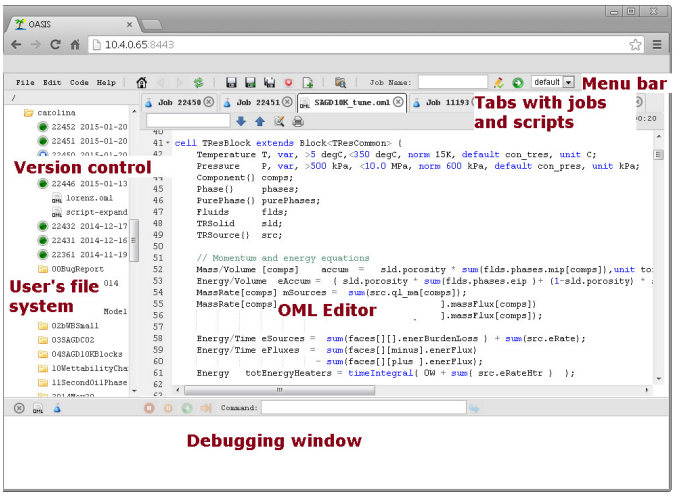

Figure 2. OASIS developing environment

OASIS ARCHITECTURE

OASIS design was based on the separate encapsulation of its three main components: *Interfaces*, *Drivers* and *Models* (Figure 3). The Interfaces provide the controls and display features to create scripts and visualize results by using a common engine to drive multiple usage applications. The Drivers (such as Case Driver, Recording Driver, Non-linear Driver, etc.) can be understood as a stack that provides the desired functionality. Finally, the Model is the mathematical formulation the user wants to solve, which describes the numerical interpretation of physical/chemical process of interest. OASIS Modeling Language (OML) was created for writing models where variables, equations, parameters and functions represent the problem to solve. Cells are simple self-assembling, re-usable objects which contain members accessible to the user model.

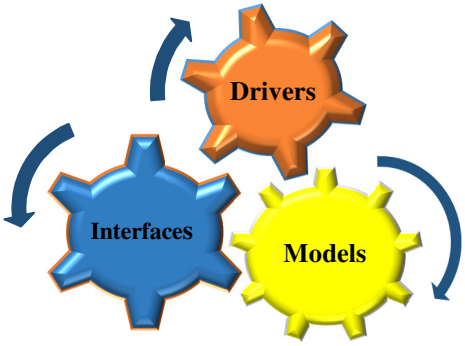

Figure 3. OASIS architecture

OASIS DRIVERS

In general, a Driver is a system-supplied cell that is installed on top of a user model that manages and/or accumulates the results of the model in structures below it. From the viewpoint of the user interface, a Driver behaves like a Model in the sense that it exports a table of members with all available results. The members are the same as in the model one hierarchical level below except that they may be annotated (i.e., further indexed) to distinguish between multiple solutions (such as individual time steps). A Driver can be thought of as a filter on the model interface, where the inputs are the indexed members of the child model, and the outputs are the modified and/or augmented members presented by the driver to the interface.

Drivers may be combined in series or operated in parallel as required, depending on the model contents and user objectives. Any model containing one or more equation requires a Non-linear Driver to find solutions. If the model is dynamic, a Time Driver is required as well in order to step the model forward in time.

Case Driver

The purpose of the Case Driver is to load and run a model with the appropriate communication between the solution and other drivers. The Case Driver is responsible for setting the model name parameters, initializing variables and ultimately commanding the execution of the run. It interacts with the Recording Driver to record variables and states at prescribed frequencies (for restarting or reporting purposes). It is also responsible for the loading of any restart case; restoring variables and statuses and running the model (step forward under the control of "runScript").

Non-Linear Driver

Non-linear problems are solved using Newton-Raphson (NR) algorithm. Several preconditioners (e.g., ILU0, ILUT) and linear solvers (e.g., LU, Conjugate Gradient, GMRES, BiCGStab) are available. The source code developed in-house was augmented with open source code. There are parallel capabilities on selected items. The user can specify the preconditioner and/or linear solver. For each solver the tolerances, maximum iterations, etc. can be set within the model. An important difference from other simulation packages is that the model can contain more than one system of equations. Grouping the equations in different systems was an innovative design tailored to take advantage of the different characteristics smaller sub-models could have. For example, linear vs. non-linear, or when the solution could be achieved through a split step method.

