

Regarding the current Arc Hydrology Data Model (AHM), I think it could be simplified. It seems to me that the 5 subclasses to Hydropoint: Hydrostructure, Monitoring, FlowChange, ControlPoint should rather be represented as generic hydropoints with different subtype fields. The subtype field could be allowed to have 10 values to cover all the combinations shown on the diagram. The motivation for this is parsimony. It seems to me that the object paradigm is most useful when objects have specific attributes that are different from other classes. For example one could have a generic shape as a class, and subclasses 'circle' and 'triangle'. The circle subclass may have attributes, diameter, area, etc. The triangle subclass may have attributes 3 side lengths, vertex angles, area etc. Some of these are the same so can be put in and inherited from the generic shape class, while some are different, and not even defined for other subclasses. For example vertex angles are not defined for a circle. In the case of the subclasses to hydropoint this sort of fundamental distinction does not seem to exist. The attributes given are the same for each subclass so could easily be promoted to the superclass and subclasses eliminated.

Part of my understanding of objects comes from a still rather reduced instruction set knowledge of C and C++. Here the historical development of programming language capabilities went from simple variables and arrays, with very few allowable types of data that we grew up programming in (i.e. float, double, integer, array, character etc), to structures that allowed data objects to be comprised of a collection of datatypes, to classes that included multiple datatypes and computer code that embedded object specific operations - the functionality or methods I was harping on at the conference. This is still to a large extent lacking for the AHM. However it seems to me that the motivation for standardization of an object data structure is so that people can invest time in coding that relies on this structure. Given this perspective I do not think it is that helpful to have multiple standards - as this requires the different standards to be accounted for in any code. Extensions are a similar (but lesser) problem, because they need to be specifically handled. Therefore my opinion is that the consortium should try and arrive at a single object data structure design that accommodates the needs of as many parties as possible without extensions, then try and promote this as a standard. This will be difficult because standards require a lot of buy in. Furthermore I would be surprised if the national hydrography dataset structure (NHD) does not become a de facto standard. This is because most of the data in the US is already in this form. So at one level one may say, why are we bothering with an object data structure design. The NHD is the standard and programmers need to program to work with it, or else their programs will not be useable. Now fortunately (or through good design) the NHD seems to fit into the Arc Hydrology model (AHM) without too much modification (the NHD extension). I would suggest that the AHM be modified to contain exactly the NHD structure, plus components for things like time series that are not in the AHM. The HEC extension does not have the advantage of a large body of data to back it up, but is designed to facilitate linkage to the widely used HEC models. I would suggest that the key elements of the HEC extension necessary for linkage to HEC models as generic models also be incorporated into the AHM that contains exactly the NHD structure to have a single AHM data structure that meets (perhaps with some compromise) the needs of all three groups. I think a single AHM data structure is better than one with a bunch of extensions.

I have thought quite a lot about spatial fields, represented as grids, and it seems that the object paradigm could be quite helpful here. We also need to take advantage of the ability to define objects that mix types, e.g. contain vector and raster data. Specifically in the raster area, one of the things Steve Kopp mentioned was that in the future releases of AI8 they are going towards a transparent distinction between different formats. The same functionality will be available for data stored as grid, or jpeg, or bitmap etc. This is logical from a computer software perspective - because a modeler and user should not have to care how the data is stored in the computer. I do not know the details of ESRI grid files, and the clever tiling and compression they use, I just know I can use certain function calls to get values out that I can think of as logically organized into an array. However it does seem important from a modeler and user perspective to distinguish between what the data in the arrays means physically. This capability is not currently present. Elevation data is stored in exactly the same structure as flow direction and contributing area data, yet the operations that should be allowed on each are very different. Errors would result if raw elevation data were interpreted as flow directions, or contributing areas by a program. This (in the light of the object paradigm) says to me that there is a need to distinguish between things like elevation, flow direction and contributing area as different object types, with different methods to be associated with each. The picture in mind (figure 1) therefore comprises a generic grid object with subclasses distinct in their functionality or methods.

