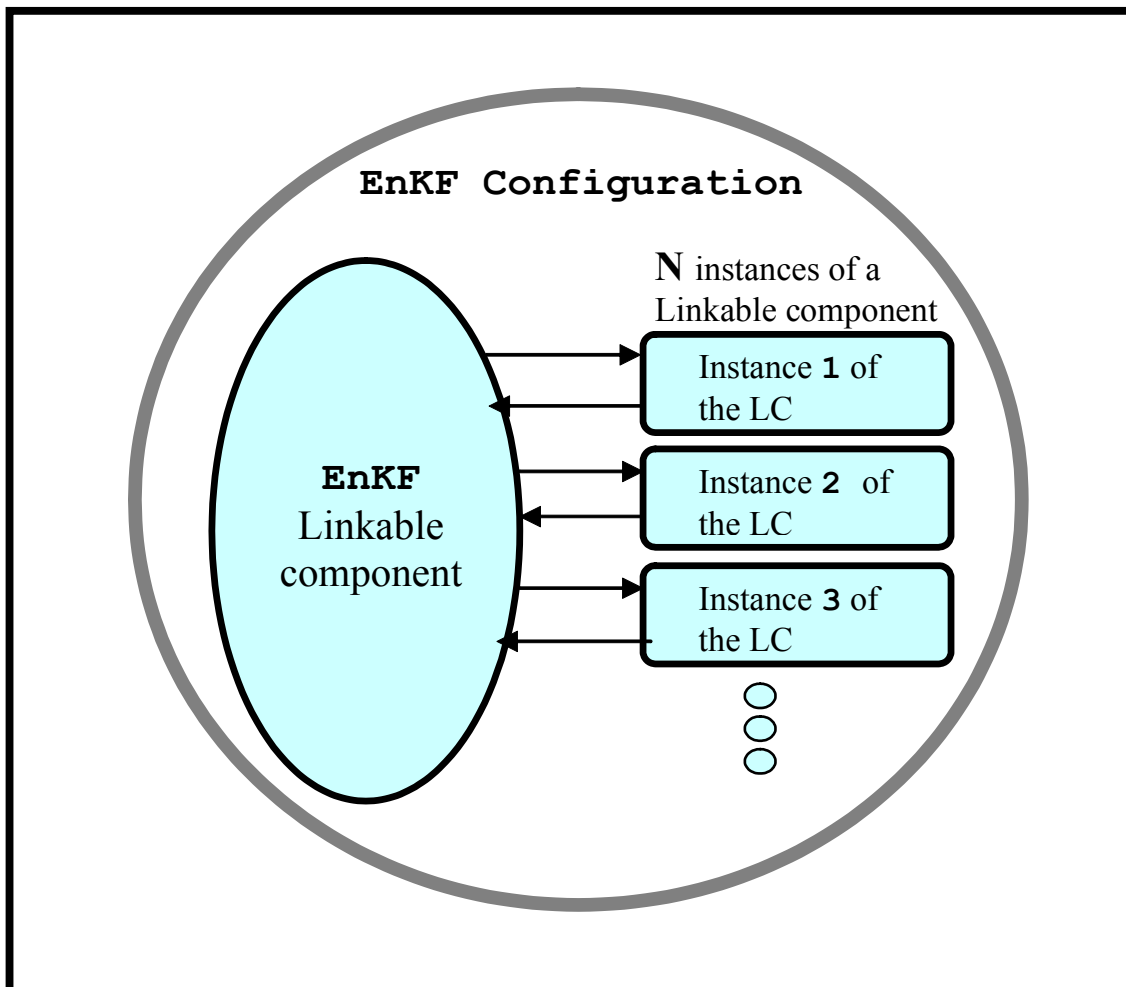


UNESCO-IHE INSTITUTE FOR WATER EDUCATION



A Generic Software Tool for OpenMI-Compliant Ensemble Kalman Filtering

Solomon Dagnachew Seyoum

MSc Thesis WSE-HI.05-04
April 2005

A Generic Software Tool for OpenMI-Compliant Ensemble Kalman Filtering

Master of Science Thesis
by
Solomon Dagnachew Seyoum

Supervisors
Prof. Dr. Ir. A. E. Mynett (UNESCO-IHE)
Ir. S. Hummel (WL | Delft Hydraulics)

Examination committee
Prof. Dr. Ir. A. E. Mynett (UNESCO-IHE), Chairman
Dr. Ir. P. Gijsbers (WL | Delft Hydraulics)
Dr. Z. Vojinovic (UNESCO-IHE)

This research is done for the partial fulfilment of requirements for the Master of Science degree at the UNESCO-IHE Institute for Water Education, Delft, the Netherlands

Delft
April 2005

This thesis is copyright material. It may not be reproduced by any means, in full or in part, except for short extracts in fair dealing; for research or private study, critical scholarly review or discourse, with an acknowledgment, without the prior consent of UNESCO-IHE and WL | Delft Hydraulics on behalf of the author.

The findings, interpretations and conclusions expressed in this study do neither necessarily reflect the views of the UNESCO-IHE Institute for Water Education, nor of the individual members of the MSc committee, nor of their respective employers.

This thesis is dedicated to my parents, brothers and friends.
Thank you for your moral support and inspirations.

Abstract

Integrated water resources management has created a need to understand and model catchment processes and particularly their interactions. Since the processes involved in catchment hydrology are complex and no single model or modelling system can adequately represent all the processes in the catchment environment, a set of distinct individual models are required to be interconnected. To address this complex and fundamental issue of model linking, the HarmonIT project, a research project funded by the European Commission, is developing and implementing a European Open Modelling Interface and Environment (OpenMI).

Ensemble Kalman filter (EnKF) is one of the most well known data assimilation methods and it has been widely used in reducing uncertainty in hydrologic model forecasts. This is done in most of the models in ad hoc basis in a batch file or by implementing the EnKF in a dedicated way in modelling packages. OpenMI offers the mechanisms for data exchange between models at run time. The EnKF algorithm treated as a separate model may be linked with other models in the OpenMI environment for the data assimilation process. The suitability of the OpenMI environment for the application of ensemble Kalman filtering has been explored in this study.

Application of EnKF requires as many computations of the model as the number of ensembles at a time step. Therefore, many instances of the model for which the ensemble Kalman filtering is performed, need to be linked with the EnKF. Currently this kind of linking is not supported by OpenMI without an additional controlling feature. Therefore an additional controlling feature, the EnKFConfiguration class, has been developed within this thesis to configure an OpenMI composition for EnKF application.

The configuration class has been used to configure an OpenMI composition for EnKF and a rainfall-runoff model called HYMOD-RR. Both the models, HYOD-RR and EnKF, are made OpenMI compliant using the wrapper provided by the OpenMI environment to facilitate the migration of legacy models to OpenMI environment.

The composition has been tested and proved that the results are the same as the batch file method and has advantage of speed and ease of customizing for other models.

Keywords:

OpenMI, OpenMI Composition for EnKF, EnKFConfiguration, OpenMI compliant

Acknowledgements

I would like to extend my sincere appreciation to Prof. Dr. Ir. A.E. Mynett, my first supervisor, for his guidance and inspiration that led to the completion of this work.

I am highly indebted to Ir. S. Hummel for his invaluable advice, comments and unreserved help whenever needed. Without him the work could not be realized.

I also wish to express my gratitude to all who helped me, in one way or the other, in carrying out this study. The following are some of the many;

- The Netherlands Fellowship Program and the EU HarmonIT consortium for granting me the scholarships for the Master of engineering and the Master of Science programs respectively;
- Prof. R. K. Price, Dr. D. P. Solomatine and the entire staff members of Hydroinformatics department of UNESCO-IHE for their unreserved help and moral support;
- WL | Delft Hydraulics for providing me comfortable working environment;
- My class mates, Khalid, Yohan, Elona and all the rest for all the good and bad times we have passed together and for sharing with me culture and history of their country;
- Dr. Ghada El Sarafy for helping me to understand the ensemble Kalman filter;
- The staff members of S&O department of WL | Delft Hydraulics;

Last but not least I would like to thank my parents, brothers and friends back home who have been encouraging my academic undertakings with prayers and moral inspiration.

Solomon Dagnachew Seyoum
Delft, The Netherlands

Table of Content

ABSTRACT	I
ACKNOWLEDGEMENTS	III
LIST OF FIGURES.....	VII
1 INTRODUCTION.....	1
1.1 BACKGROUND.....	1
1.2 RESEARCH SCOPE.....	2
1.2.1 <i>Uncertainty in modelling and data assimilation</i>	2
1.2.2 <i>Rainfall-runoff modelling and EnKF</i>	4
1.2.3 <i>OpenMI and Ensemble Kalman filtering</i>	4
1.3 OBJECTIVES OF THE STUDY	4
1.4 RESEARCH APPROACH	5
1.5 ORGANIZATION OF THE THESIS.....	5
2 OPENMI.....	7
2.1 INTRODUCTION.....	7
2.2 REQUIREMENTS.....	9
2.3 OVERALL ARCHITECTURE AND LAYERING.....	10
2.4 HOW OPENMI ADDRESSES GENERAL ISSUES OF MODEL LINKAGE	11
2.4.1 <i>Data definition</i>	11
2.4.2 <i>Meta data defining potentially exchangeable data</i>	13
2.4.3 <i>Generic model access</i>	13
2.4.4 <i>Data transfer</i>	15
2.4.5 <i>Definition of actually exchanged data</i>	16
2.4.6 <i>Event mechanism</i>	17
2.5 CONSEQUENCES OF THE OPENMI ARCHITECTURE FOR A MODEL.....	18
2.6 OPENMI: AN INTERFACE BASED OPEN STANDARD TO LINK MODELS.....	19
3 DESCRIPTION OF THE HYMOD-RR MODEL	21
3.1 THE HYMOD STRUCTURE.....	21
3.2 THE STATE VARIABLES.....	22
4 ENSEMBLE KALMAN FILTER	27
4.1 INTRODUCTION.....	27
4.2 THE DISCRETE KALMAN FILTER	27
4.3 THE EXTENDED KALMAN FILTER.....	30
4.4 THE ENSEMBLE KALMAN FILTER.....	31

5	METHODOLOGY	35
5.1	APPROACH	35
5.2	MAKING LEGACY MODELS OPENMI COMPLIANT.....	35
5.3	EXCHANGE ITEMS BETWEEN HYMOD AND THE ENKF	39
5.4	CONFIGURING AN OPENMI COMPOSITION FOR ENKF	41
6	RESULTS AND DISCUSSION	43
6.1	MAKING THE MODELS LINKABLE COMPONENTS.....	43
6.1.1	<i>HYMOD</i>	43
6.1.2	<i>EnKF</i>	44
6.2	CONFIGURING OPENMI FOR ENKF	46
6.3	DISCUSSION OF THE RESULTS	48
7	CASE STUDY	51
7.1	THE DATA	51
7.2	APPLICATION OF THE ENKF.....	53
8	SUMMARY, CONCLUSIONS AND RECOMMENDATIONS.....	59
8.1	SUMMARY.....	59
8.2	CONCLUSIONS.....	60
8.3	RECOMMENDATIONS.....	61
8.3.1	<i>Recommendations for further study</i>	61
8.3.2	<i>Recommendations for the HarmonIT project</i>	62
	REFERENCES	63
APPENDIX A	DATA DEFINITIONS IN THE OPENMI INTERFACE SPECIFICATION.....	65
APPENDIX B	OTHER INTERFACES OF THE OPENMI	66
APPENDIX C	ORG.OPENMI.STANDARD API-SPECIFICATION	67
APPENDIX D	ORG.OPENMI.UTILITIES.WRAPPER PACKAGE	82
APPENDIX E	THE ORG.OPENMI.UTILITIES.ADVANCEDCONTROL PACKAGE	86

List of Figures

Figure 2.1 Model Application Pattern	8
Figure 2.2 OpenMI architecture.....	11
Figure 2.3 Different chain layouts with the pull-mechanisms.....	16
Figure 3.1 The HYMOD Model structure	22
Figure 4.1 A complete picture of the operation of the Kalman filter	30
Figure 4. 2 The procedure of EnKF	32
Figure 5.1 The relation between the OpenMI components and the Engine core.....	36
Figure 5.2 The EngineAPiAccess interface	37
Figure 5.3 The SmartWrapper class.....	37
Figure 5.4 Typical Model Architecture when using OpenMI.....	38
Figure 5.5 Changes to the engine core.....	39
Figure 5.6 Exchange items between HYMOD model and EnKF	40
Figure 6.1Relation among the wrapper the EngineAPiAccess and the engine core for Linkable HYMOD	44
Figure 6.2 Linkable HYMOD in the OpenMI Configuration Editor User interface	45
Figure 6.3 Relation among the wrapper the EngineAPiAccess and the engine core for Linkable EnKF	46
Figure 6.4 Schematic diagram of the EnKF Configuration	47
Figure 6.5 Sequence diagram of the EnKF Configuration	47
Figure 6.6 EnKF Configuration in the OpenMI Configuration Editor User interface.....	48
Figure 7.1 Observed versus modelled discharge for the Leaf River watershed	52
Figure 7.2 Observed versus modelled discharge for the Leaf River watershed	52
Figure 7.3 Observed, modelled and updated discharges for the batch file method.....	54
Figure 7.4 Observed, modelled and updated discharges for the OpenMI composition....	54
Figure 7.5 Values of the state variables for the batch file method	55
Figure 7.6 Values of the state variables for the OpenMI composition	55
Figure 7.7 Observed, modelled and updated discharges for 10 ensembles	56
Figure 7.8 Observed, modelled and updated discharges for 99 ensembles	57
Figure 1 Data definitions in the OpenMI interface specification	65
Figure 2 Other interfaces of the OpenMI.....	66

1 Introduction

1.1 Background

Water is becoming alarmingly scarce. Competition for this scarce resource leads to the need for integrated water management. The main water issue we are facing today is concerned with the search for effective management methods that can fulfil the needs of both man and environment.

Integrated catchment management has arisen because managing environmental processes independently does not always lead to an equitable or sustainable distribution of water resources when the wider view is taken.

Integrated water resources management is the integrating concept for a number of water sub-sectors such as hydropower, water supply and sanitation, irrigation and drainage, and environment. An integrated water resources perspective ensures that social, economic, environmental and technical dimensions are taken into account in the management and development of water resources.

Integrated water management has created a need to understand and model catchment processes and particularly their interactions (Moore et al, 2004). The increasing pressure on water resources has further intensified this need. Integrated catchment management asks for integrated analysis that can be supported by integrated modelling systems.

The processes involved in catchment hydrology are complex and no single model or modelling system can adequately represent all of the processes in the catchment environment.

However, the problem for those charged with integrated management is the complexity of the process they are attempting to manage. They are therefore turning to decision support systems.

A Decision Support System combines aspects of information and modelling systems, and suits users who are not necessarily experts in modeling or data analysis, converts policy and management decisions automatically to suitable model runs and data analyses, allows the outcome of such decisions to be compared, and perhaps even gives advice on the optimal decision to be made.

The models used in decision support systems tend to address single issues. However, the catchment manager needs to understand all the possible impacts of pursuing any given policy. Implicit in this is a requirement both to understand and to be able to model not only the individual catchment processes but also their interactions.

Analytical models are used to represent and simulate the behavior of physical and social systems, such as a river basin. A set of distinct, individual models can be collected to represent all the individual activities at geographically distinct locations. In real life, these activities are inter-connected – they affect each other and are mutually dependent. What is required is a mechanism to allow the various individual models to be similarly interconnected.

In line with these requirements the HarmonIT project which is a research project funded by the European Commission, is developing and implementing a European Open Modelling Interface and environment (OpenMI) that will simplify the linking of hydrology related models. It is the ambition of the HarmonIT consortium to turn OpenMI into the European ‘standard’ for model linkage in the water domain (Gijssbers 2004).

OpenMI is not an integrated modeling system, but it is a standard data-exchange definition, for which tools and utilities are available, that allows integrated modeling systems to be built. OpenMI provides a mechanism to allow the various individual models to be similarly interconnected.

Currently, OpenMI is in its beta testing stage.

1.2 Research scope

1.2.1 Uncertainty in modelling and data assimilation

Models are simplified representations of the natural system. This definition of a model implies that models are not exact representatives of the system we want to model. Because of these simplifications errors are associated with models and therefore model outputs are uncertain. Four sources of uncertainty may be identified in modelling (Butt et al, 2004);

- random or systematic errors in model inputs or boundary condition data;
- random or systematic error in the recorded output data;
- uncertainty due to sub-optimal model parameter values and
- errors due to incomplete or biased model structure (Butt et al, 2004);

Of the methods used to reduce model uncertainty, data assimilation is the most efficient and most often used. It is a methodology which can optimize the extraction of reliable information from observations and combine it with, or assimilate it into, numerical models. It is a combination of the model output and measurement data to make a better prediction for the future. Recent methods assume that the analysis is an optimal blend of the observed values and a background state (first guess), given by a previous forecast. The weights that prescribe the blend, are determined by error covariance matrices

There are several data assimilation methods in use. Auto Regressive Moving Average (ARMA), Artificial Neural Network (ANN), Genetic Programming (GP), Kalman Filter (KF), Extended Kalman Filter (EKF), Ensemble Kalman Filter (ENKF), etc.

Kalman filter is one of the most well known data assimilation methods. It was first introduced by R. E. Kalman in 1960. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown (Welch and Bishop, 2004a).

However, the Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by a linear stochastic difference equation. But what happens if the process to be estimated and (or) the measurement relationship to the process is non-linear? To address this problem KF has been extended - Extended Kalman Filter in which a linearized and approximate equation is used for prediction of error statistics. However, EKF is suitable for small non-linearity. EKF is notoriously unstable if the nonlinearities are unstable (Miller, et al 1994).

Ensemble Kalman Filter is developed by Gier Evensen in 1994 which originates from Standard Kalman Filter. The EnKF is a sophisticated sequential data assimilation method. It applies an ensemble of model states to represent the error statistics of the model estimate, it applies ensemble integrations to predict the error statistics forward in time, and it uses an analysis scheme which operates directly on the ensemble of model states when observations are assimilated. The method uses non linear models to propagate the ensemble states. Some of the linearizations that make the EKF prone to failure are thereby avoided (Miller, et al 1994). The EnKF has proven to efficiently handle strongly nonlinear dynamics and large state spaces (Canizares, 1999).

1.2.2 Rainfall-runoff modelling and EnKF

Rainfall runoff (RR) models are important components of flood early warning systems. These models, like any other models, often contain several uncertainties due to insufficient model equations, simplifications in considering spatial and temporal scales to reduce computational requirements, and incorrect and incomplete data of the model.

Due to all such uncertainties in the RR-models and their inputs, their prediction usually contains large errors. In order to estimate and reduce the errors to improve the model output, data assimilation technique can be applied. Therefore the EnKF can be a very attractive data assimilation technique for reduction of uncertainty of RR models which are often highly non-linear.

1.2.3 OpenMI and Ensemble Kalman filtering

EnKF has been widely used in reducing uncertainty in hydrologic model forecasts. This is done in most of the models on an ad hoc basis (in a batch file) or implementing the EnKF in the model. This means the EnKF algorithm is incorporated in a dedicated way for each modelling package.

OpenMI offers an interface for data exchange between models at run time. However, models should be OpenMI compliant to be able to exchange data using that interface.

The EnKF algorithm, treated as a separate model, may be made a linkable component, and therefore can be linked with other models so that they exchange data at run time for the update process.

This research focuses on exploring the suitability of OpenMI for linking of EnKF with an OpenMI compliant model.

1.3 Objectives of the study

The overall objective of this study is to explore the suitability of OpenMI for application of Ensemble Kalman filtering for reducing uncertainty of model outputs.

To achieve this objective two sub objectives are identified. Firstly, the study aims at providing theoretical insight into the open modeling interface philosophy (OpenMI). What is OpenMI and why it is required. The requirements to use OpenMI are also discussed. Secondly, the study explores the suitability of OpenMI for application of ensemble Kalman filtering. It tries to identify what is to be done for application of

ensemble Kalman filtering for reduction of uncertainty in model outputs using the OpenMI interface.

1.4 Research Approach

To be able to link models using OpenMI, the model components have to be OpenMI compliant. Model components would be OpenMI compliant when they implement the OpenMI standard interface. Therefore the approach used in this study is first making a rainfall-runoff model (in this particular case HYMOD-RR) and the EnKF OpenMI compliant using utilities provided by the OpenMI consortium. After the model components become OpenMI compliant, the rainfall runoff model will be linked to the EnKF. This will be done by a generic configuration of an OpenMI composition of the rainfall-runoff model components and the EnKF component.

As the ensemble Kalman Filter needs as many computations as the number of ensembles at a time step, one EnKF component should be linked with as many rainfall runoff model components as the number of ensembles. Therefore the second step is to try to automate the creation of the links between one EnKF component and the rainfall-runoff model components.

Following establishment of these links, the data exchange process can begin. To verify the successful creation of the links and subsequent exchange of data for the ensemble Kalman filtering process, a case study will be carried out using the configured OpenMI composition and the results will be compared with results of the batch file method.

From the outcome of the research, conclusions about the suitability of OpenMI environment for ensemble Kalman filtering and its advantage and disadvantages over other methods will be drawn. Recommendation for further study will also be made.

1.5 Organization of the thesis

The dissertation is presented in nine chapters and one appendix including the introduction chapter.

The first chapter is an introduction which gives background information on the philosophy of OpenMI, the Open Modelling Interface and environment, research scope and objective of the research.

Chapter two contains a theoretical background on the concepts of open modelling interface, its requirements, the overall architecture of OpenMI and how it address the general issues of model linking and its consequences for a model. Chapter three briefly describes the rainfall-runoff model and theoretical background of the ensemble Kalman filter is presented in chapter four. Chapter five explains the methodology and approaches used to undertake the study.

