Client:
Environment Agency, UK

National Groundwater Modelling System

Phase 2 - Detailed
(Change Control Note 2005/03)

Architecture Design Document
Version 1.0 (final product Phase 2)

01 February 2006

WL | delft hydraulics
Tessella

CLIENT:	Environment Agency UK
TITLE:	National Groundwater Modelling System (NFFS CCN 2005/03) Phase 2 – Detailed architectural design Architecture Design Document
REFERENCES:	EA Contract Reference Number 11918-1-8 EA Order Number 30056390 dated 21/06/2005

VER.	ORIGINATOR		DATE	REMARKS	REVIEW		APPROVED BY	
0.1	Gijsbers		July 2005	draft				
0.2	Godfree		September 2005	draft				
0.3	Gijsbers		October 2005	draft	Godfree			
0.4	Gijsbers		October 2005	draft				
0.5	Godfree		October 2005	draft	Gijsbers			
0.6	Gijsbers		October 2005	draft				
1.0	Gijsbers		1 February 2006					

PROJECT IDENTIFICATION:		
KEYWORDS:	NFFS, National Groundwater Modelling System, IT architecture	
NUMBER OF PAGES	67	
CONFIDENTIAL:	☒ YES, until (date)	☐ NO
STATUS:	☐ PRELIMINARY ☐ DRAFT	☒ FINAL

Preface

Groundwater Models have been in use by the Agency and its predecessors for many years and these have been initiated and implemented at Regional level. These models were not linked in a national context. Groundwater models are also costly and have not been easily linked to the needs of operational customers. The fragmented approach to modelling was highlighted by the development of source protection zones in the early 1990's. It was recognised that a centrally co-ordinated, strategic approach was required to avoid duplication and reduce the risk of challenge.

To address this situation, a Strategic Review of Groundwater Modelling (R&D Project W6-034, R&D Technical Report W214; Brown and Hulme, 2001) was undertaken. The main output of the Strategic Review was the *Environment Agency Framework for Groundwater Resources Conceptual and Numerical Modelling* (R&D Technical Report W214) which contained a nationally consistent technical approach and programme for regional groundwater resources assessment and modelling.

The Head Office Hydrogeology Team presented a summary *Implementation Plan* for this work jointly with the Science Group which was accepted by the national Water Resource Management Team (WRMT) in October 2004. Regional modelling strategies were also recognised by WRMT as strategic Water Resources capital programmes. This work therefore supports the Streamlining Abstraction Processes (SAP) and Restoring Sustainable Abstraction (RSA) programmes managed by the national Water Resources Regulation team.

The Head Office Hydrogeology team is now developing a more detailed *Implementation Strategy* comprising a series of measures to support groundwater modelling, ensuring appropriate national consistency, improving efficiency and accessibility by customers. The Implementation Strategy will address concerns like national planning of model development, benefit realization, succession planning, business efficiency, IS performance, and customer accessibility to models.
An *IT Strategy for Groundwater Resource Assessment and Modelling* is being prepared to address the infrastructure and IS performance issues.

The National Flood Forecasting Projects (NFFS) is currently implementing an IT architecture for a centrally hosted flood forecasting system for the Environment Agency (EA). It is recognised that there are strong links between the proposed IT Strategy for Groundwater Modelling and the NFFS. A feasibility study has been conducted which concluded that the IT-backbone, named National Groundwater Modelling System (NGMS), can be based on the NFFS architecture (and software components) if some minor modifications and extensions are implemented. This outcome was the starting point of phase 2, the detailed architectural design. In this phase the required modifications and extensions need to be made explicit.

Phase 2 of the NGMS will produce the following documents…
1. Update of phase 1 User Requirements Document (URD)
2. Update of phase 1 Software Requirements Document (SRD)
3. Architecture Design Document (ADD)
4. User Interface Specification Document (UISD)

5. Interface Definition Document (IDD)
6. Hardware and Infrastructure Design document (HID)
7. Update of Project Implementation Plan (PIP)

The *Architecture Design* is presented in this document.

Guide to the reader

The document lying before you provides the high-level architectural software design for the *National Groundwater Modelling system*. This design document is intended to provide an answer to the following question: *"How will the required functionality, defined in the User and Software Requirement documents, be provided?"*

The level of detail that is contained within this document is limited to providing the reader an overall picture of the architecture and operation, to enable an understanding of the proposed implementation of the system.

Chapter 1 sets the context in which the Groundwater modelling system will reside; it describes the key external components that have an influence (both data and control) on the proposed system.

Chapter 2 introduces the logical model of the NGMS. It describes the main building blocks and their relationships with other aspects of the system.

Chapter 3 describes the NGMS implementation of the NFFS architecture from a high level point of view. Further details are contained within a separate infrastructure design document.

Chapter 4 details the key components of the NGMS system and how the differ from the NFFS versions. It includes detailed descriptions of the operation of the components. Information on the User Interface, including potential screen layouts etc. is contained within a separate User Interface Design Document.

Chapter 5 provides descriptions of how the individual components in the system interact to perform specific scenarios, such as "Adding an abstraction". They are intended to illustrate how the key Use Cases (described in the requirements documentation) will be realised within the NGMS.

New in this document

V.0.1 Is the initial version of the document.
V.0.2: Added system operation and design scenarios, extended component description
V.0.3 Reorder document structure, added workflows, improved component description
V.0.4 Added data slicing details
V.0.5 Updated central system aspects
V.0.6 Updated Module execution control and data slicing details
V.1.0 Finalized report for release

Contents

1 Introduction

1.1 Introduction

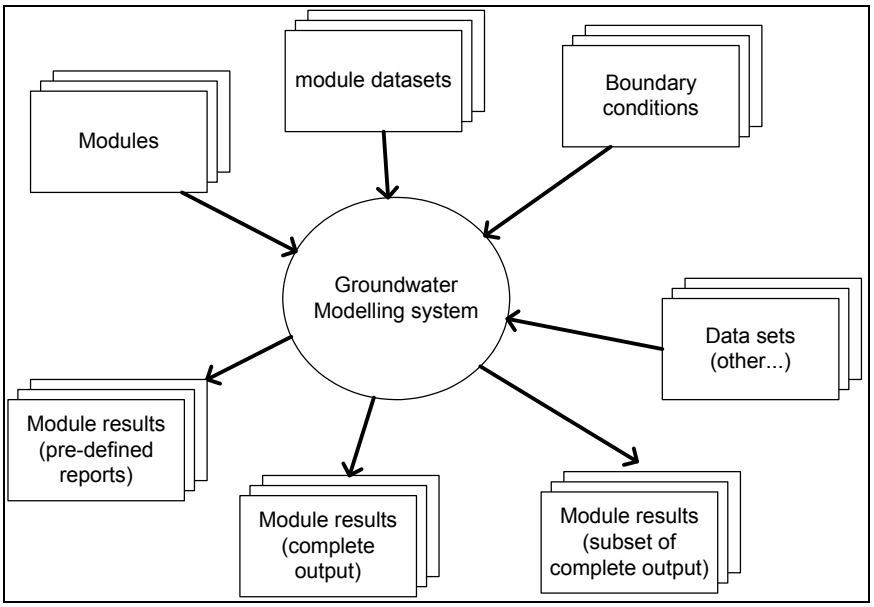
Current Groundwater Modelling systems used with the Environment agency are operated and maintained on a number of standalone servers. This means that models - and the associated input and output data - are hard to access by others than an individual modeller and that the effort required for performing a module run is large.

A centralised system has a number of key benefits:

- Ability to execute model runs efficiently and with minimal system knowledge.
- Change management controls on modules, module data sets, output etc.
- Wider access to module data (input and output)
- Increased knowledge sharing

1.2 System Context

The diagram below shows the National Groundwater Modelling System (NGMS) in the centre and all the information and control flows across the system boundary. It illustrates external systems or components that are likely to have an interaction with a Groundwater Modelling system and need to be considered when proposing a solution.



The external entities are described below:

- **Module**
  These are executable(s) that encapsulate the underlying physics of Groundwater modelling (for example Modflow96).

- **Module datasets**
  Module datasets are the parameterisation of the physical aspects of the groundwater system (aquifers etc.) and the system parameters. These parameters are derived from, among other data, geological maps etc.

- **Boundary conditions**
  The criteria that describe the situation being modelled. These include, for example, precipitation and evaporation for the recharge modules, and recharge and abstractions for the groundwater models.

- **Module results (complete output)**
  The output file created when executing the Module (e.g. Modflow). These files consist of a number of data values per model cell and per time step. The size of these output files can be quite considerable, in some cases reaching several GBs.

- **Module results (subset of complete output)**
  A subset of the data contained within the Module output file. These subsets are typically created via bespoke applications, which extract the appropriate data from the complete results data file.

- **Module results (pre-defined reports)**
  A report (graphs and tables) in a pre-defined format. These will be generated using the output from a Module run.

1.3 General constraints and assumptions

In assessing the user requirements for a Groundwater Modelling system, a number of constraints and assumptions have been noted. These have an impact on the type of functionality that can be provided by a system and the suitability for potential system architectures.

The assumptions and constraints identified at this stage are listed below:

- The Module input and output files can be large.
  They can range from ~Mb's up to a number of GB's. Initial analysis has shown that they do compress (using Winzip) quite considerably (i.e. by factor 5-10) but the resulting files are still of the order 10-100MBs.

- Only subsets of the module output files are ever used in analysis and reporting.
  The full output from a module run is rarely analysed in its entirety, usually slices are taken on a spatial or temporal basis for further investigation.

- Module run times can exceed 12 hours.
  This is not expected to be an issue as this is the expected duration for most large scale models.

- The scope of the system is to provide an environment to execute pre-configure Groundwater modules, not to provide a complete module development environment.

- As well as a centralised system, there is considerable benefit in having a version of the system that can operate in "stand-alone" mode. This could prove useful in enabling model development, testing and preparation of updates (data sets and configurations), demonstrations of models and output to third parties etc.

- Pilot user numbers are expected to be ~10-15

- Module dataset sizes range from 10MB to 20GB

1.4 Recommendation by the feasibility study

The feasibility study concluded with the recommendation to take the NFFS architecture as a starting point and develop a (minimal set of) additional functionality to provide the flexibility as requested by the groundwater modellers of EA.

1.5 Additional functionality needs

The formal requirements of the feasibility study have been updated at the start of the second phase (see v.2.0 of the User Requirements Document and the Software Requirements document). Compared to the existing NFFS capabilities, the request for additional functionality can be summarized (in 'free wording') as:

- ability to identify a suitable module and module data set
  - visualize module input data sets
  - provide model logbook
- ability to configure abstraction locations that can be served by the module data set, but are not yet incorporated in the system configuration
- ability to manage output and sub-sets
  - slice sub-sets of data from the bulky data output
  - provide statistical and mathematical post-processing functionality to condense results into groundwater specific data
- ability to provide output that meets the demand from the groundwater community
  - specific graphic views such as accretion curves, contour plots, splodge plots, winterbourne signatures, groundwater unit budgets
  - availability in web-compatible format for publication on webserver
  - data available for download in Excel and shape format
- ability to inspect and interrupt job execution
  - inspect module output and diagnostics while computation job is running
  - manual and automated interruption procedures
- restricted data access

  - distinction between common (public) and user defined (private data)
  - restricted access to confidential information
- management of network traffic and disk space
- ability to handle various versions of a basically similar core module code
- ability to trace module runs

While the above requirements reflect the wishes from NGMS, the most strict boundary condition is the fact that the development the NGMS should not threaten the installed based of NFFS or DelftFEWS systems.

Hence, care should be taken with module extensions as software adaptations to NFFS modules require strict testing as well as configuration updates for the installed systems. If an existing package does meet not the desired functionality to a large extend (e.g. 95%), development of a new module, API or plug-in should be considered.

2 High level solution architecture

The basis of the NGMS solution architecture will be the NFFS architecture with some small extensions.

2.1 Logical architecture

This chapter is extended based on the Logical Software Design as developed in the feasibility study.

2.1.1 NGMS Logical Model

The high-level user requirements (use cases) have been analysed and as a result, a logical model of the system components has been produced. This is illustrated in Figure 2-1.

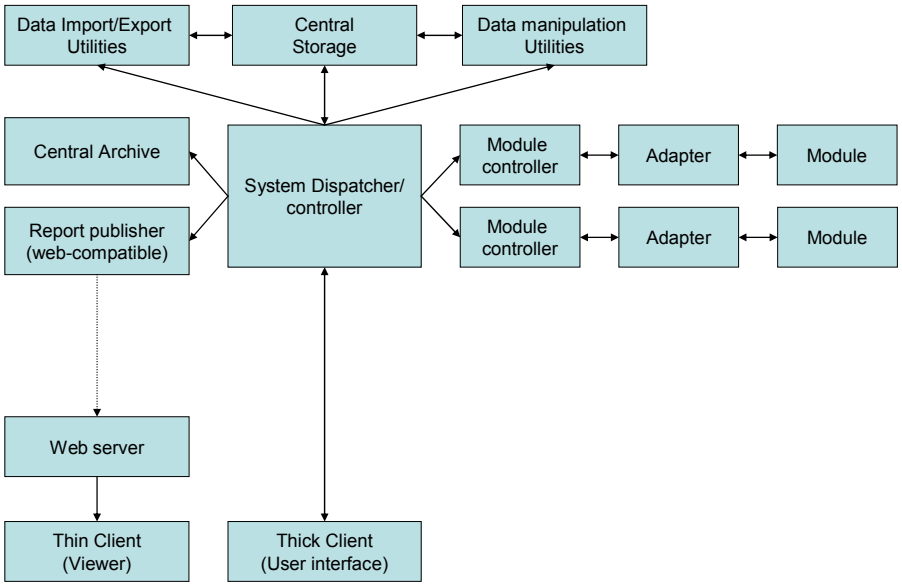


Figure 2-1 NGMS client-server logical design

Figure 2-1 does not prescribe a particular architecture (i.e. the above could all exist on a single or multiple physical machines). Note that two module controllers - and associated adapters/modules - are shown. In fact, there are likely to be more than this, subject to the constraints of the modules and the hardware infrastructure. In addition, the above does not preclude the use of a single system for all regions or independent systems for each region. For a standalone system the following configuration would be appropriate (Figure 2-2):

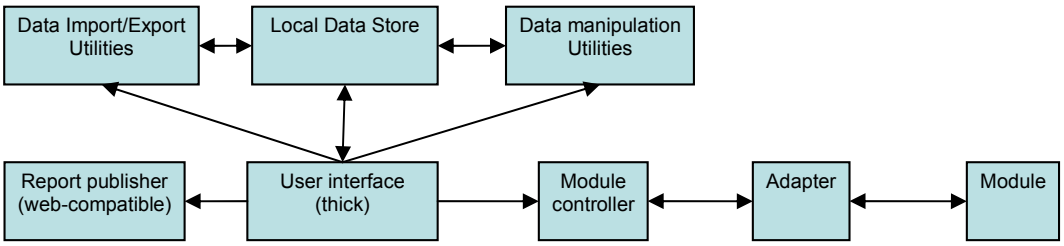


Figure 2-2 NGMS stand-alone configuration

The logical components are briefly described below. A more detailed description of the functionality of the above components is provided in the later chapters.

- **System Controller/Dispatcher**
  This component is responsible for scheduling and dispatching requests to execute module runs and other defined tasks. It maintains feedback on the status of the system components and any active module runs in progress.

- **Central Storage**

This facility provides storage for module datasets, boundary conditions and output (including pre-defined reports). This storage facility has expiration conditions. A separate section is incorporated to provide archiving facility for critical module runs (Module datasets, boundary conditions, output (including pre-defined reports), executables, logbook).

- **Data Manipulation Utilities**
  Utilities to manipulate the large raw module output files and to generate "slices" and subsets for analysis and display.

- **Data Import/Export Utilities**
  Utility to allow the import and export of data files to and from the NGMS.

- **Module Controller**
  Component to manage the execution and monitoring of the module. Usually the Modules are instantiated via the controller using a pre-defined script or batch file.

- **Adapter**
  Provides a standard published interface between the NGMS and third party modules. This enables the system to provide the data in the required format and location that is expected by the third party modules.

- **Module**
  The executable(s) encapsulating the Groundwater modelling physics (e.g. Modflow96).

- **User Interface (thick)**
  Provides the interface through which the user can:
  – Submit requests for module runs.
  – Selecting and modify input datasets
  – Select and display subsets of module output etc.

- **User Interface (thin)**
  Provides the user interface to browse data sets published on the web server

- **Report Publisher (web-compatible)**
  Enables generation of reports according to predefined layout formats in web-compatible file formats (HTML, JPEG, GIF, PNG)[1]

2.1.2 Logical mapping of the NGMS on the NFFS architecture

Figure 2-3 illustrates how the logical model could be mapped onto the existing NFFS architecture. Additional components are required as well as the modification of the existing controllers and utilities. As with the logical model detailed earlier, there is no prescription of the infrastructure architecture, i.e. the use of one server or the distribution of components across multiple servers/locations.

[1] *To ensure privacy of data, a manual action will take place to transfer this output to the EA web server according to standard EA procedure*

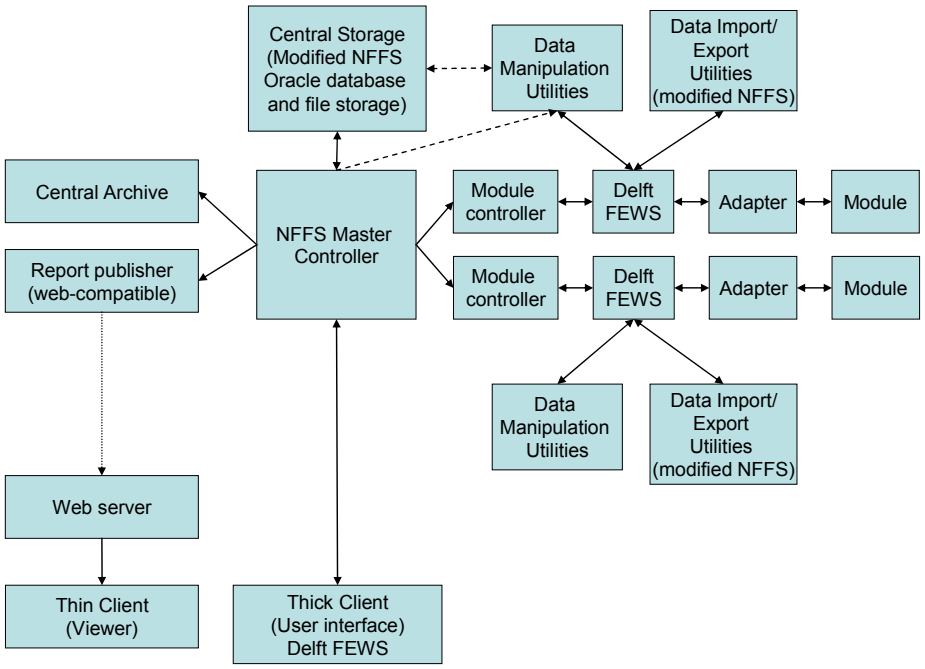


Figure 2-3 Logical model for the NGMS based on NFFS architecture

2.2 Physical architecture

2.2.1 NFFS System Architecture

The NFFS system is based upon a number of independent Java based clients and processes. Communication and data transfer between the central system and the "compute servers" as well as the (thick) clients, is based upon the Java Messaging Service (JMS).

