Computational Modeling Systems: Supporting the Development of Scientific Models*

Terence R. Smith   Jianwen Su†

Amr El Abbadi   Gustavo Alonso   Amitabh Saran‡

Department of Computer Science
University of California
Santa Barbara, CA 93106
USA

March 31, 1994

*Work supported in part by NSF grant IRI-9117094.
†Work also supported in part by NSF grant IRI-9109520.
‡

Abstract

We develop a conceptual model of the iterative process of scientific modeling. This model provides an appropriate specification for a large number of concepts relating to those aspects of the scientific modeling process that require significant computational support. In particular, we develop, define and exemplify the unifying concept of *representational structures.* We employ the conceptual model in developing a framework for a "computational modeling system" (CMS) that supports scientific activity within a simple, unified, computational environment. A high level computational modeling language (CML) is presented with which the modeling concepts that are definable in the computational framework may be constructed and manipulated in a simple, uniform manner. Finally, we describe the design and implementation of a specific computational modeling system, *Amazonia*, which is intended to support scientific research in the area of earth science. A particularly significant contribution of the paper is the development of a strong conceptual and theoretical basis for the concept of a CMS.

Contents

1 INTRODUCTION

In many areas of scientific and engineering research, there is a fundamental need for comprehensive and integrated computational support for the development, evaluation and application of *symbolic models* [Req80] of a broad range of phenomena. Activities that require computational support during the iterative development of symbolic models [Rob91, HTM93] of phenomena range from the acquisition and manipulation of raw data to the construction and evaluation of complex sets of mathematical equations [SSU90, Doz90, Chu93, IEE93, SSD89, SSD91, SSD93, HQGW93, HGW93, Lea92, OM93]. Current computational support for these activities typically involves a heterogeneous collection of tools supporting limited aspects of scientific modeling activities, such as numerical computations, data management, and image processing. Rarely is such support integrated into a single, comprehensive system, and there is typically little or no support for the modeling enterprise as a whole. It is our observation that in many earth science research projects, for example, a very significant proportion of the research effort is focused on computational issues that are largely irrelevant to the scientific research. Significant increases in efficiency are possible for applications in which computation is a major activity.

Integrated computational support for scientific modeling activities is a difficult goal to achieve for a variety of reasons. First, the construction of scientific models of phenomena is a complex, highly interactive, and highly iterative process. Second, the modeling process is typically distributed over a number of conceptually and physically distinct environments and subenvironments. Third, scientific modeling activities typically involve a large and evolving body of concepts that relate both to the phenomena of interest and to the phenomenon of the modeling process itself and there are frequently multiple representations of these concepts. Fourth, scientists are typically conservative in adopting "new approaches" until they are of clear net advantage to their scientific research.

The lack of comprehensive and unified computational environments that hide *scientifically irrelevant* computational issues leads to many difficulties and inefficiencies [SSAE93b, SSAE93a]. One instance is the management of scientific data. In essence, current database support has not been designed on the basis of any deep and general characterization of scientific modeling activities. In particular, it does not take account of the fact that it is almost impossible, from a scientific point of view, to separate the set of activities that are typically labelled as "database support" from other modeling activities, except to the degree that scientists find it of value to employ the concept of "data" in relation to a certain class of scientific entities and a certain class of operations. Hence database environments that support the handling and management of "data" are typically separate from and unrelated to the programming language environments and other environments that are better able to support the body of concepts involved in the scientific modeling process; they do not provide any support for the fact that scientific modeling typically occurs in a variety of environments; and they provide little or no explicit support for the fact that there are frequently multiple representations for many entities and relationships.

A *computational modeling system (CMS)*[1] is intended to provide scientific investigators with computational support allowing them to achieve their scientific goals more efficiently. In particular, a CMS is intended to provide a unified computational environment in which scientific investigators are provided with sets of tools that support a significant range of the activities relating to the process of constructing, evaluating and applying symbolic representations of phenomena. In its most general sense, a CMS may be viewed as a fusion of three relatively distinct components: a

[1]In previous reports, we have labelled such systems *modeling and database systems* or MDBS. This has not turned out to be the best of names, and we find CMS easier and more appealing.

computational environment that is designed on the basis of a comprehensive and consistent model of scientific modeling activity; a *knowledge base* that may be tailored for any specific domain of scientific investigation; and *persistent programming support* that combines compiler, database and graphics technology and that is transparent with respect to scientifically irrelevant details of the computation.

We believe that a CMS should be based on a comprehensive and appropriate model of scientific modeling activities. Such a model should include a characterization of the development, representation, and evaluation of the concepts that are employed by scientists in their modeling of both the phenomena of interest and the process of modeling itself. In developing and evaluating the concept of a CMS, we have interacted closely with multidisciplinary groups of scientists, since we believe that support for scientific modeling activities *must* be based on a deep understanding of scientific activity.

The main contributions of the research that are reported in the current paper include:

- the development of a conceptual model of the process of scientific modeling that provides an appropriate specification of a large number of concepts relating to those aspects of the process of scientific modeling that may be provided with significant computational support.
- The development of a framework that represents a translation of our conceptual model of scientific activity as a simple, unified, computational framework that we term a *CMS*.
- The definition of a simple computational modeling language (CML) with which the modeling concepts that are definable in our computational framework may be constructed and manipulated in a simple and uniform manner.
- The design and implementation of a specific CMS, *Amazonia*, that is intended to support scientific research in the area of earth science.

We believe that a particularly significant contribution of the paper is the development of a strong conceptual and theoretical basis for the concept of a CMS. This basis provides clarification for a large number of issues that relate to scientific modeling activity and its computational support. We note that other approaches to the development of such support are typically based on evolutionary extensions of existing systems (e.g., [PSS$^+$87, Wol89, Wol90, WH92, vOV91, dHvO92, Seg93, AS93, MP94, Jon90, Jon91]).

We structure the paper as follows. In the following section, we provide a simple example of scientific modeling that is drawn from the earth sciences. We use this example throughout the paper in order to illustrate many of the issues that are raised and discussed. We then describe in some detail the conceptual model of scientific modeling activities that we have developed during our research. This model provides the basis for our design of a CMS in general and of our "data model" in particular. In order to express our concepts as clearly as possible, we have incorporated them in the paper in terms of "informal" definitions. We then illustrate the model in terms of our earth science example. In the following section, we develop a simple computational modeling language (CML) that is closely related to our conceptual model of scientific activity. We end the paper with a discussion of systems support for a CMS and the current architecture of the CMS under development. In the appendices, we indicate the mathematical foundation on which our conceptual model rests, together with additional information on how our conceptual model captures the semantics of general scientific modeling situations.

2 THE AMAZON PROJECT: AN EXAMPLE OF SCIENTIFIC MODELING ACTIVITY

In order to motivate our conceptual model of scientific activity and to illustrate many of the issues that we raise in this discussion, we now provide a brief description of a particular example of a set of computational modeling activities drawn from the geological and hydrological sciences. For illustrative purposes, we focus on a *highly-simplified* version of a problem that is under investigation by a group of earth scientists who are constructing, testing and applying models of the flow of water, solutes and sediment within the Amazon river basin [2].

The simplified problem that we use to illustrate our concepts involves the construction of acceptable models of the flow of water and other materials in a natural drainage basin. The goals of the scientific modeling activities include the ability to

- construct, evaluate and store acceptable representations of observed land surfaces (including networks of river channels), over which the flow of water is to be modeled, observed rainstorm events, and observed flows of water at a given set of locations on the surfaces.
- Construct representations of flows and properties of flows that would occur over the observed surface in response to observed rainstorms, given a representation of some flow generating process.
- Compare appropriately representations of properties of the modeled flows with representations of "corresponding" observations of "real-world" flows.

For the sake of simplicity, we will assume in the remainder of the paper that all aspects of the modeling process are based on the use of functions that are evaluated at discrete points over some set of rectangular, spatial grids. This is a typical assumption in many similar modeling situations.

2.1 Scientific Projects and Current Computing Environments

One may view a specific set of modeling activities as taking place within the context of a *project*. The long-term goals of the Amazon Project, for example, include the development and application of spatially-distributed models of water, sediment and solute transport in the Amazon basin. This project is embedded within the larger context of the EOS Project [EOS90]. A goal of the EOS Project is to provide a large collection of teams earth scientists with common access to a huge database of earth system observations by means of the EOS-DIS (Data and Information System) [Doz90]. Typical direct uses of the EOS-DIS are expected to be heavily focused on the search and retrieval of datasets by research teams, followed by the incorporation of useful subsets of these datasets into the "local" databases of the research teams.

A given research team may support a variety of research projects, each of which may have various *subprojects*, and may also interact with research teams involved in related projects. The set of EOS-DIS/project/subproject/"related-project" interactions typically involve some form of sharing of datasets and of modeling procedures. The Amazon Project, for example, has subprojects relating to the various hydrological models of the Amazon basin at different scales of spatial resolution, to models of the vegetation coverage, to models of geochemistry and to models of continental scale geomorphology. Each subproject has access to subsets of the data and modeling procedures that are available to the project as a whole and typically interacts with research teams involved in other research projects. The hydrological modeling subteams, for example, interact with scientists of

[2] The research in question is being carried out at the University of Washington, as part of NASA's EOS project, by our co-investigators on NSF- and other NASA-supported projects.

other projects that involve climatic modeling. [3]

We have found from our joint research with the multidisciplinary scientists that there is a general lack of adequate computational support for the *iterative* construction and testing of their models. The generation of research results in many research project appears to be heavily impacted by the large size and processing requirements of the models. This problem is further compounded by the requirement to develop the models using a large and heterogeneous collection of datasets and the frequent need to couple several complex models. Most researchers lack genuine *database support* for managing ever-growing collection of datasets, which may contain information in the form of satellite images and large collections of digital datasets relating, for example, to topography, vegetation, geology, soils, meteorology and hydrology.

In large part, these data-handling difficulties result from the use of existing file systems as a repository for datasets. The varied contents, formats, lineage, and the large size of datasets result in an unmanageable collection of files scattered over a network. The absence of database modeling and management further complicates the task of maintaining these datasets, which undergo continued growth and evolution. In addition, efficient access to the contents of these datasets is severely restricted in the current environment. Additional difficulties arise from the fact that while modeling and database activities are closely related from a scientific point of view, they are artificially separated because of inadequate computational support. Furthermore, there is no computational support for coordinating the modeling and database activities of researchers involved in joint projects.

2.2 Representations of Observed Phenomena

We now provide a brief and greatly simplified description of a few of the modeling activities involved in the Amazon Project. A first set of modeling activities involves a variety of operations involving representations of "observations" on the phenomena of interest (ie involving "datasets"). For current purposes, it suffices to consider three such classes of representations: *digital elevation models* (DEMs), *rainfall measurements* and *river stage observations*. It is typically the case that sequences of procedures are applied to such instances, with some of the procedures being relatively complex. Various classes of "errors" are common in instances of these three classes of representations, and sequences of applied procedures may lead to the accumulation of errors in the resulting representations. Although difficult, it is frequently important to attempt to evaluate such errors. In much of this type of processing, it is typical for scientists to work in an iterative mode that involves computation, visualization, and procedure modification.

The DEMs may be viewed as empirically-derived, low-level representations of the surfaces over which flows are to be modeled. The basic information may be viewed as a set of pairs of locations and associated surface elevations. Particular DEMs may be characterized in terms of several variables including the locations of the sets of points, the coordinate systems in which the points are represented, the errors in the representations of the points and elevations, and the logical and physical data models for the representation. Significant processing issues arise for scientists in cases in which they must integrate several distinct sets of data. For example, it may be necessary to "register" underlying grids of points.

We assume that for the modeling problem of interest, the DEMs must be processed in order to

[3] We note at this point that it is important to model the concept of project in order to support the idea of data/procedure sharing and to support the notion of queries that concern, for example, which subproject uses a particular algorithm.

extract representations of the relevant concepts that are to be employed in modeling surface features that are significant for modeling flows over such surfaces. Important issues, therefore, relate to the set of concepts that are employed in modeling relevant surface features and the logical and physical representations of such concepts. There are no standard representations, either logical or physical, in which to represent the intermediate or final outputs of the processing. In the following we assume that appropriate concepts include, among others, *channel segment, channel segment slope area, channel segment drainage area, basel, basin.* The processing is complex and requires decisions concerning the use of a variety of heuristic procedures that typically involve relatively intensive computations.

The basic information in *rainfall measurements* may be viewed as a set of triples of location, time and rainfall depth. The information may have missing values and typically involves errors. The estimation and insertion of missing values and corrected values may require sophisticated models of rainfall measurement and rainfall events. A significant operation involves interpolation of rainfall values over some area, typically represented as a complete rectangular grid of points. It is therefore necessary to choose and apply one of a large number of interpolation procedures, some of which may require sophisticated models of rainfall generating processes that relate to the elevation of the land surface.

As in the case of DEMs, rainfall measurements must be processed in order to extract representations of the relevant concepts that are to be employed in modeling rainfall features that are significant for modeling flows over such surfaces. If it is desired to extract *storm events* from interpolated information, it is necessary to define the both the concept of storm and representations for this concept, as well as implementing and applying procedures that will extract rainstorm events.

The basic information in *river stage observations* may be viewed as a set of triples of location, time and river flow height. Issues of missing values and errors may again be important. Important procedures are to extract representations of river flow rates from representations of flow depths and to partition the representations of flows into phenomenologically important components. Both procedures induce errors that lead to problematical representations. Typically, it is necessary to use relatively sophisticated hydrological models of river flow in order to partition the flows. Again it is necessary to choose a set of modeling concepts and associated representations and to extract such representations from representations of the observations. For example, the concept of *hydrograph* is a central one in this case.

In Figure 1, we illustrate some of the key entities and procedures that are involved in the hydrological modeling activity.

