MODSIM 2003 conference, Townsville, Australia
14-17 July 2003

Full, peer-reviewed paper

It's TIME for a New Environmental Modelling Framework

J.M. Rahman[a,b], S.P. Seaton[a,b], J-M. Perraud[a,b], H. Hotham[a,b], D.I. Verrelli[c,b] and J.R. Coleman[a,b]

[a]CSIRO Land and Water, GPO Box 1666, Canberra, ACT, 2601, joel.rahman@csiro.au

[b]Cooperative Research Centre for Catchment Hydrology

[c]Department of Civil and Environmental Engineering, The University of Melbourne, Victoria

Abstract: The Invisible Modelling Environment (TIME) is a new environmental modelling framework being developed within the Catchment Modelling Toolkit project in the CRC for Catchment Hydrology. TIME differs from existing modelling frameworks in a number of ways, particularly in its use of metadata to describe and manage models as well as the flexibility given to model developers to 'pick and choose' the components of TIME relevant for a given project. Functionality that is embedded as an immutable 'core' layer in other frameworks is included in applications under TIME on an as-needed basis using optional, interchangeable components. This flexibility extends to components that manage data and models, recognising that one approach does not necessarily fit all applications. TIME includes a number of small framelets supporting extension in key areas such as data representation and visualisation. All fundamental data types, such as rasters and time series, are defined within the data framelet, which supports the definition of new, compatible data types. The visualisation framelet allows the definition of 'layers', each providing a visual representation of some type of data, such as rasters or polygons. Multiple layers can be placed on a single 'view', such as overlaying a polygonal map on a raster. Views can be surrounded by 'decorators' such as axis and titles, each of which can be combined independently. TIME includes a number of tools, which operate generically on models, including an automatic user interface generator and various model optimisation tools. TIME is developed on the Microsoft .NET platform and supports the development of models in a variety of languages, including Visual Basic.NET, Fortran 95.NET, C# and Visual J#. TIME is currently being used to develop a range of modelling applications, including a library of rainfall runoff models and a model supporting assessment of stream ecosystem health under various flow scenarios.

Keywords: *Modelling Frameworks, .NET, Catchment Modelling Toolkit, TIME*

1. INTRODUCTION

The Catchment Modelling Toolkit project (the Toolkit) within the Cooperative Research Centre for Catchment Hydrology (CRCCH) is producing a cohesive suite of environmental modelling applications. Central to the delivery mechanism is the use of environmental modelling frameworks to allow models to be developed and integrated quickly and consistently.

Environmental modelling frameworks are software products that provide support for the development of new environmental simulation models. While the operational characteristics of frameworks vary considerably, certain common elements apply. Frameworks typically provide a template for new model components, along with support and visualisation for common data types.

Several frameworks were evaluated and used extensively for the toolkit project before a technology opportunity made it feasible to develop a new framework that better suited the needs of environmental modellers within and beyond the CRCCH.

2. MODELLING FRAMEWORKS

Domain specific software frameworks have been an active area of research in software development (Gamma et al., 1994) as well as many disciplines for some time, and environmental modelling is no exception (Argent et al., 2001).

Frameworks for environmental modelling typically support rapid model development and integration by generic, reuseable components for data handling and visualisation. Some frameworks go as far as to provide custom development tools, such as modelling languages (Reed et al., 1999) or graphical design tools (Maxwell and Costanza, 1997). Other frameworks, such as Tarsier, rely on model developers using third party development tools and languages (Rahman et al., 2003a).

Both approaches have their pros and cons. For example, it is much easier to achieve good runtime performance using commercially available compilers than by developing a domain

specific language. Custom modelling languages often lack the flexibility of a commercial development tool that may limit their applicability to larger modelling applications. However, when using a custom modelling language, it is much easier to shelter the model developer from language details such as memory management and provide a much more customised experience. Furthermore, by writing a custom language interpreter or compiler, it is possible to write framework components that reason about models in other ways, such as creating user interfaces based on the variables and functions found in the model code (Rahman et al., 2003b). Commonly, frameworks based on custom modelling languages are suitable for model developers with little programming experience in commercial programming languages, while developers needing to create more customised user interfaces will typically adopt a framework that uses a commercial development tool.