Time Driver

The Time Drivers provide the time stepping functionality for dynamic models. Time discretization is done through an implicit Euler scheme. Time dependent equations are expressed through the "timeDeriv" keyword in OML which indicates to the engine that a time stepping algorithm should be used. Figure 4 shows a simple example for solving Lorenz attractor problem [1] using OASIS.

```
cell Lorenz {  // http://en.wikipedia.org/wiki/Lorenz_system
  * x, var, norm 0.1, default 0, record;
  * y, var, norm 0.1, default 20, record;
  * z, var, norm 0.1, default 20, record;

  eqn one timeDeriv(x) = (1e-2 s^-1)*(-10*x+10*y);
  eqn two timeDeriv(y) = (1e-2 s^-1)*(28*x-y-x*z);
  eqn tre timeDeriv(z) = (1e-2 s^-1)*(-8*z/3+x*y);
}

set TimeDriver.maximumSteps=10000 steps

stepto 4000 steps

plot "Lorenz"(*) {
   line  y vs. x options color red,  marker square,style none
   line  z vs. x options color blue, marker square,style none
}
```

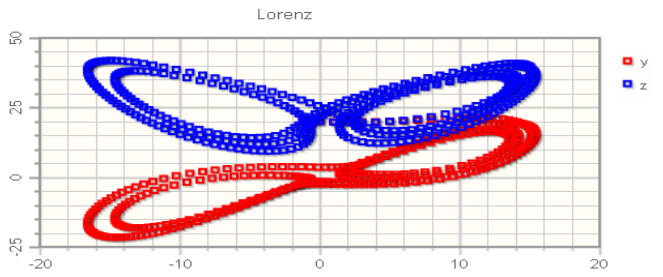

Figure 4. Sample syntax for time stepping problem.

Since the computation requirement of an implicit scheme differs considerably between models, an automatic time-stepping strategy was implemented. Effectively, the time step will be increased or decreased according to the convergence in order to reduce the run time. This adaptive step scheme was based on control theory and the formulation proposed by Soderlin et al [2]. The new time step (dt) is calculated as a weighted product of the ratios of tolerance (ε), the current (r_n) and previous residual (r_{n-1}) and the current (dt_n) and previous time step (dt_{n-1}). Given the weighting parameters $\beta 1$, $\beta 2$, and $\beta 3$, the next time step is calculated as:

$$dt = \left(\frac{\varepsilon}{r_n}\right)^{\beta 1}\left(\frac{\varepsilon}{r_{n-1}}\right)^{\beta 2}\left(\frac{dt_n}{dt_{n-1}}\right)^{\beta 3}$$

Optimization Driver

As many engineering problems require an optimized solution within a given set of constraints, the Optimization Driver was implemented to meet this need. Some of the available libraries and applications are: non-linear least squares (for curve fitting); constraint optimization (used for well design, geo-fingerprinting and others); and genetics algorithm (used for the steam scheduling project, integer optimization). OASIS has an in-house developed algorithm as well as an interface with IPOPT [4].

Recording Driver

The Recording Driver was developed to work with a database where results (and intermediates) values are stored. The user can customize the level of data to be recorded. This ranges from a full debugging mode where the user can record every single variable, expression, intermediates and every solver attempt, to only recording the significant results. Mysql provides the database functionality.

OASIS USER INTERFACES

OASIS is a web-based application. Pre and post processors are part of the Integrated Development Environment (IDE), which means that the entire problem design, solving and result analysis is done from one single OASIS IDE interface. The IDE is composed of a graphical user interface (GUI) and a text user interface (TUI), which is the users' workspace for job and case management. The GUI's main function is interactive control for input data specification and instructions and provides output of graphical representation (e.g., 2D plots and 3D visualization shown in Figure 5). The TUI provides similar control on the model structure and data. It functions as an interface for writing in OML. The target was that any session of the GUI could be scripted in the TUI. Currently, the TUI can be used to: enter a given model (with syntax highlighter and syntax check features); export results in in different formats; generate 2D plots from scripts; and, control OASIS's debugger.