The results of the study and discussion of the results are given in chapter six and chapter seven contains a case study to verify the result obtained in chapter six. The summary, conclusion and recommendations are presented in chapter eight, followed by references.

2 OpenMI

In this chapter theoretical background of the OpenMI is given. Section 2.1 provides the introduction part and the requirements of OpenMI are listed in section 2.2. The overall architecture and layering and how OpenMI address the general issue of model linking are provided in sections 2.3 and 2.4 respectively. Section 2.5 gives the consequences OpenMI architecture for a model and finally section 2.6 reasons out why OpenMI is called an interface based open standard to link models.

2.1 Introduction

OpenMI stands for Open Modelling Interface and Environment. It is being developed and implemented by the HarmonIT project, which is research project funded by the European Commission. It aims to deliver a standardized way to link water related computational models that run parallel. It thus enables process interaction being represented more accurately, compared to sequential linkages

A model application is the entire model software system that is installed in a computer. Normally a model application consist of many parts, the most common being the user interface, input files, the engine and output files. The engine is where the calculations take place. The user supplies information through the user interface upon which the user interface generates input files for the engine. The user can run the model simulation e.g. by pressing a button in the user interface, which will deploy the engine (see Figure 2.1). The engine will read the input files and perform calculations and finally the results are written to output files. When an engine has read its input files it becomes a model. In other words a model is an engine populated with data. A model can simulate the behaviour of a specific physical entity e.g. the River Rhine. If an engine can be instantiated separately and has a well-defined interface it becomes an engine component. An engine component populated with data is a model component. There are many variations of the model application pattern described above, but most important from the OpenMI perspective is the distinction between model application, engine, model, engine component, and model component.

Basically, a model can be regarded as an entity that can provide data and/or accept data. Most models receive data by reading input files and provide data by writing output files. However, the approach for OpenMI is to access the model directly at run time and not to use files for data exchange. In order to make this possible, the engine needs to be turned into an engine component and the engine component needs to implement an interface

through which the data inside the component is accessible. OpenMI defines a standard interface that engine components must implement to become OpenMI compliant engine components. When an engine component implements this interface it becomes a linkable component. A similar pattern can be applied for databases or other kinds of data sources. By turning them into components and implementing the OpenMI interface they become linkable components that provide direct access to its data at run time.

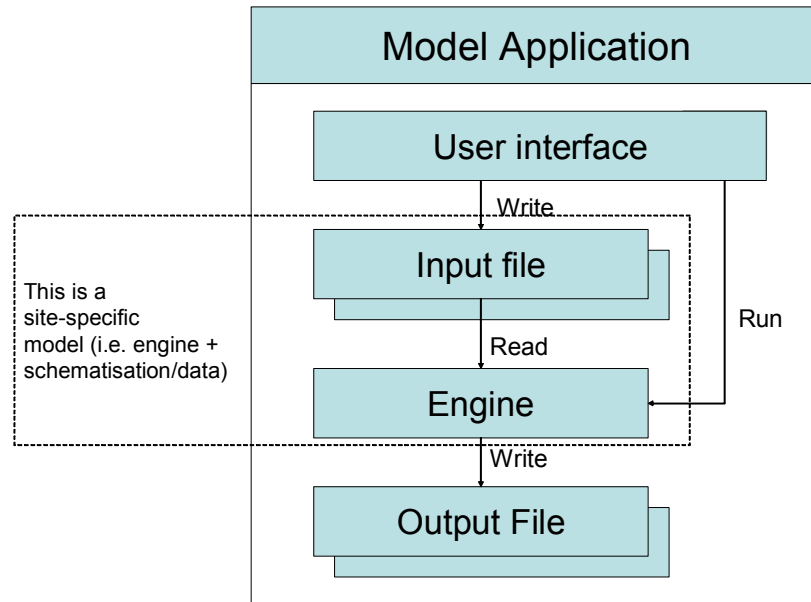


Figure 2.1 Model Application Pattern

[Source: adopted from (Gijssbers, 2004)]

OpenMI is based on the ‘request & reply’ mechanism (Gijssbers, 2004). OpenMI is a pull based pipe and filter architecture which consists of communicating components (source and target components) which exchange data in a predefined way and in a predefined format. OpenMI defines both the component interfaces as well as how the data is being exchanged. The components in OpenMI are called linkable components to indicate that it involves components that can be linked together.

From the data exchange perspective, OpenMI is a purely single-threaded architecture where an instance of a linkable component handles only one data request at a time before acting upon another request. Data exchange in the OpenMI-architecture is triggered by a component at the end of the component chain. Once triggered, components exchange data autonomously without any type of supervising authority. If necessary, components start their own computing process to produce the requested data. Only when output needs to converge to a certain criteria, a controlling component might need to be incorporated.

2.2 Requirements

OpenMI is developed to full fill the following requirements addressing software technical/implementation issues as well as water domain related issues.

Software technical requirements

- OpenMI should enable linkage of a (legacy) model with another model, database or other data source.
- The HarmonIT project is targeted at linking legacy code. Therefore reengineering efforts should be minimal, i.e. it shall be possible to migrate existing model systems to comply with OpenMI without necessarily reprogramming the entire model system. In addition, it should be easy to add in new models.
- OpenMI should be 'open', which has been interpreted as providing an complete, clearly defined, and thus 'open', interface that enables a third party to develop OpenMI compliant components that should co-operate with other OpenMI-components in an OpenMI application without any problems.
- The OpenMI specification should be based on mature and proven paradigms rather than currently available technologies. The architecture should be component based and multilayered.
- Linkage mechanisms should focus at high abstraction levels, using protocols which are independent from implementation choices
- Any OpenMI environment should be as thin as possible.
- Implementation(s) of the OpenMI-architecture should be able to run on Windows, Linux and UNIX operating systems as well as on networks with mixed platforms.
- The OpenMI architecture should have sufficient flexibility to enable adaptation to future hardware, operating systems etc.
- The OpenMI specification should enable distributed computing. It should not hinder parallel processing, but preferably encourage/support these needs. Therefore multi-threading and concurrency issues need to be addressed.
- OpenMI should not hinder batch processing/calculations.
- OpenMI should allow components to be developed using at least the following programming languages: C/C++, C#, Fortran, Delphi/Pascal, Java and Visual Basic.
- OpenMI must include a protocol for exception and error handling.

Domain related requirements

- The HarmonIT project will provide standards and tools in order to facilitate linkages between model engines and models (models populated with data) during

- execution of these models. The focus therefore is on transfer of data, typically boundary conditions.
- OpenMI will support linkages of the sequential linkages as well as dynamic linkages enabling implicit, explicit and weak implicit coupling. Linkages can be uni-directional and bidirectional. Feedback loops should be facilitated.
 - OpenMI should not dictate the same time step interval when two models are linked.
 - OpenMI should target at exchange of boundary conditions between (model) components. Preferably, the architecture can accommodate exchange of other data sets as well.
 - OpenMI should facilitate exchange of uncertainty data.
 - OpenMI should offer ways to access model parameters for (automated) calibration purposes.
 - To enable these linkages, OpenMI will create explicit ways to define spatial and temporal reference, data types, parameters/quantity, unit, locations in space, time;
 - (Topological) information describing networks and grids should be separable from object (node/branch/cell) properties, so dedicated tools can handle them separately.
 - OpenMI should accommodate efficient exchange of bulky data sets.
 - OpenMI should not hinder development of automated linkage tools, which can initiate, link (process, domain, space and time), execute and terminate network of components.
 - OpenMI should allow that data related to the quality of the components (if such data is available) could be passed so that these data could be available for checking (e.g. mass balances).
 - OpenMI will not define a common data model for persistent data objects in the water domain (e.g. cross sections or structures).

2.3 Overall architecture and layering

The interfaces of the OpenMI architecture, i.e. the Open Modelling Interfaces, are specified in the namespace `org.OpenMI.Standard`. Software components that implement and use these interfaces properly are called OpenMI compliant.

A default implementation of these interfaces is provided in the `org.OpenMI.Backbone` package (see Figure 2.2). This Backbone is part of the OpenMI environment as being developed by the HarmonIT project. This environment provides facilities to support wrapping of legacy code (the `org.OpenMI.Utilities` namespace), a configuration and deployment environment (the `org.OpenMI.Configuration` namespace) and front-end tools to enable interaction with its users (the `org.OpenMI.Tools` namespace).

In case wrapping legacy models is chosen for the implementation of the system, the architecture imposes only few rules on how the wrapper should be implemented.

OpenMI architecture

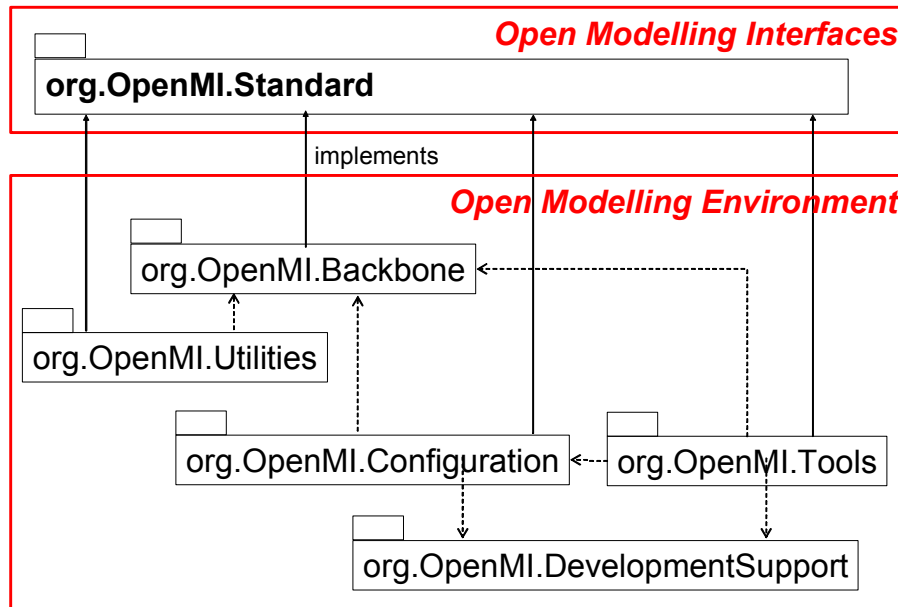


Figure 2.2 OpenMI architecture

[Source: adopted from (Gijsbers, 2004)]

2.4 How OpenMI addresses general issues of model linkage

The OpenMI standard addresses the following items required for model linkage:

2.4.1 Data definition

To define the data that is exchanged, a distinction is made between where, when, what and the (numeric) values itself. The data is described by identifying the values (data) itself, the geometry on which the data is defined, the time(s) for which the data is valid and the actual quantity it represents.

Where: The space where the values apply is indicated in a finite element way by an ordered list of elements (the element set), where conceptually each element consists of a number of connected vertices. An element holds an ordered set of vertices where the element shape type determines the minimum number of vertices of an element as well as the semantics of ordering. In this way topology is described. Within one element set all

elements must have the same type. Elements may, but do not need to be geo-referenced. Elements described in more detail by vertices are geo-referenced, as the vertices contain the coordinates to locate the element in a geo-reference system

When: Time in OpenMI is defined either by a time stamp or a time span. A time stamp is a single point in time, whereas a time span is a period from begin to end time. Each of these times is represented by the Modified Julian Date.

What: The physical semantics of the values is described by a quantity, combined with a unit in which its value is expressed. Water level (m) and amount of flow (cubic meter/hour) are examples of quantities. The physical nature of a quantity also related to the shape type of an element. E.g. a water level can be given in a point or along a poly-line or over a plane, but not in a volume. OpenMI does not use a standardized data dictionary. However, to ensure that linkages between quantities are correct, two aspects received specific attention. First of all, for every quantity, units are defined as well as a conversion formula of the form $(A*x + b)$ to enable unit conversion from the quantity's unit to standard SI units. This allows straightforward linking of quantities of different units without a performance penalty of unnecessary transformations. Secondly, the dimension needs to be provided for each quantity. By convention, this dimension is expressed as a combination of base quantities.

Values: The values themselves are represented as a one-dimensional array of scalars or vectors contained in a so-called value set. Information about the values (space, time, quantity) and their ordering is assumed to be known by the user. A single scalar or vector value exists for every element in the element set defined.

How: In the simplest case, the values returned are exactly the data as it is computed by a computational core of the linkable component. Nevertheless, under some conditions, specification of additional data operations is desired to obtain the data in the way it is needed. Most common situation will be the need for temporal aggregation over a time span. This functionality can be used if one model, running a small time step, needs to feed another model, which runs at a large time interval. The functionality may also be used to suppress a temporal interpolation method in case the time steps of the provider and the acceptor do not coincide. In addition to temporal data operations, spatial data operations are foreseen as well as a miscellaneous group of data operations.

2.4.2 Meta data defining potentially exchangeable data

An essential part of standardizing data exchange is the meta-data telling which components, model schematizations and data sets are involved and what can be exchanged in terms of quantities (what does it represent), element sets (where does it apply), time (when does it apply) and data operations (how should it be provided). The availability of this meta-data varies from fixed in the code or semi fixed in a model script to highly variable, e.g. determined by the (run-time) settings of the scenario to run.

For most models, the model code determines which quantities can potentially be provided as output or are (potentially) needed as input. Often the code determines the element set where data can be exchanged (e.g. on the boundaries, or on the full-domain). In a site-specific model, the code is populated with site-specific schematization data (e.g. a network or grid with attribute data). With this data, the exact elements are known, including their position in the topology.

A so-called 'exchange model', is defined to describe the data items that can be provided or accepted by a component. Each item that can be exchanged contains information on the role of the data (input or output), the quantity it represents and the element set where it applies. The exchange model also contains information on the data-operations that can be provided by the delivering component. The combination of a quantity on an element set has been chosen as one typically links the boundaries of one model to the boundaries of another model (e.g. the 'bottom' of a river with the 'top' of a ground water model).

An exchange model is specific for a combination of a model code with a model schematization. However, one combination of model code and model schematization may have several exchange models, each one for a different purpose. E.g. the Rhine model using the SOBEK channel flow code may have an exchange model tailored to linkage with a ground water model, and another exchange model for linkage to an estuary model.

The exchange model is mainly used during design/configuration time to set up the links between different models.

2.4.3 Generic model access

Generic model access is essential for the component based paradigm as adopted in OpenMI. At the core of OpenMI are two basic interfaces to access a model component, namely the 'exchange model' interface that describes the data that potentially can be exchanged (see Section 2.4.2) and the 'linkable component' interface to link the components together at run-time and exchange data. Since all access to a component is

through these interfaces, generic OpenMI implementations can be made independent of underlying type of engine or component being used. This approach allows the addition of new components to an existing OpenMI run-time environment without modifications to the environment.

GetValues() is the most important run-time method of a linkable component. It returns the data requested, if needed by invoking a computation. All other methods are supportive, e.g. during the initialization phase, to handle links, and to clean up. For some situations, e.g. to enable iteration, it is useful if a linkable component can manage its state on request. A state management interface has been introduced for this purpose. Again it is up to the code developer which state-related data is 'saved' and how it is done (e.g. in memory, in a file). To enable event publishing, a linkable component is inherited from a Publisher-interface.

Basically all components exchanging (model) data are linkable components. Examples of linkable components are:

- Simulation engines for rivers, groundwater, general 3-D flow, and rainfall-runoff processes.
- Measuring device that need to be accessed online.
- Monitoring databases containing historic data.
- Data driven models such as Artificial Neural Networks

Time related issues

Characteristic to water resources modelling and management is the importance of time. Often, data is exchanged for the same quantity and spatial elements, but for different time slots. Therefore, a time indication has been incorporated as a separate argument in the GetValues()-call.

Each GetValues() request initiates a processing activity (e.g. computation) when needed to respond properly. As many water related models progress over time, the time argument is the controlling variable for any processing activity. Linkable components that do not progress over time can neglect the time-argument. They just do their work and return the data.

In case the requested timestamp does not match the time stepping in the computation and the computation is already ahead in its computation, an interpolated value is to be delivered. Note that the model code developer decides how this interpolated value is computed and if any buffering is applied to increase performance. When the computation has not yet reached the request time stamp, two alternatives are available:

- Initiate a computation to compute the values at the requested time.
- Extrapolate the solution

Under normal conditions the first alternative is chosen. For bi-directional links, one of the components will need to extrapolate its solution in order to prevent deadlock situations. The above mentioned obligation has been formulated from the perspective of a simulation engine. However, (monitoring) databases and other linkable components, which do not progress in time, should also be able to return a value, whether it is the actual, interpolated or extrapolated one. It is up to the software developer to introduce an intelligent (or customizable) wrapper for this purpose.

For those situations where the exact time information is required, an additional interface is provided to obtain the discrete time stamps available. If no value can be provided, and no representation has been defined for missing values, an exception needs to be thrown.

From a more detailed technical point of view, it could be stated that at present the OpenMI is limited to a time step based exchange (including data transformations and/or operations where necessary) of data which form the interacting boundary conditions of the individual components

Wrapping legacy code

For legacy code, wrapping will most often be the technological choice to migrate to OpenMI. An existing model engine (i.e. a computational core often developed in Fortran 90) is encapsulated in a so-called wrapper that meets the interface specification of an OpenMI linkable component. The OpenMI interface allows it to be treated in a generic way by an OpenMI run-time environment. Actually, the 'wrapper' turns the computational core into a linkable component for OpenMI

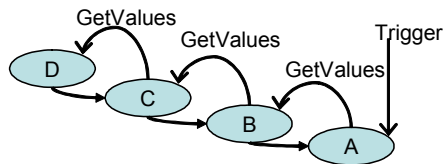
2.4.4 Data transfer

Linkable components can exchange data by a pull mechanism, meaning that a (target) component that requires input asks a source component for a (set of) value(s) for a given quantity on a set of elements (i.e. locations) for a given time. If required, the source component calculates these values and returns them. This pull mechanism has been encapsulated in one single method, the `GetValues()` method. Dependent on the status of the source component, this call may require associated computation and even more requests for data. An important feature is the obligation that components always deal with requests in order of receipt.

Pull driven communication

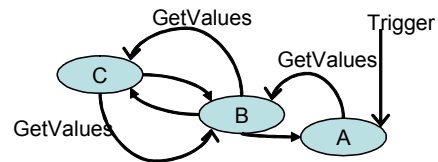
Since the ‘target’ pulls the data when needed, this mechanism is called pull-driven. Linkable components can be connected in a chain, where the last component in the chain triggers the entire stack of data exchange. The following figure illustrates three chain layouts:

1. Linear chain (uni-directional)

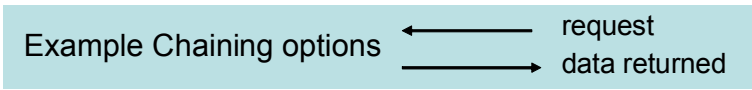


A requests B, B requests C, C requests D
D does its work and returns data to C, C does its work and returns data to B, etc.

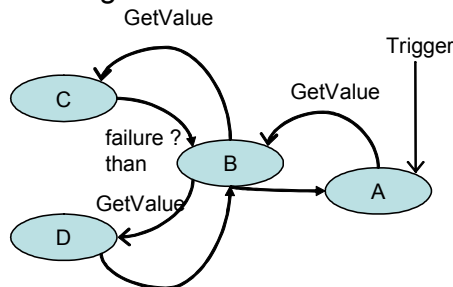
2. Linear chain (bi-directional)



A requests B, B requests C, C requests B
B returns a best guess to C, C does its work and returns data to B, B does its work and returns data to A



3. Logical decision chain



A requests B, B requests C,
C does its work and returns data to B
if C fails B requests D
B returns data to A

Figure 2.3 Different chain layouts with the pull-mechanisms
[Source: adopted from (Gijsbers, 2004)]

2.4.5 Definition of actually exchanged data

The link: purpose

A link defines actual data exchange of a semantically similar quantity between two linkable components (the source component or provider and the target component or acceptor).

The link: content

The link specifies the following information:

- The source component and the target component.

- The source quantity and target quantity, i.e. two aliases for a semantically similar quantity. The source quantity may differ from the target quantity by name as well as by unit, only in that a unit conversion is required to transform the values. Of course the semantics and the dimension of both quantities should be similar.
- The list of data operations that must be applied by the source component before providing the data.
- The element set on which the quantity is defined internally in the source linkable component.
- The element set on which the quantity is requested by the target component (and thus delivered by the source component).

Links are unidirectional (i.e. from the source component to the target component). A link refers to a single semantic quantity only, which, however, may have two different names on each side of the link. This means that with bi-directional communication or with multiple quantities, multiple links must be configured.

In water related data exchange the spatial geometry underlying the data exchange typically is persistent over time. Therefore, currently, the standard assumes that the element sets incorporated in the link are constant over time. Nevertheless, it is allowed to add and remove links at run-time in case the geometry should change.

The link as an argument

The link has a string ID which is passed as an argument in the `GetValues()`-call of the linkable component. This allows independent communication over different links from one source component to multiple target components. This means that data exchange takes place over entire (target) element sets. In case the target component is only interested in the information on a part of the element set, the target component should perform the selection itself or the link should be configured to contain only the appropriate part of the target element set. Both ways avoid the need to define generic geometric data selection algorithms, thus keeping the standard simple and allowing high performance implementations. Of course, such selection algorithms can be provided as utilities but they are not part of the OpenMI standard.

2.4.6 Event mechanism

In addition to the above mentioned functionalities to link components, a lightweight event mechanism has been introduced (interfaces defined in Figure 2). Via this mechanism a wide range of messages can be passed to enable call stack tracing, progress

monitoring and to flag status changes which might trigger other components (e.g. visualization tools) to request for data via a `GetValues()` call.

