Jason Love[1*], Robert Heaphy, Ph.D.[2], Seth Kenner[1]

[1]RESPEC, Rapid City, South Dakota
[2]Pod Associates Inc., Albuquerque, New Mexico
*Jason.Love@respec.com

BACKGROUND

Both people and the ecological integrity of aquatic systems rely on healthy watersheds. Strategic protection and water-quality improvement in the nation's watersheds—rivers, lakes, wetlands, and streams, as well as in our estuarine, coastal, and ocean waters—require holistic, integrated solutions that emphasize accountability. One of the EPA's key objectives is improving watershed-based approaches.

Watershed-scale environmental models are increasingly being used to assist in developing watershed-based approaches for protecting and restoring our nation's waterbodies. Anticipating environmental problems by developing and applying mathematical models to simulate pollutant movement through a watershed has been the subject of intensive EPA research for decades. An outcome of these efforts is the Hydrological Simulation Program – FORTRAN (HSPF), which is an important tool in watershed-based assessment and planning. HSPF is thought to be the most accurate and appropriate management tool presently available for the continuous simulation of hydrology and water quality in watersheds.

Since its initial development over 30 years ago, the HSPF model has been applied throughout North America as well as in numerous countries and climatic regimes around the world. In North America, it is a key model used in developing complex total maximum daily load (TMDL) assessments and for assessing the effectiveness of restoration scenarios. For more than 2 decades, an HSPF-based watershed model has been used to simulate nutrient and sediment load delivery to the Chesapeake Bay as part of the Chesapeake Bay restoration effort. HSPF uses information such as the time history of rainfall, temperature, evaporation, and parameters related to land-use patterns, soil characteristics, and agricultural practices to simulate the processes that occur in a watershed. The initial result of an HSPF simulation is a time history of the quantity and quality of water that is transported over the land surface and through various soil zones down to the groundwater aquifers. Runoff flow rate, sediment loads, nutrients, pesticides, toxic chemicals, and other quality constituent concentrations can be predicted. The model uses these results and stream channel information to simulate instream processes such as hydraulics, sediment transport, and nutrient cycling. From this information, HSPF produces a time history of water quantity and quality at any point in the watershed.

The current HSPF model is written in over one hundred thousand lines of Fortran-77 code. For the majority of applications, HPSF uses an out-of-date binary file called a Watershed Data Management (WDM) file to store time series data. The WDM file was a highly effective format when it was created, but it has not kept up with advancing technology that has caused increasing limitations on its use.

CHALLENGES POSED TO HSPF TODAY

When HSPF was created, the developers implemented a top-down design and coding approach to leverage the best practices of structured design and coding at that time. The design was intended to make the system relatively easy to extend so that users could add their own modules with relatively little disruption to the existing code. At that time, it was "hoped that HSPF will become a valuable tool for water resource planners." Today, over 30 years later, the model remains largely unchanged and is one of the premiere watershed-scale environmental models in use. This is a truly remarkable testament to the vision and expertise of these developers. However, the change in technology over the last 30 years and our scientific understanding of environmental processes mandates that upgrades be made to HSPF so that it can evolve and continue to be the premiere tool for water resource planners. Some of the challenges that have emerged are identified below. In general, these challenges are common to all legacy software; they include maintaining the code, upgrading functionality, and integrating with modern technology and other software tools.

The User Control Input (UCI) file, which contains all of the nontime-series information and parameters needed for an HSPF run, is based on an 80-column ASCII "Punch Card" input. This approach required defining a large number of fixed format tables to specify all of the HSPF simulation flags, parameters, and initial conditions. The PERLND section, which represents pervious land segments, has over 130 distinct tables alone. The 80-character line length limitation means that related data must be spread out over many tables; this, along with the large file sizes (can exceed 10,000 lines), make the data difficult to understand.

For a developer to include new parameters in a UCI file, an existing table must be found with sufficient room at its end or a new table must be defined. Either method requires nontrivial modifications to HSPF to process the revised or new tables. Making changes to the size of existing data in a table breaks the compatibility for existing UCI files. A real understanding of these tables requires a detailed knowledge of the HSPF code that is possessed by few users.

