BUILDING A DATA MODEL FOR A WATER YIELD ESTIMATION SOFTWARE TOOLSET

Jean-Michel Perraud[1], Biao Wang[1] & Jai Vaze[1]

[1]Water for a Healthy Country Flagship, CSIRO Land and Water, Canberra ACT 2601, AUSTRALIA

E-mail: jean-michel.perraud@csiro.au

Abstract

The Catchment Water Yield Estimation Toolset (CWYET) is a software toolset for estimating water yield over up to hundreds of catchments, featuring capabilities for calibration, catchment cross-verification, ensembles of models and scenario modelling such as impact of climate change. These uses require building alternate model configurations by alternating measured or hypothetical climate inputs, model parameterization, objective functions, state initialization, etc. Figuratively, this is not different from the assembly of Lego™ blocks. This information needs to be managed for the traceability and reproducibility of scientific experiments. We present in this paper an entity-relationship data model (ERM), i.e. a conceptual representation of the data, in the broad sense, associated with these catchment model "Lego" building blocks. While this ERM information modelling technique is widespread in the business systems design, it seems to be uncommon in scientific software design. CWYET has multiple requirements, sometimes conflicting, for this ERM. One central requirement is to build and execute catchment simulation models from a software scientific workflow system, the Hydrologists' Workbench. To capture the design of the data model, we use the Microsoft Entity Framework, using the so-called "model first" approach. In order to maintain some capacity to evolve and adapt the data model for future need, we use code generation to reduce the coding tedium to a minimum. We present an assessment of the benefits and smaller inconvenients of this approach, notably compared to the more ad hoc approach to the management of model configuration that existed prior to this endeavor.

Introduction

The Catchment Water Yield Estimation Tools (CWYET) is a modelling framework for estimating daily catchment water yield and runoff characteristics in regulated and unregulated catchments (Vaze *et al*. 2011a and Vaze *et al*. 2011b). One background motivation for this toolset is a need to develop a modelling framework which can be used by different water management and research agencies across Australia that allows them to undertake the modelling in an objective, consistent and reproducible manner.

CWYET has been applied in research and decision support projects, some with a substantial requirement for reproducibility and an audit trail. This can be a challenge in a context where the toolset will still need rapid evolution for the research purpose. The computational load required by the tool, due to the combinatorial effect of alternate inputs, catchment models, calibration techniques, etc. often requires distributed computation on a computational cluster. These contexts have some bearing on how the data and model configurations are structured.

For a variety of logistical and historical reasons, CWYET has tracked the definition of modelling tasks and results in practice by relying on storing information in XML files and comma-separated value files (CSV). These formats are a compromise insofar as they are both machine-readable and to an extent human-readable. Over the years, while goals have been served adequately, several shortcomings are apparent. The reliance on a file system (and folder and file names as identifier) can be an issue depending on the organizational context. Storage infrastructure is upgraded, and these changes gradually compromise the provenance trail of the modelling results. People move on to other roles, and are at best less available to answer queries on past work. The XML based storage of model configuration information proved cumbersome to evolve for new modelling endeavours. Most importantly, it is all too easy to overwrite these files, intentionally or not.

In this paper we step back from this existing infrastructure for the management of models and data associated with CWYET. We summarily document and analyze the fundamental needs in this scope. Note that this is a subset of CWYET needs, and in particular we are not concerned with user interfaces in this paper. We propose a software solution in the form of a data layer using current or recent technologies, and more in line with the state of the art in the business world.

Terminology

The word 'model' in this paper can refer to two very different things. One is a hydrologic model, the other is an entity-relationship model, a term from the field of software engineering. Where ambiguous, these will respectively be referred to as “catchment model” for the former, and “data model” or ERM for the latter.

Needs

CWYET serves both the needs of research projects and more applied modelling exercises feeding in the management of water resources. It is used to estimate water yield over up to hundreds of catchments, featuring capabilities for calibration, catchment cross-verification, ensembles of models, and scenario modelling such as impact of climate changes.

For both research and decision support, but in particular for the latter, reproducibility and transparency is essential. These are arguably obvious, paramount needs to most readers, but their adequate implementation is eminently difficult.