By convention a linkable component has to throw an exception if an internally irrecoverable error occurs. This exception should be based on the `Exceptionclass` as provided by the development environment.

2.5 Consequences of the OpenMI architecture for a model

The OpenMI enables model engines to compute and exchange data at their own heartbeat, without any external control mechanism. Deadlocks are prevented by the obligation of a component always to return a value whatever the situation. When each model is asked for data it decides how to provide it – it may already have the data in a buffer because it has previously run the appropriate simulation, or by running its own simulation or calculation, or by making a best estimate via interpolation or extrapolation. Or it may not be able to provide the requested data, so will raise an exception. The exchange of data at runtime is automated and driven by the predefined links, with no human intervention.

To become an OpenMI linkable component, a model has to:

- be able to expose information (what, where) to the outside world on the modelled variables which it is can provide, or which it is able to accept;
- submit to runtime control by an outside entity;
- be structured in a way that initialization is separate from computation, where boundary conditions are collected in the computation phase and not during initialization;
- be able to provide the values of the modelled variables for the requested points in time and space;
- be able to respond to a request, even when the component itself is time independent; if such response requires data from another component, the component should be able to pass on the time as well in its own request.
- In case some values in the value set are missing, this should be flagged in the value set.
- In the exceptional case that an entire value set is unavailable, an exception needs to be thrown. Be aware that such exception will stop the entire computation process and thus should be prevented whenever possible.

2.6 OpenMI: an interface based open standard to link models

OpenMI provides an intelligent mechanism to link models at run-time in terms of describing, defining and transferring data while models run at the same time. It thus enables process interaction being represented more accurately, compared to sequential linkages. OpenMI is NOT a common data-model specification, and it certainly is NOT an integrated modelling system.

OpenMI is ‘interfaced based’ as

- it ‘standardized’ part is defined as an software interface specification
- this interface acts as a ‘contract’ between software components
- the interface is not limited to specific technology platforms or implementations
- by implementing this interface a component has become a (technology specific) OpenMI compliant component

OpenMI is ‘open’ as:

- its specification is publicly available via the Internet (www.OpenMI.org)
- it enables linkages between different kinds of models, different disciplines and different domains
- it offers a complete meta data structure to fully describe in a rather primitive, but extensive way, what numerical data can be exchanged in terms of semantics, units, dimensions, spatial and temporal representation and data operations.
- it provides a means to define exactly what is linked, how and when
- it’s default implementation and software utilities as developed by the HarmonIT project are available under an open source software license

OpenMI is a ‘standard’ as:

- it standardizes the way data transfer is specified and executed;
- it hence allows any model to “talk” to any other model (e.g. from a different developer) without the need for co-operation between model developers or close communication between integrators & model developers;
- its generic nature does not limit itself to a specific domain in the water discipline or even in the environmental discipline;

3 Description of the HYMOD-RR Model

This chapter provides the description of HYMOD-RR model.

3.1 The HYMOD Structure

HYMOD is a five parameter conceptual rainfall-runoff model introduced by Boyele and recently used by Wangener (Vrugt et al, 2003).

The model consists of a simple two-parameter rainfall excess model connected with two series of linear reservoirs (three, identical, for the quick and a single reservoir for the slow response) in parallel as a routing component (see Figure 3.1). The model assumes that the soil moisture storage capacity, c , varies across the catchment, and therefore, that the proportion of the catchment with saturated soils varies overtime. The spatial variability of soil moisture capacity is described by the following distribution function.

$$F(c) = 1 - \left(1 - \frac{c(t)}{CMAX}\right)^{BEXP} \quad 0 \leq c(t) < CMAX \quad (0.1)$$

The structure requires the optimization of five parameters: the maximum storage capacity in the catchment, $CMAX[L]$, the degree of spatial variability of the soil moisture capacity within the catchment, $BEXP[-]$, the factor distributing the flow between two series of reservoirs, $ALPHA[-]$, and the resident times of the linear reservoirs, $Rq[T]$ and $Rs[T]$. The actual evapotranspiration is equal to the potential value if sufficient soil moisture is available; otherwise it is equal to the available soil moisture content.

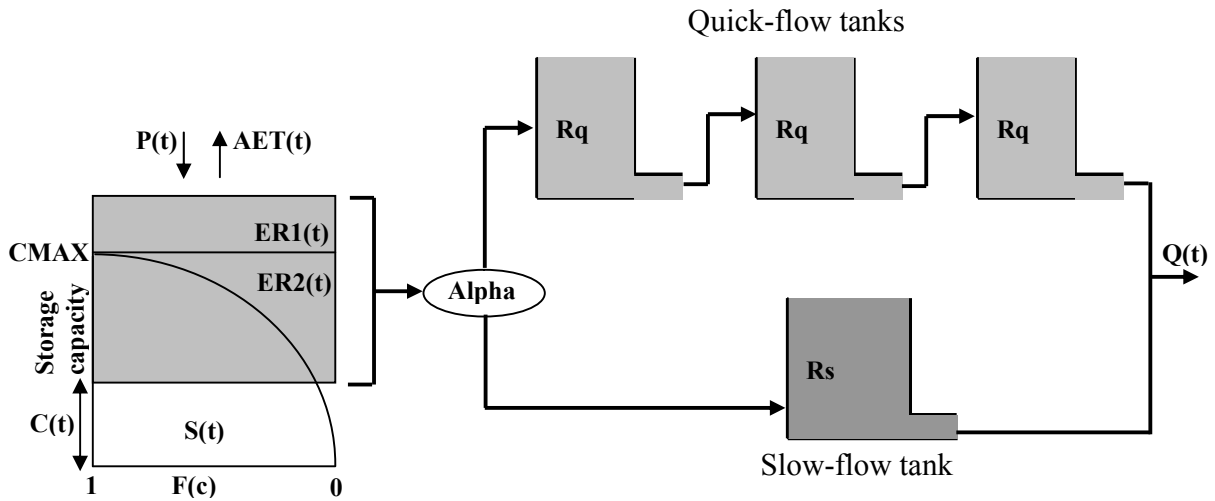


Figure 3.1 The HYMOD Model structure
 [Source: adopted from Wagener et al, 2001]

Effective rainfall ($ER1(t)$ and $ER2(t)$) is produced depending on the current catchment moisture state described by the storage capacity distribution $F(c)$. The parameter C_{MAX} describes the maximum storage capacity in the catchment. The effective rainfall is distributed with respect to parameter $ALPHA$ and either routed through three linear reservoirs with residence time R_q in series, or a single reservoir with residence time R_s . Variable $Q(t)$ is the resulting stream flow at time step t . The remaining variables are the storage $S(t)$, the precipitation input $P(t)$, and the actual evapotranspiration $AET(t)$ (Wagener et al, 2001).

The input of HYMOD model at time n consists of the mean areal precipitation, $P [mm.d^{-1}]$, and evapotranspiration, $E [mm.d^{-1}]$ measured over a certain catchment area.

3.2 The state variables

The HYMOD model has five states (Vrugt et al, 2003). The five model states at time t are defined below;

1. X_{loss} the amount of excess rainfall,
2. X_{slow} the amount of water residing in the slow flow tank at time t ,
3. X_{quick1} the amount of water residing in the first quick flow tank at time t ,

4. X_{quick2} the amount of water residing in the second quick flow tank at time t ,
5. X_{quick3} the amount of water residing in the third quick flow tank at time t ,

The five states are described with the help of the parameters by the following equations.

First the amount of water in the catchment at time t is calculated with

$$c(t) = C_{\max} \left(1 - \left(1 - (b_{\exp} + 1) \frac{xloss(t)}{C_{\max}} \right)^{\frac{1}{b_{\exp} + 1}} \right) \quad (0.2)$$

The amount of excess rainfall, in case the soil in the catchment is saturated, is given by;

$$ER_1(t+1) = \max(P(t+1) - (C_{\max} - c(t)), 0) \quad (0.3)$$

$$\bar{P}(t+1) = P(t+1) - ER_1(t+1) \quad (0.4)$$

Where $P(t+1)$ is the amount of rainfall at time $t+1$ and $ER_1(t+1)$ is the water that floods the catchment (surface runoff). The evaporation at time $t+1$, $\bar{E}(t+1)$ is calculated with the following equation;

$$J = \min\left(\frac{c(t) + \bar{P}(t+1)}{C_{\max}}, 1\right) \quad (0.5)$$

$$\bar{x}loss(t+1) = \frac{C_{\max}}{b_{\exp} + 1} \left(1 - (1 - J)^{b_{\exp} + 1} \right) \quad (0.6)$$

$$ER_2(t+1) = \max(\bar{P}(t+1) - \bar{x}loss(t+1) - xloss(t), 0) \quad (0.7)$$

$$\bar{E}(t+1) = \min(\bar{x}loss(t+1), E(t+1)) \quad (0.8)$$

Where $E(t+1)$ is the measured amount of evapotranspiration at time $t+1$. From these equations it can be concluded that the amount of evapotranspiration depends on the rainfall and the measured evapotranspiration is not always used by the model. After the value of the evapotranspiration is known, the state $xloss(t+1)$ will be updated as;

$$xloss(t+1) = \bar{x}loss(t+1) - \bar{E}(T+1) \quad (0.9)$$

The following equation represents the states of the reservoirs or tanks, $xslow(t+1)$, $xquick1(t+1)$, $xquick2(t+1)$, and $xquick3(t+1)$, but first the amount of water for the quick and slow flow reservoirs have to be defined.

The amount of water to the quick flow (U_q) is;

$$U_q(t+1) = \alpha U_2(t+1) + ER_1(t+1) \quad (0.10)$$

And the amount of water to the slow flow (U_s) is;

$$U_s(t+1) = (1-\alpha)ER_2(t+1) \quad (0.11)$$

The equations for the slow flow state, $xslow(t+1)$ and the run off from the slow flow reservoir, $Q_s(t+1)$ can now be written as;

$$xslow(t+1) = xslow(t)(1-R_s) + U_s(t+1)(1-R_s) \quad (0.12)$$

$$Q_s(t+1) = xslow(t+1) \frac{R_s}{1-R_s} \quad (0.13)$$

And in a similar way the quick flow states and their outflow, $Q_q(t+1)$ can be defined,

$$xquick1(t+1) = xquick1(t)(1-R_q) + U_q(t+1)(1-R_q) \quad (0.14)$$

$$Q_{q1}(t+1) = xquick1(t+1) \frac{R_q}{1-R_q} \quad (0.15)$$

$$xquick2(t+1) = xquick2(t)(1-R_q) + Q_{q1}(t+1)(1-R_q) \quad (0.16)$$

$$Q_{q2}(t+1) = xquick2(t+1) \frac{R_q}{1-R_q} \quad (0.17)$$

$$xquick3(t+1) = xquick3(t)(1-R_q) + Q_{q2}(t+1)(1-R_q) \quad (0.18)$$

$$Q_{q3}(t+1) = xquick3(t+1) \frac{R_q}{1-R_q} \quad (0.19)$$

The total discharge from the model is calculated by summing the resulting run off from both the small and last quick tank.

$$Q(t+1) = (Q_s(t+1) + Q_{q3}(t+1)) * 22.5 \quad (0.20)$$

The above described equations are presented in the Matlab code of the HYMOD program (Koster, 2004). (Vrugt et al, 2003) provided some information about the dimensions of the measurements and the parameters.

4 Ensemble Kalman Filter

This Chapter details the procedure of Kalman filter data assimilation process. Section 4.1 provides the introduction part followed by the explanation of the discrete Kalman filter, the extended Kalman filter and the ensemble Kalman filter in sections 4.2 to 4.4 respectively.

4.1 Introduction

The Kalman filter is the statistically optimal method for the sequential assimilation of data into linear numerical models and provides an estimate of the state of the system at the current time step based on all measurements of the system available up to and including the current time (Canizares, 1999).

When the filter is applied to non-linear models successive linearization of the non-linear dynamics have to be carried out. It is this approach that is known as Extended Kalman Filter (EKF).

The Ensemble Kalman Filter (EnKF) is based on the Monte Carlo simulation. It is a sequential data assimilation method where the error statistics are predicted using Monte Carlo or ensemble integration.

4.2 The Discrete Kalman Filter

The Process to be estimated

The Kalman filter addresses the general problem of trying to estimate the state $x \in \mathfrak{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_{k+1} = A_k x_k + B_k u_k + w_k \quad (4.1)$$

with a measurement that is

$$Z_{k+1} = H_{k+1} x_{k+1} + v_{k+1} \quad (4.2)$$

The random variables w_k and v_{k+1} represent the process and measurement noise respectively. They are assumed to be independent (of each other), white, and with normal probability distributions.

$$p(w) \sim N(0, Q) \quad (4.3)$$

$$p(v) \sim N(0, R) \quad (4.4)$$

In practice, the process noise covariance and measurement noise covariance matrices might change with each time step or measurement, however here we assume they are constant.

The $n \times n$ matrix A_k in the difference equation(4.1) relates the state at the previous time step to the state at the current step, in the absence of either a driving function or process noise. The $n \times 1$ matrix B_k relates the optional control input $u \in \mathcal{R}^n$ to the state x_{k+1} . The $m \times n$ matrix H_{k+1} in the measurement equation (4.2) relates the state to the measurement z_{k+1}

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback- i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate.

The time update equations can also be thought of as predictor equations, while the measurement update equations can be thought of as corrector equations. Indeed the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems as shown below in Figure 4.1 (Welch and Bishop, 2004a).

The Kalman filter provides a linear unbiased estimate of the state vector that combines model forecast and measurements while the inherent uncertainties are taken in to account. In order to obtain an optimal solution in terms of a minimum error covariance, the following assumptions have to be made (Ghil and Malanotte-Rizzoli, 1991);

1. The initial state vector is assumed to be Gaussian with known mean $E\{x_0\}$ and known error covariance matrix $P_0 = Cov\{x_0\}$
2. The model noise w_k is a stationary white noise process with zero mean and known covariance matrix $Q_k = Cov\{w_k\}$
3. The measurement noise v_k is a stationary white noise process with zero mean and known covariance matrix $R_k = Cov\{v_k\}$

4. The initial state x_0 , the model noise w_k and the measurement noise v_k are mutually independent

The optimal state estimate and the associated error covariance matrix can be calculated recursively. First during the forecast step, the optimal state estimate is propagated from time k to time $k+1$. Moreover the model errors are propagated during the same period using the model dynamics forced by model noise. This step is represented by the following equation:

$$\bar{x}_{k+1} = A_k \hat{x}_k + B_k u_k \quad (4.5)$$

$$\bar{P}_{k+1} = A_k \hat{P}_k A_k^T + Q_k \quad (4.6)$$

where the notions \bar{x} and \hat{x} , indicate the time update of the state estimate and the measurement update respectively. This notation also hold for the estimate of the covariance matrix, p .

If measurements are available at time $k+1$, the state vector and the error covariance matrix are updated during the analysis step using the following equations:

$$\hat{x}_{k+1} = \bar{x}_{k+1} + K_{k+1} (z_{k+1} - H \bar{x}_{k+1}) \quad (4.7)$$

$$\hat{P}_{k+1} = (I - K_{k+1} H) \bar{P}_{k+1} \quad (4.8)$$

where $K_k \in \mathcal{R}^{n \times p}$ is the Kalman gain or weighing matrix, which describes the weights given to the innovation vector (difference between observed and forecasted variables) in the calculation of the analysed state variables. The Kalman gain is conventionally obtained from the equation

$$K_{k+1} = \bar{P}_{k+1} H^T [H_{k+1} \bar{P}_{k+1} H_{k+1}^T + R_{k+1}]^{-1} \quad (4.9)$$

where $R_k \in \mathcal{R}^{p \times p}$ is the error covariance matrix of the measurement noise.

The process continues recursively with the time update for the next time step. Equations (4.5) - (4.9) constitute the Kalman gain filter algorithm that provides the optimal estimate of the state in a least square sense when all the aforementioned assumptions are fulfilled. In a real application the assumptions are normally violated by (Canizares 1999):

- Non-linear system dynamics and/or measurement equation
- Unknown or partly known initial conditions, $E\{x_0\}$ and P_0
- Unknown or partly known noise statistics, Q_k and R_k

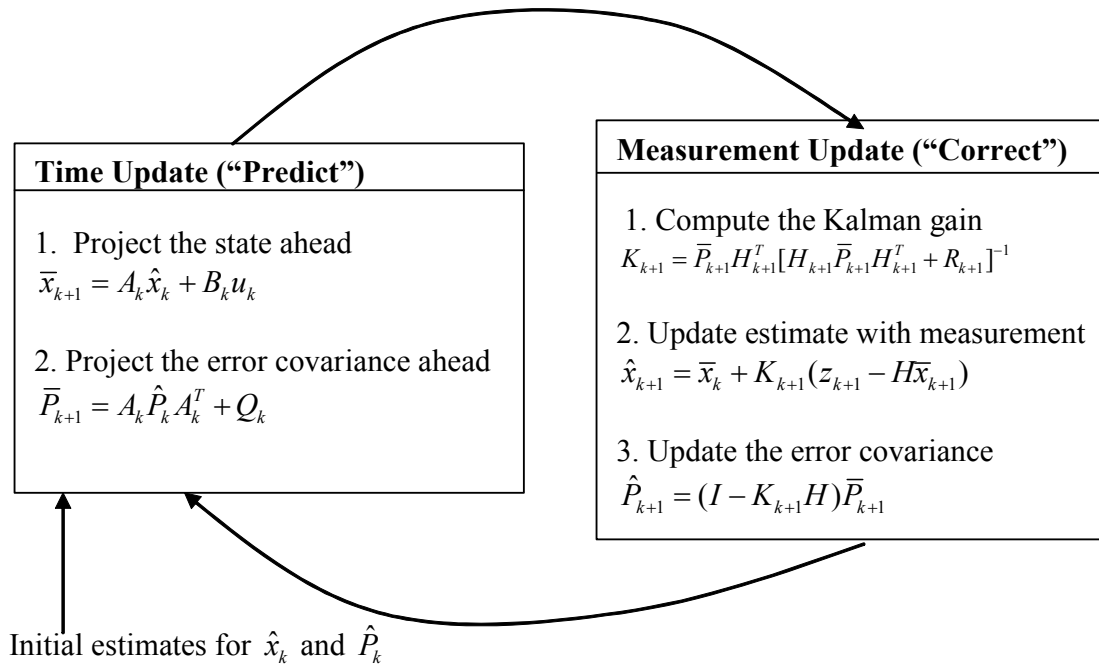


Figure 4.1 A complete picture of the operation of the Kalman filter
 [Source: adopted from (Welch and Bishop, 2004a)]

4.3 The Extended Kalman Filter

As described above, the Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by a linear stochastic difference equation. But what happens if the process to be estimated and (or) the measurement relationship to the process is non-linear? Some of the most interesting and successful applications of Kalman filtering have been such situations. A Kalman filter that linearizes about the current mean and covariance is referred to as an extended Kalman filter or EKF.

In something akin to a Taylor series, the estimation around the current estimate can be linearized using the partial derivatives of the process and measurement functions to compute estimates even in the face of non-linear relationships.

The Process to be estimated

$$x_{k+1} = f(x_k, u_k, w_k) \tag{4.10}$$

with a measurement that is

$$Z_{k+1} = H_{k+1}(x_{k+1}, v_{k+1}) \quad (4.11)$$

where the random variables w_k and v_{k+1} again represent the process and measurement noise as in discrete Kalman filter. In this case the non-linear function in the difference equation (4.10) relates the state at the previous time step to the state at the current time step. It includes as parameters any driving function u_k and the zero-mean process noise w_k . The non-linear function H_{k+1} in the measurement equation (4.11) relates the state to the measurement.

In practice of course one does not know the individual values of the noise and at each time step. However, one can approximate the state and measurement vector without them as

$$\bar{x}_{k+1} = f(\hat{x}_k, u_k, 0) \quad (4.12)$$

and

$$\bar{Z}_{k+1} = H_{k+1}(\bar{x}_{k+1}, 0) \quad (4.13)$$

where \hat{x}_k is some a posteriori estimate of the state (from a previous time step k).

It is important to note that a fundamental flaw of the EKF is that the distributions (or densities in the continuous case) of the various random variables are no longer normal after undergoing their respective nonlinear transformations. The EKF is simply an ad hoc state estimator that only approximates the optimality of Bayes' rule by linearization. Some interesting work has been done by Julier et al. in developing a variation to the EKF, using methods that preserve the normal distributions throughout the non-linear transformations (Julier et al, 1996).

4.4 The Ensemble Kalman Filter

The Ensemble Kalman Filter (EnKF) introduced by Evensen is based on Monte Carlo simulations. The EnKF is sequential data assimilation where the error statistics are predicted using Monte Carlo or ensemble integration (Evensen, 1994). An ensemble of model states is integrated forward in time. From the ensemble, all the statistical information together with statistical moments (such as mean and covariance) can be calculated.

The EnKF presents two important advantages as compared to the extended Kalman filter: It solves the closure problem presented in (Evensen, 1994) and reduces drastically the computational load and the storage requirement (Canizares 1999).