- The key system entities, the central system, the Shell servers (the "compute" engines) and thick clients, are all de-coupled from each other. This ensures independent and concurrent operation as well as flexibility in selecting their location (both in terms of servers and sites).
- It allows the system to make appropriate use of available network bandwidth. Data is trickled down to clients (both end-user and to the Shell servers) effectively as background processes, allowing normal operation to take place concurrently. All data packets are compressed before being packaged within JMS.
- Such de-coupling also provides resilience as component failure; the failure of a single Shell server should not effect the operation of others.
- In summary, the NFFS architecture provides a controlled run-time framework for the associated modules. It ensures that Module processes are managed and that data and module configurations are held within a controlled environment.

Physical mapping of roles in the NFFS

Figure 2-4 provides an overview of the main components within NFFS.

- The MasterController (MC) components are the heart of the server. The MC is responsible for centralized data storage, data synchronisation. The MC contains a TaskManager which maintains a task list and schedules and dispatches tasks to shell servers via JMS.
- The Operator Client (OC) components hold the GUI components for presentation. Data is presented from a local data store which, by messaging with the MC, is continuously synchronised with the central data store.
- The FEWS Shell Servers (Shell servers) executes tasks and runs the numerical models via adaptors. The tasks are run from a local data store which is synchronised, by messaging with the MC, with the central data store. Results data and logging are handed back to the central system via JMS.

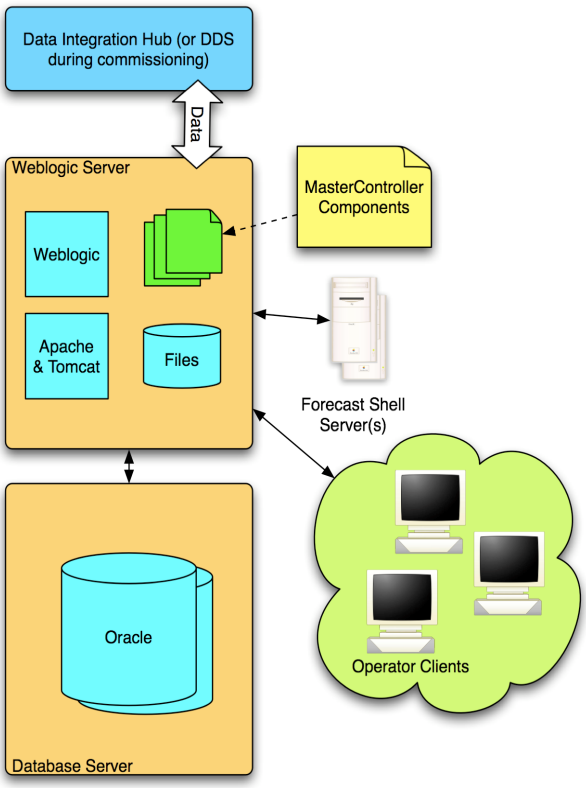


Figure 2-4 Main components within the NFFS architecture

Installations

The NFFS consists of three complete installations of the architecture (see Figure 2-5):

Online system (Leeds)

- Used for operational forecasting
- The “production system”

Online system Peterborough

- Same as Leeds
- Both sites are synchronised with each other

Offline system (Leeds)

- Used for testing and calibration
- Acceptance of new models etc.

Live data feeds from Telemetry and the Met Office are provided by the Data Integration Hub, which also acts as a distribution mechanism for NFFS generated forecast time series.

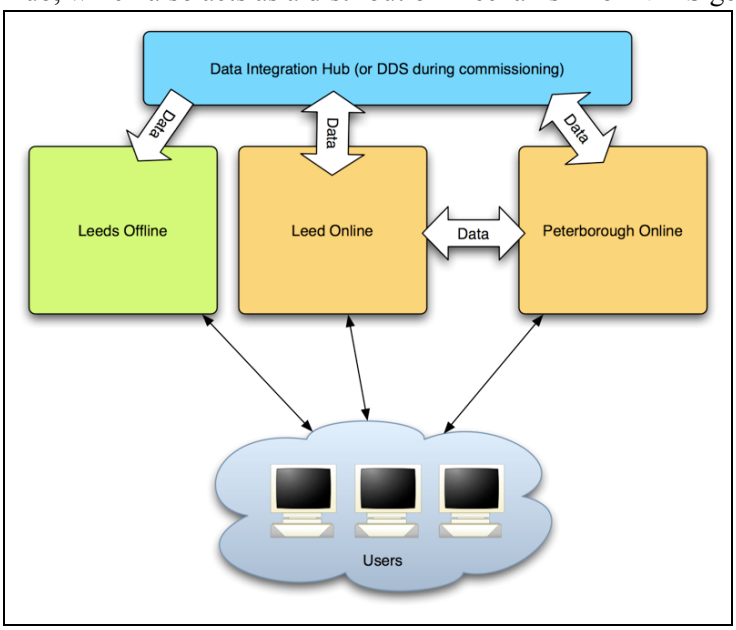


Figure 2-5 High level view of the implementation of NFFS within EA

2.2.2 Physical mapping of the NGMS on the NFFS architecture

Figure 2-6 and Figure 2-7 describes a possible physical implementation of the system. A central server is used to contain the system controller/dispatcher, a number of utilities and the central data-store. Separate machines could be used as the "compute engines" for the execution of the modules.

This approach allows a scalable solution; if particular modules require additional computing power then appropriate servers can be specified and added to the system. Note that the system illustrated does not mean that a particular Shell server contains only one module, a server could contain multiple (or all) of the module types.

Two systems are illustrated, the centralised "multi-user" (Figure 2-6) and the standalone (Figure 2-7).

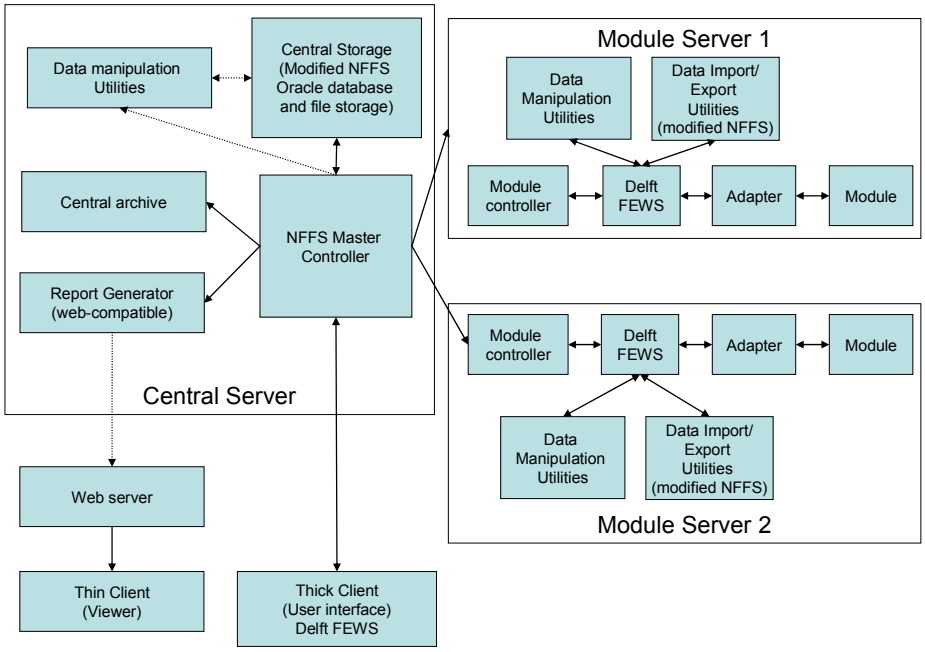


Figure 2-6 Physical mapping in centralised system layout

It is foreseen that the data manipulation utilities are a FEWS module residing on the Shell servers and the client. For reasons of performance it might be decided to have data slicing facilities work directly upon the central data store.

For the standalone system there is no requirement for the Master Controller component, module runs are dispatched directly from the thick client.

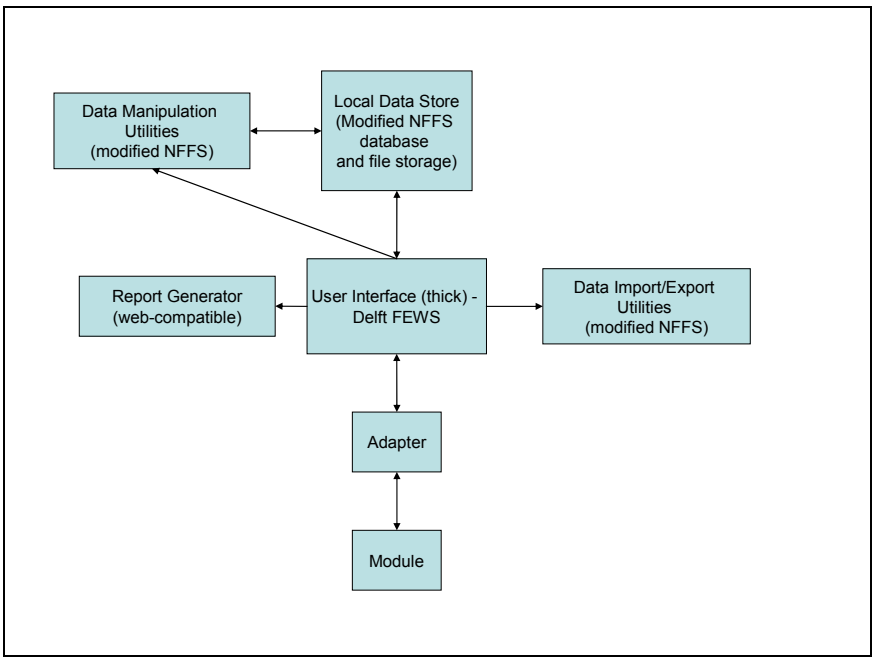


Figure 2-7 Physical mapping in stand-alone system layout

Installations

Figure 2-8 presents the proposed implementation of the NGMS. It consists of two complete installations of the architecture and a number of stand-alone applications for custodians, enabling them to prepare new module data sets and configuration updates. The two complete installations are hosted by CIS, possibly in Peterborough or Leeds. The Online system is the "production system" used by users and custodians for operational groundwater management assessments. The Off line (testing) system is used by custodians for testing and acceptance of new module data sets, configuration updates etc.

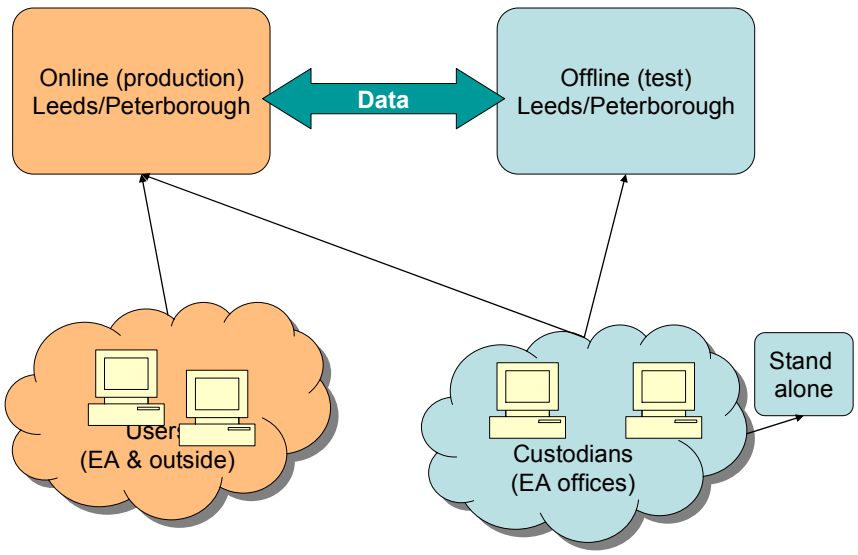


Figure 2-8 High level view of the proposed implementation of NGMS at EA

2.3 System Infrastructure

Details of the current NFFS infrastructure, together with the proposals for the NGMS pilot phase infrastructure are contained within the Hardware and Infrastructure Design.

3 High level solution components for NGMS

This section describes the design of the key components. Issues are identified for future discussion. Where the component is part of the current NFFS system, the functionality is described, and any major modifications required are identified.

3.1 Server components

3.1.1 System Controller/Dispatcher

Item	Details
Key Functionality	• Dispatch of tasks to the Module Controllers via JMS. • Monitoring of the dispatched tasks. • Data management (focal point for process access to central data-store)
Architecture Description	• Set of independent Java processes. (i.e. not managed EJBs etc.) The use of the J2EE WebLogic infrastructure is used only to provide the required JMS layer. • The communications with Module Controllers and thick clients is via JMS. • Can be implemented on a distributed architecture (i.e. the components that make up the controller can reside on different physical servers). • The platform is not restricted to HP or WebLogic.
Changes from NFFS Implementation	The System controller/Dispatcher must be able to handle execution templates that define any required run-time status messaging. These will be contained within the existing TaskProperties.xml payload.
Issues	-

3.1.2 Central storage

Item	Details
Key Functionality	Storage of system data (task list, status, configurations etc.) Acts as a repository for; • Module data set identification templates (incl. input data displays) • Module datasets • Module boundary condition datasets

	• Module run output datasets (complete) • Module run output datasets (subsets) • What-if scenarios • Raw output generation templates • Data slicing and post-processing templates • Pre-defined report templates. • Report output files. • Modeller's Logbook
Architecture Description	Implemented via Oracle database(s) and file storage.
Changes from NFFS Implementation	The NGMS system will not require the raw output from the Module runs to be stored in its native form. The proposal is that the data is extracted from the raw file, handed over as grid time series (pi_map stacks.xsd) and stored within the Oracle database. This will provide the advantage in that the extraction of data from the central system (either to clients or the Shell servers) can be controlled and restricted. See Data Manipulation utilities for the data slicing and post-processing templates. The following updates to the data model are required: - Concept of Standard/common datasets, which are fairly static and rarely need re-synchronising to the local data stores. - User specific datasets (both input and output), which are more transient in nature. - Requirements for capacity management (based on quotas for users) - User specific access to data-sets - Structured modeller's logbook which is automatically filled where possible
Issues	Storing raw output over to archive

3.1.3 Central archive

Item	**Details**
Key Functionality	Acts as an archive for critical module runs • Run-time log by executable of critical meta-data (no in/output) • Modeller's log book (see below) • All module input data sets (incl. native files) • All module object code (executable) • All module output data sets (incl. native files)
Architecture Description	Implemented via Oracle database(s) and file storage.
Changes from NFFS Implementation	For jurisdictional reasons, the NGMS-archive may need to hold at least one native output file to validate any rerun. See central storage section above for details of transfers etc.

Issues	The archive may be combination of hard disk from the latest module run versions and hard disk or tape for the older versions *Legal requirements should be analysed by EA*
Explanatory	Critical module runs are: *Standard Scenarios* • Naturalised (zero abstraction) • Historical • Fully License (predictive) • Recent Actual (baseline) *Other* • EA-approved Climate change scenarios • Essential calibration runs (other than historical).

3.1.4 Synchronisation Code

Item	Details
Key Functionality	Provides the underlying mechanism for synchronisation of the data: • Between the central system and the Shell servers • Between the central system and the operator clients
Architecture Description	Set of APIs usable by both Master Controller components and FEWs components.
Changes from NFFS Implementation	• More selectable criteria (by user/dataset/output). • Breaking up of large data items into smaller components • Additional API functionality for direct component JMS messaging (to be used as part of the real-time status messaging)
Issues	-

3.1.5 Access and Security

Item	Details
Key Functionality	Need to determine key functionality. For the pilot it should be kept very simple: • Segregate users • Only 2 roles – custodian and viewer • Limited role management (creation of accounts/assignment of roles etc.) • Data access restrictions on data set level only
Architecture Description	
Changes from NFFS Implementation	The facility already exists to stamp all messages with a userId and hence track operations etc.. Additional components are required to provide user and role management (including interfaces). Updates to data model to including tagging of datasets with roles and their access rights.

	Modification of data access and synchronisation to include user-specific restrictions.
Issues	Data access restrictions within a data set (is this a realistic request ? how ill it work in practice ?)