CWYET model runs often require computational clusters to run, and sometimes concurrent read access to data. The scientific data (distinguished from model configuration data) is increasingly found in a netCDF format (Rew et. al 2006), although the exact data schemes can vary depending on the modelling need (indeed CWYET can contribute to their definition).

We also aim to construct and manage CWYET models and data from the Hydrologists' Workbench (HWB) (Cuddy & Fitch, 2010). HWB is a modelling workbench building on the Trident scientific workflow software (Barga *et al.* 2008). The work of Perraud *et al.* (2010) on the appropriate granularity of activities in a calibration workflow motivates this desire to make CWYET modelling capabilities available through HWB.

The majority of the code implementing CWYET is based on .NET, as is HWB. While we want a data model that is decoupled from this technology, we do want .NET software to access this data layer easily. In particular there are toolsets for object-relational mapping and entity-relationship modelling, such as Nhibernate, which are in use with .NET code.

Analysis

CWYET is used at regional to continental scales. The main conceptual modelling unit is a catchment, which has one or more of “cells” for semi-distributed modelling of runoff within it. There is no built-in assumption that these cells are gridded, though this is usually the case. Each cell is modeled by a lumped conceptual rainfall runoff model ().

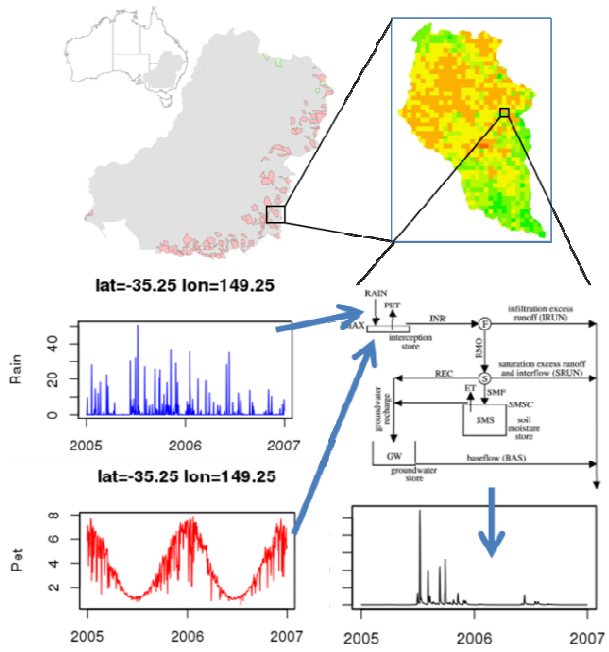

Figure 1: CWYET with a gridded model structure

It is important to note that the structure in Figure 1 is but one of the possible combination. Using a cliché, one can think of the definition of a variety of catchment models as a parallel to building constructs with Lego™ blocks of varying colors. A list of the main parts of the CWYET model definitions, and examples for each of them, follows:

The structure of the catchment model, e.g. lumped or semi-distributed, gridded; what model structure represent the water fluxes (Sacramento, GR4J, ...)

The mapping of input climate time series to the input variables of the specific structure of the catchment model. The source of the data may consist of a netCDF file with a 3D schema lat-lon-time, or a series of CSV files with file name conventions derived from the series geolocation.

The parameterisation to apply to the model. A set of model parameters may be applied identically to all grid cells, or to a subset thereof, for instance when transferring parameter sets from calibrated catchments to individual grid cells.

The initialization of the model state variables prior to the first time step of the simulation, typically setting the 'water buckets' of lumped conceptual rainfall-runoff models.

The specification of the state variables of the model that are recorded as output time series, e.g. “record runoff from each individual grid cell, and the catchment baseflow and runoff variables”

The specification of the statistics applied to the output time series, e.g. “get the mean annual runoff depth for each grid cell, and the Nash-Sutcliffe efficiency of the daily streamflow for the whole catchment”

