Abstract. Computer based simulation of continuous and discrete systems, requires formal mathematical models of computation that calculate state variables at discrete points in time. Almost invariably the algorithmic implementation of the mathematics is iterative in nature, such that by iterating the algorithm values of state variables progress from one point in time to another.

With such iterative models the authors believe that in order to fully understand the model it is important to remain aware of the order in which the various parts of the overall algorithm are executed within the context of a single iteration, especially when the model employs a hybrid of continuous and discrete models of computation. Unfortunately many formal graphical modeling languages have a tendency to hide this aspect of models. Furthermore languages using hierarchical component model constructs to deal with complexity don’t extend the ability to compose a model from sub-models to include the ability to control the model of computation at each point in the hierarchy. Some aspects of the model thus remain global, breaking the encapsulation afforded by the hierarchical decomposition.

This paper presents a graphical modeling language for simulation of continuous and discrete systems that provides visibility of the ordering of execution of the elemental parts of iterative mathematical models, and allows greater flexibility in how models of computation are arranged in a component model hierarchy.

1. INTRODUCTION

Models used in computer based Simulation employ algorithms that are iterative in nature such that they calculate the states of the model at discrete points in time. This is true for the modeling of both discrete and continuous systems however models for the latter may sometimes employ non-causal numeric integration algorithms that require a nested iterative approach to from one solution point in time to another. The part of the overall algorithm that is responsible for transitioning iterations the model is termed here the Model of Computation (MoC). MoC refers to any formalism that describes how certain types of computations are performed (1). If one were implementing a relatively simple simulation in say C or Java the MoC would be realized in loop control code associated directly with advancing the simulation. The code body within such loops realizes the parts of the algorithm that are focused on representing mathematically the subject of the model. Typically the modeler will break the loop body into functional components that must be executed in a particular order to advance the model’s state properly.

When there are many different systems or phenomena being modeled, text-based coding languages methods become unwieldy and so modelers often turn to graphical languages supporting hierarchical compositions of coupled, modular component models, such as Simulink® to deal with complexity.

A difficulty with modeling and simulation tools, based on graphical languages, seems to be that the user loses sight of how the model in each time-stepped iteration. In particular, the ordering of execution of the individual functional components of the complex model, are not represented clearly to the viewer.

A difficulty in supporting hierarchical composition of coupled models seems to be the tendency for tools to use a global approach for the MoC that manages the numerical integration of continuous-time system states. Large and complex models may have components with vastly different numeric integration accuracy requirements. A global approach forces all differential equations to be solved according to those with the highest accuracy needs, thus creating efficiency issues.

To address these issues this paper presents a graphical modeling language supporting hierarchical compositions of coupled models that provides, visibility of: functional component ordering within the notion of an iterative algorithm; and arbitrary placement within the hierarchy of MoC for managing numeric integration to some desired accuracy.

2. LOSING SIGHT OF FUNCTIONAL ORDERING

When using languages to develop models for simulation it is typically the structure of the iterative Model of Computation that drives the structure of the software implementation (Figure 1:).

In this form, the implemented model mathematics can be thought of as having two parts. The first part is the outer loop control mathematics used to implement the iterative Model of Computation. The second part is the inner contextual mathematics representing the systems of interest. The functional components of the contextual mathematics are the individual equations that transform one set of state variables into another.
The totality of the state of the contextual mathematics \( (S) \) directly reflects that which is being modeled. State is transitioned within an iteration of the model via the execution of the functional components in a particular order. A text-based language makes this ordering explicit in code while also organizing the contextual mathematics into iteration control loop structures that reflect common Models of Computation (MoC).

In other words text-based languages tend to present a mathematics orientated view of both the Model of Computation and the contextual mathematics that make up the model.

\[ S = S_0 \]

\[ Y_1 = f_1(S) \]
\[ S_1 = f_2(Y_1) \]
\[ Y_2 = f_3(S, Y_1) \]
\[ S_2 = f_4(Y_2) \]

**Figure 1:** Typical structure of a model’s iterative algorithm using a pseudo text-based language.

As the systems and their models become more complex so too does the structure of the model code - especially when different people are developing different parts of the overall model – and it becomes more difficult to understand what is being represented.