Even with graphical user interface (GUI) tools such as BASINS (Better Assessment Science Integrating point & Non-point Sources), creating the UCI file for a new watershed is a complex and difficult process. The UCI file is complicated for modern software to interact with, and running software tools to optimize the calibration and perform sensitivity and uncertainty analysis is difficult.

When HSPF was created, the small amount of available memory on computers required Fortran COMMON blocks to "reuse" memory quickly. Over the course of an HSPF simulation run, a single COMMON block memory location could hold different data types (integer, single- and double-precision floats, and characters) for many differently named variables. This COMMON block architecture makes the code much harder to understand and overly burdensome to safely modify. A significant portion of the HSPF code is required to manage the memory usage.

Fortran-77 static array sizes result in limitations on virtually everything in HSPF. A few examples of fixed limits include a maximum model size (the number of sequence operations), a

maximum number of reach outlets, a maximum number of rows and columns in an F-Table (function table), and a maximum number of water-quality constituents. Granted, however, some constricting limits have been increased over time. Because of the static sizes, land use, translation factors, and most of the simulation parameters are fixed at the beginning of the run. In some cases, the complicated Special Actions Module can be used to mimic these real-world varying conditions, but the complexity of this module significantly increases the time to develop the input file and considerably slows down the model execution times.

Time-series data are stored in WDM and other legacy file structures in HSPF. The 32-bit computer architecture that is used in WDM files limits the size of a WDM file. WDM files are poorly supported by Commercial Off-The-Shelf (COTS) software and, therefore, pre- and postprocessing of time-series data can be cumbersome. Currently, the process typically relies on in-house solutions to augment the small number of compatible software products in the public domain that are becoming incompatible with today's operating systems. Even with external support utilities, WDM files are difficult to use and provide limited functionality compared to modern solutions.

DESIGN OBJECTIVES FOR THE NEW HSPF, HSP^2

The overall goal of this project is to mitigate the aforementioned challenges that are currently facing the HSPF model so that it will continue to be a premiere watershed model into the foreseeable future. The major objectives to accomplish this goal and create the HSP^2 (Hydrologic Simulation – Python) include the following:

- Retain all current HSPF functionality from the engineering user's point of view and provide a migration path from legacy applications; however, internal functionality that is invisible to the user, such as the Fortran-77 COMMON block, may be modified or deleted as necessary. Documentation within the new HSP^2 code is used to transparently show the translation path between HSPF and HSP^2.
- Elevate the engineering code out of the nonengineering code to make the engineering and science in the model clear. Because the user should not be lost in the complexity of the memory management and the data input and output aspects of the code, HSP^2 should implement the HSPF algorithms (as found in the HSPF documentation) as clearly and directly as possible.
- Restructure HSPF for maintainability, remove all fixed limits (e.g., number of operations, three-digit references, and constant parameters), and remove naming constraints for model data and time-series datasets. Automatic conversion from an existing UCI file will preserve the original names, but names can be changed at any time in the HSP^2 data file. Names for operations, parameters, and time-series datasets can be an arbitrary sequence of ASCII characters; however, to avoid problems with other tools, languages, and operating systems, using names that are acceptable to most computer languages and under 64 characters long is preferred.
- Provide users with greater capability such as the ability to easily add new types of data, parameters, and code modules. Tools should be created so that the user can determine what changes were made from a previous model run and automatically create a minimal operation

sequence to optimize the run time. The user should also have a tool that provides a simpler, but more powerful, replacement for the current Special Actions Module to allow changes in land use and other parameters over a simulation without having a significant performance penalty. Modern data visualization tools should be available, such as a graphing utility that can plot a 25-year time series that the user can pan and zoom down to the hourly data if needed.

- Work with other external water resources modeling software and tools. It should also provide easy compatibility with engineering software (such as DAKOTA) to provide sensitivity studies, parameter optimization, and uncertainty analysis. A simple method should be available to pass a set of new values for the simulation without the complexity of parsing and then rewriting the entire equivalent of the UCI file. In addition, the code should work with open-source Geographic Information System (GIS) software.
- Be independent of operating system and hardware, and it should maintain, or even improve, execution time. A Message Passing Interface (MPI) based parallel version of HSP^2 should be available for use in clusters. HSP^2 should be compatible with multiple cores and graphical processing units (GPUs) for acceleration, but it need not require specific hardware.
- Be open source and freely distributed over the web.