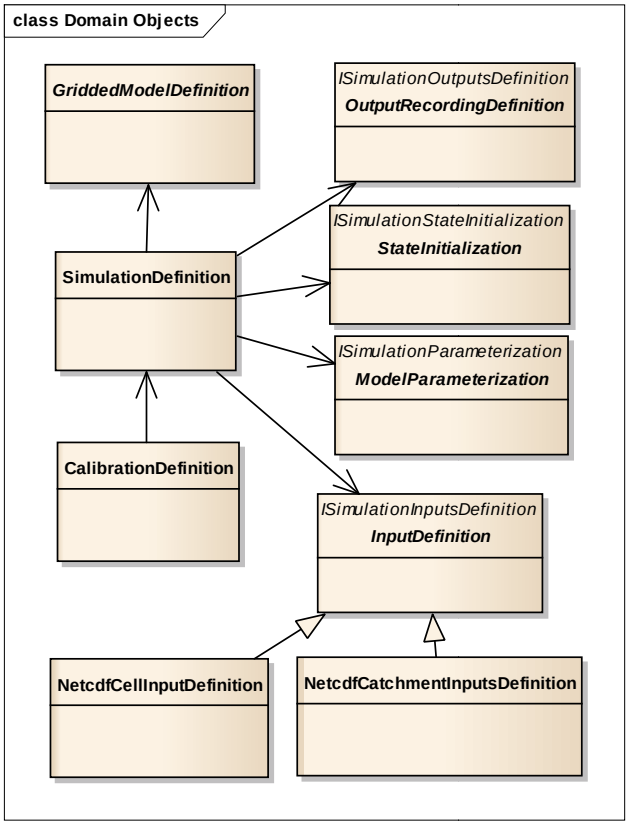

Figure 2 Summary core data model for CWYET

The list above covers *only* the core definition of a catchment model, but CWYET comprises other aspects that need a similar building block approach. Each catchment model may be used for several simulations, covering different time spans, producing different output statistics. The definition of catchment model calibration needs similar flexibility for the specification of the objectives, and the calibration strategy used (split-sampling, optimizing algorithms).

From this analysis one can derive a set of entities capturing each aspect of the modelling task definitions. Figure 2 shows the main core entities for CWYET, and their relationship. A calibration definition references a simulation definition which in turn references the components definition of the structure of the model. The simulation definition is a composition of *abstract* entities covering the aspects previously listed. Defining abstract entities permits the flexibility to compose water yield modelling scenarios "Lego-style". For instance, Figure 2 includes two concrete types of input definition. Both are using netCDF files as a data sources, but one extracts data for a gridded cell based model, while the other extracts data intended for the lumped modelling of catchment.

The definitions above may appear self-evident to many readers, especially to those versed in environmental modelling software framework. Despite this well-shared conceptualization, software implementation is, in our experience, often at best very partially fulfilling these needs, especially when it comes to allowing flexibility to defining each of these aspects and in particular *persisting them sustainably*. This paper puts an equal emphasis on explaining the implementation process, object of the next section, as on the design and analysis.

Implementation

The entities of the CWYET model definition must be accessible to components and services in both the "business" layer (i.e. the modelling engine) and the data layer (i.e. the database system storing them). This is a common situation for software applications (Meier *et al.*, 2008). These two objectives often require differing *views* on these entities, sometime leading to a mismatch in the requirements for these entities. Object-relational mapping (ORM) techniques and toolsets have evolved to address this mismatch. This section summarizes the implementation of the CWYET model definition entities using Microsoft Entity Framework (EF) (Lerman, 2009), chosen chiefly for its platform, .NET. It enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write.

There are three broad approaches to building these entities: data first (there is a pre-existing database), model first (design entity graphically and generate code and database), and code first (derive the database by analysing the code).