3.2 Shell server (FEWS instance) components

3.2.1 Synchronisation code

See section 3.1.4

3.2.2 Adapter

Item	**Details**
Key Functionality	The adapters provide an interface between NGMS (via the NFFS Published Interface) and third party modules (see Figure 3-1)
Architecture Description	• On the NFFS-side, the General Adapter converts NFFS data objects into XML (according to the published interface format) • On the module side, the the XML files are converted in native data formats by the Module Adapter • The General Adapter kicks off the export/import activities as well as the module run itself • Adapters should be developed as separate components (.exe or .jar)
Changes from NFFS Implementation	• Adapter structures can be obtained from NFFS but the adapters themselves have to be developed.
Issues	• If the source code of the computational core is available, it is recommended that the core is migrated to the OpenMI interface standard (www.openmi.org). When both groundwater codes and recharge codes support this interface, process interaction can be simulated, hence bringing integrated catchment modelling a large step forward. A generic module adapter can support the conversions between NFFS published interface and the OpenMI interfaces. • Adapters are generally developed by the module owner but in case of the groundwater modules the software is often freeware or owned by EA. • It is foreseen that slight different versions of module cores (e.g. Modflow VKD and Modflow 96) can be handled by the same module adapter. The pilot should prove this • The pilot should indicate whether extra data size reduction is needed by having adapters that control basic output generation (parameters, timeframe etc.). In this case, an additional template (Raw Output Generation Template) is needed. The module adapter will use the content of

	this template to create the model control file • Unresolved during the pilot will remain the question whether NGMS should enforce numerical data exchange through PI-formats (grid time series) in case two modules can run in sequence with native output being taken as input

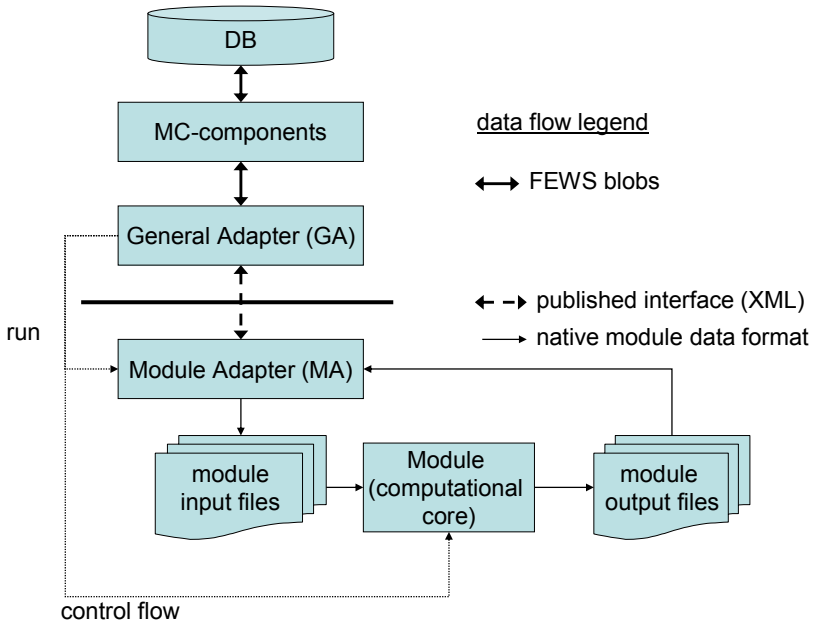


Figure 3-1 The role of Adapters as data translators within the NFFS

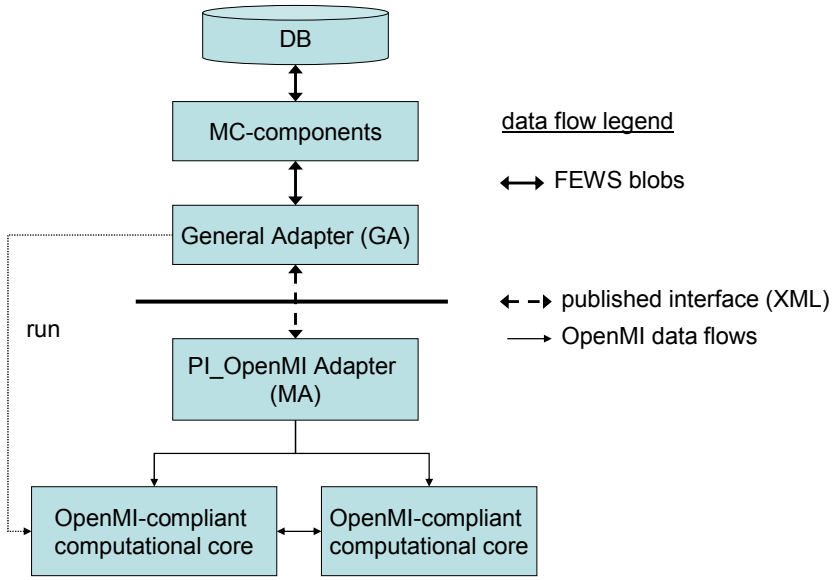


Figure 3-2 The role of Adapters as data translators when introducing the OpenMI approach

3.2.3 Module

Modules are external to the system and interface via the module adapters. In the framework of the NGMS, they are assumed to run on a Windows platform but may also run under other operating systems.

3.2.4 Module Execution Controller

Item	Details
Key Functionality	The Module Execution Controllers reside on the Shell servers. They provide the management of the execution of the modules. • Initiate execution of the task (via a batch or script file). • Obtain status information and report back to task controller/dispatcher. • Accommodate manual interruption • Accommodate automated interruption in case the expected computation time is exceeded or in case the expected output file size is exceeded.
Architecture Description	• Probably not the existing MCProxy but a new component that is called via the FEWs General Adapter. • Requires additional configuration file (Execution Template)to manage logging information . • Will utilise the new "Status Message" functionality to feed back information to the central system through JMS
Changes from NFFS Implementation	Allow intermediate status messaging and manual interruption or automated interruption when a threshold is exceeded.
Issues	First version developed in the pilot phase

3.2.5 Data slicing API (Data manipulation utilities)

Data manipulation utilities perform a pre-defined operation on the raw output dataset (which is the result of a module run).

The key role of the data manipulation utilities are to ensure that only manageable sized data selections are downloaded to any clients, and that the large results data-sets and input configurations remain on the central system.

Operations can be executed on Shell servers, on the central system under the control of the system controller/dispatcher and on the thick client, to allow operation in a stand-alone mode.

Item	Details

Key Functionality	Data slicing: Selection of a subset of the data (by parameter, time frame, spatial extent/location)
Architecture Description	• Data slicing module performs operations on a selected output dataset that is available in NGMS format (i.e. working directly on the central or local data store). • Data slicing operation instructions are provided in a populated data slicing template (XML-object) that is provided as part of the workflow or task run • Operations can be executed on Shell servers, on the central system under the control of the system controller/dispatcher and on the thick client, to allow operation in a stand-alone mode. • The Data Slicer should be implemented as an API (separate module) for use throughout the system. It could still be used by FEWs but acts on the local data store directly. • It could be linked to synchronisation, i.e. a user requesting data set doesn't need to kick off a task to get the data. This Gives the possibility for an easier move to a thin client in the future.
Changes from NFFS Implementation	• To prevent complex changes to existing modules, a new dedicated (and optimized) module will be added • More details in section 0 • A User interface will be developed to specify the slicing operation by populate slicing templates
Issues	First version developed in the pilot phase

3.2.6 Post-processing module (Data manipulation utilities)

Item	**Details**
Key Functionality	Perform a pre-defined data aggregation operation on a (sliced) output data set (which is the result of a module run). The available operations will include: • predefined operations to generate groundwater specific graphs and maps (e.g. stream accretion, groundwater unit budgets etc) • Basic statistical operations (average, minimum, maximum. standard deviation, accumulation) • Basic comparison operations (difference, proportion, change over time/location)
Architecture Description	• The NFFS modules for Data Transformation and Interpolation can be pre-configured to do the job with non or minimal extension • Data is available in NGMS format, and will be directly extracted from the (local) data-store • By default, the utility would be executed on the Shell server. and the thick client to allow operation in a stand-alone mode. • The statistical/comparison data operations specification (and the type of target file) will be contained within the existing XML

	configuration files of the NFFS-modules.
Changes from NFFS Implementation	• Add contour mapping library to NFFS
Issues	• Module adapters need to provide vector data to enable horizontal views on groundwater unit budgets. • Reusing the existing NFFS modules does not (yet) allow user defined configurations for post-processing

3.2.7 Data Import/Export Utilities

Item	**Details**
Key Functionality	• Import the external datasets into the central system for storage. • Export data from the central system to a defined location. • Allow the archiving of module run data (input and output) to a defined location. • Allow the exporting of web-compatible reports to a defined location
Architecture Description	By default, the utility would be executed on the central system to perform operations on the appropriate dataset (under the control of the system controller/dispatcher (via a scheduled task). The export utility would have to take into account the potentially large data files sizes. Limits would have to be placed on what size files could be exported and whether alternative mechanisms should be put in place. In the NFFS implementation, archiving is achieved by allowing a limited number of forecast "packages" (input files, output files, module configurations etc.) to be stored within the central data-store. These are retrievable by the users for future analysis.
Changes from NFFS Implementation	For the NGMS, the key module datasets (input and output) that will be used by the system for processing or display will be stored in the Oracle database. Data that is only utilized by the module adapter may be stored as (compressed) flat files. • A contour mapping library needs to be added to accommodate production of Shapes • Export facilities need to be extended to accommodate data export in Shape and/or Excel format •
Issues	The pilot should indicate the preferred storage mechanism (table/Blob) in the Oracle database

3.2.8 Report Generator (web-compatible)

Item	Details
Key Functionality	The Report Generator produces web-compatible pre-formatted reports at the request of the central controller/dispatcher (via a task). Transfer to the EA web server will be a manual action according to EA publication procedures. The web pages are static (they provide minimal user interaction via the browser).
Architecture Description	• During the pilot, the web server is for use internally within the EA. • It is likely to be hosted on the regular server hardware used by the EAsinet (i.e. not a dedicated web server platform).
Changes from NFFS Implementation	Report content options and report formats are configuration changes rather than modifications to the components.
Issues	

3.2.9 Module Execution Controller

Item	Details
Key Functionality	The Module Execution Controllers reside on the Shell servers. They provide the management of the execution of the modules. • Initiate execution of the task (via a batch or script file). • Obtain status information and report back to task controller/dispatcher. • Accommodate manual interruption • Accommodate automated interruption in case the expected computation time is exceeded or in case the expected output file size is exceeded.
Architecture Description	• Probably not the existing MCProxy but a new component that is called via the FEWs General Adapter. • Requires additional configuration file (Execution Template)to manage logging information . • Will utilise the new “Status Message” functionality to feed back information to the central system through JMS
Changes from NFFS Implementation	Allow intermediate status messaging and manual interruption or automated interruption when a threshold is exceeded.
Issues	First version developed in the pilot phase

3.3 Operator Client (FEWs instance) modules

3.3.1 User Interface (thick client)

Item	Details
Key Functionality	The thick client acts as the main interface to the system for the users. A number of key functions are provided: • Define abstraction locations • Modify boundary condition datasets • Select report types • Define what-if scenarios • Define data slice templates • Select post-processing options • Select module (and module datasets) package to execute. • Submit package to be executed • Select subsets of module results (includes selecting previous output runs) • View subsets of model results • Modify module datasets (limited; undefined) • Modify boundary condition datasets (pre-defined type of changes) • Modify boundary condition datasets • Download module datasets • Upload configuration updates
Architecture Description	• Java client. • Local data store (file and database) • Use of JMS to synchronise requested data from the central system
Changes from NFFS Implementation	A number of new GUI-plugins will be introduced (after the pilot) to provide dedicated user interface support The following new plugins are expected: • AbstractionLocator • SliceLocator • ExecutionSpecifier • Data management/system administration Plugins: to be adapted • GridDisplay → add ModuleDataSet Browser functionality Additionalle plgins will be needed to manage data downloads, runs etc.
Issues	User/role specific restrictions to data access

3.3.2 Logbook

Item	Details
Key Functionality	Model journal/logbook holding meta-information about a module run. It will become part of an archived Module run, and is populated by combination of manual entry (at Operator client) and automation (at Shell Server or MC)
Architecture Description	Flat data structure in XML, supported by a GUI-for data entry, as well as a set of functions to accommodate automated filling.
Changes from NFFS Implementation	New
Issues	Will not be part of the pilot

3.4 External Client components

3.4.1 User Interface (thin client) – Simple report viewer

Item	Details
Key Functionality	The thin client (browser based) provides a reduced functionality interface to the modelling system. The basic functionality includes: • Select pre-defined reports • View pre-defined reports • Download re-defined reports
Architecture Description	Static web pages are served by the Apache web server to the browsers. The Web Application server (Report publisher) copies the pre-formatted reports to a pre-defined directory structure.
Changes from NFFS Implementation	• Report locations • Report formats • Web page format
Issues	Report formats will be configured during the pilot phase.

3.5 Template/configuration extensions

This section defines the updates to the various configuration/template files.

3.5.1 Raw output generation template

If the raw output generation by module runs puts too much requests on disk capacity, a solution might be introduced to manage the output production. in this solution, a raw output production template is populated for each module run with the output parameters and

temporal settings as required. The module adapter will have to convert the data in this template into the module control file of the computational core.

The raw output generation template will have to cover:
- Variable
- Vertical layers
- Time frame (begin, end, step)
- File format
- Destination files

3.5.2 Data Slicing template

To accommodate data slicing, user defined instructions should be provided to the Data Slicer module. The instructions will be captured by populating a Data Slicing template, i.e. an XML- configuration file which needs to cover:

- the Input TimeSeriesSet (grid time series)
- the Output Time Series Set (grid time series)
- the Spatial extent of the slice (grid cell centre points that are part of the slice)
- the Temporal extent (start date, end date, step)

Inclusion/exclusion rule
In those cases where polygons are defined, the following in/exclusion rule will apply:
All cells with a cell centre inside the polygon boundary are included in the slice. All cells having their cell centre outside the polygon boundary are excluded.

More details are provided in section 0

3.5.3 Post processing template

Post-processing templates are normal XML-configuration files for the Data Transformation module and the Interpolation module.

3.5.4 ModuleExecutionController configuration file

Associated with each Module is a ModuleExecutationConfiguration XML file. This configuration file contains the information required by the ModuleExecutionController to monitor and control the associated module. It includes the following information:

- Path to module executable
- Path to the Module status file (or runtime logging file)
- Frequency of status checking
- Module status Mask - Mask for status file parsing (i.e. what to look for in the Module status file)
- Module interruption Mask - Mask for status file parsing
- Module interruption – Action to be performed on detecting the interruption pattern.
- Return code to the General Adapter

3.6 Interfaces

The NGMS will require no new published interfaces.
The NGMS will require various new XML-configuration files (see previous section).

4 System operation

4.1 NGMS Workflows

This section provides a high level description of the major workflow scenarios in NGMS.
Workflows are used to define logical sequences of tasks such as running a module, slicing data, post-processing data etc. The workflow simply defines the sequence with which the configured modules are to be run. Workflow analysis plays a crucial role in system design as it clearly identifies the modules to be run, while each module should be fed with the relevant data.

Within NGMS the following workflows have been identified.
1. Run Module Adapter
2. Run Data Slicer
3. Run Post-Processor
4. Run Report Generator
5. Archive results (not described)

All workflows can be run separate or in the following workflow-composites ‘1+2’, ‘1+2+3’, ‘1+2+3+4’, ‘2+3’, ‘2+3+4’, ‘3+4’.

4.1.1 Run Module Adapter workflow

Figure 4-1 illustrates the general workflow to run a module via a module adapter. This workflow will be implemented for Modflow96, ModflowVKD, 4R, EA recharge code and possibly others. Section xxx will discuss the run-time status messaging aspects in more detail.

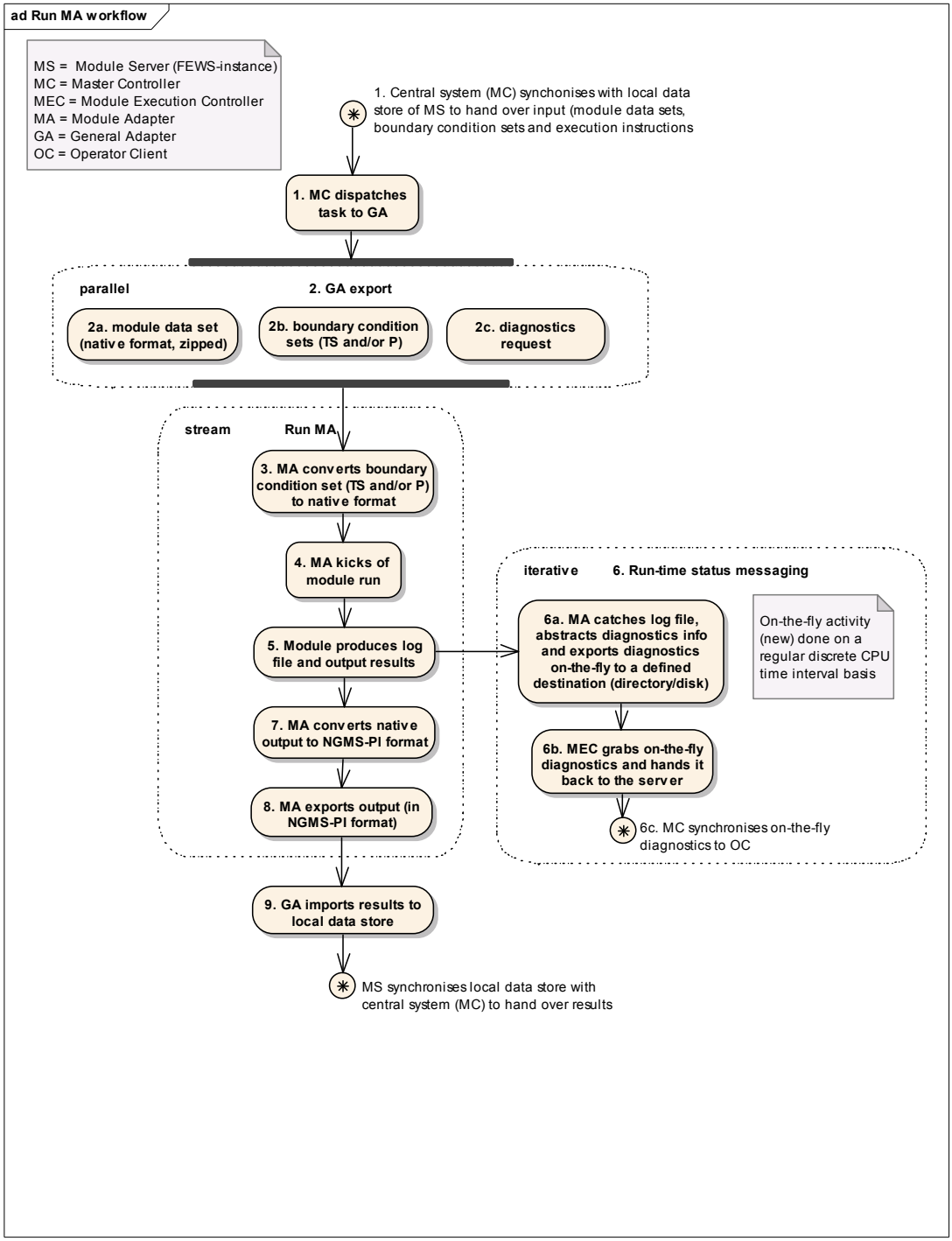


Figure 4-1 Activity sequence to run Module Adapter workflow

4.1.2 Run Data Slicer workflow

Figure 4-2 illustrates the basic workflow for the Data Slicer. The details are given in section 0

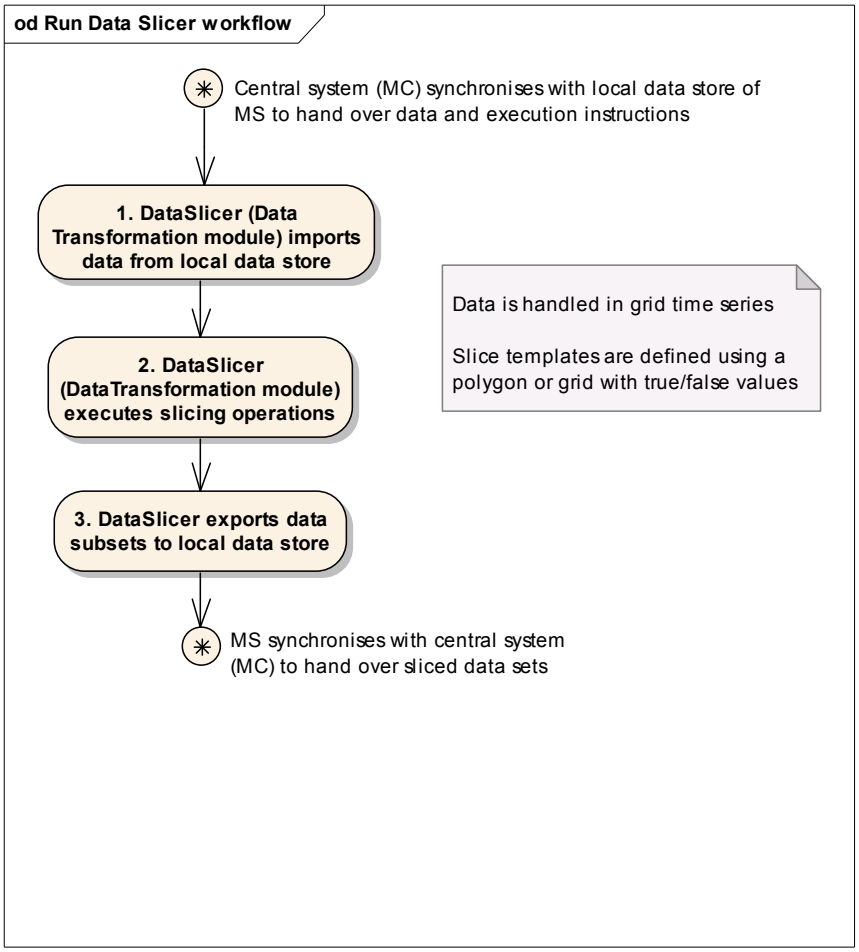


Figure 4-2 Activity sequence to run Data Slicer workflow

4.1.3 Run Post Processor workflow

Figure 4-3illustrates the workflow for post-processing data from sliced module output to groundwater specific data sets, ready for display.