TIME attempts to find a middle ground, by using a commercial development platform (.NET) that supports the dynamic discovery of system properties at runtime. .NET allows the elementary integration of components written in different languages, including Visual Basic, Fortran and C++ (Meyer, 2001). In each case a .NET specific compiler, for a given language, is required. Additionally, .NET provides a language independent mechanism for discovering components, and the properties of components, at runtime. These properties include inherent properties, such as the class structure and the fields and methods of a class, along with custom metadata tags that can be defined by application and framework developers.

TIME makes use of the metadata capabilities of .NET to automate several tasks, such as user interface generation, that are not possible with many modelling frameworks based on commercial development tools. By automating these common tasks, the model code does not become directly coupled to the implementation of those tasks, relieving model developers from code maintenance tasks stemming from framework evolution.

3. TIME

TIME is a new environmental modelling framework intended to support several key stages of model development. TIME supports the development of new model components, using one of a number of languages, along with the testing of those model components in a generic test-bed, providing a high level of data handling, analysis and visualisation.

TIME then supports the integration of modules into applications with highly customised, visually rich user interfaces, using a number of reuseable components for data handling and visualisation.

TIME includes support for a number of key data abstractions for environmental modelling, including Rasters, Time Series, Points, Lines and Polygons and Node Link Networks. Additionally, reuseable components support data analysis, visualisation and high level model processing.

3.1. Architecture

At the highest level of abstraction, TIME is represented as a layered system (Figure 1), with components in each layer interacting with the layers below it. Each layer contains a family of components and, in some cases, small frameworks supporting a specific aspect of model development. These small frameworks, or framelets (Pree, 2000), handle aspects such as visualisation, IO and user options management.

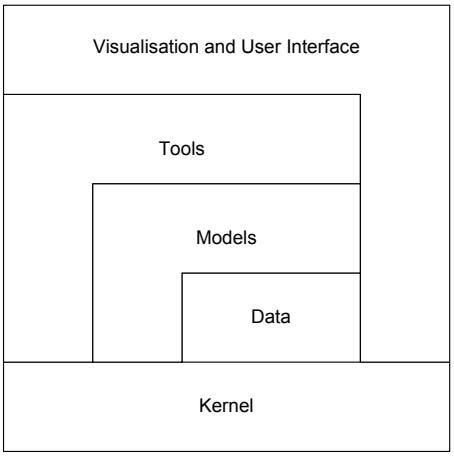

Figure 1: Main Architectural Layers of TIME

The core framework is contained in the Kernel layer, which contains definitions of the various metadata tags, as well as the abstract parent classes for models and data representations. The Data layer contains the key data representations, along with various mechanisms for performing data IO operations. The Models layer contains all the modelling components in the system and is the layer in which most developers create components. The Tools layer provides various components that perform generic processing of data and models, including data statistics and parameter optimisation. The Visualisation and User Interface layer contains the visualisation framelet, components for visualising each data type as well as various high level components for user interaction.

3.2. Core Framework

The Kernel layer is sufficient to develop many simple models that do not explicitly deal with data objects such as Rasters or Time Series.

The set of metadata tags (Table 1) includes tags for classifying and documenting properties of models, as well as tags for classifying the scope of components.

Tag	Used To	Applied To
Input	Classify model fields	Fields
Output	Classify model fields	Fields
Parameter	Classify model fields	Fields
State	Classify model fields	Fields
Minimum	Minimum allowable value	Fields
Maximum	Maximum allowable value	Fields
WorksWith	A component works with a particular type of data	Classes
Ignore	Exclude component or field from generic tools	Classes, Fields
Summary	Provide free text description of fields or components	Classes, Fields
CalculationUnits	Specifies the Units the component uses internally	Fields
DisplayUnits	Specifies the Units that should be used to display inputs and results	Fields
UserOption	Flags a field as a default the user can change and maintain across sessions	Field

Table 1: Metadata tags defined in TIME Kernel layer

The core framework is made up of the abstract parent classes `Model` and `Data`, the interface `Geometry`, and the support class `Subject` with its corresponding interface, `Observer` (Figure 2). `Model` is the parent class of all modelling components and includes the abstract method `runTimeStep`, implemented by all child classes.

