

# Land-Water Interoperable Models

## Project Summary

*Prepared for Our Land and Water*

*June 2020*



Prepared by:

Sandy Elliott<sup>1</sup>, Tom Evans<sup>1</sup>, Rogerio Cichota<sup>2</sup>, Utkur Djanibekov<sup>3</sup>  
Alex Herzig<sup>3</sup>, Bethanna Jackson<sup>4</sup>, Daniel Lagrava Sandoval<sup>1</sup>  
Stephen McDonald<sup>5</sup>, Juan Monge<sup>6</sup>, Annette Semadeni-Davies<sup>1</sup>  
Mike Taves<sup>7</sup>, Christophe Thiange<sup>8</sup>, Ronaldo Vibart<sup>5</sup>, Steve Wakelin<sup>9</sup>, Sharleen Yalden<sup>1</sup>, Jing Yang<sup>1</sup>,  
Harry Yoswara<sup>5</sup>

<sup>1</sup> NIWA

<sup>2</sup> Plant and Food Research

<sup>3</sup> Manaaki Whenua – Landcare Research New Zealand

<sup>4</sup> Victoria University of Wellington

<sup>5</sup> AgResearch

<sup>6</sup> Market Economics

<sup>8</sup> DairyNZ

<sup>7</sup> GNS Science

<sup>9</sup> Scion




For any information regarding this report please contact:

Sandy Elliott  
Principal Scientist  
Catchment Processes  
+64-7-859 1839  
National Institute of Water & Atmospheric Research Ltd  
PO Box 11115  
Hamilton 3251  
Phone +64 7 856 7026

NIWA CLIENT REPORT No: 2020182HN

Report date: June 2020

NIWA Project: AGR19202

Quality Assurance Statement		
	Reviewed by:	Neale Hudson
	Formatting checked by:	Rachel Wright
	Approved for release by:	Scott Larned

© All rights reserved. This publication may not be reproduced or copied in any form without the permission of the copyright owner(s). Such permission is only to be given in accordance with the terms of the client's contract with NIWA. This copyright extends to all forms of copying and any storage of material in any kind of information retrieval system.

Whilst NIWA has used all reasonable endeavours to ensure that the information contained in this document is accurate, NIWA does not give any express or implied warranty as to the completeness of the information contained herein, or that it will be suitable for any purpose(s) other than those specifically contemplated during the Project or agreed by NIWA and the Client.

# Contents

- Executive summary ..... 6**
  
- 1 Introduction ..... 10**
  
- 2 Overview of the modelling approach ..... 11**
  - 2.1 Model integration software ..... 11
  - 2.2 Model components ..... 11
  - 2.3 Standards and linkage ..... 12
  - 2.4 Introduction to the Aparima case study ..... 14
  
- 3 Model standards and approaches adopted ..... 16**
  - 3.1 BMI as the standard for linking ..... 16
  - 3.2 Naming conventions ..... 17
  - 3.3 File formats ..... 20
  - 3.4 File-based coupling approach ..... 22
  
- 4 Model components ..... 24**
  - 4.1 Web service for retrieval of pre-computed stream flows ..... 24
  - 4.2 Nitrogen loss lookup ..... 25
  - 4.3 Overseer nitrogen loss ..... 27
  - 4.4 Steady state nitrogen routing in streams ..... 29
  - 4.5 Steady state groundwater routing ..... 30
  - 4.6 Soil moisture accounting/rainfall-runoff model (subcomponent of LUCI) ..... 32
  - 4.7 Dynamic groundwater flow model ..... 33
  - 4.8 Dynamic stream flow routing ..... 35
  - 4.9 APSIM dynamic nitrogen source simulation ..... 36
  - 4.10 Dynamic groundwater nitrogen routing ..... 39
  - 4.11 Dynamic stream contaminant routing ..... 41
  - 4.12 Economics and optimisation ..... 43
  
- 5 Model assemblies ..... 46**
  - 5.1 Linkage diagrams ..... 46
  - 5.2 Linkage of components ..... 48

<b>6</b>	<b>Incorporation of models into the Delta Shell environment .....</b>	<b>49</b>
6.1	Setting up and running models in the interactive environment .....	49
6.2	Visualisation of results.....	49
<b>7</b>	<b>Model sharing and documentation.....</b>	<b>53</b>
<b>8</b>	<b>Intellectual property, governance and co-ordination.....</b>	<b>54</b>
8.1	Approaches used in the project.....	54
8.2	Some future governance and management needs .....	55
<b>9</b>	<b>Recommended future technical approach to interoperability.....</b>	<b>56</b>
<b>10</b>	<b>Summary of key results and findings .....</b>	<b>58</b>
<b>11</b>	<b>Acknowledgements .....</b>	<b>60</b>
<b>12</b>	<b>References.....</b>	<b>61</b>

#### Tables

Table 4-1:	BMI interfacing functions for the flow routing code.	35
Table 4-2:	Summary of contaminant routing model operation via Initialize() and Update() functions.	42

#### Figures

Figure 2-1:	Schematic of components coupled in Delta Shell.	12
Figure 2-2:	Aparima case study location.	14
Figure 2-3:	Schematic summarising the ecotope generation process.	15
Figure 3-1:	A representative model result, presented in SQLite Browser.	21
Figure 3-2:	Example hydrograph from the LUCI model.	22
Figure 4-1:	Sequence diagram of interaction between a master controller and target point model component.	26
Figure 4-2:	Schematic of the as-designed data flow for the nitrogen leaching loss model.	28
Figure 4-3:	Map showing boundaries of Aparima catchment, groundwater model and QMAP geology (Turnbull and Allibone 2003).	32
Figure 4-4:	Schematic showing how data were exchanged between APSIM and the Interoperable Model framework using several bespoke tools.	37
Figure 4-5:	Schematic showing how data were exchanged between APSIM and the Interoperable Model framework using coordinated by the APSIMHandler.	38
Figure 4-6:	Example instruction file used by the APSIM Handler to run a scenario involving several cropping rotations to determine fertiliser management options.	39
Figure 4-7:	Class structure of dynamic stream contaminant routing model.	42

Figure 5-1:	Example of a prototype 'wiring diagram' depicting model components and their interactions for the static nitrogen model.	47
Figure 6-1:	Example of prototype of data visualisation of flow time-series retrieved through a web service and displayed within Delta Shell.	50
Figure 6-2:	Example of map display of flow results for dynamic hydrologic model for the Aparima catchment.	51
Figure 6-3:	Example of display of flow time series from the dynamic hydrologic model for the Aparima catchment.	52

## Executive summary

### Project purpose and scope

This report is the final project report for Stage 2 of the “Interoperable Modelling Systems for Integrated Land and Water Management” programme in the Our Land and Water National Science Challenge (OLW). The programme addresses Theme 2 of the Challenge by providing modelling tools to support “Innovative and resilient land and water use”, and Theme 3 by building collaborative capacity within the modelling and model-user communities.

The programme aimed to develop an interoperable modelling system that is suitable for national use in integrated spatial assessment of environmental, production and economic implications of land use and land use change. The uses of the system will include assessment and accounting of productivity potential and water quality contaminant dynamics at farm and catchment scales.

Interoperability refers to an approach to modelling whereby individual model components are coupled in a flexible way including exchanging data between components, allowing for re-use and substitution of model components within an overarching framework. It is proposed that the availability of better, more trusted, and targeted modelling tools within an interoperable modelling framework will result in more effective use of integrated modelling for improved production and environmental management.

A staged approach has been developed to achieve these aims. Stage 1 of the programme established a proposal for work to be undertaken in Stages 2 and 3 of the programme, and was documented previously (Elliott et al. 2017). Stage 2, which is the subject of this report, focussed on implementing and demonstrating the initial set of models and data within the framework. Stage 3 (anticipated for tranche 2 of OLW) proposes to enrich the range of models in the framework and demonstrate and evaluate the use of the framework in multiple contexts, including linking to social and cultural attributes.

Work for this stage was conducted by a Technical Group from eight science organisations (NIWA, AgResearch, DairyNZ, GNS Science, Manaaki Whenua/Landcare Research, SCION, and Victoria University of Wellington) which incorporates specialists from a range of science providers, covering the required areas of technical expertise- including catchment hydrology, production systems, water quality, agro-economics and computer science.

### Key aspects of the modelling

Interoperable models require software to co-ordinate running of the various model components and managing data exchange. In this project, a system called Delta Shell was selected, following assessment of such frameworks in Stage 1. Delta Shell<sup>1</sup> was developed by Deltares for model coupling, user interaction, and visualising input and output datasets, and some of their core hydraulic and water quality models have been set up in Delta Shell. Delta Shell operates in the Microsoft Windows environment and is based on the Microsoft .Net C# language. It runs on a desktop computer. The user interface includes mapping and time-series display components 'out of the box'.

Following workshops at the start of the project, twelve model components of key interest to OLW were selected for implementation in Delta Shell. The components were organised into a set of static models (not time-stepping) and a set of dynamic models, covering key aspects of catchment hydrology and water quality. An economic, production, and optimisation component was also set up.

---

<sup>1</sup> <https://oss.deltares.nl/web/delta-shell>

These components demonstrate some key aspects that an interoperable system of models for land-water needs to cater for. Each component was set up using a standard model interface, the Basic Model Interface (BMI), and established for the case study site (Aparima catchment in Southland). The static components that were implemented are:

- Lookup table for nitrogen loss.
- Nitrogen transport through groundwater – MODFLOW.
- Nitrogen transport in streams – CLUES stream transport component.
- Production, economics and optimisation (LUMASS).
- Overseer nitrogen loss (partially completed).

The dynamic components that were implemented are:

- Web service to query pre-computed time series from the TopNet hydrologic model.
- Soil moisture accounting/rainfall-runoff model – from LUCI.
- Dynamic groundwater flow – MODFLOW.
- Dynamic stream flow routing (from TopNet).
- Dynamic nitrogen loss – APSIM soil-plant model.
- Dynamic groundwater nitrogen transport – MODFLOW.
- Stream nitrogen transport.

Delta Shell contains a set of interface specifications (objects and methods) that can be used for model components. Adopting those specifications would mean committing to a somewhat complex interface that is specific to Delta Shell, .NET, and the desktop computing environment. Instead of using Delta Shell's own specification, we decided to use an interface standard called BMI (the Basic Model Interface), which is open source and used by multiple frameworks. Deltares' own models are migrating more to using BMI, and Delta Shell has ways to run components that are set up with BMI. Hence BMI was considered by the technical group to be a preferable interface.

Two key sets of linked components ('assemblies') were set up to trial and demonstrate model coupling. They were a set of static models for nitrogen generation and transport, and a set of models for dynamic runoff generation and transport. In the dynamic model, the model components could exchange information at each time-step.

These components were linked using the DIMR (Deltares Integrated Model Runner) programme within Delta Shell. The programme specifies the components used in an assembly and their order of running, while allowing for dynamic models to exchange data each time-step (hence allowing feedback between components). An interoperable modelling system requires a mechanism to exchange data between model components. The BMI standard provides mechanisms for passing data through common memory, but we found that the DIMR implementation of BMI, and the BMI specification in general, had restrictions which were too limiting. To overcome this limitation, we adopted a file-based data exchange approach, using the set of standard files.

A set of standard file formats and naming conventions was adopted, to organise and facilitate data interchange. The file formats are well-recognised open standards, and many software programmes have methods to read in and output such data. A set of naming conventions based on Community

Surface Dynamics Modeling System (CSDMS) standard names was also adopted, although we customised the names to better represent the types of model quantities that are needed for land-water modelling.

We adopted an 'ecotope' approach to spatial representation, which involved conducting simulations for each spatial unit defined by the intersection of property boundary, land use, soil class, climate zone, irrigation (whether present) and slope class, which can be mapped onto a common grid basis. In hydrological modelling such areas are more commonly known as HRUs (Hydrological Response Units) – we have used the name ecotope to reflect an intention for application to both land and water modelling. The results from each ecotope then provided inputs for other models, such as inputs to each stream segment, after results were summarised at the subcatchment basis (based on the grid of ecotope locations in conjunction with subcatchment boundaries). This approach allows for future extension, for example, introducing other spatially-varying driving factors, allowing farm management variables to vary by property, or for farm-level predictions to take account of farm system behaviour. However, some limitations remain, such as the inability to represent the detailed placement of edge-of-field mitigation measures or individual paddock management.

We used a GitHub document repository to share model components, assemblies, data and related software. This system is publicly-available and widely used for software development projects. We did not use version management or revision facilities of Git in the current project, mainly because individuals or small groups were responsible for developing and maintaining their own components. However, we envisage more sophisticated use of Git in the future.

Model components and assemblies were tested in the Aparima catchment (1280 km<sup>2</sup>) upstream of Thornbury (Figure 2-2), which has multiple land uses but is dominated by pasture. The Aparima model components and model assemblies were not calibrated or applied to scenarios – such work was outside the scope of the current project. The main emphasis was to demonstrate how the model components and assemblies could be set up and run in a real catchment.

### Key results and recommendations

The project team succeeded in implementing a set of coupled models within an interoperable modelling framework, with application in a trial catchment. This meets the key technical requirements for this Stage of the programme.

Eleven model components covering water quantity, quality, production and economics were set up within a framework (Delta Shell) using established standards for model interfaces (BMI), variable naming (from CSDMS) and various standard file formats. We also attempted to implement Overseer, but could not fully achieve this, partly due to reliance on an organisation external to the project team.

Two sets of model components were successfully combined into assemblies to address a) steady state contaminant transport and b) dynamic coupled flow calculation. The catchment models resolved to the level of property, soil, climate class, and topographic class variation, using a common spatial framework of 'ecotopes'.

In addition, models for production, economics, and environmental losses were set up using the BMI in conjunction with the LUMASS optimisation engine.

As a separate exercise, we demonstrated how pre-computed flow time-series that are stored in an external server could be imported and displayed within Delta Shell using Sensor Observation Service (SOS) standards and a Delta Shell plug-in accessing data provided through that standard.



We decided that it was preferable to use BMI instead of Delta Shell's native interface, to allow greater flexibility and component re-use. However, we found that some key aspects of BMI were not fully implemented in the Deltares model runner/coupler DIMR. Some simplifications and work-arounds in our coupling approach were therefore required.

We found that file-based data exchange overcame limitations of BMI's interface for memory-based data exchange (for example, BMI does not allow for tabular data to be exchanged or implemented). The wide and multi-language availability of libraries for reading and manipulating the chosen standard file formats assisted with importing and exporting data from model components. File-based exchange could potentially become a problem with closely-coupled models that need to exchange data frequently; in those circumstances, memory-based exchange may need to be implemented.

The models were set up for a trial catchment, the Aparima, and assemblies were run successfully within the DeltaShell environment.

Model output was able to be displayed within the Delta Shell environment using the visualisation libraries and user interface native to Delta Shell, although some additional code was required to import model outputs into the DeltaShell environment.

All the model components, apart from Overseer, have been provided as free and open-source components available on a data repository.

Despite these successes, the project team recommends that an alternative coupling approach based on running model components be trialled. Such a system has several advantages over the Windows desktop approach of Delta Shell. The requirements for framework software, and developer expertise (to set up and orchestrate model assemblies) remain.

We used an ad-hoc 'wiring diagram' approach to defining linkages, data exchanges and timing. It would be desirable to develop more formalised methods in the future, although standard URL diagrams are not particularly suited to this purpose.

The project technical team developed good structured working relationships with ongoing collaborative and collegial interactions. It was also desirable to introduce a project manager to facilitate administrative task and performance monitoring. The governance group was increasingly less involved with the project over time, an aspect that should be improved in future work. We also identified that is important to use a software developer or group of developers with good computer science knowledge, alongside model specialists.

This project has met the key technical requirements of this stage of the programme. It is anticipated that a workshop will be held in the future to present key findings and recommendations, and to consider a pathway forward, including future funding, governance, and IP considerations.

# 1 Introduction

This report is the final project report for Stage 2 of the “Interoperable Modelling Systems for Integrated Land and Water Management” programme in the Our Land and Water National Science Challenge (OLW). The programme addresses Theme 2 of the Challenge by providing modelling tools to support “Innovative and resilient land and water use”, and Theme 3 by building collaborative capacity within the modelling and model-user communities.

The programme aimed to develop an interoperable modelling system that is suitable for national use in integrated spatial assessment of environmental, production and economic implications of land use and land use change. The uses of the system will include assessment and accounting of productivity potential and contaminant dynamics at farm and catchment scales. The latter include nitrogen, phosphorus, suspended sediment and microbial contaminants.

Interoperability refers to an approach to modelling whereby individual model components are coupled in a flexible way that permits exchange of data between components, allowing for re-use and substitution of model components within an overarching framework. It is proposed that the availability of better, more trusted, and targeted modelling tools within an interoperable modelling framework will result in more effective use of integrated modelling for improved production and environmental management.

A staged approach has been followed to achieve these aims. Stage 1 of the programme established a proposal for work to be undertaken in Stages 2 and 3 of the programme, and was documented in a report (Elliott et al. 2017). Stage 2, which is the subject of this report, focussed on implementing and demonstrating the initial set of models and data within the framework. Stage 3 (anticipated for tranche 2 of OLW) proposes to enrich the range of models in the framework and demonstrate and evaluate the use of the framework in multiple contexts, including linking to social and cultural attributes.