**Figure 2:** Typical structure of a systems orientated model using a graphical language.

At this point, in order to deal with the complexity, people often turn to graphical languages that support hierarchical compositions of models. Such languages, at least at the higher levels in the model hierarchy, shift the model away from mathematical constructs towards constructs that reflect the actual systems being modeled (Figure 2:).

Although such languages greatly improve the user’s understanding of what is being modeled and how functional components relate to each other it is done so at the expense of how they are being modeled mathematically. Even when the systems are decomposed to their lowest levels to a point where the model elements represent particular mathematical functions there is still no complete representation indicating the execution order of the model elements, especially when there are feedback loops, nor the MoC under which each functional elemental is being iterated (Figure 3:).

In other words graphical languages tend to present a systems orientated view, and when elemental mathematical functions are shown it is only the contextual mathematics of the model that is seen.

**Figure 3:** Example of a mathematical functional model using a typical graphical language.

In not providing any formal representation for the modeler to specify execution ordering or iteration control mathematics in a graphical modeling language simulation environments utilizing these languages must have an implicit inbuilt understanding of how each and every functional component is to be executed (it is not uncommon that an element will have more than one execution point, i.e. the methods a simulation environment would call to execute it).

A negative effect of this is that it can sometimes be difficult to integrate into such a simulation environment component models that have been implemented in different environments because they would need to be encapsulated into some pre-defined functional component construct the target environment can execute (2).

If a language provides a means of fully specifying execution ordering of functional components then it becomes possible to have model definitions that are independent of the simulation environments that execute them. This supports the concept of portable modeling which aims to minimize a model’s dependency on any particular simulation environment (3).

Simulation environments that support such a definition do not rely on models presenting a fixed functional...

---

1 The couplings represent contextual information such as system state or events that are exchanged between the various component models, which would normally be labeled.

2 Simple couplings as shown in the diagrams between functional elements provide necessary but insufficient information towards defining execution ordering of functional elements.
component interface. Rather the modeler may create models of arbitrary internal complexity and with arbitrary functional components to be executed by the simulation environment.

3. SPECIFYING ORDER

The following is constrained to contextual mathematics; the mathematics that defines the behaviour of that which is being modeled and is iterated by the Model of Computation (MoC)(Figure 1:).

3.1 Modeling Elements and Relationships

We begin by recognizing there are two basic mathematical modeling elements that must be represented, Variables and Functions. Functions read and/or update Variables. A Simulation Environment (SE) executes Functions as a single point of execution, that is from the point of view of the SE a Function represents an atomic element of mathematics whose execution cannot be interrupted. In relation to the notion of an iteration of the model we define the following rule.

When iterating a model’s contextual mathematics the Simulation Environment will execute a Function at most once.

It is possible that a Function need not be executed at all according to rules of the (MoC) the iterations of the model (see section 4). It is important to note that both Functions and Variables can be arbitrarily complex. Functions may contain any amount of mathematics (and indeed internal variables). Any internal specification for a Function is irrelevant to the SE and is thus irrelevant to this discussion. How the internals of Functions are implemented in software is also irrelevant to the SE so long as there is a means of executing it (as a whole). Variables can be primitive or complex data types and/or structures. Implementations may even include methods associated with the reading and updating of the Variable (as for object-orientated languages).

During the lifecycle of a model’s iteration we define three distinct relationships between Functions and Variables (that are shared between multiple Functions). They are: Update, Read Pre-Update and Read Post-Update. These interactions are shown graphically in Figure 4:.

The Update relationship means that a Function may write to a Variable whether it actually does or not on any particular iteration of the model is a matter for the internal mathematics of the Function.