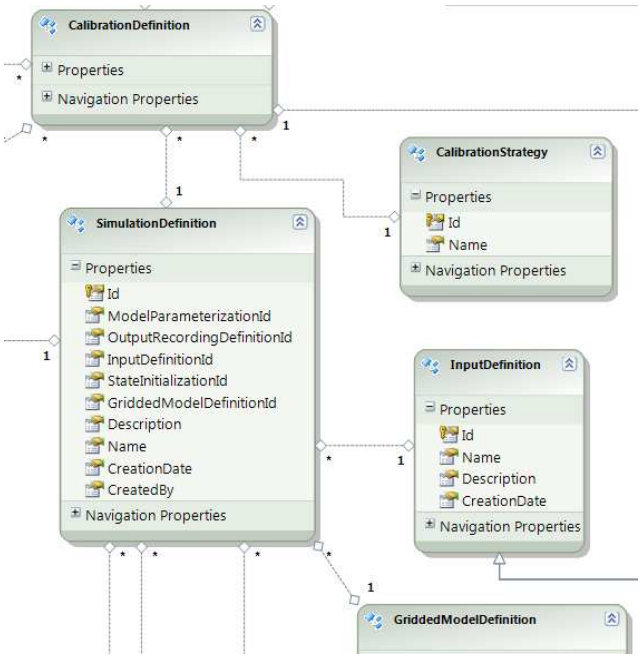

Figure 3: Graphical EMD builder in Visual Studio

While CWYET has a legacy data layer in its architecture, we choose to re-engineer it with the minimum of legacy constraints. We implement the CWYET entities with the model-first approach. Of particular interest in EF is the availability of a graphical data modelling tool (Figure 3)

which permits a view clearer than by starting to directly code the entities, and without the explicit need for the particulars of the database storage logic.

The process of implementation of the CWYET data model in a database is shown in Figure 4. From the data model definition in the EMD, and for each entity defined in that model, two things are generated. First, the Microsoft toolset generates the SQL script that can subsequently create a database complying with this data model. C# code with class definitions is also generated, one for each entity defined as well. Two files are created: one with only data, and overwritten every time, and the second created only once, to host the *behavior* of the class. The behavior consists of code defining what the objects can do, such as constructing an executable catchment model from the data in this (resp. these) entity (resp. entities) that define(s) this catchment model. Adding the behavior to the classes can only be a manual process here. Lastly, code for *repositories* is created. Basically, the Repository pattern just means putting a façade over your persistence system so that you can shield the rest of your application code from having to know how persistence works (Miller, 2009). This also promotes a consistent way to access, query and filter entities of a certain type. Note that the only manual part of the process is the definition of the behavior of the entities; as much as possible the "plumbing" of the data layer is automatically generated. The code for the entities is generated using so-called T4 templates (Text Template Transformation Toolkit, see Allen (2010) for instance), available in Visual Studio 2010 yet unknown to most developers.

This workflow, partly standard to the EF toolset, partly customized, stems from the desire to follow the practices presented in Meier *et al*. (2008) and Miller (2009). It is important to note that the code generated for entities is completely independent from the "persistence" layer, i.e. from an SQL Server database in this instance. This is consistent with the best practice advocated as having "persistence-agnostic business objects". One advantage of this pattern is that should the type of database change, having such a layering limits the risks of having to change the business logic.

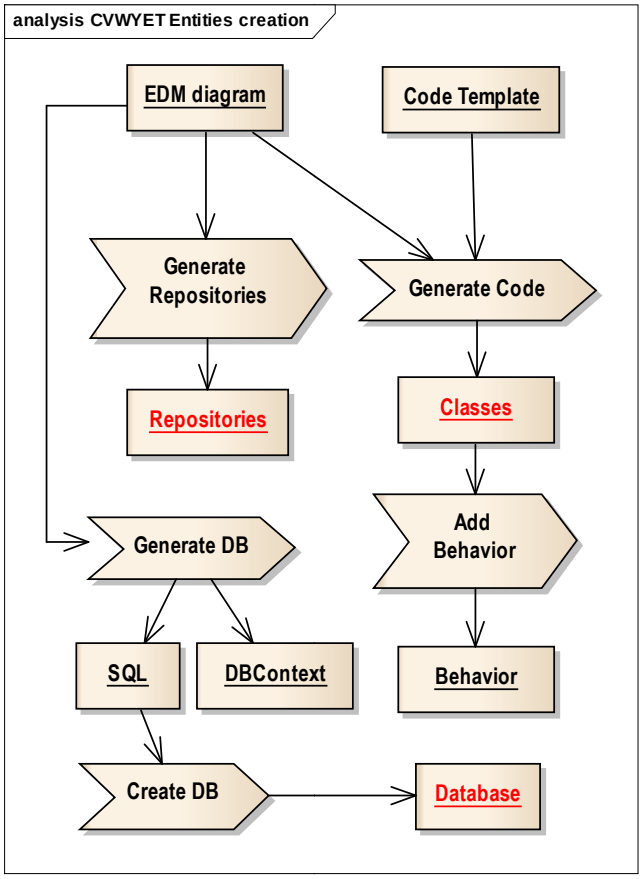

Figure 4: workflow of the creation of the code and data store.