Work for this stage was undertaken by a Technical Group comprising researchers from eight science organisations (NIWA, AgResearch, DairyNZ, GNS Science, Manaaki Whenua/Landcare Research, SCION, Victoria University of Wellington). This collaborative approach made specialists from a range of science providers available to the project, covering the required areas of technical expertise. These included catchment hydrology, production systems, water quality, agro-economics and computer science.

This report:

- Provides an overview of the approach taken to model interoperability.
- Describes the standards and conventions used in this project.
- Presents information on each of the model components, including how they were set up and application in the case study.
- Describes how the model components were linked and applied in the integration software, including visualisation.
- Identifies key intellectual property management, governance, and stewardship needs.
- Recommends future approaches to interoperability.

## 2 Overview of the modelling approach

This section provides an overview of the model integration steps and approaches taken in this study, which are described in more detail later in the report.

### 2.1 Model integration software

Interoperable models require software to co-ordinate running of the various model components and manage data exchange. In the Stage 1 report (Elliott et al. 2017), a model framework called CSIP was recommended, but following discussions with stakeholders, the second alternative, Delta Shell was selected for use in Stage 2.

Delta Shell<sup>2</sup> (Donchyts et al. 2014; Deltares 2020) is a system developed by Deltares for model coupling, user interaction, and visualising input and output datasets. Some of the core hydraulic and water quality models created by Deltares have been set up in Delta Shell. It is free and largely open-source, which was an important criterion for adoption in the OLW project. Deltares also provides free hydrodynamic and water quality software, which was a further reason for adoption of Delta Shell in the OLW project. Delta Shell is based on Microsoft Windows and utilises the Microsoft .NET C# language. It runs on a desktop computer. The user interface includes mapping and time-series display components 'out of the box'.

### 2.2 Model components

Following workshops at the start of the project, several model components of key interest to OLW were selected for implementation in Delta Shell. These components were organised into a set of static models (not time-stepping) and a set of dynamic models, covering key aspects of catchment hydrology and water quality. An economic, production, and optimisation component was also envisaged. These components demonstrate some key aspects that an interoperable system of models for land-water management needed to cater for, but without addressing every need. Each component was set up using a standard model interface, the Basic Model Interface<sup>3</sup>, and established for the case study site (the Aparima River catchment in Southland). The components, which are detailed later in the report, are:

#### Static model components

- Lookup table nitrogen loss.
- Nitrogen transport through groundwater – MODFLOW.
- Nitrogen transport in streams – CLUES stream transport component.
- Production, economics and optimisation (LUMASS).
- Overseer nitrogen loss (partially completed).

#### Dynamic model components

- Web service to query pre-computed time series from the TopNet hydrologic model.
- Soil moisture accounting/rainfall-runoff model – from LUCI.
- Dynamic groundwater flow – MODFLOW.

---

<sup>2</sup> <https://oss.deltares.nl/web/delta-shell>

<sup>3</sup> <https://bmi-spec.readthedocs.io/en/latest/> - a set of functions for querying, modifying, and running models

- Dynamic stream flow routing (from TopNet).
- Dynamic nitrogen loss – APSIM soil-plant model.
- Dynamic groundwater nitrogen transport – MODFLOW.
- Stream nitrogen transport.

## 2.3 Standards and linkage

### 2.3.1 Component interface standards

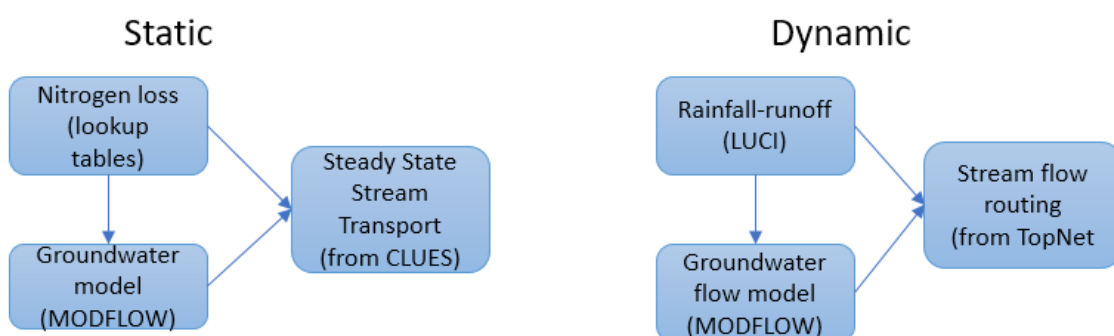
Delta Shell contains a set of **interface specifications** (objects and methods) that can be used for model components. Adopting those specifications would mean committing to a somewhat complex interface that is specific to Delta Shell, .NET, and the desktop computing environment.

Instead of using Delta Shell's own specification, we decided to use an interface standard called BMI (the Basic Model Interface), which is open and used by multiple frameworks. Deltares' own models are migrating more to using BMI, and Delta Shell has ways to run components that are set up with BMI. Hence BMI was considered by the technical group to be a preferable interface.

### 2.3.2 Model assemblies

Two key sets of linked components ('assemblies') were set up to trial and demonstrate model coupling, as shown schematically in Figure 2-1.

These model components were coupled using the programme DIMR (Deltares Integrated Model Runner) within Delta Shell. The programme specifies the components used in an assembly and their order of running, including allowing for dynamic models to exchange data each time-step (hence allowing feedback between components). While BMI and DIMR have ways of exchanging data items that are stored in computer memory, we found those facilities were not suitable for our models, and so we adopted a file-based data exchange approach.



**Figure 2-1: Schematic of components coupled in Delta Shell.** In the dynamic model, information is exchanged each timestep.

### 2.3.3 File formats and naming standards

A set of standard file formats and naming conventions was adopted, to organise and facilitate data interchange. The file formats are well-recognised open standards, and many software programs have methods to read in and output such data. The standard formats were:

- SQLite<sup>4</sup> for tabular data.
- YAML<sup>5</sup>, a text-based markup format, for small data structures such as parameter sets and model configuration files.
- GeoTiff<sup>6</sup> for spatial grids.
- ESRI shapefiles<sup>7</sup> for vector spatial data.
- NetCDF<sup>8</sup> files for array-based data (grids and time-series).

A set of naming conventions based on Community Surface Dynamics Modelling System (CSDMS<sup>9</sup>) standard names was adopted, although we customised the names to better represent the types of model quantities that are needed for land-water modelling. Some of these names are long and cumbersome, so a set of alias/shorthand names was also developed. While our method for coupling did not require strict adherence to these conventions, we encouraged their adoption to facilitate and clarify data exchange items.

#### 2.3.4 Data exchanges

An interoperable modelling system requires a mechanism to exchange data between model components. The BMI standard provides mechanisms for passing data through common memory, but we found that the DIMR implementation of BMI, and the BMI specification in general, had restrictions which were too limiting. To overcome this limitation, we adopted a file-based data exchange approach, using the set of standard files.

#### 2.3.5 Spatial and temporal representation

Land-water models operate on a variety of spatial scales and types. One of the requirements of the current project was to allow for representation of individual properties. This can introduce considerable complexity; for example, a farm can have its individual management, several soil types and slopes, and may contribute to streams in different catchment represented by a stream network underlaid by a groundwater grid. It is difficult to devise a fully generalised system to fully cater for the range of spatial and temporal systems without introducing considerable complexity or computational demand (which might arise from conducting all calculations at a lowest-spatial-denominator level).

We instead adopted an 'ecotope' approach, which involved conducting simulations for each spatial unit defined by the intersection of property boundary, land use, soil class, climate zone, irrigation (whether present) and slope class, which can be mapped onto a common grid basis. In hydrological modelling such areas are more commonly known as HRUs (Hydrological Response Units) but we have used the name ecotope to reflect an intention to undertake both land and water modelling. The results from each ecotope then provided inputs for other models, such as inputs to each stream segment, after results were summarised at the subcatchment basis (based on the grid of ecotope locations in conjunction with subcatchment boundaries). This approach allows for future extension, for example, introducing other spatially-varying driving factors, allowing farm management variables

---

<sup>4</sup> <https://www.sqlite.org/index.html>

<sup>5</sup> [https://docs.ansible.com/ansible/latest/reference\\_appendices/YAMLSyntax.html](https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html)

<sup>6</sup> <https://www.ogc.org/standards/geotiff>

<sup>7</sup> <https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>

<sup>8</sup> <https://www.unidata.ucar.edu/software/netcdf/docs/>

<sup>9</sup> [https://csdms.colorado.edu/wiki/About\\_CSDMS](https://csdms.colorado.edu/wiki/About_CSDMS)

to vary by property or farm-level predictions to take account of farm system behaviour. However, there are some limitations, such as the inability to represent the detailed placement of edge-of-field mitigation measures or individual paddock management.

The models can also differ by temporal resolution and scale. We adopted a day as the basic unit for data exchange in dynamic models, although individual models may operate internally at a finer or coarser time-step.

### 2.3.6 Model repository and management

We used the GitHub document repository to share model components, assemblies, data and related software. This system is publicly-accessible and widely used for software development projects. We did not use version management or revision facilities of GitHub in the current project, mainly because individuals or small groups were responsible for developing and maintaining their own components. However, we envisage more sophisticated use of GitHub in the future.

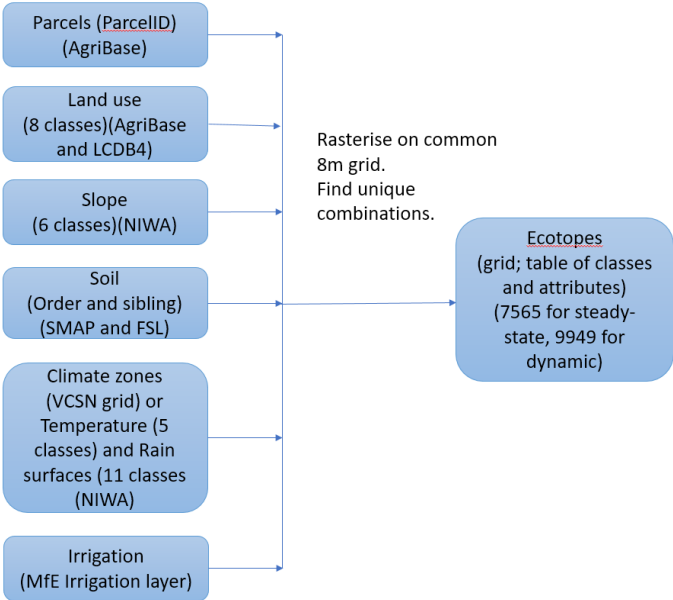
## 2.4 Introduction to the Aparima case study

Model components and assemblies were tested in the Aparima River catchment (1280 km<sup>2</sup>) upstream of Thornbury (Figure 2-2). The catchment is dominated by pastoral land use, although there is about 15% native, 14% exotic forest, 6% tussock, and a small amount of cropping and urban land use.



Figure 2-2: Aparima case study location.

Ecotopes were generated as shown in Figure 2-3. This was a pre-processing exercise conducted with software outside Delta Shell. In the future, model components could be set up to generate ecotopes through open-source spatial processing libraries such as GDAL or using LUMASS spatial processing facilities (<https://bitbucket.org/landcareresearch/lumass>).



**Figure 2-3: Schematic summarising the ecotope generation process.**

The Aparima model components and model assemblies were not calibrated or applied to scenarios. Such work was outside the scope of the current project. The main emphasis was to demonstrate how the model components and assemblies could be set up and run in a real catchment.



## 3 Model standards and approaches adopted

### 3.1 BMI as the standard for linking

Substantial work on Stage 2 began with the selection of Delta Shell (Donchyts et al. 2014; Deltares 2020) as the computational framework for the interoperable model system. In a system of interoperable models, the framework is responsible for controlling the execution of the individual models and coordinating the passing of data from one model to another and to the users of the system. For the framework to do its job, the component models must share a common set of commands – called an Application Programmer Interface or API – that carries out the model’s execution and data-passing tasks. The selection of a framework sets the API to which the component models must conform.

Delta Shell supports the use of two linking APIs: its own native interfaces, as defined by its developers at Deltares, and the Basic Model Interface or BMI, an interface developed by the Community Surface Dynamics Modeling System or CSDMS led by researchers at the University of Colorado<sup>10</sup>. The Interoperable Models technical group chose to use BMI for linking the component models to the framework because it is compatible with, but not limited to, use in Delta Shell.

The version of BMI used in the Interoperable Models programme is defined by Deltares in an open source project called Open Earth<sup>11</sup>. A newly programmed model can provide implementations for the 17 functions that are declared in Open Earth’s BMI specification and be natively compliant with BMI. An existing model with a non-compliant programmer interface can be provided with a “wrapper” that makes the model BMI-compliant by translating each of the BMI functions into equivalent actions that can be carried out in the model’s native context.

For example, every BMI-compliant library must provide a function named “initialize” that takes one string as an argument and returns an integer value. This function is used to initialize a model from a configuration file. The actions that this function carries out may be as simple or as complex as the model requires, but it must always be named “initialize,” take a single string argument and return a single integer value that indicates success or failure in initializing the model. A model that is not BMI-compliant might initialize itself through a series of functions that read from multiple files, or through calls to a database. A BMI wrapper for this model would provide a single initialize function that manages the model’s start-up actions according to the contents of one file. This might require the model’s developers to create a new configuration file specifically for use by BMI.

Whether a model is natively BMI-compliant or embedded in a BMI-compliant wrapper, the result is a program library (a DLL in the Windows computing environment) that can be linked directly into Delta Shell or controlled by a program called DIMR (the Deltares Integrated Model Runner).

To run linked models in DIMR does not require writing and compiling a new model-linking plugin for Delta Shell. The models to be linked, the order of their execution, whether they are run sequentially or in parallel, and which variables are shared between programs are all specified in a DIMR configuration file. The DIMR configuration file is written in XML, which can be composed and edited with a text editor and customized for use with the specific combination of models required for a particular study without the need to construct new Delta Shell modules in C#. For this reason, the

---

<sup>10</sup> [https://csdms.colorado.edu/wiki/BMI\\_Description](https://csdms.colorado.edu/wiki/BMI_Description)

<sup>11</sup> <https://github.com/openearth/bmi>





To further characterise objects with the aim to avoid ambiguity, adjectives may be added to an object or sub-object using a tilde '~':

```
object_subobject~attribute_quantity      land_surface~10m-above-air__temperature
object~attribute~attribute_subobject__quantity  tree~oak~bluejack_trunk__height
```

Variables denoting flow rates per unit area between two objects ('control volumes') can be specified using the base quantity 'flux'. CSDMS specifies different types of fluxes, two of which are relevant for our project:

```
mass_flux          [kg m-2 s-1]
volume_flux        [m s-1] = [m3 m-2 s-1]
```

Since fluxes relate to two objects, i.e., source and sink, the direction of the flux can be explicitly specified using the adjectives 'incoming' and 'outgoing', for example:

```
atmosphere_aerosol_radiation~incoming~shortwave__absorbed_energy_flux
atmosphere_aerosol_radiation~outgoing~longwave~upward__energy_flux
```

One objective of the CSDMS standard names is to enable automatic matching between different models. To increase the chance of having matches, the convention seeks to avoid incorporating 'assumptions' into the names and to use very basic terms that do not imply specific characteristics of an object. For example, CSDMS uses the term 'channel' as opposed to river, creek, or ditch, although they're referring to the same concept but often are associated with a certain size of that object. Please note that assumptions need to be understood in a broad sense and include conditions, simplifications, approximations, and other clarifications.<sup>13</sup> They can be specified as part of the Model Coupling Metadata (MCM) file using the <assumption> tag in CSDMS.

### 3.2.2 Application of naming rules

In this section we provide examples of standard names we developed for our static nitrogen model (Figure 2-1 and Figure 5-1), specifically the 'Typology Lookup' (Section 4.2, Step 2 in Figure 5-1). The typology look-up provides nitrogen loss values for ecotopes under non-pastoral farming land uses. Ecotopes represent a unique combination of environmental and land-use characteristics and represent the land-based spatial computation unit (Section 2.3.5) for the static and dynamic models. Input items required to look up the correct output item, i.e., the nitrogen loss value, for an individual ecotope are rainfall, slope, soil type, and land use. As outlined in the previous section, the standard names define quantities or identifiers that are associated with specific objects used in the model. The input item rainfall refers to the water content in the atmosphere and can be described by applying the 'object\_subobject' pattern, i.e.,

```
atmosphere_water__<quantity>
```

where atmosphere represents the root object and water the sub-object referred to by the quantity name. Since the static model uses average values, the rainfall is provided as a 10-year average value expressed as a flow rate, i.e., mm x yr<sup>-1</sup>. The latter can be coded using the 'volume\_flux' pattern explained in the previous section. The former descriptive part can be prepended using the 'description' pattern together with the multi-word pattern to form the full name used in the project:

---

<sup>13</sup> [https://csdms.colorado.edu/wiki/CSN\\_Assumption\\_Names#Boundary\\_Condition\\_Assumptions](https://csdms.colorado.edu/wiki/CSN_Assumption_Names#Boundary_Condition_Assumptions)

atmosphere\_water\_\_10-year\_average\_precipitation\_volume\_flux

Slope as input parameter is used in different model components in the project that refer to different objects. In this case slope refers to the average slope value of a given ecotope and was consequently modelled as:

ecotope\_\_slope

The soil type input parameter refers to a specific identifier associated with a specific soil classification. Since the same model could be implemented using different soil classifications, we associate the soil-type identifier with the model rather than the ecotope or land-surface object. The identifier, i.e., the 'quantity' in this case, is simply referred to as 'code':

model\_soil-type\_\_code

We apply the same principle to the land-use input parameter and:

model\_land-use-type\_\_code

The output of the typology look-up is the nitrogen loss given in  $\text{kg} \times \text{ha}^{-1} \times \text{yr}^{-1}$ . Since the loss is provided as a mass flow rate, we apply the 'mass\_flux' pattern as described in the previous section to form our quantity name:

<object>\_\_mass\_flux

To name the object the nitrogen flux is associated with, we need to know a little bit more about what the typology look-up is referring to. In our case the model component refers to the mass of nitrogen provided by nitrate leaching from the root-zone of the soil into the groundwater. Therefore, the root object is soil and more specifically the water in the soil:

soil\_water\_<x>\_\_mass\_flux

Since we only have leaching once the soil reaches field capacity, we refine the object description by referring to the soil water in the saturated zone of the soil:

soil\_water\_sat-zone\_<x>\_\_mass\_flux

and more specifically to the nitrogen in the soil water:

soil\_water\_sat-zone\_nitrogen\_\_mass\_flux

Because we use the same nitrogen flux also as input in a different context, we further refine the name using the 'adjective' pattern and explicitly state that we refer to the nitrogen flux leaving the saturated zone of the soil to avoid ambiguity:

soil\_water\_sat-zone\_nitrogen~outgoing\_\_mass\_flux

### 3.3 File formats

Neither Delta Shell nor BMI has specific, binding requirements with respect to file formats for data or model configurations. CSDMS does recommend YAML<sup>14</sup> for configuration files, and the Interoperable Models developers followed their suggestion, for the most part. Where pre-existing models had their own configuration file formats – as in the case of the hydrologic routing model extracted from TopNet – those were adopted without alteration.

