Twenty Sixth Annual CNS-CNA Student Conference
Toronto, Ontario, Canada. June 10-13, 2001

FOUNDATION FOR A VISUAL REACTOR SIMULATION TOOLKIT

David Gilbert, Department of Electrical and Computer Engineering
McMaster University, Hamilton Canada
gilbert@mail.cas.mcmaster.ca

ABSTRACT

The current research goal is to implement a well structured object based simulation system, for use by the scientists, operators and engineers at the McMaster Nuclear Reactor. Neutron flux, poison buildup, and coolant flow are modeled using finite difference equations. A Cartesian grid with variable resolution, is used to represent the model's geometry. Each grid may directly represent an array of partial differential equations or may be composed of sub-grids, each being treated internally as a separate computational object. The final project will handle model configuration, integration of related simulations, code generation, execution, and output rendering. The simulation tool kit is not yet complete, this paper represents a description of the preliminary status of the project.

I. INTRODUCTION

The purpose of the current project is to develop a programming tool for physics models capable of simulating neutron flux, poison buildup, and water fluid flows within the McMaster Nuclear Reactor (MNR) a small light water moderated research reactor located on the McMaster campus. During early conversations about the form of the design several goals were identified.

1. Model's physics must be fully configurable.
   (a) Physical laws should be expressed in a mathematically natural way.
2. Model's geometry must be fully configurable.
   (a) A Cartesian mesh with variable granularity is used.
3. Libraries should be easy to edit and extend, and should help the beginner not limit the expert.
4. Physical laws and geometrical structures should have a natural or 1:1 association.
5. The model should be capable of running in real time, so that it will provide a virtual window into the reactor at all times.

Along with these very general goals additional requirements include input and output mechanisms which can work over the Internet, so that scientists may rely on a central server to share their information. Model states must be checkpointed from time to time, to build up a history database, and the model must be able to communicate with standard codes approved by the AECL, already in use at the MNR, including CATHENA, 3DDT, WIMS and ASSERT [1].

Many general purpose fluid flow packages are already available (Fluent [2] for example). Commercial and industrial packages have several disadvantages which prevent the MNR from using them. Many computational fluid dynamic packages are designed to solve a specific set of problems, and are not generic enough for modeling a multi-group neutron diffusion problem. Since commercial packages are sold for profit, source codes to the packages are not always available. Without source codes fine tuning the problem and code verification is impossible.

At the opposite extreme general numerical libraries are available for the efficient solution of sparse matrices. Netlib[3] is an excellent repository of libraries and numerical solving tools, although these libraries require a great deal of programming in order for the user to take advantage of them. A problem's specification may be more difficult to organize than the code which is used to generate the solution.

Several general packages for the specification of reactor models already exist. The Modular Modeling System (MMS) [4] is an example of a nodal system which allows the user to draw on a library of prebuilt objects (pumps, valves and pipes for example) constructing a reactor cooling model in a schematic fashion. A modeling system described by Nilsson [5] follows the same object oriented approach. Both models use Microsoft Windows front end editors to organize the reactor components, and both models are written in specialized modeling languages, MMS is written in ACSL, and Nilsson's model components are written in an object-oriented modeling language called OMOLA. The Modular Modeling System is described in [4] as a more accessible supplement to codes like TRAC, RELAP and RETRAN [6], rather than a replacement. Many reactor monitoring and analysis jobs do not require the complexity of codes like RETRAN, and are either being done with the overly detailed codes, or not being done at all because of the great expense involved.

The MARS [7] project aims to combine two standard codes (RELAP and COBRA-TF) into a single multi-dimensional fluid flow analysis tool. Dynamic memory allocation features of FORTRAN90 were used, and a graphical user interface was added to the final code combination to make it easier for engineers to work with the codes.

