The architecture of the E2 catchment modelling framework

[1]Perraud, J.-M. , [1]S. P. Seaton, [1]J. M. Rahman, [1]G. P. Davis, [2]R. M. Argent and [1]G. D. Podger

[1]CSIRO Land and Water, [2]University of Melbourne, E-Mail: jean-michel.perraud@csiro.au

Keywords: *Catchment modelling; E2; TIME; Model development framework*

EXTENDED ABSTRACT

Catchment management is becoming a more complex task. The sharing of water resources between traditional water users and the environment, the introduction of water and contaminant trading and water quantity and quality targets has necessitated a whole-of-catchment modelling approach. A variety of software products address this need. They range from abstract, general-purpose modelling frameworks which are not tailored to catchment modelling, to software that is applicable only to a specific catchment. Many of these catchment models are 'hardwired' with algorithms that may not be appropriate for another catchment, or do not allow for the reuse of prior modelling knowledge. E2 is a catchment modelling framework within the Cooperative Research Centre for Catchment Hydrology (CRCCH) Catchment Modelling Toolkit. It allows the integration of components and models from other Toolkit products, and is designed to enable flexibility in model choice. The core capabilities of the framework extend beyond the modelling of physical processes in unregulated catchments, and the architecture was designed from the ground up to also enable the representation of regulated systems and ecological responses to physical conditions.

E2 has no intrinsic assumption about the time step of the input data and, with an appropriate choice of models warranted by the data available, can run over a range of time steps and spatial scales. The catchment is modelled using a structure of sub-catchments, nodes and links. These key elements have been designed as modular, extendable modelling units. E2 is built upon The Invisible Modelling Environment (TIME) and shares several of its characteristics, notably the use at run-time of model metadata attributes and reflection. In particular it relies on the use of abstract software interfaces, flexibility by composition of objects using a "plug-in" approach, and object oriented design patterns, wherever appropriate to enable this flexibility. This helps to foster minimal software coupling between the models dealing with different processes, and for each of these processes there is thus the possibility to choose from a library of candidate models. This flexibility extends to the user interface and the persistence mechanism used to save the system configuration. The user interface is designed to accommodate a choice of different methods when building scenarios, enabling, for example, alternate approaches to defining the node-link network. The use of "wizards" guides the definition of an overarching workflow between the tasks required to set up the network, and at many steps in this workflow an extensible list of alternate methods to perform the task is offered. The persistence mechanism uses a relational database, and can handle complex dependencies between objects in the system.

The high level features of E2 are described by Argent *et al.* (2005). The present paper describes the software design process and key architectural aspects in the various software layers that were required to enable the features and flexibility of the framework. The modelling engine, user interface, on-disk persistence mechanism and calibration tools all rely on the use of software interfaces, software reflection and various software design patterns to achieve flexibility and enable extensibility of the modelling framework. The modelling engine contains four main software elements: nodes, links, sub-catchments and functional units, and each can have alternate concrete software implementations. This engine also relies on a standardised software representation of mass balance and unit consistency throughout the system. The user interface relies on the use of a wizard with selected points of extension for alternate methods to perform a given task. The persistence mechanism uses software reflection to save and load model configurations, which has no overhead in terms of code for model developers. If objects are too complex to rely solely on software reflection, another mechanism using a software interface that still keeps the overhead at a minimum can be used. The calibration tool uses the builder and factory patterns to build composite parameter sets at run time by grouping model parameters.

1. INTRODUCTION

Catchment management requires increasingly an integrated modelling approach. There is a growing demand for modelling systems and decision support tools taking into account the biophysical, water management, economic and ecological processes taking place in catchments. Numerous software packages support the modelling of one or more of these processes. However the integration of these software packages to support an integrated approach is often a very difficult exercise, if it is to go beyond a relatively simple data exchange. Also many catchment models have fixed algorithms that may not make the best use of the modeller's prior knowledge of the catchment, nor have the flexibility to adapt the model structure for changing modelling requirements.

