Model and Data Integration and Re-use in Environmental Decision Support Systems

Technical report IDSIA 38-98

Andrea E. Rizzoli [1]
IDSIA - Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
Lugano - Switzerland

J.Richard Davis
CSIRO Land and Water
Canberra - Australia

David J. Abel
CSIRO Mathematical & Information Sciences
Canberra - Australia

20 May 1998

Abstract

This paper presents a software architecture for the management of environmental models. The Systems Theory representation of models is embedded in an object-oriented approach that emphasises the separation of models from data, thereby promoting model and data integration and re-use. The concepts presented here correspond to the requirements of a Model Management System (MMS). It is finally shown how a Decision Support System can use this approach to implement the MMS in order to facilitate problem definition (via the domain base) and problem solution (via the model base).
Keywords: Model management, model integration and re-use, environmental decision support systems.

0.1 Summary

A very large number of water-related models have been developed over the years, covering such topics as rainfall-runoff, in-stream water quality, flood prediction, ground-water accession, etc. There has been an increasing demand from managers for models that can handle more complex environmental problems. It is difficult to simply link the existing models together into integrated models for these problems because the models were never originally designed to standards that allowed such linking. Some difficulties are: intertwining of user interfaces with each of the models; lack of modular construction; differing computer languages; poor documentation of conceptual limitations and assumptions; and lack of standardization of input and output structures.

The advent of object-oriented approaches in computer science has made the systematic, modular development of new models more feasible and this technology is being increasingly adopted by modellers when writing new models. However, standards need to be set by the modelling community to ensure maximum re-usability of component models. The following sections provide a basis for an object-oriented approach using the concept of encapsulation of component models.

Existing models pose a further problem. There has been such an investment in these legacy models that it is impractical to discard them or rewrite in object-oriented form. There are various approaches to legacy models although none is as efficient as using properly designed object-oriented models. The HYDRA project [13] and the TwoLe decision support system [28] are explorations into such an approach.

0.2 Introduction

Environmental problems lie at the cross-roads of multiple disciplines, and for this reason are often described by a set of interacting models. For instance, in a model of lake eutrophication, one model can describe the limnological processes and another one the ecological processes. The latter can also be decomposed into sub-models such as algal uptake of nutrients and food-web processes.

These integrated models need to be embedded in a Decision Support Systems (DSS) to help managers to assess environmental impacts of various policies and decisions. A standard DSS is composed of at least three modules: a data base management system (DBMS), a model base management system (MMS) and a dialogue generation and management software (DGMS) [33].

Among the necessary characteristics of the MMS module are [26]:

1. The MMS should be able to create new models quickly and easily.

2. The model building-blocks of the MMS should contain cognitively meaningful chunks of knowledge to the user.

3. The MMS should be able to inter-relate models with appropriate linkages, thus providing the functions of model integration, model decomposition, sequential model processing, and concurrent model processing.

4. The MMS should be able to manage the model base with functions analogous to data base management. The MMS should be able to decompose a query into a sequence of data retrievals and model invocations and retrievals.

5. The MMS must have a meta-level encyclopedia, analogous to a DBMS's data dictionary, which includes a repository of data, heuristics, tasks, models, users, and the relationships between them. The following characteristic can be added in order to include the legacy models now in widespread use.

6. The MMS must be able to incorporate executable models written by other modellers and to connect them to other models in a seamless way.

Proposals to develop MMSs have arisen in Management Science and Operation Research [7], [8], [18], [26], Artificial Intelligence [11] and System Theory [15], [25], [30], [34], [35]. Some authors in particular have dealt with issues that have some common ground with the ideas exposed in this work. For example, Bhargava and Kimbrough [3] discuss the embedded language technique. They remark that model languages, from natural language to mathematical formulae, are the common tools used by modellers to express their ideas, but they are not flexible enough to contain information about the modelling domain, such as information sources, logical and causal relationships among model entities, data descriptions, etc. They propose an embedding language that incorporates this extra information.

Hong, Mannino and Greenberg [18] pursue an approach based on measurement theory to represent the mapping from the domain world to the model base. The domain world is composed of an individual-level and of a class-level. The aim is to enhance model re-usability and integration via a general description of the model mechanism, independent of the specific application.

Another notable example of an approach to model management can be found in SYMMS, a model management system implemented on a UNIX workstation [22]. SYMMS uses a model description language that allows the user to define general-type modules and atomic-version modules, instances of the general-type ones.

The previously cited works and experience have originated mainly from the management sciences field, where the focus is on solving and integrating mathematical programming problems. Our work is aimed at presenting the architecture of a MMS well suited to solve environmental problems. This class of problems is characterised by strong spatial and temporal characteristics to the point where the MMS design needs to include features that assist in the linking of spatial and temporal models. Moreover, environmental modellers are often interested in comparing different model formulations to solve the same problem, given the uncertainty present not only in data and parameters, but also in the model structures; for this reason, the MSS design incorporates this requirement.

All these authors are agreed on the use of object technologies (OT) [17] to answer the above requirements. Object technologies include object-oriented analysis, design, and programming paradigms as tools to develop software projects. Object-orientation is mainly based on three concepts: abstraction, encapsulation, and hierarchy. Paraphrasing Booch's definitions [5], abstraction has the purpose of denoting the essential characteristics of an object, providing crisply defined conceptual boundaries; encapsulation separates and puts in compartments the elements of an abstraction and separates the interface of an abstraction from its implementation; and finally hierarchy is a ranking or ordering of abstractions.

In the following, we explain how the model representation technique presented in this paper promote model and data re-use with particular attention to environmental models.

0.3 A MMS for environmental models

The vast majority of models describing environmental systems and natural resources are dynamic models, since modellers are mostly interested in the change of systems over time in response to external actions. For instance, global warming models try to infer a relationship between the level of emissions and the change in temperature in different parts of the globe, over a period of several years.

Traditional modelling has often relied on the well assessed formalism proposed by Systems Theory where inputs, states, outputs, and parameters represent the data. The state transition function and the output transformation are the operations that can be performed on these data. Inputs and outputs can be connected to create the more complex models required in environmental modelling.

In this work, the design of a MSS based on Object Technology is proposed using the same mathematical formalism of Systems Theory; these two theoretical foundations are applied in order to enhance and facilitate some essential functions like model prototyping, data access, and model integration. The reason for such a MMS is due to the increasing complexity of environmental models, which are often made of several multi-disciplinary components, involving fields such as hydrology, ecology, control engineering, etc. The modeller is faced with the difficult task of integrating technologies, and to do this, has to integrate the sub-models which build the big picture. Moreover, the body of knowledge in the environmental sciences, and consequently the number of models, is constantly increasing. Model bases, model directories have been created to be able to tap into this knowledge, and software technologies are at work to made these resources available (see, for instance, [16] and [19]).

Another issue in building DSS to support solutions of real-world problems is the time and cost to assemble the modelling systems and data resources and then to integrate these into information systems usable by planners or managers. Not unusually, this means that DSSs are completed only after the problem has effectively been solved by other means.

In the remainder of this paper two main entities are introduced: the *domain base* and the *model base*. These entities are introduced in order to separate the description of the data items (the domain) from the description of how some of these data are combined to infer expected behaviours (the models).