2.3 Constructing and Evaluating Models of Phenomena

An critical task in the construction of representations of the phenomena is to select the set of concepts that are to be employed in the modeling process and to select representations for these concepts. For example, in modeling the flows over land surfaces, one might use the concept of a *surface point flow* and a representation of this concept in terms of a triple of a location, a time and a *flow vector*. In modeling flows in channel sections, one might employ the concept *channel segment flow* and choose a representation of flow that involves a *channel segment* that possesses a specific length and direction with an *inflow* at its upper end an *outflow* at its lower end. It is also necessary to select a representation of a conceptual model of the flow generating process *surface flow model*, which itself may be composed of submodels *slope flow model* and *channel flow model*, together with representations of (and approximate representations of) these models. One may,

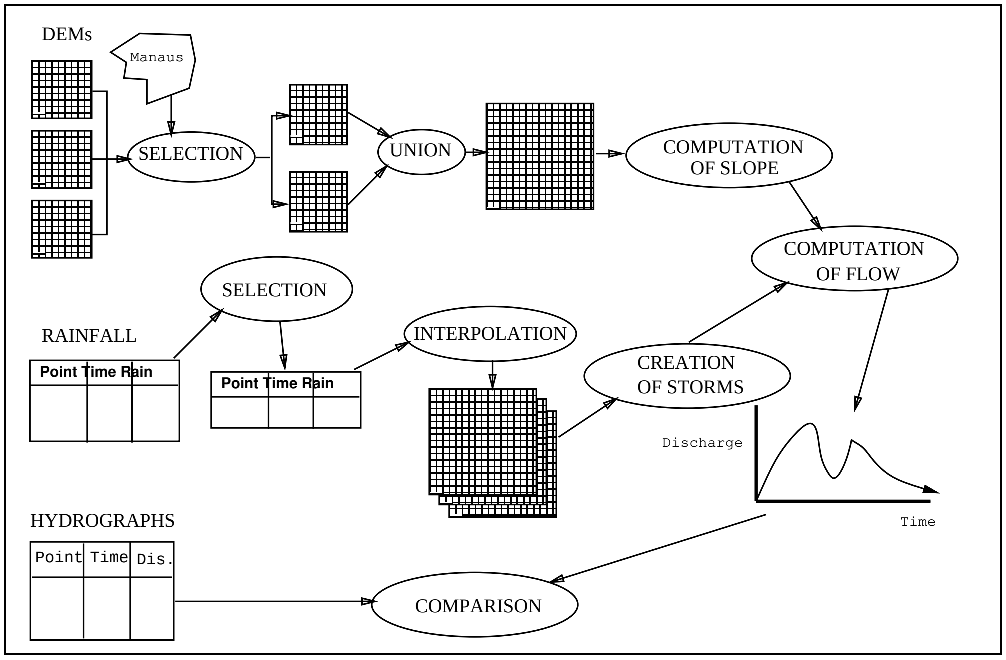

Figure 1: Illustration of operations involved in a subproblem of the Amazon Project

for example, select concepts relating to conservation conditions; choose representations of these conditions that involve partial differential equations; select approximations that involve difference equations; and select appropriate values of parameters in the difference equations. It may be the case that appropriate concepts or appropriate representations of concepts may not be available for some aspects of the modeling process. In this case a major task involves the discovery of both the concepts and their representations.

Given a satisfactory representation of a conceptual model of the flow generating process, it is then necessary to select and apply *solution procedures* to representations of the approximating equations. The application of these procedures involves, as input, the representations of the surfaces and rainstorms, as well as representations of initial flow conditions over the surfaces. These inputs may require additional processing, such as the computation of land surface slope from the DEMs. Various transformations may then be applied to the representations or the outputs of this processing. For example, it may be desirable to compute the representations of *hydrographs* at various locations on selected *channel segments*.

The evaluation of a modeling procedure may take a variety of forms. In the present example, it may take the form of applying transformations to representations of the observed and computed *hydrographs* whose output may be interpreted to indicate the "closeness of fit". Evaluations may also involve significant visualization.

It is typically of value to employ concepts that relate to the modeling process itself, such as *model* and *project*. As an example of the value of such concepts, it is frequently the case that many repetitions of complex sequences of procedures occur during the modeling process. On the basis of a concept of *sequences of operations* for which one may define a general template, it is possible to save sequences of operations in a form that reduces the effort of modification and reprocessing. We may view this process as one in which the scientist is "modeling" the modeling process. The communication of both the process of modeling and the results of the process of modeling is of critical importance, since the basic criteria of the *universality of concepts and the verification and*

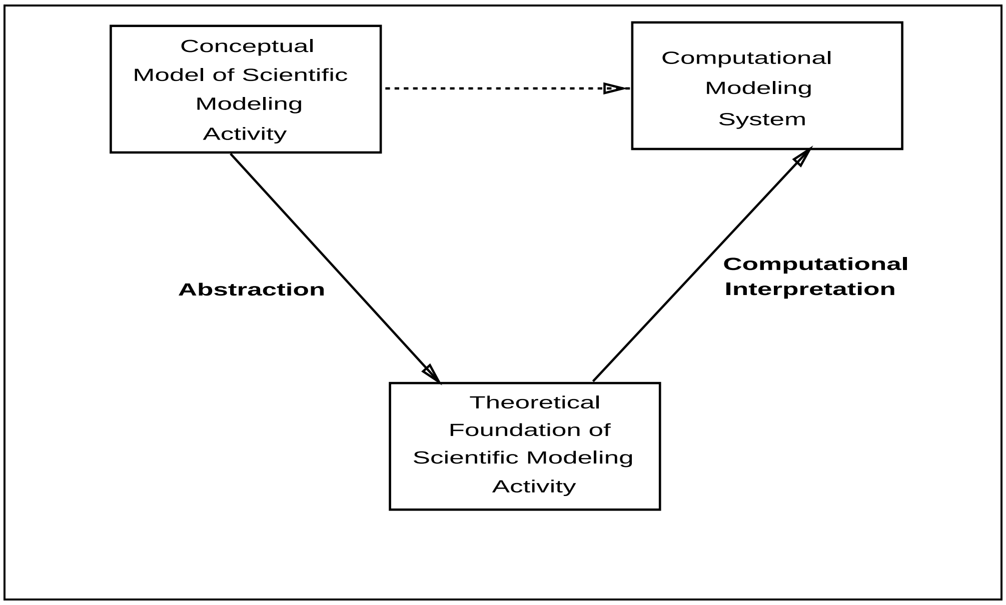

Figure 2: CMS Development Strategy

reproducibility of results lies at the core of the scientific method. Such communication typically requires a set of concepts that describe the modeling process in unambiguous terms.

3 A CHARACTERIZATION OF SCIENTIFIC MODELING ACTIVITIES

We now provide a characterization of the iterative process by which scientific models of phenomena are constructed. The contribution of this discussion is a set of concepts and associated representations that may be used to describe scientific modeling activities. Our goal in providing this characterization is to construct a foundation for those aspects of the scientific modeling process that can and should be supported by a CMS. In Figure 2 we illustrate the relation between our conceptual model of science, the logical foundations for this model and the specification of a CMS. In Appendix I, we provide a brief description of a logical foundation for this conceptual model of scientific activity. This foundation has value as an abstraction of the conceptual model, as a framework for constructing a consistent specification for a CMS and in providing information concerning fundamental computational issues that relate to the construction of a CMS.

3.1 Concepts and Representations in Scientific Modeling

It is generally accepted that the most fundamental goal of science is the discovery and investigation of *appropriate symbolic models* (or *symbolic representations*) of phenomena. Fundamentally important classes of concepts that form a basis for most scientific modeling activities characteristically possess some interpretation in terms of *identifiable entities* in some domain that is being modeled; or in terms of transformations between such entities; or in terms of relationships between entities. For our current purposes we employ the following

Definition. A modeling concept is a triple $[\mathcal{W},\ \mathcal{S},\ \mu]$ in which $\mathcal{W}$ is a domain of entities and relationships, $\mathcal{S}$ is a set of of symbolic representations in some language, and μ is a mapping from $\mathcal{S}$ onto $\mathcal{W}$. □

We term $\mathcal{W}$ the world of the concept, $\mathcal{S}$ the symbolic representation of the concept, and μ the interpretation of the concept. In some cases, this mapping is informal and cannot be completely specified within a CMS (as in the case of symbolic representations of real world phenomena) while in other cases it can be fully specified within a CMS (as in the case of symbolic representations of other symbolic representations). We emphasize that in our definition, a concept cannot be defined apart from some symbolic representation.

It is clearly possible to construct modeling concepts whose domain $\mathcal{W}$ contains the representations of other concepts. In other words, we may conceptualize concepts themselves. From the point of view of CMS, an important distinction is embedded in the following

Definition. Concepts for which $\mathcal{W}$ is part of the domain of phenomena under investigation are termed *phenomenological concepts.* Concepts for which $\mathcal{W}$ is the domain of phenomena of the modeling process itself are termed *methodological concepts.* □

In relation to phenomenological concepts, scientists typically develop, evaluate and employ a large body of concepts that possess interpretations in terms of the phenomena that are being modeled. This body of concepts may be viewed as encapsulating a large corpus of scientific knowledge concerning the phenomena. In relation to methodological concepts, it is clear that scientific investigators frequently find it of value to represent and to refer directly to aspects of the *process of constructing and testing models.* In other words, they typically make reference to a large body of concepts that possess interpretations in terms of the modeling process itself. Examples of such concepts include *dataset, database, equation, model, algorithm* and *project.* [4] We note that some of these concepts are related to the manner in which scientific research and the sharing of resources is organized. This, for example, is the case with the important concept of *project.*

The process by which modeling concepts of both classes are discovered, evaluated and applied lies at the core of scientific modeling activities. It is the manner in which such concepts are represented, however, that determines their real value in the modeling process. As we emphasize in our definition, a concept *cannot be defined in any meaningful manner apart from its representation in some language.* In particular, the scientific value of any set of modeling concepts is usually related very closely to the nature of the representation of the concepts. While there are frequently several "equivalent" representations of a given set of concepts, some representations contain more information about the entity being represented than other representations, while some representations make it easier to infer properties of the entity being represented than do other representations. Hence we may construct the following

Definition. Representations containing extractable information about the entities that they represent are termed *informative representations.* Representations that contain no extractable information about the entities that they represent are termed *nominal representations.* □

The power and success of symbolic modeling activities in science frequently involves the discovery, evaluation and application of "clever" informative representations, such as the *place-value representation* for numbers. We note, however, that nominal representations also possess a valuable function, since they permit scientists to make efficient reference to arbitrarily complex entities.

While it is not possible to provide a succinct, universally-acceptable characterization of scientific

[4]When reference is made to methodological concepts during the process of constructing and evaluating models of phenomena, they are typically accorded the same "first class status" that is more often associated with the phenomenological concepts. In other words, they may be directly represented and manipulated.

modeling, for the purposes of defining a CMS we employ the following general

Definition. Scientific modeling activities are processes in which scientists
• construct, evaluate and apply collections of phenomenological **and** methodological concepts;
• construct *specific instances* of the representations associated with the concepts and carry out sequences of inferential operations on collections of these instances;
• interpret and evaluate the representations that arise as a result of the inferential processes;
• construct, interpret and evaluate statements in some modeling language concerning the concepts and their instances. □

Furthermore, if one examines the nature of the modeling concepts in nearly any area of science, it is typically the case that they are organized in terms of a structure that possesses a strong hierarchical component. The central issue for designing a CMS, we believe, is to find an appropriate and comprehensive characterization of such activities. We now turn to this central issue.

3.1.1 Concepts, Representational Structures and Scientific Activity

As defined above, modeling concepts involve representations of classes of *identifiable entities*; representations of processes that operate on such entities; and representations of relationships among the various sets of entities and processes, together with some mapping from the representations to the entities, processes and relationships. We have found it of great value, as have other researchers [Req80], to refine our definition of the representational component $\mathcal{S}$ of a concept $[\mathcal{W},\ \mathcal{S},\ \mu]$ in terms of the representations of the entities, the processes and the relationships. In particular, we have the following fundamental

Definition. A *representational structure* (or R-structure) for the set of representations $\mathcal{S}$ of a concept $[\mathcal{W},\ \mathcal{S},\ \mu]$ is a quadruple $[\mathcal{D},\ \mathcal{T},\ \mathcal{R},\ \mathcal{I}]$ in which $\mathcal{D}$, or the *representational domain* (R-domain) of the R-structure, is the set *representations* of all entities in $\mathcal{W}$; $\mathcal{T}$, or the *transformations* of the R-structure, is a set of transformations $\mathcal{T}$ that operate on the representations in $\mathcal{D}$ and that are representations of processes in $\mathcal{W}$ that operate on the entities of $\mathcal{W}$; $\mathcal{R}$, or the *relationships* of the R-structure, is a set of relationships defined over the representations in $\mathcal{D}$ that are representations of relationships in $\mathcal{W}$; $\mathcal{I}$ is a finite set of representations of *particular instances* of representations in $\mathcal{D}$ that are given in explicit form for convenience. □

Since an R-domain $\mathcal{D}$ may contain an infinite number of entities, the representations of the entities are typically given in some implicit form. The transformations $\mathcal{T}$ may map representations in $\mathcal{D}$ into representations in $\mathcal{D}$ or in other domains. Finally, we may associate with the R-domain of an R-structure a (possibly empty) set of constraints on their values that are expressed in terms of transformations and relations. about the representations in $\mathcal{D}$.

Example. We may specify the representation $\mathcal{S}$ for the modeling concept *Polygon* in terms of some R-structure **Polygons** using an R-domain in which any polygon is represented as a sequence of points $< p_1, ..., p_n >$, in which the points p_i are from the R-domain of **Points**. The transformations on these representations would include, for example, *area*, which maps a representation of a polygon into a representation of its area. The relationships on these representations might include *intersects*, which is true for two polygons if they intersect. In order to make this representation of the concept *Polygon* reasonable, it is necessary to associate with the R-structure expressions representing constraints, such as as $p_1 = p_n$ and *no two edges of the boundary intersect, except at their end points.* □

It is of value to distinguish among classes of R-structures on the basis of the degree of "abstractness" of the representations that they contain.

Definition. A *concrete R-structure* possesses an R-domain $\mathcal{D}$ of informative representations. An *abstract R-structures* possesses an R-domain $\mathcal{D}$ of nominal representations. □

In general, the modeling activities of scientists are focused on the discovery and application of concrete R-structures, since the manipulation of the representations in such structures provide interpretable information about the phenomena of interest. While it not possible to infer useful information about some entity on the basis of the representations in the R-domain of an abstract R-structure, such representations have both a scientific and a computational value. [5] We first describe in greater detail that nature of concrete R-structures; we then describe abstract R-structures; and we then provide scientific and computational reasons that justify the distinction.

3.1.2 Concrete R-structures

The main components and attributes of R-structures include the R-structure name, the R-domain $\mathcal{D}$, the set of transformations $\mathcal{T}$, the set of relationships $\mathcal{R}$, the set of constraints on R-domain elements, and the set of instances $\mathcal{I}$.

