Tarsier and ICMS: Two Approaches to Framework Development

J.M. Rahman[a,b], S.M. Cuddy[a,b] and F.G.R. Watson[b,c]

[a] *CSIRO Land and Water, GPO Box 1666, Canberra, Australia 2601 (Joel.Rahman@cbr.clw.csiro.au)*
[b] *Cooperative Research Centre for Catchment Hydrology*
[c] *California State University, Monterey Bay, California, United States*

Abstract: Modelling frameworks provide models with support components that handle tasks such as visualisation, data management and model integration. Within these broad requirements different approaches to framework development are possible. Tarsier is a modelling framework that supports the development of models in a high level language, such as C++. This approach allows Tarsier model developers to craft object oriented solutions to large modelling problems. ICMS is a software system that supports the development of models in a custom modelling language that allows modellers with little programming experience to develop, integrate and visualise catchment models. Both frameworks provide sophisticated tools for model linking, data management, and data analysis and visualisation. By focusing on different user groups, Tarsier and ICMS have evolved into quite different environments, yet both satisfy the definition of a modelling framework. This paper concentrates on the components within each framework and the strengths and weaknesses of the different approaches.

Keywords: Modelling frameworks; Tarsier; ICMS; Framework design

1. INTRODUCTION

The term environmental modelling framework is being applied to a wide range of software products that provide structure and functionality to support the development and application of environmental modelling applications. Breaking this definition into its constituent parts we begin to see the scope of the term:

- A software product may be an application, an application shell or a set of libraries for developing applications
- Frameworks provide a common structure for software components and their interaction and functionality that relies on this common structure to perform operations that might typically be implemented individually for each application
- Modelling frameworks allow new models to be developed within the common structure and applied by a variety of users
- Environmental modelling frameworks focus their support on environmental modelling applications by catering to domain specific requirements such as spatial and temporal data types and visualisation tools.

Within this definition for environmental modelling frameworks, there is considerable room for different designs and implementations to emerge. Beyond immediately obvious aesthetic differences, environmental modelling frameworks tend to target different groups of model developers and users and provide these groups with a variety of structure and functionality. Other differences emerge during the use of a framework, such as speed of model execution or the ability to customise the appearance of a model.

We consider two environmental modelling frameworks, Tarsier and ICMS, which have evolved in different directions under different design goals and pressures. Tarsier [Watson et al., 2001] is a set of libraries, along with an application shell, that can be used to develop integrated environmental modelling systems with highly customised user interfaces, such as decision support systems. Tarsier provides functionality for managing the interaction between models, data and visualisation tools, as well as the IO requirements of models. Tarsier supports development of models using Borland C++. ICMS [Reed et al., 1999] is an application that supports the development, testing, integration and application of environmental models. ICMS

uses a custom modelling language and a set of simple data types to allow developers with limited programming experience to implement models. The Open Modelling Engine [Rizzoli, 1998] at the core of ICMS provides a common structure for models, while the ICMS application provides interactive tools for linking modelling components.

Tarsier and ICMS provide an interesting contrast on a number of framework design issues. The frameworks target significantly different portions of the model development community, which may drive some of these differences. Of most interest is the considerably different approaches the two frameworks take to managing data and model integration.

2. TARSIER

Tarsier is an environmental modelling framework centred on a kernel of model and data management functionality and an application shell used to house model user interfaces and data visualisation tools. Tarsier provides powerful dynamic visualisation tools to model users without requiring extra effort from model developers. Tarsier has evolved through its application to a number of modelling applications, including the South East Queensland Environment Management Support System [Watson et al., 2001]. By focussing on reusability of code, the framework has been able to evolve to the point that it supports multiple developers creating semi-autonomous modules that communicate within the Tarsier framework. By developing both the framework and its applications in a high level language, model execution speed suffers little from framework overheads.