0.4 The domain base

The deductive modelling process typically stems from a deep study of the real world. This leads to the identification of some classes of basic entities to be included in the model. Then, the modeller starts to hypothesise about the kind of relations existing among these entities and then formalises the relationships, using mathematical tools. Often the modeller arrives at a satisfactory description of a phenomenon only after a lot of exploration with the structure of the model. A tool that would allow the user to substitute a model for another, to re-use a model and to easily manage data links would speed up this process.

While most modelling software environments separate models from data, their focus is on models which can be fed with alternative data sets. Parameter calibration is the main problem. The approach of the MMS presented in this paper is reversed, data sets can be transformed via models which are applied to them. This approach allows the modeller not only to perform traditional parameter calibrations, but also examine alternative model structures, comparing how different models behave against the same data sets and evaluating the relative performance indicators. It is therefore required that models be written with respect to the available data in order to be able to substitute one model for another, automating data formatting and conversions and enforcing consistency of data usage in models. The implementation of these requirements leads to the possibility of effective re-use of models and data.

In practice if one wants to substitute one sub-model for another in a complex inter-connected model, then it would be necessary to re-establish all the interface connections between the new sub-model and the surrounding ones and to redefine the data base access path for each parameter and input, state, and output variable. This approach is used in various MMS, both academic and commercial. However, re-usable models must define which kind of data they use and produce, and in which format, but not state the explicit data base access paths. A mechanism which implements this different approach is explained in the following sections, but first the concepts of domain classes, domain objects and models are introduced.

0.4.1 Domain classes and objects

Domain objects are instances of general structured data types (*domain classes*). Domain classes are designed to achieve generality and re-usability: models are written mapping their data sets to the data attributes of domain classes. When a model is applied to an instance of a problem, it uses the domain class attribute names to access the corresponding attributes in the domain objects, which in turn point to data sources. Note that different models can refer to the same domain class. Thus, domain classes play the role of data types and therefore define generalisations of modelled objects that have some common features. Usually, a domain class provides a general description of an entity; e.g., the class of flat watersheds. A domain object is an instance, a particular case.

Domain classes are based on the concept of abstract data types and are inspired by a mix of the concept of object-oriented classes [5], frames [21], [32], database schemas [24], and prototype systems [20]. The analogue with data types (and therefore with classes and database schemas) provides generalisation, while similarities with prototype systems maintain the possibility of expanding the problem definition by incrementally modifying existing domain class definitions.

The union of all the domain classes defines the structure of the *domain base*. Domain classes can be decomposed into sub-classes. Sub-classes can be organised in a hierarchy, using classification and encapsulation principles.

The classification relationship, used to structure the domain base, is based on the concept of inheritance. Domain classes can be classified according to their nature (*is-a* relationships). For instance, the class of steep watersheds and the class of flat watersheds derive their basic properties from a more general watershed class, but the flat watershed class also has some characteristic data attributes, such as the flood plain which is not an attribute of steep watersheds.

Encapsulation allows different description levels and provides the capability of describing big components as the sum of smaller ones (*part-of* relationships). For example a watershed can be composed of a number of sub-watersheds. This leads towards an object-oriented implementation of the data structures which is analogous to a non-normal form of a database table. Information is thus kept together in self-standing information pools.

The domain base structure depends on the particular solution strategy that the modeller has in mind. It is therefore a good practice to try to be as general as possible in designing the domain classes. It will be possible to specialise the descriptions later in the modelling process.

0.4.2 The structure of domain classes

A domain class is an abstract data type which is characterised by its set of data attributes. The data attributes of the domain class provide a description of the data and will be filled with actual data values when the instance of a domain object is created.

Data attributes are classified according to their role: input, output and local. Domain objects may communicate and exchange information. This flow of information will take place through the interface of the domain class defined by the input and output data attributes. The local data attributes are hidden to external domain classes. For example, a domain class which describes a watershed might have as input data attribute the rainfall in some gauging stations, as output attribute the measured watershed flow, and as local data the soil types over the watershed area.

For the purpose of classification and encapsulation, *basic* and *compound* domain classes can be defined. A compound domain class can have subparts, while a basic one is the elemental building block. A basic domain class $BDClass$ can be then defined by the following data structure:

$$BDClass = < BDClass_b^{(1)}, \ldots, BDClass_b^{(n)}, \mathbf{i_b}, \mathbf{l_b}, \mathbf{o_b} > \tag{0.1}$$

Where $BDClass_b^{(i)}$ is the superclass (or a set of superclasses) of the basic class $BDClass$ and, if present, it must be another basic class. The basic class inherits the data attributes from the superclass. $\mathbf{i_b}$, $\mathbf{l_b}$, $\mathbf{o_b}$ are the sets of input, local and output data attributes defined as:

$$\mathbf{i_b} = < (DBData, i_1), \ldots, (DBData, i_m) > \tag{0.2}$$

$$\mathbf{l_b} = < (DBData, l_1), \ldots, (DBData, l_p) > \tag{0.3}$$

$$\mathbf{o_b} = < (DBData, o_1), \ldots, (DBData, o_q) > \tag{0.4}$$

$DBData$ (Domain Base Data item) is the abstract data type used to catalog and store data attributes. Among its fields we find:

- name, a unique identifier for the data;

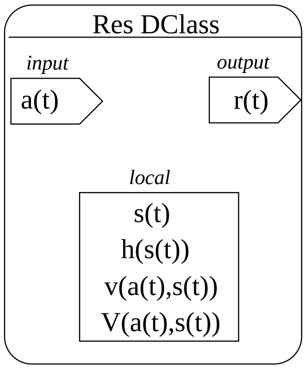

Figure 0.1: A basic domain class. The reservoir domain class defines the data types used to describe a reservoir.

- description, a general textual documentation of the meaning of the data item;
- dimension, the physical units used to measure the data;
- format, the data type (e.g. n by m matrix of reals, etc.);
- value, a function returning the actual value of the data attribute or a reference to another data attribute in another domain object (the usage of this field as a reference is explained later when dealing with compound domain classes).

Figure 0.1 shows an example of a basic domain class representing a reservoir. The input data attribute set $\mathbf{i_b}$ consists of only the inflow of water $a(t)$. The output set $\mathbf{o_b}$ by the reservoir water release $r(t)$. The local data attribute set $\mathbf{l_b}$ includes the reservoir storage $s(t)$, the elevation function $h(s(t))$ which converts the volume of water to an elevation with respect to a reference value, and the minimum and maximum storage-discharge functions $(v(a(t), s(t)), V(a(t), s(t)))$ which define the shape of the reservoir discharge [23].

This example will be used throughout the paper, in order to show how this MMS can be applied to the case of the design of a DSS for water management, from data specification to the solution of the optimisation problem.

A compound domain class is defined by the tuple $CDClass$:

$$CDClass = < \mathbf{superclasses}, \mathbf{components}, \mathbf{i_c}, \mathbf{l_c}, \mathbf{o_c}, \mathrm{M} > \tag{0.5}$$

Where:

$$\textbf{superclasses} =< DClass_s^1, \ldots, DClass_s^n > \tag{0.6}$$

$$\textbf{components} =< (DClass_c^1, compo_1), \ldots, (DClass_c^m, compo_m) > \tag{0.7}$$

The **superclasses** set is defined as the set of superclasses from which the current domain class inherits. It is composed of both basic and compound domain classes. While a basic domain class must inherit only from basic superclasses, a compound domain class can be derived from other compound domain classes.