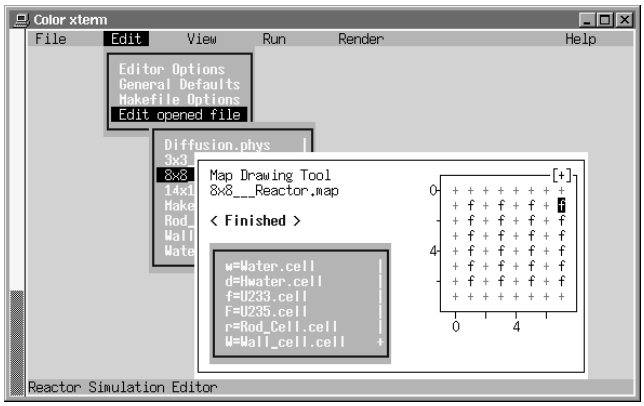

Figure 1: Cell editor

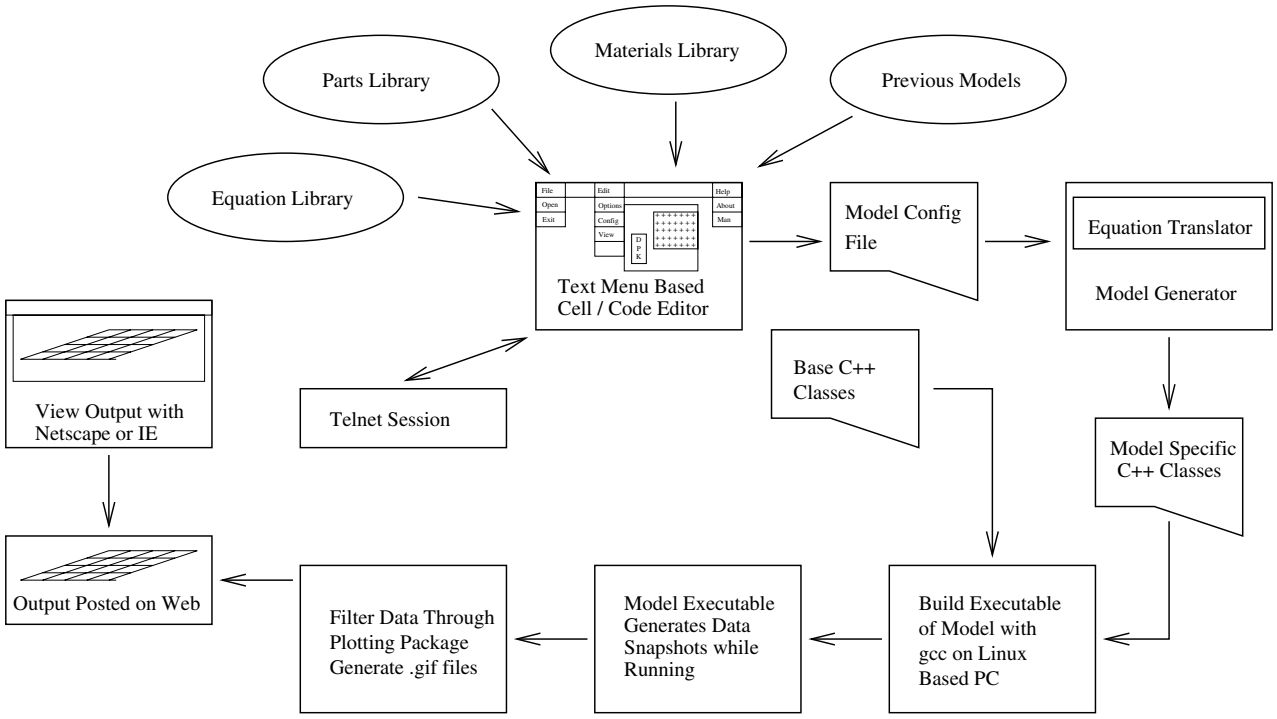

Figure 2: Model data flow overview

At this time the codes used for licensing the MNR (CATHENA, WIMS) do not have an easy to use interface, and there is also no general purpose code in use at the MNR to perform routine or experimental analysis. The proposed simulator aims to fill this need. The interior of the prototype model is exposed to the user, in the form of partial differential equations to allow maximum flexibility. A front end editor organizes the equations for the user and associates them with a geometry simplifying the job of model configuration, the cell editor is shown in Figure 1.