The Ensemble Kalman Filter scheme

As the model is non-linear, the definition of the process of the state space is as follows:

$$x_{k+1} = M(x_k) + w_k \quad (4.14)$$

where M is the non-linear model with the input state vector x_k . And the equation for the measurement is :

$$z_{k+1} = H(x_{k+1}) + v_{k+1} \quad (4.15)$$

The first procedure of using the EnKF is to create initial ensemble, then for each ensemble member, the procedure can be divided into two steps:

- Time or system update phase
- Measurement update phase : using measurement data to update the model output

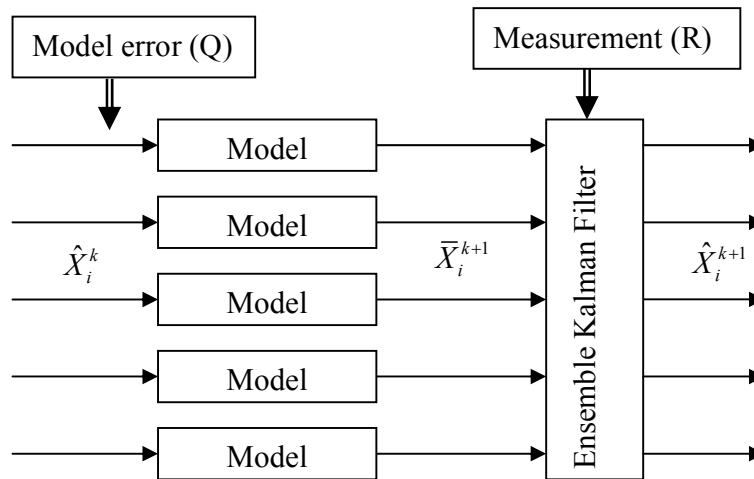


Figure 4. 2 The procedure of EnKF

Firstly, an initial ensemble has to be created. Consider an ensemble of size N of initial state $x_{0,i}$ with $i = 1, \dots, N$ is generated. The ensemble mean is an estimate of the best guess initial condition $E\{x_0\}$ and the ensemble covariance represents the uncertainty $P_0 = Cov\{x_0\}$ in the first guess initial state. It means using the covariance P_0 generate the white noise to perturb the state vector X_0 .

Given an error covariance matrix, an ensemble of finite size will always provide an approximation to the error covariance matrix. However, when the size of the ensemble N increases, the errors in the Monte Carlo sampling will decrease proportional to $1/\sqrt{N}$ (Evensen, 2003).

Secondly, the time update of each ensemble member is done by the following equation:

$$\mathbf{x}_{k+1,i} = M(\mathbf{x}_{k,i}) + \mathbf{w}_{k,i} \quad (4.16)$$

where M represents the non-linear model and the $\mathbf{w}_{k,i}$ is the white noise.

Any quantile of the ensemble forecast can be used as the forecast of the state vector (Canizares, 1999). (Evensen, 2003) uses the mean value, which is calculated as

$$\bar{\mathbf{x}}_{k+1} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_{k+1,i} \quad (4.17)$$

The expectation of the error in every ensemble member is formulated using:

$$E_{k+1} = [\mathbf{x}_{k+1,1} - \bar{\mathbf{x}}_{k+1}, \dots, \mathbf{x}_{k+1,N} - \bar{\mathbf{x}}_{k+1}] \quad (4.18)$$

The error covariance matrix of the forecast is obtained from:

$$P_{k+1} = \frac{1}{N-1} E_{k+1} E_{k+1}^T \quad (4.19)$$

The Kalman gain can be calculated as in Equation (4.9). Finally by comparing the measurement z_{k+1} to the model estimates the measurement update is:

$$\hat{\mathbf{x}}_{k+1,i} = \bar{\mathbf{x}}_{k+1,i} + K_{k+1} (z_{k+1,i} - H(\bar{\mathbf{x}}_{k+1,i})) \text{ for } i = 1, \dots, N \quad (4.20)$$

where $H\bar{\mathbf{x}}_{k+1,i}$ is the output calculated by the model for each ensemble member and

$$z_{k+1,i} = z_{k+1} + v_{k+1} \text{ for } i = 1, \dots, N \quad (4.21)$$

where z_{k+1} is the actual measurement vector and v_{k+1} is the measurement error generated from a distribution with zero mean and covariance matrix R_{k+1} . As introduced by (Burgers et al, 1997) it is essential to add a random perturbation to the measurements otherwise the assumption being random variables will not be valid

5 Methodology

This chapter describes the approach used to explore suitability of OpenMI for ensemble Kalman filtering. The theoretical background for each step of the approach is also discussed.

5.1 Approach

The approach used to explore the suitability of OpenMI for ensemble Kalman filtering is as follows.

First the models which are supposed to exchange data using the OpenMI interface are made OpenMI compliant, i.e. they are turned in to linkable components. Therefore, the HYMOD-RR model and the EnKF algorithms are wrapped to turn them in to linkable component.

Next the linkable components should be linked, that means links should be created between the exchange items of the two linkable components. However for ensemble Kalman filtering many instances of a linkable component for which the filter is going to be performed should be linked to the EnKF linkable component. Therefore the next step is to automatically establish these links.

Now, when these links are established, HYMOD-RR can exchange its variables with the EnKF so that the EnKF can make the updates and return the updated variables to HYMOD-RR. These variables will be input to the HYMOD-RR for the next time step. This cycle continues until the final time step for the HYMOD-RR is reached.

5.2 Making legacy models OpenMI compliant

An important requirement in the design of the OpenMI standard was that migrating legacy water simulation software code should be as easy as possible. Migration of a model should only require some small changes and not a complete rewrite in a different programming language on a different platform (Westen et al, 2004).

To migrate a legacy code an existing model engine (i.e. a computational core often developed in Fortran 90) is encapsulated in a wrapper that meets the interface specification of an OpenMI linkable component. The OpenMI interface allows it to be

treated in a generic way by an OpenMI run-time environment. Actually, the 'wrapper' turns the computational core into a linkable component for OpenMI (Gijssbers, 2004).

In order to facilitate model migration, the OpenMI consortium implemented a default wrapper with all the necessary functionality to interface with the OpenMI framework. This default wrapper, the “SmartWrapper”, implement the Linkable component interface. The SmartWrapper talks to the computational code using an interface that is called the EngineAPIAccess interface in the OpenMI project.

The main requirement of the user of the wrapper is to understand how the model is going to be used while it is run in the OpenMI environment.

Then the task of making the model OpenMI compliant is then to;

1. Reorganize the model engine so it can be compiled to a WIN32 dll which exports functions close to the methods in the IEngineAPIAccess,
2. Create a .Net component that implements the IEngineApiAccess and uses the engine core dll through the WIN32Api.

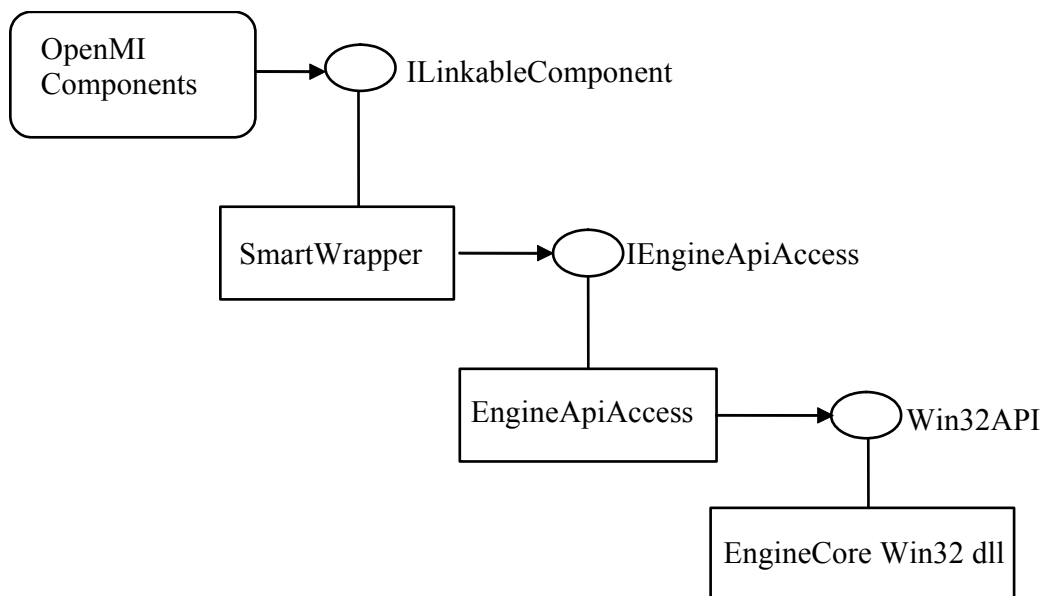


Figure 5.1 The relation between the OpenMI components and the Engine core
 [Source: adopted from (Sinding et al, 2004)]

OpenMI components will access the SmartWrapper through its ILinkableComponent interface. The SmartWrapper will access the EngineApiAccess object through the IEngineApiAccess interface, and the EngineCore will be accessed through the WIN32API.

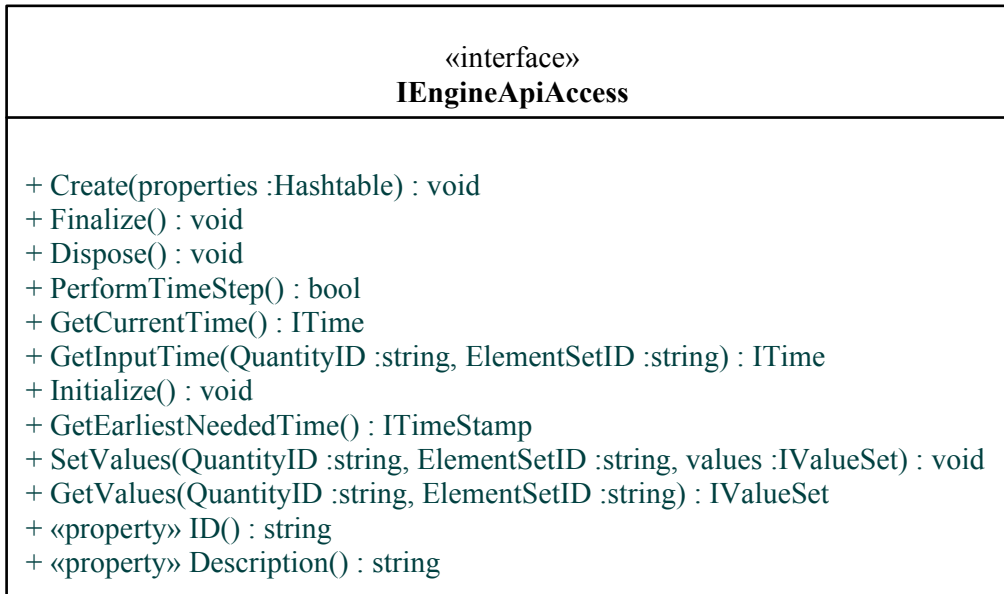


Figure 5.2 The EngineAPIAccess interface
 [Source: adopted from (Sinding et al, 2004)]

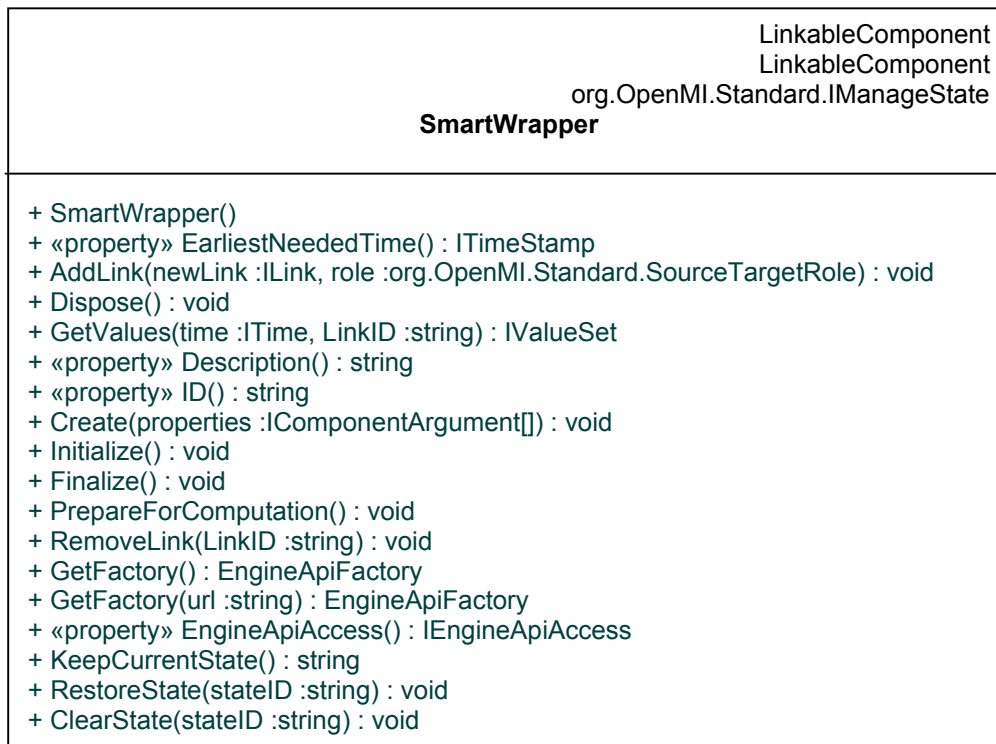


Figure 5.3 The SmartWrapper class
 [Source: adopted from (Sinding et al, 2004)]

Normally a model application consist of many parts, the most common being the user interface, input files, the engine and output files. The engine is where the calculations take place. The user supplies information through the user interface upon which the user interface generates input files for the engine. The user can run the model simulation e.g. by pressing a button in the user interface, which will deploy the engine. The engine will read the input files and perform calculations and finally the results are written to output files.

The approach for OpenMI is to access the model directly at run time and not to use files for data exchange. In order to make this possible, the engine needs to be turned into an engine component and the engine component needs to implement an interface through which the data inside the component is accessible

However legacy codes need to be structured in a way that initialization is separate from computation, where boundary conditions are collected in the computation phase and not during initialization.

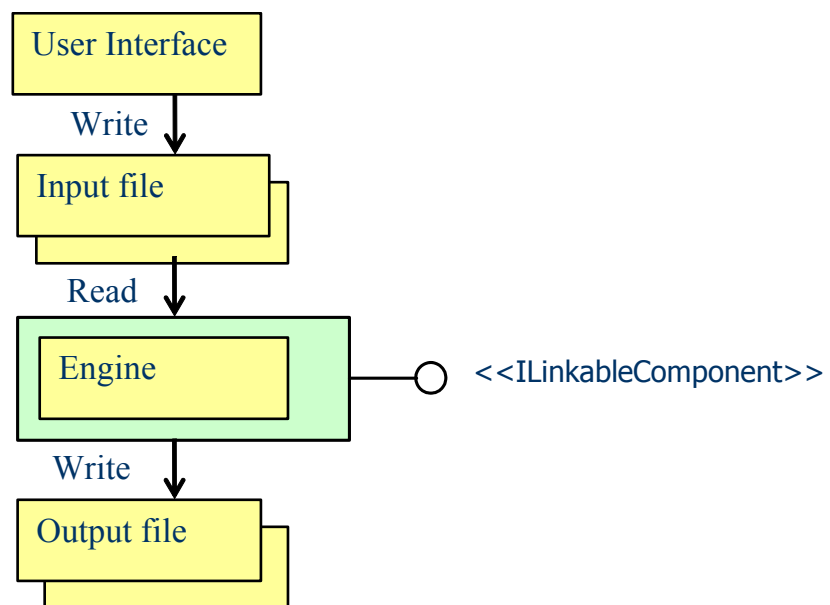


Figure 5.4 Typical Model Architecture when using OpenMI

The first step in migrating legacy models is to separate reading the static data (schematization) from the time-step calculation and reading of dynamic data (boundary

conditions). Instead of calculating all time-steps at once the code should be able to compute just one time-step and return.

The next step is to convert the main loop into a subroutine that can execute one time-step at a time. This may require making some local variables static because otherwise their value will be lost when the subroutine returns.

The following step is to make sure the input data can be transferred to the computational code. This will usually involve replacing the file reading routines with a small buffer to hold the boundary conditions for the current time-step. The results for each time-step should also be kept in a small buffer in the code.

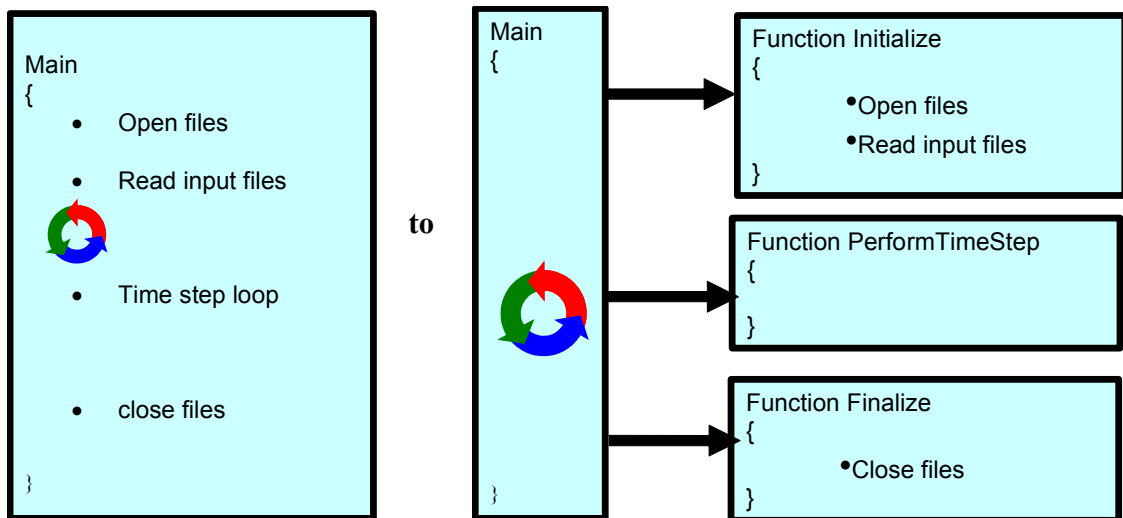


Figure 5.5 Changes to the engine core

5.3 Exchange Items between HYMOD and the EnKF

To apply the EnKF to the HYMOD model, the model must be able to pass the state variables and the measurements at time $k+1$, after performing the timestep calculation at time k , to the EnKF. The EnKF should also pass the states to the model after performing its update, thus these will be inputs to the model for the next time step.

While making the model, for which the ensemble Kalman filtering is to be performed, OpenMI compliant, one should bear in mind that the model should be able to exchange all its state variables and the observed values and the model output as well because the

ensemble Kalman filtering process is a combination of the model output and measurement data to make a better prediction for the future.

The input exchange items for HYMOD are the five model states as described in section 3.2 and the measurements of rainfall, evapotranspiration and the discharge and the calculated discharge. The HYMOD model then performs its time step computation and outputs these exchange items for the next time step. These outputs from the HYMOD then will be input exchange items for the EnKF.

The outputs of the EnKF, which are the updated values, then will be the input exchange items for HYMOD for the next time step

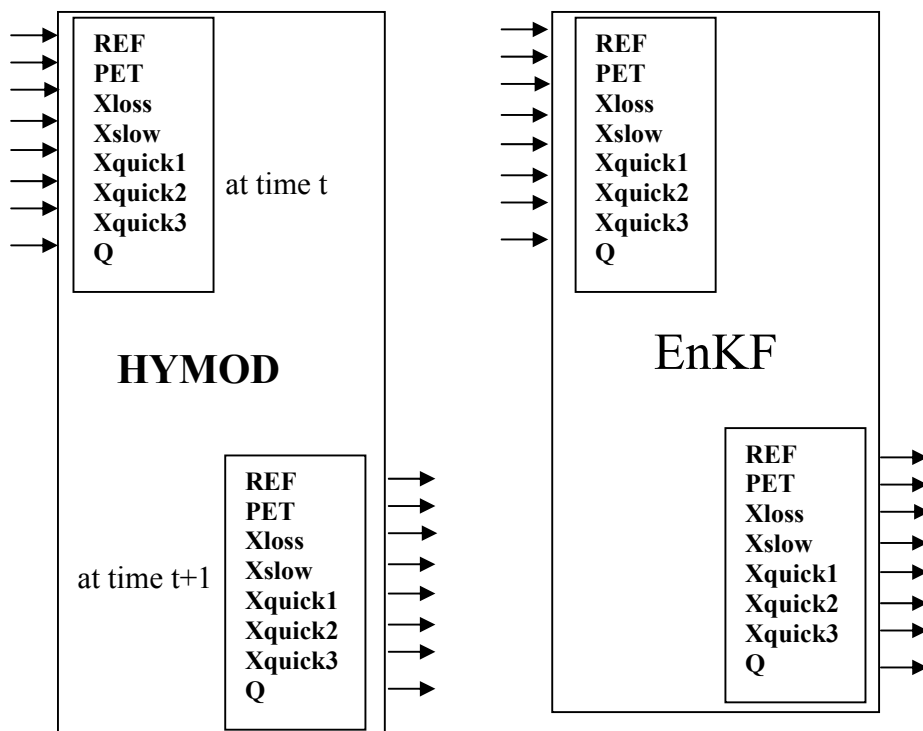


Figure 5.6 Exchange items between HYMOD model and EnKF

5.4 Configuring an OpenMI composition for EnKF

The data exchange and synchronization mechanism of OpenMI is designed in such a way that LinkableComponents can autonomously exchange data without any centralized functionality to manage the data exchange. However, in some specific situations, extra control capacity is needed to direct convergence of computational results. This functionality typically is desired for iteration purposes, as well as for optimization and calibration (Sinding et al, 2004).

OpenMI currently does not support a linking mechanism of many instances of a linkable component to a single another linkable component without an additional controller. However, for application of the ensemble Kalman filtering, instances as many as the number of ensembles, of the linkable component for which the Kalman filtering is to be performed, need to be linked with one ensemble Kalman filter linkable component.

To be able to link many instances of a linkable component with the ensemble Kalman filter linkable component, a separate configuration class is needed. This separate configuration class should be able to provide a generic mechanism for automatic creation of the links between the EnKF linkable component and the RR linkable components as many as number of ensembles.