Definition. The *name* of a concrete R-structure is an informative or a nominal representation that may be used to refer to a given R-structure as a single entity. □

R-structure names are typically *mandatory* for all but a special class of R-structures. Since concrete R-structures are the representations of modeling concepts, it is important that scientific investigators have the ability to refer to them as *single entities*. This idea is in accordance with our principle that concepts relating to the *process of modeling*, as well as concepts relating to the phenomena, be given a first class status. We permit the naming of R-structures to be optional only in cases in which they take the form $\{\mathcal{D},\ \emptyset,\ \emptyset,\ \emptyset\}$ and in which $\mathcal{D}$ is constructed only by the application of *aggregate constructors* to predefined R-domain representations. As an example, we note that it is frequently convenient in modeling exercises to construct aggregates of previously defined elements for which a semantics defined in terms of a special collection of transformations and relationships is irrelevant, as for example, in the case of many "dataset" entities.

Definition. The R-domain of a concrete R-structure is a set of informative representations for each of the entities of an associated concept. Representations of a new R-domain may be constructed by applying *aggregate constructors* to finite sets of elements that may include previously-defined R-structures, the components of previously-defined R-structures, or the elements of such components. If the elements are representations from previously-defined R-domains, new R-domain representations may be constructed by applying *value constraints* to the old representations. □

Since many R-domains contain infinite numbers of representations, we typically provide a representation for R-domain elements in terms of *procedures for generating any representation in the domain*. We note, for example, that the above definition permits us to construct R-structures whose R-domains contain transformations from previously-defined R-structures. This exemplifies how "second class" elements in one R-structure may be endowed with a first class status in a new R-structure. We assume the existence of a finite set of *aggregate constructors* that typically include the *identity constructor* I; the *set constructor* {}; the *tuple constructor* []; the *sequence constructor* <>; and the finite *function constructor* $\mathcal{F}$. [6] The *constraints* that may be placed on the the components of existing R-domain representations are expressed in terms of the transformations

[5] We may view abstract R-structures as abstractions over a set of equivalent concrete R-structures.

[6] It appears to be the case that we generally need a relatively small number of constructors.

$\mathcal{T}$ and relationships $\mathcal{R}$ that are defined on the R-domain elements that are used in the definition of new R-domain elements. Clearly, R-domains defined in terms of constraints will be subsets of the unconstrained R-domains, and will form part of an inheritance hierarchy. The inductive approach to the construction of new R-domains requires that we assume the existence of an initial and well-specified set of R-structures.

Definition. A *primitive* R-structure is one whose R-domain elements are not defined in terms of the elements of existing R-domains. A *derived* R-structure is one whose R-domain elements are defined in terms the elements of previously defined R-domains. □

Typical examples of primitive R-structures include *integers, floats, booleans, strings.*

There may be more than one representation in an R-domain that corresponds to an entity in the domain of application. In such cases, one may map such elements into their *equivalence classes.* There may be equivalent R-domains, for a given set of entities, since it is frequently possible to construct different representations of the same set of entities. Such representations may be equivalent in the sense that there exist isomorphic mappings between the elements of equivalent R-domains [7]. A typical reason for constructing different but equivalent representations is that it may be more convenient to compute certain properties of the phenomena using one representation rather than another.

Example. The R-structure **Polygons** that was defined above is clearly a derived R-structure based on an R-structure for Points, which in turn may be based on other R-structures (such as the primitive R-structure **Floats**). We may construct an R-structure **Convex-polygons** by adding a convexity constraint to the representations in the R-domain of **Polygons**. In the case of the point-based concrete representation in **Polygons**, it is clear that a cyclic permutation of the point sequence that represents a polygon will also represent the same polygon. In the case of *Polygons* we also may construct additional R-structures in which we employ sequences of **Linesegments** or sets of **Half-planes** to represent a polygon. □

Definition. The set of transformations $\mathcal{T}$ of some R-structure map the representations in the R-domain $\mathcal{D}$ into the representations of previously defined R-domains. □

To a significant degree it is the set of transformations and relationships involving R-domain elements that provides semantics to R-structures, since very different phenomenological entities may possess structurally equivalent representations in the associated R-domain. Apart from general classes of transformations, there are some important special classes that include

- a set of transformations defined on the representations of a given R-domain that map representations that are equivalent (in the sense that they represent the same entity) into equivalence classes.
- A set of transformations that map the equivalence classes of representations in one concrete R-domain into the corresponding equivalence classes in some other concrete R-domain that is itself equivalent to the first R-domain in the sense that it contains (different) representations of exactly the same set of entities.
- A set of transformations that map the equivalence classes of representations onto the nominal representations of entities in the associated abstract R-domain.

Definition. The set of relationships $\mathcal{R}$ associated with an R-structure are a set of boolean-valued transformations defined over the R-domain of the R-structure and the R-domain of previously-defined R-structures. □

Transformations and relationships are supported for the expressive convenience of the user. The

[7] Equivalent entities represent the same entity in the phenomenological world.

transformation that maps equivalent representations in some R-domain into equivalence classes may be also be viewed as an *equality relationship* that is defined over pairs of representations in some domain. The set of transformations and relationships defined over existing R-domains provide the basis for the definition of new R-domains.

Definition. Constraints that are defined on the values of the components of R-domain representations are expressions involving only predefined transformations and relations. □

Example. The transformations such as *area, centroid, perimeter* that are associated with **Polygons** differentiate the representation $< p_1, \; ...p_n >$ of a polygon from structurally equivalent representations for a chain of linesegments. Relations defined over **Polygons** may include *intersects*, *convex* and *Point_in_polygon*. □

Definition. The set of instances $\mathcal{I}$ of R-domain elements associated with an R-structure are a finite set of *explicit* representations. □

The set $\mathcal{I}$ is provided for convenience, since the specification of the elements of an R-domain, as described above, may provide no more than a prescription for generating R-domain elements.

Definition. An instance of an R-domain representation in $\mathcal{I}$ is termed a *dataset*. The set of all datasets $\cup\mathcal{I}$ that are associated with a collection of R-structures that constitute a CMS is termed the *database* of the CMS. □

3.1.3 Abstract R-structures and their Value

For every collection of equivalent R-structures, we assume the existence of an abstract R-structure whose components include:
• A name, with the same naming convention as for concrete R-domains.
• An R-domain of nominal representations (sometimes referred to as *object identifiers* or oids) that may be mapped onto the entities of some domain of phenomena.
• A set of signatures for all of the transformations that are defined on the representations in the associated concrete R-domains.
• A set of signatures for all the relationships that are defined on the elements of the associated concrete R-domains.
• A set of signatures for all the constraints that are defined on representations in the associated set of equivalent concrete R-domains.
• A set of oids for specific instances of the elements of the R-domain.
Since the representations of entities in abstract R-domains are nominal, there is no need for more than one representation for any entity, and there is no need to have more than one abstract R-structure for any set of equivalent concrete R-structures.

Although abstract R-structures are largely concerned with nominal information, they may in principle be given the same expressive power as concrete R-structures. [8] Abstract R-domains are of value in relation to the process of scientific modeling and its support in terms of CMS for at least four reasons. First, it is clear that the ability to give a "name" to any particular entity is of value in scientific research. [9] Second, abstract R-domains provide a simple, "representation-free" method for formulating definitions of the representations in new concrete R-domains. Third, they provide a simple mechanism for specifying the inheritance of concrete representational structures,

[8] One may, for example, define finite transformations and relations in terms of sets of tuples of oids.

[9] Note, for example, all the names that are given to numerous special classes of polygons, such as triangles, quadrilaterals, parallelograms, and squares.

transformations, relationships and constraints from super-domains to subdomains. For example, if an R-domain is defined in terms of the entities of some other R-domain to which constraints have been applied, as in the case of **Convex_polygons**, the sub-domain should inherit all of the concrete representational structures, all of the transformation, all of the relationships and all of the constraints that are associated with the super-domain. Since it is a single, abstract representative of a class of equivalent concrete R-structures, an abstract R-structure provides a convenient mechanism for expressing such inheritance. Fourth, if one views the nominal representations in an abstract R-domain in terms of globally unique *oids*, the representations serve a useful computational function in the sense of object oriented systems, and facilitate the management of efficient computation.

3.1.4 Scientific Modeling Activities and Inductively-defined Collections of R-structures

It is relatively straightforward to represent the symbolic modeling activities in any domain of science in terms of an inductively-constructed collection of abstract and concrete R-structures. The modular nature of R-structures and the simple inductive manner in which new R-structures may be constructed from previously defined R-structures makes it easy to extend some collection of R-structures. As we note below, a computational modeling language (CML) with a remarkably simple syntax is sufficient for creating R-structures at any "level" within the inductively-defined collection of R-structures. Since the tools required to construct and employ such a collection are extremely simple from a conceptual point of view, the notion of a collection of abstract and concrete R-domains provides an good basis for a CMS.

It is possible to represent the set of symbolic expressions $\mathcal{S}$ of any "reasonable" modeling concept $[\mathcal{W}, \mathcal{S}, \mu]$ in terms of R-structures, as we note in Appendix I [10]. Since we may construct R-structures for the symbolic representations of both phenomenological concepts and modeling concepts, and since any symbolic model is no more than some finite subset of symbolic representations from some collection of modeling concepts, R-structures are "complete" in terms of their expressiveness with respect to the representation of scientific modeling activities. In Figure 3 we show a fragment of an admissible collection of R-structures that may be constructed to support representations of the concept of polygon.

We may, for example, construct R-structures whose R-domains contain elements that represent
- primitive "abstract" entities, such as integers;
- R-structures that represent complex abstract entities, such as complicated geometrical figures;
- R-structures that contain "empirically-derived" entities, such as datasets and images that are formatted in a certain manner;
- R-structures that contain representations of transformations;
- R-structures that contain abstract mathematical models;
- R-structures whose R-domain elements are representations of modeling processes that have been executed using the representations and transformations of other R-structures;
- R-structures whose R-domain elements are representations of a subset of (predefined) R-structures.

As we note below, we we may represent the concept of a *project* in terms of such an R-structure. Hence R-structures may be constructed that permit scientists to represent any *reasonable* entity that is useful in their modeling as a first class element in the CMS.

With no change to the definition of R-structure, we may easily construct various "classes" of

[10] This process is limited only by the problems associated with certain classical paradoxes.

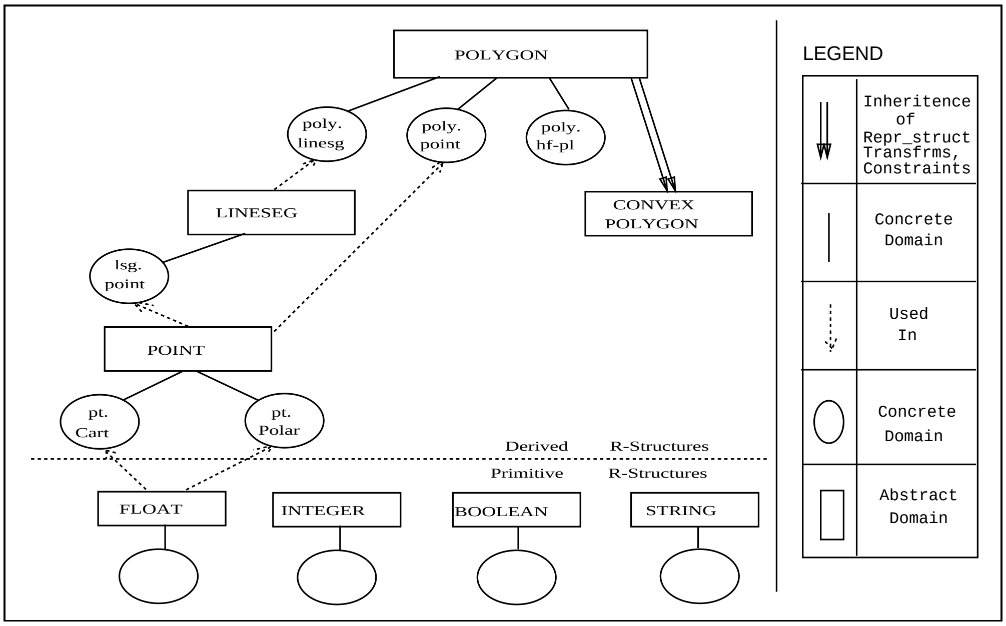

Figure 3: A fragment of an admissible collection of R-structures

R-structures that capture many of the semantically-important distinctions that scientists make between various classes of concepts. We now briefly indicate some of these distinctions and provide a more detailed description in Appendix II.
• *Empirically* versus *theoretically determined entities and relationships*: is a distinction in which the the first part refers to symbolic expressions that are largely determined by observation and measurement, while the the second part refers to those that are largely determined as the outcome of inference procedures applied to more fundamental symbolic expressions.
• *Methodological* versus *phenomenological entities and relationships*: is a distinction based on the notion that methodological R-structures have an interpretation in terms of the entities and relationships of the modeling process itself whereas phenomenologic R-structures possess an interpretation in terms of phenomena that are being modeled.
• *Explicit* versus *implicit representations of entities and relationships*: is a distinction based on whether the representation of some entity or relationship is direct or indirect. In the latter case, additional computation is required to obtain a representation of the desired entity.
• *Exact* versus *approximate representations of entities and relationships*: is is distinction based on the trade-off between the amount of computation that is required to obtain representations of the properties of the entities and the degree to which there is any error in the representations of these properties.
• *Visually-meaningful* versus *non-visually-meaningful representations*: is a distinction based on the trade-off between representations that are efficient with respect to the specification of the computation of properties and representations that are expressive with respect to the human visual/cognitive system.

It is also easy to define high-level methodological R-structures that are particularly expressive with respect to representing the *process* by which the space of possible R-structures is explored. Important examples of such R-structures include representations for the concept *modeling schemas* and for the concept of *project*. There are many situations in which it is useful to have a modifiable representation of some subsequence of modeling operations involving R-domain elements and representations, since the process of iterative model modification often requires that many complex sequences of operations be repeated frequently with large numbers of modifications. Hence it seems natural to provide support for creating, storing and manipulating representations of such sequences of operations that may be referred to and and manipulated in various ways as first class elements.

Given a computational modeling language CML with commands such as *create, access, apply, store, delete* that apply to R-structures and their components, it is useful to have the following

Definition. The R-structure, **modeling schemas**, incorporates
• An R-domain $\mathcal{D}$ whose elements are appropriate *sequences* of command templates $< e_1,\ e_2, ... >$. Each command template is of the form $< CML\_command, arg_1,\ arg_2, ... >$ where the typed arguments may be replaced by appropriate instances of appropriate entities, which may include R-structures, the components of R-structures, or elements from the components of R-structures, such as R-domain elements or transformations.
• Transformations $\mathcal{T}$ that may include operators on the R-domains elements such as *concatenate, run, debug.*
• A set of instances $\mathcal{I}$ of modeling schemas containing specific sequences of operators and arguments.
□

The concept of a *project* is currently the highest level concept that we represent in a CMS. As we note below, it permits us to define easily the idea of subprojects and the interrelationships between related projects in terms of the sharing of various R-structures and their component elements. In particular, within the CMS framework a *project* is defined simply as subset of the available R-structures, together with appropriate computational systems support, that is in some sense sufficient for the iterative modeling and database activities in some domain of scientific application.