E2 is an integrative catchment modelling platform using the Catchment Modelling Toolkit (CMT) as its primary source of component models and tools. It is built upon The Invisible Modelling Environment (TIME, Rahman *et al.* 2003), and has been designed as an extensible framework enabling model choice. This paper presents the design and architecture of E2, and the software mechanisms that underpin it. Note that it is assumed that the reader has a good understanding of object-oriented programming.

2. KEY GOALS

E2 is a catchment modelling framework based conceptually on two previous software packages, the Environmental Management Support System (EMSS, Cuddy 2003) and the Integrated Quantity and Quality Model (IQQM, Podger 2004). The features of other well-known catchment and river system modelling tools (e.g. SWAT, QUAL2E) were reviewed in order to ensure there was enough flexibility in the product.

The key requirements for E2 were identified as:

- Enabling choice wherever warranted, e.g. rainfall-runoff modelling, flow routing model, but also for various methods of definition of the river network and sub-catchments,
- Being a transparent system, i.e. the modeller can explore the state variables of the models in depth if required
- Ability to model a variable list of conservative and non-conservative water constituents
- Including advanced tools facilitating the calibration of the models in a computationally efficient manner
- Minimising the up-front data requirements to start building a network,
- The modelling engine itself must not be tied to a fixed temporal or spatial scale,
- Allow for adapting the quantity of output data (e.g. number of time series) to the memory constraints of the computer,
- Designed from the ground up with the aim to implement water management rules and support ecological response models.

The high level architecture of E2 consists of three layers: user interface, modelling engine and handling of data input-output, a fairly standard approach in most recent softwares. This paper will illustrate the design of key architectural elements in these layers.

3. MODELLING ENGINE

Three sub-layers can be distinguished in the modelling engine itself:

- physical layer, embodied by the interconnected nodes, links and sub-catchments, dedicated to representing any biophysical process,
- management layer, that comprises the modelling units that deal with any human-induced process and which inspects and acts upon the physical layer, and
- a layer for tools acting on the previous two layers, e.g. a tool cropping a sub-network out of a full network, for calibration purposes.

E2 relies on a structure of projects and scenarios. A project is a container for a series of scenarios, and a repository for a central list of stored parameter sets that can be applied to the models in the scenarios. The scenarios in a project do not necessarily have the same spatial structure, since for example a scenario representing the addition of a dam would modify the structure of the river network.

Each scenario has a network runner, which is an aggregate of a network (a collection of sub-catchments, nodes and links), a component that centrally stores the temporal information of the simulation (start, end and time step), and two

software components in charge of recording time series from and feeding time series into the models. These last two components ("Player" and "Recorder") are central to addressing two of the core requirements: being able to record at run-time any model variable, and adapting the size of the data handling to the memory availability of the system.

Players and Recorders rely on software reflection (Rahman et al. 2004) to specify the association of a given time series and the property of a model that is played to, or recorded from. Software reflection is a cornerstone of the E2 architecture, and is used extensively in the modelling engine (e.g. to build parameter sets at run-time) as well as in the other main software layers.

3.1. Spatial sub-systems

E2 is a catchment modelling framework that uses a hierarchical, nested structure, especially with respect to the spatial scale. For each modelling sub-system defined at a certain scale or relating to a type of process, a software interface was defined whenever possible. The advantages of this approach were explained by Gamma *et al*. (1994). In E2 this fosters the decoupling of the software classes operating at different spatial scales or modelling separate physical processes, and is a key mechanism enabling model choice.

The catchment is broken up as a series of sub-catchments. Their outflows contribute to the node-link network representing the river system itself, as lateral inflow to the links. A standard representation of a sub-catchment is as a collection of Functional Units (FU), a generalisation of the concept of Hydrological Response Units. A FU is a part of a sub-catchment that has homogenous characteristics for the modelling purpose at hand. In most cases this means homogeneity in terms of hydrology and constituent generation. FUs do not have any explicit spatial representation, although they have an area as a property. The rationale for this choice can be found in Argent *et al*. (2005). A standard implementation of a FU is a series of three models: a rainfall-runoff model, a constituent generation model, and a constituent filtering model. The generation and filtering models are usually a collection of models that deal independently with each water constituent.