SOLUTION

The solution derived for this project consists of the following two primary tasks:

1. Convert the code to a modern, widely accepted, open-source, high-performance computing code
2. Convert the model input and output files to modern, widely accepted, open-source, data model, library, and binary file format.

NEW PROGRAMMING LANGUAGE

The first task was to select a computer language from a group that were considered before the project began. A description of each language is provided below:

- **Fortran**—It has significantly evolved since FORTRAN-77. The latest standard is Fortran 2015. Fortran is used to create high-performance numerical libraries. Modern versions of Fortran support both dynamic and static arrays and code vectorization. Unfortunately, fewer schools are teaching Fortran to their students, and it is problematic because it requires backward compatibility to older Fortan statements, which makes the language more complex than necessary.
- **C, C++, Java, C#, Objective C**—These C (and C-derived) languages are in wide use and are well supported. They are used heavily by software professionals to create high-performance numerical codes; however, they are generally too complex for scientists and engineers for simple daily use.

- **Pascal, Modula**—Although they were popular in the 1970–1990 decades, these languages are basically obsolete today.
- **Go, Haskell, F#**—These languages are not widely known, especially to engineers and scientists.
- **Julia**—This would be a good choice except that it is too new, not yet well known, and does not currently have extensive support libraries.
- **Python**—As an interpreted, object-oriented, high-level programming language with dynamic semantics, Python has become one of the most popular open-source languages. It is also one of the easiest computer languages to learn.

Python was chosen as the new language for HSP^2. With its scientific libraries, Python provides MATLAB-like capabilities and performance. Python's high-level, built-in data structures, combined with dynamic typing and binding, make it attractive for quick, day-to-day engineering exploratory codes, rapid development, and as a scripting or glue language to connect existing components.

Because Python is an interpreted and dynamically typed programming language, the execution of Python code can be slow compared to compiled, statically typed programming languages, such as C and Fortran. However, projects such as Numba have now allowed us to overcome this challenge. Numba is a just-in-time specializing compiler that compiles annotated Python and Numpy code to LLVM. It uses the LLVM compilation framework to parse, compile, and optimize assembly code in the same manner as the fastest compiled languages, such as C and Fortran. Although Numba uses powerful libraries beneath Python for performance, the code that a developer writes is always pure Python, which makes it easier to author, debug, verify, and maintain. Some other strengths of Python include the following:

- Open-source, cross-platform, and functional on a wide number of platforms, including supercomputers and other high-performance computing environments
- Strong position in scientific computing with a large community of users, easy-to-find help, and well documented
- Extensive scientific libraries and analysis packages
- Great performance because of the close integration with time-tested and highly optimized codes written in C and Fortran
- Strong support for parallel processing with processes and threads, MPI, and GPU computing
- Support for interactive work, including execution, visualization, and debugging.

Because of the scientific Python library stack, Python is sometimes called the "poor man's MATLAB." The scientific library stack (visit *SciPY.org* on the Internet) includes Numpy, SciPy, Matplotlib, IPython, Sympy, and Pandas. Other useful libraries include PyTables, H5py, Bokeh, Blaze, and mpi4py.

Python is expressive. Some of the following examples will illustrate how it can "elevate" the engineering code in HSPF from the other code (which represents a vast majority of HSPF code).

In the first example, the HSPF documentation for the air temperature elevation difference (Equation 1 in the ATEMP routine in the HPERAIR section) is as follows:

AIRTMP = GATMP – LAPS * ELDAT

where GATMP is the gage temperature; LAPS is the 24-element array that represents the hourly dry-lapse values, which are overwritten with a wet-lapse value during precipitation events; ELDAT is the elevation difference between the segment being processed and the station that is providing the gage temperature; and AIRTMP is the calculated air temperature. HSPF can perform this calculation in either English or metric units as specified in the UCI file, so units are not shown here.