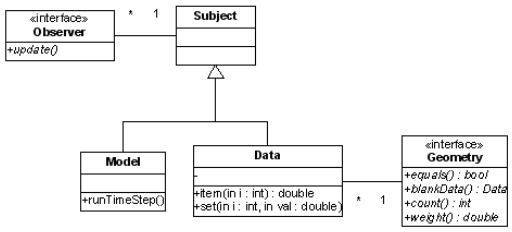

Figure 2: Core Framework, including Data/Geometry framelet.

`Data` is the parent class of all data types and includes methods for generic, one dimensional access to all data types. All child classes of Data must implement the `item( in i : int )` query and the `setItem(in i : int, in val : double )` method to provide this common interface. The one dimensional methods are typically used by generic tools, such as the data statistics, that do not depend on the spatial or temporal context of the data. Child classes are free to add additional accessor methods that take advantage of their context, such as two dimensional (row, column) indexing of rasters, or access to time series' based on date and time.

The context of a data object is specified by its `Geometry`, with all data objects sharing a common `Geometry` considered 'compatible' for the purposes of various operations, such as mathematical operators (+, -) or custom methods, such as regression. This separation allows efficient representation of spatial data, such as polygonal coverages, linked to attribute tables. The shape and location of polygons is represented by a shared `Geometry`, while each column of the attribute table becomes a `Data` object that can be processed and manipulated generically.

Following the separation of data and geometry, most data types are represented by two classes:

- A class implementing the `Geometry` interface and storing the spatial / temporal context of the data, and
- A class representing values stored by the data type, made accessible using both the generic, one dimensional accessor methods and any additional data type specific methods.

The `Subject` parent class of `Model` and `Data` allows `Observers` to subscribe to model and data objects to receive notification of changes (Gamma et al., 1994).

3.3. Visualisation Framelet

The Visualisation Framelet (Figure 5), within the Visualisation and User Interface Layer contains parent classes for all components involved in the visualisation of data. These include the Layer and ViewDecorator. Both graphs (such as time series plots and scatter plots) and spatial maps are handled by the same basic system.

Subclasses of `Layer` contain logic to draw a particular representation of some data type instance onto a `Canvas`. A `Canvas` can contain multiple layers overlaid on the one plot. `Canvas` manages the drawing of individual layers, translating from the world coordinates of a layer to screen or device coordinates. A `Canvas` may be 'decorated' by one or more `ViewDecorators`,

such as `Axis`, `Titles` and `Legends`. The arrangement of Canvas and `ViewDecorator` follows the Decorator design pattern (Gamma et al., 1994), with both the canvas and the decorators implemented as subclasses of `View`.

Various components can make use of `View` objects, including printing components, bitmap generators and web-based mapping tools. The most common such component is a `ViewControl` that allows a View to appear on an area of a window, within a graphical user interface.

Complementary components provide generic functionality for user customisation of visualisations and for defining user interactions with plots (such as zoom, pan and interrogate).

3.4. Miscellaneous Framelets

The TIME code base includes several other loosely coupled framelets for several key aspects of model and application management. The IO subsystem abstracts the operations of reading and writing data in different file formats from data sources (typically files). The User Options component provides a generic mechanism for managing application customisation by users, using a metadata tag (UserOption) and a management component that stores values for options in the Windows Registry. The Units component (Figure 3) represents common units (such as mm, $mg.L^{-1}$ or m^3) and provides conversions between compatible units. Units are represented as either Simple Units (length, mass, time or currency), or Compound Units that combine existing units using multiplication or division. Thus a concentration unit is a mass unit divided by a volume unit, where the volume unit is the cube of a length unit.

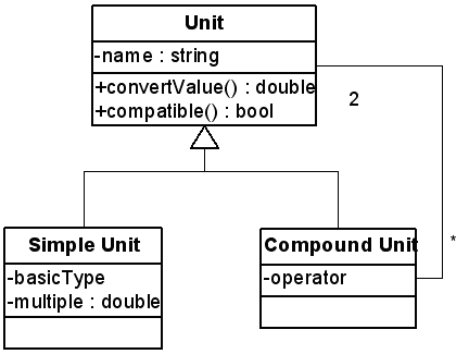

Figure 3: TIME representation of Units, with Simple Units and Compound Units.

4. GENERIC COMPONENTS AND TOOLS

The Data, Tools and Visualisation & User Interface layers (Figure 1), provide numerous generic components built with the core framework and the additional framelets. These components are used heavily by TIME developers when writing and testing model components, as well as application developers integrating modelling components into polished applications.