A compound domain class is composed of a set of composing classes which are contained in the **components** set and represents the parts that make up the compound domain class. Again, these classes can be either basic or compound and therefore defined according to Eqns. 0.1 or 0.5. Note that a name is assigned to identify each composing class ($compo_1$, $\ldots$,$compo_n$), in the same way that the data elements of a complex data type are named in a programming language (e.g., type `complex` is a record composed of a real part `Re` which is of type `double` and of an imaginary part `Im` of type `double`).

In a compound domain class the data attributes of the composing classes are hidden because they are nested (encapsulated) in these classes. The basic domain classes which make up the compound class are linked through the mapping defined by M which connects the inputs of the compound class to the inputs of the basic classes, the output of the basic classes to the outputs of the compound class and the outputs of the basic classes to inputs of the basic classes:

$$\mathrm{M} = \begin{cases} \mathbf{i_c} \rightarrow \mathbf{i_b} \\ \mathbf{o_b} \rightarrow \mathbf{i_b} \\ \mathbf{o_b} \rightarrow \mathbf{o_c} \end{cases} \tag{0.8}$$

Figure 0.2 shows the layout of a compound domain class describing a very simple water system which includes a number of subclasses (one catchment, one reservoir and one water consumer). The composing classes are interconnected via a mapping: the reservoir receives the water inflow from the upstream catchment, and so on. This is all the information needed to build a compound domain class.

0.4.3 Basic and compound domain objects

A domain object is an instance of a basic domain class. The reservoir class seen in the previous example can be used to generate a reservoir domain object when the modeller defines the shapes of the storage-discharge functions

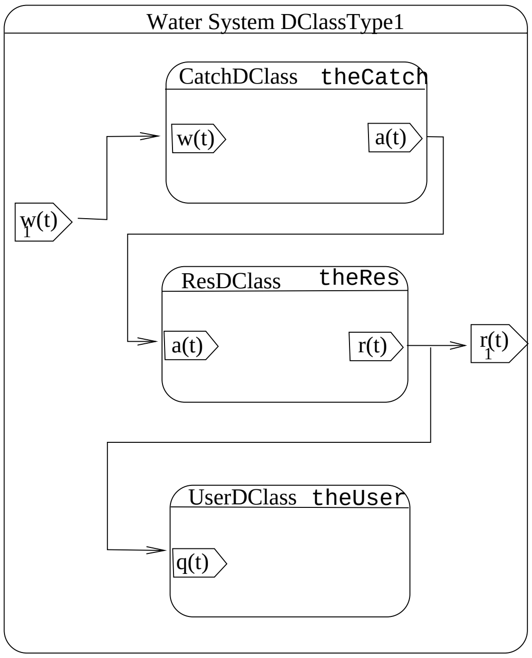

Figure 0.2: A compound domain class. The spatial organisation of basic domain classes tries to model the spatial relationships observed in the real world data.

and also specifies where to load and store the time series data associated with the water inflow, storage and release.

A basic domain object $BDObj$ can be then defined by the structure:

$$BDObj = < BDClass, \mathbf{D_b} > \tag{0.9}$$

where $BDClass$ is the originating basic domain class and $\mathrm{D_b}$ is the set of the values to be assigned to $\mathbf{i'_b}$, $\mathbf{l_b}$, and $\mathbf{o'_b}$.

Note that $\mathbf{i'}$ and $\mathbf{o'}$ are subsets of, respectively, $\mathbf{i}$ and $\mathbf{o}$, since not all input and output variables are read or stored in the data base, but they can be linked to data attributes in other domain objects, as shown in Figure 0.2.

In the same way, a compound domain object $CDObj$ is defined by:

$$CDObj = < CDClass, \mathbf{D_c} > \tag{0.10}$$

where $CDClass$ is the class used to generate the instance of a compound domain object and $\mathbf{D_c}$ is all the data needed to create the sub domain objects as defined in Eq. 0.5 and set the values of the data attributes $\mathbf{i'_c}$, $\mathbf{l_c}$ and $\mathbf{o_c}$.

Using the example compound domain class of Figure 0.2, a water system domain object can be created by looking in the domain base for some basic domain objects to associate with the basic model classes. For instance, the Maggiore water system would assign the Ticino Catchment domain object to the `theCatch` catchment domain class, Lake Maggiore to `theRes` and, finally, Ticino River Agricultural Users to `theUser`.

0.4.4 Data mappings, aggregations and transformations

Choosing where to put data attributes, whether into a basic domain class at the bottom level of an encapsulation or in a compound one at the top, is based on data visibility. Sometimes a decision cannot be made, since the decision depends on the kind of model which will use the data attributes and different models may wish to use the same data set at different representation levels. For instance, the modeller could be interested in writing an aggregate model for a very large watershed which makes use of the groundwater permeability coefficients over the whole area which are stored in a matrix. In another model, the same data could be used related to a subarea and a sub matrix should be extracted to represent those properties. *Data mappings* are introduced to overcome this kind of problems.

Data mappings occur when data attributes in sub-domain classes are mapped into data attributes in the compound domain class. This allows the modeller to operate on the whole set of data and not on single instances. Typical data "mappers" are vectors, matrices, lists. A data mapping preserves the dimension of the data organising it in a structure which has a greater cardinality than the single elements. A data mapping M is represented by:

$$\mathrm{M} : l_t \rightarrow L_T$$

Where l_t is a vector of the sub-domain class data attributes and L_T is a vector in the local section of the super-domain class that has the same structure. For instance, in two domain classes `Layer1` and `Layer2` (which represent two layers in a stratified lake) there is a data attribute representing the nutrient concentration N; in the domain class of type `Lake` it can be defined a vector containing those two concentrations as elements:

$$N_{Lake} = \begin{pmatrix} N_{Layer1} \\ N_{Layer2} \end{pmatrix}$$

Data aggregation is another operation which can be performed when some data attributes at the compound level are "intensive" representations of the extensive data attributes contained in the sub classes. For instance, the area of a catchment is the sum of the area of the sub-catchments; the temperature of a stratified lake can be considered equal to the weighted averages of the temperatures of the single layers into which it has been partitioned.

A data aggregation transforms a set of data of given cardinality into a representation with a lower cardinality (often into a single data item). A data aggregation is typically expressed by a function.

An example is given by:

$$L_T = f(l_t)$$

where

$$l_t = \begin{pmatrix} l_{st,1} \\ l_{st,2} \\ \vdots \\ l_{st,n} \end{pmatrix}$$

L_T represents the local data attribute in the compound domain class and l_t a vector which contains the data attributes of the composing domain classes. The cardinality of L_T is equal to the cardinalities of the single elements $l_{st,j}$. For instance, suppose that two domain classes $sc1$ and $sc2$ of type `subcatchment` have the data attribute `area`. A data aggregation `area` can be defined as a local attribute of the compound domain class c of type `catchment`, which is composed of the two sub-catchments, and the aggregation function is defined as:

$$A_c = a_{sc1} + a_{sc2}$$

Finally, *data transformations* provide low-level data modifications among domain objects which potentially have communication problems. These problems may arise because data attributes may have different spatial or temporal scales, units of measurement, etc. A data transformation is usually a simple static function (e.g. changing Celsius degrees to Fahrenheit), but it can also be considered a model on its own.

0.5 Models and the model base

The objectives of our MMS proposal are to make different models interoperate seamlessly (model integration) and be able to test alternative models against different working conditions (model re-use).