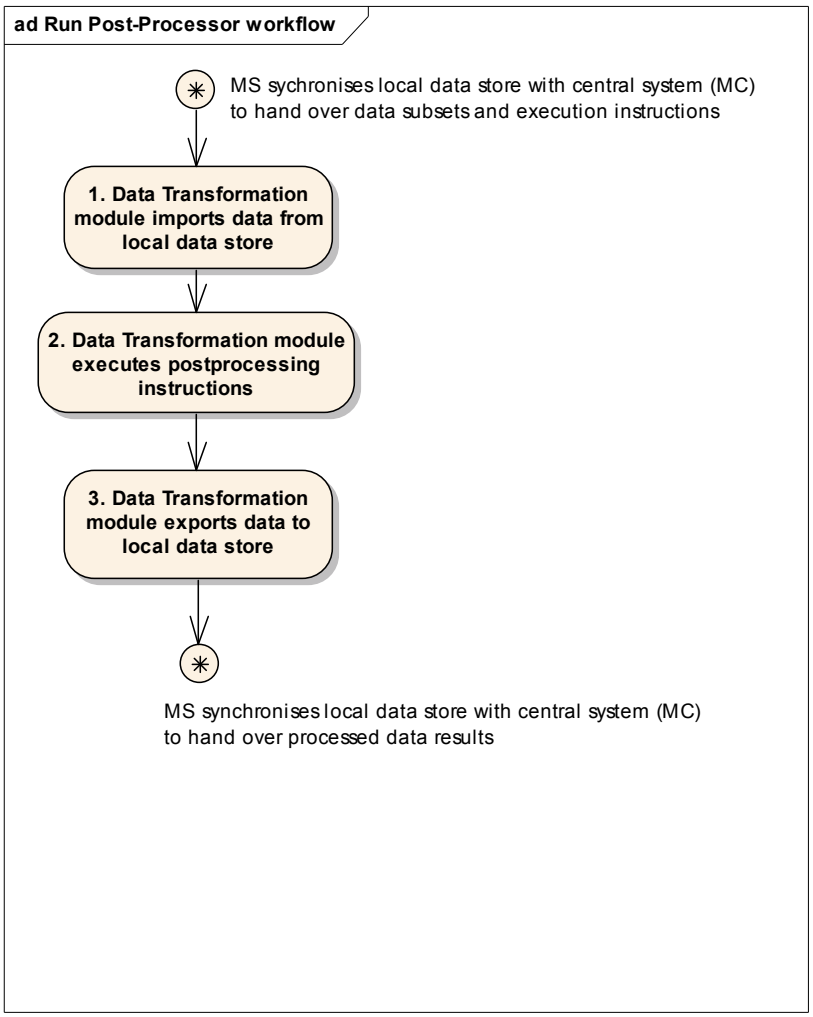


Figure 4-3 Activity sequence to run Post-Processor workflow

The following post-processing activities can be performed by proper instruction of the DataTransformation module:

- Temporal averaging
- spatial averaging
- Differences for one parameter between current run and a reference data set for entire grid
- Differences between two parameters of a single run
- Accumulation of parameter time series at selected locations
- duration/exceedance curve for one parameter at selected locations
- transient water budget for predefined area (accounting for storage change, stream flows, abstractions, recharge, groundwater flows)
- accumulative water budget for predefined area (accounting for storage change, stream flows, abstractions, recharge, groundwater flows)

- stream accretion over period for predefined river
- winterbourne signature over period for predefined river stretch

If the groundwater module produces horizontal vector fields as a combined set of scalar outputs, the data Transformation module can also be applied to generate:

- horizontal projection of groundwater flow over period in predefined layer at predefined location
- groundwater flow passing the boundary of groundwater management units

Appendix A provides a detailed overview of pseudo-scripts to generate the required data sets for the various types of reports

4.1.4 Run Report Generator workflow

Figure 4-4 illustrates the workflow that is associated to generating the web-compatible content of report that is exported to an external destination.

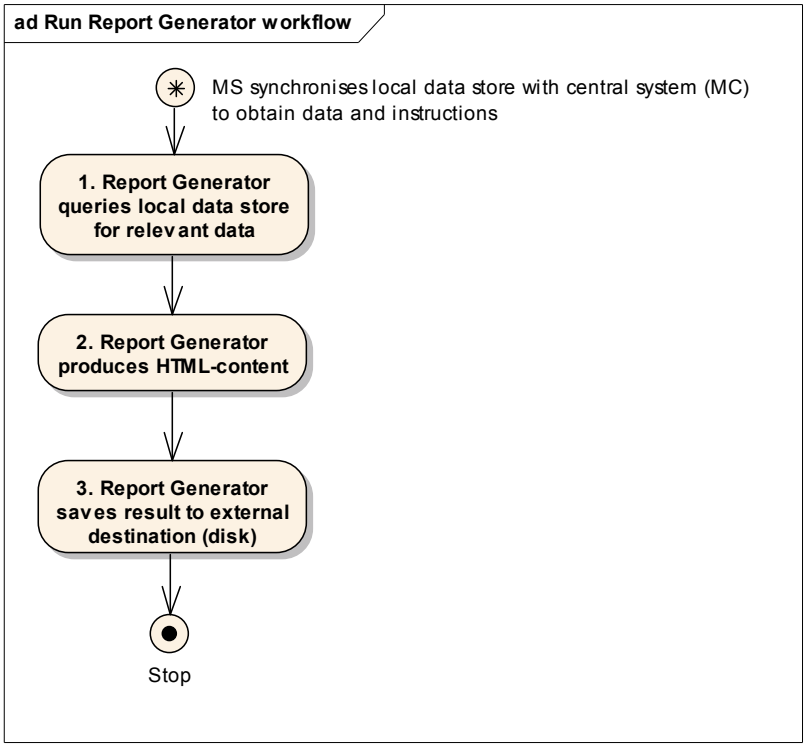


Figure 4-4 Activity sequence of Run Report Generator workflow

4.2 High level interaction

This section provides a high level description of how the NFFS architecture operates in typical operational scenarios. It also describes how an NGMS implementation may differ.

4.2.1 Submission of a model run on the central system

NFFS:
Model runs can be submitted to the central system for execution at a specific time or "as soon as possible". Details of the tasks to be performed are defined on the client application (via the selection of which workflow to run, various execution options etc.) and then sent to the central system for scheduling.

Note that a Module run is more than just executing ISIS etc. various other post-processing modules can also be incorporated into the workflow and are handled in the same way.

Tasks can be submitted via the thick client (Operator Client) or via a web-based system Administration Interface. The Master Controller components (Task Manager etc.) have the responsibility of adding a task to the current list and ensuring that it is dispatched to an appropriate Shell server for execution.

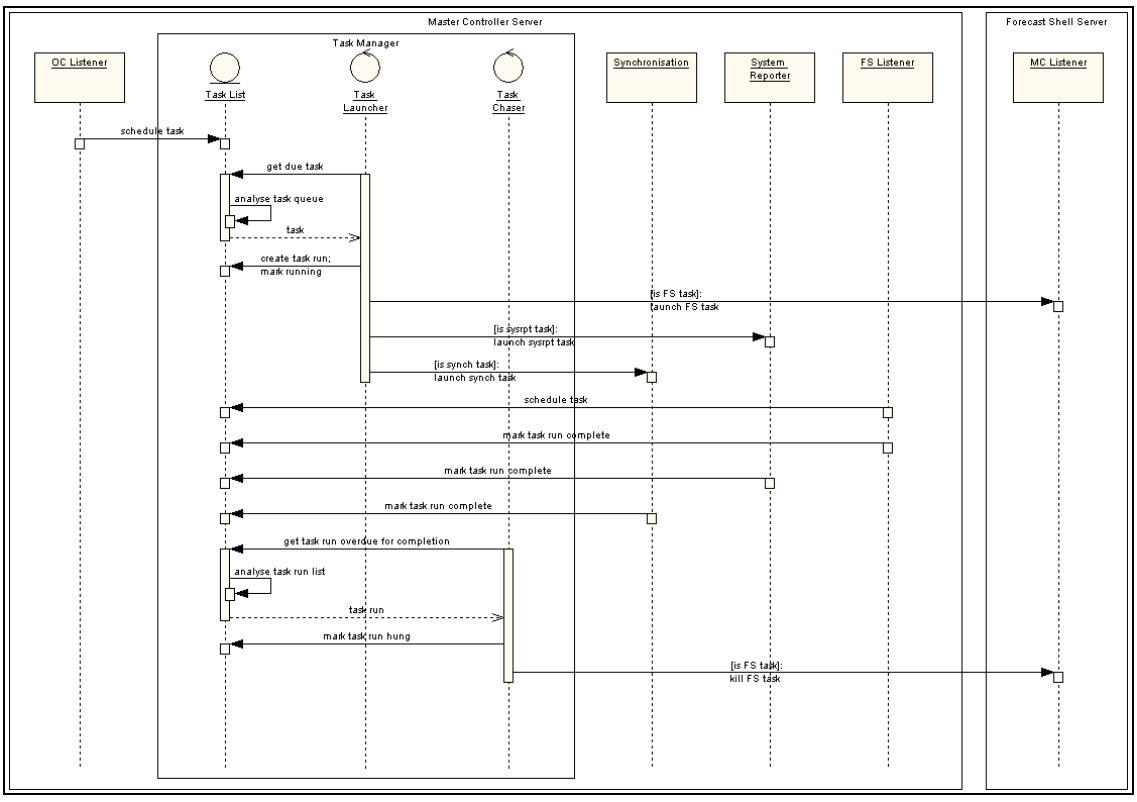


Figure 4-5 Interaction between the system components during the submission of a task to a Shell Server

NGMS differences:
A modified user interface will be provided to generate the appropriate "task/workflow package". This is all generated behind the scenes; the user will select various options (e.g. in the case of adding an abstraction they will select its location etc.) via a graphical interface.

Data slicing and post-processing options will be treated like any other module, i.e. they can be included in a workflow (both pre-defined and dynamic) and executed on a Shell server.
(The user interface converts the various options into the Raw Output Generation Template, Data Slicing Template etc. as appropriate).

4.2.2 Execution of the model run on a Shell server

NFFS:
Details of the task to be executed are sent to the appropriate Shell server via a JMS message. What determines whether a Shell server is appropriate or not depends upon a number of configurable factors, such as how busy the server is, whether the module is allowed to run on that particular server etc.

On the Shell server, a module controller (MCListener) waits for JMS messages and takes the appropriate action. If the message is to perform a model run, an instance of FEWS is instantiated and passed a reference to the appropriate workflow package that must be executed. FEWs will look to its local data-store for the contents of the workflow in question. FEWs will execute the workflow, starting up the appropriate modules via the General and Module specific adapters. All datasets are sourced from the Shell server's local data store, which is regularly re-synchronised with the central system. Usually, the first task the FEWs will undertake is to check with the central system whether it has the most recent data, and if not, it will perform a synchronisation.

Note that FEWs will use the data that is available in the local data-store, it does not request specific datasets (by name, type etc.) from the central data-store (a slight variation to this is the download of approved forecasts from the central data-store to the Operator Client local data-store at the request of the user).

The module adapters ensure that the data flowing in and out of the modules is in the appropriate format (and stored in the correct location). In addition, the adapters allow the passing of status information back to the FEWs from the modules.

On completion of the Module execution, the module results dataset is stored in the local data-store. This is then synchronised back to the central system to allow dissemination of the results.

Log entries are synchronised from the Shell Servers local data store to the central data-store on a regular basis during the module execution. The main data synchronisation takes place at the end of the module run.

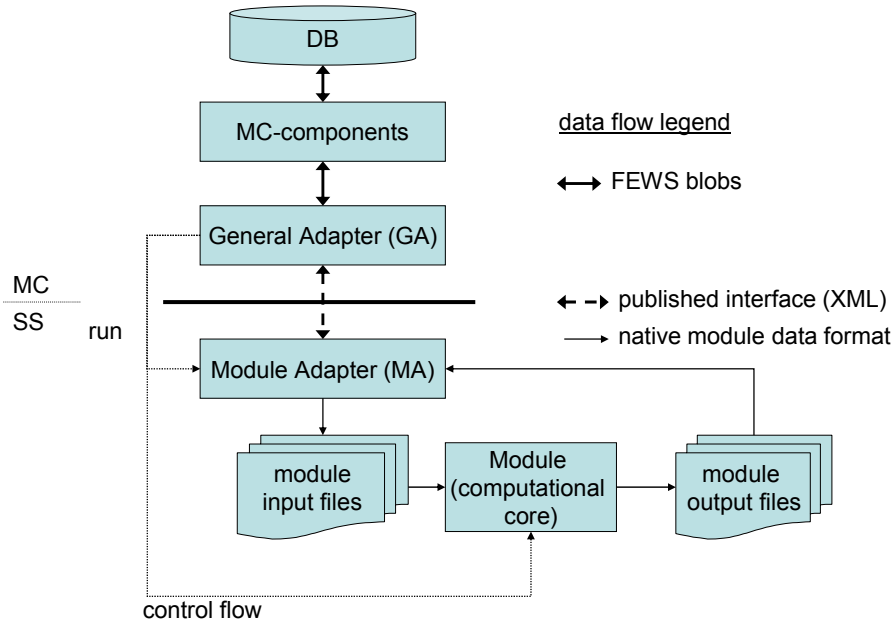


Figure 4-6 Adapter role's during a module run

NGMS differences:
The operational requirements of the NGMS indicate the need for two distinct types of datasets:

- Common (or standard) datasets
  These would be fairly static, probably only changing every 6 months or so. They would be the definitive datasets, accessible ("read only") to all users.
- User (custodian or user) Specific datasets
  These would be the datasets that are created in normal day-to-day operations. They include both input and output datasets.

The proposal is that in normal operation, a user would make a number of module runs (modifying/adding abstractions etc.), until finally settling on the agreed input and output datasets. The custodian would then have the functionality to promote any "User Specific" datasets into the Common/Standard datasets area, allowing their (read-only) use by other users of the NGMS. The User-Specific datasets could have a low retention period (weeks?) to maintain system capacity (this might be the case just on the Shell servers, i.e. the datasets remain on the central server – subject to the users data-space limits – but are removed from the data-stores on the Shell servers. If they were required again on the Shell servers, due to a user re-running an investigation etc,. The Shell servers would have to re-synchronise that dataset prior to the execution of the module.)

Currently FEWS assumes that all the datasets are present in the local data-store i.e. the task doesn't specify what datasets to use for this module run. This requires modification such that specific datasets can be requested. The current synchronisation API is based on time and the TaskRunID. In NGMS this might not be sufficient, there will need to be additional criteria required to uniquely identify the required subset of data. Examples are,

- Get dataset X rather than get dataset from 2 hours ago

- Select datasets based on userid and specific modules.

The pattern is in place already within NFFS for this enhancement.

In order to allow more control of the execution of Modules and to provide the ability to obtain real-time feedback on the progress of Module execution a new component, the ModuleExecutionController is required. It resides on the Shell servers and has the responsibility of starting the appropriate module, following a request from the FEWS (via the general adapter). The ModuleExecutionController provides the following functionality (see Figure 4-7);

- Instantiates the appropriate Module
- Queries the Modules output log for status information
- Communicates status information to the system via a new direct JMS messaging "status message" functionality (API).
- Responds to a "kill module" command (issued via the "status message" functionality).

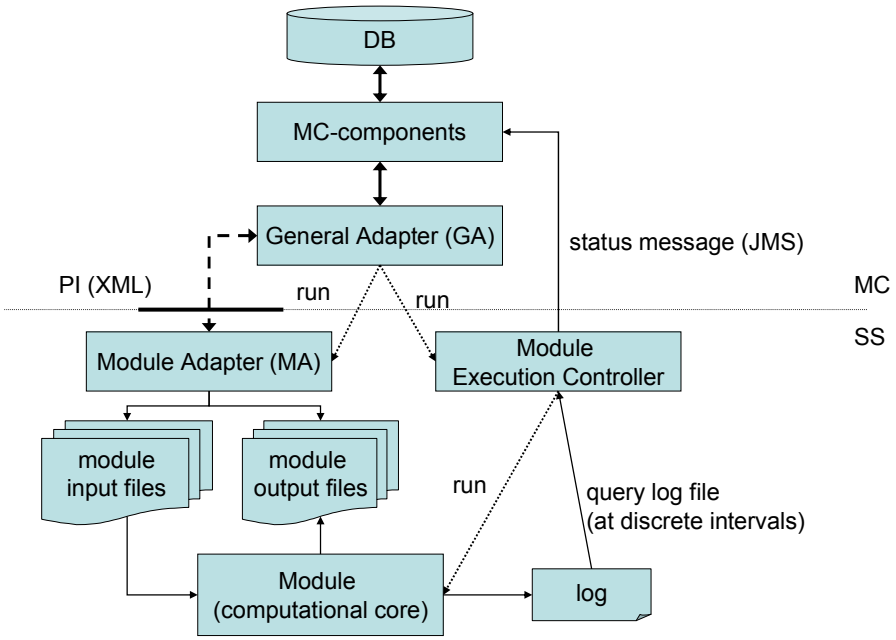


Figure 4-7 Module Execution Controller's role in a module run

Figure 4-8 illustrates the setting in case the openMI is utilized as a data exchange mechanism. Dependent on the capabilities of the computational core, water balance errors or convergence data might directly be asked from the core without interference of a log file.