The HSPF module HPERAIR.FOR implements this equation in one line that is identical to the above (line 145 in HSPF Version 12.2); however, it also requires a large amount of explicit memory management to obtain and save the data. The following required code is from different sections in HPERAIR.FOR:

```
C   process value in table - type elev-diff
  TBNO= 5
  TBSB= 1
  NVAL= 2
  CALL RTABLE(TBNO,TBSB,NVAL,UUNITS,
  M      RVAL)
C
  ELDAT = RVAL(1)
  AIRTMP= RVAL(2)

C    read air temperature at the gage from inpad
   REQFG= 2
   TSNAM= 'GATMP '
   CALL HREQTS (GATFP,IVL1,REQFG,MESSU,MSGFL,DATIM,OPTYP,LSNO,
  I       TSNAM,TSSUB,SECNAM,MSECNM,OPFGNM,FLGVAL,
  O        GATMP)

 IF (AIRTFP .GE. 1) THEN
   PAD(AIRTFP + IVL1)= AIRTMP
  END IF
```

The memory management for this one-line equation spans lines 60–67, lines 137–141, and lines 280–285 in the HPERAIR module as shown above. In addition, there is the explicit loop over all of the time intervals in the entire simulation in a higher level calling program that is easily missed.

The entire corresponding HSP^2 code looks as follows:

```
ts['AIRTMP'] = ts['GATMP'] – laps * ui['ELDAT']
```

The HSP^2 coding convention defines two Python dictionaries (ts and ui) to contain time-series datasets and user model data, respectively, and are available to every module. These dictionaries are initialized by the main program to provide only the specific data for the current operational sequence operation. HSP^2 maintains the Fortran naming to provide correspondence between the two codes, but HSP^2 uses uppercase to indicate time-series datasets and user (model) data that are external to the module and lowercase for values that are local to the module. Therefore, the expression ts['GATMP'] retrieves the entire named times series, and ui['ELDAT'] retrieves the current segment's ELDAT data. Because the "laps" array name is lowercased, it indicates that it was calculated in the lines above this example and is not available to external modules. The expression ts['AIRTMP'] makes the resulting computed time series automatically available to the other modules in the same operational sequence operation and also makes it potentially available for persistent storage if it is selected by the user in the appropriate SAVE table. In addition, this entire expression is automatically vectorized by Numpy (or Numba if used) to perform this calculation over the entire simulation time without the need for an explicit loop. The vectorized code can automatically use multiple Central Processing Unit cores in parallel. In both cases (using either Numpy or Numba), the resulting expression is transparently converted from Python to efficient machine language.

The actual HSP^2 code also has a comment beginning with "#$" (at the end of the code line) to show the corresponding Fortran lines (HSPF Version 12.2) that are implemented by the Python line. Therefore, the final HSP^2 line looks as follows:

```
ts['AIRTMP'] = ts['GATMP'] – laps * ui['ELDAT']  #$60-67,137-141,145,280-285
```

This allows the automatic creation of a cross index to be automatically created between HSPF and HSP^2 code lines.

In the last example, consider the following two lines from the HSP^2 HPERWAT module:

```
PETADJ = (1.0 - FOREST) * (1.0 – SNOCOV) + FOREST
PETADJ[(AIRTMP < PETMAX) & (PETADJ > 0.5)] = 0.5
```

In the first line, FOREST is a fixed-model parameter and SNOCOV is a time series over the entire simulation, which was calculated a few lines earlier in that module. As a result, this is a fully vectorized expression that calculates the array PETADJ over the entire simulation period without any explicit looping and can take advantage of multiple cores because each time interval is independent. There was no previous declaration of PETADJ because Python can determine that it will be a double precession array that is the same size as SNOCOV and will automatically allocate a Numpy array of the correct type and size.

The second line computes a Boolean array for each time interval over the entire simulation. The Boolean array is only true for any time interval in which the air temperature is less than the model parameter PETMAX and the previously calculated value of PETADJ is also greater than 0.5. Whenever the Boolean array is true, the final value of PETADJ is set to 0.5; otherwise, it is left with the value computed in the previous line.