When we started our work on model integration and reuse (see [13] and [27]), techniques such as distributed computing on three-tiered architectures (client – broker – server) were being developed. During the early application of our design to environmental problems the HLA (High Level Architecture) approach of the DMSO [14] was described and, recently, a special issue of the journal Decision Support Systems [4] has summarized a number of approaches.

All of the above mentioned works pointed out the need of model encapsulation in order to provide consistent interfaces to model functionalities. In this section we describe our approach to encapsulation of environmental models.

A model class is an abstract data structure used to encapsulate the mathematical formulation of a given process to be modelled.

There are two kind of model classes: basic and compound. A basic model class has a flat structure in the sense that is not composed of any other model; a compound model consists of other models. For instance, a reservoir can be considered a compound model when it is seen as an ecosystem composed of fish, zoo-plankton, and phytoplankton; conversely, when it is used to describe a simple storage of water in a water management system, the ecological components are neglected, it can be characterised as a basic model.

A key concept in our system is the generality of model formulations: in order to improve model re-usability, model classes provide a way to write the model formulae in terms of data attributes of domain classes. Only when a model is assigned to an instance of a domain class (a domain object) is it linked to the data. Before that, it contains only the instructions on how to retrieve the data. This means that if n model classes can be associated with a given domain class, and if m instances of that domain class can be created (that is, m different domain objects), there will be a possible number of at least m times n different model instances (assuming for simplicity that only one model instance per model class is generated).

Potentially, many models can refer to a single domain class. Thus, a structure for retrieving and storing models, a model base, is needed. It has been shown by many authors [11], [30] that the access operations to a model base must be analogous to the access operations allowed on a data base. In particular a model management system must allow the user to find a model corresponding to a given set of selection criteria, to modify a selected model and to compose a new model, possibly assembled from existing ones. It is not the aim of this work to discuss the issue of model selection (see Falkenheimer and Forbus [10]), but to describe how this approach can be integrated in a model and solver selection tool for the solution of natural management problems.

Models must be re-usable with respect to domain classes. In a model, inputs, states, and outputs are variables, which can assume different values during a simulation run, while parameters are quantities that either are constant or have a limited range of variation. Inputs, states, outputs, and model parameters must find a correspondence in the data attributes of a domain class in the modelling domain. Thus, a model class is not *a priori* linked to a particular domain object but to a whole class of objects. If a new model class must use data attributes present in different domain classes, then a new domain class must be created, using the principles of aggregation or inheritance. Thus, input variables are mapped to input data, state variables and parameters to local data and output variables to export data in the domain class. Mapping a model variable to a domain class data attribute means to assign a context to that variable. Because, models can be expressed in very general terms, their formulations can be applied to different physical situations. Consider the simple example of the following model describing a decay phenomenon.

$$\frac{dy(t)}{dt} = -Cy(t)$$

The independent variable t represents time; y can be the piezometric head in a reservoir or the voltage over a condenser: the meaning depends on the domain class to which the model is assigned, that is, to the context.

0.5.1 Basic models

A basic model class $BMClass$ is defined by the following data structure:

$$BMClass =< DClass, \mathbf{u_b}, \mathbf{x_b}, \mathbf{y_b}, \Theta, \phi, \eta > \tag{0.11}$$

Where $DClass$ can be either a basic domain class or even a compound domain class, $\mathbf{u_b}$ is the set of model inputs, $\mathbf{x_b}$ the set of model states, $\mathbf{y_b}$ the set of model outputs, Θ is the set of model parameters. The state transition equation and the output transformation which describe the model in the Systems Theory approach are ϕ and η, respectively.

The input, state, output and parameter sets are defined as:

$$\mathbf{u_b} =< (MData, u_1), \ldots, (MData, u_m) > \tag{0.12}$$

$$\mathbf{x_b} =< (MData, x_1), \ldots, (MData, x_n) > \quad (0.13)$$

$$\mathbf{y_b} =< (MData, y_1), \ldots, (MData, y_p) > \quad (0.14)$$

$$\Theta =< (MData, \theta_1), \ldots, (MData, \theta_q) > \quad (0.15)$$

These sets constitute the model interface.

Model variables and parameters are represented by the data type $MData$ (Model Data item) with the following fields:

- *name*, a unique identifier for the symbol;
- *description*, a textual description of the meaning of the variable or parameter;
- *dimension*, the unit of measure;
- *format*, the data type, with its cardinality;
- *link*, the reference to the data source. This field is detailed in the following paragraph.

The *link* field is of great importance, since it specifies where models get their inputs, states and parameters and where they put their outputs. It has been stated in Eq. 0.11 that model classes are associated with domain classes in order to create model instances using the data stored in the related domain object. For example, an instance of a discrete-time/discrete-space model of the reservoir is created by retrieving the continuous storage-discharge functions of its reservoir domain object and discretising them to create the release tables. These tables are a characteristic parameter of the reservoir model and, using the same set of storage-discharge functions, many alternative models can be generated, changing the discretisation of the inflow, storage and control inputs of the reservoir. In this case the *link* field must provide a *reference* to the data attribute of the domain class and a *data transform function* which is the function needed to convert the data from the format which is used in the domain class to the one used in the model class.

Another example may be provided by the inflow to the reservoir $a(t)$. One might be interested in simulating different models of the same reservoir against the same input data set. In this case the *link* field is used to reference the $a(t)$ data attribute of the reservoir model class. The optional data transform function could convert the data sampling of the time series stored in the reservoir domain object to the sampling needed to run the model. While it is up to the user to define these links and to implement the data transform functions, these are one-off operations which are valid for all the model instances generated from the same model class.

Figure 0.3 shows which model data attributes are linked to the ones of the domain class. The model input a_t is linked to the input data attribute $a(t)$ of the reservoir domain class. The other model input s_t is the reservoir storage at time t as is therefore related to the local data attribute $s(t)$, as the model output s_{t+1} that is the storage at time $t+1$. It is noteworthy that both a model input and an output refer to the same local data attribute, this is common when a discrete-time model is used to describe the behaviour of a continuous process, such as the water balance in a reservoir. Finally, h_t, r_t, and R_t are linked to the corresponding data attributes in the reservoir class. These links contain the data transform functions which map the continuous functions in the reservoir class to the discrete representation needed in the model.

In Figure 0.4 it is also shown how different model classes can refer to the same domain class. In this example, once the modeller has created an instance of the domain class, the Valtellina catchment, two model instances can then be derived which differ by the kind of model formulation which was employed (the ARX model, which takes into account exogenous inputs such as rainfall, and the ARMA model).

In Figures 0.3 and 0.4 some of the model inputs and outputs are "dangling" (such as u_t in Figure 0.3) because the basic model is used as a component of a compound model. In this new situation the *link* field refers to one of the elements in the input, output, state and parameter sets of the compound model. An explanation of this usage is presented in Section 0.5.2 when describing the compound models.

A basic model can also be related to a compound domain class. In fact, the same domain object could be described by models at different scales of resolution. The modeller can organise the knowledge about the modelling domain in a structured way, where domain classes are made up of sub-domain classes and so on. The same modelling domain can be accessed by a model (written by another modeller) that has a more shallow view of the domain. Thus, it is sometimes convenient to have a basic model operate on a domain class which has a deep data structure (that is composed of sub-classes). The basic model will be able to access these data through data aggregations and mappings. An example is reported in Figure 0.5 where a black box ARX Model describes the compound domain object which represents the Tresa catchment. In the representation of the Tresa compound domain object there are four sub-catchments on which four different rainfall measurements ($w_i(t)$ for $i : 1 \ldots 4$) were gauged. These measurements are aggregated into their average $w(t)$.