Definition. The R-structure, **projects**, incorporates
• An R-domain $\mathcal{D}$ whose elements are tuples $[e_1,\ e_2, ...]$ in which e_1 is a any admissible subset of R-structures except *projects*; e_2 is a subset of instances of modeling schemas; and other elements represent other aspects of a project, such as the researchers of the project, the problems under study, and the duration of the project.
• The relations $\mathcal{R}$ of the project contain a *subproject* relation on pairs of projects, which in essence defines a partial ordering over the set of all projects.
• The set of instances $\mathcal{I}$ of projects contains explicit representations of all projects. □

In a sense, one may view such an R-structure as defining the access of groups and subgroups of researchers to various subsets of the existing set of R-structures.

3.1.5 A Model of Scientific Modeling Activity

Using the notion of R-structure, one may view the process of constructing symbolic models of phenomena as one in which an enormous space of potential R-structures is incrementally explored and evaluated in terms of the interpretations provided with the associated set of concepts. As a result of such search, the most useful R-structures are maintained for modeling purposes, while those that are found to have less value are abandoned. [11] Hence we may characterize the component of scientific modeling activity that is supportable in a simple manner by a CMS in terms of the following

Definition. In terms of R-structures, scientific modeling activities are processes in which scientists

[11] It is clear that scientific knowledge is augmented by an institutional memory of both the more useful and the less useful structures.

• construct, evaluate and apply collections of R-structures that are of value in relation to modeling phenomena in specific domains of application **and** in relation to modeling the process of modeling itself;
• construct *specific instances* of R-domain elements ($\mathcal{I}$) and apply sequences of specific transformations to sets of instances of R-domain elements;
• construct, evaluate and apply various statements about representations of domain elements, transformations, and relations in an R-structure. □

The evaluative component of the process involves the interpretive components $\mathcal{W}$ and μ of the various concepts that are associated with the R-structures. It does not appear at this time that this component can be supported simply and in a non-controversial manner in terms of a CMS.

We are now in a position to define the concept of a CMS, as in the following

Definition. A CMS is an admissible, extensible, and inductively defined collection of R-structures that is appropriately expressive with respect to some domain of scientific investigation; that includes an R-structure for projects; and that is provided with adequate and transparent computational support. □

4 THE HYDROLOGICAL EXAMPLE REPRESENTED IN TERMS OF R-STRUCTURES

We illustrate the preceding ideas by showing how the hydrological modeling problem may be decomposed and represented in terms of both phenomenological and methodological R-structures. In constructing an appropriate collection of R-structures for the computational modeling of a given set of phenomena, we note that the set of R-structures may involve four distinct subsets:
• a set of primitive R-structures that may include **Integers**, **Floats**, **Booleans**, and **Strings**;
• a set of R-structures that are generally accepted as "standard" in many applications;
• a set of R-structures that have been "imported" from other projects;
• a set of R-structures that are constructed inductively from the total set of available R-structures.
In relation to standard R-structures, we note that there are, for example, various *data transfer standards* that may serve as a basis for specific concrete R-structures. An excellent example is the recently adopted Federal Spatial Data Transfer Standard (SDTS) [SDT92] that provides logical specifications and data structures for an important subset of spatial data types.

4.1 R-structures for the Hydrological Application

In Figure 4, we illustrate a small fragment of a collection of (abstract) R-structures that may be employed representing the hydrologic flow problem.

Examples of supporting R-structures include:
• those representing geometrical entities and associated operations, such as **Points**,**Line_segments** and **Rectangular_Grids**, and any of the entities, transformations and relationships of geometry that are of value in the construction of various classes of models. We may, for example, represent the entities in the R-domain of **Points** as tuples of **Floats** $[f_1,\ f_2]$; entities in the R-domain of **Line_segments** as tuples of **Points** $[p_1,\ p_2]$; and entities in the R-domain of **Rectangular_point_grids** as sets of **points** that are subject to the constraint that they form a complete, rectangular array.
• R-structures that may employed in representing simple temporal concepts, such as **Time** and

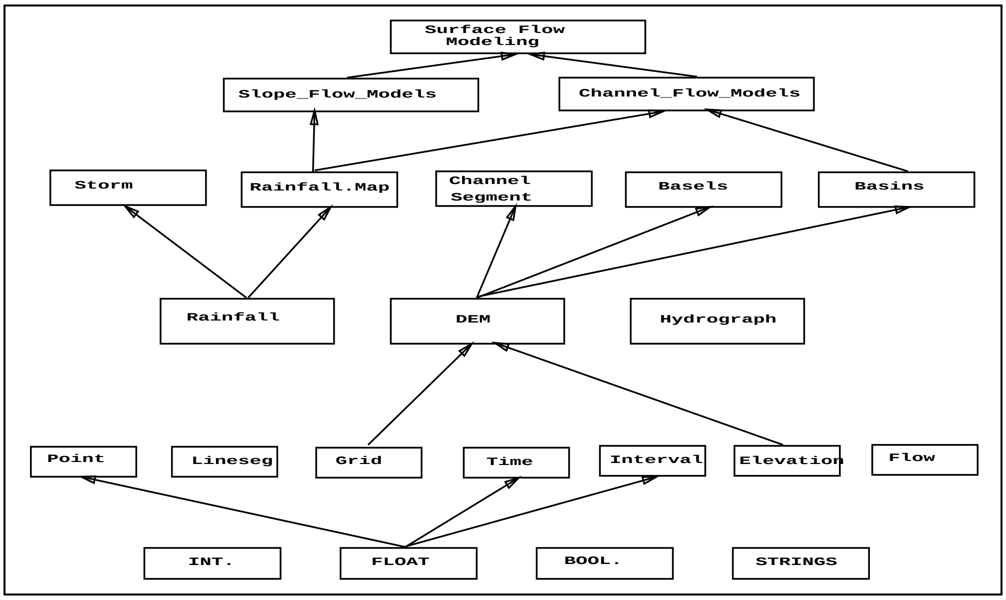

Figure 4: A Fragment of an Admissible Collection of R-structures for the Hydrological Modeling Problem

Time_intervals. The former may be represented simply as a member of the R-domain of **Floats** (ie $[f]$).
• R-structures, supporting simple and basic scientific concepts, such as **elevations**, represented in terms of a single element from the R-domain of **Floats**; **Flows**, represented in terms of a single element from the R-domain of **Floats** $[f]$; and **Flow_vectors**, represented in terms of a tuple of two representations from the R-domain of the R-structure **Flows**, namely $[f_1,\ f_2]$, and **Surface_flows**.

Higher-level R-structures include:
• empirical R-structures, such as **Rainfall_obs, DEMs, Hydrograph.obs**
• empirically-derived R-structures, such as **Channel_segs, Basels, Basins** from DEMs, **Rainfall_maps, Storms** from **Rainfall**
• high-level R-structures for modeling phenomena in relatively abstract terms,**Slope_flow_models**, **Channel_flow_models**, **Surface_flow_models**.

The elements of the R-domain of the structure **DEMs** may be represented in terms of sets of tuples of **Points** and **Elevations**: $\{[p,\ e]\}$. One may transform these basic representations of ground surfaces into various other representations of complex drainage surfaces. In particular, one may derive R-structures such as **Channel_segs**, **Basels**, and **Basins** which are relatively complex entities. The elements of the R-domain of the structure **Rainfalls** may be represented as tuples of values of rainfall as measured at a certain location and a certain time: $\{[p,\ t,\ r]\}$. We may associate a variety of transformations with this structure, including *interpolation* transformations that take a rainfall dataset as input and produce a set of estimated rainfalls at each point of some (rectangular) grid.

The R-domain of the explicit, approximate R-structure **Surface_flows** contains representations of flows of water over some surface. We may assume a representation of the form: $\{[p,\ t,\ [f_1,\ f_2]]\}$ in which p is a representation from the R-structure **Points**, t is a representation from the R-structure **Times**, and $[f_1,\ f_2]$ is a representation from the R-structure **Surface_flow_vectors**. A typical transformation on the elements of this R-domain would be: $compute\_hydrograph$: **Points** $\times$

Surface_flows $\Rightarrow$ Hydrographs.

The implicit, approximate R-structure **Surface_flow_models** may be viewed as possessing an R-domain in which various members of a class of discrete modeling equations are represented. These modeling equations are intended to be employed in computing a representation of:

- the depth and direction of the flow of water over land surface elements, given a "continuous" land surface, an initial surface flow and an input of rainfall;
- and the resulting flow rate of water in channels that occurs at the base of the land surface elements.

For example, we may assume that the representation of any given set of "flow equations" in the R-domain take the form: $[\alpha_{1,1}, ..., \alpha_{1,n_1}, ..., \alpha_{m,1}, ..., \alpha_{m,n_m}], ..., [\Delta x, \Delta y, \Delta t, ...]$ in which the $\alpha_{i,j}$ represent the coefficients that may be specified by the user in order to determine the particular flow model to use, while the remaining parameters, such as Δx may be specified by the modeler in order to determine the precise nature of any solution procedure that is applied to the model. The transformations associated with this domain include **solve_method_i**, which takes as input an element of the R-domain of the R-structure **Surface_flow_models**, an element of the R-domain of the R-structure **Drainage_basins** (which contains representation of the land surface over which the water flows), an element of the R-domain of the R-structure **Rainstorms**, and an element of the R-domain of the R-structure **Surface_flows** that is intended to represent the initial flow pattern over the surface. In the sense that the R-domain of **Surface_flow_models** belongs to an *implicit* R-structure, it should be noted that the full representation of a particular set of flow equations is partitioned between the domain element representation and some transformation that is defined to operate on such an element. These input elements are transformed into a sequence of elements of the R-domain **Surface_flows** that represent *explicitly* the flow over the surface for a corresponding sequence of times. In summary, we may view the process of applying the **solve_method_i** transformation as the "running" of a particular "model" of flow (namely the equations) with a given set of initial conditions, boundary conditions, and forcing function.

At any given stage in the process of modeling one may envisage the scientist as involved in a variety of operations, including:

- the construction and evaluation of particular R-structures. and in particular, the construction of representational domains $\mathcal{D}$, representations of transformation ($\mathcal{T}$) and representations of relations between between R-domain elements ($\mathcal{R}$).
- Construct *specific instances* of R-domain elements ($\mathcal{I}$) and apply sequences of specific transformations to sets of instances of R-domain elements.
- Construct, evaluate and apply various statements about the representations of domain elements, transformations, and relations in an R-structure, and associate them with the R-structure.

For example, the sequence of operations in Figure 1 may occur at some time in the modeling process. We note that this sequence may be viewed as a *model* of the scientists modeling activities and that a representation of this sequence that may be created, stored and manipulated. In general, we may create R-structures whose R-domains consist of elements that represent a large space of sequences. Specific instances of such sequences may be stored, accessed and modified in ways that further the state of knowledge concerning the phenomenon. The scientist may, for example, wish to modify some of the data inputs (such as Dem for a new area), or modify some of the parameters in the model of surface flow.

It is clear that in constructing such R-structures we must have the capability of representing the operations that are carried out by the scientist as well as the R-domains, transformations and relationships. As noted above, we may construct R-structures whose R-domains contain representations of specific sequences of operations (*modeling schemas*).

5 A High-Level Computational Modeling Language

In this section we describe a simple, largely declarative language for expressing a wide class of scientific modeling and database operations. The language, which we term Computational Modeling Language or CML, is intended to express such operations at the conceptual level of the scientific investigator. The design of the language is based on the concept of R-structures, and permits investigators to express easily and naturally most of the operations that are employed in the iterative development of their models, while hiding computational issues that are irrelevant to the scientific goals of the investigators. [12] We first describe a set of basic operations for manipulating R-structures and then illustrate the creation of useful methodological concepts, such as modeling schemas and projects, with the use of CML.

The primary functionalities of CML include

• the definition, creation, manipulation and storage of new R-structures and their constituent parts, including the specification of R-domain elements in $\mathcal{D}$, the transformations in $\mathcal{T}$ and the relations in $\mathcal{R}$, and explicitly-represented instances of R-domain elements in $\mathcal{I}$, as well as the creation of constraints;

• item the application of transformations to R-domain elements in general and to R-domain instances in particular;

• the search for specific R-domain elements and transformations that satisfy appropriate constraints.

In order to provide these functionalities, CML includes a small set of simple commands, which include *create*, *delete*, *modify*, *access* and *store* (R-structures, R-domains, transformations, relationships, and instances) and *apply* (for applying transformations to R-domain elements.) We note in passing that the commands associated with the first two functionalities derive from the theoretical foundations of a CMS described in Appendix I, while the commands of the third functionality derive from the need to store and retrieve representations on memory devices.

5.1 Primitive Operations in CML

The *create* command in CML permits the construction of abstract and concrete R-structures as well as their components. According to the definitions of abstract and concrete R-structures, a concrete R-structure in a CMS must be associated to an *abstract* R-structure. Either the user or the system must establish this relation. [13] In general, if the user attempts to create a concrete R-structure that has no corresponding abstract R-structure the system will create it. An advantage

[12] It is intended that CML support the majority of scientific required in terms of a very simple functionality and syntax.

[13] It seems pointless to construct an abstract R-structure unless at least one corresponding concrete R-structure has been constructed.

of this approach is that the system can then maintain consistency in the inheritance structure. In what follows we assume this consistency property, regardless of whether it is maintained by the user or the system.