There may be many Functions updating the same Variable. Of these it may be that there are some, whose execution ordering is indeterminate meaning they could at least in principle be executed in parallel. From a mathematical perspective this is acceptable provided updates are commutative, that is at the end of all updates the state of the Variable will be the same regardless of Function execution order.

![Figure 4: Types of relationship between Functions and Variables.](image)

With the Read Pre-Update and Read Post-Update relationships a Function may read the state of a Variable but must not write to it. In terms of execution ordering the following rules apply (and illustrated in Figure 5:).

A Read Pre-Update type relationship between a Function and a Variable ensures that Function will be executed before any other Function that can update the Variable.

A Read Post-Update type relationship between a Function and a Variable ensures that Function will be executed after all other Functions that can update the Variable.

![Figure 5: Execution ordering of Functions relative to a Variable.](image)

Functions that have a Read Pre-Update relationship with a Variable will be reading the state of that Variable as it was last updated in some previous iteration of the model. Such Variables often represent state of that which is being.

When a Function is reading post-update from many Variables an implication of the above rules is that the Function cannot be executed until all other Functions that can update all these Variables have been executed (if indeed they need to be executed).

The modeling elements defined in the previous section can be easily incorporated into the standard idea of Component Models that use input and output Ports on their interface and child Component Models internally (Figure 6:) thus providing a means of specifying models hierarchically.

---

3 An additional reason for not executing a Function when iterating the model could be none of the Variables a Function reads from have been updated since it was last executed. A simulation environment supporting this would be useful for models that include discrete event behaviour.

4 A Function that writes to a Variable may also read from it, however for the purposes of defining execution ordering of Functions this is irrelevant.
Ports can be treated as Variables in that Functions relate with them in the same way as defined earlier.

3.2 Component Models Definitions

Figure 6: Component Model with internal Functions, Variables and their interactions.

4. NESTED MODELS OF COMPUTATION

To support the modeling of discrete systems we use a Model of Computation for a slight variation of the Discrete Time System Specification (1). We refer to this as the Discrete Time (DT) MoC.

To support the modeling of continuous systems we focus on a Model of Computation for the Differential Equation System Specification (1). We will refer to this as the Differential Equation Solver (DES) MoC.

In a hybrid model where multiple MoC are working together there must be something in common mathematically to allow them to coordinate their operation. The DT MoC described here has the necessary features to allow multiple independently operating instances of the DES MoC to be nested within an instance of the DT MoC.

MoC, being independent of that which is being modeled are usually embedded into the Simulation Environment (SE). An ambition of this paper is support multiple localised instances of the DES MoC in particular (to allow different parts of the model to solve differential equations independently). We refer to an instance of a MoC as an Iteration Controller (IC).

IC interact between each other and with the modeler’s contextual mathematics through special control Variables, some of which are instantiated along with the IC while others are declared by the modeler within the contextual part of model.

For the purposes of this paper we need only elucidate the common mathematics and behaviour necessary to show how the MoC work together and relate to Functions as defined earlier.

4.1 Discrete Time MoC

In the DT MoC time $T_{i+1}^{DT}$ is advanced after the $i^{th}$ iteration of the DT MoC, such that

$$T_{i+1}^{DT} = m \min (T_{n}^{DT}, T_{m}^{DE}), n = 1 ... N, m = 1 ... M, \text{ and } T_{i+1}^{DT} > T_{i}^{DT}$$

Where, $T_{n}^{DT}$ is one of N control Variables defined in the contextual part of the model and updated by a Function to a future time value, and $T_{m}^{DE}$ is a similar control Variable owned and updated by one of M DES Iteration Controllers.

The $T_{i}^{DT}$ control Variable is owned and updated by a DT Iteration Controller. When appropriate its value is assigned to the global time control Variable $T$ which in turn can be read post-update by Functions. The $T_{i}^{DT}$ control Variables are declared by the modeler (Figure 7:).