4.1. Data Types and Data Processing

TIME includes built in support, including IO, for a range of data types (Table 2).

Data Type	Summary
Raster	Two dimensional regular grid of data, located within a geospatial context
TimeSeries	Temporal arrangement of data on one of several fixed timesteps
Node Link Network	Abstract representation of physical networks, such as river systems
Sites	Collections of points in space
Poly Lines	Linked collection of multi segment lines
Polygons	Collection of closed polygonal regions
Cross Sections	Surveyed, or generated river cross sections
Arrayed Data	Ordered list of values with no spatial or temporal context

Table 2: Standard Data types within TIME

In addition to many data type specific processing tools, such as terrain analysis of rasters, TIME includes many operations that act on any data type. These include mathematic operations (such as adding two data objects together), statistics (such as the mean of a data object or the correlation between two objects) and a rules-based data processing engine.

4.2. Visualisation

The Visualisation and User Interface layer contains numerous components built with the Visualisation framelet. This includes standard Layers for each of the main data types except Arrayed Data, along with special layers for displaying alternative representations of data. These include scatter plots, that take any two data objects of the same type (e.g. two time series, or two rasters), cumulative frequency graphs and flow duration graphs and probability density plots.

Various Decorators can be used to improve the appearance and utility of visualisations, including axis, titles, labels and legends. All Layers and Decorators provide numerous options for

developers and users to tailor the appearance of their visualisations.

4.3. Model Processing

A key advantage of the metadata based environment of TIME is the support given to developing 'model processing tools' (Rahman et al., 2003b). Model Processing Tools provide generic functionality by adapting to new models automatically at runtime. The tools are able to investigate the inputs, outputs and parameters of a model and discover additional information, such as the numeric range of parameters.

TIME currently includes a range of tools that make use of the metadata, including the automatic generation of graphical user interfaces and command line interfaces for models and several parameter optimisation tools. The interface generators provide tailored components for each field of the component, allowing users to select inputs and parameters and view state variables and outputs.

For time-stepping models, the interface generators are able to automatically attach input time series to any double precision input, and record output time series of any model output or state variable. This saves the model developer the effort of using and creating time series objects, while allowing the new model component to remain flexible, with configuration decisions being made at runtime rather than compile time. This is particularly important for model components that may be used extensively throughout a large integrated application. When testing a single instance of such a component, it may be desirable to store time series of each output and state variable. However, storing many time series for each instance of the component in the context of the integrated model may be unnecessary or even impractical, given memory constraints.

5. USING TIME

Developers can use TIME to create model components, as well as modelling applications. A model component encapsulates the core scientific algorithm of a model, deferring administrative details such as data handling, visualisation and user interaction to other components. Model component developers use the generic tools of TIME as a test bench for using and calibrating the model.

A modelling application encapsulates one or more model components, into a high level user interface that uses and tailors various TIME components.

Modelling components can be written in one of several .NET languages, while modelling applications can combine components in different languages.

5.1. Developing Models

All models are implemented as a class within a .NET language. Model classes inherit from the parent class, Model, and implement the abstract method `runTimeStep()`. Additionally, models define fields for their inputs, outputs, parameters and state variables, and document them accordingly with metadata tags.

Figure 4 shows the complete code for ToyModel, a simple TIME model written in C#. ToyModel includes two inputs, `rainfall` and `actualET`; a state variable, `netRainfall`; a parameter, `coefficient` and an output, `runoff`. The variable declarations for each have been marked with appropriate metadata tags that can be used by model processing tools within TIME.

```
using System;
using TIME.Core;

public class ToyModel : Model {
  [Input,Minimum(0.0)] double rainfall;
  [Input,Minimum(0.0)] double actualET;
  [State] double netRainfall;
  [Parameter,Minimum(0.0),Maximum(1.0)]
    double coefficient;
  [Output] double runoff;

  public override void runTimeStep( ) {
    netRainfall =
      Math.Min( 0.0, rainfall-actualET );
    runoff = coefficient * netRainfall;
  }
}
```

Figure 4: Example TIME Model in C#.