2.1 Architecture

The Tarsier kernel defines a number of parent classes that provide the common structure for models, the protocols for communication between models and the use of data by models. Models are implemented as a subclass of the core kernel class User, inheriting the common model structure and protocols. The Tarsier model structure and communication protocols are based on the Observer pattern of client-supplier computing [Gamma et al., 1994]. The Observer pattern classifies objects as either Observers, objects that are interested in the state of another object, or Subjects, objects that can be observed. Observers register an interest in a Subject by "subscribing" to the object. When some property of the Subject changes, such as a cell of a raster changing value, all Observers subscribed to the Subject are notified. Tarsier classifies classes as Users or Usees. Users are modules that rely on (use) other Tarsier modules, where Usees are modules that can be *used* by Users. Data types, such as Raster and Time Series represent Usees, while all models, as well as visualisation tools and data analysts, are Users. Users are themselves a specialised class of Usees, allowing models to be composed of sub models, each of which uses data and other models. The Observer pattern decouples a usee from its users and common users from each other. For example, if a model manipulates a raster map, the model doesn't need knowledge of any visualisation tool accessing the map. The manipulation represents a change of some property of the map and triggers update notifications to all of the map's users. Any visualisation of the map is notified and given the opportunity to redraw. This decoupling of models and visualisation facilitates dynamic visualisation of all model results without overhead to the model developer. In addition to managing interaction between data and models, User and Usee also include support for model and data persistence. Model developers don't need to write code to save model configurations, such as parameter values or references to time series inputs.

The Tarsier kernel includes two classes for managing the Usees in the system, the Usee Registry and the Usee Manager. The Usee Registry keeps track of the types, or classes, of Usees that Tarsier knows about, while the Usee Manager keeps track of instances of those classes. Conceptually, we can think of the Registry as a catalogue of available models and modelling components and the Manager as the stock currently held in a warehouse. For example, the Usee Registry might contain entries for the Raster data class, the Time Series data class and a Rainfall Runoff Model class. Until the operator starts running a model the manager is empty, but once a rainfall runoff model is started, the manager will contain entries for the inputs to the model, such as a rainfall time series, the outputs, such as a runoff time series as well as the model itself.

All Tarsier modules are encapsulated in Dynamic Link Libraries (DLLs) that are loaded by the application shell at run time. Each DLL provides one or more capabilities, such as data types, models or visualisation tools, which are recorded in one of the registries. The registration process also provides a function responsible for creating an instance of the model. When a new instance of one of the capabilities is required, such as a new raster map, the kernel invokes the creation

function for that capability and adds a reference to the new object in the appropriate manager.

2.2 Model Development

Model developers use the Borland C++ Builder development environment to develop models and user interfaces for Tarsier. The key model components are a model class, inheriting from User and, usually, a user interface window, or form, inheriting from User Form. Both classes provide model specific functionality by implementing key functions left deferred in the parent class. Models implement the Execute function to provide the algorithm for a single timestep of the model, and RegisterFields to support the Tarsier IO system by recording properties of the model that should be saved to and restored from disk.

For very simple models, Execute typically includes the complete algorithm for the model. In more complicated models, Execute usually coordinates the algorithm by delegating work to other functions, such as data retrieval and output calculation.

RegisterFields informs the Tarsier kernel about key model properties, such as parameters and data inputs. The kernel supports a number of basic field types, including Boolean, integer, floating point and strings, along with reference types such as usees, and compound lists of other fields.

2.3 Model Integration

The Tarsier framework supports model integration based on the concepts of Users and Usees. If two models are using the same data they are implicitly linked and each will be notified when the other changes the data. When model A uses the output data of model B as input, A is able to respond immediately to data changes triggered by B. While this scheme works well in many situations, problems arise when loops, such as B inputting the output from A, occur.

Since Tarsier models are developed in a high level language, model developers are free to implement custom model integration schemes for individual applications. This approach was used in the EMSS [Watson et al., 2001] to integrate individual subcatchment models to a stream routing model.