For data files, the team chose to use SQLite in the case of the steady-state models and netCDF for the dynamic models. Both of these file formats are supported by Delta Shell's built-in features, and program libraries to support both formats are available for all the programming languages used in this project.

YAML files are composed of plain text, grouped as keywords and values. In this respect it is much like XML, JSON, and numerous other formats. YAML is a superset of JSON, that is, a YAML parser can always read a JSON file, but the reverse is not necessarily true. Both YAML and JSON use the colon (':') character to separate keywords from values. YAML files contain little other special punctuation and are intended to be readable by humans as well as computer programs. In developing BMI compliant models or model wrappers, the Interoperable Models developers used YAML to specify operational settings such as input and output data file names or working directories. A line in a configuration file that specifies a particular netCDF file for output might look like this:

```
Output_File: out_data.nc
```

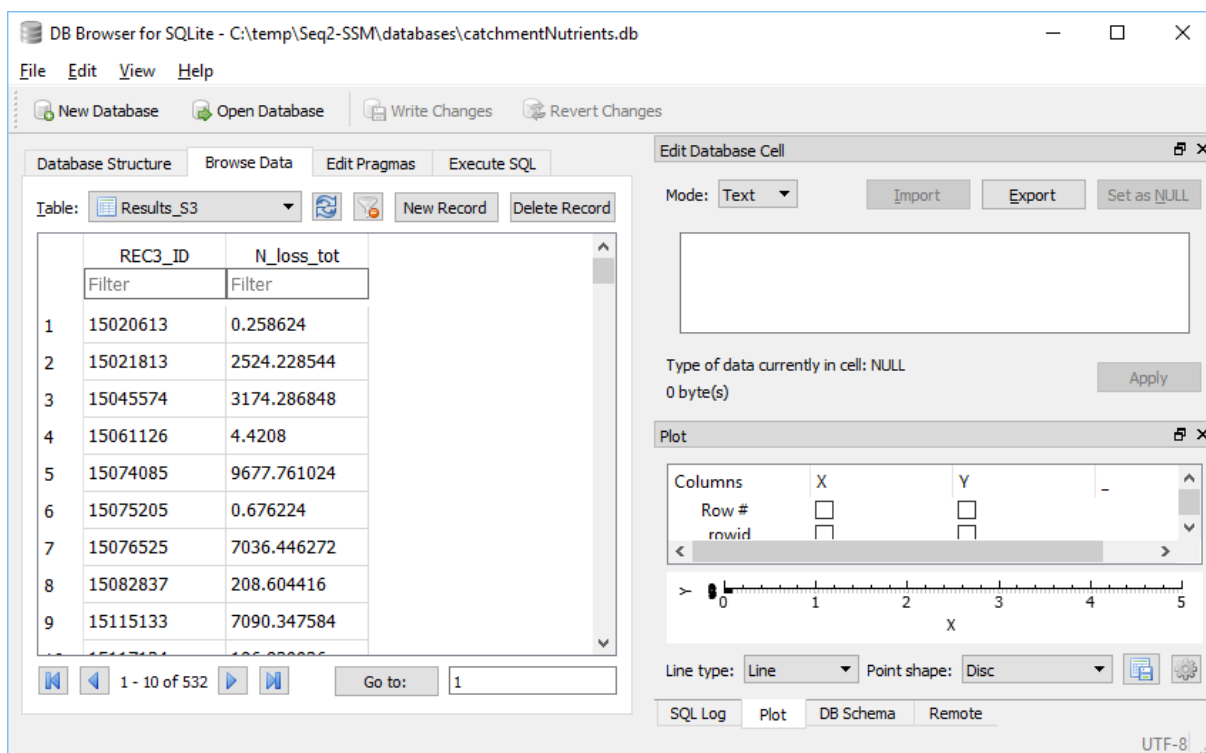
SQLite is a small implementation of the relational database mode and the SQL language<sup>15</sup>. SQLite files contain tables of data, which can be linked by shared values in selected columns, referred to as "keys." The interoperable models team used SQLite's database tables extensively for the steady-state models, where the results took the form of a single numerical value for each reach or catchment corresponding to a REC<sup>16</sup> number. The SQLite library can be accessed in the Python language and in Delta Shell. SQLite files can also be viewed in standalone programs such as SQL Browser. Here is a model result, presented in SQLite Browser.

---

<sup>14</sup> <https://yaml.org>

<sup>15</sup> <https://www.sqlite.org>

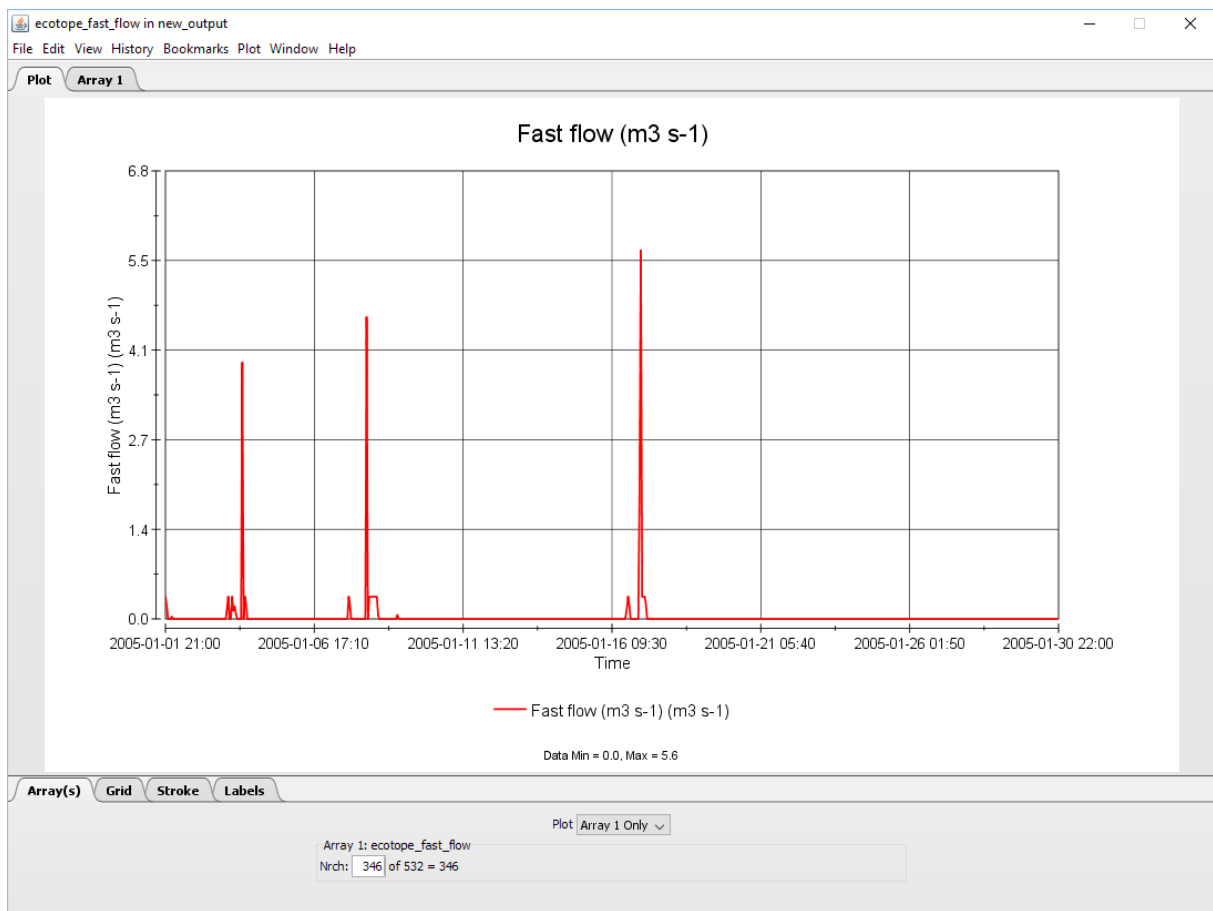
<sup>16</sup> River Environment Classification, <https://www.mfe.govt.nz/more/science-and-data/classification-systems/freshwater-classification-system> or <https://niwa.co.nz/freshwater-and-estuaries/management-tools/river-environment-classification-0>



**Figure 3-1: A representative model result, presented in SQLite Browser.**

Unlike the general-purpose formats, YAML and SQLite, netCDF is specifically designed for scientific data, especially datasets presented in arrays<sup>17</sup>. The Interoperable Models team used netCDF to store time-series results, such as hydrographs. NetCDF is self-documenting, which means that it uses standardized methods to store metadata, such as units of measurement, directly in the data file, along with the data itself. Like the SQLite format, netCDF is supported both by standard libraries available to programmers, and with standalone applications for loading, dumping, and viewing data. An example of a hydrograph from the LUCI model, displayed in Panoply, a netCDF viewer, is shown in Figure 3-2.

<sup>17</sup> <https://www.unidata.ucar.edu/software/netcdf/>



**Figure 3-2: Example hydrograph from the LUCI model.**

### 3.4 File-based coupling approach

DIMR's implementation of BMI provided tools for initializing and executing models both sequentially and in parallel, but passing data among the models was problematic. DIMR does allow variables to be shared among models, but with the restriction that those variables can only have scalar values. This limitation presented a serious obstacle to the use of DIMR in the interoperable models program and in the Aparima test cases.

It is easiest to express this difficulty with an example. The Aparima models are organized around stream reaches and sub-catchments identified by numbers in the REC stream database. In the system's rainfall-runoff model, for example, each reach has a corresponding sub-catchment and the link between the two is represented by the sharing of an eight-digit identification number. Rain falling on sub-catchment 15009980 produces runoff which appears as an increased flow in reach 15009980. In the runoff model there are three vectors representing: rainfall amounts, runoff values, and in-stream flow values, for 9,749 reaches and sub-catchments. A fourth vector holds the ID numbers for the reaches. Using a single scalar value for each of these quantities would have required a configuration with nearly 40,000 individual elements, instead of 4 vectors.

To work around this limitation, the team chose to pass data from one model to another using files, rather than in-memory transfers. Some simple tests confirmed that this approach would work with DIMR, and a hybrid BMI-plus-data-files approach was used for the actual tests. For steady-state models, data was passed in SQLite tables, and in dynamic models, vectors were passed in netCDF files.

The file-based approach meant that more coordination was required among the models than would have been necessary with an approach based entirely on BMI. For example, the name of the data file used for passing values had to be part of each model's configuration, and relocating a collection of models from one computer or file system location to another required careful editing of each model's configuration files so that they all referred to the same file in its new location.

This also required more modification of the individual model programs. In a strictly DIMR implementation, it is possible to indicate in the DIMR configuration file that variable A in one model is the same thing as variable B in another model, and DIMR will correctly handle the passage of values between the two. In the file-based approach it was necessary for all models using a given variable to use the same name for that variable, at least in the input and output sections of the program.

## 4 Model components

The following sections provide descriptions of each of the model components. For each component we address the following information (where appropriate for the component):

- A description of the nature and purpose of the component.
- Key aspects of component implementation, such as the language and setting up the model for use in Delta Shell, and inputs and outputs.
- A summary of progress in component development.
- Description of implementation in the Aparima case study.
- Key findings and future recommendations.

The formatting of this information varies, due to the varying nature of the components.

### 4.1 Web service for retrieval of pre-computed stream flows

#### 4.1.1 Description of the nature and purpose of the component

A web service was used to retrieve pre-computed flows from the TopNet model. The data were set up on a server hosted at NIWA, and a web service was set up to provide the data using the Sensor Observation Service (SOS) standard web protocols established through the Open Geospatial Consortium. While the standard is directed mainly at delivering sensor observations, it can also be used for displaying other time-series data. A plug-in was written within DeltaShell to retrieve data from the web service, and display the resulting time series.

The web service plugin<sup>18</sup> was written in C# and can be included in Delta Shell. It allows us to select given reaches for a given interval of time, and it will retrieve the corresponding time-series for simulated river flow as needed.

#### 4.1.2 Key aspects of component implementation

We used a web-based service developed by 52North<sup>19</sup> to retrieve data. It is used at NIWA to store observed stream flows around the country, but we extended its use to store simulated stream flows.

52North helped us by providing a data-ingestion script, so that we could store our simulated results, which are natively in netCDF file format, into the database used by the web service. Once this was ready, we wrote a simple Delta Shell plugin that can talk to the SOS API (Application Programming Interface) to retrieve the pre-compute flow data. The plugin is based on SOSClientJSON<sup>20</sup>, which is a generic service to query the SOS API using JSON. The plugin is also able to show time-series of the flow (see Section 6.2).

---

<sup>18</sup> <https://github.com/daniel-lagrava-niwa/SosService>

<sup>19</sup> <https://github.com/52North/SOS/>

<sup>20</sup> <https://github.com/daniel-lagrava-niwa/SOSClientJSON>



### 4.1.3 Final state of component development

We developed the code and it was tested on the SOS service running on an internal NIWA server.<sup>21</sup> Some values were hardcoded, namely the server address and the procedure. The code is available on GitHub on the aforementioned sites.

While we were able to show the time-series, we did not provide more advanced visualisation, as this was meant to be a prototype of a service to interact with SOS.

### 4.1.4 Key findings and future recommendations

We found that SOS is a useful platform to store simulation results. There are projects at NIWA to provide simulation results using this platform, which have been encouraged by achievements in the OLV project.

There currently exists an official *R* client that can talk to SOS.<sup>22</sup> In that same vein, it may be useful to provide such official clients for users of other programming languages. We think that the C# plugin developed here could be a candidate to be extended and improved to act as a client for Windows users.

## 4.2 Nitrogen loss lookup

### 4.2.1 Description of the nature and purpose of the component.

This component maps ecotope attributes to corresponding nitrogen loss rates. When provided with a combination of land use, climate, slope, soil type, irrigation class and precipitation class, it will retrieve the nitrogen loss rate associated with that ecotope. The nitrogen lookup data are based on typologies derived by AgResearch.

### 4.2.2 Key aspects of component implementation

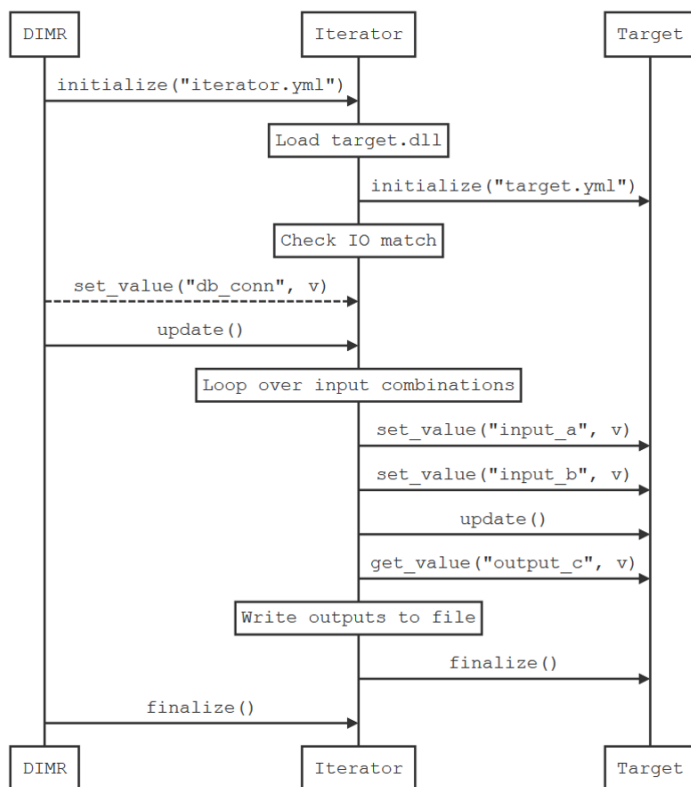
The lookup functionality was implemented in a generic way. The lookup data are described entirely in a configuration file, thus making the component itself independent of the dataset it exposes. This approach allows for the lookup component to be used with any key-value based dataset. The configuration of the dataset supports multiple keys, value clustering (mapping input values to classes), wildcards and fallback values (when no match is found).