In creating either an abstract or a concrete R-structure, it is necessary to associate the appropriate R-domains, transformations, relations and instances with the R-structure. In the case of abstract R-structures, it is necessary to specify a set of *super* domains, that is R-structures from which this R-structure will inherit concrete representations, transformations and constraints. In particular, the R-domain of a super R-structure *contains* the R-domain of any sub R-structure. In the case of the concrete R-structures it is necessary to indicate the corresponding abstract R-structure and the constraints on the R-domain representations. For the abstract R-structure, we have:

```
CREATE ABSTRACT R-STRUCTURE  DEM
  WHERE SUPER R-STRUCTURES { Rectangular_Grid_Maps }
        R-DOMAIN            r-domain(DEM)
        TRANSFORMATIONS     transformations(DEM)
        RELATIONS           relations(DEM)
        INSTANCES           instances(DEM)
```

Abstract R-structures provide "external" specifications of concrete R-structures similar in spirit to *abstract data types* [BDMN73]. An abstract R-structure may be instantiated by more than one concrete R-structure. For a concrete R-structure, we have:

```
CREATE CONCRETE R-STRUCTURE  DEM.peg
  WHERE R-DOMAIN =      r-domain(DEM.peg)
        TRANSFORMATIONS transformations(DEM.peg)
        RELATIONS       relations(DEM.peg)
        INSTANCES       instances(DEM.peg)
```

(where *peg* represents *point_elevation_grid*). In both case, a set of related expressions may also be specified.

Definitions must now be provided for the components of the R-structures. For abstract domains, the user may "name" the elements using values of type **string**. Although these names play the same role of object identifiers, this option provides more flexibility and clarity. Implementation wise, CML stores the element names as a default implicit attribute (invisible to the user). The names may be attached to the elements when they are created or explicitly by command. For example, if `Y` holds an identifier of a **DEM** element, the following creates a new name `Manaus` for the element.

```
CREATE  Manaus = Y IN r-domain(DEM)
```

Note that while each element in an R-structure can have 0, 1, or more names, they must be unique in an R-structure and all its substructures, i.e., consistent with the inheritance hierarchy.

The R-domain of a concrete R-structure $\mathcal{S}$ specifies the set of representations of all the entities that it represents. This specification involves a structural component and a "value" component that is specified in terms of a set of constraints on the subcomponents in the structural representation. The following example illustrates the creation of the concrete domain of the R-structure **DEM.peg** in which elevation data is represented by a set of **point**, **real** pairs:

```
CREATE DEFAULT r-domain(DEM.peg)
  WHERE ABSTRACT R-STRUCTURE = DEM
        STRUCTURE = [name:string, resolution:integer,
                     location:[L1:point, L2:point, L3:point, L4:point],
                     P_E:set of [Location:point, Elevation:real]]
        CONSTRAINTS = ...
```

In the structural component of the specification, `[name:string,...]` denote the tuple construct and `name` is a field name and **string** the type for the field.

It is also possible to create concrete R-structures whose R-domain elements have a representation that is hidden from (or "transparent" to) CMS. We call such R-structures *virtual* concrete R-structures. In terms of implementation, they are treated as black boxes and are managed by the data manager as plain (binary or ASCII) files. Such virtual R-structures are extremely useful in allowing operations from other software modules to be incorporated and used with the CMS environment.

As an example, the following creates a virtual concrete R-domain for the R-structure **DEM** in a format that is interpretable by MATLAB:

```
CREATE r-domain(DEM.matlab_format)
  WHERE ABSTRACT R-STRUCTURE = DEM
        STRUCTURE = FILE
        SUPER DOMAIN = NONE
```

It is also possible to introduce transformations from MATLAB, such as *display_dem*, into the set of transformations associated with the R-structure DEM. Hence one may create a uniform view of integrated software packages.

In specifying the set of transformations for an abstract R-structure, it is necessary to provide a set of transformation signatures (i.e. their names and input and output types):

```
CREATE  transformations(DEM)
  WHERE TRANSFORMATION DEM.union: set of DEM -> DEM
        TRANSFORMATION DEM.spatial_projection: DEM -> polygon
        TRANSFORMATION DEM.compute_slope: DEM -> [name:string,...]
        TRANSFORMATION DEM.max_height: DEM -> real
```

In the above example, four transformations are defined on the R-structure **DEM**. The transformation **DEM**.*union*, for example, maps a set of DEM elements and merges them into a single DEM element (stored in the structure of the *default* or *canonical* concrete domain, discussed below) and produces a single DEM element. If the element does not already exist in DEM, the new element will be inserted.

The set of transformations and relationships that are associated with a concrete R-structure are represented in terms of the executable code for the transformations and relationships. As we note below in greater detail, CML currently allows transformations and relationships to be written in an imperative style of CML with local variables; in other programming languages (such as C, FORTRAN, etc.); or in languages provided by other software tools. An important relationship that must be associated with concrete R-structures in applicable cases is related to the equality predicate "$\approx_C$", that is used to indicate when two representations in a given concrete R-domain represent the same entity. An important set of transformations that must be defined on the R-domains of *equivalent* R-structures map equivalent sets of representations in the two R-domains onto each other, since it is required that the R-domains of equivalent R-structures be *isomorphic* in the following sense. Let $\mathcal{D}_1$, $\mathcal{D}_2$ be two different concrete R-domains of an abstract R-domain $\mathcal{D}$.

- Each representation e in $\mathcal{D}_1$ can be mapped by $f_{\mathcal{D}_1 \to \mathcal{D}_2}$ to an "equivalent" representation $f_{\mathcal{D}_1 \to \mathcal{D}_2}(e)$ in $\mathcal{D}_2$ and when the latter is mapped back, the two elements are equal: $e \approx_{\mathcal{D}_1} f_{\mathcal{D}_2 \to \mathcal{D}_1} \circ f_{\mathcal{D}_1 \to \mathcal{D}_2}(e)$. That is, when restricting the equivalent classes of elements in $\mathcal{D}_i$ with respect to $\approx_{\mathcal{D}_i}$, $f_{\mathcal{D}_i \to \mathcal{D}_j}$, is an isomorphism.
- The implementations of all transformations must be defined on the equivalent classes $\approx$ and isomorphic under the $f_{\mathcal{D}_i \to \mathcal{D}_j}$ mappings.

In CML, the user or the database administrator is expected to define the isomorphisms. The syntax for using the isomorphisms $f_{\mathcal{D}' \to \mathcal{D}}$ is a function term $\mathcal{D}$, where $\mathcal{D}, \mathcal{D}'$ are two concrete domains. For instance, if e is the identifier of an object in an abstract domain $\mathcal{D}$ and $\mathcal{D}_1$ is a concrete domain of $\mathcal{D}$, then $\mathcal{D}_1(e)$ represents the structural value of the object in terms of the concrete domain $\mathcal{D}_1$.

In CML, each abstract domain has exactly one *default* (or *canonical*) concrete domain. The default domain is used whenever the concrete domain is neither explicitly specified nor inferable. This representation is used to avoid nondeterminism and to make implementation easier. Elements that belong to an abstract domain may be stored in different concrete domains but in CML the default concrete domain stores all elements.

It should also be noted that difficulties in implementing inheritance arise because of the multiple concrete R-structures that are permitted for each abstract R-structure. For example, it may not be appropriate for **DEM.peg** to inherit from some other concrete R-structure of **Rectangular_Grid_Map** in cases in which the structures are incompatible. For this reason, users must designate which specific concrete R-domains of the super abstract R-domains may be used for inheritance; furthermore, the structures must be compatible under the usual subtyping rules [Car84, CW85, Car88].

Application of Transformations to and Accessing of R-domain Elements A key operation in CML is the application of transformations to elements from R-domains. Such applications may be expressed in CML in terms of the *apply* statement. Suppose the variable Y contains a set of DEM element identifiers. The command "`APPLY DEM.union TO Y`" results in a (new) element of type DEM; it also returns the identifier of the new element which can be stored in another variable to be used later.

The *apply* command has a large number of important applications which include the creation of explicit instances of representations (or *datasets* for storage in an R-structure.) The following example shows how we may create a new explicit instance of an R-domain element of the R-structure **DEM_SLOPES** using the transformation **DEM**.*compute_slope*:

```
CREATE element IN instances(DEM_SLOPES)
  WHERE VALUE = APPLY DEM.compute_slope TO Y
```

where Y was defined previously.

CML provides an important but simple command `access` for querying about R-structures, their four main components and elements of their components. We illustrate the use of this command with examples relating to the access of *datasets* (or explicit instances of R-domain elements.) Assume that the abstract R-structure **DEM** has already been defined and that *Manaus* is a variable holding a spatial object identifier. Using the predicate *intersect* on pure spatial objects, we can find all DEMs whose spatial projection overlaps the spatial projection of the *Manaus* object using the following query:

```
Y = ACCESS { X IN DEM
             WHERE DEM.spatial_projection(X) INTERSECT
                  DEM.spatial_projection(Manaus) }
```

We now present a slightly more complicated example. Suppose **Rainfall_Map** is an abstract domain representing measured rainfall data with a (default) concrete domain **Rainfall_Map.tpr** whose elements possess the representation {[*year*:**Int**, *month*:**Int**, *date*:**Int**, *hour*:**int**, *P*:**Points**, *rain*:**Floats**]}. Then, the following query extracts all rainfall data for the Manaus region within a particular period.

```
R = ACCESS { S IN Rainfall_Map
             FOR T IN Rainfall_Map.tpr(S)
             WHERE T.year = 1989 AND T.month = 1 AND T.date = 21
               AND 1 <= T.hour <= 12 AND T.P IN spatial_projection(Manaus) }
W = APPLY map TO Rainfall_Map.rain_extract, R
```

In the last statement, *map* applies the transformation *rain_extract* to each element in the set R and returns the union.

It should be noted that in the actual implementation of CML, it is not required that each transformation of an abstract domain be implemented by *each* concrete domain. Indeed, as shown in the above example, such implementational heterogeneity seems to be of advantage. However, translations between other concrete domains should be provided.

The Specification of Transformations We now consider in greater depth the issue of the creation of transformations. The transformations associated with CML may be classified according to whether they are defined in CML, in other programming languages, or with the use of other software tools. Transformations may be created in CML by the *create* command. A major advantage of creating transformations within the CML environment is that it is easy to aggregate previously defined transformations into new transformations. This is important in the construction of new R-structures, since the new R-domain elements frequently involve aggregate constructors. Hence transformations that are applicable to these elements will frequently involve, as components, transformations that apply to the R-domains to which the components of the element belong. Transformations that are not defined in CML are viewed by CML as "black boxes" or binary executables. We first consider how CML may be employed in defining transformations and then discuss how to incorporate the representation of transformation from other languages into the CMS framework.

Each transformation includes a name, a list of parameters and their corresponding types, the type of the output value, and a sequence of CML assignments defining the computation. The following is a simple example where a transformation *rain_extract* is defined. The function retrieves a certain subset of relevant rainfall measurements from any **Rainfall_Map** representation.

```
CREATE TRANSFORMATION rain_extract(yy:int,mm:int,dd:int,hh:int,area:polygon)
  RETURN rr:SET OF [year:int, month:int, day:int, hour:int, loc:point]
  BEGIN
  rr = ACCESS { t:[year:int, month:int, day:int, hour:int, loc:point]
              FOR o IN Rainfall_Map
              WHERE t IN TPR(o)
              AND t.year = yy AND T.month = mm AND T.date = dd
              AND t.hour = hh AND T.P IN area };
  END
```

Recall that **Rainfall_Map.tpr** is a concrete R-structure of **Rainfall_Map** which represents rainfall maps by sets of time, point, and rain measurement triples. It is easy to see that the above transformation can be easily extended to extract rainfall measurements for a period of time.

Local variables can be declared and used in the body of a function but their scopes are enclosed by *begin* and *end*. Similarly, new domains and elements can also be created with scopes within the function. Hence, they are not persistent automatically (the *store* command can be used to save the domain definitions and their elements.)

Suppose now the domain **Polygons** has a concrete domain represented by a sequence of points. Since only the order of the points matters in the representation, the same polygon may be represented by two distinct sequences which differ in the starting point. The following transformation is used in defining the equality relationship on polygons:

```
CREATE TRANSFORMATION polygon_eq (x:polygon, y:polygon)
  RETURN eq: BOOL;
  BEGIN
  VAR eqs:SET OF BOOL;
  eqs = ACCESS { TRUE
                FOR z IN prefix(SEQ(x))
                WHERE seq_eq(append(left_div(x,z),z), SEQ(y)) };
  eq = nonempty(eqs);
  END
```

In the above example, *left_div* is a function defined on sequences with two input arguments. It simply returns the left quotient of the first argument by the second. The function *append* is the usual sequence concatenation operation and *seq_eq* is the equality predicate.

We now discuss how transformations may be implemented with the use of other languages and tools. We noted earlier the use of virtual concrete R-structures to bridge the gap between the data manager and the file manager. Virtual R-structures provide with the user with great flexibility in programming necessary functions (transformations). We illustrate this functionality in terms of a MATLAB function that displays a DEM element.

```
CREATE TRANSFORMATION display_dem (e:DEM)
  SOFTWARE: MATLAB
```

```
FILE = matlab_format(e)
BEGIN
load FILE.dat
plot(FILE)
END
```

Here *matlab_format* is an isomorphic translation of **DEM** elements into representation used by MATLAB and store them into a file. Since **matlab_format** is a virtual concrete domain, the file name is returned. The commands enclosed by *begin* and *end* are in fact MATLAB commands. CML replaces the file names and simply passes the commands to MATLAB.

5.2 The Creation of Modeling_schemas and Projects

We now describe how methodological concepts, such as *modeling_schemas* and *projects*, may be instantiated as high-level R-structures with the use of CML. As we noted previously when defining the R-structure **modeling_schemas**, representations of CML commands enter explicitly into the definition of some of these R-structures. We note that in a very general sense a model schema possesses the characteristics of a transformation, since there are certain inputs (R-domain elements and transformations) and certain outputs (R-domain elements). We require, however, persistence of the intermediate data elements in order to facilitate model debugging. Suppose **Spatial** and **Temporal** are two R-structures representing spatial and temporal objects and that the two transformations **Spatial**.projection and **Temporal**.projection are respectively defined on each. The following defines a transformation **Surface_Flow**.compute, which computes a representation in terms of a *Hydrograph*, of the flow of surface water over some area of interest.

```
CREATE TRANSFORMATION Surface_Flow.compute (o:Spatial, t:Temporal)
  RETURN H:Hydrograph
  BEGIN
  X = ACCESS { D IN DEM
               WHERE Spatial.projection(D) INTERSECT Spatial.projection(o) }
  Y = APPLY DEM.union TO X
  Z = APPLY DEM.slope TO Y
  A = APPLY Rainfall_Map.extract_by_range TO TEMPORAL.projection(t),
                                             SPATIAL.projection(o)
  B = APPLY Rainfall_Map.interpolation TO A
  C = APPLY Rainfall_Map.find_storm TO B
  D = ACCESS { S IN Surface_Flow
               WHERE Spatial.projection(S) INTERSECT Spatial.projection(o)
               AND Temporal.projection(S) = Temporal.projection(t) }
  H = APPLY SURFACE_FLOW.compute.solve_method to Z, C, D
  END
```

Using this transformation, it is now straightforward to define an appropriate R-structure for such modeling schemas:

```
CREATE MODELING_SCHEMA Surface_Flow_Model
  WHERE INPUT TYPE Spatial, Temporal,
```

```
OUTPUT TYPE Hydrograph
FUNCTION = Surface_Flow.compute
```

Once instances of such model schemas are created and stored, they may be accessed and modified like any other R-domain instance and transformations may be applied to them. Such transformations include *start*, *run* with or without *till before* or *after*, *halt*. For example, the *start* function initiates an execution by creating a process and returning the identifier but it does not execute any CML commands. The user can request specific executions with the *run* command. We note that it is in principle possible to store and save executions of model schema instances, and even higher level constructs if there is any scientific value in so doing.