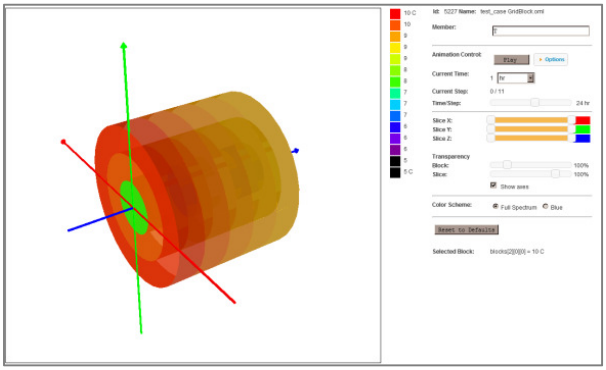

Figure 5. Example of 3D visualization

In addition to the IDE environment described below, a prototype was tested for creating a graphical interface for a given OML script. It recognizes variables, inputs and outputs within the OML script to dynamically create an input panel. Figure 6 shows a model panel for computing the cumulative steam-to-oil ratio (CSOR).

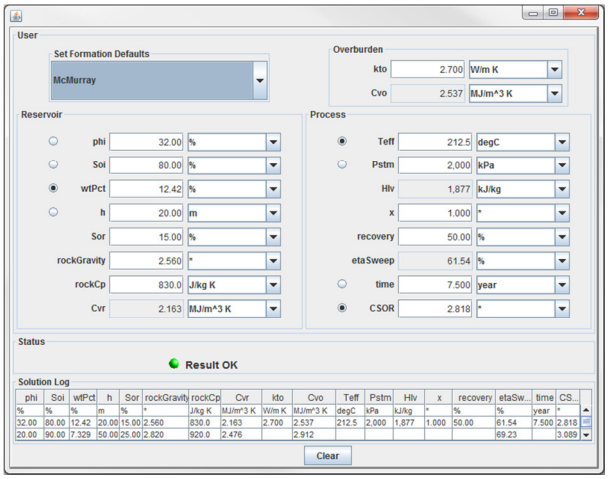

Figure 6. Sample model panel

OASIS MODELS

The Model contains the mathematical formulation of the problem to be solved or integrated. A formulation is ultimately a set of equations that are dependent on an equal number of unknowns that characterize the physics of the system. A Model exports a MemberTable that contains information on its contents. The contents are structured as multidimensional arrays of Member objects. Such objects describe the attributes of a member and enable a driver or user interface to read (and in the case of parameters, write) its value. A Model has various update methods to invoke recalculation for new parameter/variable values.

A residual, is the difference between left-hand side and right-hand side of an equation and its value depends on the values of the variables for the specific iteration. The residual measures how accurately the values of the variables satisfy the physics. The numerical representation of the equation can be done with any discretization method e.g., finite difference, finite volume, finite element, or an analytical equation that describes the relationship between the input variables. The Model solution is managed by a driver, which iteratively modifies the variables until the residuals for all the equations are within the specified tolerance. This is accomplished by a nonlinear equation solution method such as Newton's method.

In the case of dynamic (time stepping) models, the model formulation represents the equations of a single time step; i.e., a fully discretized system of Partial Differential Equations (PDE) plus constraints (boundary conditions). Management and accumulation of sequential time steps is a driver function. The driver can use multiple substeps to integrate with time.

In the case of parametric evolution models, the model formulation represents the change of the system behavior with the parameter, i.e., a fully discretized system of PDEs plus constraints (boundary conditions) according to a parameter-stepping algorithm such as arclength continuation.

A Model contains all of the physics and unique information without prescribing the numerical methods to be used, user interfaces, or other generic mechanisms such as handling restart records. This design enables the rapid development of "plug and play" models that consist of very little more than the equations.

Adhering to OML philosophy, a grid and its components, i.e., blocks, faces and vertices, are models. The grid represents a discretized region of space with specified physics. The coordinate system or tessellation is defined by a geometry, which can by chosen at runtime. By "plugging" the grid model into models with governing equations, users can build their own models with grid dimensions.

EXAMPLES

The following sections show examples of engineering applications that illustrate the advantages of OASIS.

Example 1: A basic SAGD Model

Theoretical Model

The SAGD process can be described by a system of equations that conserve mass, momentum and energy, coupled phase equilibrium, and constraints. For a reservoir of Nc components and Np phases, the mass conservation equation for each component, *c,* can be written as:

$$\frac{\partial\left(\emptyset \sum_{p=1}^{Np} \rho_p S_p x_{p,c}\right)}{\partial t} = -\nabla \cdot \left(\sum_{p=1}^{Np} \rho_p x_{p,c} v_p\right) + \left(\sum_{p=1}^{Np} \rho_p x_{p,c}\, q_p\right) \quad c = [1..Nc] \quad (1)$$

Where ρ_p, v_p and S_p are the density, velocity and saturation of a phase *p*, $x_{p,c}$ is the molar fraction of component *c* in phase *p* and q_p represents the sources (injector/producer).