The following tasks should be performed by the configuration class for automatic creation of the links;

1. Instantiate as many RR linkable components as the number of ensembles;
2. Query the exchange items of the RR linkable component;
3. Modify the quantityIDs and elementSetIDs of the exchange items according to the number of ensembles for the purpose of making the quantity ids of each of the RR linkable components unique. This can be done by attaching the ensemble number to both the quantityIDs and elementSetIDs ;
4. Create exchange items for the EnKF linkable component;
5. Create and add links between output exchange items of the RR model linkable components and input exchange items of the EnKF linkable component with similar semantics;
6. Create and add links between output exchange items of the EnKF linkable component and input exchange items of the RR model linkable components with similar semantics;
7. Instantiate a computation of the composition by invoking the GetValues() method

6 Results and Discussion

This chapter presents the results obtained in this study. This includes making the HYMOD-RR model and the EnKF algorithm linkable components and configuring an OpenMI composition for EnKF. Discussion of the results and difficulties encountered are also presented here.

6.1 Making the models Linkable components

As it is discussed in the methodology chapter the approach, the first thing to do for configuring an OpenMI composition for ensemble Kalman filtering is to make the model for which the Kalman filtering is to be performed (in this case a conceptual rainfall-runoff model called HYMOD) and the EnKF algorithm OpenMI components.

6.1.1 HYMOD

The HYMOD-RR code which was originally written in FORTRAN 90 is made a linkable component by performing the following steps.

- Step 1. Identifying the exchange items. To make either input or output correction for the RR model, it is required to exchange the items, in addition to the five state variables as described in section 3.2, the measurements of rainfall, evapotranspiration and the discharge and the calculated value of the discharge by the RR model;
- Step 2. The model code is restructured so that initialization, time step calculation and finalization parts are separate subroutines. To set values of the state variables and get the computed values of the state variables with the measured values for the time step two subroutines are added. Functions to return the current time of the model engine and the time for which the next input is needed are also added;
- Step 3. A dynamic link library (dll) is prepared to export all the subroutines and functions to the C# platform;
- Step 4. Writing the EngineApiAccess code to implements the IEngineApiAccess interface which actually creates the communication between the wrapper and the computational code. The EngineApiAccess is a very thin wrapper

which main responsibility is to make the engine core dll behave like a .Net component that implements the IEngineApiAccess interface. The IEngineApiAccess interface is much closer to the interface as compared to the ILinkableComponent interface (Sinding et al, 2004);

Step 5. Preparing the LinkableComponent code which implements the ILinkableComponent interface.

The relationship among the wrapper, the EngineApiAccess and the engine core for the HYMOD-RR is shown in Figure 6.1.

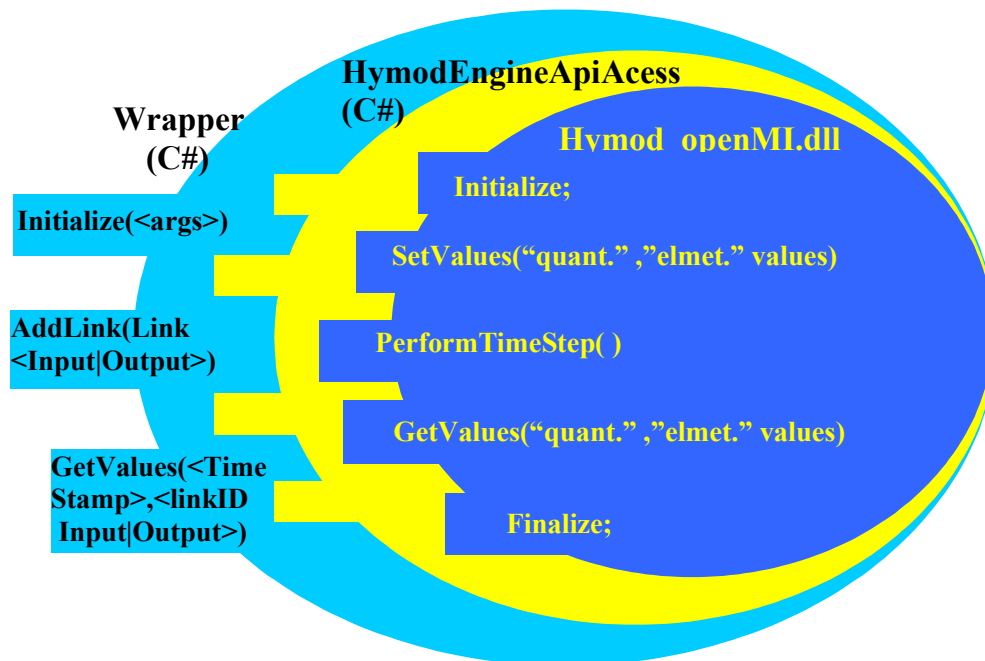


Figure 6.1 Relation among the wrapper the EngineApiAccess and the engine core for Linkable HYMOD

6.1.2 EnKF

The ensemble Kalman filter algorithm is also made OpenMI compliant in a similar way as HYMOD-RR except that an engine core dll was not prepared. Instead executable file of the EnKF which was customized for the HYMOD-RR model is used. This executable of the EnKF needs input files containing values of the state variables and the measured variables including the discharge for each ensemble. The EnKF executable reads input files which actually contain the model outputs from each instance of the HYMOD-RR

linkable component, made the update and writes the updated values in to the same files used as input files.

Therefore, the EngineApiAccess for the EnKF is prepared to perform the following tasks;

1. Prepare the files used to initialize the EnKF;
2. Prepare the input files by writing the model outputs from each instance of HYMOD-RR linkable component to a separate input files;
3. Start a process to execute the EnKF executable in the command line and ;
4. Reads the update values of exchange item from the output files to make them accessible for the wrapper.

The relationship among the wrapper, the EngineApiAccess and the EnKF executable is shown in Figure 6.3.

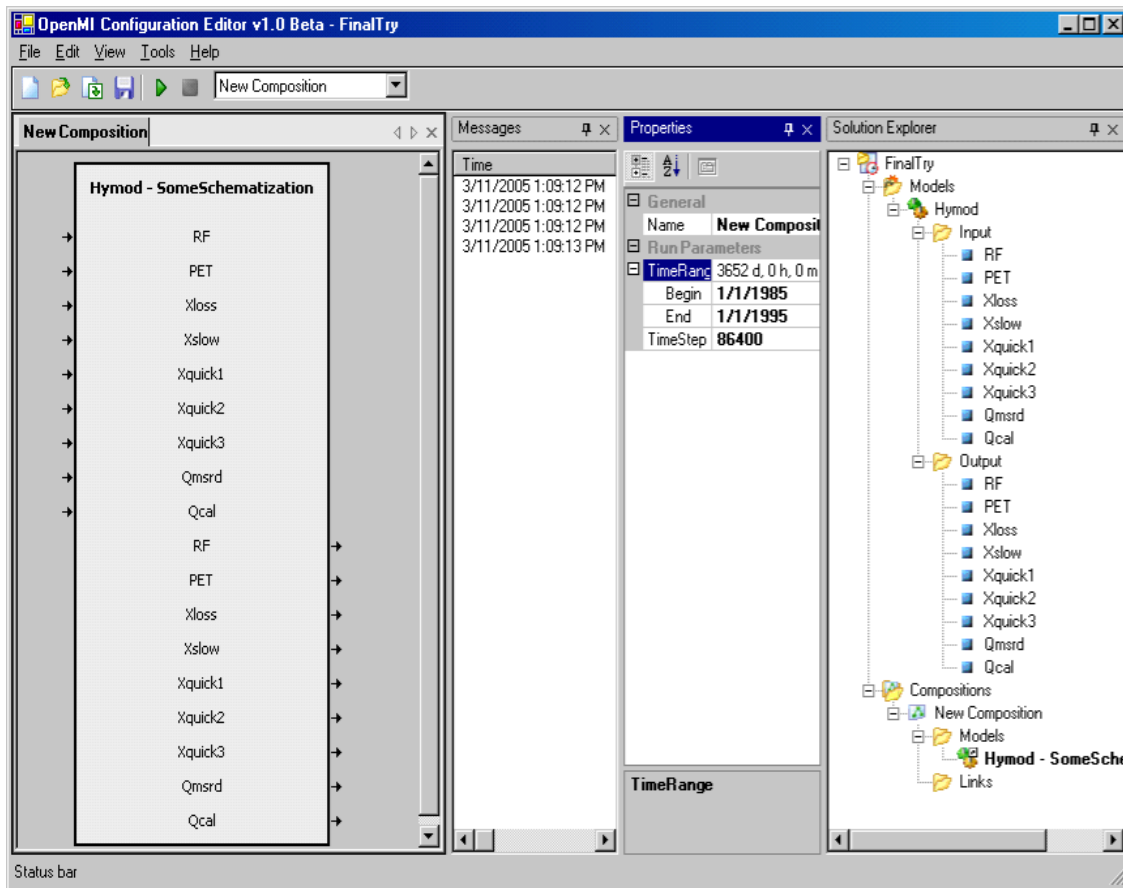


Figure 6.2 Linkable HYMOD in the OpenMI Configuration Editor User interface

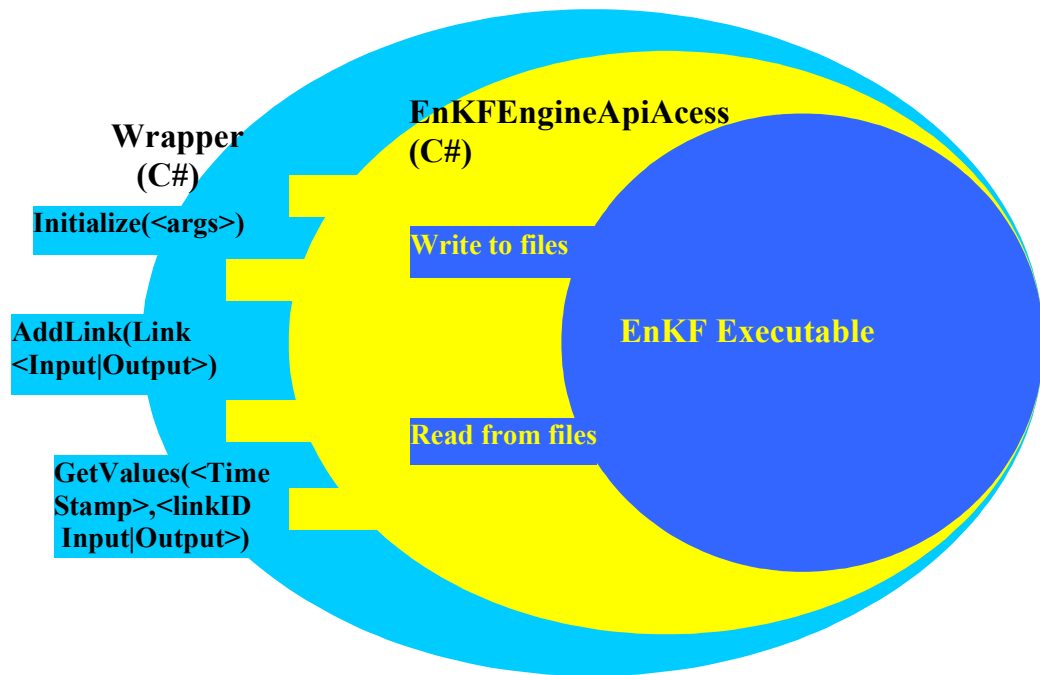


Figure 6.3 Relation among the wrapper the EngineApiAccess and the engine core for Linkable EnKF

6.2 Configuring OpenMI for EnKF

As it is explained in section 5.4, OpenMI does not support a linking mechanism of many instances of a linkable component to a single another linkable component without additional controller.

Therefore, an additional class is developed to configure the OpenMI composition for ensemble Kalman filtering. This class, called the EnKFConfiguration class, itself is a linkable component providing default implementation for most of the methods in ILinkableComponent interface. All the tasks outlined in section 5.4 are implemented in the initialize () and GetValue () methods.

The EnKFConfiguration is like a hard-coded system. Hard-coded systems are those systems where the establishment of links and the deployment and execution of a combination of components is completely encapsulated in the source code (Blind M et al, 2004). The establishment of links between the exchange items of the EnKF linkable component and the HYMOD linkable component instances, deployment and execution of their combination is completely encapsulated in the EnKFConfiguration class.

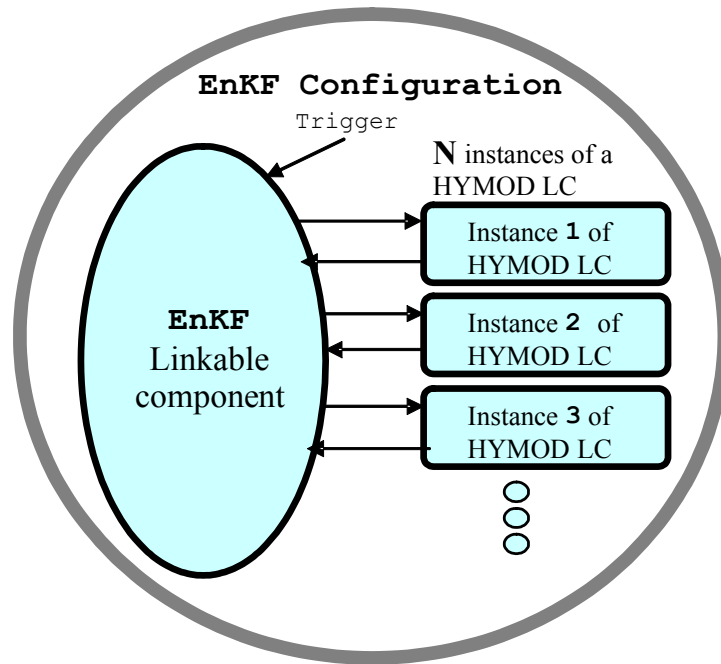


Figure 6.4 Schematic diagram of the EnKF Configuration

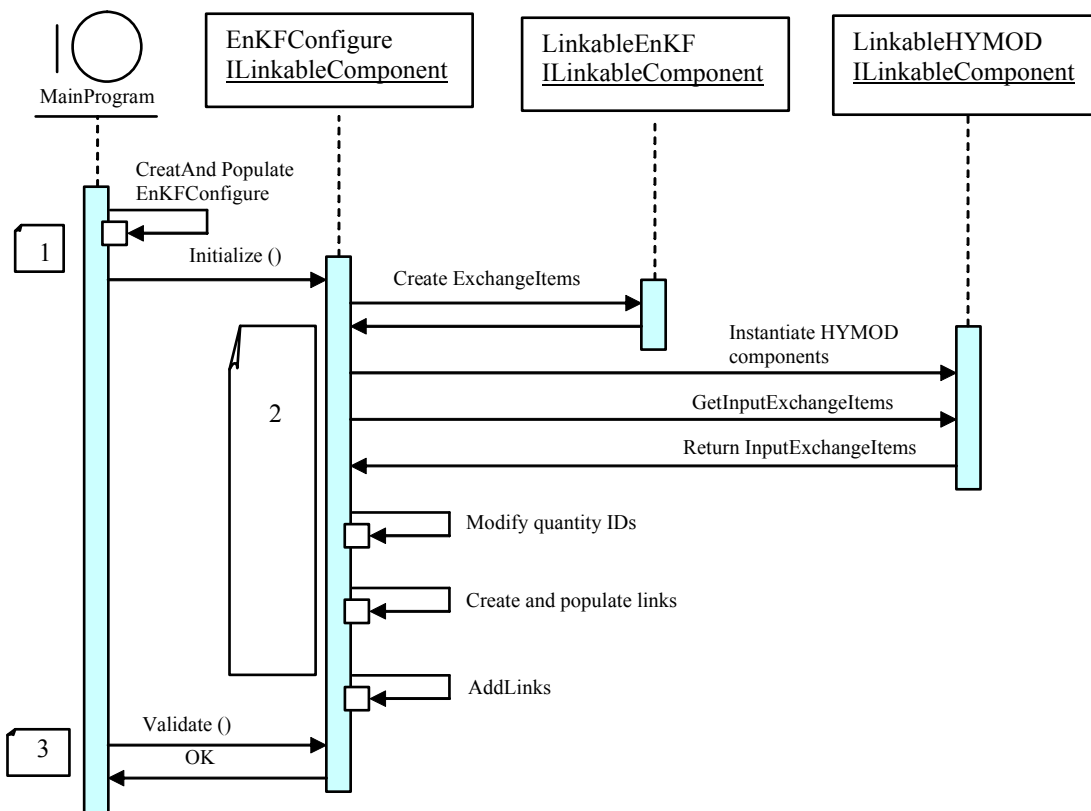


Figure 6.5 Sequence diagram of the EnKF Configuration

step 1 Create and populate the EnKFConfiguration

step 2

- Instantiate the HYMOD linkable components;
- Instantiate the EnKF linkable component;
- Initialize the linkable components with proper input data;
- Create Quantity and ElementSet objects by querying ExchangeItems;
- Create and populate links;
- Add links between similar quantityIDs and elementSetIDs

step 3 Validate the EnKFConfiguration LinkableComponent

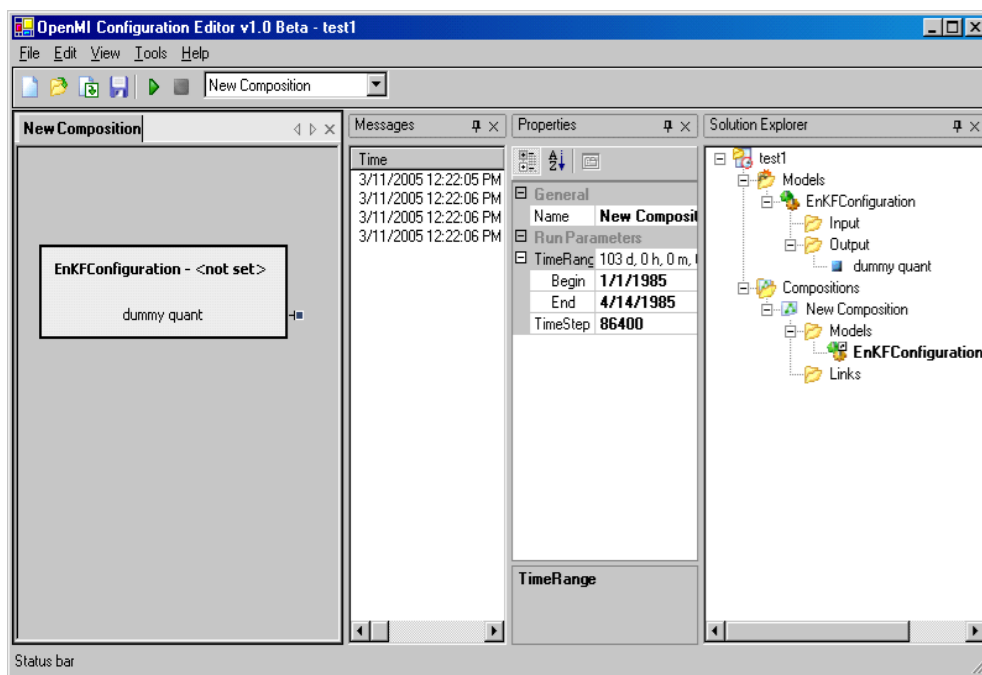


Figure 6.6 EnKF Configuration in the OpenMI Configuration Editor User interface

6.3 Discussion of the results

Before migrating the models or in other words making them OpenMI compliant, it is important to have a clear understanding of the intended use of the model when it is running as OpenMI component. This will help in identifying the exchange items and in restructuring the model.

In the present case, the purpose of making the models (both HYMOD-RR and the EnKF) OpenMI compliant is to make possible the ensemble Kalman filtering process for the RR model in OpenMI environment.

To make either input or output correction for the RR model, it is required to exchange the items, in addition to the five state variables as described in section 3.2, the measurements of rainfall, evapotranspiration and the discharge and the calculated value of the discharge by the RR model.

The link between the EnKF linkable component and the HYMOD Linkable components is bi-directional. When the composition is run, the triggering component is the EnKF. It therefore asks the HYMOD-RR linkable components for the values of the exchange items, the HYMOD-RR linkable components asks back the EnKF linkable component for the same. However at the initial time, since no computation has started, the buffer is empty and this creates an exception. To overcome this problem the EnKF linkable component is made to provide initial values of the exchange items the same for all the ensembles. These initial values are read from a file.

Though the main idea of OpenMI is not to use files for data exchange, in the case of EnKF linkable component files are used to set and get values for the exchange items. This is because at the time the code of the EnKF was not yet available to produce the engine core dll. As a result many files have to be opened and closed specially as the number of ensembles increases. Exception of the “file is being used by another process” has been encountered few times. This problem can be avoided by making an engine core dll not to use files for data exchange. However, this has also proved that, it is not mandatory to make a dll engine core to make models OpenMI compliant using the wrapper provided by the OpenMI environment as long as the model can execute for a time step and return the values needed.

HYMOD-RR is a simple lumped conceptual rainfall-runoff model. This means that the whole catchment area is treated as a point in space. For this reason the number of element sets is one. However for more complex physically based models the number of element sets would be more than one. In this case, the way input files are read and output files are written would also be different. Therefore, preparing an engine core dll for the EnKF is necessary to make a generic linkable component.

7 Case Study

This Chapter describes the case study undertaken to test the configuration of OpenMI for ensemble Kalman filtering. The logical way to prove whether the configuration works properly, that means to prove actually the link between the EnKF and the instances of the HYMOD-RR is created and the models be able to exchange data using the OpenMI interface, is to undertake a case study and compare the results with the results obtained from application of the EnKF to HYMOD-RR in a batch file.