The lookup component is essentially a point model, which means it can only be applied to a single computational unit at a time, only accepts scalar inputs and only produces scalar outputs. The problem of applying a point model over a collection of computational units (i.e., grid cells) seemed generic enough that a sister component was built to perform this task: the BMI iterator. Figure 4-1 shows how the master controller, the iterator and the point model interact. The iterator isolates the master controller entirely from the point model (the lookup in this instance).

---

<sup>21</sup> <http://wellsensorobsp.niwa.co.nz:8080/52n-sos-aquarius-webapp/service>

<sup>22</sup> <https://github.com/52North/sos4R>



**Figure 4-1: Sequence diagram of interaction between a master controller and target point model component.** In this instance DIMR, through an iterator component and to a target such as the nitrogen loss lookup.

#### 4.2.3 Progress in component development

The model was successfully implemented as a BMI component. A simple test example was set up to confirm that known outputs are obtained with known inputs, and the model was also tested using the BMI model runner.

#### 4.2.4 Implementation in the Aparima case study

While the component technically works, the data it contains do not allow it to provide specific nitrogen loss rates for every single ecotope occurring in the Aparima. This is because of fundamental differences in the methodologies used to define typologies and ecotopes. The current module falls back on a default nitrogen loss for ecotopes that could not be matched. Further validation of the broader system is needed to assess the impact of this limitation and if necessary, to optimize the fallback value.

#### 4.2.5 Key findings and future recommendations

The lookup and iterator components encapsulate well-defined functionality in reusable BMI components. The use of such components makes the system easier to compose and to manage, effectively hiding complexity from the master controller.

A review of the currently embedded nitrogen loss data is recommended so that it may better fit the Aparima case study.

## 4.3 Overseer nitrogen loss

### 4.3.1 Description of the nature and purpose of the model component

Through involvement in the project, AgResearch aimed to:

- Develop a model component for calculated N leaching loss values that would conform to the OpenEarth BMI specification.
- Identify an input source for calculating N leaching loss values (below the root zone) for a given set of model parameters.
- Design an interoperable tool that can load N leaching loss values per ecotope into the Interoperable Static model.

### 4.3.2 Key aspects of the model component implementation

- Overseer was identified as the definitive source of N leaching loss calculations.
- For connecting through to the Interoperable Static model, AgResearch developed a BMI-compatible DLL that connects the input source of N leaching loss values based on YAML configuration files, and writes the output of processed data to an SQLite output table.
- Designed model flow:
  - AgResearch would receive, from the interoperable model, an input file of ecotope values (id, soil, rain, slope and enterprise type).
  - AgResearch would map each unique ecotope to the closest existing Overseer model farm file.
  - AgResearch would interface to Overseer using the farm id and the ecotope soil and slope values.
  - Overseer would run the existing farm file with the new soil and slope values and generate a nitrogen leaching loss value for the farm.
  - Overseer would interface the nitrogen leaching loss value back to AgResearch.
  - AgResearch would interface the nitrogen leaching loss value for each ecotope back to the interoperable model.

Figure 4-2 provides a schematic of the as-designed data flow for the nitrogen leaching loss model:

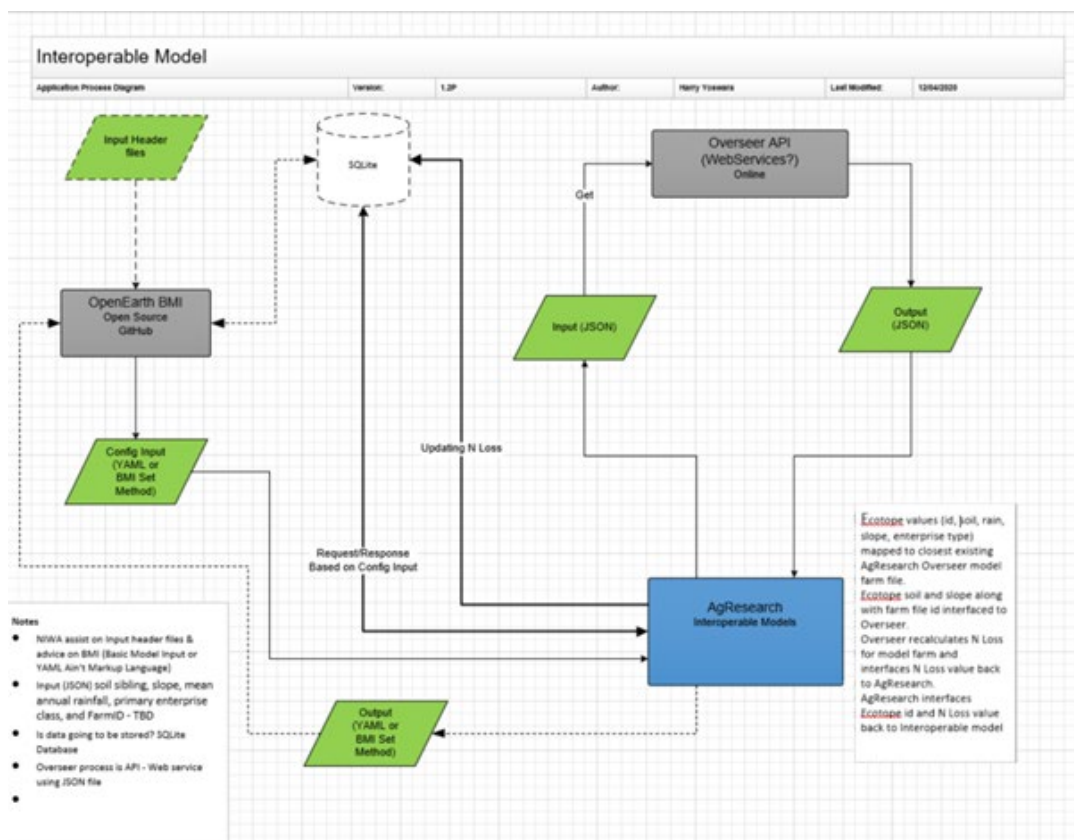


Figure 4-2: Schematic of the as-designed data flow for the nitrogen leaching loss model.

### 4.3.3 Progress in model component delivery

Discussions were held with Overseer Limited about the potential of opening an API to allow us to load various values against AgResearch’s model farms presently in Overseer (to mimic ecotope value), and then receive calculated N leaching losses based on these values. Overseer Limited agreed in principle this could be done, although it would require a new API to be developed (because of the number of ecotopes that would need to be processed), rather than using their existing one. Given the timelines of the project, and conflicting priorities for Overseer Limited, AgResearch, in conjunction with Overseer Limited, were not able to develop this API.

As a workaround, AgResearch (in conjunction with NIWA) developed a BMI compatible DLL that connects the input source of N leaching loss values (based on YAML configuration files), and writes the processed data to an SQLite output table, which served as a look-up table.

Using a SQLite lookup table (as a substitute for N leaching loss numbers derived from Overseer), the project team was able to successfully test-run the static model using the BMI compatible DLL, compute nitrogen losses per ecotope, and use these values in the Aparima catchment model.

## 4.4 Steady state nitrogen routing in streams

### 4.4.1 Description of the nature and purpose of the component

Transport of mean annual loads of nitrogen down the stream network was undertaken using model components from the CLUES<sup>23</sup> catchment model (Elliott et al. 2016). The component takes mean annual loads of Total Nitrogen (TN) entering each segment of a dendritic drainage network (in this case, the REC2 network), accumulates the contaminants down the network as streams converge, and decays the load according to a first-order, length-based decay coefficient, where the decay coefficient is a function of flow. Concentrations of TN for each stream segment are determined by dividing the load by the flow rate, and making a correction to account for the translation between flow-weighted concentrations and median concentrations. The model formulation and parameters are as in CLUES. However, a modification was allowed to enable inputs to each segment to come from groundwater sources and from 'direct' sources (which bypass the groundwater system), plus point sources.

### 4.4.2 Key aspects of component implementation

The stream routing model was re-written in Python, from its initial FORTRAN code. This was because the Python code is small, and there are libraries in Python to facilitate input and output in the selected standard file formats.

Model parameters and input file locations were set up in a YAML configuration file. Tabular data, including baseflow and direct inputs, flow rates, and segments were read into the programme from a single SQLite file as part of the BMI initialisation method. The main calculations (routing, concentration calculation) were done in the step BMI method, while output of results into a single SQLite file was undertaken in the finalise BMI method.

An additional 'helper' component was built in Python to assemble and join model inputs into a single file, where the inputs may be contained within different SQLite tables and databases, and with different naming conventions to those used in the routing programme. This involved joining data using the SQLite3 package in Python, with segment ID's serving as the joining column. This enabled flexibility in the location and naming of the model inputs. The same helper component was used to translate output names and locations, if desired.

A further helper component was developed to take direct losses specified on an ecotope basis into stream inputs specified on a stream segment basis. That required pre-calculation of the area of each ecotope in each subcatchment (each subcatchment is associated uniquely with a stream segment), and then calculating an area-weighted summary of ecotope losses using SQLite to determine the total direct losses for each segment.

### 4.4.3 Progress in component development

The model was successfully implemented as a BMI component. A simple test example was set up to ensure that known outputs are obtained with known inputs, and the model was also tested using the BMI model runner.

---

<sup>23</sup> <https://niwa.co.nz/freshwater-and-estuaries/our-services/catchment-modelling/clues-catchment-land-use-for-environmental-sustainability-model>

#### 4.4.4 Implementation in Southland case study

The model was set up and run successfully with input data for the Aparima, including inputs from the groundwater and surface water models, with default parameters.

#### 4.4.5 Key findings and future recommendations

Implementation of the steady-state stream routing model from CLUES was straightforward and successful. The helper component for assembling and renaming outputs could be of use more broadly. The file sizes are small, so that file-based data exchanges were suitable. There are no IP concerns regarding this component, because the model and parameterisation are already published.

### 4.5 Steady state groundwater routing

#### 4.5.1 Description of the nature and purpose of the component

A groundwater model was used to simulate steady-state flow of water from rainfall recharge sources, through aquifers, and discharge to gaining streams. MODFLOW-NWT (Niswonger et al. 2011) was used to simulate this interaction, which uses a finite-difference grid to simulate saturated groundwater flow in 3-dimensions.

Groundwater–surface water interactions for this project only considered gaining streams, where flow of water is only from groundwater to surface water. Losing streams (where groundwater receives recharge from streams), or routing to accumulate flow were not required. For these reasons, MODFLOW models that use the DRAIN package can be used to represent streams. Multiple streams in finite-difference grid cell are represented by multiple boundary conditions for that cell, with varying conductance terms that are proportional to the length of the stream within each cell.

#### 4.5.2 Key aspects of component implementation

MODFLOW-NWT is public-domain Fortran software, but embedding BMI methods within the source code would require a significant rewrite. Instead, a BMI interface for groundwater simulation was written using a combination of C++ and Python programming languages. A low-level C++ interface, named GroundwaterBMI, provides the OpenEarth BMI compatible DLL required for DIMR and DeltaShell. The Python API (Python.h) was used to control high-level Python modules based on FloPy, a Python package for MODFLOW, and related software (Bakker et al. 2016). This high-level Python interface allows spatial properties and boundaries to be controlled programmatically, and for results to be processed and stored in SQLite and netCDF outputs.

The C++ interface was designed to be adaptable for any model and can be used for steady or dynamic time modes with nutrient transport (discussed in later sections). The Python interface was designed to also adapt to different time and nutrient transport modes, but required case-specific scripting to coordinate data and models for the Aparima case study.

BMI functions were provided for both C++ and Python interfaces, including ‘initialize’ and ‘update’. The GroundwaterBMI C++ module is initialized with a YAML configuration file with two required keys: ‘run\_dir’ for the run directory with the simulation files required for the case study, and ‘args’ (for arguments) to configure options for a simulation. The run directory must contain a Python module file named ‘run.py’ which contains a Python class named GroundwaterModel, which characterises the high-level logic required to run the case-specific groundwater model. An instance of GroundwaterModel is created using an ‘initialize’ class method, which takes the path to the YAML

configuration file as a parameter and passes the 'args' to initialise the GroundwaterModel instance. A 'timeword' argument allows the models to be run using 'steady' or 'dynamic' modes.

The 'initialize' method reads the YAML configuration file, runs a steady-state MODFLOW model once and writes output files. While the input sources for several types of data (e.g., rainfall recharge, groundwater abstraction or river stage) could be re-written, this was not done for steady state groundwater routing. Outputs from the MODFLOW model are post-processed and stored into both SQLite and netCDF output files, which include the following variables:

- soil\_water\_sat-zone\_top\_\_depth - groundwater hydraulic head [m] or water table elevation, based on the MODFLOW grid
- soil\_water\_sat-zone\_\_volume\_flux - groundwater flux on water table [m/s] or rainfall recharge, based on the MODFLOW grid
- model\_groundwater\_\_coverage - fraction of catchment area covered by groundwater model, tabulated per catchment, with values between 0 and 1
- reservoir\_water~outgoing\_\_volume\_flux - groundwater flow to streams [m<sup>3</sup>/s], tabulated per catchment, based on the total flux leaving the groundwater model from the boundary condition used to represent streams

The 'update' method for the steady mode only runs the same model again, but with perhaps shorter runtime, as the initial heads are provided from the most recent simulation. The SQLite tabular catchment data are rewritten each run, while the outputs with timestep dimension are appended to the 'time' dimension of the netCDF output file.

Internal methods were added to the Python interface to help translate polygon-based information to grid-based information, and vice-versa. This allows spatially-varying data to be translated between regular MODFLOW grids and catchment polygons. This translation used a catchment weighting matrix to consider fractions of overlap areas between the two spatial domains.

#### 4.5.3 Progress in component development

Development of the GroundwaterBMI C++ component is complete, as it only wraps methods written in Python. The Python development for the steady model is also complete, although would need to be re-structured to work for different regions, as it would require site-specific models and data to be processed.

The steady groundwater model can be run in several ways, such as from a stand-alone command-prompt (with "python run.py --timeword=steady"), from DIMR, and it was briefly but successfully tested with DeltaShell to interact with nitrate loads from the lookup tables routed downstream using the Sparrow model.

#### 4.5.4 Implementation in the Aparima case study

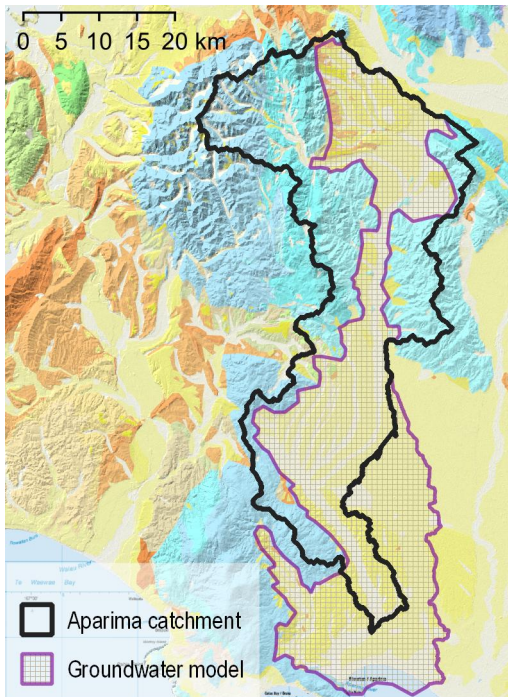
A previously developed groundwater model (Rawlinson et al. 2015) was used to simulate groundwater flow within the Aparima Freshwater Management Unit (FMU). This model was translated from a finite-element mesh to a finite-element MODFLOW grid prior to this study, and has not been calibrated. Recharge is based on a REC3 dataset with soil drainage.<sup>24</sup>

---

<sup>24</sup> pers. comm. Dr Christian Zammit, NIWA Christchurch 2015)



The boundary for this model does not completely overlap the Aparima catchment upstream of Thornbury (Figure 4-3). The groundwater model extends to cover the full Aparima FMU (including marine discharge), and is clipped to remove relatively impermeable bedrock geology units. As a consequence, areas of the Aparima catchment in the Hill Country do not have underlying groundwater model coverage, which requires special handling using the `model_groundwater__coverage` variable. The groundwater model also extends to the ocean, but these areas are outside the case study, so are absent from catchment outputs.



**Figure 4-3: Map showing boundaries of Aparima catchment, groundwater model and QMAP geology (Turnbull and Allibone 2003).** Bedrock is shown with blue shades, and sand & gravel with light yellow shades.

#### 4.5.5 Key findings and future recommendations

Recommendations for future implementation of the steady state model include allowing more variable inputs, such as recharge rates, and inclusion of groundwater abstraction locations.