At reservoir conditions, the velocity can be expressed by Darcy's law as:

$$v_p = -\left(\frac{K k_{rp} \nabla \Phi_p}{\mu_p}\right)_{p=oil,water,gas} \quad (2)$$

With the potential term as: $\nabla \Phi_p = \nabla \mathrm{p}_p - \rho_p g\, \nabla \mathrm{z}$. (3)

Here *K* represents the absolute permeability; k_{rp} the relative permeability of the phase; μ_p the viscosity of the phase.

Combining the previous equations gives:

$$\frac{\partial\left(\emptyset \sum_{p=1}^{Np} \rho_p S_p x_{p,c}\right)}{\partial t} = -\nabla \cdot \left(\sum_{p=1}^{Np} \rho_p x_{p,c} \frac{K k_{rp} \nabla \Phi_p}{\mu_p}\right) + \left(\sum_{p=1}^{Np} \rho_p x_{p,c} q_p\right) \quad c = [1..Nc] \quad (4)$$

The energy conservation equation is then:

$$\frac{\partial\left((1-\emptyset) U_r + \emptyset \sum_{p=1}^{Np} U_p \rho_p S_p\right)}{\partial t} = \nabla \cdot \sum_{p=1}^{Np} H_p \rho_p \frac{K k_{rp} \nabla \Phi_p}{\mu_p} + \nabla \cdot (\varkappa_t \nabla \mathrm{T}) + \sum_{p=1}^{Np} \rho_p H_p\, q_p + \dot{Q} \quad (5)$$

The constraint equations are given by $\sum_{p=1}^{Np} S_p = 1$ and $\sum_{c=1}^{Nc} x_{p,c} = 1$. The mole fractions and saturations of the system must satisfy $0 \le S_p \le 1$ and $0 \le x_{p,c} \le 1$.

Finally, the phase equilibrium is expressed in terms of K-values describing how a given component distributes in different phases:

$$K_{p1,p2,c} = \frac{x_{p1,c}}{x_{p2,c}} \quad (6)$$

The SAGD equations were written in OML. The OASIS model for SAGD consists of three separate files: the main SAGD file, a PVT file, and a rock properties file. The main file contains the grid definition and the abstraction of how blocks, faces, components, phases, phase fluxes and wells are represented. The PVT file contains the properties of each component in each phase. Finally, the rock properties file contains the thermal properties of the rock, porosity modifiers and the relative permeability curves. Figure 7a shows a

sample extract from the code. From Figure 7a, it can be seen that a reservoir block is characterized by temperature (T), pressure (P), components (comps) and phases (phases). The script contains the solid definition (sld) and a set of sources (mass and/or energy). The equations govern the process. The high level definitions for those equations are also shown in the figure.

```
cell TResBlock extends Block<TResCommon> {
    Temperature T, var, >5 degC,<350 degC, norm 15K, default con_tres, unit C;
    Pressure    P, var, >500 kPa, <10.0 MPa, norm 600 kPa, default con_pres, unit kPa;
    Component{} comps;
    Phase{}     phases;
    PurePhase{} purePhases;
    Fluids      flds;
    TRSolid     sld;
    TRSource{}  src;

    // Momentum and energy equations
    Mass/Volume [comps]    accum  =   sld.porosity * sum(flds.phases.mip[comps]),unit tonne/m^3;
    Energy/Volume  eAccum =  ( sld.porosity * sum(flds.phases.eip )+ (1-sld.porosity) * sld.volEnergy ),unit kJ/m
    MassRate[comps] mSources =  sum(src.ql_ma[comps]);
    MassRate[comps] mfluxes  =  sum(faces[][minus].massFlux[comps])
                              - sum(faces[][plus ].massFlux[comps]);

    Energy/Time eSources =  sum(faces[][].enerBurdenLoss ) + sum(src.eRate);
    Energy/Time eFluxes  =  sum(faces[][minus].enerFlux)
                          - sum(faces[][plus ].enerFlux);
    Energy   totEnergyHeaters = timeIntegral( OW + sum( src.eRateHtr )  );


    eqn         energyBalance timeDeriv(eAccum) = ( eFluxes + eSources ) / volume;

    eqn[comps]  massBalance   timeDeriv(accum[comps]) = ( ( mfluxes[comps] + mSources[comps] ) )/ volume ;


    MassRate[comps]    totInjMRate = sum( src.mRateInj[comps] );
    MassRate[comps]    totPrdMRate = sum( src.mRatePrd[comps] );
    Volume/Time totInjVRateW = sum( src.vRateInj[Water] );
    Volume/Time totInjVRateO = sum( src.vRateInj[oil] );
    Volume/Time totInjVRateG = sum( src.vRateInj[Gas] );
```