This situation is common when the modeller wants to integrate a legacy model, written independently of the description of the domain base, in this framework.

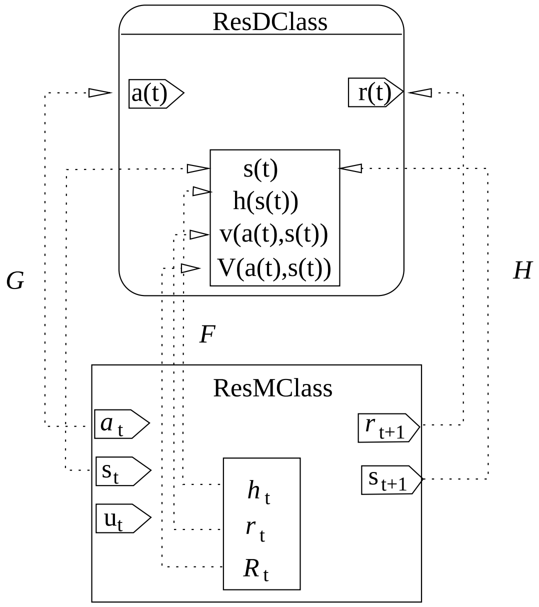

Figure 0.3: A Reservoir basic model class is linked to a Reservoir class, Symbolic links are drawn to associate a data source with the model variables

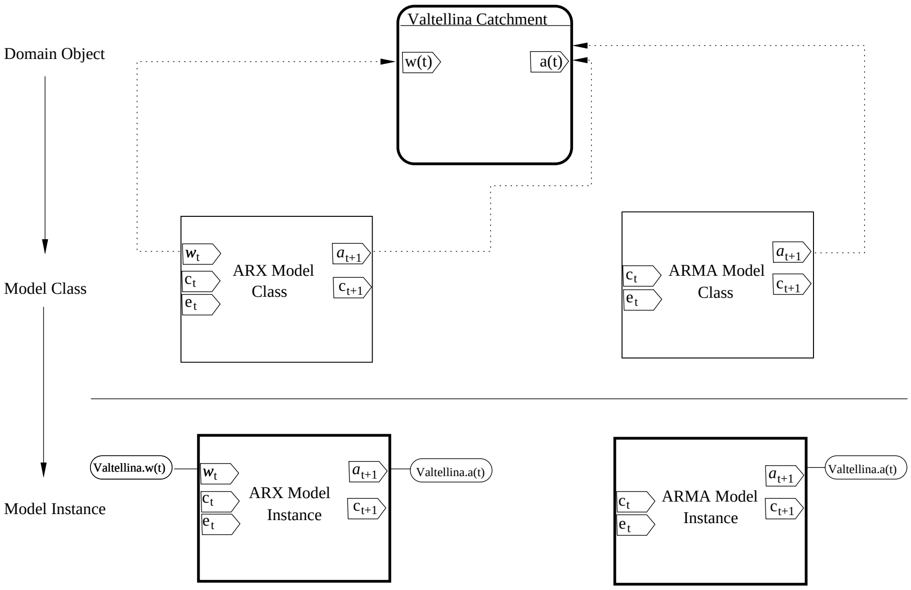

Figure 0.4: The same domain object can generate multiple model instances

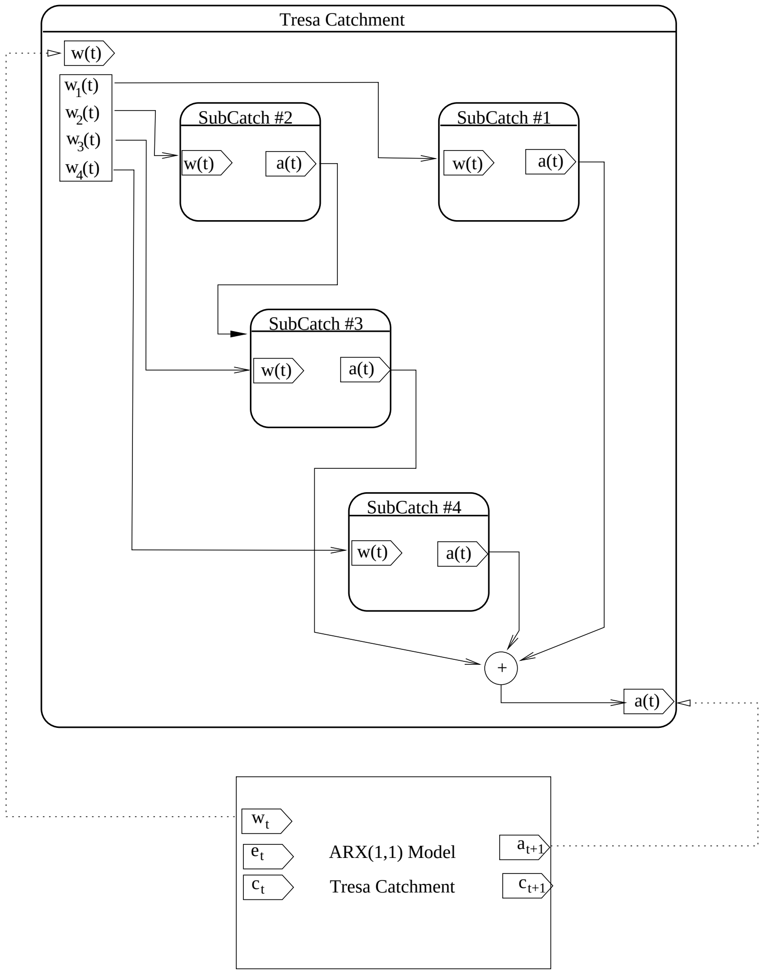

Figure 0.5: A basic model can be associated with a compound domain object.

0.5.2 Compound models

Like basic models, compound models are built on domain classes. Frequently it happens that there is one and only one set of domain objects, arranged according to a particular structure, that satisfies a particular compound model. An example is provided by the compound model of a watershed. When a real world watershed is modelled, its structure it is often so complex that there will be only one set of interconnected sub-watersheds that describes the structure of the compound watershed. On the other hand, a compound model for a stratified lake, composed of a set of basic models for each layer, can be applied to a wide variety of cases, not only to a particular lake.

A compound model is identified by:

- a domain class (with a non-empty set of sub-parts);
- a unique model identifier;

where the set of sub-parts of the domain class is put in relation to:

- sub-domain class model unique identifier;
- influence links: "data from" and "data to" domain classes.