Unlike transformations, executions of model schemas are persistent, in the sense that the intermediate results are saved. In addition, the data dependencies between steps can be found according to the syntax from the body of a transformation defining a model schema. In CMS, a subsystem called GOOSE, monitors the computations occurring in the system, and is able to extract and store the model schema and related information, such as the data lineage.

Projects In addition to containing information about the users and the project itself, each project is also a database system consisting of structured datasets. Projects are represented as a special type of R-structure. In particular, a project contains a description (e.g., members and objectives) and a set of R-structures:

```
CREATE PROJECT Amazon_Eco_System
    WHERE INVESTIGATORS = user1, user2, ...
          R-STRUCTURES = { DEM, Rainfall_Map, ... }
```

Different projects can be related to each other by a sub-project relationship. Each subproject has access to the R-structures and elements of the corresponding project.

```
CREATE PROJECT Hydrology_Models SUB-PROJECT OF Amazon
    WHERE INVESTIGATORS = user1, user2, ...
          R-STRUCTURES = { Surface_Flow, ... }
```

5.3 Metadata

CML currently stores *metadata* in the form of relations. The *abstract R-structure table* includes information about each abstract R-structure in the system including its name, the R-domain, a list of super R-structures, a list of transformations and a list of instances.[14] Similarly, the *concrete R-structure table* stores information about concrete R-structures, including the names, corresponding abstract R-structures, concrete super R-structures from which representational structures and transformations may be inherited, R-domains, transformation implementations, and a list of instances. (the names and locations for executable code). The *transformation table* records all transformations, including representational equivalences, isomorphisms and translations. For each model schema, CML creates a relation which includes the identifiers for the model runs and a list of functions names used in the particular run. Finally, for each abstract domain, an *instance table* contains each of the elements. Queries are permitted on the metadata itself and one may, for example,

[14] The current implementation of CML is based on POSTGRES [RS87]. Hence array types are used to hold lists.

list all abstract R-domains, or find all concrete R-domains for which a particular transformation is implemented.

6 SYSTEMS SUPPORT FOR COMPUTATIONAL MODELING

In this section, we discuss the system level support for CMS-like environments in general and for the CMS currently under development. We discuss the system level support for the management of R-structures (§6.1), for high level language CML (§6.2), and for managing the execution of computational models (§6.3).

6.1 Support for the Creation and Use of R-Structures

As defined above, a CMS may be viewed as a management system that supports the iterative process of scientific modeling in general and scientific database activities in particular. As noted in the hydrologic example, such activities are often embedded in a complex structure of projects and subprojects that involve sharing algorithms and datasets. In view of this, CMS uses a multi-database organized according to a tree-like structure that mirrors the project/subproject organization, as exemplified in Figure 5. Each project is implemented as a database. The database (project) at the root of the tree consists of R-structures and elements that are globally available. At each level, the users of a particular database can only access and manipulate R-structures to which they are given access. In particular, a subproject can only access R-structures that are a subset of those R-structures accessible to the corresponding project. For every node, the union of the databases along the path from the root to the node must be a consistent database. This is necessary to allow moving R-structures and elements from database to database. For instance, the user may move an R-structure from a leaf database to its parent database using a *store* command. Suppose **DEM** is going to be "released" to the *Surface Flow* group, the following expression can be used:

```
STORE DEM IN Amazon_Eco_System
```

We are implementing these ideas into our CMS for the Amazon Project.

6.2 Support for the High-Level Computational Modeling Language

CML provides a set of high-level operations that address a significant proportion of the needs of the scientific database users. CML is currently being designed as an interpreted language that may be used interactively by the user. A graphical version of CML (VCML, or visual CML) is also being implemented. As in other conventional database systems, CML employs the notion of a *session*, during which the user interacts with the system by executing CML commands. All CML sessions are persistent in the sense that variables and their values are retained across sessions. Users may belong a number of projects and own several R-structures and elements. For access control, *read, write,* and *execute* access rights are maintained for each R-structure, element, and project.

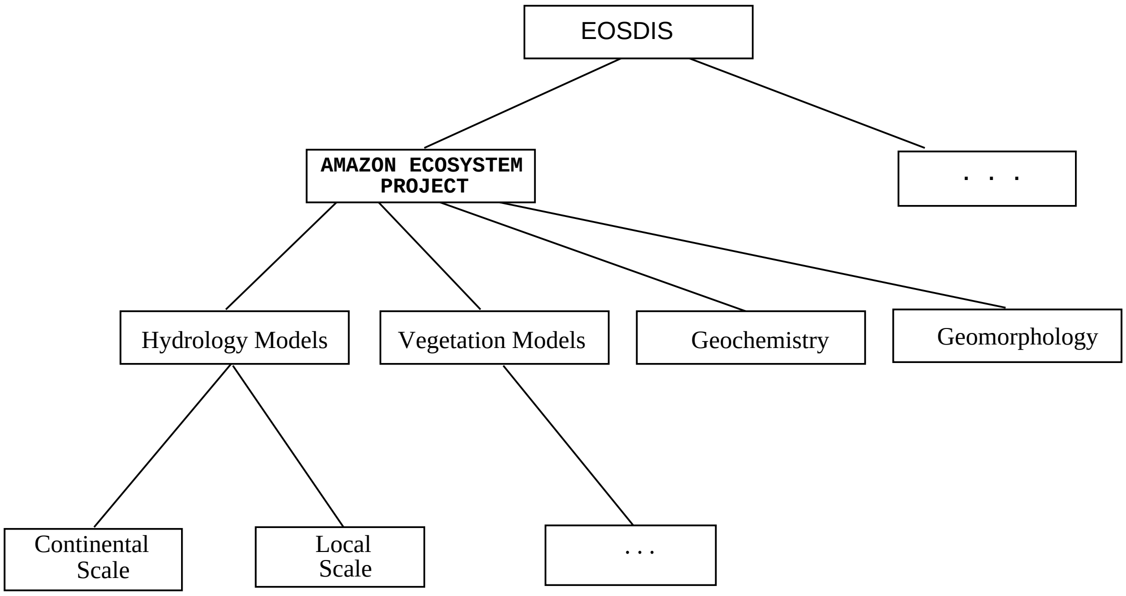

Figure 5: An example of a data bank

6.3 Dynamic Support For Model Schemas

The lack of support for the computational aspects of the modeling schemas in many systems is due to the absence of a common framework and language. CMS provides such a framework. In its current version, the actual implementation of the system support for modeling schemas is embedded in a special subsystem called GOOSE [AE93, AE94]. Its main role is to track the data interdependencies and to provide persistent data structures that represent the computational aspects of modeling schemas.

Existing systems generally provide the functionality to perform computational steps separately. This means that models must be executed step by step, and that users must carry forward their results "manually", without mechanisms to store information about the processes that have taken place. Furthermore, different locations or modifications to the algorithms or datasets require repetitions of the entire sequence. This is often totally impractical in complex applications in which long chains of transformations frequently involve large numbers of intermediate steps. Moreover, this approach does not support the use of pre-existing models in building more complex models, which is greatly facilitated by access to the processes that produce the model outputs, rather than the outputs alone. Thus, a key step in supporting modeling schemas is to have the ability to describe a sequence of execution steps *as a unit.*

We have already shown how several CML statements can be combined into a single transformation and the transformation embedded within a modeling schema. Once this has been done, it is possible to define higher level models that encapsulate complex execution procedures and yet can be treated in a similar fashion to other models within the system. This rises very interesting possibilities that are not found in other modeling systems. In first place, it allows modular design of complex models. The similarity with modular and structured programming techniques is not a coincidence. The models being built are eventually translated into software systems, and therefore it is necessary to provide software engineering tools. Our approach introduces the scientific research world to concepts such as code reusability and version control, opening new research areas and allowing known techniques to be used to advance the state of the art in information and modeling systems.

Modeling schemas are based on two elements, data that is used as input and is obtained as output, and algorithms that perform the transformation between input and output. To represent dependencies, we first need to represent these entities. Hence, the two main system constructs that GOOSE supports are the *object* and the *process*. An object is a data structure that links together the actual data and the metadata associated with it. Using objects to manipulate and refer to data has the advantage that both the interpretation of the data and the data itself can be accessed from a common reference. In most systems, the meaning of the dataset is exogenous and not found in the dataset itself [Tob79] which forces the user to resort to external sources to find such information. Similarly, a process is a data structure that stores information about a particular computation. In particular, a process consists of a set of inputs, a computation and a set of outputs. The computation can be *primitive* (in this case, a construction from the language previously discussed) or *composite*, a sequence of computations (which in turn can be primitive or composite). The purpose of a process is to allow the system to name and manipulate the processing that takes place (the algorithms that are applied to the data).

These two abstractions are both simple and powerful. They allow us to represent very complex models using a common notation suitable for computational purposes. For instance, the CML command *RESULT = apply Surface_Flow_Models.solve_method to Z,C,D* is represented within GOOSE as the following process:

PROCESS Step_x
    INPUTS Z,C,D
    OUTPUT RESULT
    TRANSFORMATION Surface_Flow_Models.solve_method

Obviously, there is little advantage in representing single steps, except for the possibility of encapsulating the particular computation. The power of the construct is best exemplified in the encapsulation of the entire sequence of operations, such as

PROCESS Water_Runoff_Model
    INPUTS Location, Time_Period
    OUTPUT Surface_Flow_Result
    TRANSFORMATION Surface_Flow_Analysis

which corresponds to the execution of a modeling schema. It is now possible to refer to this process as a black box that hides the particular details of the intermediate steps. Once the concept of a process is in place the language becomes easily extensible. Starting from the basic transformations and operations allowed by the language, the user can define increasingly complex processes by creating processes with composite computations. For instance, the example can be seen either as a single project (upper part of Figure 6) or as the combination of three processes (lower part of Figure 6). This effectively allows the user to select the level of resolution at which a particular model is to be analyzed and treated. During the development stages it is likely that different sections of a large model might be designed in parallel by different users. At this point it is useful to see the entire model as a combination of processes that can be run separately (or the global model can be run in a step-wise fashion, allowing the user to examine intermediate results). Once the model has been developed and tested, other users may want to use it as a global model, in which case it would be defined as a single process.

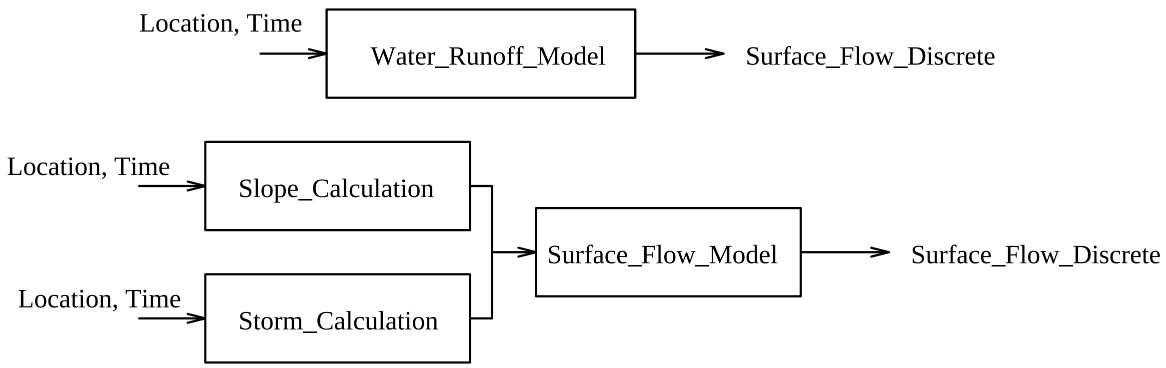

Figure 6: Two interpretations of the same model

7 A SYSTEMS ARCHITECTURE TO SUPPORT A CMS

The architecture of CMS, illustrated in Figure 7, involves several subsystems. A *visual user interface* represents much of the CMS functionality in an intuitive, user-friendly, and graphical form. In particular, the visual interface supports a visual version of CML as well as the text-based version. For example, it is possible to carry out operations on modeling schemas in terms of simple manipulations on graphs representing the schema. The core modeling functions of CMS are supported in the *Integrated Modeling Environment.* The CML component supports operations that are defined in the language, and passes operations defined in external software packages. The *GOOSE* environment is incorporated to provide support for the concept of modeling schemas. One may view the components of CMS and lower level modules as parts of a distributed database system. This performs the task of processing CML queries, managing the storage of data and its access over a distributed network of data sites. The current implementation of CMS uses POSTGRES as its data manager and Prospero [Neu92] as its (distributed) file manager. The integration of the Prospero file system with CML and POSTGRES allows POSTGRES to access virtual database object files "locally", although the files may reside physically on remote machines and be accessed using ftp or other methods. In the following, we briefly discuss data access, and CML command processing.

7.1 Data Access Subsystem

A significant issue that arises in relation to CMS is to provide storage mechanisms for R-structures that may contain large and complex elements. Such mechanisms should not only provide efficient access to the datasets of the collection of R-structures that define a CMS, but also the means to store, retrieve, and modify the various classes of elements that are involved in R-structures, such as transformations. Significant difficulties arise in the provision of such support as a result of the physical distribution of these elements. Within the Amazon Project, for example, while the R-structure corresponding to rainfall data may be organized in terms of a logically centralized abstract R-structure, the actual physical representation is distributed. CMS provides support for a logically centralized view of an R-structure that may be physically distributed. In particular, we are currently employing the Prospero file-system to provide such a view. Prospero allows files corresponding to an R-structure to be organized in a variety of ways that include temporal, spatial, spatio-temporal and content-based indexing.