7.1 The data

The case study used for this purpose is the application of EnKF to the HYMOD-RR model of the Leaf River watershed. The Leaf River watershed is a humid 1944 km² catchment located north of Collins, Mississippi. HYMOD-RR was applied to model the rainfall-runoff process of the watershed. The measurement data that was available and used is the same as the data used in Vrugt et al (Vrugt et al, 2003). It consists of hydrologic data of the Leaf River watershed approximately for a period of 11 years (28 July 1952 to 30 September 1962). The data, which was obtained from the National Weather Services Hydrology Laboratory (HL), consists of mean areal precipitation (mm/d), potential evapotranspiration (mm/d), and stream flow (m³/s).

For most of the considered period HYMOD-RR model performs very well, therefore using EnKF would not really be necessary. Nevertheless there are some periods of extensive rainfall, where the model does not perform so perfectly (Koster, 2004).

Figure 7.1 shows the observed discharge versus the predicted discharge by the model for the whole period and Figure 7.2 shows the same for the period in which the model does performance is low, this is the period from the data points 3100 to 3200.

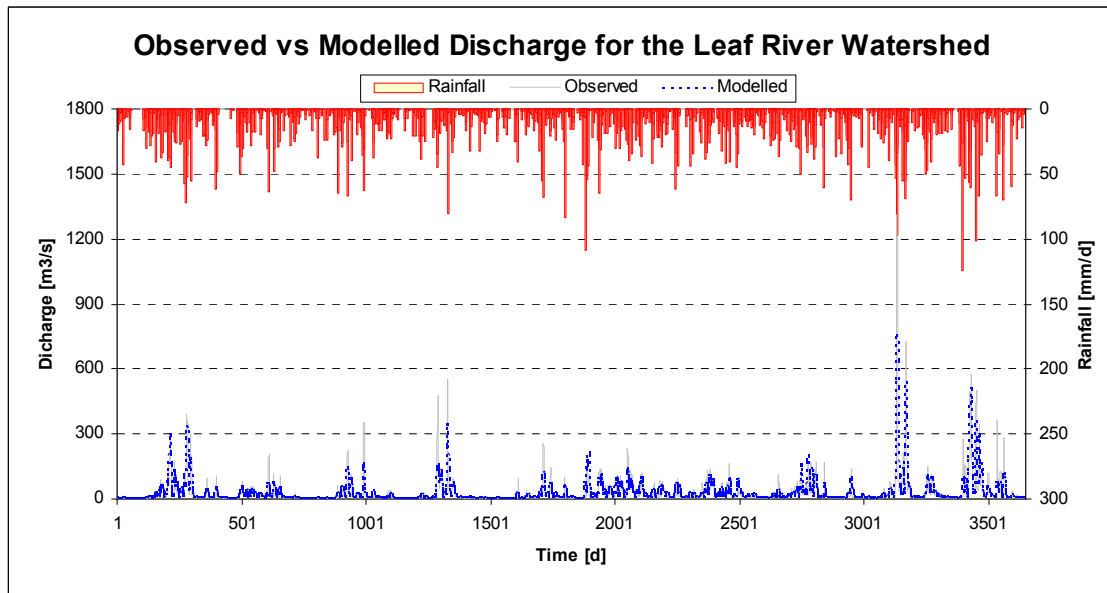


Figure 7.1 Observed versus modelled discharge for the Leaf River watershed

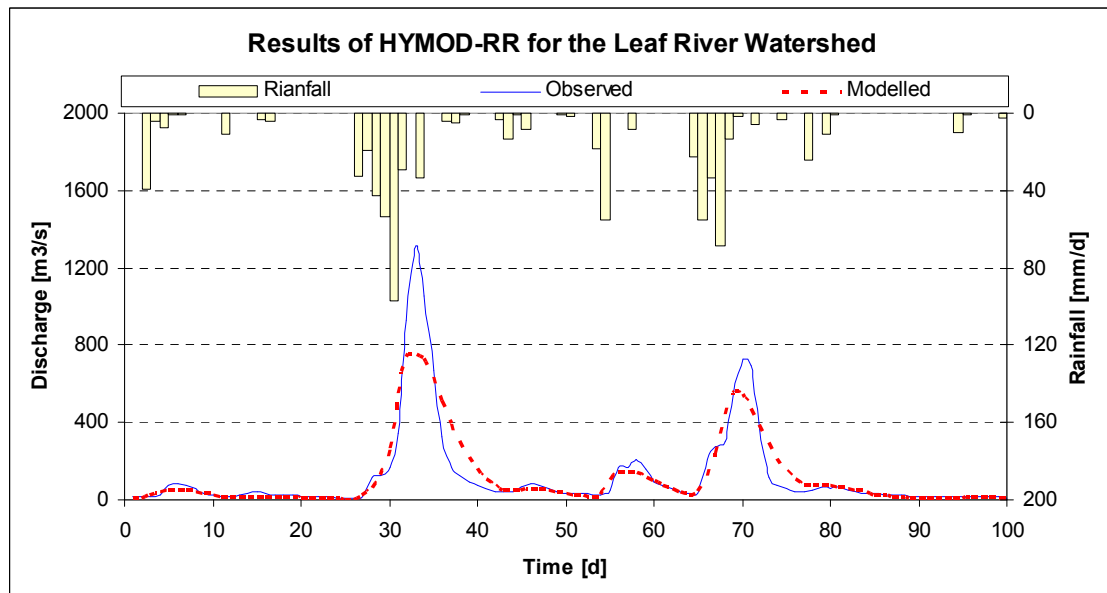


Figure 7.2 Observed versus modelled discharge for the Leaf River watershed

7.2 Application of the EnKF

Ensemble Kalman filtering is performed for output correction of the HYMOD-RR model for the Leaf River watershed for a selected period where the model showed low performance. This period is from the data point 3100 to 3200 as plotted in Figure 7.2.

Standard deviations for the observed discharge (measurement error), for the modelled discharge and for the five model states (model error) are assumed arbitrarily to have the values shown Table 7.1.

Table 7.1 Standard deviation for the states

States	Standard deviation
Q observed	0.01
Xloss	0.0
Xslow	0.0
Xquick1	0.1
Xquick2	0.0316
Xquick3	0.0316
Q calculated	0.0

First, the EnKF is applied in a batch file, taking the number of ensembles to be 30. Then the EnKF is applied using the OpenMI composition for EnKF for the same number of ensembles. The plots of the observed discharge, the model output without the update and the result after the update are shown in Figure 7.3 and 7.4 respectively for the batch file method and for the OpenMI composition for EnKF. The values of the five states are also plotted for each method in Figure 7.5 and 7.6 respectively.

As it is evidently shown from the comparison of the plots and from the computed root mean squared errors (RMSE), the two methods have shown almost identical results. This proved that the link between the instances of the HYMOD-RR and the EnKF is actually created and the linkable components (the models) exchange their data using the OpenMI environment.

Table 7.2 Root mean squared errors

Method of EnKF application	RMSE
Batch file	55.55852
OpenMI environment	55.12199
Model without update	92.00113

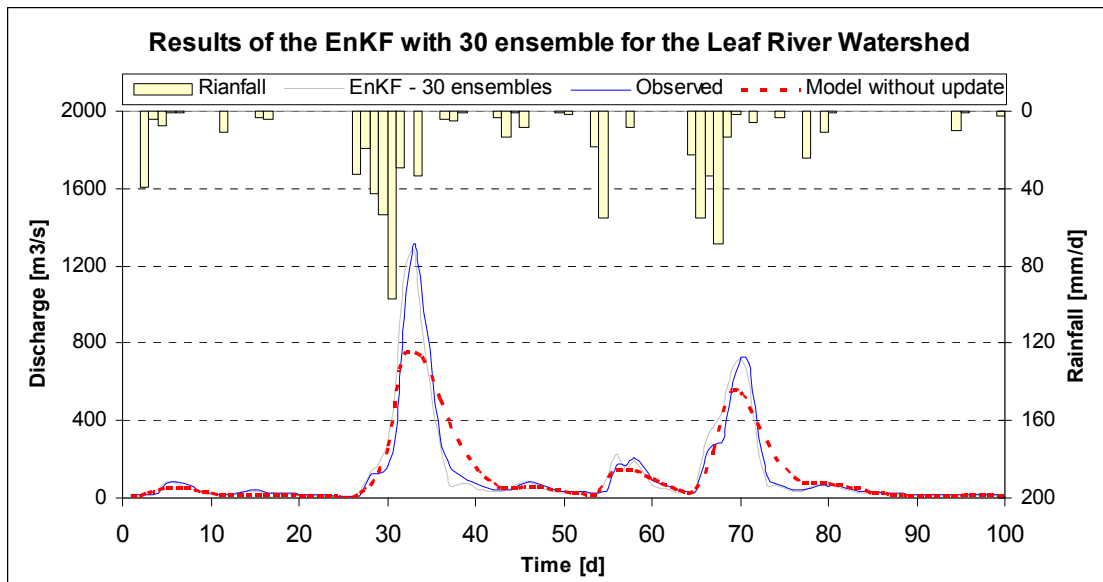


Figure 7.3 Observed, modelled and updated discharges for the batch file method

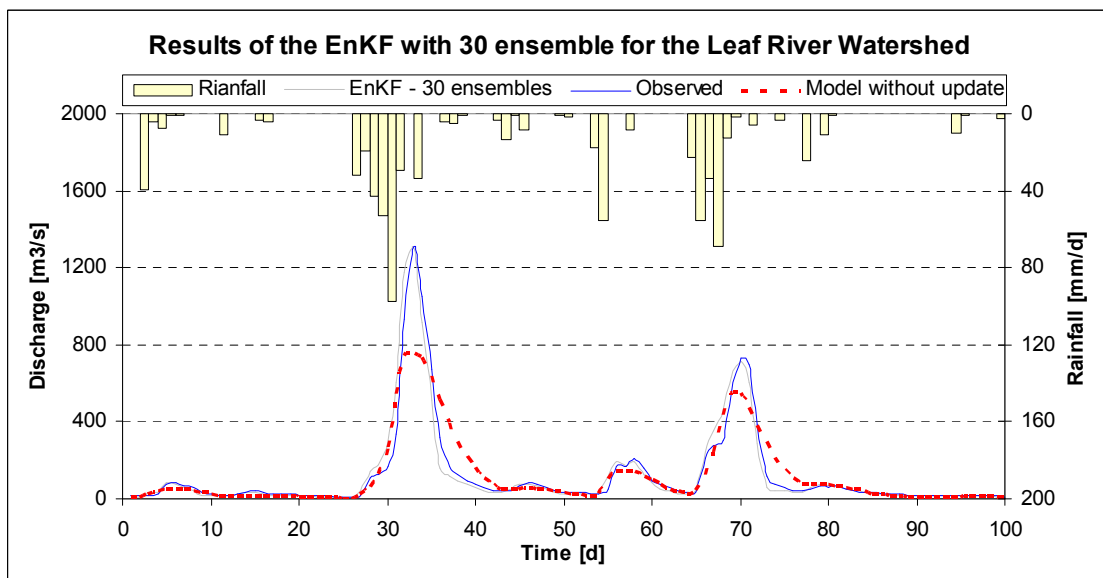


Figure 7.4 Observed, modelled and updated discharges for the OpenMI composition

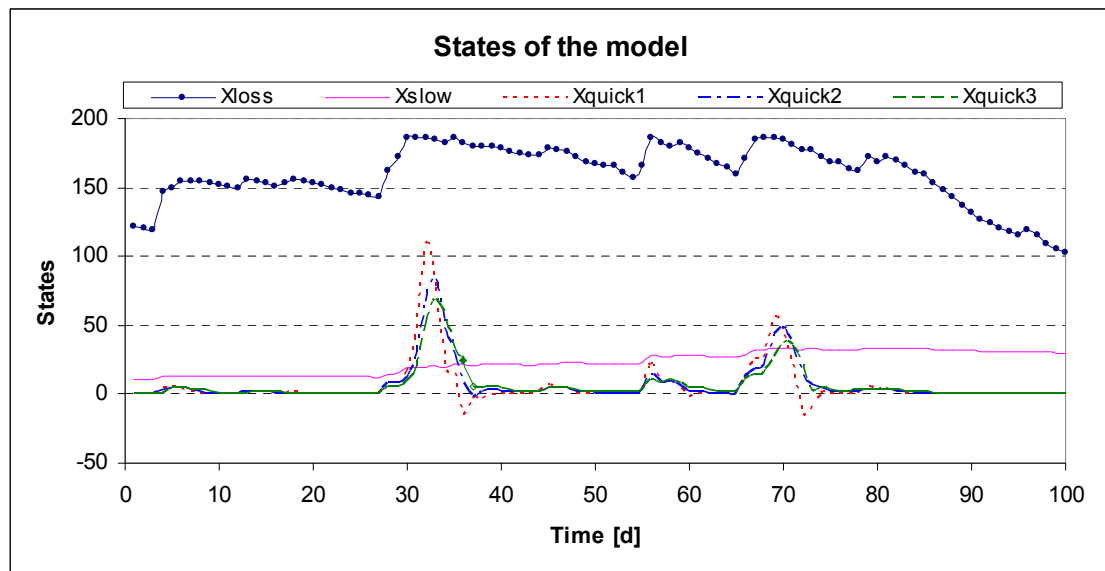


Figure 7.5 Values of the state variables for the batch file method

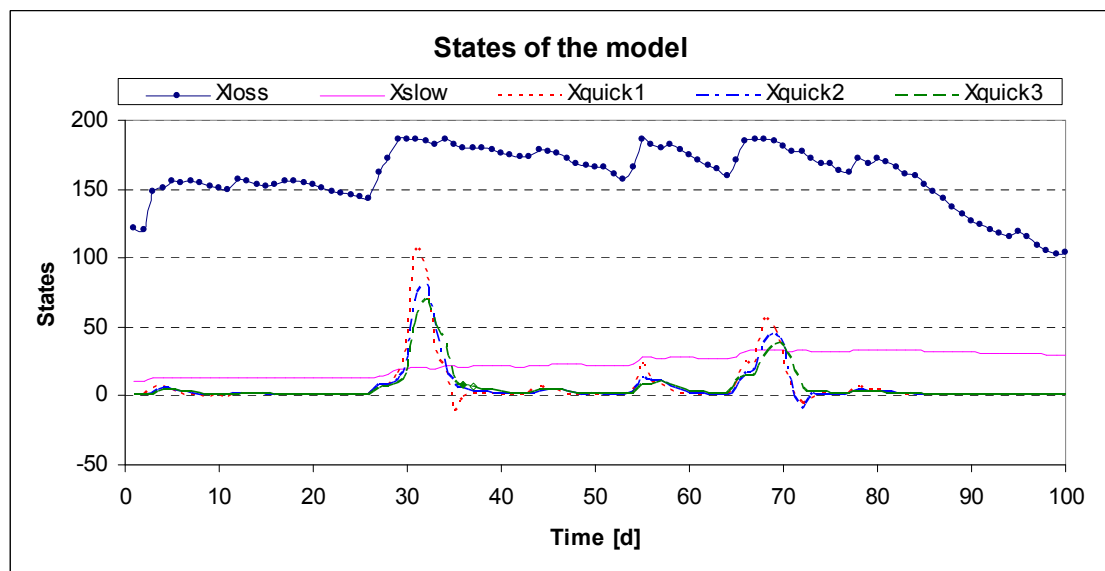


Figure 7.6 Values of the state variables for the OpenMI composition

The OpenMI composition is also found to be faster than the Batch file method. In the considered case more than 40% less computational time was required for the OpenMI composition. This is due to the fact that in the OpenMI composition case the rainfall-runoff components initialized only once at the initial time step where as in the batch file method case the rainfall-runoff model initialized every time step for each ensemble.

After proving that the OpenMI composition for EnKF works properly, the composition was run for different number of ensembles. Ensembles of 5, 10, 30, 50 and 99 were tried.

As it is expected, the updated discharge became closer to the observed one as the number of ensembles increased. The computation time also increased as the ensembles increased. However, the percentage of reduction of the RMSE reduces as the ensembles increased. The RMSE reduced by 14.26% when the number of ensembles increased from 10 to 30, while it reduces only by 4.05% when the number of ensembles increased from 50 to 99.

Table 7.3 Root mean squared errors corresponding to the number of ensembles

No. of Ensembles	RMSE	% Reduction of RMSE
without update	92.00113	0.000
5	75.153	18.313
10	64.43219	14.265
30	55.12199	14.450
50	49.2958	10.570
99	47.30092	4.047

The plots of the observed discharge, the model output without the update and the result after the update are shown in Figure 7.7 and 7.8 respectively for the number of ensembles equal to 10 and 99.

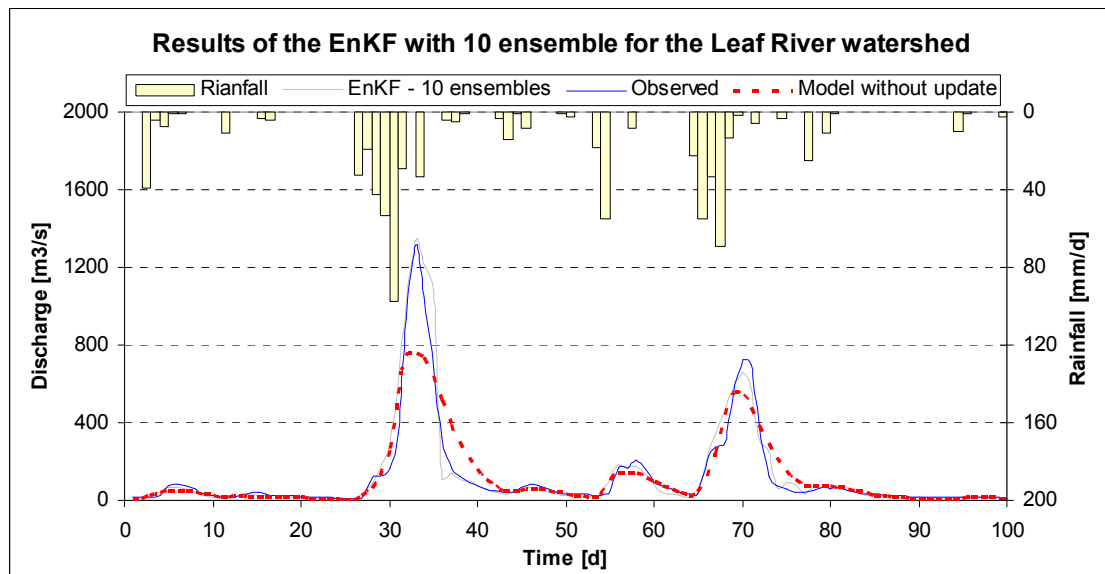


Figure 7.7 Observed, modelled and updated discharges for 10 ensembles

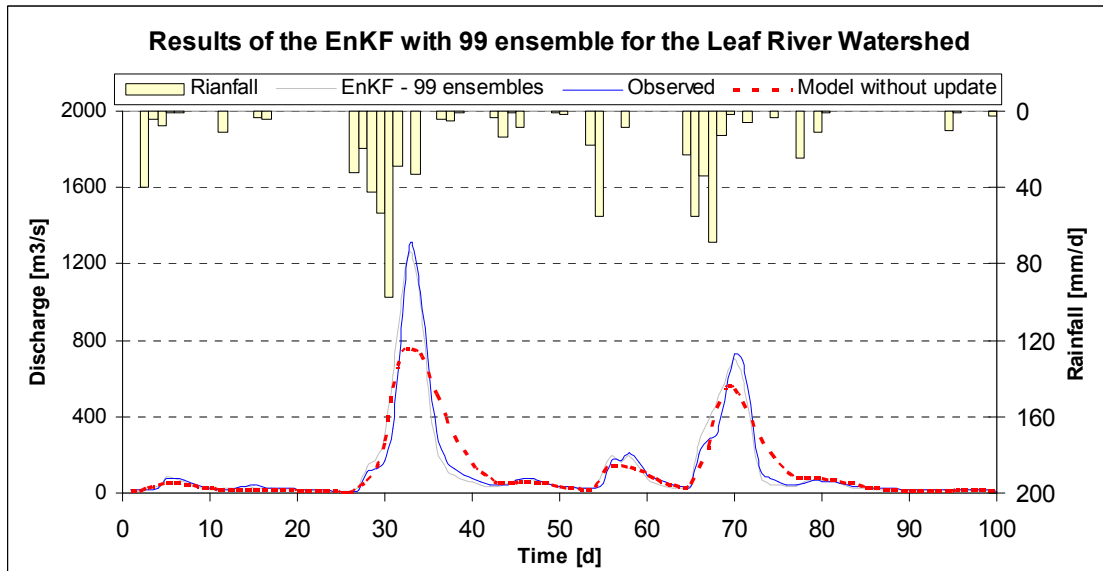


Figure 7.8 Observed, modelled and updated discharges for 99 ensembles

8 Summary, Conclusions and Recommendations

This chapter presents (i) a short summary of the study, (ii) conclusions drawn from the study, (iii) some recommendations for further study on issues which couldn't be accomplished in the present study due to shortage of time and scarcity of resources and (iv) few recommendations for the HarmonIT project from an end user point of view.

8.1 Summary

Integrated water resources management has created a need to understand and model catchment processes and particularly their interactions. Since the processes involved in catchment hydrology are complex and no single model or modelling system can adequately represent all the processes in the catchment environment, a set of distinct individual models representing the individual activities in the catchment are required to be interconnected in a similar way as the activities are interconnected.

To address this complex and fundamental issue of model linking, the HarmonIT project, a research project funded by the European Commission, is developing and implementing a European Open Modelling Interface and Environment (OpenMI).

Data assimilation is the efficient and often used method to reduce model uncertainty. Ensemble Kalman filter (EnKF) is one of the most well known data assimilation methods. EnKF has been widely used in reducing uncertainty in hydrologic model forecasts. This is done in most of the models in ad hoc basis in a batch file or by implementing the EnKF in a dedicated way in modelling packages.