Sub-catchments, FUs, constituent generation and filtering models are defined as interfaces or abstract classes (Figure 1.). The previous paragraph only detailed a possible implementation, albeit the standard one. The main advantage of this systematic reliance on software interfaces and abstract classes is that it is possible to replace these default models at different granularity with more custom models if required. A sub-catchment may then be modelled in a different way, e.g. as a fully grid-based model, without any modification to the rest of the modelling engine. Similarly, if a standard representation of a functional unit is not appropriate for the modeller, another concrete implementation may be used. This possibility has already been used to support the modelling of irrigation areas (Hornbuckle *et al*. 2005).

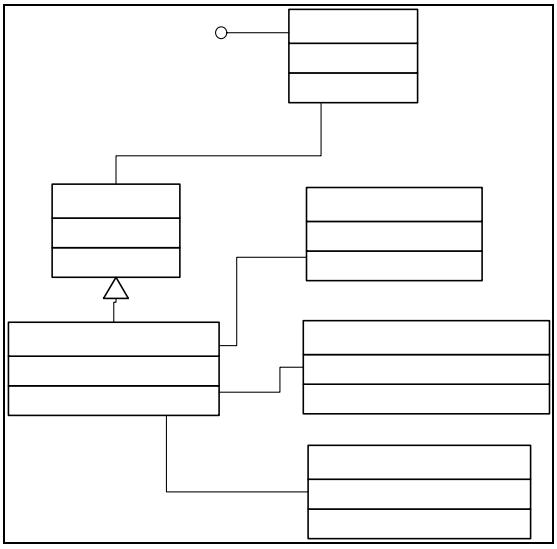

Figure 1. Sub-catchment structure in E2

3.2. Links and nodes

The river system is modelled using a structure of links connected via nodes. This is a common conceptual structure also used in e.g. IQQM and EMSS. Conceptually links and nodes are both network elements with one or more inflows and an outflow. The difference is that nodes are usually thought of as having no spatial extent. These network elements support the most demanding part of the modelling exercise from a mathematical point of view, and as a consequence it is also a challenging part of the framework in terms of software engineering.

From a software standpoint, links themselves are simple and have a small number of properties and methods. Their software architecture relies on flexibility by composition of objects rather than direct inheritance (Gamma *et al*. 1994). The task of propagating the characteristics of the flood wave is delegated to an instance of a `FlowRouting` class, which itself delegates the task of manipulating the constituents to an instance of an `InStreamProcessingModel`. Reservoirs are represented as links rather than nodes, the rationale

for this being that reservoirs have a significant spatial extent. Reservoirs can also arguably be seen as just another flow routing scheme, and they are thus another instance of the `FlowRouting` class.

Nodes are well suited to support the modelling of processes that occur at a given point in the system with little or no spatial extent. Nodes feature an expandable list of instances of a `NodeModel` class that can be used to model these processes. Node models are called sequentially in the order they are in the list, and modify the states of the node and any other objects they are related to. Current examples of node models are models forcing time series of flows or water quality constituents, water demand and extraction.

3.3. Mass balance and units

Work at a variety of temporal and spatial scales, with a variable list of constituents and a choice of different models raises some significant challenges. Notably, the handling of units and the related issue of mass balance may be taken for granted at an abstract level but is surprisingly difficult in practice from a software engineering perspective. Many readers will likely have come across lines of code with obscure unit conversion factors, and will be aware of how easily these lead to errors.

In order to limit the risk of inconsistencies, the E2 modelling engine represents physical quantities in S.I. base units (BIPM 1998). The component models used in E2 may have their parameters expressed in other units, but their output into the modelling engine must conform to the S.I. standard. The context-dependent conversion to other units is delegated to the presentation layer, and is facilitated by the unit handling framework provided by TIME.