Internally linked cell structures are represented as objects with uniform interfaces. Objects are always a collection of linked cells. Objects are merged one at a time by the model designer into hierarchical maps. Each group of objects communicates with its neighboring group by being placed close enough so that a cellular interface is made.

II. DESIGN OVERVIEW

A simple text menu based cell and code editing tool is used to configure the simulation. The menu is designed to launch geometry and code editors, similar to the fashion of the classical Borland C editors which produced code for DOS computers. The text menu based implementation was chosen over a graphical implementation for two reasons: it runs easily over a telnet session, allowing scientists to work remotely, and it is easier to implement than a full graphical user input (GUI) system. If time permits an X based GUI may be designed to replace the text menus, which would also allow remote access, with the addition of a more sophis-

ticated geometry input mechanism.

For the front end editor to produce an efficient executable model, it first writes a configuration file which is passed through the model generator (see Figure 2). The model generator reads information about the constants, geometry and physics of the model as specified by the configuration file and checks that the model's design is internally consistent. The information in the configuration file is translated into a collection of C++ classes specifically tailored for the current simulation. The model specific C++ classes inherit general features from the base modeling class and the final code is passed to gcc for compilation.

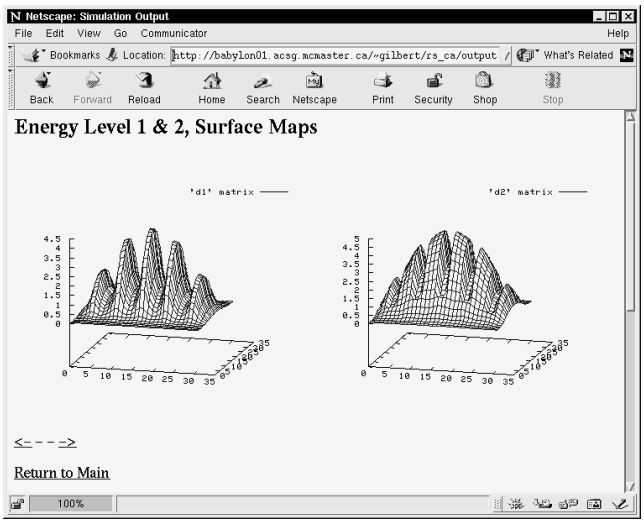

Figure 3: Netscape used to view output

While the simulator is running, it generates periodic data snapshots of some area of interest. Snapshots are automatically rendered by gnuplot, a batch driven scientific data plotting program, and the rendered output files are copied to the users web directory so they can be remotely viewed (see Figure 3). The data snapshots will be stored in an online database, so that a user may interactively view the simulator's history.

III. CARTESIAN EMBEDDED MAPS

All maps are rectangular grids belonging to one of two types. A map is either a map of simple cells, where each cell computes physical properties based on its constants and the constants and properties of its neighbors as defined by its partial differential equation, or a map has other maps as its cells, effectively sub-maps as shown in Figure 4.

Cell maps solve for variable properties (flux, temperature etc.) as defined by the modeler. Maps may be grouped with adjacent maps which use a different formula basis, and which solve for a different set of variables. Maps may have any dimension, and may be nested at any level. All maps use identical interface functions, like Read_Element(), Output_State(), Import_State(), and Solve_Map_SOR().

By default when maps share their values with bordering maps, if a variable is not defined by both adjacent maps, then the numerical value is mirrored at the edge. This way a quantity of interest (poison buildup for example) may be defined for only a small region of the geometry if it does not apply to the entire model. Memory is conserved, and the modeler need not be concerned with the details of the implementation.