OpenMI offers the mechanisms for data exchange between models at run time. The EnKF algorithm treated as a separate model may be linked with other models in the OpenMI environment for the data assimilation process. The suitability of the OpenMI environment for the application of ensemble Kalman filtering has been explored in this study.

For linking models using the OpenMI environment, the models have to be OpenMI compliant. Model components would be OpenMI compliant when they implement the OpenMI standard interface. The OpenMI environment provides a default wrapper to facilitate the process of making legacy models OpenMI compliant. Using this wrapper, which implements the ILinkable interface of the OpenMI standard, and by creating a .Net component that implements the IEngineApiAccess interface, a rainfall-runoff model called HYMOD and the EnKF have been made OpenMI compliant.

Application of EnKF requires as many computations of the model as the number of ensembles at a time step. Therefore, many instances of the linkable component for which the ensemble Kalman filtering is performed, need to be linked and be able to exchange data with the EnKF linkable component. However, the OpenMI environment currently does not support this kind of linking with an additional controlling mechanism.

An additional controlling feature, the EnKFConfiguration class, has been developed to configure an OpenMI composition for EnKF application. This configuration class performs the following tasks;

1. Create and initialize the HYMOD-RR linkable component as many as the number of ensembles;
2. Initialize the EnKF linkable component;
3. Modify the QuantityIds and elementSetIDs to make the them unique for each instances of the HYMOD-RR linkable component;
4. Create the ExchangeItems for the EnKF linkable component; and
5. Establish the links between exchange items of similar semantics;

The OpenMI composition for EnKF has been tested if it has accomplished the required task that is whether the EnKF and the HYMOD-RR linkable components exchange their data in the way required for the ensemble Kalman filtering process or not. This has been done by comparing the results of the ensemble Kalman filtering process for HYMOD-RR model of the Leaf River watershed using the OpenMI composition for EnKF and the batch file methods. The two methods have produced the same results. This proved that with a relatively simple additional configuration class, the OpenMI environment is suitable for application of ensemble Kalman filtering.

8.2 Conclusions

In this research an attempt has been made to explore the suitability of OpenMI for ensemble Kalman filtering. From the study it has been found that an additional controlling feature is required to configure an OpenMI composition for ensemble Kalman filtering. The OpenMI environment currently without an additional controlling feature does not support linking of several instances of a linkable component to other single linkable component, which is the main requirement for ensemble Kalman filtering process.

The tools and utilities provided by the OpenMI environment to migrate legacy models are found to be very useful in making the models OpenMI compliant and are relatively easy to use them than rewriting the whole model codes from scratch in a new programming language.

The OpenMI composition for EnKF has advantage over the batch file method for ensemble Kalman filtering because;

1. In the OpenMI case the HYMOD-RR needs to be initialized only once at the initial time step while in the case of the batch file method the model has to be initialized each time step;
2. By preparing a dll for the EnKF the need for opening and closing files at each time step to set and get the values of the exchange items can be avoided and therefore the computational time can further be reduced significantly.

Making OpenMI suitable for ensemble Kalman filtering by just developing a configuration class can be taken as part of a demonstration that OpenMI has achieved its goal of providing a mechanism for data exchange between models at run time.

8.3 Recommendations

8.3.1 Recommendations for further study

1. Although the idea of using OpenMI is not to use files for data exchange, in the case of the present study for EnKF files are used. Therefore, it is recommended to develop an engine core dll for the EnKF code and avoid using files for the data exchange. By doing so the computation time can be reduced significantly because the need to open and close many files to read and write the values of the exchange items can be avoided;
2. In the present study HYMOD-RR which is a simple conceptual rainfall-runoff model is used. Since it is demonstrated that the OpenMI environment is suitable for ensemble Kalman filtering, it is recommended to try the same for a more complex physically based models such as SOBEK-RR and others;
3. The OpenMI composition for EnKF always starts from the initial time and run till the end time step specified in the graphical user interface. There may be a need to stop the process somewhere in the middle and continue from where it stops later. In this kind of situation, the values of the exchange items at the time the composition stops should be saved and used as initial values for the next run. Adding this feature in the OpenMI composition is recommended.
4. In the OpenMI composition for EnKF, the filter can be activated or deactivated. However, the whole instances of the model run even the filter is deactivated. When the filter is deactivated, only one instance of the model run is sufficient

because all the instances are produce the same results Therefore it is recommended to add features to be able to run only one instances of the model when the filter is deactivated.

8.3.2 Recommendations for the HarmonIT project

1. For ease of using the wrapper provided by the OpenMI environment, it would be advisable to include examples and detailed procedures to prepare and implement the EngineApiAccess and LinkableComponent classes in the guide book;
(after completion of the thesis it was noted that this is included in the guide book)
2. To implement a generic EnKF configuration class as part of the advanced control package of OpenMI; in this way it can be possible to use this class with any other linkable component for which ensemble Kalman filtering is to be performed;
3. To use the OpenMI environment comfortably a very good understanding of object oriented programming is required by the user. It would be quite useful for end users if this dependency on objected oriented programming skill can be minimized as much as possible.

References

Blind M, Gijbbers P, Gregersen J, Westen S, Vurro M and Gavardinas C (2004) HarmonIT Document Series Part B – Guidelines, Version 0.25.

Burgers PJ, Leewen V and Evensen G (1997) On the analysis scheme in the ensemble Kalman filter, *Monthly Weather Review*, Submitted Dec 1996, Revised Apr 1997.

Butts MB, Payne JT, Kristensen M and Madsen H (2004) An evaluation of the impact of model structure on hydrological modelling uncertainty for streamflow simulation, Submitted to The Distributed Model Intercomparison Project (DMIP), *Journal of Hydrology DMIP Special Issue*.

Canizares R (1999) On the application of data assimilation in regional coastal models, PhD thesis, IHE Delft, The Netherlands.

Evensen G (1994) Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast the error statistics, *Journal of Geophysical Research*, 99-C5, 10, 143-10, 162.

Evensen G (2003) Ensemble Kalman Filter: Theoretical formulation and practical implementation, *Ocean Dynamics* Vol 53: 323-367.

Ghil M and Malanotte-Rizzoli (1991) Data assimilation in metrology and oceanography, *Advanced in Geophysics*, 33:141-266: In Canizares R (1999) On the application of data assimilation in regional coastal models, PhD thesis, IHE Delft, The Netherlands.

Gijbbers P (2004) The HarmonIT document series Part C - the org.OpenMI.Standard interface specification, Draft Report Vol 0.91.

Gijbbers P and Gregersen J (2004) The OpenMI Standard in a nutshell, Vol. 0.99.

<http://www.harmonit.org>

Julier, Simon and Uhlman J (1996) A General Method of Approximating Nonlinear Transformations of Probability Distributions, Robotics Research Group, Department of Engineering Science, University of Oxford. In: Welch G and Bishop G (2004a) An Introduction to the Kalman Filter, UNC-Chapel Hill, TR-95-041.

Koster T (2004) Input correction in rainfall-runoff models using Ensemble KalmanFiltering, TU Delft and WL| Delft Hydraulics.

- Miller RN, Ghil M and Gauthiez F (1994) Advanced data assimilation in strongly nonlinear dynamical systems, *Journal of Atmospheric Sci.*, 51, 1037-1056. In: Reichle RH, Mclaughlin DB, and Entekhabi D (2002), *Hydrologic Data Assimilation with the Ensemble Kalman Filter*, *Monthly Weather Review*, Volume 130, 103-114
- Moore R, Tindall and Fortune D (2004) Update on the HamonIT project: The OpenMI standard for model linking, *Proceedings of the 6th International conference on hydroinformatics*, World scientific Publishing company.
- Reichle RH, Mclaughlin DB, and Entekhabi D (2002), *Hydrologic Data Assimilation with the Ensemble Kalman Filter*, *Monthly Weather Review*, Volume 130, 103-114.
- Sinding, Gregresen J and Gijbers P (2004) *The HarmonIT document series Part F – Org.OpenMI.Utilities technical documentation.*
- Vrugt JA, Gupta HV, Bouten W and Sorooshian S (2003) A Shuffled Complex Evolution Metropolis algorithm for optimization and uncertainty assessment of hydrologic model parameters, *Water Resources Res.*, 39(8), 1201, doi:10.1029/2002WR001642.
- Wagener T, Boyel DP, Lees MJ, Wheater HS, Gupta HV and Sorooshian S (2001) A framework for development and application of hydrological models, *Hydrology and Earth System Sciences* 5(1), 13-26.
- Welch G and Bishop G (2004), *Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast the error statistics*, *Journal of Geophysical Research*, 99-C5, 10, 143-10, 162.
- Welch G and Bishop G (2004a) *An Introduction to the Kalman Filter*, UNC-Chapel Hill, TR-95-041.
- Westen S, Fortune D and Gregersen J (2004) *OpenMI – New opportunity for model developers*. *Proceedings of the 6th International conference on hydroinformatics*, World scientific Publishing Company

Appendix A Data definitions in the OpenMI interface specification

data definitions

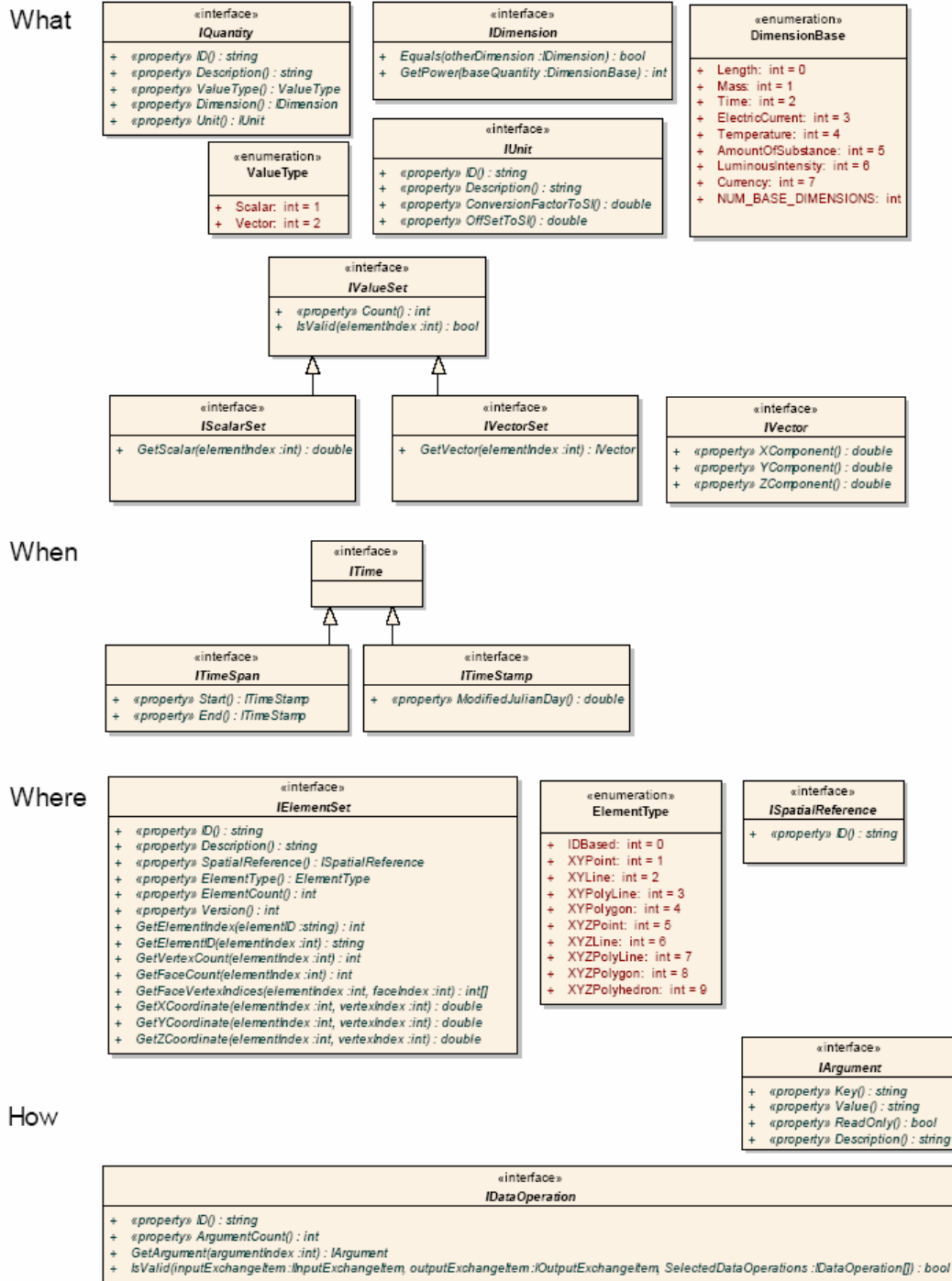
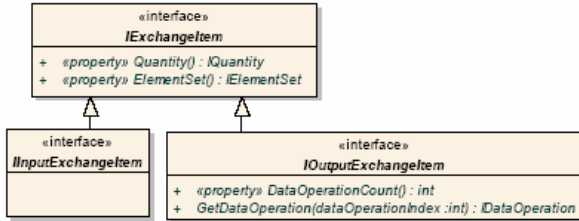


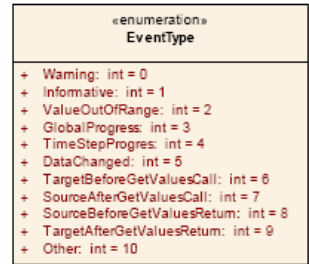
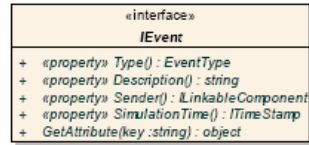
Figure 1 Data definitions in the OpenMI interface specification
 [Source: A4-leaflet of org.OpenMI.Standard v.0.99 by Gijbers, P.]

Appendix B Other interfaces of the OpenMI

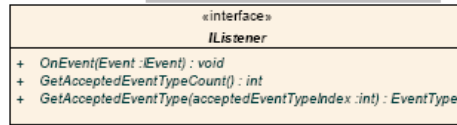
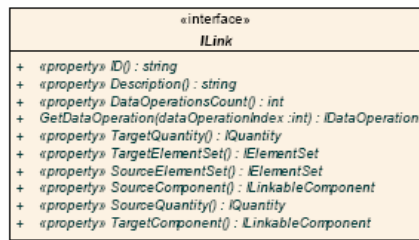
meta data to express what can be exchanged



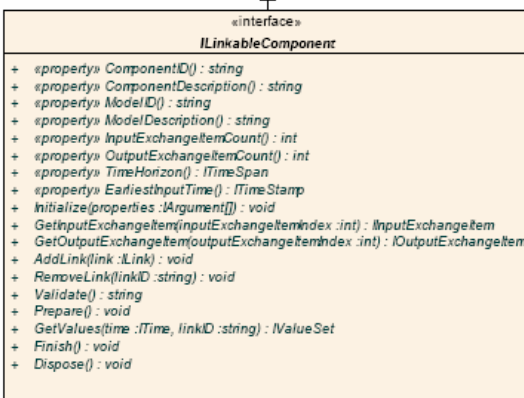
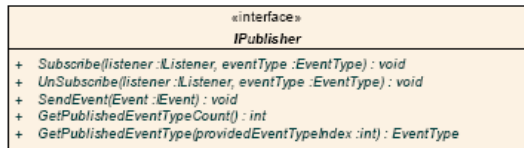
message definition



specification what will be exchanged and how



component interfaces for generic component access



advanced component interface extensions (optional)

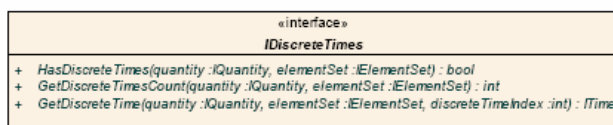
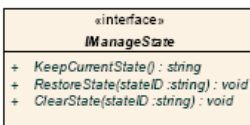


Figure 2 Other interfaces of the OpenMI
 [Source: A4-leaflet of org.OpenMI.Standard v.0.99 by Gijbers, P.]

Appendix C **org.OpenMI.Standard API-specification**

[Source: adopted from (Gijsbers, 2004)]

org.OpenMI.Standard API-specification

org.OpenMI.Standard.IArgument

Argument Interface, key-value pairs, applied as arguments for a data operation or arguments to populate components at instantiation with (model specific) data

Key property {get}

Argument identification ('Key' in: Key=Value pair)

Value property {get}

Argument value ('Value' in: Key=Value pair)

Description property {get}

Description of the argument

ReadOnly property {get}

Boolean determines if argument value can be modified by the user

org.OpenMI.Standard.DimensionBase

Enumeration for base dimensions

Length field

Dimension base length

Mass field

Dimension base mass

time field

Dimension base time

ElectricCurrent field

Dimension base electric current

Temperature field

Dimension base temperature

AmountOfSubstance field

Dimension base amount of substance

LuminousIntensity field

Dimension base luminous intensity

Currency field

Dimension base currency

NUM_BASE_DIMENSIONS field

Total number of base dimensions involved

org.OpenMI.Standard.IDataOperation

DataOperation Interface, identifies data operation and contains associated argument values

GetArgument(System.Int32).method

Get n-th argument, key

Returns

argument object

Initialize(org.OpenMI.Standard.IArgument[]).method

Initialize (called to populate data operation with specific – i.e. selected - information)

Arguments

properties

Array of argument objects to be used for initialization

IsValid(org.OpenMI.Standard.IInputExchangeItem, org.OpenMI.Standard.IOutputExchangeItem, org.OpenMI.Standard.IDataOperation[]) method

Indicates if the combination of data operations is valid for the selected input/output combination

Arguments

inputExchangeItem

The inputExchangeItem as selected in a GUI (i.e. the content will be at the target side of the link)

outputExchangeItem

The outputExchangeItem as selected in a GUI (i.e. the content will be at the source side of the link)

SelectedDataOperations

The array of already selected data operations (excluding the current one)

Returns

Boolean

ID property {get}

Identification string

ArgumentCount property {get}

Number of arguments for this data operation

org.OpenMI.Standard.IDimension

Dimension interface, describes the dimension, expressed in base dimensions, of a Quantity

GetPower(org.OpenMI.Standard.DimensionBase) method

Get power for the selected base dimension

Arguments

baseDimension

baseDimension instance to compare with

Returns

integer, power of selected base dimension

Equals(org.OpenMI.Standard.IDimension) method

Check if a Dimension instance equals to another Dimension instance.

Arguments

otherDimension

Dimension instance to compare with

Returns

Boolean, Dimension instances are equal

org.OpenMI.Standard.IDiscreteTimes

DiscreteTimes Interface, to obtain discrete time information associated to the quantity-element set combination of a linkable component

HasDiscreteTimes(org.OpenMI.Standard.IQuantity,org.OpenMI.Standard.IElementSet) method

Is Quantity/Elementset defined on discrete time steps?

GetDiscreteTimesCount(org.OpenMI.Standard.IQuantity,org.OpenMI.Standard.IElementSet) method

Get number of discrete time steps for Quantity/Elementset

GetDiscreteTime(org.OpenMI.Standard.IQuantity,org.OpenMI.Standard.IElementSet,System.Int32) method

Get n-th discrete time stamp or span for Quantity/Elementset

Arguments

quantity

The quantity

elementSet

The element

discreteTimeIndex

index of timeStep

Returns

Discrete time stamp or span

org.OpenMI.Standard.ElementType

Shape Type of an ElementSet

IDBased field

Identifier based

XYPoint field

Points in the (horizontal) XY-plane

XYLine field

Lines/Line segments in the (horizontal) XY-plane

XYPolyLine field

Polylines in the (horizontal) XY-plane

XYPolygon field

Polygons in the (horizontal) XY-plane

XYZPoint field

Points in the 3-dimensional space

XYZLine field

Lines/Line segments in the 3-dimensional space

XYZPolyLine field

Polylines in the 3-dimensional space

XYZPolygon field

Polygons in the 3-dimensional space

XYZPolyhedron field

Polyhedron (volume) in the 3-dimensional space

org.OpenMI.Standard.IElementSet

ElementSet Interface , rigid interface to query an element set for its content in terms of elements, optionally composed of vertices

GetElementIndex(System.String) method

Index of element 'ElementID' in the elementset

GetElementID(System.Int32) method

ID of 'ElementIndex'-th element in the elementset

GetVertexCount(System.Int32) method

Number of vertices for the element specified by:

Arguments

ElementIndex

element index in element set

Returns

Number of vertices in element with ElementIndex

GetXCoordinate(System.Int32, System.Int32) method

X-coord for the vertex with VertexIndex of the element with ElementIndex

Arguments

ElementIndex

element index

VertexIndex

vertex index in the element with index ElementIndex

GetYCoordinate(System.Int32, System.Int32) method

Y-coord for the vertex with VertexIndex of the element with ElementIndex

Arguments

ElementIndex

element index

VertexIndex

vertex index in the element with index ElementIndex

GetZCoordinate(System.Int32, System.Int32) method

Z-coord for the vertex with VertexIndex of the element with ElementIndex

Arguments

ElementIndex

element index

VertexIndex

vertex index in the element with index ElementIndex

ID property {get}

Identification string

Description property {get}

Additional descriptive information

SpatialReference property {get}

Spatial reference system for the element set

ElementType property {get}

Shape Type of the element set

ElementCount property {get}

Number of elements in set

org.OpenMI.Standard.EventType

Shape Type of an ElementSet, extensions for shape types in Z-direction need to be defined

Warning field

Warning event

Informative field

Informative event

ValueOutOfRange field

ValueOutOfRange event

GlobalProgress field

Indicates progress as percentage of global time horizon

TimeStepProgress field

Indicates progress as percentage of requested time step

DataChanged field

Indicates change of (computed) data in the component

TargetBeforeGetValuesCall field

Call by target component to inform source component on an upcoming request. Useful while debugging.