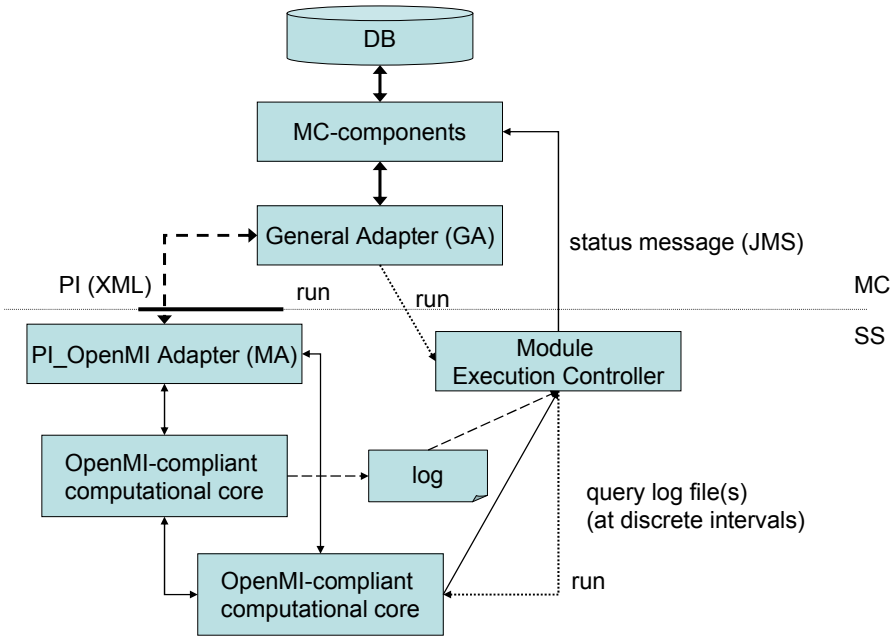


Figure 4-8 Module execution controller's role with OpenMI compliant codes

A number of the NGMS modules (e.g. Modflow) operate under the principle of a "control file". This native file provides information to the module related to its execution, i.e. number of time steps, convergence criteria etc. The pilot will have to indicate if data sizes become too large, thus requiring additional control on the output generation. In this the Module Adapter dynamically will have to create this "control file". It does this with the aid of a number of xml templates that are associated with the particular task run:

- Execution Template
- Raw Output Generation Template

The details of the above are covered in section 3.5.

4.2.3 Viewing of datasets and model run results by the user:

NFFS:
The thick client (Operator Client) utilises its own local data store for the presentation and analysis of the module datasets (input and output). The datasets are synchronised from the central system both on a regular (every few seconds) and ad hoc basis. Users can select specific datasets that are stored on the central system and then choose to download them to the local data-store.

NGMS differences:
Within NFFS, the majority of the data is downloaded automatically to the client (current forecasts, latest weather details etc.). For NGMS this is unlikely to be the required functionality, most downloads will be driven by the user, they actually request to download a specific dataset (input of output) or post-processed results. This will require an appropriate GUI.

4.2.4 Data Synchronisation

NFFS:
Data synchronisation takes place between the central data-store and local data-stores. The local data-stores may be located on the Shell Servers and the Operator clients. Details of where synchronisation takes place and what can be transferred is detailed below:

MC ⇔ Shell Server
The synchronisation between the Master Controller and Shell servers is handled by two main components:
FSListener resides on the Master Controller server. Its role includes:

- Receive data messages (both requests and uploads) from the Shell Server as part of the Shell Server's synchronisation.
- Receive event alerts from the Shell Server.
- Receive log message entries from the Shell Server.
- Receive task completion messages from the Shell Server.

MCListener resides on the Shell server. Its role includes:

- Accepting task dispatch requests from the MC.
- Starting a Forecast Shell to honour the request.
- Responding to queries from the MC on the status of the SS.

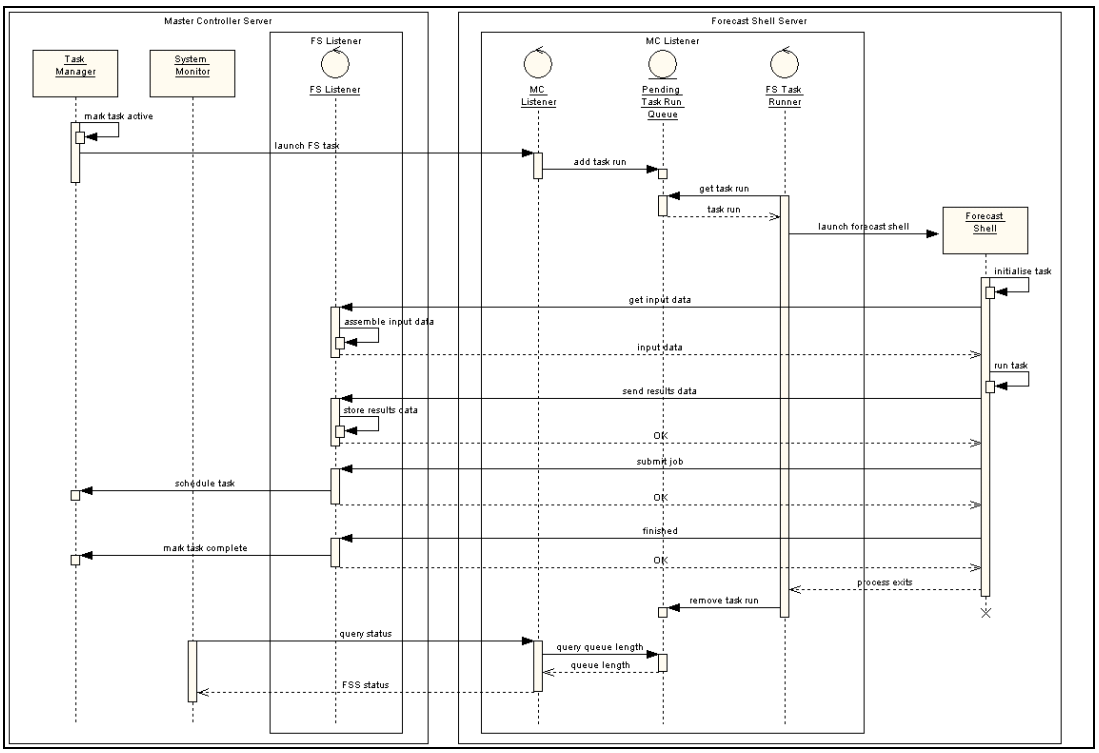


Figure 4-9 Interaction between the Master Controller Server and the MC Listener residing on the Shell Server

MC ⇔ Operator (thick client)

The synchronisation between the Master Controller Server and the thick client (Operator Client) is implemented via an OC Listener on the Master Controller server, and the FEWs component making use of a common NFFS Synchronisation API (see Figure 4-10).

Currently the data synchronisation is based on:

- Timeframe (i.e. all data before or between a set of dates)
- TaskRunID (all data associated with a particular instance of a task run)
- Modifiers (include Sync level on time series, severity, source of log entries etc.)
- Named table basis

A pattern exists within the current implementation to allow easy expansion of the synchronisation options.

NGMS differences

The following updates to the Synchronisation API are expected:

- Distinction between synchronisation of common and user specific datasets.
- Intelligent handling of very large data synchronisations (break-up into appropriate size "chunks").

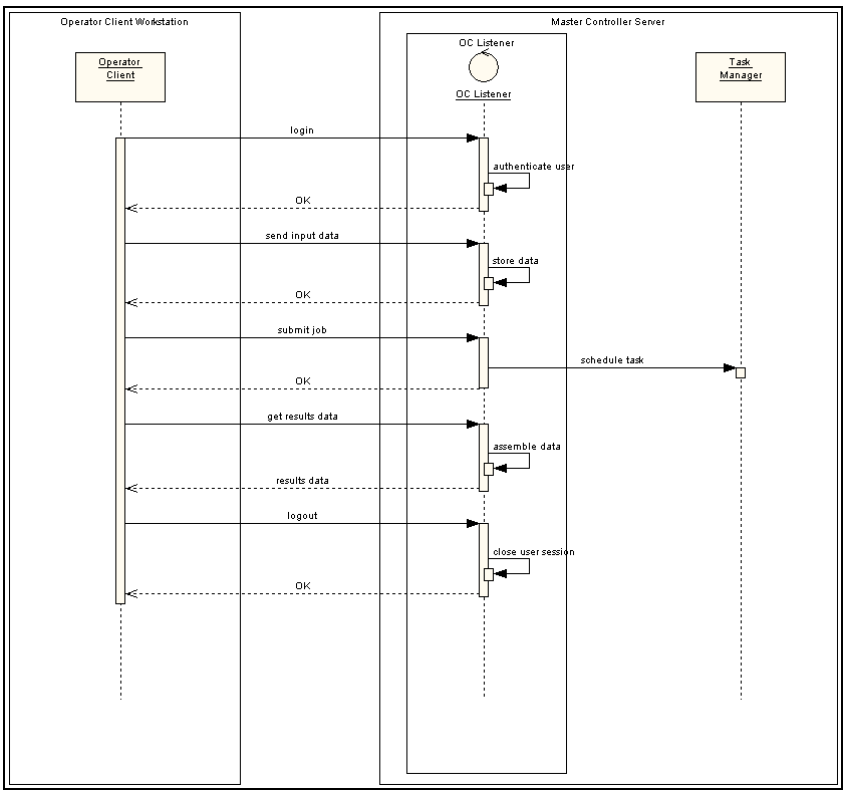


Figure 4-10 Interaction between the Master Controller Server and the Operator Client

4.2.5 Report Generation

NFFS
Within NFFS, the generation of a report is just another task that can be dispatched from the central system to a Shell server. The reports are based on configurable templates and consist of HTML. The reports are set to be produced automatically after a forecast run and transferred to the NFFS web server.

NGMS differences:
Within NGMS the initial requirement is for a manual process of distributing reports. The process is detailed in the steps below:

The user generates a report within the Operator client, based on the report configurations and the datasets that exist in their local data-store.
The generated HTML report is then saved to a specified location and then copied by the user to the appropriate location for inclusion in the appropriate intranet site.

4.2.6 Central Storage

NFFS
A central Oracle database is used to store time series data (and additional system operational specific data). It operates on a "Rolling Barrel" concept; different time series types can have different retention periods (typically 5-30 days). After this period the data is removed from the system. The time series themselves are stored as BLOBS. The Master Controller and central storage components know nothing about the content of these "BLOBS" (other than the meta data that is associated with each blob). They are discrete, indivisible entities, which are only processed by a FEWS client (either the end-user client or on a Module sever).

NGMS differences
The meta-data associated with records will require modification to reflect the different method of working.

Within NGMS there are two main differences to the storage:

- Datasets are split into those that have long lifetimes and are thought to be standard or baseline datasets and those that are user-specific, which may have a much shorter retention period.
- It is likely that each dataset will be tagged with details of the group/role that is allowed access. This change is also required in the local data-stores.

4.2.7 User Access Management and Control

NFFS
Currently with NFFS there is no User Access management or control within the Operator client itself. There is access control to the use of the application and the region that can be selected.

NGMS differences
NGMS requires the addition of access control to datasets stored within the system as well as limiting the functionality they can make use of. A user's membership of a group (and associated role) will determine:

- What activities they can perform via the Operator Client
- Which datasets they can select as part of a module run
- Which datasets they can select as part of any data slicing or post processing tasks
- Which datasets they can download to the local Operator client
- Which data sets they can view and manipulate within the local data-store

Additional data structures to contain:
- User-role mapping
- Role-access rights mapping

In addition, updates are required to a number of the existing data structures to include role access criteria. The configuration and management of the user/role/access permissions will require new user interfaces and associated modules.

The data manipulation routines will also require modification to interrogate access permissions prior to any dataset access.

4.2.8 Archiving

NFFS
In the NFFS, the user manually selects a forecast to be archived. The system automatically identifies all the input time series that were used to create that forecast. This data is then copied and stored as a complete compressed record. This action is performed on a Shell server.

NGMS differences:
A similar process is required for NGMS. However, the type of records to retain and their contents will be different. It is also likely that the interface to select the records to archive will also require modification.

4.3 Low level component interaction scenarios

This section provides a description of how the individual components in the system interact to perform specific scenarios. It lists the components involved and describes their behaviour using text/pseudo-code and appropriate diagrams.

4.3.1 Submitting a task from the operator client

Most of the actions to be performed by the user are covered by the following section. Performing Module runs, data slicing, post processing etc. take the form of:

1. The user making a number of selections via a GUI
2. The client application processing these selections and creating the appropriate task and task properties information (execution template, raw data template etc.).
3. The client submitting the task (and task properties) to the central system for processing. The existing TaskProperties.XML files can be used to contain information on raw data output, data slicing and post processing options.
4. Central system processing and dispatching the task to an appropriate Shell server
5. Shell server executing task (via a module and adapters)
6. Results returned to the central system
7. Client informed of execution status.

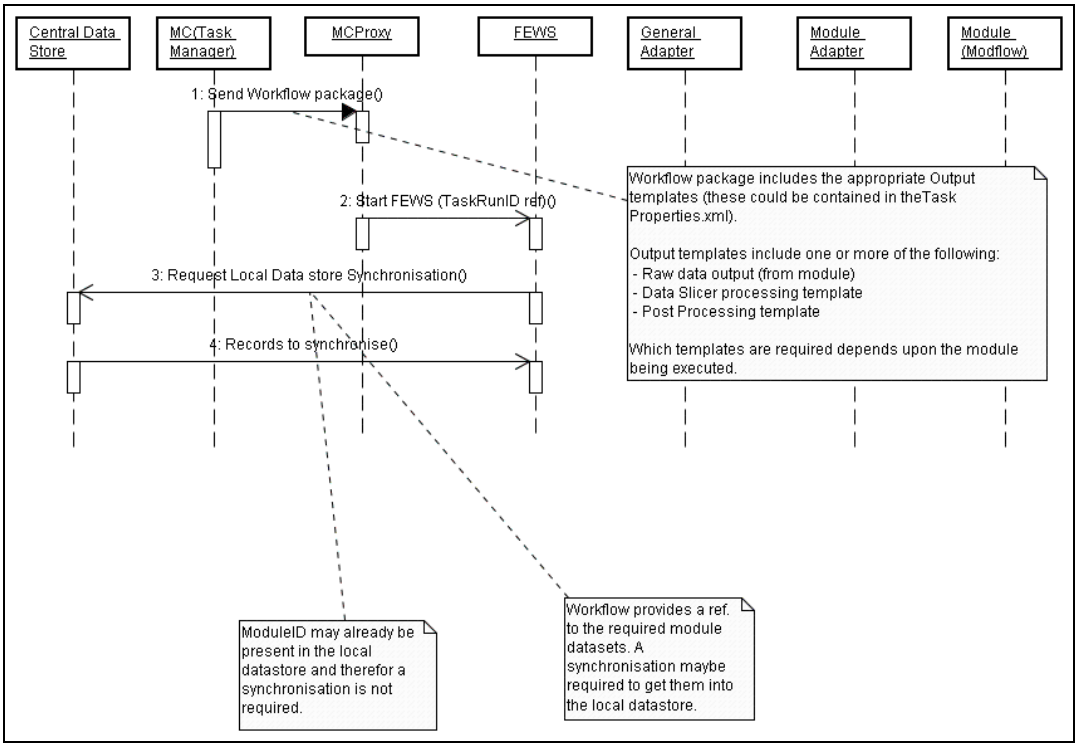


Figure 4-11 Task submission

4.3.2 Module Execution and Control

4.3.2.1 Task Dispatch

The central Master Controller components have the responsibility for scheduling and dispatching tasks to the Shell servers.

The process to dispatch a task to a Shell server is as follows:

1. The request to execute a defined task is sent via JMS from the central system to the Shell Server. This message also contains a "TaskProperties" configuration file, which is currently un-used in NFFS but could be used by NGMS to contain information on data slicing, post processing etc.
2. The MCProxy component responds to the task request and instantiates FEWS with the appropriate reference to the defined task (which has been synchronised from the central to the local data-store at an earlier stage).
3. FEWs processes and executes the associated workflow.