Inheritance in C++ is used to merge the map specific data structures and formulas with generic functions so that each map may have a uniform interface. If a map has sub-maps, then the top level Solve_Map_SOR() function calls the Solve_Map_SOR() function of its children, each

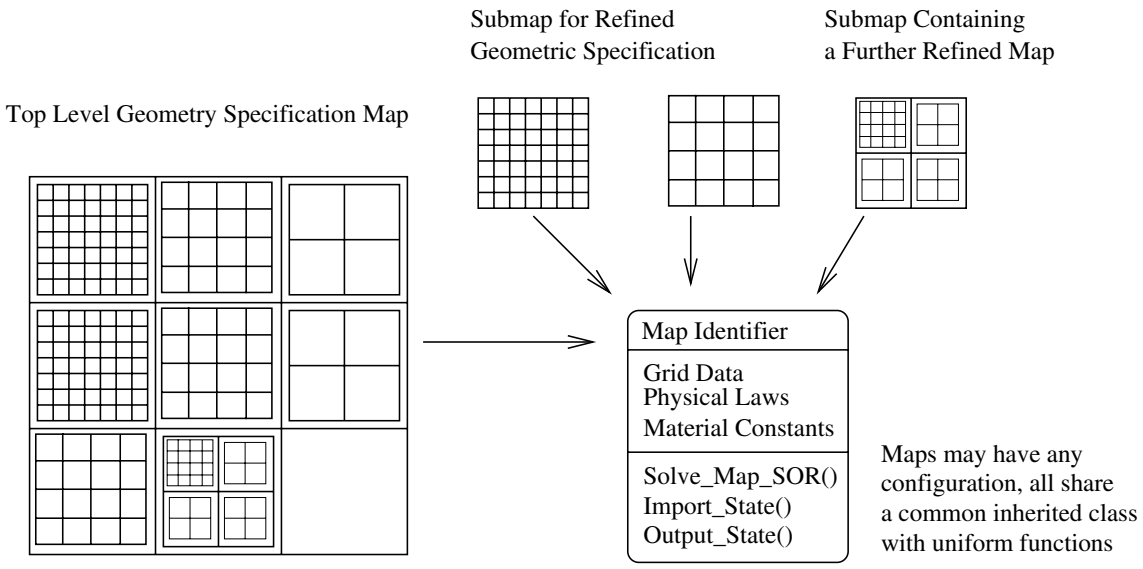

Figure 4: Map layout

in turn, until the entire grid has reached a new equilibrium.

IV. AN EQUATION TRANSLATOR

One of the fundamental problems in allowing a model's physics to be fully configurable is that it means the model must be able to handle any imaginable equation, that fits in with its general structures. One option is to allow the model designer to compile their own object codes, written in the native language of the model. The trouble with this approach is the modeler is forced to think more about programming than physics. Another approach would be to have the modeler select pre-compiled physics equations from an exhaustive menu. This constrains the configuration to whatever physics the system designer is aware of, as no menu is ever fully exhaustive.

This project follows a compromise between these two alternatives. An equation translator along with an easy to edit library of equations is supplied. The equation translator takes formulas expressed in a mathematically natural way and converts these equations into a form that a C compiler can understand.

The equation translator runs in three modes (configurable for each cell):

1. A strict mode which only accepts linear equations following a tightly defined syntax.
2. A mixed C mode where recognized simulator symbols can be intermingled with raw C code.
3. A literal mode, where C code is passed directly to the C compiler, and no translation is done.

The basic element of any simulation is a cell. Each cell has a set of equations associated with it that define the physics of that cell as well as a