### 4.6 Soil moisture accounting/rainfall-runoff model (subcomponent of LUCI)

#### 4.6.1 Description of the nature and purpose of the component

The model provides as output: overland flow and “quick” subsurface flow components to surface water bodies, and either direct recharge to groundwater or an estimation of the groundwater contribution to water bodies based on routing groundwater recharge via two linear reservoirs with associated residence times. The two ways of treating groundwater were to respect that in some cases, groundwater recharge would be fed to a sophisticated groundwater model, while at other times a surface water routing model might still require a simplified estimate due to either computational limitations or lack of coverage in groundwater model spatial domains. The implemented model takes in driving input rainfall data and potential evapotranspiration data. It tracks soil moisture over two domains: drainable water (water between field capacity and full soil saturation) and plant available water (water between permanent wilting point and field capacity). It also considers plant water stress, reducing potential evapotranspiration to account for stomatal



closure when soil moisture levels drop below a critical threshold (generally assumed to be half depletion of plant available water). Both infiltration excess and saturation excess overland flow are accounted for. Parameters include maximum surface infiltration capacity (mm/hour), maximum drainage rate to groundwater (mm/hour), drainable water (mm), plant available water (mm), three reservoir volumes (mm) and associated residence times (hours) (for “quick” subsurface flow, interflow and baseflow regimes respectively).<sup>25</sup>

#### 4.6.2 Key aspects of component implementation

Although the core algorithms are very much tried and tested simplified soil moisture accounting and rainfall runoff modelling procedures (used in Maxwell et al. in LUCI<sup>26</sup> and elsewhere), a new “clean” implementation of the algorithm in Python was developed for this project. This was done at least in part to meet the requirement to take in and pass data on a much more regular basis than in previous applications, and also to respect the specific data needs of the other dynamic models (a dynamic groundwater model and a surface water routing model).

#### 4.6.3 Progress in component development

The model was successfully implemented in BMI and passed data to both the groundwater and surface water routing models.

#### 4.6.4 Implementation in Southland case study

The soil moisture accounting/rainfall runoff model was set up and run successfully with input data for the Aparima, providing outputs to the groundwater and surface water models, with default parameters.

#### 4.6.5 Key findings and future recommendations

We achieved success insofar as the rainfall runoff component and other components (groundwater and surface water routing models) were all successfully implemented within BMI and were able to run together. However, achieving this was not straightforward and in future it would be worthwhile going beyond the proof of concept/demonstration of possibility and work through a more “calibrated” and full application of multiple models, getting the combined results to a point where they could support policy and decision making. Unsurprisingly, it took longer for the project team as a whole to work out the complexities of calling multiple models in a dynamic way than it did the static ones, so the project timeframe did not allow calibration/validation against observed output data etc. Many found the conventions in DIMR non-intuitive and there was little general documentation. Once things had been worked through however, there was more than enough power in the framework to enable the interoperability. Maintaining a record of how to achieve interoperability would very likely reduce the time required to take this work forward in the future.

### 4.7 Dynamic groundwater flow model

#### 4.7.1 Description of the nature and purpose of the component

The groundwater model component described in Section 4.5 was designed to enable a dynamic mode by running transient MODFLOW models, which allow groundwater flows to change with time. The dynamic groundwater flow model was also designed to take catchment recharge inputs from

---

<sup>25</sup> For further detail on the model implemented, including diagrammatic representations, please see Maxwell, D.H., Jackson, B.M., McGregor, J. (2018) Constraining the ensemble Kalman filter for improved streamflow forecasting, *Journal of Hydrology*, v560, pp 127-140.

<sup>26</sup> [www.lucitools.org](http://www.lucitools.org)

LUCI, and used to drive groundwater transport of nitrogen through aquifers (described in Section 4.10).

#### 4.7.2 Key aspects of component implementation

The GroundwaterBMI component described in Section 4.5.2 was designed to also support a dynamic mode, enabled by initialising the module with the argument 'timeword=dynamic'.

The 'initialize' method for a dynamic mode runs one steady-state MODFLOW simulation to establish the initial flow conditions, which are written to the output netCDF file at the first-time index. An initial steady-state groundwater simulation provides a numerically stable flow field for subsequent transient simulations.

The 'update' method for the dynamic mode does the following:

- Modifies the MODFLOW model to a transient simulation with one stress period with the duration provided by, for example, 'update(dt=1)' to simulate 1 hour of groundwater flow.
- Reads the LUCI netCDF file to establish time offsets and units for the output netCDF file from the groundwater module (first update only).
- From the LUCI netCDF file, reads the catchment recharge data (identified by the CDSM name soil\_water\_sat-zone\_top\_\_recharge\_volume\_flux), translate these values to a MODFLOW grid for recharge input with the MODFLOW Recharge Package.
- Runs the transient MODFLOW-NWT simulation.
- Post-processes results (same output variables described in Section 4.5.2) and appends along the time dimension to the output netCDF file.

Each MODFLOW simulation called by 'update' assigns the initial heads from the previous run's final heads, which allows a series of separate simulations to be chained together and behave similar to one MODFLOW simulation with multiple stress periods.

#### 4.7.3 Progress in component development

As with the steady groundwater model, development of the C++ and Python BMI components is complete. The dynamic groundwater model can be run in several ways, such as from a stand-alone command-prompt (with "python run.py --timeword=dynamic"), and from DIMR.

This component was successfully tested with DeltaShell to received recharge from LUCI.

#### 4.7.4 Implementation in the Aparima case study

The Aparima MODFLOW model (described in Section 4.5.4) was only steady state, so additional aquifer storage parameters were added to the model using the Python BMI interface.

Since the dynamic model receives external recharge rates from LUCI for a subset of the model (see Figure 2-1), the original recharge rates outside the Aparima catchment were preserved to maintain semi-realistic flow throughout the model.

#### 4.7.5 Key findings and future recommendations

A key finding was that a series of separate transient MODFLOW models behave similar to a single transient MODFLOW model with multiple stress periods. This is important for the approach used for this BMI implementation, because it is not feasible to pause/update a continuously running MODFLOW simulation.

Recommendations for future dynamic implementation include: use of more realistic aquifer storage parameters, allow data inputs from groundwater abstraction locations, and rates of abstraction.

### 4.8 Dynamic stream flow routing

The dynamic stream flow routing routes the surface runoff and baseflow along the stream network. In the “Surface water – groundwater – stream flow” coupling framework, it will route both surface water from rainfall-runoff model (e.g., LUCI) and baseflow from groundwater model (e.g., MODFLOW) along the stream network, provide a stream boundary for the groundwater, and also provide stream water available for irrigation purpose for the rainfall-runoff model if a suitable module is included in the rainfall-runoff model.

The dynamic stream flow routing here is based on the kinematic wave channel routing algorithm (Goring 1994), in which runoff produced by each sub-catchment is propagated as “particles” through the stream network to the basin outlet. This method has been applied in the TopNet hydrological model for flow routing (Clark et al. 2008).

The flow routing algorithm was originally structured as a series of subroutines in Fortran, with input and outflow formats as NetCDF. To facilitate interoperability with other models, the code has been redesigned as a Fortran module, with an interface that complies with BMI standards. Table 4-1 illustrates the BMI interfacing functions for the flow routing code.

**Table 4-1: BMI interfacing functions for the flow routing code.**

Interfacing function	Input / Output	Description
initialize()	Input: a configuration file which includes the time and time step setup, stream network, and initial flow in the stream network. Output: No	It initializes the stream network and get ready for flow routing.
update()	Input: the surface runoff and baseflow from each sub-catchment; water take amounts and relevant locations in the stream network. Output: routed flow at the outlet of each sub-catchment; river stage at each stream section of the stream network.	At each time step, it will accept surface runoff and baseflow from each sub-catchment and then route the water along the stream network after the stream water take.
finalize()	Input: No Output: No	It finalizes the program, i.e., deallocate the memory.

When implemented in the “Surface water – groundwater – stream flow” coupling framework, the code was compiled as a dynamic library and wrapped in Python to facilitate interaction with rainfall-runoff model (LUCI) and groundwater model (MODFLOW). Python also dealt with time step issues – the streamflow routing is run on hourly time step while MODFLOW is run at daily time step, and file exchanges; the agreed data exchange format between these three models is NetCDF, but NetCDF does not run in Fortran code in the Windows environment.

The “Surface water – groundwater – stream flow” coupling framework has been tested in the Aparima catchment, after carefully setting up configuration files and exchange files for each model.

Although the coupling framework with other two models (LUCI and MODFLOW) has been demonstrated, time constraints did not allow the coupled models implemented for the Aparima catchment to be calibrated, and the simulation results have not been analyzed,. This should be done in the future to make the model implementation more appropriate for decision support purposes.

In addition, the results of the dynamic stream flow routing are very sensitive to the selection of initial state variables in the setup. We recommend that model runs representing several years are used for future modelling purposes.

The flow routing model has the capability to simulate lakes and impoundments, but this function was not tested.

## 4.9 APSIM dynamic nitrogen source simulation

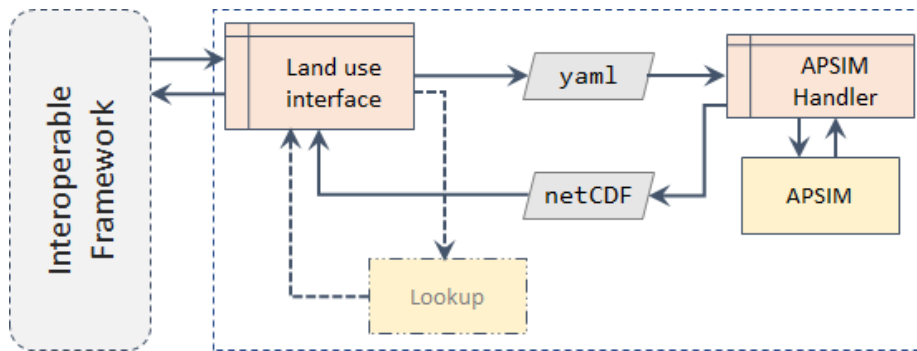
Plant and Food Research (PFR) was tasked to develop a component that simulated farm systems using the APSIM model (Agricultural Production Systems Simulator)<sup>27</sup>. This component provides estimates of biomass production and nitrogen leaching losses (at 150 cm soil depth) from various ecotopes defined by the Interoperable Models Framework, on a daily basis. PFR was also responsible for developing dynamic simulations that involve crop rotations, and to liaise with AgResearch who were responsible for simulations of pastoral systems. The technical details of this development, linkage with the Framework, limitations in the approach, opportunities for future development are discussed below.

### 4.9.1 General approach

APSIM is itself a framework composed of different models (see documentation on [www.apsim.info](http://www.apsim.info)<sup>26</sup>). The various sub-models simulate in detail a multitude of soil- and plant processes, as well as their interactions with weather and responses to management. For this project APSIM NextGeneration (version 2019.08.01) was used. APSIM is in active development and is employed on many projects in several countries around the world. As such, modifying the model itself to accommodate the interface needed for interaction with Delta Shell was considered potentially disruptive. In addition, the numbers of inputs required (especially to set up the management of different farming systems), was beyond the complexity envisaged for the work plan in the Interoperable Modelling Framework. The approach taken to avoid these issues was to develop a component, termed APSIMHandler, that controls the interaction between the Interoperable Framework and APSIM simulations. This component was developed in C# language, similarly to Delta Shell, and therefore should be able to interact with it seamlessly using the BMI interface. Currently, only basic communications occur directly between components – the information required for setting up new simulations is provided via text files (YAML format). The outputs from APSIM are collated and converted to a netCDF format as required by the Interoperable Framework. It should be noted that at present, APSIM is not able to provide outputs for all land uses. This made it necessary to identify other source of data to complement APSIM. A schematic of how the interaction of APSIM with the Interoperable Framework mediated by a ‘land-use interface’ is shown in Figure 4-4, with a lookup table used as alternative data source, as an example.

---

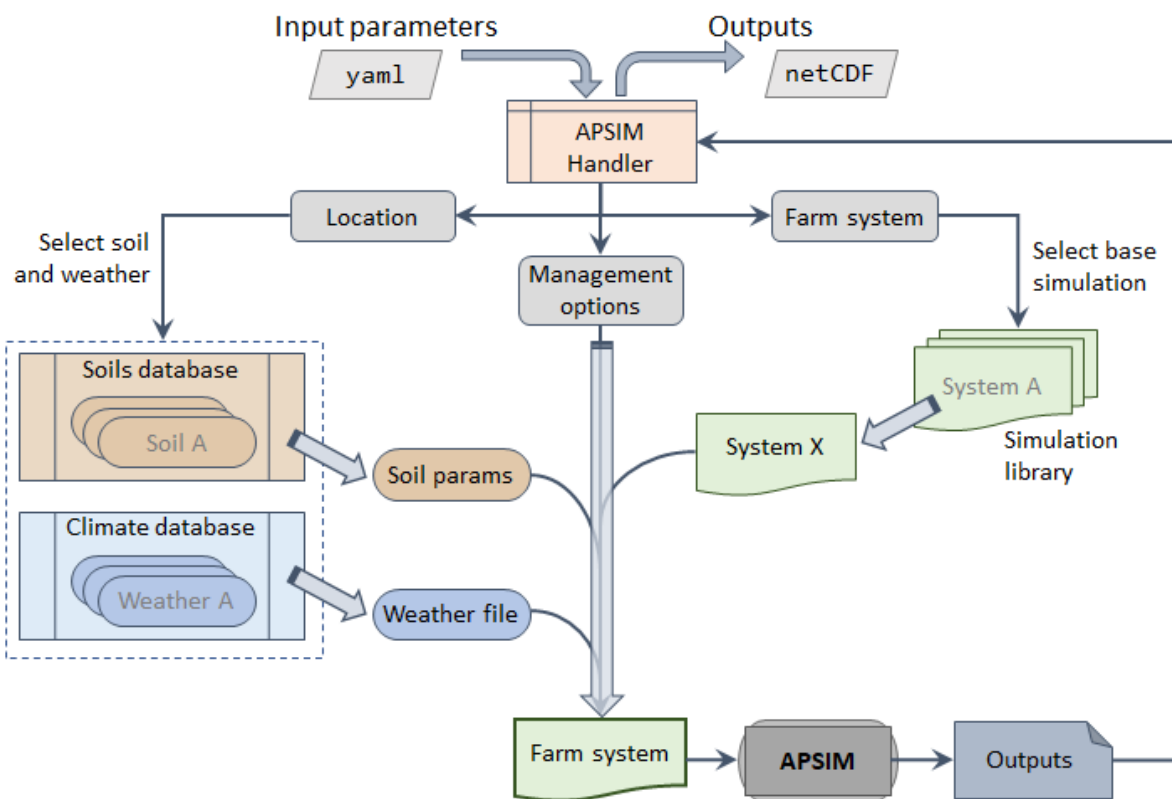
<sup>27</sup> <https://www.apsim.info/apsim-model/>



**Figure 4-4: Schematic showing how data were exchanged between APSIM and the Interoperable Model framework using several bespoke tools.** The Lookup indicates how outputs for land uses not currently simulated by APSIM may be incorporated.

#### 4.9.2 APSIM Handler workflow

The general workflow of APSIMHandler is shown in the schematic below (Figure 4-5). Starting from the top-left (input parameters), the handler acquires information from the Interoperable Framework, such as nature of farming system, soil type and location. This is used by the handler to select a representative simulation (farm typology) from a limited assortment. In addition, parameters are gathered from a soil library and a suitable climate file is selected and converted to the appropriate APSIM format. The selected simulation is then modified to include the specific soil parameters, the link to weather file, and management options. The handler then calls APSIM to run the simulation, producing an output file (an SQL database) that is read by the handler to collect the relevant data required by the Interoperable Framework. This procedure goes on until all the simulations required have been run, at which time a netCDF file is created with all the collected outputs. This file (and data content) is then used by other models in the Framework. The APSIM files generated in this process (weather, simulation, and outputs) can then be deleted.



**Figure 4-5: Schematic showing how data were exchanged between APSIM and the Interoperable Model framework using coordinated by the APSIMHandler.** The APSIMHandler may initiate several cycles of APSIM simulations to create the all the data required by the Interoperable Framework.

### 4.9.3 Supporting data

The information required by the APSIMhandler includes basic location information (currently latitude and longitude) – this is used to obtain appropriate climate data (from NIWA’s VCSN grid), and soil type (sibling name from S-map), which is provided directly to the Interoperable Framework. This relies on the Interoperable Framework accessing data from two databases:

- APSIM soil library, soil data from S-map (or some equivalent alternative) with required completeness and formatted for use in APSIM (a json file).
- Climate data, a set of netCDF files with weather data for the period to be simulated (daily values of rain, radiation, minimum and maximum air temperature, and wind speed).

These data have been made available for the case-study in the Aparima catchment. The APSIMHandler convert climate data supplied by NIWA in standard netCDF format to a text file for APSIM use. The soil data from SMap is provided by Manaaki Whenua Landcare in xml file format (for classic APSIM – version 7.10), which APSIMHandler converts to json format (the format of the version used in this project). The arrangement of soil parameters has been changing recently, and it may be necessary to alter the converter to ensure data are formatted correctly in future.

In addition, APSIMHandler obtains information about the farm system type (and potentially some management options) from the Interoperable Framework. To improve flexibility, the handler can define the simulation to be run in two ways:

- a simulation can be picked up from a set of pre-defined APSIM simulations, each with the basic setup of various farm systems (as json files), or
- a base simulation is modified using instruction for various farm types supplied in a configuration file (in YAML format).

The second option enables setting up management options that add nuance to the type of farm being simulated. This could potentially be used for testing scenarios that represent changes in management, rather than changes in overall land-use. Currently few generic farm types (dairy, sheep and beef, cropping) are being used. This approach creates difficulties in the Interoperable Framework when it may be necessary to control spatial variations – for example, data regarding the spatial variations in crop rotations and fertiliser levels are currently inadequate. Using APSIMHandler in isolation, it was possible to run simulations with a variety of cropping rotations and evaluate different fertiliser management options at selected locations in the Aparima catchment.