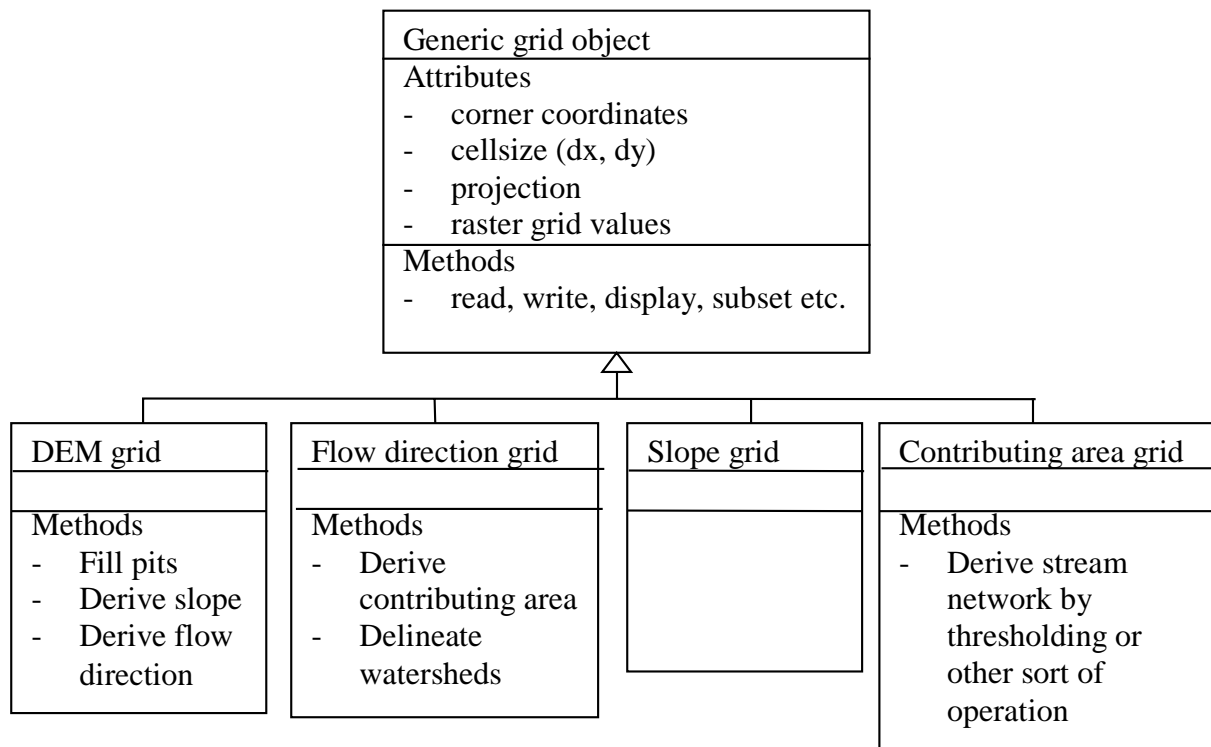


Figure 1. Possible landscape object structure. The distinction between subclasses is not the data structure, but the methods that are allowed for each.

[Note that in the figure above I have extended ESRI's current notion of a grid to include projection information and rectangular (with $dx \neq dy$) grids. These are in my opinion shortcomings of the existing structure. In working with the National Elevation dataset, on 1 second data in geographic coordinates the effective longitude and latitude grid spacing changes

with latitude. Our models for slope, contributing area and flow directions need to be adjusted to account for this detail (mine do - but this feature is not utilized with square grid data). The projection information should also allow for automatic display and working with data in a different projection and representation of non-north aligned grid data together with north aligned grid data.]

The object approach, it seems to me, also provides (or should provide) the capability to integrate fundamentally different types of data into objects, such as vector representations of streams and raster representations of landscape/hillslope properties. The notion of structures in C++ does this. A structure can comprise a one dimensional array representing a vector and 2 dimensional array representing a raster, plus header information. The notion of classes in C++ mixes code (methods) with this, removing the distinction between data and methods. This may be useful in a watershed object model. If a watershed object contains elevation data and a user asks for watershed area it is plausible for the code embedded in the object to on the fly evaluate flow directions, delineate watersheds and return the answer etc. It then becomes an implementation/efficiency consideration whether the data is saved or recomputed when needed - a detail the user could be insulated from. Figure 2 expands on this concept.