The source code for model components typically only references components in the kernel and data layers of TIME, reducing the amount of the API that developers must learn. When testing and using models however, users have access to many generic tools found in the Tools and Visualisation layers. These include graphical or command line user interface generation for the model and parameter optimisation. The generic tools also support attaching data to inputs and outputs, such as attaching an input precipitation series to the rainfall property of ToyModel.

5.2. Developing Modelling Applications

Modelling application developers create TIME-based applications using a .NET language and the

suite of data, model, tool and visualisation components in the framework.

The Rainfall Runoff Library (Perraud, 2003), represents a significant application built within TIME and integrates models with data, visualisation and optimisation tools within a sophisticated user interface.

Application development requires a broader and deeper understanding of TIME than developing model components, but the use of many reuseable components simplifies this task considerably.

6. CONCLUSION

Modelling frameworks simplify, and allow generalisation of a number of aspects of model development, including coding, model integration and data visualisation. Key differences between frameworks typically include tradeoffs of performance and usability between frameworks based on commercial development tools and those utilising custom model development languages. TIME is a new framework that attempts to achieve a 'best of both worlds' configuration by using a commercial development environment that offers similar advantages of usability and customisability of custom languages. This decision, along with a depth of functionality in traditional framework areas, such as visualisation, allows TIME to be learned quickly and used effectively for the development of individual model components, while still providing a foundation as well as customised modelling applications.

7. REFERENCES

Argent, R.M., R.A. Vertessy, J.M. Rahman and S.P. Seaton, From Components to Decisions: The Role of Software Engineering and Frameworks in Catchment Decision Support, in *Proceedings of MODSIM 2001*, Canberra, December 2001, Vol 4, pp 1589-1594.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: elements of reusable object oriented software*, Addison Wesley, 1994.

Maxwell, T. and Costanza R., A language for modular spatio-temporal simulation, *Ecological Modeling*, 103, 105-113, 1997.

Meyer, B., .NET is Coming, *Computer*, pages 92-97, August 2001.

Perraud, J-M., G.M. Podger, J.M. Rahman and R.A. Vertessy, A New Rainfall Runoff Software Library, *Proceedings of MODSIM 2003*, Townsville, July, 2003.

Pree, W. and Koskimies, K., Framelets – Small and Loosely Coupled Frameworks, *ACM Computing Surveys*, 32, 1es, 2000.

Rahman, J.M., S.M. Cuddy and F.G.R. Watson, Tarsier and ICMS: Two Approaches to Framework Development, *Mathematics and Computers in Simulation*, 2003, in press, 2003a.

Rahman, J.M., S.P. Seaton, and S.M. Cuddy, Making Frameworks More Useable: Using Model Introspection and Metadata to Develop Model Processing Tools, *Environmental Modelling and Software*, in press, 2003b.

Reed, M., S.M. Cuddy, and A.E. Rizzoli, A framework for modelling multiple resource management issues - an open modelling approach. *Environmental Modelling and Software,* 14, 503-509, 1999.

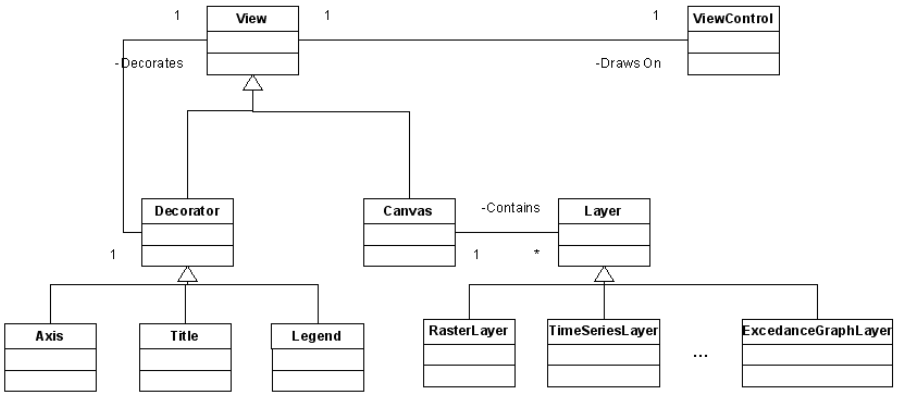

Figure 5: TIME Visualisation Framelet, with a ViewControl containing a single View, which can be either a nested set of Decorators, or a Canvas. A Canvas can contain multiple Layers, each responsible for drawing some data in a particular style.