An example of a YAML file containing a series of instruction to set up APSIM simulations is shown below. This example defines a rotation involving barley, fodder beet and wheat:

```
simulationSetup:
clockSetup:
  - Name: SomeTimePeriod
    StartDate: 1973-03-01
    EndDate: 2018-03-31
  Surface.SurfaceOrganicMatter:
    InitialResidueMass: 1250
  Manager:
    - Name: RotationManager
      Parameters:
        CropList: AutumnBarley,FodderBeetLiftedEarly,AutumnWheat
        RotationHasPasture: true
        RotationLength: 3
        YearToKillOffPasture: 2
    - Name: CropsAutomaticFertiliser
      Parameters:
        FertiliserManagementIsEnabled: true
        FertiliserLevel: High
```

**Figure 4-6:** Example instruction file used by the APSIM Handler to run a scenario involving several cropping rotations to determine fertiliser management options. In this instance the APSIMHandler was run in isolation from the Interoperable Framework.

## 4.10 Dynamic groundwater nitrogen routing

### 4.10.1 Description of the nature and purpose of the component

Nutrient transport in groundwater was simulated using MT3D-USGS (Bedekar et al. 2016), which uses nitrogen loading from land-use and rainfall recharge to simulate transport through a MODFLOW simulation. Nitrogen may eventually return from aquifers to rivers.

### 4.10.2 Key aspects of component implementation

The GroundwaterBMI component described in Section 4.5.2 was designed to also support a transport model run with MT3D-USGS. The model requires spatially-varying mass-loading of nitrogen on the water table, and uses the advective flow result from MODFLOW to drive mass transport through the aquifer.

The 'initialize' method for a dynamic nitrogen mode runs one steady-state MODFLOW simulation followed by a 100-year-long MT3D-USGS transport simulation to establish the initial flow and concentration conditions, which are written to the output netCDF file at the first-time index.

The 'update' method for the dynamic transport mode does the following:

- Modify and run a transient MODFLOW model, as described in Section 4.7.2.
- Modify the duration of the MT3D-USGS model to have the same duration, in hours.
- Read the nitrogen loading values, translate this to the MODFLOW/MT3D grid.
- Run the MT3D-USGS transport simulation.
- Post-process results, which include:
  - soil\_water\_sat-zone\_\_nitrogen\_concentration - groundwater N concentration at water table [ $\text{kg}/\text{m}^3$ ], based on the MODFLOW/MT3D grid
  - Nitrogen transport from groundwater to surface water [ $\text{kg}/\text{m}^3/\text{s}$ ], per catchment

Each MT3D simulation called by 'update' assigns the initial concentrations from the previous run's final concentrations, which allows a series of separate simulations to be chained together and behave similar to one MT3D mass transport simulation with multiple stress periods.

For the steady flow model, input groundwater recharge remained constant. This was based on mean catchment drainage values from NIWA REC3 values, weighted to a finite difference grid. Nutrient loss rates were obtained externally from an SQLite file 'catchmentNutrients.db', which provided 'N\_loss\_tot' values for each catchment. These values were translated to concentration recharge values for MT3D-USGS simulations, which were run each 'update()' step.

### 4.10.3 Progress in component development

As with the groundwater models, development of the C++ component is complete, but more work is required to complete dynamic methods with the Python BMI. Nitrogen loss from the steady flow model is supported, using 'N\_loss' from a SQLite file named catchmentNutrients.db .

The groundwater nitrogen routing models can be run in several ways, such as from a stand-alone command-prompt (with "python run.py --timeword=steady"), and from DIMR.

This component was successfully tested with DeltaShell to receive recharge from LUCI, but dynamic nitrogen loading was not implemented.

### 4.10.4 Implementation in the Aparima case study

The Aparima model (described in Section 4.5.4) needed additional datasets to run with MT3D-USGS, such as effective porosity and initial nitrogen concentration. Denitrification was not enabled.



#### 4.10.5 Key findings and future recommendations

Similar to the dynamic groundwater model (Section 4.7.5), a key finding was that a series of separate transient MT3D models behave similar to a single transient MT3D model with multiple stress periods, which is important since it's not feasible to pause/update continuously running MODFLOW and MT3D-USGS simulations.

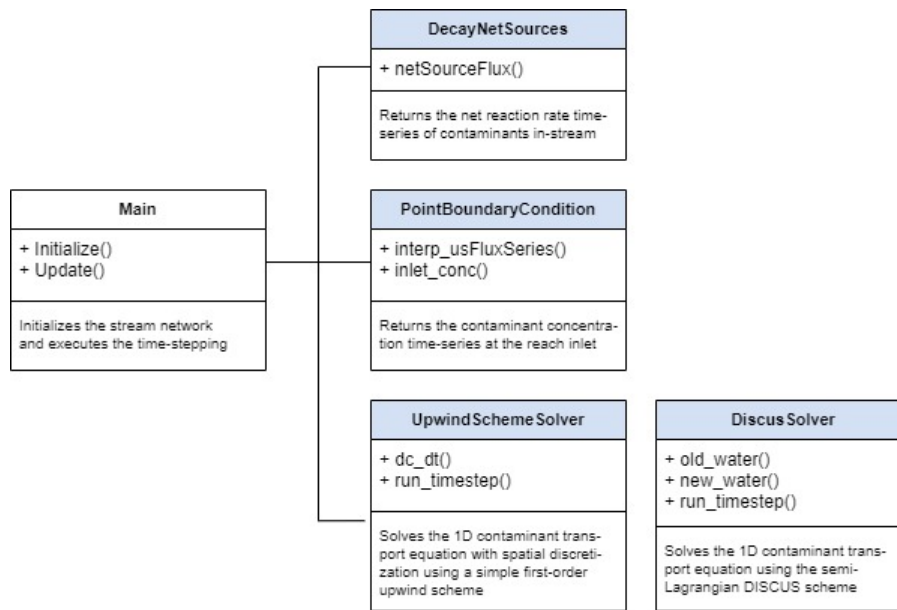
Recommendations for future development of the nitrogen routing component include investigating the effects of aquifer denitrification rates.

#### 4.11 Dynamic stream contaminant routing

The dynamic stream contaminant routing component models the transport of contaminants through the stream network on a daily basis. It is based on the one-dimensional contaminant transport equation, assuming transport in the stream is advection-dominated (such that dispersion may be ignored). Source and sink terms modelled include contaminant fluxes gained from surface water and groundwater flows, and lost from surface water takes (abstractions). Simple in-stream dynamics are also included, assuming these can be modelled as a first-order decay.

The model is run on a reach-by-reach basis in hydrological order. Source and sink terms are applied as point sources/sinks at the reach inlet. While concentrations are reported on a daily basis, the internal timestep of the model is shorter in order to satisfy the accuracy and stability requirements of the numerical method chosen. Similarly, concentrations are reported at the reach outlet, however the reach is sub-divided internally into a number of computational segments.

The model is coded in Python, using classes as shown in Figure 4-7. The class structure is intended to allow for sections of the code to be swapped out in future, e.g., to use different numerical methods or alternative representations for in-stream dynamics. At present, the model has been run using two different numerical methods. The first is a finite-difference method which uses a simple first-order upwind scheme to represent the spatial derivative and then employs the RK45 function from the scipy library to solve the resulting ordinary differential equation in time. The second is a semi-Lagrangian method based on the DISCUS scheme (see Manson and Wallis, 2000). These methods are interchangeable without requiring modification to the code.



**Figure 4-7: Class structure of dynamic stream contaminant routing model.** Shading indicates classes which are intended to be able to be swapped out, e.g., to use different numerical methods as in the two solver schemes. See text for further details.

The main module of the code is designed to comply with BMI standards and implements functions Initialize() and Update(). These functions are responsible for input/output and executing the time-stepping, as described in Table 4-2 below.

**Table 4-2: Summary of contaminant routing model operation via Initialize() and Update() functions.**

Function	Tasks	Requires	Outputs
Initialize()	Initializes the stream network ready for contaminant routing, including: <ul style="list-style-type: none"> <li>Reading and validating input data.</li> <li>Setting up computational locations.</li> <li>Initializing netcdf dataset for storage and output.</li> </ul>	<ul style="list-style-type: none"> <li>A configuration file including the model start and stop times and time-step size, in-stream decay rate and maximum computational segment length.</li> <li>One (or more) netcdf files describing the stream flow and velocity, as well as surface water and groundwater contaminant fluxes.</li> </ul>	None
Update()	Executes the time-stepping, including, on a reach-by-reach basis: <ul style="list-style-type: none"> <li>Initializing boundary condition and decay rate objects.</li> <li>Initializing solver object.</li> <li>Calling solver routine to calculate contaminant concentrations.</li> </ul>	Dependent on Initialize().	<ul style="list-style-type: none"> <li>Routed contaminant fluxes and concentrations at the outlet of each reach for each timestep, saved in a single netcdf file.</li> </ul>

Input data required by the model includes daily timeseries of stream flow and velocity for each reach, as well as the surface water and groundwater contaminant fluxes. These data are supplied by the component models in the “surface water – groundwater – stream flow” coupling framework in netCDF format. The exchange is currently assumed to occur via a single netCDF file containing all quantities, however this may change in future. Output from the model includes daily timeseries of contaminant concentrations and fluxes at the outlet of each reach, stored in a single netCDF file.

To-date, the model has been implemented only on a small test network using theoretical cases, and was not included in the Aparima case study due to time constraints. Sample netCDF output from the dynamic stream flow routing component has been used for model input, however there remains some uncertainty around data exchange, because for example, surface water and groundwater contaminant fluxes were not available at the time. Furthermore, although the model is BMI compliant, it requires an additional wrapper to be run using DIMR, which has not yet been implemented.

Further development of the dynamic stream contaminant routing component in the Interoperable Model framework would require the BMI-compliant wrapper to be completed to allow the model to be controlled using DIMR. It would also be advantageous to test the model’s validity for the Aparima case study catchment.

## 4.12 Economics and optimisation

In this section we describe the technical details of this component. For a detailed description of the application of this component for the Aparima case study area, including data sources, scenarios, and results, please refer to the companion report “Interoperable Modelling - Spatial Economic Optimisation”<sup>28</sup>.

### 4.12.1 Description of the nature and purpose of the component

The main purpose of this model component is the provision of spatial optimisation capabilities to conduct land-use scenario analyses – these will enable exploration of ecological, economic, and policy objectives. In addition, we have demonstrated use of this component for automatically generating an ecotopes layer from vector and raster input layers; this may be used as an alternate source of this input layer required for the steady-state and dynamic composite models created as part of this project.

### 4.12.2 Key aspects of component implementation

We selected the free and open source spatial modelling and optimisation framework LUMASS<sup>29</sup> as base software for this component. It provides spatial optimisation as well as raster-based processing functionality out of the box. In addition to the interactive visual modelling environment used for model development and data visualisation, it provides a command line-based application that engages the LUMASS engine to execute optimisation and process models in server or HPC environments. In this project we developed a BMI-compliant wrapper around the LUMASS engine that enables the integration of any model developed within the LUMASS modelling environment (process or optimisation) to be integrated into a BMI-compliant composite model. To test the BMI-compliant LUMASS engine, we developed a command line-based test application that utilises the BMI-wrapped engine to run i) a process model to create an ecotope layer from various input layers,

---

<sup>28</sup> Document saved on project github repository

<sup>29</sup> <https://bitbucket.org/landcareresearch/lumass>

and ii) spatial optimisation scenarios to explore the impact of N leaching and sediment yield reduction payments on economic and environmental indicators.

## Language

We used the C-bindings of the BMI interface provided by the [openearth github repository](https://github.com/openearth/bmi)<sup>30</sup> to wrap the LUMASS engine written in C++. The C-based BMI interface provides the advantage that it can be used directly by the Deltares Integrated Model Runner (DIMR) that we used as a coupling application to build and run the steady-state and dynamic composite models.

## BMI component setup

To setup a LUMASS model for use within a BMI-compliant composite model, the following files and information are required:

- LUMASS model file: This is either an XML-based LUMASS model file (\*.lmx) describing a process model (that may include spatial optimisation components) or a LUMASS Optimisation Settings file (\*.los) that describes (a) spatial optimisation scenario(s). Please refer to the LUMASS Optimisation HowTo for a detailed description of LUMASS Optimisation Settings files<sup>31</sup>
- BMI-compliant LUMASS engine: The BMI-wrapped LUMASS engine library, i.e., LumassBMI.dll on Windows or libLumassBMI.so on Linux.
- A YAML-based configuration file (\*.yaml): This file contains a general section (EngineConfig) with information required by the LUMASS engine library to process a specific model, e.g., the LUMASS installation directory and the path to the respective model file (i.e., \*.lmx or \*.los) and a model specific section (ModelConfig) with information on model specific settings. Please refer to the description on the project's github repository for further information:
  - *AparimaOptimisation*<sup>32</sup>
  - *MakeEcotopes*<sup>33</sup>

## Inputs and outputs

The inputs required and the outputs produced by a LUMASS model depend on the model type. Optimisation models require either a vector (\*.vtk) or raster (\*.img, \*.kea) layer with an associated attribute table or a standalone SQLite database (table). The (attribute) table stores for each spatial unit (e.g., ecotopes, parcels) the performance of a set of different land-uses (e.g., dairy, sheep and beef, forestry) with respect to a given set of criteria (e.g., net revenue, nitrate leaching, sediment yield). Additionally, an optimisation model requires information on the objective of an optimisation scenario (e.g., maximise revenue) and constraints (e.g., no dairy on slopes > 15 degrees). This type of information is supplied in the LUMASS optimisation settings file (\*.los). An optimisation model produces a set of outputs for each scenario. These include a map of allocated resources (land-use types) to the given parcels or ecotopes, and a series of summary statistics optionally subdivided into

---

<sup>30</sup> <https://github.com/openearth/bmi>

<sup>31</sup> [https://bitbucket.org/landcareresearch/lumass/downloads/OptimisationHowTo\\_1.2.zip](https://bitbucket.org/landcareresearch/lumass/downloads/OptimisationHowTo_1.2.zip)

<sup>32</sup> [https://github.com/niwa/interoperable\\_land\\_water\\_models/tree/master/Examples/BMI/LumassBMI/Optimisation](https://github.com/niwa/interoperable_land_water_models/tree/master/Examples/BMI/LumassBMI/Optimisation)

<sup>33</sup> [https://github.com/niwa/interoperable\\_land\\_water\\_models/tree/master/Examples/BMI/LumassBMI/Ecotopes](https://github.com/niwa/interoperable_land_water_models/tree/master/Examples/BMI/LumassBMI/Ecotopes)

spatial regions. A more detailed description of the required inputs and outputs of LUMASS optimisation models can be found in the LUMASS Optimisation HowTo.<sup>34</sup>

LUMASS process models typically operate on raster data (incl. associated attribute tables) and standalone tables (CSV, XLS, SQLite) and produce outputs of the same type. As input data for the sample MakeEcotopes model<sup>35</sup> applied for the Aparima catchment, we used a REC (shape file), soil type classification (shape file), groundwater model zones (raster layer), and a slope layer (raster). The output file the model produced was a raster layer (tiff) depicting the spatial distribution of all unique combinations of the input data and a SQLite database table containing the individual parameter combination for each of the given input data layers for each of the unique combinations identified.

#### 4.12.3 Final state of component development

The development of the BMI-compliant optimisation component is insofar complete as it fulfils the project's requirements to be able to incorporate the component into a BMI-compliant composite model to run optimisation-based scenario analyses for ecological, economic, and policy objectives. However, the implementation currently does not allow to incorporate LUMASS process models into dynamic composite models that require data exchange at the time-step level. The development of this functionality is currently being carried out in a different project.

#### 4.12.4 Key findings and future recommendations

We have successfully demonstrated the use of the BMI-compliant LUMASS engine for running steady-state optimisation-based scenario analyses for ecological, economic, and policy objectives and static process models for the generation of input datasets for the steady-state and dynamic composite models developed in this project. Because of scheduling issues, we were unfortunately not able to test the optimisation component as part of the steady-state or dynamic composite models developed.

Future development should be focused on i) testing the integration of the optimisation component into the steady-state and dynamic composite models, for example to analyse and design land-use transitions over longer time horizons, and ii) on the further development of its economic modelling capabilities as outlined in the conclusions section of the companion report "Interoperable Modelling - Spatial Economic Optimisation".

---

<sup>34</sup> [https://bitbucket.org/landcareresearch/lumass/downloads/OptimisationHowTo\\_1.2.zip](https://bitbucket.org/landcareresearch/lumass/downloads/OptimisationHowTo_1.2.zip)

<sup>35</sup> [https://github.com/niwa/interoperable\\_land\\_water\\_models/tree/master/Examples/BMI/LumassBMI/Ecotopes/MakeEcotopes](https://github.com/niwa/interoperable_land_water_models/tree/master/Examples/BMI/LumassBMI/Ecotopes/MakeEcotopes)

## 5 Model assemblies

### 5.1 Linkage diagrams

Model assemblies require several model components to be run in a co-ordinated way, with exchange of data. At the highest level, assemblies can be represented as simple linkage diagrams showing the main components and an indication of their inter-relationship. Such simple diagrams are shown for our static and dynamic model assemblies in Figure 2-1. However, there is considerable underlying complexity relating to the timing of running of different components and exchange of data which is not represented in the top-level linkage diagram. For coupling, there is a need for communicating the details of data exchange and timing. While there are standard approaches used in software development for describing model components and their interactions, such as Sequence Diagrams in the UML (Unified Model Language) system<sup>36</sup>, we did not find UML to be readily understandable by modellers or particularly well suited for describing interactions.