These two lines can be expressed in one line:

PETADJ = where ((AIRTMP < PETMAX) & (PETADJ > 0.5), 0.5, (1.0 - FOREST) × (1.0 - SNOCOV) + FOREST)

However, this line is not considered as clear. The objective of HSP^2 is the clarity of implementing the engineering algorithm and not the smallest line count.

NEW FILE FORMAT

The second task was to select a file format that is more compatible with today's engineering software than the WDM files. Multiple different formats were considered but the HDF5 (Hierarchial Data Format, version 5) file format was ultimately selected.

HDF5 is standard for large-scale, scientific and engineering data storage. The HDF5 technology suite is designed to organize, store, discover, access, analyze, share, and preserve diverse and complex data in continuously evolving, heterogeneous computing and storage environments. HDF5 supports all types of data that are stored digitally, regardless of origin or size, including the GIS data, pictures and images, and data from the WDM and UCI files that are common to HSPF applications. HDF5 includes tools and applications for managing, manipulating, viewing, and analyzing the data in the collection. The HDF5 data model, file format, application program interface, library, and tools are open and distributed without charge under a liberal, Berkeley Software Distribution-like license for general use.

The HDF Group (at *www.hdfgroup.org*) maintains and extends HDF5, and it provides a free viewer for HDF files, which is called HDFView. Figure 1 is a screen capture from HDFView showing an example of an F-Table in an HSP^2 file. The left pane clearly shows the hierarchical file structure within the single physical file.

Reading and writing HDF5 files from Python (via Pandas) or from MATLAB (and most other tools) is quite simple. For example, from Python (Pandas library), the code to read the FTable shown above is as follows:

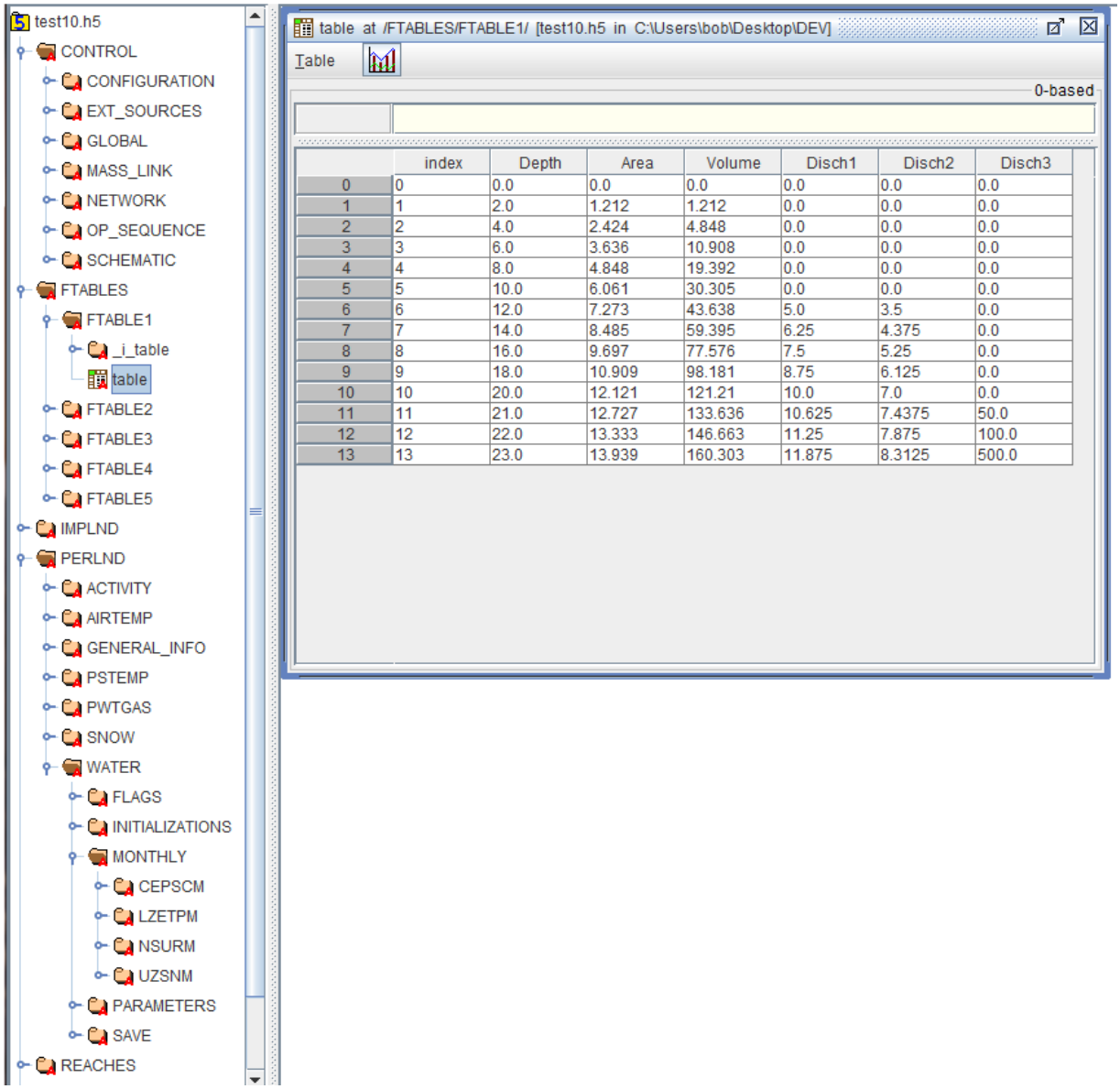
ftable = pandas.read_hdf("test10.h5", "/FTABLES/FTABLE1")