2.4 Data Representation

Tarsier data types are stored in DLLs and registered as capabilities in the same way as models. The Tarsier kernel does not include knowledge of any data type and is not restricted to any particular set of data models. The Tarsier framework currently includes DLLs providing Raster, Time Series, Node Link Network and Polygonal data models.

2.5 Visualisation and User Interfaces

Tarsier provides powerful visualisation tools for each included data type, including a 3D renderer for raster terrain data. In addition to these standard tools, the framework includes a number of components for quickly developing model-specific visualisation aids, such as animated flow charts (Figure 1). By using the user interface design tool in Borland C++ Builder, and the Tarsier visual components package, developers can quickly design structural flowcharts for their models that react to changes in model state each timestep.

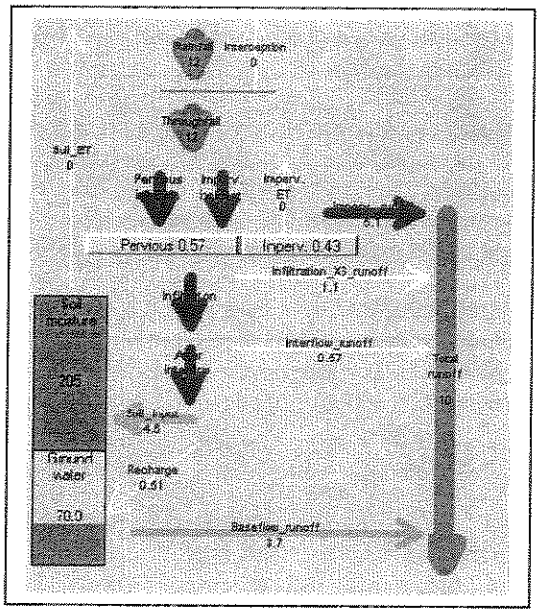

Figure 1. Stocks and flows view of a rainfall-runoff model in Tarsier.

3. ICMS

ICMS is an integrated model development environment based on the Open Modelling Engine [Rizzoli et al., 1998]. ICMS allows model developers with little programming experience to develop and integrate modelling components using the custom modelling language MickL and an interactive diagramming tool. ICMS compiles the MickL model to native 80x86 code that is activated at runtime. The compiled models run within the ICMS environment taking advantage of the system's dynamic visualisation tools. ICMS is

being used to deploy a number of modelling applications [for example Letcher et al., 2000] and has also served as an aid to teaching general modelling concepts. ICMS includes a rich set of model management tools allowing developers to package up libraries of models which can be distributed to other ICMS users. The visual approach taken to integrating models makes ICMS a powerful tool for describing problems and finding model structures that match the conceptual view of a system.

3.1 The Open Modelling Engine (OME)

The Open Modelling Engine (OME) is a software architecture for integrating modelling components. The OME represents modelling components as classes and instances of those classes. An OME class defines a series of data templates – properties of a component that can be used for input to, output from or calculations within the component – and one or more models. An instance of a class includes instances of each data template and is configured to run one of the models from the class.

Each model within a class is implemented as a series of three MickL functions: Initialisation, Finalisation and Main. The OME Runtime manager invokes these functions before, after and during model execution, respectively.

The OME structure provides a structure similar to traditional object oriented systems where classes and objects encapsulate both a set of data and the operations that are performed on that data.

3.2 MickL

MickL is a dynamically typed procedural programming language with implicit declarations and C style syntax. MickL shares C syntax for standard programming constructs, such as conditionals and loops, and includes a library of subroutines for manipulating data instances (Figure 2).

3.3 Data Representation

Each data instance of an object is a scalar, vector or a matrix at any point in time. These three simple types are sufficient, when combined with the higher level of abstraction offered by OME classes, for a large number of modelling applications. In many cases models treat matrix objects as grids with different semantics for mathematical operations such as multiplication.

```
function Main()
{
  // work out the outflow from the
  // generation rates and the land uses
  Outflow = GenRates * LandUses;
  return 0;
}
```

Figure 2. An example of ICMS MickL code.