Figure 7: Graphical representation of control Variables for Time and future time for an iteration of DT IC.

4.2 Differential Equation Solver MoC

The DES MoC utilises numeric integration algorithms to calculate the integral values at a future time given the integral and derivative values calculated in the current iteration of the model’s contextual mathematics, i.e. the differential equations.

There are two relevant categories of numeric integration algorithms, causal and non-causal. Causal algorithms require one iteration of the contextual in order to determine a solution for the future integrals, while non-causal algorithms require multiple iterations. Higher order (higher accuracy) non-causal algorithms perform a series of low order integral estimates at algorithm specific times across integration interval. These estimates occur within their own iterations of that part of the model that calculates the derivatives.

This is in addition to the need to calculate the derivatives given once the higher order integrals solution has been calculated. It is only on the solution iteration of the DES IC and associated contextual mathematics that collectively conform to the rules of the DT MoC hence that iteration of the DES IC can run as part of the surrounding DT IC iteration. The iterations calculating low order estimates however must be performed at the end of the surrounding DT IC iteration.

After the last estimate iteration of a DES IC it updates its $T_{m}^{DE}$ Variable such that:

$$T_{i}^{DE} = T_{i}^{DT} + h$$

$^{5}$ Variables are not shared with the contextual parts of the model.

$^{6}$ DT and DES MoC need keep their own time Variables. The contextual parts of models do not need to make the distinction hence a global time Variable is defined which the ICs update appropriately.

$^{7}$ Non-causal integrators have an advantage in that they do not need to retain a history of multiple integral solutions for higher order solvers.
where $T_i^{\text{DT}}$ is the time used in the current iteration of the DT IC and $h$ is the integration step size of the numeric integrator.

This indicates to the surrounding DT IC the time at which the DES IC next needs to provide the solution for the integrals it calculates.

The structure of the relationship of a DES IC and associated contextual mathematics that calculates with a surrounding DT IC is well illustrated in Figure 8: using a pseudo text-based language.

\[ S = S_0 \]

\[ Y_i = f_i(S) \]
\[ S_i = f_i(Y_i) \]

\[ Y_2 = f_2(S_1, S_2) \]
\[ \frac{ds_2}{dt} = f_2(s_1, Y_2) \]

\[ \text{DT IC} \]
\[ \text{DES IC} \]
\[ \text{DT MoC Mathematics} \]
\[ \text{Contextual Mathematics (discrete state calculations)} \]
\[ \text{DES MoC Mathematics} \]
\[ \text{Contextual Mathematics (derivatives calculation)} \]

**Figure 8: Structure of nested DT and DES Iteration Controllers using a pseudo text-based language.**

A DES IC shares integral and derivative Variable pairs with contextual mathematics. These Variable pairs are declared by the modeler (Figure 9:).

\[ \text{Integral/Derivative pair} \]

\[ \text{Function} \]
\[ \text{Function} \]

**Figure 9: Graphical representation for integral/derivative Variable pair.**

The derivative Variable must be updated by at least one Function but must not be read pre-update. The integral control Variable can only be read post-update by Functions.

### 4.3 Hybrid Component Models

The contextual mathematics in various parts of a model must be compatible with the MoC under which they execute. Within a hybrid model where more than one MoC is required for execution it stands to reason that some mathematics will only be compatible with a specific MoC while other mathematics will be compatible with any MoC, for example simple algebraic calculations.

With text-based languages the mathematics is generally structured together according to the MoC (Figure 8:). However with graphical languages the model is structured more from the perspective of the systems being modeled. This means that even the simplest component models, i.e. those composed entirely of Functions, may well contain different parts of mathematics that are dependent on or independent of specific MoC (Figure 10:). Indeed with Functions themselves, as defined in this paper being arbitrarily complex internally, they too could have this hybrid characteristic.