SourceAfterGetValuesCall field

Call by source component to indicate that request has been received

SourceBeforeGetValuesReturn field

Call by source component to indicate that is about to return the valueset

TargetAfterGetValuesReturn field

Call by target component to indicate that values have been received

org.OpenMI.Standard.IEvent

IEvent. Interface, generic meta-data structure to pass event information

GetAttribute(System.String) method

Get the value of a Key=Value pair, containing additional information on the event

Type property {get}

Type of event

Description property {get}

Additional descriptive information

Sender property {get}

LinkableComponent that generated the event

SimulationTime property {get}

Current SimulationTime ("- " if not applicable)

org.OpenMI.Standard.IExchangeItem

ExchangeItem Interface , holds Quantity-ElementSet combinations that can be exchanged

Quantity property {get}

Quantity

ElementSet property {get}

ElementSet

org.OpenMI.Standard.IInputExchangeItem

InputExchangeItem Interface, sub-class of IExchangeItem, holding quantity –element set combinations that act as input to a specific linkable component

org.OpenMI.Standard.ILink

Link Interface, holds the actual information of the link, including the components, quantity and element set on both the source and target side, as well as the selected data operations to be executed by the source component

GetDataOperation(System.Int32).method

Get n-th data operation item

Returns

Data operation

ID property {get}

Identification string

Description property {get}

Additional descriptive information

DataOperationsCount property {get}

Number of data operations

SourceComponent property {get}

Source linkable component

SourceQuantity property {get}

Source quantity

SourceElementSet property {get}

Source elementset

TargetComponent property {get}

Target linkable component

TargetQuantity property {get}

Target quantity

TargetElementSet property {get}

Target elementset

org.OpenMI.Standard.ILinkableComponent

LinkableComponent Interface, implements org.OpenMI.Standard.IPublisher. Interface for generic model access to the component and the data it can exchange

Initialize(org.OpenMI.Standard.IArgument[])method

Initialize (called to populate component with specific information)

Arguments

properties

array of IArguments to be used for initialization

GetInputExchangeItem(System.Int32) method

Get n-th input exchange item

Returns

Input exchange item

GetOutputExchangeItem(System.Int32) method

Get n-th output exchange item

Returns

Output exchange item

AddLink(org.OpenMI.Standard.ILink) method

Add an Input or Output Link (Called when initialize has been called for all components)

Arguments

link

Link to be added

RemoveLink(System.String) method

Remove a link

Arguments

LinkID

Link to be added

Validate method

Validation of the component status and its links

Returns

Returns an empty string if the component is valid otherwise returns a message string

Prepare method

Prepare for computation (Called just before computation starts, called when all links have been added)

GetValues(org.OpenMI.Standard.ITime,System.String) method

Get Values for a certain TimeStamp or TimeSpan, for a certain Link (= Quant./Elm.set)

Arguments

time

timestamp or timespan

LinkID

involved link

Returns

ValueSet (scalar or vector)

Finish method

Finish (Called when computation is done, close files, network connection, de-allocate memory where easible)

Dispose method

Dispose (i.e. cleanup; called when deployment stops)

ComponentID property {get}

Identification string of the component/engine

ComponentDescription property {get}

Additional descriptive information of the component/engine

ModelID property {get}

Identification string of the site specific model/study area

ModelDescription property {get}

Additional descriptive information of the site specific model/study area

InputExchangeItemCount property {get}

Number of input exchange items

OutputExchangeItemCount property {get}

Number of output exchange items

TimeHorizon property {get}

time horizon (begin and end date for which data can be retrieved)

EarliestInputTime property {get}

Earliest needed input time (Queried by providing component when they want to clear their buffers)

org.OpenMI.Standard.IListener

Listener Interface, enables components to grab events

OnEvent(org.OpenMI.Standard.IEvent) method

Method called when event is raised

Arguments

Event

Event that has been raised

GetAcceptedEventTypeCount method

Get number of accepted event types

Returns

Number of accepted event types

GetAcceptedEventType(System.Int32) method

Get accepted event type with index acceptedEventTypeIndex

Arguments

acceptedEventTypeIndex

index in accepted event types

Returns

Accepted event type

org.OpenMI.Standard.IManageState

Manage State Interface, to preserve, restore and clear the internal state of a component. (To be implemented optionally, in addition to the linkable component interface.)

KeepCurrentState method

Store the linkable component's current State

Returns

State identifier

RestoreState(System.String) method

Restore the linkable component's current State

Arguments

stateID

State identifier

ClearState(System.String) method

Clears a state from the linkable component's memory

Arguments

stateID

org.OpenMI.Standard.IOutputExchangeItem

OutputExchangeItem Interface, holds the output combinations of a quantity-element set, including a description of data operations that can be offered on the associated value set

GetDataOperation (System.Int32) method

Get n-th data operation

Returns

Data operation description

DataOperationCount property {get}

Get number of data operations

org.OpenMI.Standard.IPublisher

Publisher Interface, provides meta data on available events, accommodates subscription to those events and publishes the events

Subscribe(org.OpenMI.Standard.IListener,System.String) method

Subscribes a listener

Arguments

Listener

The listener

EventType

The event type

UnSubscribe(org.OpenMI.Standard.IListener,System.String) method

Unsubscribes a listener

Arguments

Listener

The listener

EventType

The event type

SendEvent(org.OpenMI.Standard.IEvent) method

Sends an event to all subscribed listeners

Arguments

Event

The event

GetPublishedEventTypeCount method

Get number of published event types

Returns

Number of provided event types

GetPublishedEventType(System.Int32) method

Get provided event type with index providedEventTypeIndex

Arguments

providedEventTypeIndex

index in provided event types

Returns

Provided event type

org.OpenMI.Standard.IQuantity

Quantity Interface, describes the (physical) quantity that can be exchanged

ID property {get}

Identifier

Description property {get}

Additional descriptive information

ValueType property {get}

Quantity's value type (vector, scalar)

Dimension property {get}

Quantity's Dimension

Unit property {get}

Unit in which quantity is expressed

org.OpenMI.Standard.IScalarSet

ScalarSet Interface, holds an array of doubles for a certain quantity on a certain element set (ordering corresponds to elements in element set). Implements org.OpenMI.Standard.IValueSet interface

GetScalar(System.Int32) method

Value for one of the elements in the set

Arguments

ElementIndex

index in the scalar set

Returns

double scalar value

org.OpenMI.Standard.ISpatialReference

SpatialReference Interface, holds a string reference to (known) spatial reference systems

ID property {get}

Identifier indicating which spatial reference to use

org.OpenMI.Standard.ITime

Time Interface, 'Abstract' interface, base for TimeStamp and TimeSpan

org.OpenMI.Standard.ITimeSpan

TimeSpan Interface, describes a continuous period over time

Start property {get}

Time span's begin time stamp

End property {get}

Time span's begin time stamp

org.OpenMI.Standard.ITimeStamp

TimeStamp Interface, describes an instantaneous moment in time

ModifiedJulianDay property {get}

Get TimeStamp expressed as ModifiedJulianDateAndTime (JulianDateAndTime - 2400000.5) Number of days since 1858/11/17 12:00:00.00, and fraction of 24hr. See for example <http://aa.usno.navy.mil/data/docs/JulianDate.html>

org.OpenMI.Standard.IUnit

Unit Interface, describes the unit in which a quantity is expressed

ID property {get}

Identification string

Description property {get}

Additional descriptive information

ConversionFactorToSI property {get}

Conversion factor to SI ('A' in: $SI\text{-value} = A * \text{quant-value} + B$)

OffSetToSI property {get}

OffSet to SI ('B' in: $SI\text{-value} = A * \text{quant-value} + B$)

org.OpenMI.Standard.IValueSet

ValueSet Interface holds an array of doubles for a certain quantity on a certain element set (ordering corresponds to elements in element set). Base for VectorSet and ScalarSet

Count property {get}

Number of elements in the set

org.OpenMI.Standard.IVector

Vector Interface, containing the values of the X,Y and Z component of a vector

XComponent property {get}

Vector component in X-direction

YComponent property {get}

Vector component in Y-direction

ZComponent property {get}

Vector component in Z-direction

org.OpenMI.Standard.IVectorSet

VectorSet Interface, holds an array of vectors for a certain quantity on a certain element set. Implements org.OpenMI.Standard.IValueSet interface

GetVector(System.Int32) method

Vector for one of the elements in the set

Arguments

ElementIndex

index in the vector set

Returns

vector

org.OpenMI.Standard.ValueType

Value(Set)Type for Quantity

Scalar field

Scalar

Vector field

Vector

Appendix D **org.OpenMI.Utilities.Wrapper package**

[Source: adopted from (Sinding et al, 2004)]

org.OpenMI.Utilities.Wrapper.IEngineApiAccess:

The IEngineApiAccess is the interface the ModelEngine component must implement when used with the SmartWrapper.

Methods:

Create (Hashtable properties): void Public

The Create method will be invoked just after creation of the object that implements the IEngineApiAccess interface

Parameters:

- Properties : Hashtable with the same contents as the Component arguments in the ILinkableComponent interface. Typically any information needed for initialization of the model will be included in this table. This could be path and file names for input files.

Return values:

- none

Exceptions:

- none

Initialize(): Void Public

This method will be invoked after the Create method has been invoked and before any other methods in the IEngineApiAccess interface has been invoked. Typically, the model engine will be populated when the Initialize method is invoked. E.g. by reading input files.

Parameters:

- None

Return values:

- none

Exceptions:

- None

Finalize(): Void Public

This method will be invoked after all computations are completed.

Parameters:

- none

Return values:

- none

Exceptions:

- None

Dispose(): Void Public

This method will be invoked after all computations are completed and after the method Finalize has been invoked.

Parameters:

- None

Return values:

- none

Exceptions:

- None

PerformTimeStep(): bool Public

This method will make the model engine perform one time step.

Parameters:

- None

Return values:

- Returns true if the time step was completed, otherwise it will return false

Exceptions:

- None

GetCurrentTime(): ITimeStamp Public

Get the current time of the model engine

Parameters:

- None

Return values:

- The current time for the model engine

Exceptions:

- None

GetInputTime(): ITimeStamp..Public

Get the time for which the next input is needed for a specific Quantity and ElementSet combination

Parameters:

- quantityID (string):
- elementSetID (string)

Return values:

- Time for the next required values associated to a Quantiy and ElementSet combination

Exceptions:

- None

GetEarliestNeededTime(): ITime..Public

Get earliest needed time, which can be used to clear the buffer. For most time stepping model engines this time will be the time for the previous time step.

Parameters:

- None
-

Return values:

- Time

Exceptions:

- None

GetValues(): IValuesSet..Public

Get values from the model engine.

Parameters:

- String: QuantityID associated to the requested values
- String: ElementSetID associated to the requested values

Return values:

- The values

Exceptions:

- None

SetValues(): Void Public

Set values in the model engine.

Parameters:

- String: QuantityID associated to the values
- String: ElementSetID associated to the values
- IValuesSet: The values to set

Return values:

- Void

Exceptions:

- None

Property: ID

IDstring for the model engine.

Property: Description

Description string for the model engine.

Appendix E **The org.OpenMI.Utilities.AdvancedControl package**

[Source: adopted from (Sinding et al, 2004)]

1 General description

1.1 Advanced control functionality why and when

The data exchange and synchronization mechanism of OpenMI is designed in such way that LinkableComponents can autonomously exchange data without any centralized functionality to manage the data exchange. However, in some specific situations, extra control capacity is needed to direct convergence of computational results. This functionality typically is desired for iteration purposes, as well as for optimization and calibration.

A separate utility package has been designed to support these advanced control features. The package, named org.OpenMI.Utilities.AdvancedControl, utilizes the GetValues mechanism of the LinkableComponent, as well as the state management functionality (implemented through the IManageState interface) to direct the convergence of computational results. The controllers themselves are LinkableComponents as well, so their data (i.e. the new parameter values or boundary conditions) can be accessed by a model based LinkableComponent as well.

The following control functionality has been identified:

- The iteration controller is needed when implicit models are linked bi-directionally and iterations within time-steps are needed
- The calibration controller component is responsible for calibrating one model or a set of linked simulation engines. The optimisation controller can be used to find values of variables that minimize or maximize an objective function while satisfying a set of constraints. In the OpenMI framework this will normally be used in conjunction with a set of linked models.
- The logical switch allows switching inputs depending on an input condition

1.2 Iteration controller

When multiple implicit model engines are linked bi-directionally, iterations within time-steps may be required to obtain correct numerical values. An implicit model engine requires the input value on the next time-step in order to calculate the output value on the next time-step. An explicit model engine requires only the input on the current time-step to calculate the output on the next time-step. When simulation engines are linked

bi-directionally and no iterations within time-steps are used, the link is effectively explicit. This may mean that the simulation engines have to use a much smaller time-step.

An alternative may be to use iterations within time-steps. The simulation engines have to support saving and restoring of the engine state when the iteration controller is used. When the iteration controller is used, the models do not exchange values directly but through the iteration controller (see Figure 14). The iteration controller is in control of the iterations and requests the models to step back one time-step. The iteration controller decides when the iterations have converged. The iteration controller throws an exception when there is no convergence in the iteration.

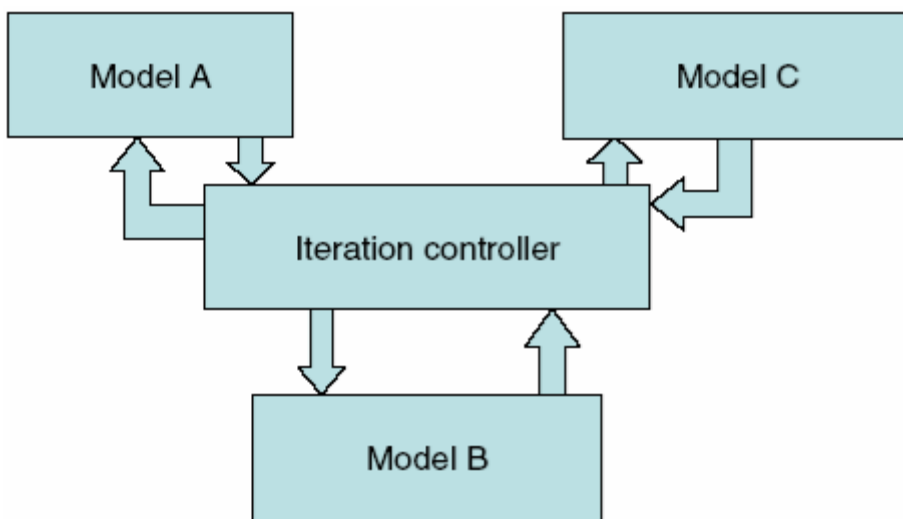


Figure A-1 Position of iteration controller within a model combination

The requirements of an IterationController are;

- All bi-directional links are linked through the iteration controller and not directly. This is necessary because the iteration controller is in control of iterations. It allows the iteration controller to check that the values are converging.
- The iteration controller decides when to stop iterating, based on a stopping criterion.
- The iteration controller has a configurable number of minimum and maximum iterations. A flag is used to indicate if the iteration controller throws an exception when the maximum number of iterations is exceeded.
- The iteration controller is responsible for managing the simulation engine states during the iterations.
- The iteration controller is itself linkable component and it is triggered externally. From the outside the iteration controller and the linked simulation engines look like a LinkableComponent.

- The iteration controller does not have a Graphical User Interface (GUI), although a simple example GUI would be useful for testing. The GUI and the iteration controller should be completely separate components.

1.3 Calibration controller

Although models can be calibrated individually, when models are linked together into a linked model this linked model has to be calibrated as well. The calibration controller is a generic component to calibrate linked models. It will provide a generic interface so that specific calibration algorithms can be used. In most cases the objective of calibration is to minimise an error function. The error function is usually defined as the difference between measured data and calculated data. The purpose of the calibration is to find the set of parameter values that minimise the error function. Calibration can be seen as an optimisation problem because the error function is optimised and the parameter values are the unknowns in the optimisation. Figure A-2 illustrates the data flow around a calibration controller.

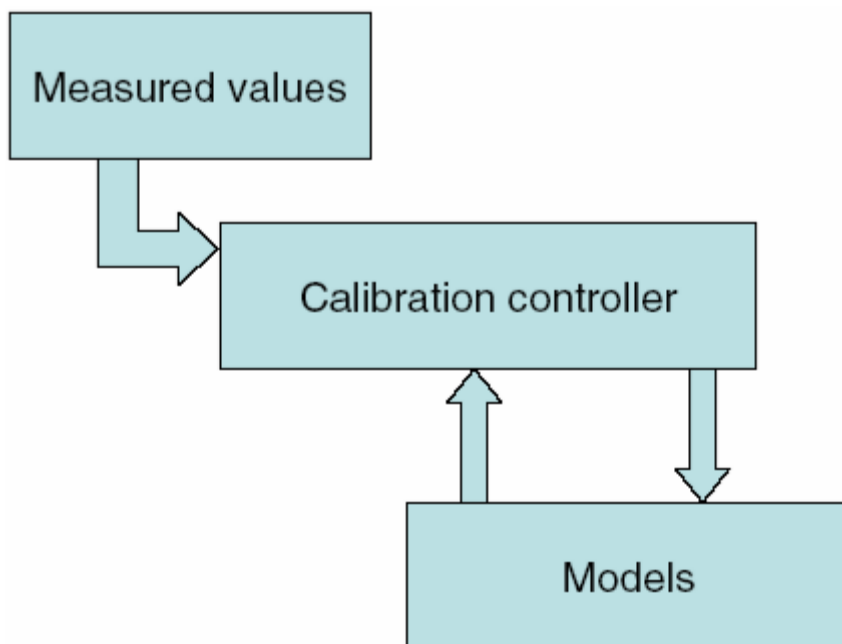


Figure A-2 Position of calibration controller

The requirements of a CalibrationController are:

- Manual and automatic calibration should be possible. With the manual calibration, the user should be able to change parameter values and run the simulation. With automatic calibration, the user should be able to choose the parameters that are going to be calibrated.
- The calibration component should be able to compare computed with measured values and calculate statistics for the calibration.
- A mechanism is needed to specify the following:

- Model parameters that can be changed. The simulation engines will perform a `GetValues()` call to the calibration controller to retrieve the model parameters.
- Minimum and maximum value for each parameter.
- Initial value for each parameter.
- A mechanism is needed to change the parameter values after each iteration.
- A mechanism is needed to decide when to stop calibrating.
- A mechanism to make the calibrated values “fixed”, e.g. to write them out to the input file. Once values are calibrated, the parameters are not inputs any longer. It should be possible to make calibrated values persistent.
- The calibration controller will send events about calibration progress and statistics

1.4 Optimization controller

The main difference between optimization and calibration is that in calibration usually a large number of values is calibrated and in optimization usually a small number of values is calibrated. Optimization can also be to make a choice between different scenarios instead of finding the best value of a parameter. Optimization may also be the choice between different control strategies or to optimize a control strategy.

Optimization problems are made up of three basic ingredients:

- An objective function that we want to minimize or maximize. The objective function can be linear or non-linear.
- A set of unknowns or variables which affect the value of the objective function. The variables can be continuous or discrete or mixed.
- A set of constraints that allow the unknowns to take on certain values but exclude others. The constraints can be linear or non-linear.

The optimization controller is a generic component for finding the values of the variables that minimize or maximize an objective function while satisfying a set of constraints. The optimization controller will have a generic interface so that any optimization algorithm can be linked to it.

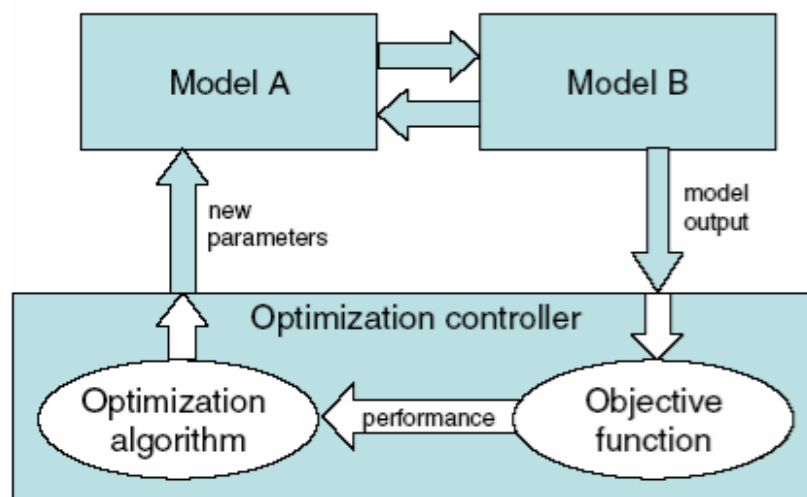


Figure A-3 Position of the optimization controller (Gray is OpenMI compliant)

The requirements that have been kept in mind for the OptimizationController are:

- Specify an objective function. The objective function could be a model in itself, with one output value. This would allow for good flexibility in the system because it is easy to change the objective function by just plugging in another one.
- Specify if the objective function should be minimized or maximized. This could be set during the initialization of the optimization controller.
- Specify the variables to be optimised. This could be done by linking variables in models to the optimisation controller.
- Specify constraints
 - Specify a minimum and maximum for variables
 - Specify linear constraints
 - Specify non-linear constraints
- Specify a starting value for variables.
- The optimization controller will raise events about optimization progress.
- The optimization controller will raise an error event when no solution can be found that satisfies all constraints.
- Manual and automatic optimization should be possible. For manual optimization, it should be possible to adjust values.
- The optimization algorithm (linear programming, dynamic programming, conjugate gradient, etc.) should be user definable and should plug in to the optimization controller so that it is easy to plug in different algorithms.

1.5 Logical switch

The logical switch is used to switch between different inputs depending on a logical condition. An example of a logical condition is a threshold for a water level or a threshold for a flow.