The data access subsystem as a whole has three main components. The Data Managers in the

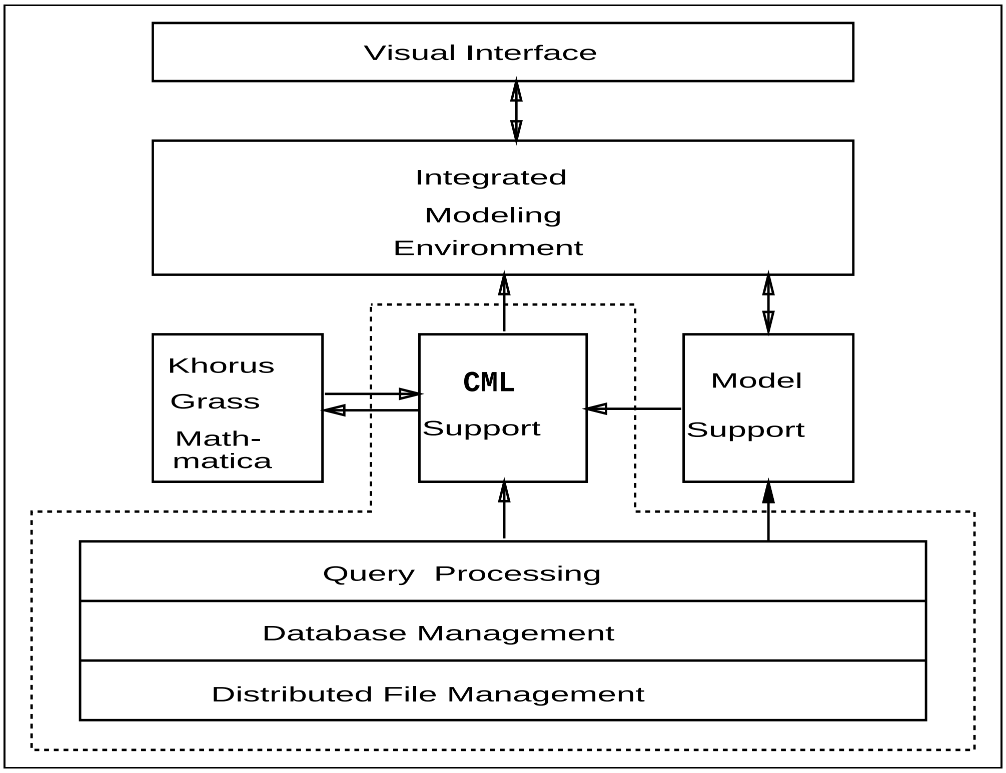

Figure 7: The High-Level Architecture of CMS

local and remote sites co-operate using the Data Access Server Protocol (DASP). The Clients of the Data Access subsystem like the MBBL user, the backend DBMS of the CMS communicate with the data manager using the Data Access Client Protocol (DACP). These modules cooperate with Prospero to provide the Distributed Data Service.

We provide extensions to the notion of Prospero filters, which are used with file directories, in the context of databases. The select filter mechanism allows users to selectively view data objects in a given directory. The derived filter mechanism permits users to organize objects based on the selected attributes of the objects (i.e., it creates virtual levels in the name space). Finally, the union filter mechanism allows users to combine seamlessly multiple directories stored at different locations into a single directory. In order to extend these notions in the context of databases, we need to apply the filters to the files instead of restricting them to directories. These filters can be used to implement efficient indexing for scientific data.

7.2 Query Translation

This subsystem includes the command interpreter for CML commands. The interpreter translates CML commands to POSTQUEL. At the CML level, the user views the distributed database as a single entity. Thus, query translation from CML to several distributed (and heterogeneous) databases, is very complex. Furthermore, structural representations of concrete R-structures may be very complex e.g., multiple levels of set nesting, which might not be supported by the underlying R-DBMS. Standard mapping rules have been designed to take care of these, and are explained in the following sub-section.

As shown in Figure 8, each CML command goes through the phases of Lexical and Syntactic

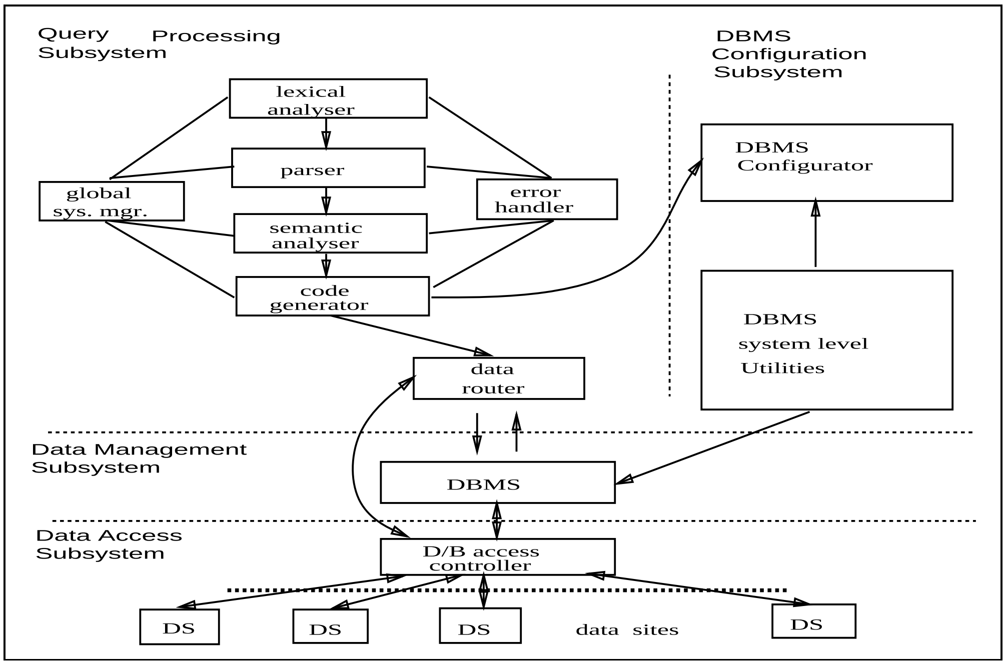

Figure 8: A Detailed Architecture for Database Support

Analysis before any *code translation* is done. We have designed various algorithms for decomposing CML commands into primitive constructs that operate on simple databases. These algorithms are employed in the Semantic Analysis phase to use the underlying DBMS for querying. Mapping rules for data structure translations are also employed here. It is in the Code Generation phase that the DBMS Configurator (as described in the next section) is invoked, to generate syntactically correct commands, in the query language supported by the DBMS, from primitive CML constructs. A Global System Manager keeps track of all the symbol tables and other system meta information for the four phases.

Data for CML may be stored as part of the underlying DBMS or might just exist as flat UNIX files. It is the function of the Data Router to route the query to the appropriate data handler.

7.3 Mapping data structures from CML to POSTGRES

Although theoretically speaking the mapping of a CMS database into a relational database is not difficult, the translation affects directly the CML interpreter. We have adopted a simple strategy for the translation. For tuple constructs, a direct representation is used. For set constructs, the general rule is to break the CML relation containing the set into two parts, i.e., one that contains the attributes other than the set(s) and the other that contains tuples related to the set(s).

For example, an element Manaus in CML (in the R-structure of **DEM**) may have the structure $[dem\_oid, resolution, \#row, \#col, \{[point, elev], [point, elev], \ldots\}]$. CML will store the element (in POSTGRES) as two relations, named *Manaus* and *PE_PTR*, respectively, where their attributes include:

```
DEM = [dem_oid, resolution, # row, # col, PE_PTR]
PE_PTR = [point, elev]
```

Finally, for sequence or array constructs, the implementation adds a new, "hidden" attribute

SEQ_NO in the POSTGRES relation. This keeps the sequence number of the tuple at the time of insertion. Entries in the relation could be managed in a way similar to that of normal array data structure. The above scheme may be extended for an arbitrary number of sets in a tuple and for depths of set nesting.

7.4 A Translation Technique

Now we discuss an interesting technique developed during the implementation for mapping CML commands into POSTQUEL queries. Instead of running an algorithm to generate the relational queries, we have designed and implemented a "generic" *configurator* which takes as input a specification of the mapping from internal representations of commands to concrete syntax in some query language and produces the syntactically correct queries.

We have employed a 2-tier approach to implement the configurator. The first tier comprises specifications for mapping primitive CML constructs to the constructs of the query language of the underlying DBMS. The specifications are *loaded* when the user initiates a CML session. There is provision for specifying more than one configuration system at the time of *loading*. This means, for instance, that the user could give specifications for both SQL and POSTQUEL at login, and subsequently use them to generate commands for both kinds of DBMS's.

The second tier consists of the actual data for which the DBMS command has to be generated. All *primitive* CML commands are stored in predefined structures that are accessible to the configurator. Depending upon the specifications provided in tier 1, and the underlying DBMS, the configurator generates a syntactically correct query.

7.5 Modeling Support

Finally, the modeling support is implemented using GOOSE [AE93, AE94]. GOOSE is organized in two levels: the object and the process levels. The organization of the architecture reflects this conceptual division, the *process base* provides support for persistent projects, the *object base* provides support for persistent objects. The *interpreter* serves as the interface with the rest of the system, while the *modeling engine* implements the algorithms and functionality of GOOSE. The *operations log* is a library that maps the transformations contained in the projects to sequence of operations in the appropriate language. The process base, the object base and the operations log are passive repositories of information that organize to facilitate access to modeling data and allow the system to process queries such as "what datasets are used in a given model", "what datasets are used to create an object", "what transformations have been applied to a particular dataset", etc. The actual search on these databases and the control of the different operations take place in the modeling engine, which uses the interpreter to extract information from the rest of the system regarding what computations are taken place.

Each computation performed is followed by an operation in the corresponding object. For instance, each object has a *sources* and a *usage* attribute to store the lineage and the dependencies of the object. Thus, a command such as *RESULT = apply Surface_Flow_Models.solve_method to Z,C,D* results in the creation of the object RESULT with Z,C,D as sources. The objects that correspond to Z,C, and D are updated so their usage attribute contains RESULT. Thus, each particular command in the language corresponds to an operation on an object (the complete translation is beyond the scope of this paper). In this way GOOSE can keep track of the data interdependencies created by applying transformations. For instance, the lineage of an object is the source objects and algorithm used to create it. For any particular object, its lineage can be calculated by accessing its "sources" attribute. The particular algorithm can be obtained by accessing the project also mentioned in that same attribute. If necessary, this can be repeated for the objects that are sources, and so forth, which result in the complete lineage of the object. Once this lineage has been established, several other operations become possible. If for any of the objects that are in the lineage there is a new version, it is possible that the object in particular is out of date since it is based in an old version. If the user specifies it, the system could automatically propagate any changes to an object in the lineage to all other objects that are derived from it. Note that all the information required to do this is available in the form of processes, metadata information in the objects and the actual commands are stored in the operations log.

Once a process has been defined, its execution can be controlled by the system. Assume that there is a certain computation that uses satellite information updated daily. The process corresponding to this calculation can be set so as to be automatically execute every time a new version of the inputs becomes available. Each execution results in a new version of the outputs (and the sources of each output can be easily determined from its lineage). The user can then combine all the different versions of the outputs to study the evolution over time of the phenomenon.

8 CONCLUSIONS

In this paper we have developed a conceptual model of the process of scientific modeling. This model provides appropriate specifications for a large number of concepts relating to the iterative process of scientific modeling that need significant computational support. Our approach is based on close research interactions with multidisciplinary groups of scientists and on the development of support for specific research problems. Throughout the paper we have used hydrological modeling examples from a specific EOS project to illustrate the concepts and notions developed in the paper. Our conceptual model, however, captures scientific activities in general. Based on this conceptual model and its central concept of *representational structures*, we have also developed a framework for a computational modeling system (CMS) that supports scientific activity within a simple, unified, computational environment. A CMS may be viewed as a fusion of three relatively distinct components: a computational environment that is designed on the basis of a comprehensive and consistent model of scientific modeling activity; a knowledge base that may be tailored for any specific domain of scientific investigation; and persistent programming support that combines compiler, database and other technologies. We have also described a high-level computational modeling language (CML) with which the modeling concepts that are definable in the computational framework may be constructed and manipulated in a simple, uniform manner. In particular, we were able to express many of the examples from the hydrological modeling problem much more conveniently than is the case in existing environments. Finally, we described the design and implementation of a specific computational modeling system, *Amazonia*, for supporting scientific research projects in several areas of earth science. The Amazonia approach to a CMS integrates data management with the modeling support environment of a distributed system. In summary, we have developed the paradigm of a CMS for supporting scientific activity. We have developed the theoretical and conceptual basis, proposed a high level computational modeling language, and discussed various design and implementation details.

References

[AE93] G. Alonso and A. El Abbadi. GOOSE: Geographic Object Oriented Support Environment. *Proceedings of the ACM Workshop on Advances in Geographic Information Systems, Arlington, Virginia, USA*, pages 38–49, November 1993.

[AE94] G. Alonso and A. El Abbadi. Cooperative Modeling in Applied Geographic Research. *To appear: Proceedings of the International Conference on Cooperative Information Systems, CoopIS'94, Toronto, Canada*, 1994.

[AS93] P. Adams and M. Solomon. An Overview of the CAPITL Software Development Environment. *Technical Report TR-1143, Computer Science Department, University of Wisconsin-Madison*, April 1993.

[BDMN73] G. M. Birtwistle, O.-J. Dahl, B. Myhrhaug, and K. Nygaad. *SIMULA Begin.* Auerbach Press, Philadelphia, PA., 1973.

[Car84] L. Cardelli. A semantics of multiple inheritance. In G. Kahn, D.B. MacQueen, and G. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 51–67. Springer-Verlag, 1984.

[Car88] L. Cardelli. Structural subtyping and the notion of power types. In *Proc. ACM Symp. on Principles of Programming Languages*, 1988.

[Chu93] W. Chu, editor. *Proceedings of the NSF Scientific Database Projects*, AAAS Workshop on Advances in Data management for the Sicentist and Engineer, Boston, MA., February 1993.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[dHvO92] S. de Hoop and P. van Oosterom. Storage and Manipulation of Topology in Postgres. *Proceedings of EGIS'92, Munich, Germany*, pages 1324–1336, March 1992.

[Doz90] J. Dozier. Looking ahead to EOS: The earth observing system. *Computer in Physics*, May/June 1990.

[EFT84] H. D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Springer-Verlag, 1984.

[EOS90] EOS: A Mission to Planet Earth, NASA, Washington, D.C., 1990.

[HGW93] N.I. Hachem, M. Gennert, and M. Ward. The Gaea System: A Spatio-Temporal Database System for Global Change Studies. *AAAS Workshop on Adavances in data Management for the Scientist and Engineer, Boston, Massachussetts, USA*, pages 84–89, February 1993.