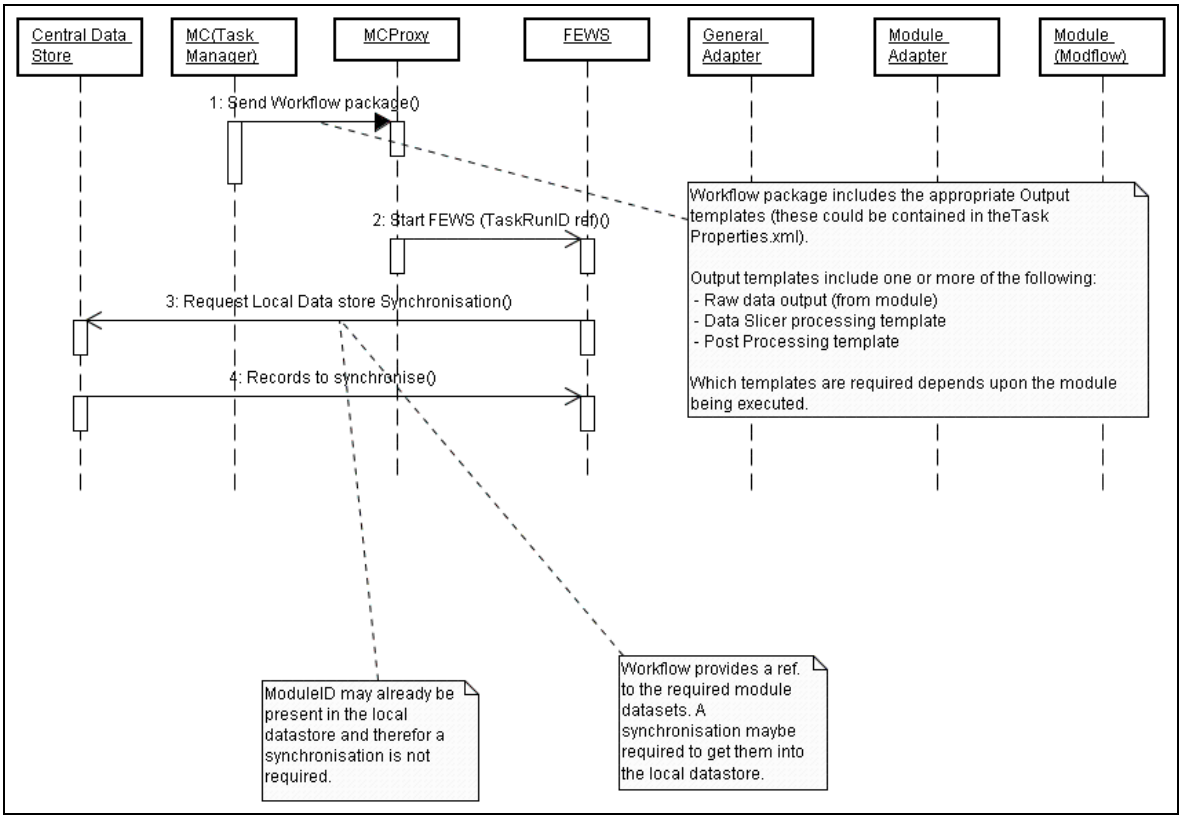


Figure 4-12 Task dispatching

Currently in NFFS, the system assumes that the datasets are already present in the local data-store, i.e. it is not passed an explicit reference to the datasets that are required for the module execution. There may be an enhancement to add a step to provide "re-synchronisation on demand" to minimise the amount of background synchronisation that is required.

4.3.2.2 Task Execution

Task execution takes place primarily on a Shell server. The steps that take place are detailed below:

1. FEWS is instantiated following a Task Dispatch (see above section).
2. FEWS instantiates the General Adapter, passing details of the required Module and Boundary Condition datasets.
3. The General adapter instantiates an appropriate Module Adapter.
4. The module Adapter creates the appropriate input files (in the module's native format) in the required locations for the Module. It may also create the required control file, which sets the execution options for the module.
5. The General Adapter instantiates a ModuleExecutionController (providing details of the module in question, together with a reference to the appropriate module execution control configuration file).
6. The ModuleExecutionController instantiates the required Module (which then loads the required input files and proceeds with execution).
7. Execution monitoring takes place by the ModuleExecutionController (see next section)
8. On Module execution completion (indicated by the ModuleExecutionController), the General Adapter obtains the resulting output data (it is converted from the Modules native format via the Module adapter).
9. FEWS stores the results in the local data-store, which is then subsequently synchronised with the central data-store.

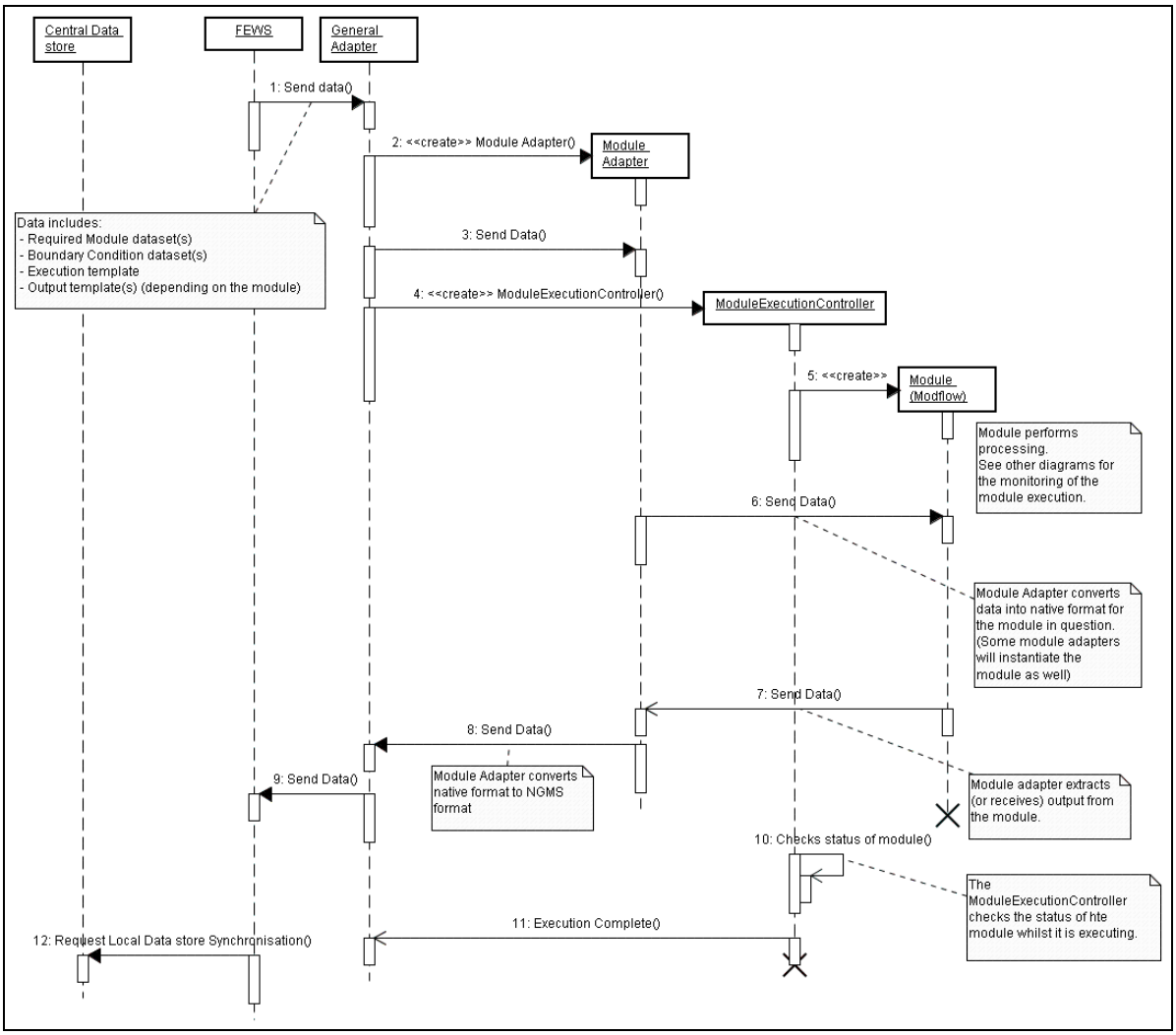


Figure 4-13 Task execution (on a Shell server)

4.3.2.3 Monitoring Module Execution

The ModuleExecutionController checks the execution status of the module. This can be performed on an ad-hoc basis (as a response to a specific request from the client) or as a regular event. Depending upon the Module, this information may be obtained by parsing a defined output/log file or requesting the status directly from the module.

Associated with each Module, is a ModuleExecutationConfiguration XML file. This configuration file contains the information required by the ModuleExecutionController to monitor and control the associated module. It includes the following information:

- Path to module executable
- Path to the Module status file (or runtime logging file)
- Frequency of status checking
- Module status Mask - Mask for status file parsing (i.e. what to look for in the Module status file)
- Module interruption Mask - Mask for status file parsing
- Module interruption – Action to be performed on detecting the interruption pattern.
- Return code to the General Adapter

The ModuleExecutionController communicates with the central system via the direct JMS Status messaging (rather than the method of synchronisation). This allows real-time feedback to the system (and user) of the status of the Module run. This communication mechanism is also used to provide the ability to request the halting of a module run (see next section).

The process of execution monitoring is detailed below:

1. The ModuleExecutionController parses the Modules status output file. The frequency of this parsing together with the criteria of what to extract is contained within the appropriate module execution control configuration file.
2. The status information is fed back to the central system via the new "status message" functionality (this is a direct message rather than via local data store synchronisation).

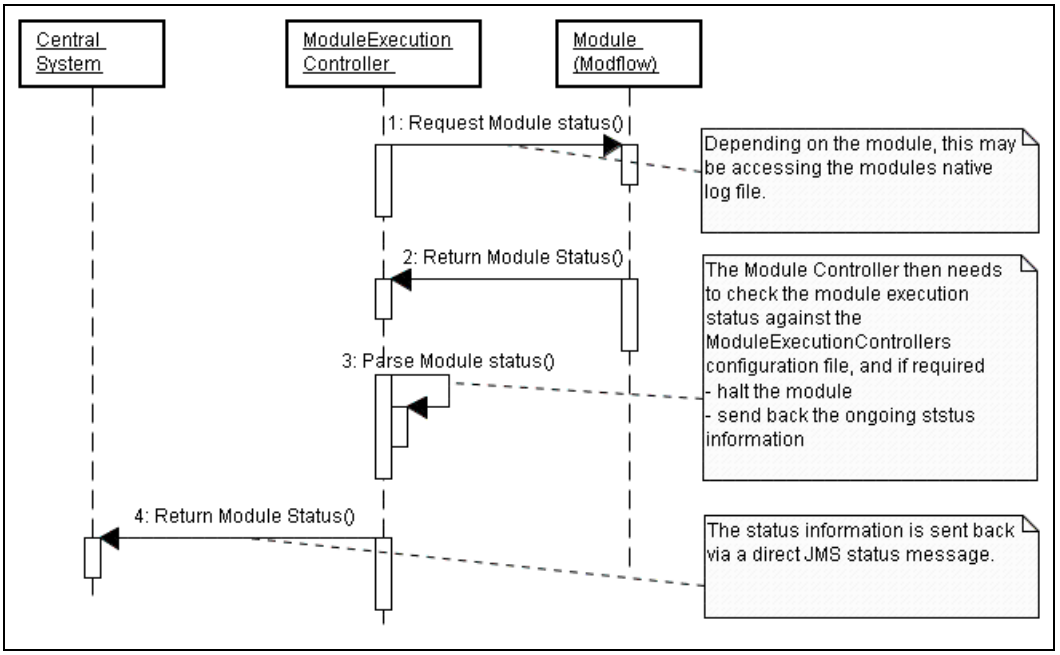


Figure 4-14 Monitoring of module execution

4.3.2.4 Halting Module Execution

At the request of the client (or if the interruption criteria contained within the ModuleExecutationConfiguration file has been met) the ModuleExecutionController will attempt to stop the module. This process may be repeated a number of times.

Depending on the module, it may be possible to perform a "soft" stop, i.e. issue a particular command to halt the execution. Failing this, the ModuleExecutionController will be required to halt the associated process.

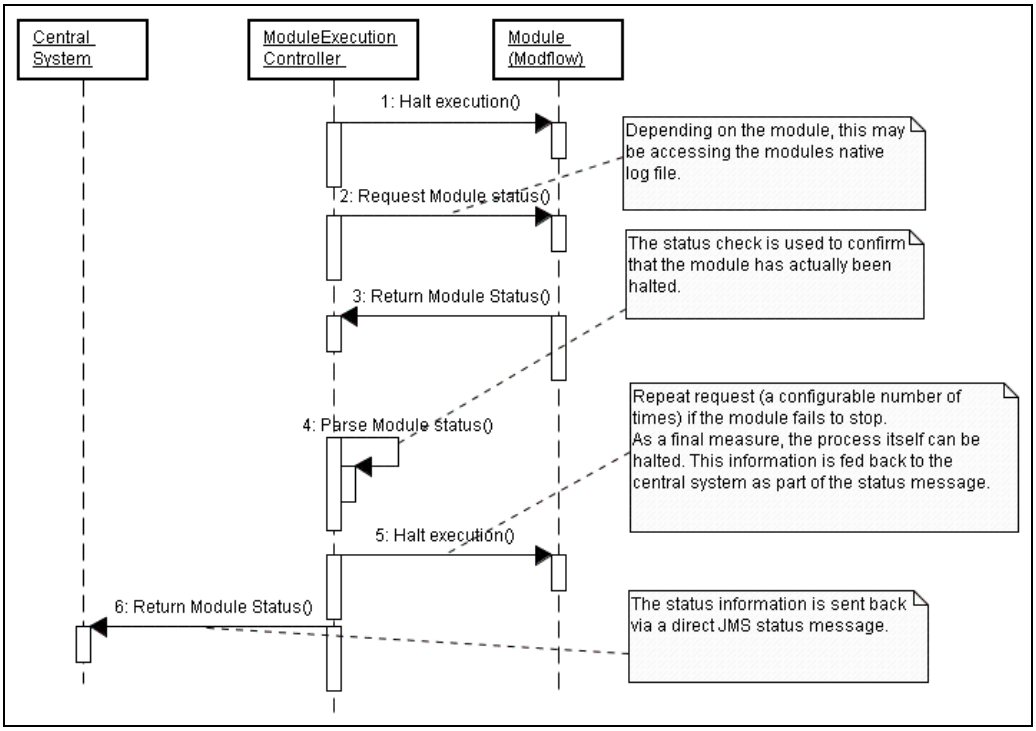


Figure 4-15 Halting of module execution

4.3.3 View dataset on client (selected from local data-store)

The approach is to utilise a dialog to display the contents of the local data-store (in a similar approach to that used by the Forecast Manager in NFFS). The dialog will only display those datasets which the current user has permission to access.

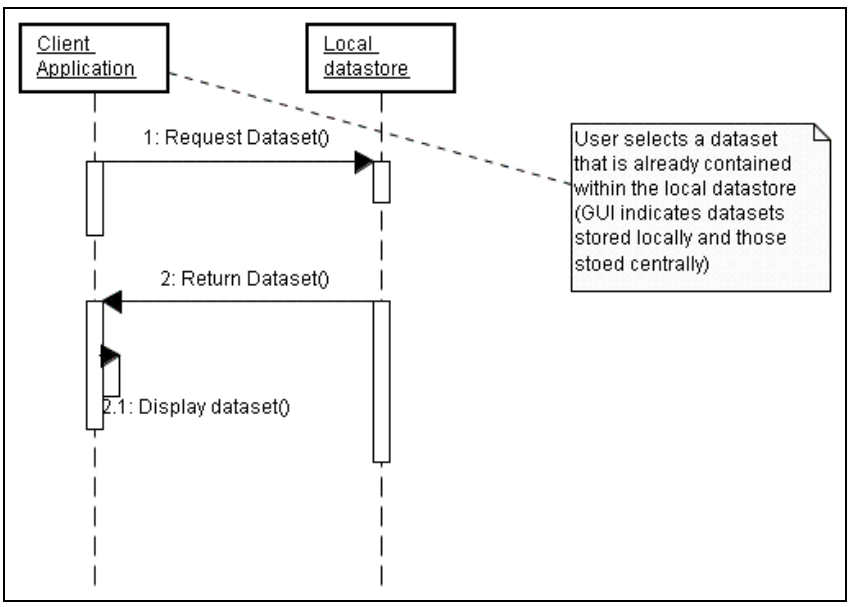


Figure 4-16 Viewing the data set on an Operator Client

4.3.4 View dataset on client (selected from central datastore)

The approach is to utilise a dialog to display the contents of the central data-store (in a similar approach to that used by the Forecast Manager in NFFS). The dialog will only display those datasets which the current user has permission to access.

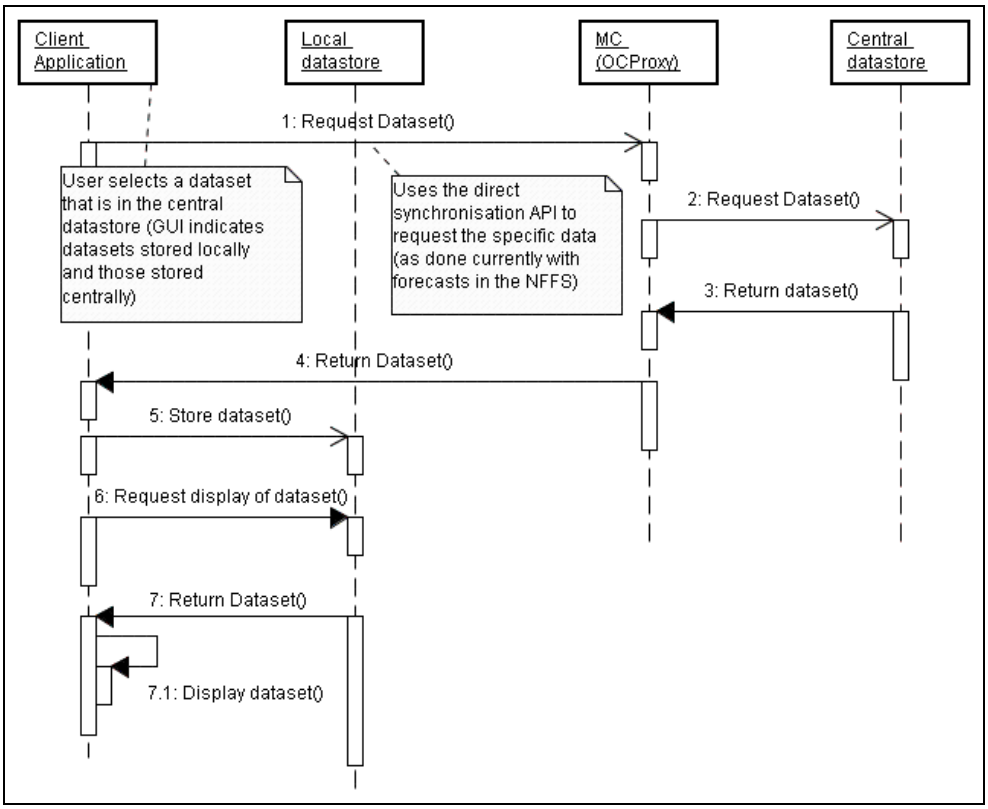


Figure 4-17 View data set by operator Client with data sets residing in central data store

4.4 Handling the data cube in NGMS

4.4.1 Mapping groundwater model data to NFFS formats

The groundwater data cube

Groundwater models typically represent a groundwater system is represented in three dimensions as being a layered system of regular or irregular 2D-grids. Each grid cell holds a number of variables (model parameters, input variables, output variables and state variables), where some of them are static and others are dynamic over time. Most model parameters do vary from grid cell to grid cell.

The semantics of a data value within a groundwater model thus can be identified by:
- its variable, expressed
- its time, expressed in absolute time or erlative time against a T0
- its spatial location (expressed in x,y,z or x,y,layer).