Masses and fluxes of water and constituents, and their concentrations, are represented throughout the system via self-contained objects. Operations like the addition of two quantities or fluxes of water solutes, water extraction for irrigation, or the conversions between concentrations to equivalent masses or fluxes can thus be encapsulated in methods, thus greatly reducing the risk of error.

3.4. System run

The modelling engine adopts a hierarchical approach to running the system. Prior to the start of the simulation, model state variables can be set to initial values. In other catchment modelling systems initial values are often expressed as additional parameters, adding unnecessary degrees of freedom when calibrating. The software system of E2 handling initial values relies instead on a variant of the Memento Pattern described by Gamma *et al*. (1994), and uses software reflection wherever possible to avoid writing custom code for every model. Prior to running every time step, the values of input time series are fed into the model, once again relying on reflection. The system is run in a hierarchical fashion, each model element passing the temporal characteristics (date and time step length) to its sub-models, leaving room for the sub-models to run at a finer temporal resolution if need be. The temporal characteristics of the run are derived from the input time series if unambiguous, or can be specified by the user.

4. USER INTERFACE

The approach of allowing for choice between alternate models or methods for performing a given modelling task poses some significant challenges in terms of user interface. A system with fixed algorithms, or a fixed workflow for building the catchment model, has the advantage of making it easier for a user interface designer to tailor the forms. In such systems the number and types of input data and parameters expected at every step are set and thus allow for designing easy to use, custom forms that reflects the predefined tasks.

E2 provides several points of extension for alternate methods of performing a given task, e.g. defining the sub-catchments and network (Figure 2). Importantly, it allows for the choice of a method that is adapted to the data availability. However, this makes it more difficult to design a user interface workflow that guides the user effectively.

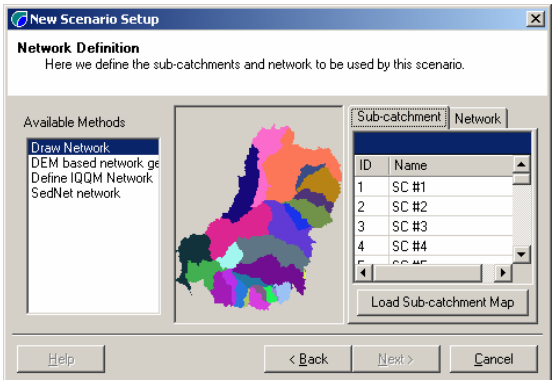

Figure 2. Network definition in a scenario setup

The overall approach for the E2 user interface is to have some broad overarching workflow that users must adhere to when creating a scenario. This is embodied by a scenario wizard stepping through

the broad categories of tasks required (definition of sub-catchments and network, then assignment of models, and finally assignment of parameters to these models). At many of the steps in the standard scenario wizard, the user is presented with a variety of options to complete the step. In the example shown in Figure 2, four alternate methods were detected for performing the task. These methods are primarily detected at run-time by exploring the executable and libraries of the application. E2 also uses a plug-ins manager component to load additional libraries (.NET assemblies), that are also explored for additional methods to perform a given task.

The underlying mechanism to find these methods is software reflection. The software types (synonym of classes, here) defined in the assembly are retrieved and inspected for relevant characteristics. For instance in the case of the network definition, the program would look for types that inherit from the class `UserControl`, and implement an interface `ICatchmentDefiner`. The level of sophistication in this detection of appropriate types can be enhanced using custom attributes, as explained by Rahman *et al*. (2004). An example is the detection of user interface controls for editing the parameters of instances of `NodeModel`, since the control can be given an attribute that specifies with which model it is working. The use of software reflection is a key technique for keeping the user interface and modelling engine decoupled, and for enabling the points of extension in E2.

5. PERSISTENCE

The persistence mechanism is based on a relatively simple but robust method which will be expanded. It has been designed to allow for a relational persistence tool, such as NHibernate (Koshcheyev 2005), to be used in the future. At the moment an E2 project file is a zip file with an "e2proj" extension. It contains a Microsoft Access database file and a number of data files which may include rasters, time series and shape files. When a project is loaded all of the scenarios and data related with each scenario are loaded. When saving a project, a new database is created and along with the necessary data files they are saved to a temporary location and a zip file of the folder is created. The new project file is then copied over the previous project file.