**Figure 10: Hybrid Component Model.**

With Functions it will be necessary for the modeler when implementing their internal mathematics to guard mathematics that depends on specific MoC from being executed under an inappropriate MoC.

The implication is that Functions must be given access to IC specific Variables that indicate when a Function is being executed under the MoC the IC instantiates.

Returning to the DES MoC if we wish to encapsulate solely the mathematics of a selected set of differential equations under a DES MoC, there is no natural structure to do so within a model hierarchy reflecting the systems being modeled. The only structures that exist are the component models that may well contain a hybrid of mathematics. If we use an arbitrary component model as a basis of encapsulation for a DES IC it is necessary to determine which of the Functions within it need to be executed under that DES IC. The idea is that through the Function and Variable definitions the Simulation Environment should be able to analyse a component model to make this determination.

The method for determining which Functions are to be executed by the DES IC is as follows.

1. Identify all the Integral/Derivative pairs that are in scope (see section 4.4).
2. Trace backwards from the Derivative Variables from step 1 through Read Post-

---

Additional Variables may be used to reset the integral at a discontinuity (and initialise it) and to influence the way in which the numerical integration algorithm operates, however these are beyond the scope of this paper.

9 Most graphical mathematically formal modelling languages inbuilt predefined mathematical Functions thus the MoC under which they can be used are already known.

10 For Functions that are exclusively dependent on one MoC or are totally independent of any MoC a simple declaration of such would suffice if the Simulation Environment can then take responsibility for execution of the Function under the correct MoC.
Update and Update relationships between Variables and Functions that are either dependent on the DES MoC or are Independent of any MoC. Stop at Functions that are purely dependent on the DT MoC. Input Ports on the component model encapsulating the DES IC or Integral Variables from step 1. All the Functions traversed in step 2 will constitute the Functions that will be under the control of the DES IC. Note it is also necessary to trace through any child Component Models in the same manner that do not have an encapsulated DES IC of their own with the exception the process does not stop on child model Input Ports.

Consider Figure 11: as an example. The analysis would identify F1 and F2 plus any any identified Functions within the child Component Model. Note there are some not unresolvable issues in regard to Variables feeding into and leading out of the Functions under a DES IC control. The issues revolve around what values are maintained in these Variables and how that is done when execution transitions between the DES IC and surround DT IC. These subtle issues are beyond the scope of this paper.

The scope of control for the DES IC, being that of the Integral/Derivative pairs and Functions declared by the modeler, is bounded by the Component Model in with it is declared plus all child Component Models that do not have their own DES IC declared.

These rules are illustrated in Figure 12:

![Figure 12: Illustration of demarcation of DES IC scope of control.](image)

It must be stressed that although the declaration of DES IC at various levels in the model hierarchy may give the appearance of nesting. Mathematically this is not the case. Mathematically DES IC can only be nested within a DT IC. This highlights the separation between the systems view of the model and the mathematical view.

### 5. SUMMARY

In this paper we have argued that graphical languages supporting hierarchical views of a model are orientated towards representations of the systems being modeled, and in doing so the user loses sight of some aspects of how the model mathematics is executed.

A solution was presented that allows a modeler to explicitly specify by a graphical means, the order in which a simulation environment will execute functional components of their model.

Furthermore a means was presented for a more flexible approach to modeling complex, hybrid discrete and continuous systems. It allows a modeler to arbitrarily encapsulate the management of numerical integrators in different parts of a model hierarchy allowing the employment of different integration algorithms and or different integration step sizes in order to accommodate system models of varying integration accuracy needs.

The benefits of these are: to make more apparent to the modeler the iterative algorithmic nature of their model; allow the creation of standard simulation environments that execute models with arbitrary internal mathematics through analysis of the model definition; and to allow complex simulations of hybrid discrete and complex systems to execute more efficiently.

### REFERENCES

Papers are limited to 6 pages in length. Authors should refer to the Guidelines For Authors for further details.