where "test10.h5" is the name of the HDF5 file, and '/FTABLES/FTABLE1' is the linux-like path to the dataset. The corresponding MATLAB code is:

ftable = h5read("test10.h5', "/FTABLE/FTABLE1")

which is essentially identical. Both reads can take optional arguments to specify the start, end, count, and stride for the returned data. Data are also written on one line in either Python or MATLAB.

HDF5 files allow arbitrary user metadata (called HDF5 attributes) to be associated with any dataset or group (directory or folder) as key, value pairs. Therefore, annotating the data to indicate its units, origin, and creation date is easy. The limit is 65 kilobytes per dataset for the total of all metadata for the dataset. This metadata is easy for the code to examine and use.



	index	Depth	Area	Volume	Disch1	Disch2	Disch3
0	0	0.0	0.0	0.0	0.0	0.0	0.0
1	1	2.0	1.212	1.212	0.0	0.0	0.0
2	2	4.0	2.424	4.848	0.0	0.0	0.0
3	3	6.0	3.636	10.908	0.0	0.0	0.0
4	4	8.0	4.848	19.392	0.0	0.0	0.0
5	5	10.0	6.061	30.305	0.0	0.0	0.0
6	6	12.0	7.273	43.638	5.0	3.5	0.0
7	7	14.0	8.485	59.395	6.25	4.375	0.0
8	8	16.0	9.697	77.576	7.5	5.25	0.0
9	9	18.0	10.909	98.181	8.75	6.125	0.0
10	10	20.0	12.121	121.21	10.0	7.0	0.0
11	11	21.0	12.727	133.636	10.625	7.4375	50.0
12	12	22.0	13.333	146.663	11.25	7.875	100.0
13	13	23.0	13.939	160.303	11.875	8.3125	500.0

Figure 1. Example of HDF5 File Data for HSP[2].

HSP^2 TODAY

Figure 2 shows the high-level design of HSP^2.