A compound model is therefore defined by:

$$CMClass = < DClass, \mathbf{components}, \mathbf{u_c}, \mathbf{l_c}, \mathbf{y_c}, \mathrm{L} > \tag{0.16}$$

Where $DClass$ is a domain class and **components** is the list of model classes which compose the compound model, defined as follows:

$$\mathbf{components} = < (MClass^1, model_1), \ldots, (MClass^m, model_m) > \tag{0.17}$$

L is the mapping describing the linkages among the model classes which compose the compound model. This mapping is defined as:

$$\mathrm{L} = \begin{cases} \mathbf{u_c} \rightarrow \mathbf{u_b} \\ \mathbf{y_b} \rightarrow \mathbf{l_c} \\ \mathbf{l_c} \rightarrow \mathbf{u_b} \\ \mathbf{y_b} \rightarrow \mathbf{y_c} \end{cases} \tag{0.18}$$

Note that the composing models are *never* directly linked. The data exchange always happens through the special local data attributes $\mathbf{l_c}$. This is to ensure the re-usability of sub-models as it is explained in the following Section 0.5.3.

The data attributes of a compound model are classified as input, local and output and are defined as:

$$\mathbf{u_c} =< (MData, u_1), \ldots, (MData, u_m) > \tag{0.19}$$

$$\mathbf{l_c} =< (LData, l_1), \ldots, (LData, l_n) > \tag{0.20}$$

$$\mathbf{y_c} =< (MData, y_1), \ldots, (MData, y_p) > \tag{0.21}$$

The data type $LData$ (local data) differs from the data type $MData$ only because it lacks the *link* field, since the purpose of these data attributes is to provide an intermediate storage to connect the outputs to the inputs of the composing sub-models and not to access the data storages.

An example of a compound model class is reported in Figure 0.6. The input data attributes $\mathbf{u_c}$ are: the exogenous input w_{t+1}; the input disturbance e_{t+1}; the catchment state c_t; the reservoir storage s_t; and the water release decision u_t. These data attributes provide the input interface to the model, which will be useful when operating the compound model, as shown later in Section 0.6.

The local data attributes $\mathbf{l_c}$ are the catchment runoff a_{t+1} which is then used as an input by the reservoir model, and the reservoir's water release r_{t+1} which is fed into the water consumer (the HydroUser model).

The output data attributes are the catchment and reservoir states at the next time step (c_{t+1}, s_{t+1}) and the step cost g_{t+1} at time $t+1$.

In Figure 0.6 the arrows are directed from the sub models' inputs to the compound model's inputs to signify that data are retrieved *from* the compound model and fed *into* the sub models. On the other hand the arrows are directed from the sub models' outputs towards the compound model's outputs. This means that the output values are stored in the outputs of the compound model and that the sub-models' outputs refer to them.

0.5.3 Interchanging and connecting models

A focal point of the MMS architecture presented in this paper is model interchangeability. This means that the same model can be applied to various objects and that the same object can be modelled by different models (in different simulation runs). When a new model is applied to an object, a new data set may be accessed. This is shown in the lattice of Figure 0.7 where the generic domain class `CatchDClass`, which describes a catchment structure, can be used to create different catchment instances: using two alternative data sets to represent the same physical catchment (`ValtellinaCatch_1` and `ValtellinaCatch_2`) and another catchment: `TresaCatchment`. In this example, two alternative model classes (`CatchMClass:ARX(m,q)` and `CatchMClass:ARMA(p,q)`) are associated with the generic domain class. This allows the modeller to produce a series of model instances, coupling the domain objects with the model classes. In

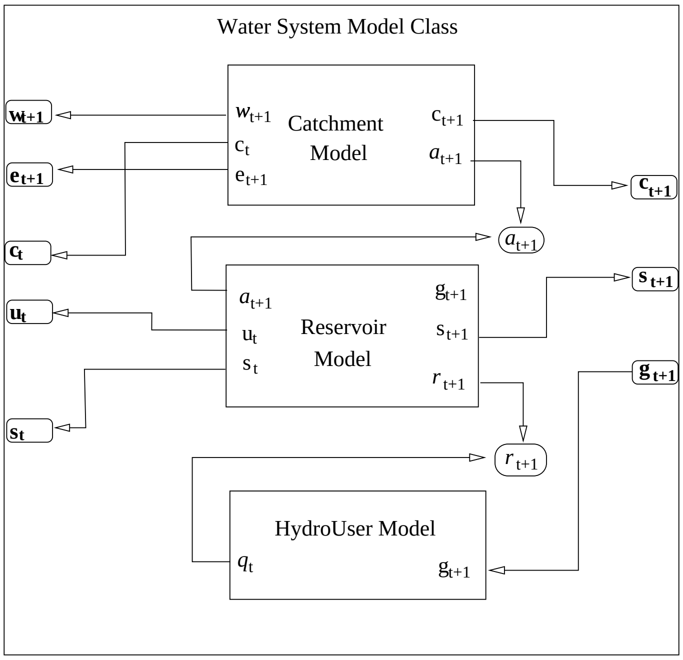

Figure 0.6: A compound model class for the Maggiore Water System

this way, for instance, `CatchModel_A` can be generated which is an instance of `CatchMClass:ARX(m,q)` and the `CatchModel_B`, which is an instance of `CatchMClass:ARMA(m,q)`, both applied to the case of the Tresa Catchment.

The advantage of this model and data organisation is more evident when assembling compound models. In Figure 0.8 two compound models of the Lake Maggiore water system differ by the catchment model instances they use: the ARX or the ARMA models. In this case model substitution is performed automatically by the MMS: the new model instance knows where to gather the input values and where to store its outputs, without any further user intervention, since all the required knowledge is embedded in the model class.

Model interchangeability is not always a straightforward process. Interchangeability is not independent of interconnection. A model can substitute for another only if both comply to the same interface requirements. For instance, model `CatchModel_A` (in Figure 0.8) does not require a value for the rainfall w_{t+1} which is instead required in model `CatchModel_B`. While this case does not stop the compound model to work, since there is an "over-abundance" of model inputs, it could happen a situation where substituting a model with one endowed with a bigger input interface set can render the compound model unusable.

In such cases, model substitution cannot be delegated to the system and

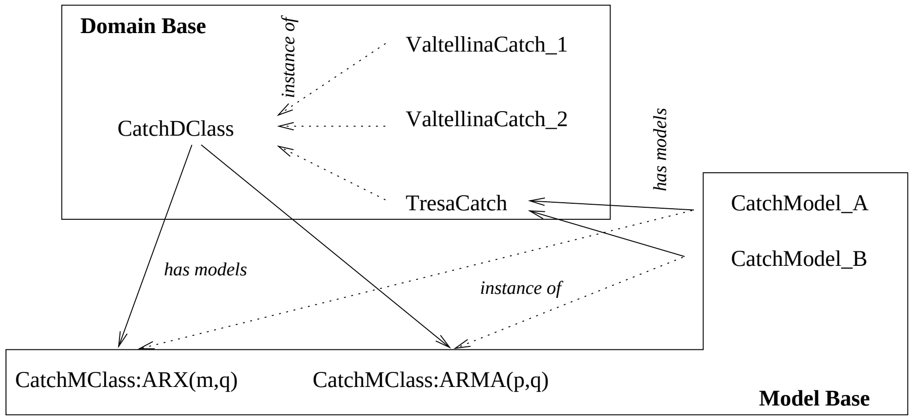

Figure 0.7: The domain base and the model base can be combined to create new models.

the user has to decide whether another model should be used or the domain and model classes need a re-design phase.

0.6 Putting the MMS to work

This section reports an example of how the proposed MMS framework is applied. One of the authors of this paper has been implementing a decision support system for the operations of multi-purpose reservoirs [28]. This system requires models for optimisation – to generate a reservoir management policy and to suggest operation decisions – and simulation – to assess policy performance and its impact. For this reason, the models must be handled by "optimisation engines" (solvers) and "simulation engines" (simulators).