Sample applications

We conclude this section by demonstrating two modelling applications using the CWYET data layer. Figure 5 shows a workflow in HWB constructing CWYET catchment model definitions. The first workflow activity retrieves a "SimulationDefinition" entity from the database, using keywords or a unique identifier. Combined with the second activity, we obtain a gridded model structure ready to execute to get output time series or to pass to e.g. a calibration workflow.

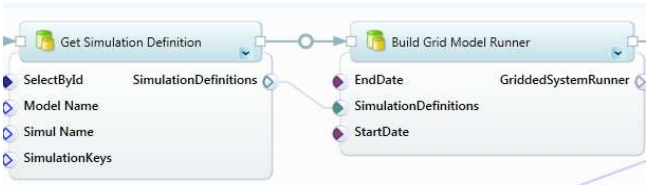

Figure 5: CWYET model building activities in a scientific workflow (HWB)

For modelers needing fuller access to the CWYET configurations, another example using the CWYET data layer is in Figure 6. This shows sample code in the Python language (http://ironpython.net and http://python.org). This code would be useful in case we needed, for the sake of illustration, to correct some of the input time series, for

instance if the gauged input rainfall time series had been further quality assured since.

```
# Update rainfall input time series for two catchments
catIds = ['701009','704193']
rh = RepositoryHelper()
u = rh.GetUnitOfWork()
simDefRepo = rh.GetSimulationDefinitionRepository(u)
# Use a LINQ expression to find all entities to correct
simDefinitions = simDefRepo.Find(lambda x: x.Name in catIds)
for s in simDefinitions: updateRainfallInputDefinitions(s)
u.Save()
rh.Dispose()
```

Figure 6 Querying and modifying model simulation definitions

Two things need to be emphasized. First, the code is concise, and the technical complexities of the SQL database storage are almost invisible (only the call to `u.Save()` is a need arising from this, and is rather intuitive). Second, we demonstrate here the use of LINQ to find a subset of CWYET entities. LINQ is powerful in this example, mostly for reasons that unfortunately cannot be fully explained in a paragraph. Suffice to say that a lot happens behind the scene to enable a query in one short line of code.

Discussion

While the process is "model-first", removing the constraints arising from legacy, some choices are naturally guided by previous experience. The product Source IMS (Welsh *et al.*, 2012) uses a sophisticated system for persistence based on the NHibernate framework. Issues of performance and backward compatibility appear in a context where an application starts from a view centered on hydrologic modelling, as opposed to software and data centric (personal communication). This is unfortunately rather hard to avoid in the context of environmental modelling. Some of the software constructs that are natural to environmental model software developers may not be straightforward to represent in a relational database.

Based on these observations, the model-first approach was chosen here for CWYET as a compromise between the persistence mechanism and the conceptual entities in the environmental modelling constructs. The data model designer (Figure 4) lets developers create relatively simple types of properties for the entities, yet reflects the software concepts of type inheritance. EF can automatically generate the database schema in SQL, but the corresponding software data entities may appear crude to a coder. Conversely, class abstraction and inheritance is available as expected from developers familiar with object oriented software, and EF is taking care of the representation of inheritance in the database. This latter capability is essential to flexibly enable alternate model configurations (i.e. alternate Lego™ blocks, in our childhood analog). Importantly for the authors, rather little needed to be learned about relational databases and associated database management software tools.

The manipulation of CWYET catchment model definitions using EF is currently superseding the legacy use of XML. It is proving easier to evolve capabilities and avoid duplication of identical configuration items throughout many XML files. The capabilities for loading, querying and filtering entities built with EF are much superior to that put on top of a file system. In theory, the script in Figure 6 could be achieved with a storage layer consisting on XML files and folder names conventions. It is more practical to use state of art tools and practices with an SQL database, where these capabilities are available with much less additional effort.

The process of designing and implementing this data layer required thinking of an environmental modelling software toolset in terms of its data at least as much as its runtime behavior. Many problems of sustainability of catchment modelling software toolsets arise arguably because of an overly model-centric view to the detriment of a data-centric one. With this in mind, it is worth noting that the learning material of EF is often using business examples such as customer order databases, and it is an interesting exercise to translate this to the context of environmental modelling.

While the benefits so far of this new data layer are compelling, there are of course some difficulties yet to overcome and questions about the scope of applicability.