At the moment, the larger models have a horizontal extend of 350 x 350 cells and some 10 layers in the vertical. Regular simulations run 30 years with a monthly time step (i.e. 360 steps). This equals some 441,000,000 data values per parameter (assuming the parameter applies in all layers). With some 10 output parameters, the cube holds some 4 billion data values which is reflected in the size of the output file (few Gigabytes). Zip-compression brings the file size down by a factor 5-10.

As a groundwater modeller is only interested in a portion of this data, data slicing will be applied to reduce the size of data cubes being synchronised between the server and the Operator Clients. However, before discussing the data slicing alternatives, it is required to understand the mapping of the groundwater model data into the NFFS formats.

The available NFFS data formats

To handle those data sets the NFFS/NGMS offers the following data types [see NFFS, DelftFEWS, Published Interfaces, v.2.5]:
- static data (meta data fixed in the configuration)
  - all spatial meta-information:
    - locations (pi_locations.xsd), indicating the x,y,z position of a point locations
    - grid cell centre points (pi_cells.xsd), indicating the x,y,z positions of grid cell centress
    - branches (pi_branch.xsd) indicating the x,y,z, position of the vertices of a polyline feature
    - polygons (pi_polygons.xsd), indicating the x,y,z position of the vertices of a (closed) polygon feature
- dynamic data (mixture of meta data and values):
  - parameters (pi_parameters.xsd), definition fixed, single values may be manipulated by NGMS

- time series
  - time series at predefined locations (pi_timeseries.xsd)
  - map stacks (time series at predefined grids) (meta-data in pi_mapstacks.xsd, values in binary format PCRaster, USGS BIL/BSQ/BIP or ESRI ASCII grid)
  - time/event series of polygons (pi_polygons.xsd) (e.g. to handle flood maps)

Given those capabilities, the NGMS typically will be implemented by handling all numerical data values as a grid time series; even if they are static parameters that vary over space but not over time (e.g. conductivity). Only, module parameters that do not vary over space and time might be dealt with as a static module parameter.

Proposed mapping into a configuration

Within the NFFS/NGMS, the data flow is predefined in the configuration. For each software module, the configuration identifies which data set enters the module and which data set leaves the module [see NFFS, DelftFEWS Configuration Guide, v.0.4 or later]. These data sets are typically defined as TimeSeriesSets where each time series set is identified (i.e. predefined) by:

- a ModuleInstance-identifier, i.e. an identifier to a configured module that produced the data
- a Parameter-identifier
- a Location-identifier, i.e. placeholder for a predefined point, branch, grid, polygon
- a timeSeriesType (external historical, external forecasting, simulated historical, simulated forecasting)
- a time step, possibly a relative time (to T0)

The timeSeriesType acts as a flag indicating how the data set should be handled. The output produced by a groundwater model will in NGMS typically be handled as simulated historical (for previous runs) or simulated forecasting (for the current run).

To identify the data set within a configuration, each parameter will have a timeSeriesSet for each grid layer. provides a typical identification of a timeSeriesSet.

```
<timeSeriesSet>
    <moduleInstanceId>ModflowVKD_WestMidlandsWorfe</moduleInstanceId>
    <valueType>scalar</valueType>
    <parameterId>X-flux.out</parameterId>
    <locationId>AquiferLayer-1</locationId>
    <timeSeriesType>external historical</timeSeriesType>
    <timeStep unit="week" multiplier="52"/>
    <relativeViewPeriod unit="week" start="-48" end="36"/>
    <readWriteMode>read only</readWriteMode>
</timeSeriesSet>
```

Figure 4-18 Illustration of a typical timeSeriesSet identification

The locationID is a reference to any definition of the spatial extent (horizontal and vertical), e.g. a grid cells centre point definition or a polygon definition.

As a grid cell centre definition holds all coordinates (x,y,z) of a grid point, a variety of options is available to define a model grid. For example, a groundwater model with 3 aquifer layers can be defined (i) by one grid cell centre point definition which holds all grid

points, or (ii) by three grid cell centre point definitions, each definition holding one layer and having another locationId for each layer.

In the first case, the model will have one timeSeriesSet per parameter. In the second case, the model will have 3 timeSeriesSets per parameter, with a different locationId for each model layer. In this case, the timeSeriesSet identification of will be have a similar set identified with locationId 'AquiferLayer-2' and one with locationId 'AquiferLayer-3'

4.4.2 Data slicing solution directions

As indicated, the concept of data slicing is proposed to shrink the size of the data sets being synchronized between the Server and the Operator Client while retaining the ability to inspect data at user-defined slices. Data slicing refers to the reduction of the size of output data, by selecting the relevant parameter, vertical extent (i.e. layers), horizontal extent and temporal extent.

Two solution directions have been identified
1. Utilize the NFFS-DataTransformation module and possibly the NFFS-interpolation module to apply user defined data slicing on predefined location definitions
2. Develop a new Data Slicing module which can handle user-defined grid definitions to create a sliced grid time series (e.g. a slice defined as a grid of 35 x 35 cells).

Solution 1 is based on the fact that NFFS has extensive facilities for data manipulation using the DataTransformation module and the Interpolation module (see NFFS Configuration guide, Section 6.2 and 6.3). Using those NFFS-modules, data slicing can be applied if the data manipulations are completely predefined in terms of input and output time series, as well as the data operations to be performed. The input and output is fixed in terms of parameters, locations and time span, but is relative to T0, the current forecast moment.

After intensive assessment, solution 1 has been rejected for the following reasons. Both existing NFFS-modules are rather complex modules, developed and optimized for other purposes. They do not allow flexible instructions to accommodate spatial temporal data slicing, while it will be difficult to accommodate vertical or diagonal slicing. Functionality extension is rejected from the perspective of the installed based (live NFFS in operation) as the code complexity.

Therefore it has been chosen to focus on solution 2, as this allows development of a utility which is optimized for high speed slicing of large data cubes with large flexibility in the spatial and temporal definition of the slice template.

4.4.3 Data Slicer design

The NGMS Data Slicer will be developer as an API that be executed at the Operator Client, the MasterController or the Shell Server. As the Data Slicer is potentially interesting for NFFS as well, it would be nice if it is possible to predefine its instructions in a fixed XML-configuration file.

To combine a predefined XML-configuration file with a flexible slice definition, the following pattern has been applied (see Figure 4-19).

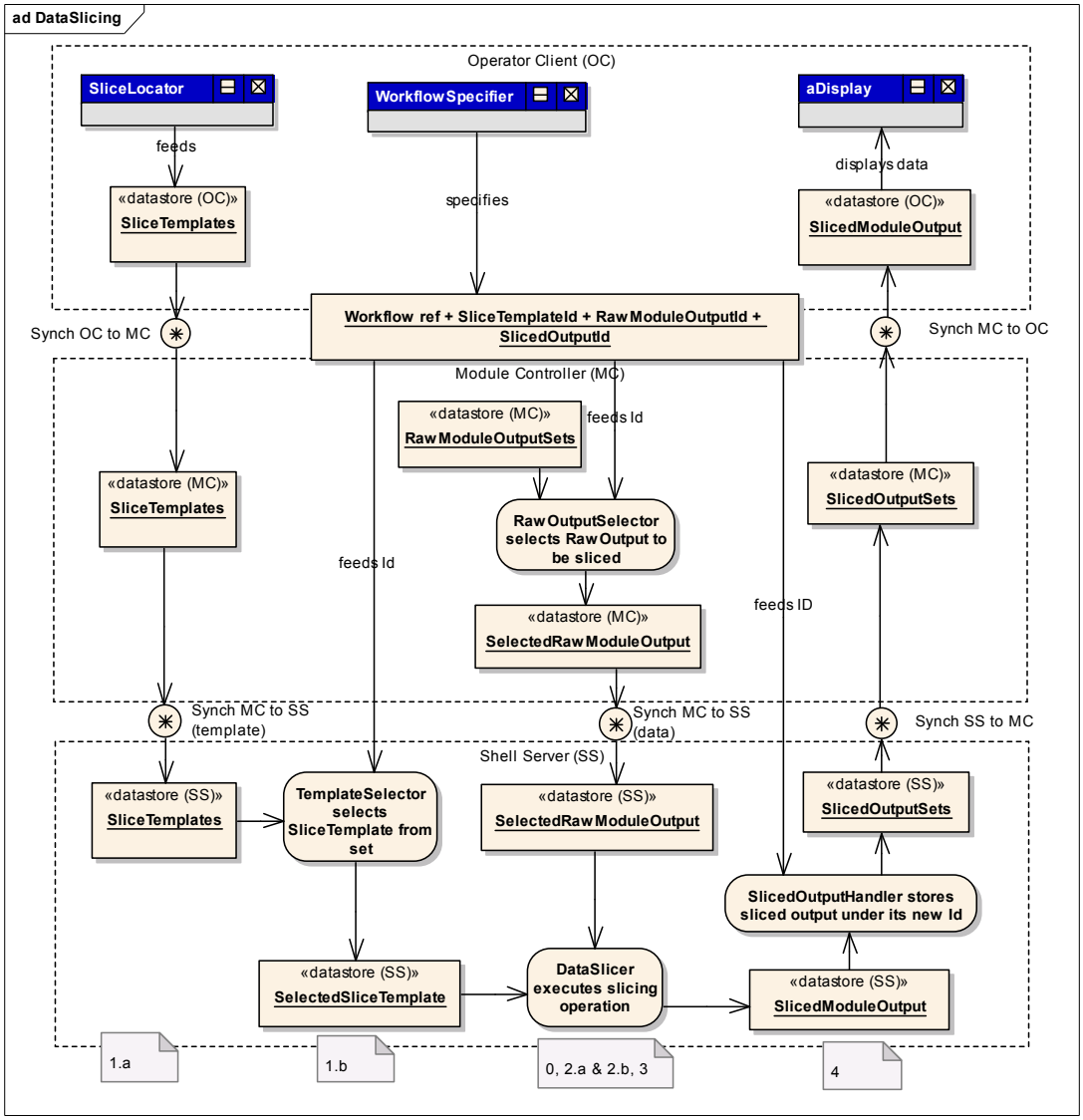


Figure 4-19 Proposed detailed design for data slicing workflow and synchronisation

0. The predefined XML-configuration file of the DataSlicer contains (dummy) placeholders for the RawModuleOutput, the SliceTemplate and the SlicedOutput.
1. a) A GUI-plugin defines a SliceTemplate and adds it to the collection SliceTemplates.
   b) When executing a data slicing workflow, TemplateSelector (an assistant class) will feed the DataSlicer with the correct template by populating the SelectedSliceTemplate placeholder based on the SliceTemplateId it receives as part of the workflows-task properties.
2. a) The database of the NGMS will hold collections of RawModuleOutputSets (i.e. time series) that are produced by one module adapter run
   b) When executing a data slicing workflow, the RawOutputSelector (an assistant class) will feed the DataSlicer with the correct input by populating the

SelectedRawModuleOutput placeholder based on the RawModuleOutputId it receives as part of the workflows-task properties.

3. The DataSlicer can do its job as the dummy placeholders are replaced by references to actual data
4. After the DataSlicer has done its job, an assistant class, the SlicedOutputHandler will label the SlicedModuleOutput with the meta-information held by the SlicedModuleOutputID.

While Figure 4-19 hints that action 2b is executed by an MC-component, this action might well be done by a Shell Server.

4.4.4 Data slicing configuration

With regard to the Data Slicing configuration, the main role is played by so-called Slicing templates. Figure 4-20 provides a possible layout of the associated configuration file. To be considered during implementation is the question "should the sliceExtent be combined with input/output TimeSeriesSet information ?"

If the SlicingTemplate contains a reference to the spatialExtent, this will be no problem. If the spatial extent definition (i.e. the coordinates) are included in this template, it size will grow drastically.

```
<slicingTemplates>
    <slicingTemplate templateID="">
        <InputtimeSeriesSet>
            <inputParameterId="my input parameter"/>
            <locationID="my model grid definition:/>
        <sliceExtent>
            <spatialExtent spatialExtentId="My user defined slice grid definition"/>
                // spatialExtentId refers to a grid definition according to pi_cells.xsd
            <temporalextent temporalExtentId="SliceAreaTemplate">
                <start=""/>
                <end=""/>
                <unit="">
                <multiplier=""/>
            </temporalExtent>
        </sliceExtent>
        <Output>
            <timeSeriesSet>
            <outputParameterId="my input parameter"/>
            <locationId="my user defined slice grid definition"/>
    </slicingTemplate>
<slicingTemplates>
```

Figure 4-20 Possible layout of the Slicing Template XML-configuration file

A Pseudo post-processing scripts

This appendix describes the post-processing scripts that are necessary to transform MODFLOW-output, stored in the NGMS-format, to datasets that can be used to create tables, charts or maps for use in reports.

Starting point is the assumption that all input necessary to run the module and output data created by the module are stored in the NGMS in the NGMS-format.

Table 1 Example of nomenclature of the used parameters

MODFLOW-parameter name	**NGMS-parameter name**	**Report-parameter name**
H1 at time # ??	H1_(t#, all locations) H1_(location ##, all t's)	Groundwater level at time # Groundwater level at Location ##
TRANS_X2 ??	TX2_(all locations)	Transmissivity of aquifer 2
etc.		

Table 2 List of used parameters and symbols

Symbol	**Explanation**
t#	start of time period
t##	end of time period
dt	time step
locL	location L
LocRP	location P of river R
areaA	area A
layerN	number N of layer in groundwater model
diff	difference
2refR	compared to Reference
CBX	Centre of cell-boundary perpendicular to X-direction (grid-orientated)
CBY	Centre of cell-boundary perpendicular to Y-direction (grid-orientated)
<location>id	number of location
_s	sum
_d	direction
_m	magnitude
_t	at a specified time
_t#:t##;dt	over a specified period and for specified time step
_locL	for a specified location
_locRP	for a specified river-location
_areaA	for a specified area
_layerN	for a specified layer
RPs	starting location Ps of a section of river R
RPe	ending location Pe of a section of river R
perc	percentage
count	counter
_L	length
_D	saturated thickness
_V	magnitude of vector
_POR	porosity

Parameter	Explanation
<Par.name>_locL	Value of a parameter for a specified location
<Par.name>_t_locL	Value of a parameter at a specified time for a specified location
<Par.name>_t#:t##;dt_locL	Time serie of values over a specified period for a specified parameter
<Par.name>_average_t#:t##;dt_locL	Average value of a parameter over a specified period for a specified location
<Par.name1>_<Par.name2>_diff_t#:t##;dt_locL	Time serie of values of differences between two parameters for a specified location
<Par.name>_diff2refR_t#:t##;dt_locL	Time serie of values of changes of a parameters compared to a reference condition for a specified location
<Par.name>_sumSV_t#:t##;dt_locRP	Time serie of sum of spatial values over a specified period
Duration curves: DQR_perc_count_locRP DQR_count_locRP	Percentage of exceeded values for a parameter: percentage exceeded value
Transient budgets: QRCH_t_areaA DSTOR_t_areaA QSTR_t_areaA QAB_t_areaA GWFLOW_t_areaA	Time series of sums for a specified area of: recharge change in storage stream flow abstractions groundwater flow
Budgets: QRCH_s_t#:t##;dt_areaA GWFLOW_s_t#:t##;dt_areaA DSTOR_s_t#:t##;dt_areaA QSTR_s_t#:t##;dt_areaA QAB_s_t#:t##;dt_areaA	Sums over a specified period of for a specified area: recharge change in storage stream flow abstractions groundwater flow
Stream accretion: QSTRA_t_locRP	Time serie of sum of river discharge over a specified river-section at a specified time upstream a specified river-location
Winterbourne signature: HSTRW_t_locRP	Situation in which the groundwater level lies above the bottom level of the river (or not) at a specified time for a specified river-location
Vector fields: HGWFLOW_d_t_layerN_locL HGWFLOW_m_t_layerN_locL	Horizontal projection of groundwater flow velocity: direction of projected vector magnitude of projected vector
Flows between GMUs: CBXid_locLid1_locLid2 CBYid_locLid1_locLid2 CBXid_L CBYid_L CBXid_layerN_D CBYid_layerN_D CBXid_layerN_V CBYid_layerN_V CBXid_layerN_POR CBYid_layerN_POR locL_GMU-ID CBXid_GMU CBYid_GMU FLCBXid_GMU_t_N FLCBYid_GMU_t_N FLCBXid_GMU_t FLCBYid_GMU_t GMUFLOW_d_t#:t##;dt_CBGMUid1_CBGMUid2 GMUFLOW_m_t#:t##;dt_CBGMUid1_CBGMUid2	Flows between Groundwater Management Units: ID of centre of boundary perpendicular to X-direction, respectively Y-direction, between locLid1 en locLid2 Length of cell boundary Thickness (saturated) of layerN at cell-boundary Velocity of groundwaterflow in layerN over cell-boundary Porosity of layerN at cell-boundary ID of the GMU at lacation L definition of flow between GMUs at cell-boundary (flow from GMUid1 to GMUid2) flux between GMUs in LayerN at cell-boundary (from GMUid1 to GMUid2) flux between GMUs over all Layers at cell-boundary (from GMUid1 to GMUid2) direction, respectively magnitude of vector of groundwater flow between GMUid1 and GMUid2

Different types of data occur:

- Grid parameters versus point parameters;
- Time series versus constant values.

In order to be able to view a report-parameter on the screen, the following activities have to be undertaken:

- Load report template with parameter name #;
- Load the associated NGMS-data set;
- Load the desired legend-set;
- Show on the screen (map, graph or table).

Two main types of NGMS-parameter sets are distinguished:

1. Direct parameter sets: module input and module output data sets;
2. Derived parameter sets: transformed from module input and module output data sets by means of some sort of calculation.

To create a derived parameter set one or more transformations are necessary. These transformations have to be predefined in post-processing scripts (at least for the time being). The scripts to create the derived parameters are described below.