[HQGW93] N.I. Hachem, K. Qiu, M. Gennert, and M. Ward. Managing Derived Data in the Gaea Scientific DBMS. *Proceedings of the 19th International Conference on Very Large Databases, Dublin, Ireland*, 1993.

[HTM93] J. Hardisty, D. M. Taylor, and S. E. Metcalfe. *Computerised Environmental Modelling: a Practical Introduction Using Excel.* J. Wiley & Sons, 1993.

[IEE93] Bulletin of the Technical Committee on *Data Engineering*, 16(1), March 1993. (Special Issue on Scientific Databases).

[Jon90] C. V. Jones. An introduction to graph-based modeling systems, Part I: Overview. *ORSA Journal on Computing*, 2(2):136–151, 1990.

[Jon91] C. V. Jones. An introduction to graph-based modeling systems, Part II: Graph-grammars and the implementation. *ORSA Journal on Computing*, 3(3):180–206, 1991.

[Lea92] D.D.E. Long and et al. REINAS: Real Time Environmental Information Network and Analysis System: Concept Statement. *Technical Report, Baskin Center for Computer and Information Sciences, University of California at Santa Cruz, UCSC-CRL-93-05*, January 1992.

[MP94] C. B. Medeiros and F. Pires. Databases for GIS. *SIGMOD Record*, 23(1):107–115, 1994.

[Neu92] B. Clifford Neumann. Prospero: A Tool for Organizing Internet Resources. *Electronic Networking: Research, Applications, and Policy*, 2(1), 1992.

[OM93] J. J. Ordille and B. P. Miller. Database challenges in global information systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1993.

[PSS+87] H.B. Paul, H.J. Schek, M.H. Scholl, G. Weikum, and U. Deppisch. Architecture and Implementation of the Darmstadt Database Kernel System. *Proc. of the SIGMOD International Conference on Management of Data, San Francisco, CA*, pages 196–206, May 1987.

[Req80] A. A. G. Requicha. Representations for rigid solids: theory, methods, and systems. *Computing Surveys*, 12(4):437–464, December 1980.

[Rob91] D. Robertson et al. *Eco-Logic: Logic-Based Approaches to Ecological Modelling.* MIT Press, Cambridge, Mass., 1991.

[RS87] L. A. Rowe and M. R. Stonebraker. The POSTGRES data model. In *Proceedings of the 13th Conference on Very Large Data Bases*, pages 83–96, Brighton, England, 1987.

[SDT92] Spatial Data Transfer Standards, U.S. Geological Survey, 1992.

[Seg93] A. Segev. Processing heterogeneous data in scientific databases. In Chu [Chu93].

[SSAE93a] T. R. Smith, J. Su, D. Agrawal, and A. El Abbadi. Database and modeling systems for the earth sciences. *IEEE Bulletin on Data Engineering*, 16(1), 1993.

[SSAE93b] T. R. Smith, J. Su, D. Agrawal, and A. El Abbadi. MDBS: A modeling and database systems to support research in the earth sciences. In Chu [Chu93].

[SSD89] *Design and Implementation of Large Spatial Databases: First Symposium.* Springer-Verlag, 1989. Lecture Notes in Computer Science; 409.

[SSD91] *Advances in Spatial Databases: 2nd Symposium.* Springer-Verlag, 1991. Lecture Notes in Computer Science; 525.

[SSD93] *Advances in Spatial Databases: Third International Symposium.* Springer-Verlag, 1993. Lecture Notes in Computer Science; 692.

[SSU90] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *ACM SIGMOD Record*, 19(4):6–22, December 1990.

[Tob79] W.R. Tobler. A transformational view of cartography. *The American Cartographer*, 6(2):101–106, 1979.

[vOV91] P. van Oosterom and T. Vijlbrief. Building a GIS on top of the open DBMS Postgres. *Proceedings of EGIS'91, Brussels, Belgium*, pages 775–787, April 1991.

[WH92] Waterfeld W. and Schek H.J. The DASDBS Geo-Kernel - An Extensible Database system for GIS. pages 69–84, 1992. in Three Dimensional Modeling with Geoscientific Information Systems, A.K. Turner (ed.), Kluwer Academic Publishers, Netherlands.

[Wol89] A. Wolf. The DASDBS GEO-Kernel, Concepts, Experiences, and the Second Step. In SSD89 [SSD89]. Lecture Notes in Computer Science; 409.

[Wol90] A. Wolf. How to Fit Geo-Objects into Databases - An Extensibility Approach. *Proceedings of the First European Conference on GIS, Amsterdam*, April 1990.

9 APPENDIX I: A FORMAL BASIS FOR CHARACTERIZING SCIENTIFIC MODELING

An important issue concerns whether the preceding characterization of scientific modeling activity provides a comprehensive and consistent basis on which to design and develop a CMS that meets the requirements of scientific modeling. We address this issue in terms of an answer to the following question. *Is there a simple, extensible language that possesses a sound and complete set of inference procedures and that also possesses an expressive power that is both necessary and sufficient for representing as "first-class entities" all of the modeling concepts, both phenomenological and methodological, that are required in scientific modeling activities in terms of an inductively-constructed collection of R-structures?* An affirmative answer to this question is of value for several reasons. First, it indicates that any appropriate instantiation of our preceding conceptualization of scientific activity may be viewed as a "model" of the formal description, hence indicating that our conceptualization is consistent; second, it may be employed in providing a formal specification for any CMS; and third, it may be used to indicate some of the computational issues that are associated with CMS.

Before answering this question, we first make explicit our underlying assumption that the *appropriate language for scientific modeling is the language of mathematics*, and that, in principle, there should be no constraints on which fragment of "reasonable" mathematics may or may not be employed in scientific modeling. Our question now becomes answerable in terms of a first order logic that initially involves only the equality predicate and the predicate for set membership $\{\epsilon\}$ in its language $\mathcal{L}^{\{\epsilon\}}$, together with the Zermelo-Frankel axioms of set theory and the Axiom of Choice (ZFC). It is widely accepted that this system is sufficient for expressing "all" of mathematics [EFT84]. In particular, it is possible on this basis to build a first-order treatment of transformations, relations and R-structures which preserves the standard notation for these linguistics constructs and permits unrestricted quantification over function and relation variables in formulae.

This basis permits us to represent in a very direct and natural manner the data models that were discussed above. Using $\mathcal{L}^{\{\epsilon\}}$, and the ZFC axioms one may extend this language with new sets of constant, function and predicate symbols that may be employed in representing any simple or complex mathematical entity; any proposition concerning such entities; and any proof of such propositions. One may, for example, define constant symbols such as $\emptyset$; function symbols such as the tuple and set constructor symbols ($[]$, $\{\}$) and $\cup, \cap$; predicate symbols such as $\subseteq$; and, above all from our point of view, symbols for *R-structures* that have the general form $\{\mathcal{D}, .\mathcal{T}, .\mathcal{R}, .\mathcal{I}\}$. Furthermore, one may aggregate any collection of defined symbols into " R-structures" and one may easily specify any set of inductively constructed R-structures in such an extended language. Furthermore, as we note below, this foundation for a CMS provides specifications for important components of computational modeling languages (CMLs) that may be designed to support scientific modeling within the CMS framework. For further reference, we may denote by $\mathcal{C}$ the additional set of symbols that are developed in some extension of the language, and by $\mathcal{L}^{\{\epsilon\}\cup\mathcal{C}}$ the extended language.

Finally, we note that there is considerable evidence that the theory for ZFC is consistent, and hence that any CMS that is constructed on this basis must also be consistent. First order logic is also characterized by such important properties as soundness, completeness and compactness. We do not view the undecidability of FOL to be a critical issue for our current purposes, since scientists face this issue all the time whenever they employ mathematics in general as their main symbolic medium and when they employ general programming languages in their computing activities.

10 APPENDIX II: CLASSES OF R-STRUCTURES

Example. *Empirically determined entities and relationships* versus *theoretically determined entities and relationships*. In order to consider how an inductively-defined collection of R-domains may embody the *empirical/theoretical* distinction, we first discuss the nature and role of this distinction in scientific modeling activities. When symbolic expressions are largely determined by observation and measurement, they are typically referred to as *empirical*, and when they are largely determined as the outcome of inference procedures applied to expressions that are not (directly) empirical, they are typically referred to as *theoretical*. [15] It is clear, however, that the iterative process of constructing and evaluating scientific models of phenomena involves symbolic expressions that span the entire range of the empirical-theoretical spectrum. "Datasets" are examples of largely empirical symbolic expressions, while many classes of "mathematical models" are best thought of as examples of theoretical symbolic expressions. There are, however, other expressions, as we exemplify below, that combine empirical and theoretical components. The iterative construction and evaluation of symbolic representations of phenomena typically involves a variety of procedures in which more empirically-based expressions are employed as an aid to constructing and evaluating more theoretically-based expressions. The main value of the empirical-theoretical distinction appears to lie in providing a classification of sets of expressions that are used in this process in order to indicate the origin of the sets of expressions, some of the essential properties of the representations (such as errors), the role that they occupy in the modeling process, and the sets of operations to which they may be subject. Hence the distinction appears to relate to a fundamental subset of activities that are involved in the process of constructing scientific models of phenomena.

In characterizing the distinction in terms of R-structures, we take the basic point of view that scientific modeling activities involve a search for an R-structure, or a set of R-structures,

[15] It is typically the case that expressions towards the theoretical end of this spectrum are more general, more expressive, and in some sense less ambiguous than expressions towards the empirical end of this spectrum.

that in some sense provides the most appropriate symbolic representation for the phenomena. In particular, we may view this process as one in which empirically-derived R-domain elements are examined in relation to some collection of R-structures in order to determine which R-structure(s) best provide a "model" of the phenomenon. Typically, it will be necessary to examine whether the set of empirically-determined elements satisfy various conditions in order to determine possible membership in the R-domain of the target R-structure. Hence, it is reasonable to assume that R-domains that are largely empirical possess a relatively simple *semantics* in the sense that the associated transformations, relationships, constraints and expressions are relatively small in number and/or relatively "weak".

The determination of the most appropriate R-structure for some phenomenon will typically require a variety of operations including:

- the construction of R-structures that are appropriate for representing the empirically-determined entities with a minimum of assumptions concerning the nature of the entities;
- the application of sequences of transformations to some of the empirically-based candidate entities, in order to determine whether such entities satisfy certain conditions;
- the application of sequences of transformations to some of the empirically-based entities in order to produce entities that define in part the target entities;
- the comparison of the empirically-based entities with entities in the target R-domain or with entities that result from transforming entities in the target R-domain.

Since it is the operations and relationships on an R-structure that are a major determinant of its semantics, and since the classes of exemplary operations discussed above involve many operations that are specific to the nature of empirical data, it seems reasonable to differentiate empirically-based R-structures on the basis of the classes of transformations and relationships to which they are subject. For example, the empirically-based representations will typically possess associated "errors" or "ambiguities" and it may be necessary to associate, with such domains, special transformations and relationships that permit the representation and manipulation of such information. As empirically-based representations are subjected to a given sequence of transformations, certain properties of the representation may change, especially in relation to the associated errors. Hence it will be necessary to keep track of the processing and the errors. One approach to this issue is to provide transformations that map empirical representations into representations of errors. We also note that it will generally be the case that membership of the representations of some phenomenon in the R-domain of some structure can never be ultimately determined, but can be assigned on a temporary basis. Hence one needs to associate with the membership status of such entities an indication of the nature of the membership and the "evidence" supporting such membership. □

Example. *Methodological entities and relationships* versus *phenomenological entities and relationships.* The distinction between *methodological* and *phenomenological* R-structures is based on the notion that methodological R-structures have an interpretation in terms of the entities and relationships of the modeling process itself whereas phenomenologic R-structures possess an interpretation in terms of phenomena that are being modeled. Methodological R-structures are required because the process of scientific modeling involves a significant degree of introspection, during which there is reference to entities and relationships that represent the modeling process. For example, it is frequently convenient to consider one model of some phenomenon as a whole in comparison with other models; or to refer to debugging of models; or to the coupling of models. Since such operations occur in the environment of modeling, it is reasonable to attempt to support them in the environment of a CMS. Among the many modeling concepts that may be represented as R-structures, we have identified two that appear to have a particular importance, namely the concept of *modeling schema* and the concept of *project*. We provide definitions and applications of representations of these concepts below. □

Example. *Explicit representations of entities and relationships* versus *implicit representations of entities and relationships*. The distinction between *explicit* and *implicit* R-structures involves a trade-off between the size of the symbolic expression that is required to represent some class of entities and the amount of computation is required to obtain representations of the properties of the entities. The distinction is well-exemplified by the different classes of R-structures that we may employ in representing surfaces in three-dimensional Euclidean space. On the one hand, we may construct *explicit representations* of such surfaces, which may, for example, be given in terms of an equation $z = z(x, y)$ that permits us to compute the $z - value$ of the surface directly from a corresponding pair of $x-$ and $y-$values. On the other hand, we may construct *implicit representations* for various classes of surfaces in terms of equations in which z is not explicitly represented in terms of x and y, but may be, for example, be implicitly represented in terms of various derivatives in x and y. Hence, there it is necessary to first solve the equation in order to determine, for example, the explicit form of the dependence of z-values on x, y-values. Clearly one may have implicit R-structures and explicit R-structures that are equivalent in the sense of representing equivalent domains of entities. □

Example. *Exact representations of entities and relationships* versus *approximate representations of entities and relationships*; The distinction between *exact* and *approximate* R-structures is based on a criterion that is related to a trade-off between the amount of computation that is required to obtain representations of the properties of the entities and the degree to which there is any error in the representations of these properties. This distinction is also exemplified by the different classes of R-structures that we may employ in representing surfaces in three-dimensional Euclidean space. Surfaces of spheres, for example may be represented *exactly* by either explicit or implicit R-structures in standard ways. On the other hand, we may construct representations of such surfaces that are based on the notion of projecting the surface of a sphere onto a plane, two-dimensional surface. However, it is not possible to preserve all of the properties of a spherical surface in the projections. Hence, the representations that are based on projection are approximate in the sense that some properties that are encoded in the exact representations are not encoded in the approximate representations. □

Example. *Visually-meaningful representations* versus *non-visually-meaningful representations*; The distinction between *visual* and *non-visual* R-structures is based on a criterion that is related to a trade-off between representations that are efficient with respect to the specification of the computation of properties and representations that are expressive with respect to the human visual/cognitive system. □