The optimisation engine implements the Bellman dynamic programming algorithm [2] to find an optimal solution to the problem. The system analyst who wants to produce a policy can "plug-in" one of the models which were devised for this purpose. The MMS provides the software architecture which makes the solver *independent* from the model formulation. Figure 0.9 shows how the model of the Maggiore Water System is linked to the solver: the input interface of the compound model fetches its values from the solver, according to the dynamic programming search routine, all the values for the model inputs are tried and the cost g_{t+1} and system state x_{t+1} (the couple c_{t+2} and s_{t+1}) are fed back into the solver. The solver also needs a decision model which defines the performance indicator J and the constraints.

The same solver can be used to solve another model, with a different input and output interface. The solver, in fact, reads the characteristics of the model interface to generate the appropriate "stimuli" during the optimisation algorithm. Typically, the solver reads the discretisation characteristics

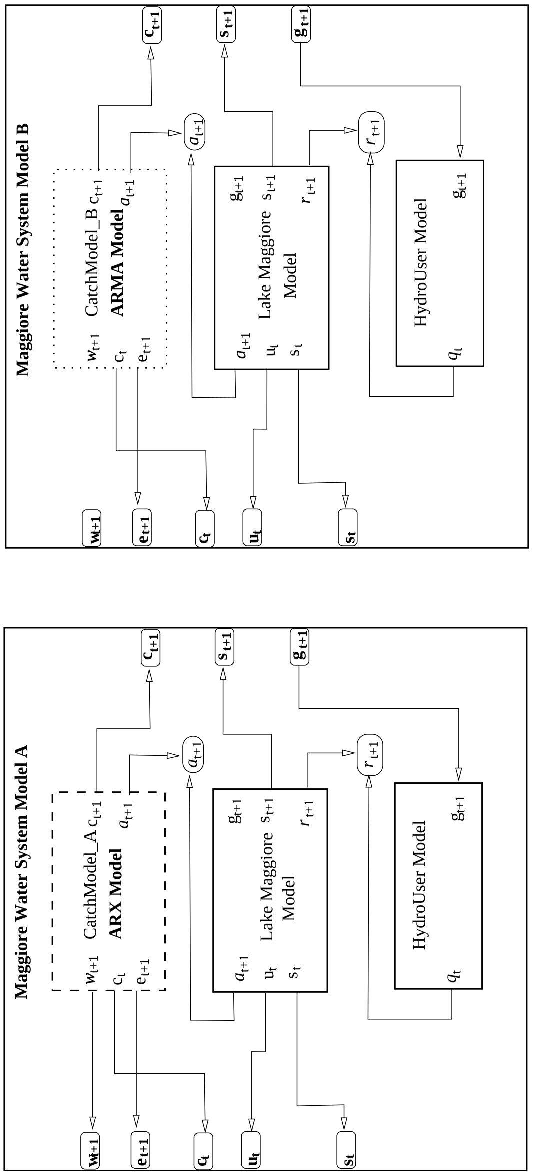

Figure 0.8: Model substitution for the Maggiore Water System

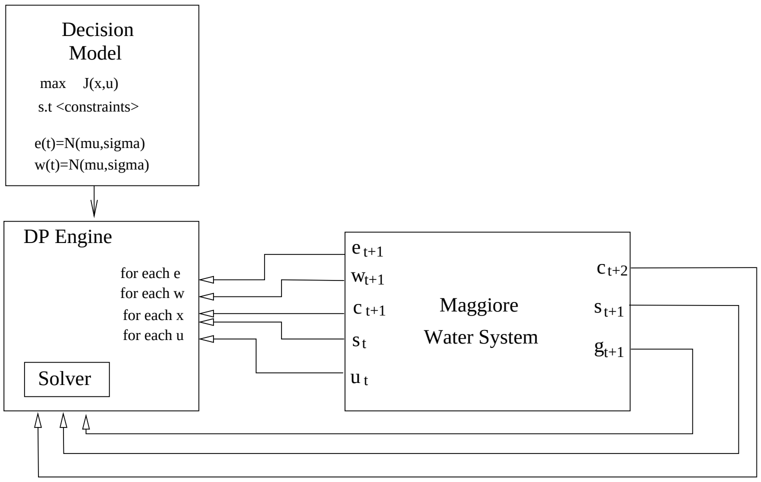

Figure 0.9: A regulator for the Maggiore Water System is produced using a solver

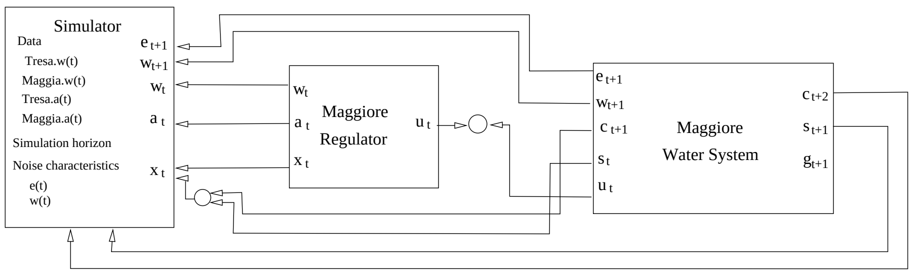

Figure 0.10: Simulation of the Lake Maggiore water system

of the input variables to be able to appropriately span the ranges of the input variables with all the possible values in the discrete sets.

The result of the optimisation is a regulating policy which returns the amount of water to be released given the time and the state values. Since the catchment state is not observable, it must be reconstructed by a Kalman filter [2]. Therefore the policy and the state re-constructor make up a regulator model which is used to produce the regulating decision. Figure 0.10 shows how the regulator is used to produce daily decisions of water release. This time the solver is a simulator which feeds time series for rainfall ($w(t)$) and for observed catchment runoff ($a(t)$). These measurements are used either to feed the simulated model and to reconstruct the non-observable part of the systems' state.

The MMS we have presented here has been embedded in a DSS and used to explore the impact of alternative interventions on the management of Lake

Maggiore, which is located at the border between Italy and Switzerland. The capability of testing different models against the same data sets was an asset in evaluating the management alternatives [31].

0.7 Conclusions

The Model Management System presented in this paper shows how the modelling knowledge and the available data, represented in the domain base, can be organised in order to enhance model integration and re-use. Models are linked to domain classes of objects and they communicate through their interfaces. Models are therefore separated from a particular domain object, and can be re-used in problems that have a similar structure. The approach proposed here fulfills the requirements of an effective MMS (Section 2).

1. The proposed architecture helps the modeller abstract the model from the data, thereby making it easier to re-use existing models and create new ones.

2. Models are associated with domain classes which are real world entities or processes. Thus, models are related to representations of the modelling domain that have a meaning to the user.

3. The MMS is deliberately designed to allow models to be linked together. Linkage is achieved via the model interfaces.

4. The MMS has the same capabilities as a DBMS. Models can be stored, retrieved, deleted, and edited as if they were data items in a data base. Domain objects and models can be made persistent and therefore be treated as data items.

5. Domain classes provide meta-level data descriptions used by models to access data types.

6. The ability to associate a basic model with a compound domain class leads to a seamless integration of a "legacy" model into the MMS.