Table 3 Post-processing scripts to create data sets for reporting purposes

Report parameter	**Necessary script**
Module input parameter (TX2_locL)	*// TX2 consists of values that are constant in time (time – independent)* -
Module output parameter (H1_t#_locL)	*// H1 values at time # for the entire grid* -
Module output parameter: H1_t_locL	*// H1 time serie between time # and time ## with step dt (default 1) at location L* -
Average parameter: H1_average_t#:t##;dt_locL	*// H1 average between time # and time ## with step dt (default 1) for the entire grid* Do for all locL load H1_t#_locL H1_sum_locL = H1_t#_locL count = 1 Do Until t=t## step dt load H1_t_locL H1_sum_locL = H1_sum_locL + H1_t_locL count = count + 1 End do H1_average_t#:t##;dt_locL = H1_sum_locL / count store H1_average_t#:t##;dt_locL End do → aggregate – mean ?
Difference parameter: H1_H2_diff_t_locL	*// difference between H1 and H2 from time # to time ## with step dt (default 1) for the entire grid* Do for all locL Do from t=t# until t=t## step dt load H1_t_locL load H2_t_locL H1_H2_diff_t_locL = H1_t_locL – H2_t_locL store H1_H2_diff_t_locL End do End do →
Difference parameter: H1_diff2refR_t_locL	*// difference in H1 between current scenario and reference set R from time # to time ## with step dt (default 1) for the entire grid* Do for all locL Do from t=t# until t=t## step dt load H1_t_locL load H1_refR_t_locL H1_diff2refR_t_locL = H1_t_locL – H1_refR_t_locL store H1_diff2refR_t_locL End do End do →

Add up spatial values parameter: QR_sumSV_t_locRP	*// river flow (QR) from time # to time ## with step dt (default 1) for location RP (river R point P)* Determine all locations L that are upstream of point RP *// (for both river and drainage locations)* Do from t=t# until t=t## step dt QR_sumSV_t_locRP = 0 QR_RP = 0 QR_L = 0 load QRIV1_t_locRP *// (flux at locRP from GW to RIV in top layer – m^3/d)* etc. *//(flux at locRP from GW to RIV in layers in between –m^3/d)* load QRIVn_t_locRP *// (flux at locRP from GW to RIV in bottom layer – m^3/d)* load QDRAIN_locRP *// (flux at locRP from GW to DRAIN – m/d)* load AREA_locRP *// (area of locRP m2)* QRIV_t_locRP = QRIV1_t_locRP = … + QRIVn_t_locRP QR_RP = QRIV_t_locRP + (QDRAIN_t_locRP * AREA_locRP) Do for all locL upstream RP load QRIV1_t_locL // (flux at locL from GW to RIV in top layer – m^3/d) etc. // (flux at locL from GW to RIV in layers in between – m^3/d) load QRIVn_t_locL // (flux at locL from GW to RIV in bottom layer – m^3/d) load QDRAIN_t_locL // (flux at locL from GW to DRAIN – m/d) load AREA_locL // (area of locL m^2) QRIV_t_locL = QRIV1_t_locL = … + QRIVn_t_locL QR_L = QR_L + QRIV_t_locL + (QDRAIN_t_locL * AREA_locL) End do QR_sumSV_t_locRP = (QR_RP + QR_L) * dt store QR_sumSV_t_locRP End do →
Duration parameters: DQR_perc_count_locRP DQR_count_locRP	*// duration-curve: river flow (QR)-exceedance from time # to time ## with step dt (default 1) for location RP (river R point P)* Determine river flow values *// → (see before: Add up spatial values)* countmax = 0 Do from t=t# until t=t## step dt load QR_t_locRP countmax = countmax + 1 End do count = 0 Do until count = countmax count = count + 1 DQR_perc_count_locRP = 100 * count / countmax determine MaxValue_QR_t_locRP DQR_count_locRP = MaxValue_QR_t_locRP remove MaxValue_QR_t_locRP store DQR_perc_count_locRP store DQR_count_locRP End do →
Transient budget parameters: QRCH_t_areaA DSTOR_t_areaA QSTR_t_areaA QAB_t_areaA GWFLOW_t_areaA	*// transient water budget during period between time # and time ## with step dt (default 1) for predefined area A* Determine all locations L that lie within area A *// (for grid, river and abstraction locations)* *// storage change* Do for all locL within areaA load AREA_locL *// (area of locL – m^2)* load SC1_locL *// (storage coefficient of toplayer at locL)* etc. *// (storage coefficients of layers in between at locL)* load SCn_locL *// (storage coefficient of bottom layer at locL)* load H1_t#-dt_locL *// (head in top layer at locL at begin of period – m+ref.)* etc. *// (heads of layers in between at locL at begin of period –m+ref.)* load Hn_t#-dt_locL *// (head in bottom layer at locL at begin of period – m)* End do Do from t=t# until t=t## step dt DSTORE_t_areaA = 0 Do for all locL within areaA load H1_t_locL *// (head in top layer at locL at timestep t – m+ref.)*

	etc. // *(heads of layers in between at locL at timestep t – m+ref.)*
	load Hn_t_locL // *(head in bottom layer at locL at timestep t – m+ref.)*
	DSTOR1_t_locL = SC1_locL * (H1_t_locL – H1_t-dt_locL) * AREA_locL
	etc.
	DSTORn_t_locL = SCn_locL * (Hn_t_locL – Hn_t-dt_locL) * AREA_locL
	DSTOR_t_locL = DSTOR1_t_locL + … + DSTORn_t_locL
	DSTOR_t_areaA = DSTOR_t_areaA + DSTOR_t_locL
	End do
	store DSTOR_t_areaA
	End do
	// *stream flow* // *(! more than 1 river at same locL possible !)*
	Do from t=t# until t=t## step dt
	QSTR_t_areaA = 0
	Do for all locL within areaA
	load QRIV1_t_locL // *(flux at locL from GW to RIV in layer 1 – m^3/d)*
	etc. // *(flux at locL from GW to RIV in layers in beteen – m^3/d)*
	load QRIVn_t_locL // *(flux at locL from GW to RIV in layer n – m^3/d)*
	QSTR_t_locL = (QRIV1_t_locL + … + QRIVn_t_locL) * dt
	QSTR_t_areaA = QSTR_t_areaA + QSTR_t_locL
	End do
	store QSTR_areaA
	End do
	// abstractions // *(! more than 1 abstraction at same locL possible !)*
	Do from t=t# until t=t## step dt
	QAB_t_areaA = 0
	Do for all locL within areaA
	load QAB1_t_locL // *(abstraction at locL in toplayer – m^3/d)*
	etc. // *(abstraction at locL in layers in between – m^3/d)*
	load QABn_t_locL // *(abstraction at locL in bottom layer – m^3/d)*
	QAB_t_locL = (QAB1_t_locL + … + QABn_t_locL) * dt
	QAB_t_areaA = QAB_t_areaA + QAB_t_locL
	End do
	store QAB_t_areaA
	End do
	recharge
	Do from t=t# until t=t## step dt
	QRCH_t_areaA = 0
	Do from t=t# until t=t## step dt
	load QRCH_t_locL // *(recharge at locL – m/d)*
	QRCH_t_areaA = QRCH_t_areaA + QRCH_t_locL * AREA_locL * dt
	End do
	store QRCH_t_areaA
	End do
	// groundwater flow // *(closing the water budget – m^3)*
	Do from t=t# until t=t## step dt
	load QRCH_t_areaA
	load DSTOR_t_areaA
	load QSTR_t_areaA
	load QAB_t_areaA
	GWFLOW_t_areaA = QRCH_t_areaA – DSTOR_t_areaA – QSTR_t_areaA – QAB_t_areaA
	store GWFLOW_t_areaA
	End do
	→

Budget parameters: QRCH_s_t#:t##;dt_areaA GWFLOW_s_t#:t##;dt_areaA DSTOR_s_t#:t##;dt_areaA QSTR_s_t#:t##;dt_areaA QAB_s_t#:t##;dt_areaA	*// water budget over period between time # and time ## with step dt (default 1) for predefined area A* Determine all locations L that lie within area A // *(for grid, river and abstraction locations)* storage change DSTOR_s_areaA = 0 Do for all locL within areaA load AREA_locL // *(area of locL – m^2)* load SC1_locL // *(storage coefficient of toplayer at locL)* etc. // *(storage coefficients of layers in between at locL)* load SCn_locL // *(storage coefficient of bottom layer at locL)*

	load H1_t#-dt_locL // *(head in toplayer at locL at begin of period – m m+ref.)*
	load H1_t##_locL // (head in toplayer at locL at end of period – m m+ref.)
	etc. // *(begin- and end-heads of layers in between at locL – m+ref.)*
	load Hn_t#-dt_locL // *(head in bottom layer at locL at begin of period – m m+ref.)*
	load Hn_t##_locL // *(head in bottom layer at locL at end of period – m m+ref.)*
	DSTOR1_locL = SC1_locL * (H1_t##_locL – H1_t#-dt_locL) * AREA_locL
	etc.
	DSTORn_locL = SCn_locL * (Hn_t##_locL – Hn_t#-dt_locL) * AREA_locL
	DSTOR_locL = DSTOR1_locL + … + DSTORn_locL
	DSTOR_s_areaA = DSTOR_s_areaA + DSTOR_locL
	End do
	store DSTOR_s_t#:t##;dt_areaA
	// stream flow // *(! more than 1 river at same locL possible !)*
	QSTR_s_t#:t##;dt_areaA = 0
	Do for all locL within areaA
	QSTR_locL = 0
	Do from t=t# until t=t## step dt
	load QRIV1_t_locL // *(flux at locL from GW to RIV in layer 1 – m^3/d)*
	etc. // *(flux at locL from GW to RIV in layers in beteen – m^3/d)*
	load QRIVn_t_locL // *(flux at locL from GW to RIV in layer n – m^3/d)*
	QSTR_locL = QSTR_locL + (QRIV1_t_locL + … + QRIVn_t_locL) * dt
	End do
	QSTR_s_t#:t##;dt_areaA = QSTR_s_t#:t##;dt_areaA + QSTR_locL
	End do
	store QSTR_s_t#:t##;dt_areaA
	// abstractions // *(! more than 1 abstration at same locL possible !)*
	QAB_s_t#:t##;dt_areaA = 0
	Do for all locL within areaA
	QAB_locL = 0
	Do from t=t# until t=t## step dt
	load QAB1_t_locL // *(abstraction at locL in toplayer – m^3/d)*
	etc. // *(abstraction at locL in layers in between – m^3/d)*
	load QABn_t_locL // *(abstraction at locL in bottom layer – m^3/d)*
	QAB_locL = QAB_locL + (QAB1_t_locL + … + QABn_t_locL) * dt
	End do
	QAB_s_t#:t##;dt_areaA = QAB_s_t#:t##;dt_areaA + QAB_locL
	End do
	store QAB_s_t#:t##;dt_areaA
	// recharge
	QRCH_s_t#:t##;dt_areaA = 0
	Do for all locL within areaA
	QRCH_locL = 0
	Do from t=t# until t=t## step dt
	load QRCH_t_loc // *(recharge at locL – m/d)*
	QRCH_locL = QRCH_locL + QRCH_t_locL * AREA_locL * dt
	End do
	QRCH_s_t#:t##;dt_areaA = QRCH_s_t#:t##;dt_areaA + QRCH_locL
	End do
	store QRCH_s_t#:t##;dt_areaA
	// groundwater flow // *(closing the water budget – m^3)*
	load QRCH_s_t#:t##;dt_areaA
	load DSTOR_s_t#:t##;dt_areaA
	load QSTR_s_t#:t##;dt_areaA
	load QAB_s_t#:t##;dt_areaA
	GWFLOW_s_areaA = QRCH_s_t#:t##;dt_areaA – DSTOR_s_t#:t##;dt_areaA
	– QSTR_s_t#:t##;dt_areaA – QAB_s_t#:t##;dt_areaA
	store GWFLOW_s_t#:t##;dt_areaA
	→

Stream accretion parameter: QSTRA_t_locRP	*// stream accretion over period between time # and time ## with step dt (default 1) for predefined river R starting from river point RPs, ending at river point RPe* Determine in downstream order all points P that lie on river R starting from RPs ending at RPe Do from t=t# until t=t## step dt QSTRA_t_locRP = 0 Do from locRPs for all locRP until locRPe

	load QRIV1_t_locRP // *(flux at locRP from GW to RIV in layer 1 – m^3/d)* etc. // *(flux at locRP from GW to RIV in layers in between–m^3/d)* load QRIVn_t_locRP // *(flux at locRP from GW to RIV in layer n– m^3/d)* QSTR_t_locRP = (QRIV1_t_locRP + … + QRIVn_t_locRP) * dt QSTRA_t_locRP = QSTRA_t_locRP + QSTR_t_locRP store QSTRA_t_locRP End do End do →
Winterbourne signature parameter: HSTRW_t_locRP	*// winterbourne signature over period between time # and time ## with step dt (default 1) for predefined river R starting from river point RPs, ending at river point RPe* Determine in downstream order all points P that lie on river R starting from RPs ending at RPe Do from t=t# until t=t## step dt HSTRW_t_locRP = 0 Do from locRPs for all locRP until locRPe load H1_t_locRP // *(groundwater level at locRP – m+ref.)* load RIVBOT_locRP // *(level of river bottom of rivR at locRP – m+ref.)* if H1_t_locRP > RIVBOT_locRP then HSTRW_t_locRP = 1 else HSTRW_t_locRP = 0 store HSTRW_t_locRP End do End do →
Vector field parameters: horizontal projection HGWFLOW_d_t_layerN_locL HGWFLOW_m_t_layerN_locL	*// horizontal projection of groundwater flow over period between time # and time ## with step dt (default 1) in layer N at location L* Determine all locations L for which the vector fields must be determined (this can be the entire grid) Do from t=t# until t=t## step dt Do for all selected locL Do for all layerN *// vector fields → to be determined by a MODFLOW-module (e.g. MODPATH)* store HGWFLOW_d_t_layerN_locL // *(direction of vector)* store HGWFLOW_m_t_layerN_locL // *(magnitude of vector)* End do End do End do →
Flows between GMUs parameters: GMUFLOW_d_t#:t##;dt_CBGMUid1_CBGMUid2 GMUFLOW_m_t#:t##;dt_CBGMUid1_CBGMUid2	*// groundwater flow over the border of Groundwater Management Units GMU over period between time # and time ## with step dt (default 1)* Determine vector fields for all layerN // → *(see before: Vector fields horizontal projection)* *// centres of cell boundaries* Determine the locations and IDs of the centres of the cell boundaries: // *(X and Y are grid-orientated)* CBXid_locLid1_locLid2 = ID of centre of boundary perpendicular to X-direction between locLid1 en locLid2 CBYid_locLid1_locLid2 = ID of centre of boundary perpendicular to Y-direction between locLid1 en locLid2 *// (for cell boundaries on model boundaries one of the locIDs will be "no cell")* Recalculate the vector fields from the cell centres to the cell boundary centres //→ *may be allready done by a MODFLOW-module, if not by a standard interpolation routine* *// length of cell boundary segments* Do for all CBXid Determine CBXid_L // *(the length of the boundary)* store CBXid_L End do Do for all CBYid Determine CBYid_L // *(the length of the boundary)* store CBYid_L End do *// calculate parameter values in centre of cell boundaries* Do for all layerN Do for all CBXid Calculate CBXid_layerN_D // *(the thickness of the layerN)* store CBXid_layerN_D

```
            Calculate CBXid_layerN_V          // (the magnitude of the vector in X-direction)
            store CBXid_layerN_V
            Calculate CBXid_layerN_POR        // (the porosity of the layerN)
            store CBXid_layerN_POR
        End do
        Do for all CBYid
            Calculate CBYid_layerN_D          // (the thickness of the layerN)
            store CBYid_layerN_D
            Calculate CBYid_layerN_V          // (the magnitude of the vector in Y-direction)
            store CBYid_layerN_V
            Calculate CBYid_layerN_POR        // (the porosity of the layerN)
            store CBYid_layerN_POR
        End do
    End do
    // assign GMU-ID to cell centres
    Do for all locL
        Determine for all cells the ID of the GMU the centre of the cell is situated in
        store locL_GMU-ID                 // (cells that don't belong to a GMU get value 0)
    End do
    // select only cell boundaries are part of GMU-boundary
    Determine the cell boundaries that lie between two different GMUs or between a GMU and
    a non-GMU-area
    Do for all CBXid
        if locLid1= "no cell" then
            CBXid_GMU = 0"_to_"locLid2_GMU-ID
        else if locLid2 = "no cell" then
            CBXid_GMU = locLid1_GMU-ID"_to_"0
        else
            CBXid_GMU = locLid1_GMU-ID"_to_"locLid2_GMU-ID
        end if
        if locLid1_GMU-ID = locLid2_GMU-ID then
            store CBXid_GMU
        end if
    End do
    Do for all CBYid
        if locLid1= "no cell" then
            CBYid_GMU = 0"_to_"locLid2_GMU-ID
        else if locLid2 = "no cell" then
            CBYid_GMU = locLid1_GMU-ID"_to_"0
        else
            CBYid_GMU = locLid1_GMU-ID"_to_"locLid2_GMU-ID
        end if
            if locLid1_GMU-ID = locLid2_GMU-ID then
        store CBYid_GMU
        end if
    End do
    // determine boundary fluxes for each cell boundary that is part of a GMU-boundary, for
    each timestep and for each layer
    Do from t=t# until t=t## step dt
        Do for all CBXid_GMU
            FLCBXid_GMU_t = 0
            Do for all layerN
                load CBXid_L                  //(length of boundary)
                load CBXid_layerN_D           // (thickness of layerN)
                load CBXid_layerN_V           // (the magnitude of the vector in X-direction)
                load CBXid_layerN_POR         // (porosity of layerN)
                FLCBXid_GMU_t_N = (CBXid_L * CBXid_layerN_D * CBXid_layerN_V
                                                        * CBXid_layerN_POR) * dt
                store FLCBXid_GMU_t_N
                FLCBXid_GMU_t = FLCBXid_GMU_t + FLCBXid_GMU_t_N
            End do
            store FLCBXid_GMU_t
        End do
        Do for all CBYid_GMU
            FLCBYid_GMU_t = 0
```

	Do for all layerN load CBYid_L // *(length of boundary)* load CBYid_layerN_D // *(thickness of layerN)* load CBYid_layerN_V // *(the magnitude of the vector in X-direction)* load CBYid_layerN_POR // *(porosity of layerN)* FLCBYid_GMU_t_N = (CBYid_L * CBYid_layerN_D * CBYid_layerN_V * CBYid_layerN_POR) * dt store FLCBYid_GMU_t_N FLCBYid_GMU_t = FLCBYid_GMU_t + FLCBYid_GMU_t_N End do store FLCBYid_GMU_t End do End do *// determine boundary fluxes for each cell boundary that is part of a GMU-boundary* Calculate the average values over the period t# to t## // → *(see before: average parameter)* Add the values of the flows between the same GMUs // → *(see before: add up spatial values)* *// presentation of GMU-fluxes at centre of each boundary between two GMUs* Detemine the centre of the boundary between two GMUs store the calculated (added) values at these locations: store GMUFLOW_d_t#:t##;dt_CBGMUid1_CBGMUid2 *//(direction of flow)* store GMUFLOW_m_t#:t##;dt_CBGMUid1_CBGMUid2 // *(magnitude of flow)* →