Section titles are 10 pt, bold, Times, left justified, and all caps. Sections are numbered consecutively through the manuscript. There is a 12pt space before and a 4pt space after the section title. The section title commences 1cm from the left margin of the column.

The manuscript is formatted for A4 paper with a 3cm inside margin and a 1.5cm outside margin (all papers will start on an odd numbered page where the inside margin is the left hand margin). The paper is to be presented in two columns, fully justified, with a .75cm gutter between the columns.

The main text is set in Times, 10 pt. Single space between lines of text. Do not indent paragraphs. Leave a 4pt spacing between paragraphs at the end of each paragraph. Text is fully justified.

Do not number the pages in the manuscript.

5.1 Subsections

Subsection headers are set in 10 pt, Times, bold, left justified with the first letter of each word capitalized. Subsections are numbered consecutively through the section. There is a 12pt space before and a 4pt space after the subsection title. The subsection title commences 1cm from the left margin of the column.

5.2 Tables

Tables can either span columns or be limited to a single column. Tables are centered on the column (page). Text within table is set in 10pt Times. The text in the table is set in single space, no leading or following spacing between rows, left justified, single line borders for all cells in the table. Column headers are set in 10 pt Times, bold, centered on the column, with the first letter of each word capitalized. There is a 2pt space before and after the column headings. The table caption precedes the table, centered on the page/column, set in 10 pt Times in plain text, and does not end in a period. The word “Table”, the Table number, and the colon are set in bold. Tables do not break across pages or columns. There is a 4pt space before and after the table caption.

<table>
<thead>
<tr>
<th>Table Component</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table text</td>
<td>10 pt Times, left justified, single spaced.</td>
</tr>
<tr>
<td>Table caption</td>
<td>10 pt, Times, centered</td>
</tr>
</tbody>
</table>

5.3 Figures

Figures can either span columns or be limited to a single column. Figures are centered on the column (page). Each figure must be followed by a caption and numbered consecutively throughout the document. The figure caption follows (is placed beneath) the figure, centered on the page/column, set in 10pt Times in plain text, and does not end in a period. There is a 2pt leading space before the caption and a 3pt trailing space after the caption. Figures do not break across pages or columns.

**Figure 13:** Example figure placed in text.

5.4 References in text

One of two modes for referencing is acceptable – as outlined below.

1.4.1 Using Endote

References are presented in the text using the format of (author, year) e.g. (Gonick & Smith, 1993). Multiple references in the same location are separated by commas within the single brackets e.g. (Carse, 1987; Duke & Geurts, 2004)

1.4.2 As per previous mode

References are presented in the text in numerical order, enclosed by square brackets. Multiple references in the same location are separated by commas [1],[2].

5.5 Footnotes

Footnotes are indicated by subscripted numbers throughout the manuscript. Footnote text is set in 8pt Times, and should not break across pages. Footnotes are fully justified, separated from the text by a single line that spans 2/3 of the column.

---

13 This is an example of a footnote.
14 This is another example of a footnote.
6. SECOND SECTION

6.1 Second Level - Use Heading 2 Style

Second section text.

6.1.1 Third level - Heading 2 style.

For further subsections, use Heading 2 as the base style and indent one more level. These are outline styles so you can use the indent key in the formatting toolbar to push down one level.

7. THIRD SECTION

7.1 Subsection - use heading 2 style again

Third section text.

REFERENCES

4.1 If using Endnote

The reference section will automatically be arranged in alphabetical order by Endnote, as per the entries here.

4.2 Reference by Endote


4.3 Using the previous system

References are numbered, listed in alphabetical order (by first author’s last name) and fully justified.

The information in a citation is presented in the following order: author, year of publication, title of paper, publication title, volume & issue number, month of publication, pages. Authors are listed in the order of their appearance in the technical paper, last name first, then first initial and middle initial. Names are separated by a semi-colon. The title of the paper is quote delimited. The title of the publication (such as a journal or conference) is italicized.

4.4 Previous System