a) Sample SAGD block definition

```
Temperature T, var, >5 degC,<350 degC, norm 15K, default con_tres, unit C;
```

b) Variable definition

Figure 7. OASIS representation of a SAGD reservoir model block.

In Figure 7b, note that each variable has a quantity, the finite solution domain (minimum and maximum), a default initial value, a preferred unit for reporting or plotting, and a "norm." In OASIS, a norm is an acceptable maximum change in that variable for a given time-step. In the example shown, T is the name of the variable and its quantity is Temperature. In every computation involving T, OASIS will check that the units are those used for temperature. The value for an acceptable solution is between 5˚C and 350˚C. From one time step to the next, T can change by no more than 15˚K (norm). If the norm is violated, the time step is decreased and the solution is re-computed. The default initial value of T is given by the value of a constant called con_tres, and the preferred unit is Celsius.

Several benchmark tests were conducted to compare OASIS SAGD model to the results from CMG's STARS [4]. Figures 8 and 9 show one such test. In general, OASIS results were in agreement with those obtained by the commercial simulator.

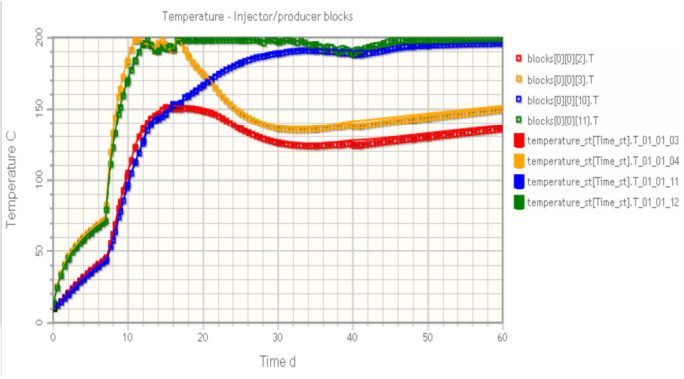

Figure 8. Comparison of results with commercial simulator – "Temperature at Injector and Producer Block". (OASIS = markers, STARS = solid line)

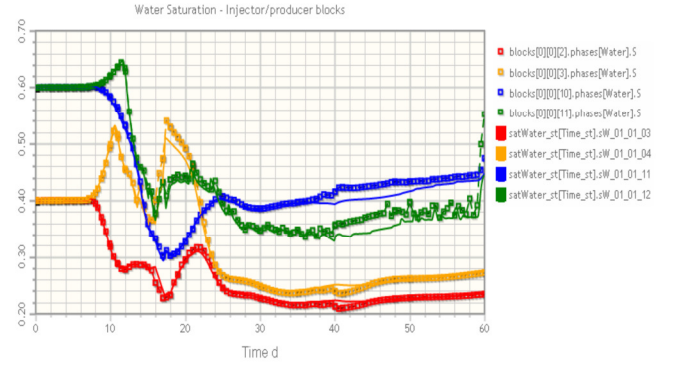

Figure 9. Comparison of results with commercial simulator- "Water Saturation in Injector and Producer Blocks". (OASIS = markers, STARS = solid line)