As an interim measure, before a more formal system is adopted or developed, we developed 'wiring diagrams' depicting model components, their interactions, and sequencing, for communication purposes. A conventional UML Sequence Diagram was also used to illustrate timing interactions when there were dynamic data exchanges on a time-step basis (rather than a simple model cascade). An example wiring diagram, prepared in draw.io software, is shown in Figure 5-1. The diagram depicts components with their data exchange items, system input and output files and exchange files, and sequencing (indicated by numbering). This somewhat ad-hoc method was suitable as a starting point, but would benefit from more formalism in the future.

---

<sup>36</sup> [https://en.wikipedia.org/wiki/Sequence\\_diagram](https://en.wikipedia.org/wiki/Sequence_diagram)

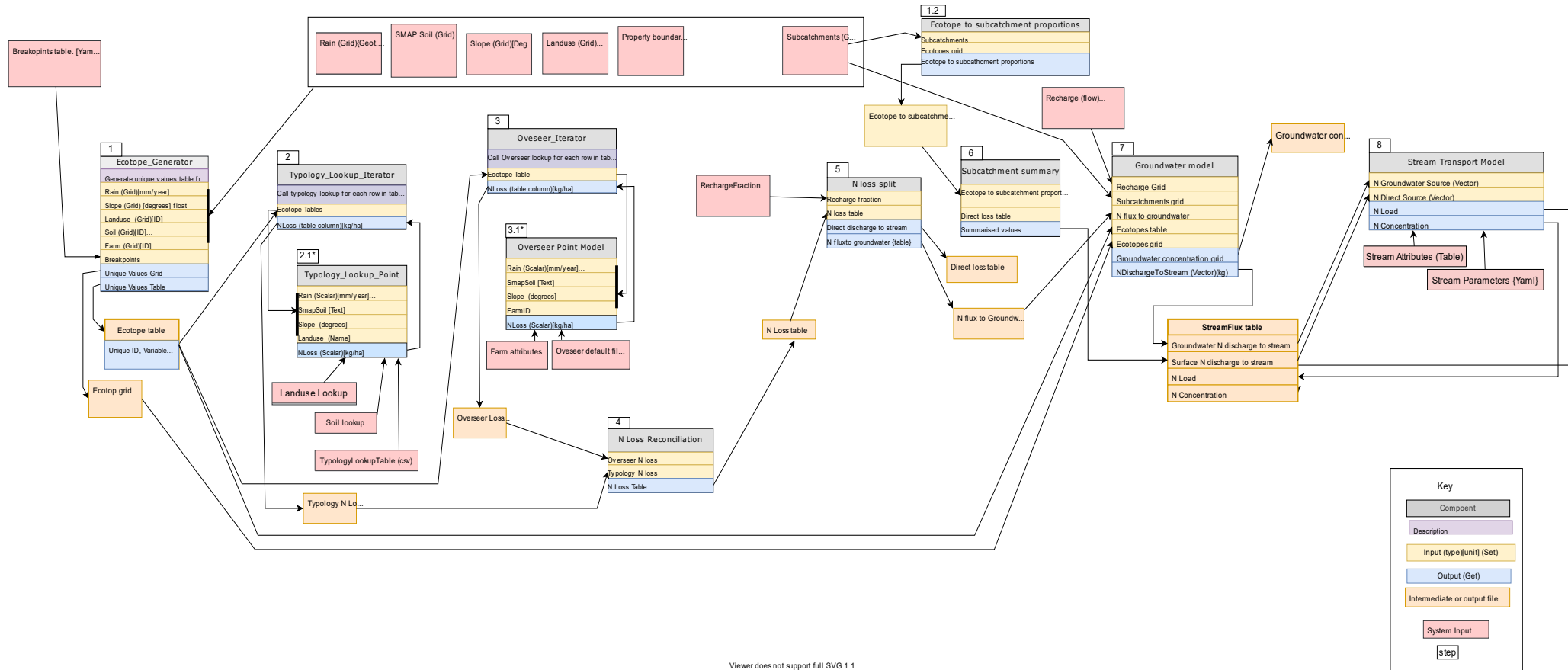


Figure 5-1: Example of a prototype 'wiring diagram' depicting model components and their interactions for the static nitrogen model.

## 5.2 Linkage of components

DIMR carries out the coordinated execution of the models as directed by a configuration file. In the simplified example below, the control section (yellow highlighting) defines the highest level of the combined model – a rainfall-runoff model and a routing model running in parallel. The next two sections (highlighted pink, then blue) link the models named in the control section to specific BMI-compliant library (DLL) files and their configuration files.

```
<control>
  <parallel>
    <startGroup>
      <time>0 1 99999999</time>
      <start name="TopRoute"/>
    </startGroup>
    <start name="Rainfall_Runoff"/>
  </parallel>
</control>

<component name="Rainfall_Runoff">
  <library>BMI_SMA</library>
  <workingDir>LUCI_RR</workingDir>
  <inputFile>Jan.2005.yaml</inputFile>
</component>

<component name="TopRoute">
  <library>bmi_toproute</library>
  <workingDir>toproute</workingDir>
  <inputFile>relative.yaml</inputFile>
</component>
```

In a typical DIMR configuration, another section would follow, providing links between variable names in the two models. In the file-based data sharing used for this project, this linkage is provided by naming the same input/output files in the separate configuration files used by each model, and to keep those names in sync when the configuration is changed.

If the file-based approach to data sharing is used in future work, it would be possible to streamline the sharing of data files between programs by naming them once in a shared meta-configuration file and having a small program that updates the individual models' configuration files to match.



## 6 Incorporation of models into the Delta Shell environment

### 6.1 Setting up and running models in the interactive environment

The web service for displaying pre-computed TopNet flows was set up as a C# Delta Shell plug-in as described in Section 4.1, with a facility to input a stream segment (as a property of the plug-in object) and trigger data retrieval from the web service.

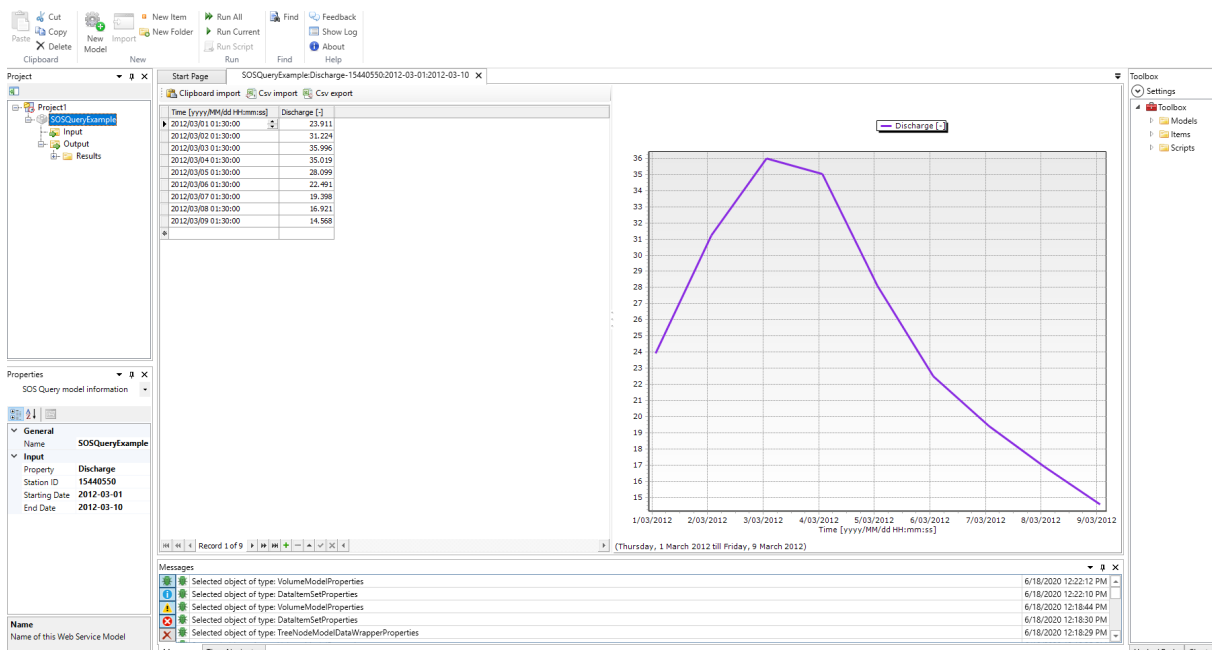
The method selected to run a sequence of BMI models within Delta Shell was to write scripts using Iron Python, a .NET implementation of the Python language that supports access to elements of the .NET Framework. Deltares provides a script editor and interface features to support Iron Python, so users can run scripts by clicking a button on the Delta Shell UI. Iron Python scripting also allows Delta Shell users to control elements of the user interface to create custom features and actions without having to compile plugins in C#.

Deltares' Open Earth software project has developed a C# class called "BasicModelInterfaceLibrary" which acts as a translator between the BMI's control and data-passing functions and Delta Shell's own .NET interface for model operations. An object that implements Delta Shell's ModelBase class, for instance, must provide a method named "Finish()" which is called by the DeltaShell framework to finish a model's computations. The BasicModelInterfaceLibrary class implements the Finish() method by calling the finalize() method on the BMI-compliant library that it wraps. Since DIMR itself is BMI compliant, it can be wrapped in this class, and then accessed by Delta Shell as if it were a one of its own models. This can be done with compiled C# code in plugins, or more simply in Iron Python scripts.

To run the models for the Aparima test case, an Iron Python script was written to create an instance of BasicModelInterfaceLibrary wrapped around the DIMR library. The script then loaded the DIMR configuration holding the Aparima model sequence into DIMR and initialized it. For the steady-state model sequence, the update function was called once to run the model sequence. For the dynamic sequence, a loop called the update function repeatedly to advance from the start time to the end time. Each model wrote its results to files that were named in their configuration files, and could also be located by the DeltaShell framework, as described in the following section. With this combination of model execution and results presentation, the interoperable model sequence was embedded in Delta Shell.

### 6.2 Visualisation of results

Delta Shell has facilities for displaying input and output data, with emphasis on map-based presentation and time-series. In the 'native' operation of Delta Shell, model components and data are a fundamental part of the .NET object structure and can be accessed through a project. The web service plug-in for retrieving pre-computed TopNet flows was set up in that way, as described in Section 4.1. An example of a prototype data visualisation is shown in Figure 6-1.



**Figure 6-1: Example of prototype of data visualisation of flow time-series retrieved through a web service and displayed within Delta Shell.**

In general, however, we did not use the Delta Shell object model for setting up model components and linkages, but used BMI and DIMR. We still used the data visualisation capabilities of Delta Shell to display model results, though. This was achieved by preparing Python scripts to import model result files into formats that can be displayed natively in Delta Shell through a callable module. Once this was done, the Delta Shell interactive facilities for displaying maps, spatial animations, and time series could be used. Example screenshots of the displays are shown in Figure 6-2 and Figure 6-3.

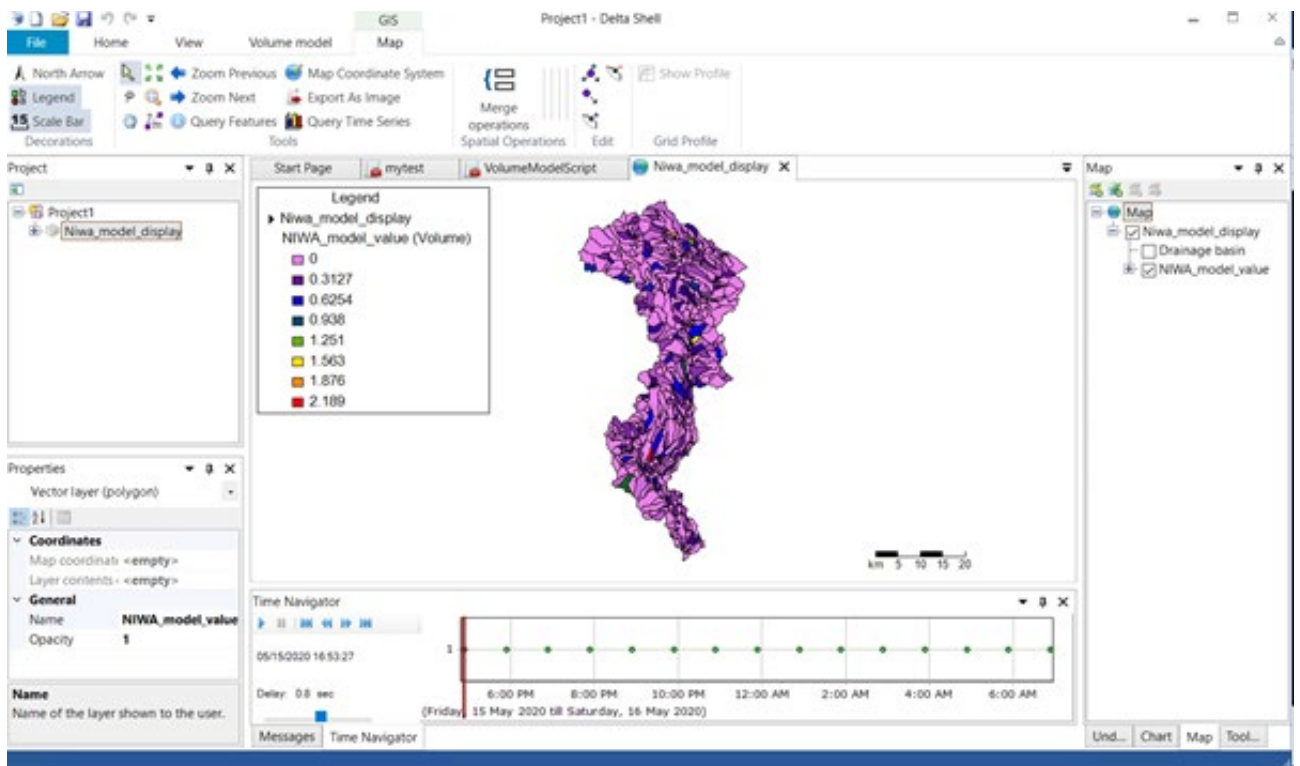
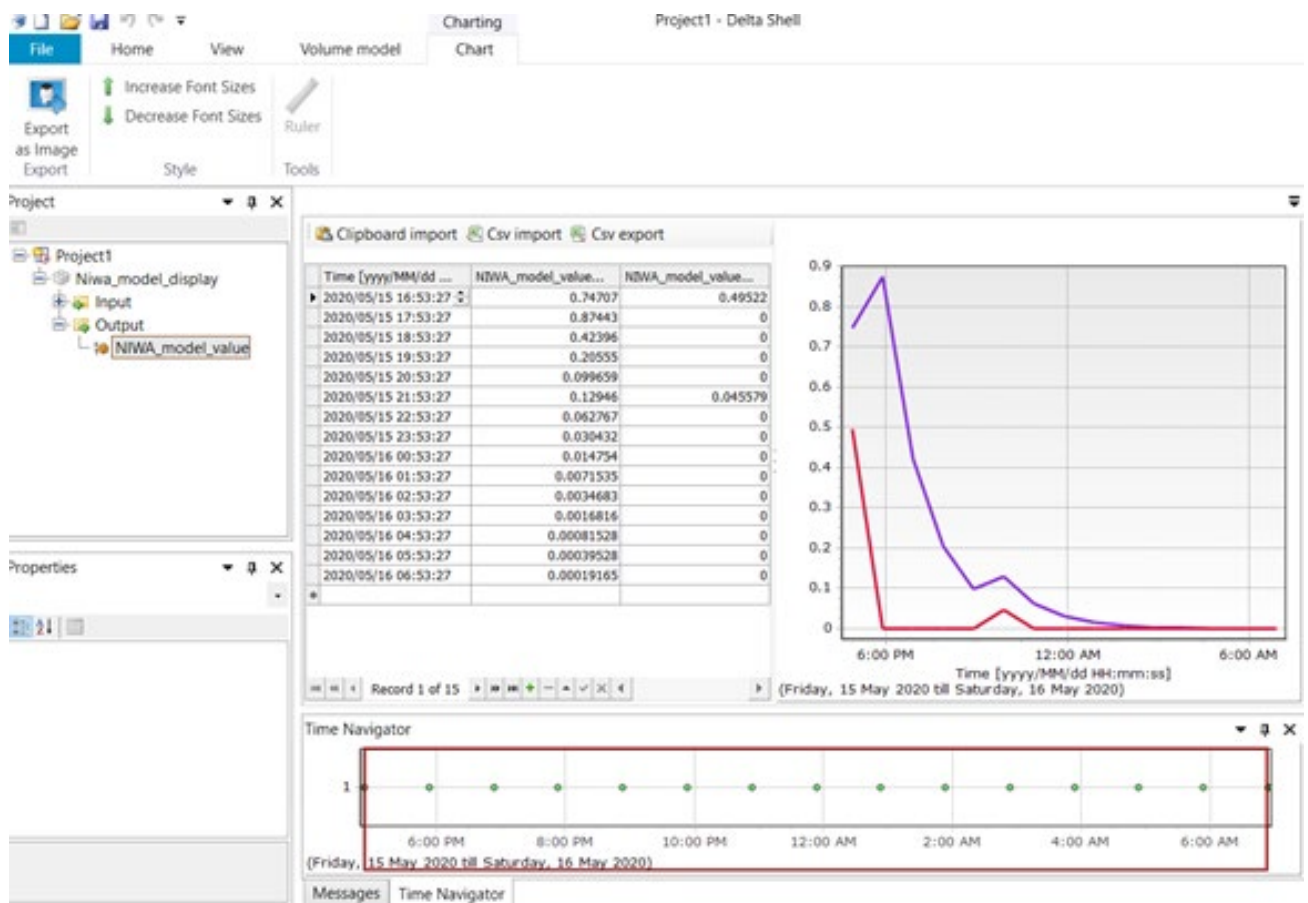


Figure 6-2: Example of map display of flow results for dynamic hydrologic model for the Aparima catchment.