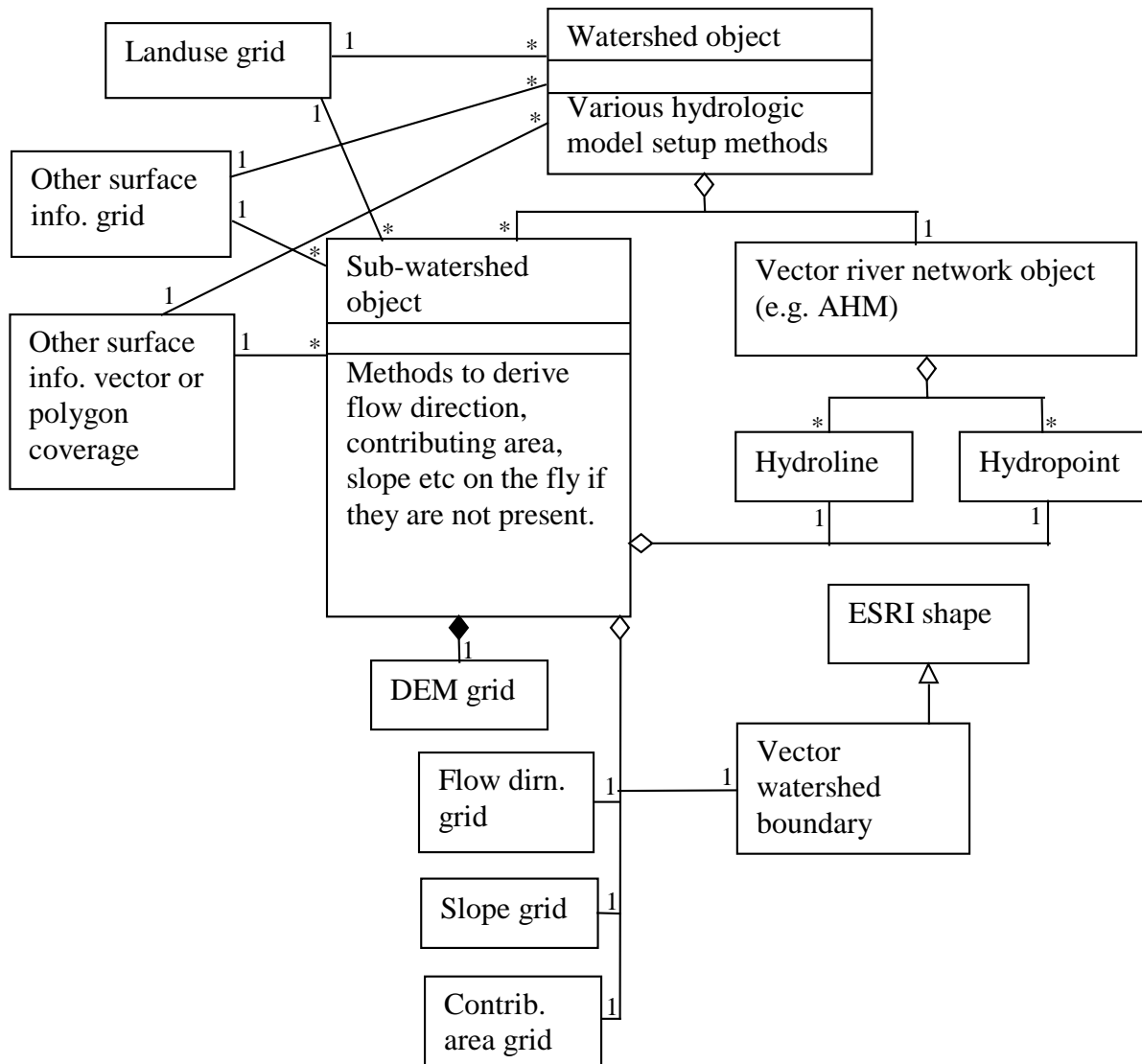


Figure 2. Possible watershed object structure.

This object model might be used (called) by a hydrologic model setup method (for example to set up HEC1 input files). The object would then return things like coefficients and parameters for each subwatershed derived from the DEM and landscape information, as well as the model element connectivity information. The object functionality would include and use on the fly where necessary the standard methods for derivation of slope and contributing area, and delineating watersheds, information needed to derive hydrologic model parameters. The derivation of geometric connectivity and methods for averaging over watersheds to determine parameters is fairly general methodology applicable to many hydrologic models. Distributed hydrologic models could be viewed as comprising spatial model elements connected following physical geometry, combined with time series parsing and combining functions that represent the time dependence and dynamic functionality of the model. A complete distributed hydrologic modeling system might comprise this spatial functionality through interaction with a GIS and a suite of time functionality components.

Another benefit of the object paradigm is encapsulation, i.e. encapsulation of methods with data. Concepts such as contributing area can be defined for a generic surface, with different methods for computation depending upon how the surface is represented in terms of a data structure, i.e. grid or TIN or contours. At a physical system level the user may want the concept of a surface object to take care of this detail so that it does not matter what form the data is in.