Example 2: Extending a model for what-if physics: wettability changes

OASIS is not intended to replace other simulators, but to overcome some of their limitations. The main goal was to make modeling different physics available to the user. The equations are visible and accessible so the user can modify any given formulation. At any stage of modeling, the user has control of what is being solved. For the purpose of illustration, this example shows the modification of the SAGD code to take into account changes in wettability. Wettability of any reservoir has a significant effect on ultimate oil produced by any water-driven mechanisms. Figure 10 shows an alternate formulation to computing the relative permeability of the fluids according to which phase is the wetting phase. The implementation of the new formulation is fairly simple in OASIS since the software was designed to test new physics

with minimum programming effort. In this case, the equations reflect that the reservoir is oil wet as long as the temperature is below a given threshold.

```
TConductivity kt = (con_kr*(1-porosity)) + (porosity*((flds.phases[Gas].S*con_kg)+(flds.phases[Oil].S*con_k

Real          alpha = kt/(ro*Cp), "thermal diffusivity";

// 1) Rock is water-wet=> oil is the middle phase.
//     Kro = f( Sw, Sg )
//     Get kro using STONE (or linear interpolation) (Pcow and Pcog)
// 2) Rock is oil-wet=> water is the middle phase.
//     Krw = f( So, Sg )
//     Get Krw using STONE  (Pcwo and Pcwg)

// Parameters for water wet
*             GS1 = max(phases[Gas].S  , 0.0000);
*             krg1  = KrG1[GS1].Krg ;
*             krog1 = KrG1[GS1].Krog;

*             WS1   = min( max(phases[Water].S, 0.0000 ), 0.9) ;
*             krw1  = KrL1[WS1].Krw ;
*             krow1 = KrL1[WS1].Krow ;
*             krocw1= 1.0;
*             kro1  = max( krocw1 * (((krow1/krocw1) + krw1) * ((krog1/krocw1) + krg1) - krw1 - krg1), 0)  ;

// Parameters for oil wet
*             GS2   = max(phases[Gas].S  , 0.0000);
*             krg2  = KrG2[GS2].Krg ;
*             krwg2 = KrG2[GS2].Krwg;

*             OS2   = min( max(phases[Oil].S, 0.0000 ),1 ) ;
*             kro2  = KrL2[OS2].Kro ;
*             krwo2 = KrL2[OS2].Krwo ;
*             krwco2= 1.0;
*             krw2  = max( krwco2 * (((krwo2/krwco2) + kro2) * ((krwg2/krwco2) + krg2) - kro2 - krg2), 0)  ;


boolean isOilWet = T < T_waterWet;

* kro = isOilWet ? kro2 : kro1;
* krw = isOilWet ? krw2 : krw2;
* krg = isOilWet ? krg2 : krg1;
```

Figure 10. Extract of OML script for wettability changes

Figure 10 shows the code where the changes were made to the condition which determines the wetting phase. There are two cases to consider: 1) the rock is water wet, therefore the middle phase is the oil; 2) the rock is oil wet, therefore the middle phase is the water. The previous case shows that the selection of one formulation over the other is governed by a simple boolean parameter called isOilWet. This parameter is evaluated based on whether a block's temperature is above a given threshold (T_waterWet). In this formulation, the rock wettability can change at any time in the simulation.

What if the wettability change to water wet is considered irreversible? Then the code can be simply modified to reflect this behavior. The Boolean statement can be changed to take the irreversible effect into account. The new code now becomes:

boolean isOilWet = T < T_waterWet && isOilWet;

Figures 11 and 12 show the relative permeability curves for the different wetting scenarios and a comparison of the cumulative water injection in the first months of the project. Both cases show quite different results. Figure 12 shows for the two selected cases the difference in water injection. A complete analysis could be carried out for different wettability possibilities (only water wet, only oil wet, oil wet changing to water wet at different temperatures, etc.). The aim of this example is to illustrate how simple is to write the new formulation to describe different physics in OASIS.

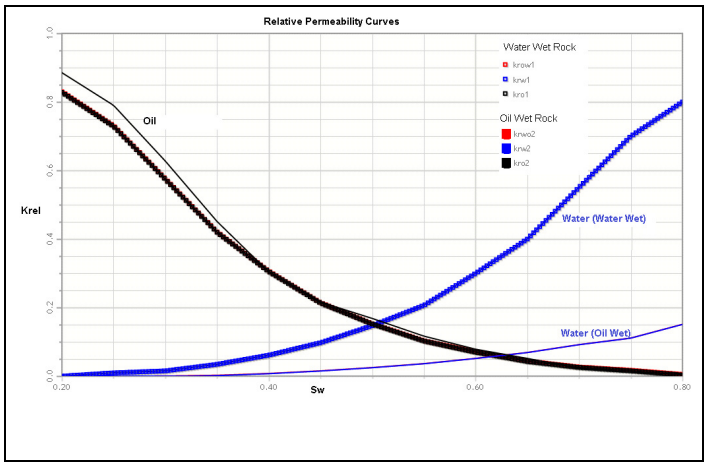

Figure 11. Relative permeability curves for different wetting fluids.