3.4 Integrating Models

ICMS provides an interactive tool for integrating models. Using the System view (Figure 3), modelling systems can be constructed by placing individual components on a canvas and drawing links between them.

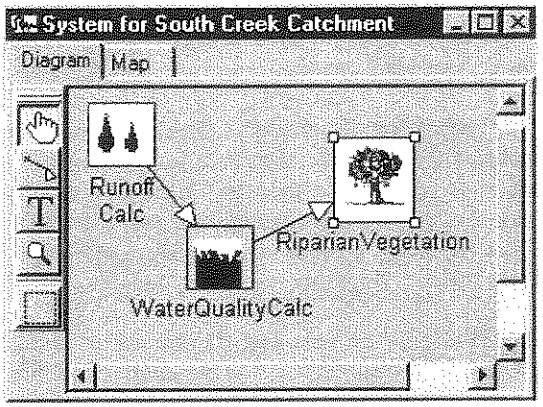

Figure 3. ICMS System View showing a set of linked biophysical processes.

Links between components are configured to determine what information is transferred and in what direction. The elegance and simplicity of the tool provides model developers with significant scope to design a modelling system. For example, the system view can be used to construct a system based on a "stocks and flows" approach, such as routing water through a series of links, or a process flow model, where each component performs one piece of the overall process.

ICMS uses the information entered in the system view to schedule model execution and transfer information between components.

3.5 Visualisation and User Interfaces

ICMS supports dynamic visualisation of each of the three fundamental data types, scalars, vectors and matrices. Each data type can be visualised in a spreadsheet view. Additionally, vectors can be viewed as line graphs and matrices can be represented as coloured raster grids.

Simple user interfaces for models are inferred directly from the OME class. These interfaces

consist of a tree view, similar to the Windows Explorer application, providing access to each data instance of a model. This system provides direct access to each parameter of a model and includes functionality for importing data sets, such as time series and rasters.

Development of custom user interfaces, such as decision support system (DSS) front ends, is achieved using the Borland Delphi development tool and producing and loading a DLL.

4. COMPARING THE APPROACHES

ICMS and Tarsier are two considerably different modelling frameworks and either one's suitability to a particular modelling application depends on a number of factors. Possibly the most important consideration is the different groups of model developers targeted by the two frameworks. Other factors include performance constraints and the need to customise user interfaces. Architecturally, the two frameworks differ considerably in how models deal with data and how models communicate.

4.1 Model Development Community

Tarsier requires model developers to have more programming expertise than developers using ICMS. For example, ICMS developers don't need to know about DLLs, the registration of capabilities with a kernel, or how the data management system works.

While requiring less programming experience, ICMS imposes more constraints on the structure of a model, possibly limiting its usefulness on large modelling projects. ICMS developers have limited scope to introduce new data types within their models or extend existing models using techniques such as inheritance. Tarsier rewards modellers with strong programming skills by giving more control over model structure and more scope in designing solutions.

The simple model development environment and the strong model structure, offered by ICMS, has made it an appropriate tool for use in courses teaching general modelling concepts, as opposed to a specific environment. Students are able to pick up the basics of the application and go on to learn the principles of modelling largely unhindered by technical environment issues.

While developing customised user interfaces for ICMS models still requires a traditional programming environment and the use of DLLs, the use of MickL for model coding allows a clean separation of responsibilities between a modeller developing a scientific algorithm and a programmer packaging a DSS.

4.2 Model Processing Capabilities

Model Processing is the ability of the framework to perform some operations on the model. There are advantages to having a modelling framework that acts as a "model-processing tool" analogous to the way a model is a data processing tool. Typical model processing tasks include saving the state of a model to disk or repeatedly executing a model in order to optimise parameters.