The following section focuses on the E2 specific challenges involved with the implementation of a persistence mechanism.

5.1. Unknown Models

Allowing choice between alternate models or methods for performing a given modelling task is not a significant problem for persistence; however the fact that models can have any object as a parameter complicates matters. The saving mechanism needs to be able to persist previously "unknown" objects. This is the case for models such as the storage models that have a `StoreGeometry` object as a parameter.

This is enabled by setting up an interface that a model developer can implement in order to save previously unknown object types. The interface is called an `ITypePersistor` (Figure 3).

```
public interface ITypePersistor
{
    string createStatement { get; }
    void saveObject( RelationalTool r, object o, int ID );
    object loadObject( RelationalTool r,int ID );
}
```

Figure 3. The ITypePersistor interface.

The `createStatement` function in Figure 3 enables a developer to create their own table structure within the database to save an object. The `saveObject` function is given an object and an ID that will be associated with that object.

The `RelationalTool` object (Figure 3) has a number of functions for executing SQL statements, saving references to other objects in the system, and saving other objects. This includes the core elements such as models or constituents and any object that has an `ITypePersistor` associated with it. The load object function simply has to load the object that was associated with the ID given.

5.2. Extending core elements of the framework

It is possible to extend the core elements of the framework such as in the case of the Irrigation FU (Hornbuckle *et al*. 2005). This irrigation FU inherits from the abstract parent class `FunctionalUnit` and implements different functionality to the standard FU described earlier. Thus a similar mechanism to the method used to handle unknown objects was needed. This is done with an interface `IInheritedTypePersistor`. The definition of this is slightly different to the `ITypePersistor`. When loading and saving the `IInheritedTypePersistor` it only needs to save anything extra that the class needs. The load object function is given an instance of the

object with all the elements of the base class already set.

When implementing either a `ITypePersistor` or a `IInheritedTypePersistor` a `WorksWith` custom attribute tag needs to be included on the class indicating which classes it can persist.

6. CALIBRATION

Calibrating a model, or a suite of models, encompassing multiple scales and processes is a difficult task, both conceptually and practically. The main conceptual difficulties are the obvious risk of non-uniqueness of parameter sets (Beven *et al*. 2001) due here to a large number of parameters, the high non-linearity of many catchment processes, and the usual sparsity and uncertainty of the data to calibrate against. A practical difficulty is the likely requirements in terms of computing power due to the large number of parameters, but by far the main difficulty is that it is hard to design an easy-to-use calibration tool for complex systems.

While software engineering *per se* can do little to address the sparsity of data, it is more feasible to reduce the tedious part of a calibration exercise and to turn a problem with too many degrees of freedom into one of a more manageable complexity. It was acknowledged from the start of the project that one key feature of E2 would be to facilitate the calibration process.

E2 currently features a calibration methodology allowing for the creation at run time of custom sets of tied parameters. This allows for representing the variability of catchment characteristics, e.g. that a conceptual soil moisture store is larger over a forested area than pasture, while exposing only one parameter in an optimisation process, without having to recode a soil moisture model with new hard-coded algorithms. The groups of tied parameters must contain only parameters that are dealing with a given process, e.g. sub-catchment outflow or in-stream processing of total suspended sediment. The software mechanism building the group of parameters relies on the *Builder* and *Factory* patterns, as shown in figure 4, in order to filter the candidate model parameters that can be grouped. The values of model parameters included in this group are tied by a factor to a "master value" held in a `GroupedItemsCharacteristics`, thus removing a degree of freedom for every model parameter grouped. The idea of grouping or tying parameters has been exploited previously in other model optimisation tools like PEST (Doherty 2002). Another key feature of the calibration tool is the ability to crop a sub-set of the full network to calibrate and input time series as required in the headwater elements of the sub-set, thus drastically reducing the model runtime by up to several orders of magnitude for large catchments.