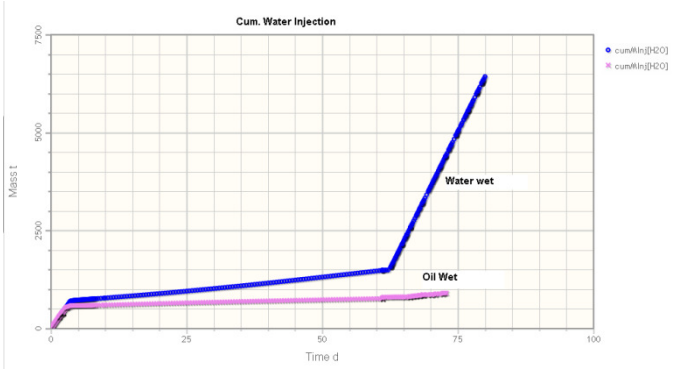

Figure 12. Water injection for water wet and oil wet cases.

Example 3: Boundary conditions and multi-system formulation

Another important feature that OASIS provides is the ability to formulate different boundary conditions and to describe models as composed by many non-linear systems. As an example of this feature, Figure 13 shows how boundary conditions are prescribed for a given grid. For this example, a SAGD model is used. A semi-analytical overburden/underburden heatloss model proposed by Vinsome [5] was attached to the lower surfaces of the domain. This is done in OML by linking the boundary to a cell that described the heatloss model.

Two implementations were completed for this model: an implicit and an explicit formulation. The implicit approach had to use the multi-system feature since if all the equations were expressed as one system such system did not converge. For this approach all the grid block equations were part of

system 0, and the faces of the blocks where the heat loss model is attached formed system 1. In this case both solutions were closed and match that calculated by STARS as shown in Figure 14.

```
// 2- Set boundary conditions
set boundaryFaceTypes[z][minus] = heatLossFaceImp

// WHAT IF?
//set boundaryFaceTypes[z][minus ] = heatLossFaceExp
```

```
cell heatLossFace extends TResCommon{

    Time time, unit day;
    Time dtime, unit day;
    Density[Signs]                  ro, default 2560 kg/m^3,
                                        "Over/Underburden density"              ;
    Energy/Mass*Temperature[Signs]  Cp= con_cp,
                                        "Over/Underburden heat capacity"        ;
    Energy/Volume*Temperature[Signs]Cpv= ro[Signs] * Cp[ Signs];
    TConductivity[Signs]            kt, default 1.496e5  J*m^-1*day^-1*K^-1,
                                         "Over/Underburden thermal conductivity";

    Temperature Tref,  default 284.15K;
    Temperature dThmin, default 0.1 K;

    Area           flowA = nbrs[ boundary ].size[x]*nbrs[ boundary ].size[y];
    Length^2/Time  alpha  = kt[ boundary ] / Cpv[ boundary ] ;

    Temperature  Th   = nbrs[ boundary ].T - Tref, unit K;
    * HLStarted, default 0;
    Time tini, default 0s;

}



cell heatLossFaceImp extends heatLossFace{
// Semi-analytical heat loss model
// Kaz Vinsome & J. Westerverld, "A Simple method for predictiong cap and base rock
// heat losses in thermal reservoir simulators", JCPT,  Resrvoir Perfornace, July-Sep 1980

    Length  dh  = (alpha * time  )^0.5 / 2  ;
    boolean calcHL = dh > 0m ;
    Temperature/Length   ph, var, default 100 K/m, norm 1.35E+09  K/m,
                                  active( calcHL ), inactiveDefault 0 K/m,    system 1 ;
    Temperature/Length^2 qh, var, default 100 K/m^2, norm 9.99E+10 K/m^2,
                                  active( calcHL ), inactiveDefault  0 K/m^2, system 1 ;

    Temperature*Length  I  = Th * dh + ph * dh^2 + 2 * qh * dh^3;

    eqn hl1  timeDeriv(Th) = alpha * (Th / dh^2 - 2 * ph / dh + 2 * qh),
                             active( calcHL ), system 1;

    eqn hl2  timeDeriv(I) = alpha * (Th / dh - ph),
                            active( calcHL ), system 1;

    Energy/Time enerBurdenLoss = calcHL
                               ? -kt[ boundary ] * (Th / dh - ph) *  flowA
                               : 0 W , unit W;
```

Figure 13 – Setting boundary conditions and using multi-system definition in an implicit formulation of the heat loss boundary conditions.