To perform these tasks automatically, the framework needs to know the various properties of a model, including its parameters and data requirements. This information is not always readily available, especially in traditional high-level languages such as C++, where compiled code retains no metadata about classes and their properties. ICMS gathers the required information from the MickL source code and the user's description of the OME class. Tarsier relies on the RegisterFields function, provided by each model, to get information about models. The Tarsier approach, while one of few approaches available with traditional languages, is error prone as it relies on the model developer initially providing information that correctly reflects the model structure and subsequently maintaining the information as the model evolves. ICMS has the advantage of processing the model code itself, allowing it to extract the required information automatically, avoiding the chance of mistakes.

4.3 Performance

The runtime performance of models in Tarsier is typically an order of magnitude faster than equivalent models in ICMS. While the performance of ICMS is not an issue on smaller applications, it is a major factor limiting the application of the system to large modelling systems.

4.4 Communication Between Models

The ICMS system view integrates models using explicit links between units whereas Tarsier allows models to be linked implicitly using shared data.

Explicit links implies there is some knowledge, recorded in the modelling system, that model A's output is linked to model B's input. With the implicit linking supported by Tarsier, models A

and B both share a data resource, C, but this is not recorded anywhere. There is no information that indicates that B is using the data as input to a modelling operation and not just calculating statistics on the data.

Explicit linking provides a number of advantages such as allowing the framework to control model scheduling and easily identifying feedback issues. Additionally, the scheme supported by Tarsier only allows usees, such as rasters and time series, to be shared.

The implicit linking scheme in Tarsier eliminates unnecessary data transfers as both models have direct access to the data they share. However, while data transfers conceptually occur in the explicit linking model, they can be eliminated using implementation techniques similar to the data sharing mechanism.

4.5 Dealing with Feedback

When integrating models, sooner or later a feedback situation arises where input to a model is in someway dependent on its output at some previous time. Both frameworks provide schemes for dealing with feedback.

In the explicit linking scheme in ICMS, the model scheduler uses the direction of links to determine a suitable order for execution. Execution begins with models that have no incoming links and proceeds to models whose only incoming links come from models that have already been executed. When a feedback loop occurs, the system cannot execute any of the models involved in the loop.

In Tarsier there is no overall scheduler, and models execute in response to update messages from their input data sources. Model execution results in new update messages being sent that may trigger execution of another model. When the result of running a model directly or indirectly results in that model *receiving* an update message, there is the potential for the system to enter an infinite loop.

Feedback can be handled differently in the implicit and explicit model integration schemes. In Tarsier, model execution results in a model being locked – placed in a state where it will ignore update messages – until execution completes. In ICMS, the model developer can use Delay components to implement feedback. The Delay component has one input and one output and places its input from the previous timestep on its output. ICMS disallows feedback loops that do not contain delay components. By using the delay component, the model scheduler is able to break the feedback loop at each delay component and schedule the system as if the loop does not exist.

5. CONCLUSION

Tarsier and ICMS represent significantly different approaches to developing an environmental modelling framework. While these two examples by no means cover the entire parameter range of environmental modelling frameworks, they do give good, contrasting, examples of the design decisions and tradeoffs that can occur in framework development. As the two frameworks evolve it seems likely that each will gain attributes of the other. Tarsier may gain ICMS' explicit model linking scheme and ICMS will provide more support for user interface development. As it stands, both frameworks support their own model development audience well, by providing an appropriate balance of support and flexibility.

6. REFERENCES

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

Letcher, R. A., S.M. Cuddy, and M. Reed, An Integrated Catchment Management System: A Socioeconomic Approach to Water Allocation in the Namoi. Hydro 2000 3rd International Hydrology and Water Resources Symposium of The Institution of Engineers, Australia. Perth, Western Australia. The Institution of Engineers ACT, 20-23 November 2000.

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.

Rizzoli, A.E., J.R. Davis, and D.J. Abel, Model and data integration and re-use in environmental decision support systems. *Decision Support Systems,* 24, 127-144, 1998.

Watson, F.G.R., J.M. Rahman, and S.P. Seaton, Deploying environmental software using the Tarsier modelling framework, proceedings of the 3rd Annual Stream Management Conference, Brisbane, Australia, 27-29 August 2001.