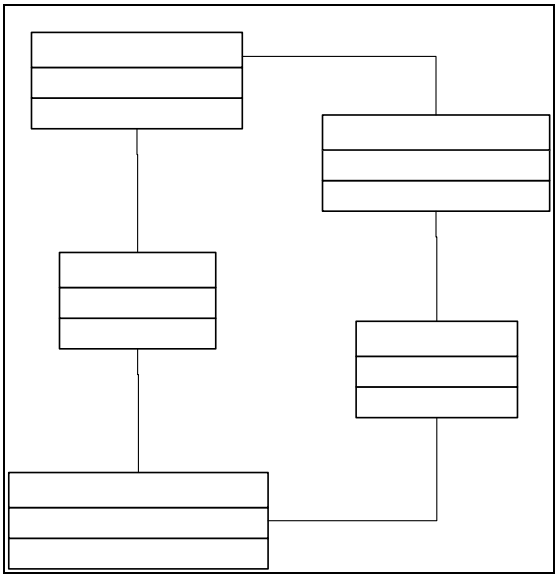

Figure 4. Parameter group builder

7. CONCLUSIONS

A flexible catchment modelling software framework, named E2, is under ongoing development within the Catchment Modelling Toolkit (www.toolkit.net.au). It blends the unique capabilities of several water quantity and quality catchment models currently in use in Australia and elsewhere, and tries to overcome some of their limitations. It has been designed from the ground up to cater not only for the physical modelling of unregulated river systems but also for regulated systems, ecological response models and possibly for economical modelling. Using a plug-in approach, modellers can tailor their catchment model not only by choosing amongst alternate models for various sub-systems, but potentially customise the user interface for a non-expert audience if required, though the latter remains to be done in practice. The architecture of E2 relies heavily on object oriented design patterns and software introspection to achieve this flexibility and manage the complexity stemming from this integration exercise. Upcoming work on E2 includes improvements of the modelling capabilities of regulated systems, water temperature, the addition of ecological response models, and enhancements to the calibration capabilities.

8. REFERENCES

Argent, R. M., R. B. Grayson, G.D. Podger, J.M. Rahman, S. Seaton and J-M. Perraud (2005), E2 - A flexible framework for catchment modelling, *Proceedings of MODSIM 2005*.

BIPM, Bureau International des Poids et Mesures (1998), The international system of units (SI), 7th edition, http://www.bipm.org/en/publications/brochure , Last Accessed August 9, 2005.

Beven, K.J. and J. Freer (2001), Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the GLUE methodology, *Journal of Hydrology*, 249, 11-29.

Cuddy, S. (2003), EMSS User Guide, pp. 105. http://www.toolkit.net.au, Last Accessed August 9, 2005.

Doherty, J. (2002), PEST, Model-independent parameter estimation, fourth edition (2002), User manual, Watermark Numerical Computing, pp. 279

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

Hornbuckle, J.W., E.W. Christen, G. Podger, R. White, S. Seaton, J-M. Perraud, J.M. Rahman (2005) Predicting irrigation return flows to river systems: conceptualisation and model development of an irrigation area return flow model, *Proceedings of MODSIM 2005*.

Koshcheyev, S. (2005), NHibernate www.nhibernate.org, Last Accessed August 9, 2005.

Podger, G.D. (2004) IQQM Reference manual, Software version 7.32, Department of Infrastructure, Planning and Natural Resources, 04/11/2004, pp. 102

Rahman, J.M., S.P. Seaton, and S.M. Cuddy (2004), Making frameworks more useable: using model introspection and metadata to develop model processing tool, *Environmental Modelling and Software*, 19, March, 2004, pp. 275-284.

Rahman, J.M., S.P. Seaton, J-M. Perraud, H. Hotham, D.I. Verrelli and J.R. Coleman, (2003), It's TIME for a new environmental modelling framework, *Proceedings of MODSIM 2003*, (4), 1727-1732.