**Figure 6-3:** Example of display of flow time series from the dynamic hydrologic model for the Aparima catchment.

### Testing, deployment, logging

We did not conduct formal testing, automated build/deployment, and logging in this project. Individual model components were set up so that they could be run with known inputs and outputs as BMI components, but more sophisticated testing was not set up, and was assumed to be the responsibility of individual model component developers. We did set up scripts to ensure that required system libraries were set up on the host computer, but did not provide further automation (for example, retrieving and linking required libraries). Also, we did not require components to provide their own logging, or provide system-level logging of running of model assemblies (for example, whether components completed their initialisation method successfully or not). Hence testing of the system relied on the developer having a good control, and ability to debug software. Formalised testing, deployment and logging would require considerable extra resources, which would be appropriate when developing future, fully operational systems.

## 7 Model sharing and documentation

Since all products of the interoperable models programme are to be open source and freely accessible, the participants agreed to use GitHub as a storage site for the code that they developed. The repository is presently in a private NIWA repository and will be moved to a public repository as the programme concludes. In addition to the model and library source code, the repository holds examples and documentation, mostly in the form of markdown (.md) files. The developers also used the Wiki feature in the repository to capture notes and discussions among the developers that aren't represented in the source code.

## 8 Intellectual property, governance and co-ordination

### 8.1 Approaches used in the project

Within the project, management of IP was through funding contracts and ancillary agreements. For example, all new code developed under the project was required to be publicly available, and some limits to the use of soil data in the Aparima catchment were established. There were not specific formal IP agreements related to classes of open-source access or for protection of third-party code that might usually be considered for open-source software development projects. It was considered that the project was a research project, and that provision of more formal arrangements would hinder the research, and it was anticipated in the primary contract that more formal arrangements would be agreed later in the programme through the programme Governance Group.

The proprietary Overseer model required special consideration. Access to the Science version of Overseer was proposed by Overseer Limited, but subject to conditions that NIWA could not accept. AgResearch, who were responsible for provision of the Overseer BMI component to the project, arranged separate contracting with Overseer Limited for access to the Overseer engine, although as noted earlier, this component was not completed.

The project plan entailed a Governance Group to steer the project, assist with IP considerations, and to assist with arranging further resourcing. This group became practically inoperative in Stage 2, despite attempts to activate it. Reasons for this may include delays in commencing the project while contract conditions were negotiated, and slower than anticipated delivery of project outputs; the group may have tacitly decided to delay their input until they could assess the progress of the technical work. There still remains an important need for such a group if future phases of the interoperable modelling initiative are to be commissioned, and it is anticipated that a meeting of stakeholders will occur following completion of stage 2 of the programme.

The technical group was organised with a technical lead, who called regular video-conference meetings (on a weekly basis during the middle stages of the project), along with separate meetings or workshops to consider particular technical issues or to share information or techniques (for example, naming conventions). Initially, a detailed Gantt chart accompanied by an issues tracker and task tracker was used in the process. While this was useful in setting the initial workflow and goals, it was not updated regularly by team members or the project manager, and was not used in later stages. The weekly meetings were used to check and share progress, to identify difficulties and follow-up activities offline. A Microsoft OneDrive shared drive was maintained for the technical group, and the GitHub repository and wiki was available to all team members, with write access to all the technical group. Co-ordination was managed by the technical lead initially, but later a separate project manager was introduced to deal with contracts and some aspects of progress tracking, to relieve workload and create a clearer separation of responsibilities for the technical lead. In the future, it is considered that a separate project manager would be helpful from the initial stage of the project due to the complexity and number of parties, without influencing technical decisions or leadership. Other challenges included personnel changes in the technical group due to staff leaving the host organisation, and conduct of economic optimisation activities somewhat separately, due to dependence on biophysical model components that could not be delivered early enough for tighter integration. Overall, the technical group built up a high degree of collaboration and co-ownership, which has helped build a community of modellers engaged in integrative land-water modelling.

Liaison with Deltares was undertaken on an ad-hoc basis, mainly by the lead developer and also through DairyNZ, to obtain advice and to address occasional technical issues. In addition, the lead developer visited Deltares early in Stage 2 to meet key staff and to undertake training based on task-based guided work.

A lead developer was used over a 2-year period to assist with system design, setting up model components in the interoperable model system, model coupling, and visualisation. This required detailed knowledge of both the science domains and software development, which is very demanding for a single person. For some of the model components, specialist software developers were available to work alongside science model developers, which was useful. In future, it would be appropriate to increase the role of a specialist software developer for integration aspects.

## 8.2 Some future governance and management needs

To summarise future needs in relation to governance and management:

- A Governance Group would need to be refreshed to provide the necessary direction, advice on IP, co-ordination of provision of funding, and make decisions regarding future stewardship, maintenance and hosting of the software and data.
- While model components developed in the project have been made available on a free and open source basis, model applications rely on datasets which are not freely available, such as the VCSN and soils data. This needs to be addressed for a future interoperable modelling system. Also, we anticipate that some existing proprietary model components such as Overseer may be introduced in the future, and a mechanism to use such models needs to be established.
- A project manager, separate from the technical lead, would help manage such activities as progress tracking, reporting, subcontracting and finances.
- In the future, greater use should be made of specialist software developer and computer science input, to complement the integrative skills and knowledge of science specialists.

## 9 Recommended future technical approach to interoperability

This project adopted the Delta Shell framework for component coupling. We determined that Delta Shell's complex native interface was not appropriate for the wide range of model components and providers, due to the complexity, and the requirement to commit to a desktop basis and to the Delta Shell object model. Instead, we adopted a simpler and more standard coupling standard, BMI, which can be used within Delta Shell. Despite this, there were limitations which have caused us to recommend alternative frameworks:

- The DIMR software for setting up assemblies was limited with respect to the types of data that could be exchanged, and the full BMI specification was not implemented, which resulted in limitations and technical difficulties (managing timesteps, for example).
- Most of the models had to be brought to the same location and within a Windows computing platform, which is somewhat restrictive for models that natively run in a different environment.
- Due to limitations in the BMI specification and its implementation, a file-based approach was used, which meant that potential benefits of running a model on a single machine with memory-based data exchange were not realised.
- The Delta Shell visual display capabilities, while having some key features of interest, were sometimes limited (e.g., limited symbology on maps).
- Model components sometimes required multiple layers of wrapping to get them to work in the framework.
- Considerable programming effort was required to set up the model assemblies within the framework.
- Direct users of the model are required to download and install Delta Shell, and possibly other non-core libraries associated with model components. This may be a barrier to users and widespread model uptake.
- Model components are generally required to be available to install on the user's computer, which could cause difficulties if some model components are proprietary.
- Updating of model components needs to be managed centrally.

Considering these limitations, the project technical team consider that a web services approach offers advantages. In that approach, model components can be set up as web services that receive data, run, and provide data in response to standard requests using standard web protocols. This is known as a Models-as-a-Service approach (MaaS), which has been adopted in several recent hydrologic interoperability systems (Jiang et al. 2017; Nelson et al. 2019; Gan et al. 2020). This overcomes many of the limitations of DeltaShell. For example, model components could be maintained on a host server by a single organisation, and modified, as long as they respond to the agreed interface. Our approach in the current project of using fairly simple initialise-step-finalise aspects of BMI, along with file-based exchange has positioned the components well for implementation in such a framework. A web service approach has the following advantages:

- Model components can be maintained locally by different organisations.



- The model components only need to implement a small set of actions.
- Model assemblies can be accessed through a web browser.
- Standard libraries for displaying data can be used within the web browser.
- Difficulties with language incompatibilities will be avoided because model components can be run locally and only need to be presented as a web service rather than being presented to a common, unifying framework language, such as Delta Shell.
- There is the potential for models to run on high-performance computers or distributed systems.
- The concept is consistent with a trend towards web-based delivery of environmental data, both static and live.

Associated with this is the trend to running models in 'containers'. This has become a widely used way to set up required software in an isolated part of the host computer without having to install a full operating system, providing good control over the software environment and data exchanges.

Such an approach still requires a central component to orchestrate running of services in an assembly, and programmer resources to write the relevant code to set up the assembly. Also, systems for orchestrating assemblies of model components are relatively immature, so close attention would need to be paid to choice of the orchestration software. In Stage 1 of the project we identified CSIP as a candidate orchestration software, but other systems may have emerged in the interim.

A potential pitfall with this approach is that some models may require large datasets to be exchanged, which would be difficult if the data needs to be exchanged over the internet. Similarly, if model components need to exchange data very frequently, then lags associated with internet data may slow running of the model assembly. In those cases, the cloud-based architecture proposed by Krämer and Senner (2015) and Krämer (2018) might be more appropriate. It deploys microservices encapsulated in containers and realises data exchange through a distributed filesystem established across compute nodes. Orchestration of services is realised by a Domain Specific Language. However, a combination with the BMI-based web-service approach described by Jiang et al. (2017) could be a viable option to explore in a future project.

We also found that model assemblies can become complicated. Our ad-hoc method of describing data source, exchange items and model run order is not standard, missed aspects such as defining units of variables, and does not lead directly to model configuration files. It would be desirable to adopt or introduce more formal, standardised ways for illustrating and describing the components and their linkages in the future.

## 10 Summary of key results and findings

The project team succeeded in implementing a set of coupled models within an interoperable modelling framework.

Eleven model components covering water quantity, quality, production and economics were able to set up within a framework (Delta Shell) using established standards for model interfaces (BMI), variable naming according to CSDMS conventions, and various standard file formats. We also attempted to implement Overseer, but could not fully achieve this, partly due to a dependency on an organisation external to the project team.

Two sets of model components were successfully combined into assemblies to address a) steady state contaminant transport and b) dynamic coupled flow calculation. The catchment models resolved to the level of property, soil, climate class, and topographic class variation, using a common spatial framework of 'ecotopes'.

Additionally, models for production, economics, and environmental losses were set up using BMI in conjunction with the LUMASS optimisation engine.

As a separate exercise, we demonstrated how pre-computed flow time-series that are stored in an external server could be imported and displayed within Delta Shell using SOS standards and a DeltsShell plug-in accessing data provided through that standard.

We decided that it was preferable to use BMI instead of Delta Shell's native interface, to allow greater flexibility and component re-use. However, we found that some key aspects of BMI were not fully implemented in the Deltares model runner/coupler DIMR. Some simplifications and work-arounds in our coupling approach were therefore required.

We found that file-based data exchange overcame limitations of BMI's interface for memory-based data exchange (for example, BMI does not allow for tabular data to be exchanged or implemented). The wide and multi-language availability of libraries for reading and manipulating the chosen standard file formats assisted with importing and exporting data from model components. File-based exchange could potentially become a problem with closely-coupled models that need to exchange data frequently; in those alternative scenarios, memory-based exchange may need to be implemented.

The models were set up for a trial catchment, the Aparima, and assemblies were run successfully within the DeltaShell environment.

It was possible to display model output within the Delta Shell environment using the visualisation libraries and user interface native to Delta Shell, although some additional code was required to import model outputs into the DeltaShell environment.

All the model components, apart from Overseer, have been provided as free and open-source components available on a data repository.

Despite these successes, the project team recommends that an alternative coupling approach based on running model components be trialled. Such a system has several advantages over a Windows desktop approach necessitated by Delta Shell. There is still a need for framework software, and developer expertise, to set up and orchestrate model assemblies.

We used an ad-hoc 'wiring diagram' approach to defining linkages, data exchanges and timing. It would be desirable to develop more formalised methods in the future, although standard URL diagrams are not particularly well-suited to this purpose.

The project technical team developed good structured working relationships with ongoing collaborative and collegial interactions. It was also desirable to introduce a project manager to facilitate administrative task and performance monitoring. The governance group became inactive during the project, an aspect that would need to be improved in future work of this nature. It is also important to use a software developer or group of developers with good computer science knowledge, alongside model specialists.

## 11 Acknowledgements

We would like to thank Environment Southland, especially Lawrence Kees, for providing datasets and advice for setting up the Aparima case study, and for engaging in model design workshops.

We also wish to thank Deltares for providing beta versions of the Delta Shell code, for hosting Tom Evans during his visit to the Netherlands, and for providing advice and troubleshooting.

Overseer Ltd kindly engaged with the project team to advance the provision of an Overseer web service to an interoperable model system.

This project was funded by the Ministry for Business Innovation and Employment through the Our Land and Water Science Challenge, with co-funding from NIWA, AgResearch, DairyNZ, GNS Science, Manaaki Whenua/Landcare Research, SCION, Victoria University of Wellington, and Environment Canterbury.

## 12 References

- Bakker, M., Post, V., Langevin, C.D., Hughes, J.D., White, J., Starn, J., Fienen, M.N. (2016) Scripting MODFLOW model development using Python and FloPy. *Groundwater*, 54(5): 733-739.
- Bedekar, V., Morway, E.D., Langevin, C.D., Tonkin, M.J. (2016) MT3D-USGS version 1: A US Geological Survey release of MT3DMS updated with new and expanded transport capabilities for use with MODFLOW: 2328-7055, US Geological Survey.
- Clark, M.P., Rupp, D.E., Woods, R.A., Zheng, X., Ibbitt, R.P., Slater, A.G., Schmidt, J., Uddstrom, M.J. (2008) Hydrological data assimilation with the ensemble Kalman filter: Use of streamflow observations to update states in a distributed hydrological model. *Advances in Water Resources*, 31(10): 1309-1324.
- Deltares (2020) Delta Shell User Manual. Version: 1.2. Draft. Deltares.
- Donchyts, G., Baart, F., van Dam, A., de Goede, E., Icke, J., van Putten, H. *Next Generation Hydro Software*. 11th International Conference on Hydrodynamics, New York City, USA. 10.13140/2.1.5088.4480
- Elliott, A.H., Semadeni-Davies, A.F., Shankar, U., Zeldis, J.R., Wheeler, D.M., Plew, D.R., Rys, G.J., Harris, S.R. (2016) A national-scale GIS-based system for modelling impacts of land use on water quality. *Environmental Modelling & Software*, 86: 131-144. <http://dx.doi.org/10.1016/j.envsoft.2016.09.011>
- Elliott, S., Callachan, S., Conland, N., Daughney, C., Eikaas, H., Evers, D., Herzig, A., Jackson, B., Monge, J., Johnstone, P., Shamseldin, A., Sharp, J., Soliman, T., Turek, G., Vogler, I., Wakelin, S. (2017) Interoperable Models for Land and Water Framework selection and preliminary design. Prepared for Our Land and Water. *NIWA Client Report* No: 2017239HN, National Institute of Freshwater and Atmospheric Research.
- Gan, T., Tarboton, D.G., Dash, P., Gichamo, T.Z., Horsburgh, J.S. (2020) Integrating hydrologic modeling web services with online data sharing to prepare, store, and execute hydrologic models. *Environmental Modelling & Software*: 104731.
- Goring, D.G. Kinematic shocks and monoclinal waves in the Waimakariri, a steep, braided, gravel-bed river. *Proceedings of the International Symposium on waves: Physical and numerical modelling*, University of British Columbia, Vancouver, Canada.
- Jiang, P., Elag, M., Kumar, P., Peckham, S.D., Marini, L., Rui, L. (2017) A service-oriented architecture for coupling web service models using the Basic Model Interface (BMI). *Environmental Modelling & Software*, 92: 107-118.
- Krämer, M. (2018) A microservice architecture for the processing of large geospatial data in the Cloud. *Technische Universität*.
- Krämer, M., Senner, I. (2015) A modular software architecture for processing of big geospatial data in the cloud. *Computers & Graphics*, 49: 69-81.

- Nelson, J., Souffront, M.A., Shakya, K., Edwards, C., Roberts, W., Krewson, C., Ames, D.P., Jones, N.L. (2019) Hydrologic Modeling as a Service (HMaaS): A New Approach to Address Hydroinformatic Challenges in Developing Countries. *Frontiers in Environmental Science*, 7: 158.
- Niswonger, R.G., Panday, S., Ibaraki, M. (2011) MODFLOW-NWT, a Newton formulation for MODFLOW-2005. *US Geological Survey Techniques and Methods*, 6(A37): 44.
- Rawlinson, Z.J., Toews, M.W., Daughney, C.D., Zammit, C., Kees, L., Moreau, M., Rissman, C. Development of a regional steady-state groundwater flow model loosely-coupled to surface water for the Southland Region. *NZ Hydrological Society Annual Conference*, Hamilton, 1-4 Dec 2015.
- Turnbull, I.M., Allibone, A.H. (2003) Geology of the Murihiku area. *Institute of Geological & Nuclear Sciences*, Lower Hutt.