In particular, it is relevant to notice the separation between the models and the data descriptions. The MMS approach presented in this paper gives data the same standing as models by the definition of domain classes. This design solution allows the re-use of data, not only of models, thus allowing the user to easily create modelling alternatives which can be applied to the same data sets.

The MMS design presented here is currently being implemented in the "Open Modelling" software [27], in the HYDRA project [13] and in a two-level Decision Support System for the operations of reservoir networks [28]. Although this design covers the requirements of a MMS, many practical details have to be settled during the implementation. Modelling using such software designs represent a major step towards more efficient modelling and, given the importance of predictive models to DSSs, more efficient DSS development.

Bibliography

[1] D. Abel, K. Taylor, D. Kuo, Integrating Modelling Systems for Environmental Management Information Systems, ACM-SIGMOD, Vol. 26, No. 1 (1997).

[2] D.P. Bertsekas, Dynamic Programming and Optimal Control (Athena Scientific, MA, 1995)

[3] H.K. Bhargava, S.O. Kimbrough, Model Management: an Embedded Languages Approach, Decision Support Systems, Vol. 10 (1993).

[4] R.W. Blanning, R. Krishnan, R. Müller. Decision Support on Demand: Emerging Electronic Markets for Decision Technologies. Decision Support Systems, Vol. 19, No. 3 (1997).

[5] G. Booch, Object-Oriented Analysis with Applications - Second Edition (The Benjamin/Cummings Publishing Company, Redwood City, 1994).

[6] CORBA, Object Management Group (`http://www.omg.org/`).

[7] D.R. Dolk, J.E. Kottemann, Model Integration and Modeling Languages: a Process Perspective, Information Systems Research, Vol. 3, No. 1 (1992).

[8] D.R. Dolk, J.E. Kottemann, Model Integration and a Theory of Models, Decision Support Systems, Vol. 9 (1993).

[9] L. Del Furia, A. Rizzoli, An Integrated Modelling Environment for Object-Oriented Simulation of Ecological Models, in: Proceedings of the 26th SCS Annual Simulation Symposium, Washington, D.C. (March 29-April 1 1993).

[10] B. Falkenheimer, K.D. Forbus, Compositional Modelling: finding the Right Model for the Job, Artificial Intelligence, Vol. 51 (1991).

[11] P.A. Fishwick, Qualitative Methodology in Simulation Model Engineering, Simulation, Vol. 52, No. 3 (1989).

[12] J.W. Forrester, Industrial Dynamics (MIT Press, Cambridge, MA, 1961).

[13] J.R Davis, D.J. Abel, D. Zhou, A. Rizzoli, P. Kilby, HYDRA: a Generic Design for Integrating Catchment Models, Presented at the American Society of Civil Engineers 21st Annual Conference on Water Resources Planning and Management Division, Denver, Co. (22-26 June 1994).

[14] DMSO, High Level Architecture (`http://hla.dmso.mil`).

[15] G. Guariso, H. Werthner, Environmental Decision Support Systems (Ellis Horwood Limited, Chichester, 1989).

[16] G. Guariso, E. Tracanella, L.Piroddi, A.E. Rizzoli A web accessible environmental model base: a tool for natural resources management. Proceedings of MODSIM 97. Hobart. Tasmania. D. McDonald, M. McAleer, A. Jakeman (eds.), 8-11 December 1997. (GAIA is available on-line at `http://www.ess.co.at/GAIA/`).

[17] B. Henderson-Sellers, J.R. Davis, I.T. Webster, J.M. Edwards, Modern Tools for Environmental Management: Water Quality, in: A.J. Jakeman, M.B. Beck and M.J. McAleer, Eds., Modelling Change in Environmental Systems (John Wiley & Sons, New York, 1993).

[18] S.N. Hong, M.V. Mannino, B. Greenberg, Measurement Theoretic Representation of Large, Diverse Model Bases: the Unified Modeling Language LU, Decision Support Systems, Vol. 10 (1993).

[19] M. Knorrenschild, R. Lenz, E. Foster, C. Herderich. UFIS: a database of ecological models. Ecological Modelling, Vol. 86 No. 2–3, pp. 141–144 (1996). (UFIS is available online at `http://www.gsf.de/UFIS/ufis/ufis_proj.html`)

[20] H. Lieberman, Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems, in: Proceedings of OOPSLA-86, Portland, OR (1986).

[21] M. Minsky, A Framework for Representing Knowledge, in: P.H. Winston, Ed., The Psychology of Computer Vision (McGraw-Hill, New York, 1975).

[22] W.A. Muhanna, SYMMS: a Model Management System that Supports Model Reuse, Sharing, and Integration, European Journal of Operational Research, Vol. 72 (1994).

[23] Nardini, A., C. Piccardi, and R. Soncini-Sessa, On the integration of risk aversion and average-performance optimization in reservoir control, Water Resour. Res., Vol. 28, No. 2 (1992).

[24] K. Parsaye, M. Chignell, S. Khoshafian, H. Wong, Intelligent Databases: Object-Oriented, Deductive, Hypermedia Technologies (John Wiley & Sons, New-York, 1989).

[25] F. Pichler, R. Moreno-Diaz, Eds., Computer Aided Systems Theory -EUROCAST '89, Lecture Notes in Computer Science Vol. 410 (Springer-Verlag, Berlin, 1990).

[26] W.D. Potter, T.A. Byrd, J.A. Miller, K.J. Kochut, Extending Decision Support Systems: the Integration of Data, Knowledge, and Model Management, Annals of Operations Research, Vol. 38 (1992).

[27] A. Rizzoli, J.R. Davis, M. Reed and T. Farley, A DSS for catchment management, in: P.Zannetti, Ed., Environmental Modelling - Vol 3, Computer Methods and Software for Simulating Environmental Pollution and its Adverse Effects. (Computational Mechanics Publications, Southampton, 1996).

[28] A. Rizzoli, R. Soncini-Sessa, Integrating and complementing Human Experience in Water Management with a Two-level DSS, in: F. Burstein, H. Linger, H. Smith (eds.) Proceedings of the Workshop on Intelligent Decision Support, IDS '96,, Melbourne (9 September 1996).

[29] P. Robertson, Integrating Legacy Systems with Modern Corporate Applications, Communications of the ACM, Vol. 40, No. 5 (1997).

[30] J.W. Rozenblit, P.L. Jankowski, An Integrated Framework for Knowledge-Based Modeling and Simulation of Natural Systems, Simulation, Vol. 57, No. 3 (1991).

[31] R. Soncini-Sessa, D. Canuti, A. Colorni, E. Laniado, F. B. Losa, A. Rizzoli, L. Villa, B. Vitali. Planning and management of a transnational water system, the case of Lake Maggiore, Italy-Switzerland. Presented at: International Workshop on barriers to Sustainable Management of Water Quantity and Quality, Wuhan, China, (12-15 May 1998).

[32] M. Stefik, D.G. Bobrow, Object-Oriented Programming: Themes and Variations, AI Magazine, Vol. 6, No. 4 (1986).

[33] R.H. Sprague jr., E.D. Carlson, Building Effective Decision Support Systems (Prentice Hall, Englewood Cliffs, 1982).

[34] B.P. Zeigler, G. Klir, M. Elzas, T.I. Oren, Methodology in Systems Modelling and Simulation (North Holland, Amsterdam, 1979).

[35] B.P. Zeigler, Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems (Academic Press, New York, 1989).