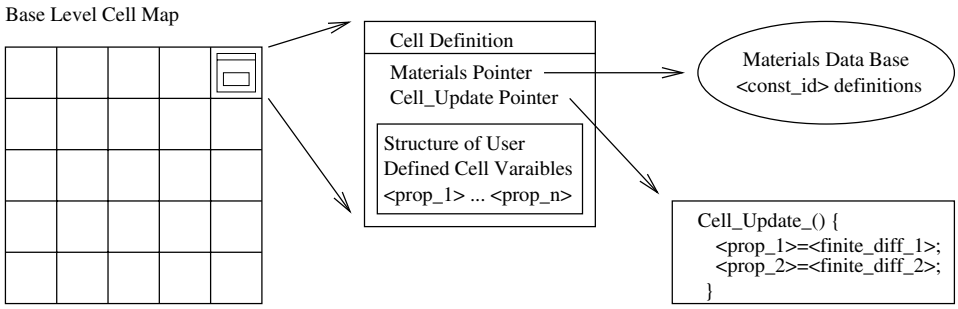

Figure 5: Cell map

set of constants that define the material properties of the cell (see Figure 5). Since not all maps are required to support the same set of variable properties, all cells need not support the same set of physical laws. A cell is described in the model configuration file as:

```
rs_cell <const_id> <prop_id> {
      <prop_1>=<finite_diff_1>;
      <prop_2>=<finite_diff_2>;
      ...
      <prop_n>=<finite_diff_n>;
}
```

An equation may have three sorts of identifiers, either constants, variable properties, or external identifiers. The equation translator must have a way to recognize these, and so constant and property structures are mentioned in the cell definition. Constant types and property types must also be defined. A property type declaration is a list of property names, followed by an identifier, and is handled exactly like a struct declaration in C, except that the model generator makes note of the form that the structure takes.

```
rs_prop struct {
      <prop_1>;
      <prop_2>;
      ...
      <prop_n>;
} <prop_id>;
```

Constant types are described in an analogous fashion. Constants have fixed values, and are defined by a pointer to the materials library.

The general steady state finite-differenced multi-group diffusion equation [8] is written as:

$$\left[\Sigma^{g}_{R_i} + \sum^{J}_{j} \frac{D^{g}_{ij}}{\Delta^{2}_{ij}}\right] \phi_{ig} - \sum^{J}_{j} \frac{D^{g}_{ij}}{\Delta^{2}_{ij}} \phi_{jg} - \qquad (1)$$

$$\sum^{g-1}_{g'=1} \Sigma^{g' \to g}_{S_i} \phi_{ig'} = \frac{\chi^{g}}{k} \sum^{G}_{g'=1} \upsilon_{g'} \Sigma^{g'}_{f_i} \phi_{ig'}$$

Equations recorded for each cell are used as part of an iterative solution to solve for the flux of the map, all ϕ_{ig} where i represents the index value of the local position, and g represents the energy group level. This equation is reorganized to solve for ϕ_{ig} and can in principle be written in standard C notation and then be merged at compile time with the iterative solver. This strategy may be used with any finite differenced equation, although the flux equation is used in the following examples.

In order to support hierarchical maps the model must declare and array of pointers for each map object. In standard C notation each cell has a field pointing to its update function, a pointer to the constants library (materials cross sections for example), a pointer to a structure of variables to

solve for (flux or temperature). Variables in C notation are identified as:

```
grid[x * this->X_MAX + y]
       ->property->phi[g]
```

Constants are identified as:

```
grid[x * this->X_MAX + y + 1]
       ->constant->Sigma_R[g]
```

The first job of the equation translator is to find properties or constant definitions in a users equation and automatically insert the pointers. The position in the grid array is computed based on the current objects dimensions. Finite difference methods work with neighboring cells, so the sub index [E], [W], [N] and [S] are automatically translated into the correct relative position formula, if the reference is to the current cell, the position pointer is dropped. Using these conventions variables can be referred to in the configuration file more simply as:

```
phi[g]     and     Sigma[S][g]
```

The first and last terms from equation (1) are:

$$\left[\Sigma_{R_i}^{g} + \sum_{j}^{J} \frac{D_{ij}^{g}}{\Delta_{ij}^{2}}\right] \phi_{ig} \qquad (2)$$

$$\frac{\chi^{g}}{k} \sum_{g'=1}^{G} \upsilon_{g'} \Sigma_{f_i}^{g'} \phi_{ig'} \qquad (3)$$

Since summation is a common operation the equation translator uses a short form to express it. In the model configuration file term (3) can be written as:

```
(Chi[g]/k) * Sum(g'=1..G,                    (4)
 nu[g'] * Sigma_f[g'] * phi[g'])
```

For the summation operator to work correctly the bounds must be known at compilation time. Summation with more subtly expressed bounds must be coded in C either using the mixed or literal cell equation mode as a for() or while() loop. In strict mode the summation operator is handled by unrolling the summation and listing each element explicitly. This allows for very efficient execution, and no chance of a non-terminating loop.