There are two main issues to overcome. First, the database may require performance optimization, though this is probably partly a question of acquiring know-how. Second, database migration capabilities will be needed to evolve the data model to address new features. Fortunately both aspects are also currently addressed by the EF team.

The scope of applicability of this data layer is a more interesting question. Overusing EF, it can become all encompassing, dogmatic and so restrictive that it ends up appearing obtuse. The definition of calibration process is an interesting case: should it be stored like any other model configuration data, or is this by nature more a workflow to persist? HWB seems obviously much more adapted to capture its definition, and we should avoid building a separate system for CWYET specific workflow management. We envisage that subsequent research and development is necessary to couple the CWYET data model with that of HWB and Trident when warranted, and notably leverage the provenance system of Trident.

Conclusion

The reengineering of the CWYET data model, using Entity Framework and Entity Data Modelling approach with a model-first technique, has brought it more in line with the state of the art in software application design and

implementation. The toolset is now in a better position for inclusion as modelling capabilities in HWB, within the scientific workflow tool Trident. Managing ensembles of simulations and modelling scenarios is easier than with the previous data storage system, and the relational database technology is a more sustainable solution. A central motivation of this reengineering was to better address the challenge of tracking provenance in modelling and simulation with CWYET. Further research and development will examine how to couple the CWYET data model with the HWB workflow definitions and provenance tracking in Trident.

Acknowledgement

The Catchment Water Yield Estimation Tool (CWYET) framework is developed as part of eWater CRC's Source Rivers and Source Catchments projects. This work was carried out in the CSIRO Water for Healthy Country National Research Flagship with support from National Water Commission and Commonwealth Department of the Environment, Water, Heritage and the Arts (DEWHA).

References

Allen, K.S., (2010). Text Template Transformation Toolkit and ASP.NET MVC, MSDN Magazine 2010 January, http://msdn.microsoft.com/en-us/magazine/ee291528.aspx

Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., Simmhan, Y. (2008), The Trident Scientific Workflow Workbench, *IEEE Fourth International Conference on eScience*, pp.317-318, 7-12 Dec. 2008, doi: 10.1109/eScience.2008.126

Cuddy, S. & Fitch, P. "Hydrologists Workbench–a hydrological domain workflow toolkit", *International Congress on Environmental Modelling and Software*, July 5 - 8 2010, Ottawa, Ontario, Canada

Lerman J., (2009), *Programming Entity Framework*, Second Edition, O'Reilly Media, p. 918, ISBN-13: 978-0-596-80726-9

Meier J., Homer A., Hill D., Taylor J., Bansode P., Wall L., Boucher R. & Bogawata A., *Application architecture guide 2.0: designing applications on the .NET platform*, Microsoft Corporation, 2008.

Miller, J., (2009), Design Patterns for Data Persistence, *MSDN Magazine*, April 2009, http://msdn.microsoft.com/en-us/magazine/dd569757.aspx#id0400058

Perraud, J.-M., Bai, Q., & Hehir, D. (2010), On the appropriate granularity of activities in a scientific workflow applied to an optimization problem, *International Congress on Environmental Modelling and Software*, July 5 - 8 2010, Ottawa, Ontario, Canada, http://www.iemss.org/iemss2010/proceedings.html

Rew, R. K., Hartnett E. J. & Caron J. (2006), NetCDF-4: Software Implementing an Enhanced Data Model for the Geosciences, *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, AMS 2006.

Vaze, J., Chiew, F. H. S., Perraud, JM., Viney, N., Post, D. A., Teng, J., Wang, B., Lerat, J., Goswami, M., (2011a). Rainfall-runoff modelling across southeast Australia: datasets, models and results. *Australian Journal of Water Resources*, Vol 14, No 2, pp. 101-116

Vaze J, Perraud J, Teng J, Chiew F, Wang B, Yang Z. (2011b). Catchment Water Yield Estimation Tools framework (CWYET). *34th IAHR World Congress 2011 - Balance and Uncertainty Water in a Changing World*. Brisbane, Australia.

Welsh, W., Vaze J., Dutta D., Rassam D., Rahman J.M., Jolly I., Wallbrink P.J., Podger G., Bethune M., Hardy M.J., Teng J. & Lerat J. (2012), An integrated modelling framework for regulated river systems, *Environmental Modelling & Software*. in press.