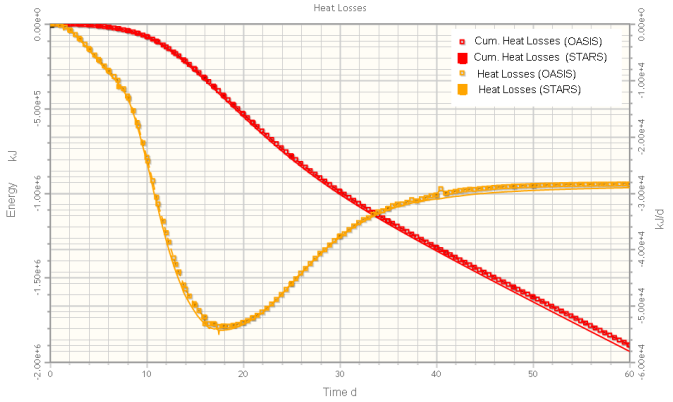

Figure 14. Comparison of results with commercial simulator - Cumulative and instantaneous heat losses.

CONCLUSIONS

In Canada, the oil and gas industry is one of the biggest investors in research. Advances in technology have made it possible to extract fossil fuels that were considered too difficult or expensive to produce. Only through technical advances can production be increased while reducing the environmental footprint of the projects. Tools are a necessity to enable new technology development. OASIS was Laricina's response to that need. OASIS is software for engineering and scientific multiphysics modeling. Since fostering innovations was its focus, it was developed as an extensible system that uses interchangeable plug and play physics. In OASIS, the user owns the equations and any model formulation can be developed and modified with minimal effort. OASIS provides a graphical user interface that is tightly integrated with the modeling environment. The OASIS project was one of several of Laricina's innovation initiatives. Growth and success are fundamentally dependent on innovation, new technology and processes, and scientific advances. Research and development initiatives are essential for addressing current and future challenges.

ACKNOWLEDGMENT

The authors would like to acknowledge Neil Edmunds for his vision and inspiration and thank Laricina Energy Ltd. for their support to the OASIS project.

NOMENCLATURE

$\beta 1, \beta 2, \beta 3$: weighting parameters

dt_n: time step increment

ε: tolerance

g: acceleration of gravity

H_p: enthalpy of phase *p*

K : absolute permeability

k_{rp} : relative permeability of the phase *p*

$K_{p1,p2,c}$: equilibrium constant

$\varkappa_t$: thermal conductivity

∇p_p: pressure gradient

q_p: source of phase *p* (injector/producer)

μ_p: phase viscosity

$\nabla \Phi_p$: potential of phase *p*

r_n : residual at step n

ρ_p: phase density

S_p: phase saturation

∇T: temperature gradient

U_r: internal energy of the rock

Up: internal energy of phase *p*

v_p: phase velocity

$x_{p,c}$: molar fraction of component *c* in phase *p*

∇z: vertical gradient

REFERENCES AND PARTIAL BIBLIOGRAPHY

1. Hazewinkel, Michiel: *Lorenz attractor*, Encyclopedia of Mathematics, Springer, ISBN 978-1-55608-010-4, 2001
2. G. Soderlind and L. Wang: *Adaptive Time-Stepping and Computational Stability*, Journal of Computational Methods in Sciences and Engineering, Cambridge Int. Sc. Pub., Vol. 2, No. 3, 2, 2003
3. F. E. Curtis, J. Huber, O. Schenk and A. Wächter: *A Note on the Implementation of an Interior-point Algorithm for Nonlinear Optimization with Inexact Step Computations*, Mathematical Programming, 136(1):209-227, 2012
4. Computer Modeling Group (CMG): *CMG STARS User's Guide*, Computer Modeling Group LTD., Calgary, Alberta, Canada, 2013.
5. K. Vinsome & J. Westerverld: *A Simple method for predicting cap and base rock heat losses in thermal reservoir simulators*, JCPT, Reservoir Performance, July-Sep 1980