Subscripted variables are commonly marked with a prime symbol (’). The equation translator allows the prime symbol to be used as part of a local variable to help distinguish it from other variables.

Term (2) can be written as:

```
(Sigma_R[g] +                                (5)
  (D[E]+D[W]+D[N]+D[S])
              /delta^2)*phi[g]
```

The [N], [S], [E], and [W] tags are converted by the equation translator into their correct relative position, and the neighboring values of D are returned in each case. Caret (^) is allowed to represent exponentiation as it does in Pascal, even though this is not normally present in C. Since term (5) has assumed that delta is uniform, and since this may not always be the case, it is convenient to have a special form of summation which more closely matches the original equation. Term (2) could also be written as:

```
(Sigma_R[g] + Sum(j=[N]..[W],                (6)
       D[j]/delta[j]^2)*phi[g]
```

In term (6) the mathematics expressed is the same as the first term in the original equation. Instead of the limits of the sum being integers, [N],

and [W], are used to suggest a clockwise tour of the four directions.

V. PROJECT STATUS

The described simulation toolkit is still in the early stages of its development. Currently the model generator can handle two dimensional geometrical grids, and plans are in place to extend the solver to three dimensions. The equation translator allows for expressing the relative position of variable properties, or constants, in neighboring cells, with the use of the [N], [S], [E] and [W] tags, and automatically inserts the correct pointer information. The summation, and exponentiation operators have not yet be implemented.

A method for integrating the existing codes into the proposed simulator has not yet been thoroughly investigated. It is anticipated that C++ class wrappers can be used to encapsulate the input and output of external executable programs, and merge their output with the new structures.

Output generation of data files over the web has been tested, and a basic version of the simulation generator, and equation translator have also been tested. Most of the base layers of the menu input system are ready.

It is hoped that when the model generator is ready it will be used to assist with routine analysis jobs by providing a quick and ready estimate of the reactors status. An early well integrated prototype capable of running a sophisticated simulation should be ready in fall 2001.

References

[1] Garland W., "*Thermalhydraulic Modeling of the McMaster Nuclear Reactor*", MNR Technical Report 97-04, McMaster University, 1997.

[2] Fluent, "*Fluent Inc, CFD Flow Modeling Software and Services*", http://www.fluent.com, 2001.

[3] Netlib, "*Netlib Repository at UTK and ORNL",* http://www.netlib.org, 2001.

[4] Smith L.P., May R.S., Divakaruni S.M., Deluba G.S., "*An Overview of the Modular Modeling System (MSS) Code and Applications*", Babcock & Wilcox, Technical Paper TP1081, 1983.

[5] Nilsson B., Eborn J., "*An Object-Oriented Model Database for Thermal Power Plants",* European Simulation Conference, 1995.

[6] Agee L. J., "*Retran Overview*", Nuclear Technology Vol. 70, July 1985.

[7] Jeong J.J., Ha K.S., Chung B.D., Lee W.J., "*Development of a Multi-Dimensional Thermal-Hydraulic System Code, MARS 1.3.1*", Annals of Nuclear Energy Vol. 26, 1999.

[8] Duderstadt J., Hamilton L., Nuclear Reactor Analysis, John Wiley and Sons, 1976.

[9] Yeung, M. R., Jiang G. B., "*Development of an Efficient Three-Dimensional Reactor Core Model For Simulator Applications*", Nuclear Technology Vol. 97, March 1992.