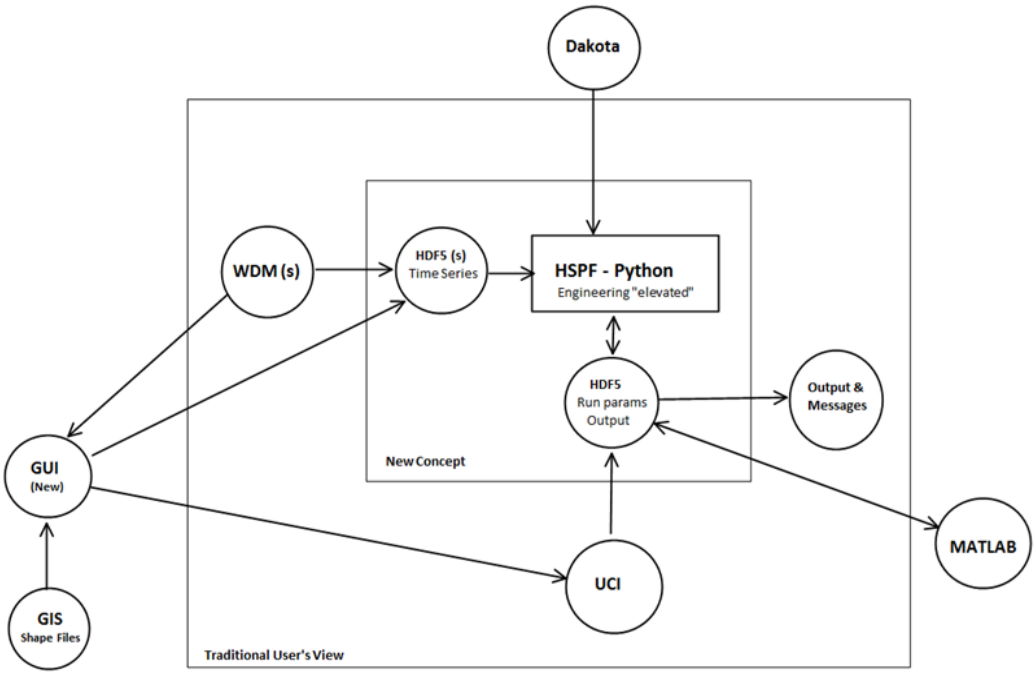


Figure 2. Conceptual HSP^2 Design

The core module is HSP^2, and it is significantly smaller and clearer than HSPF. HSP^2 uses the HDF5 to store all data for a simulation including what had been the UCI table data and the WDM time series data. Utilities for legacy HSPF models convert WDM and UCI files to HDF5. HSP^2 is designed to work with other tools, such as MATLAB and DAKOTA, to perform auxiliary calculations, sensitivity studies, parameter optimization, uncertainty quantification, and for data visualization and reporting. A future GUI is planned to make creating a new model much easier than creating the model via a UCI file.

The project has just completed converting the code for the hydrologic and hydraulic operation modules. This project has provided a great opportunity to further perform HSPF code verification and when appropriate, optimize legacy routines. The converted code has been tested against HSPF's suite of test runs and has shown agreement and similar execution times when using the Numba compiler.

The project has a series of seven interactive tutorials to explain how to use HSP^2. These tutorials begin with a basic overview and proceed to demonstrate how to run HSP^2; create plots and reports; understand HDF5; work with HDF5 data; and, finally, how users can add new data, new modules, and perform advanced simulation techniques. One of the tutorials demonstrates how to

set the units metadata (attribute) for a dataset into an HDF5 file and then to read this attribute in the code to determine if a units conversion is required before using the data.

These tutorials will be available (and free) when HSP^2 is released for general use, and more tutorials are planned for the future. The tutorials are based on Jupyter Notebooks (formerly called the IPython Notebooks), which allow users to run the demonstrated features and to modify and play with the demonstrations to really learn how HSP^2 works.

The project has created tools for automatically calculating an optimal operational sequence table considering any changes to the HDF5 file since the previous run. HSP^2 has also created some tools to view, plot, and report data from PERLND, IMPLND, and RCHRES segments.

HSP^2 WEAKNESSES

A few features of the hydrology modules are either incomplete or not fully tested. For example, the PERLND high-water table-specific codes have been converted but have not yet been tested, and the SHADE module has not yet been completed.

HSP^2 will require some new learning. Users will need to learn about HDF5 files and the new data organization. The user will need to learn Python to write new modules or modify existing modules.

HSP^2 TODO, FINDINGS, AND NEXT STEPS

The next steps are to continue to verify the accuracy of the converted code against more complex legacy applications and improve upon execution times by incorporating an intelligent network change detection tool that will determine what aspects of the application need to be resimulated based on changes made in the input file. The continued, rapid evolution of Numba will require returning to update HSP^2 to improve performance using this tool.

An immediate goal is to package HSP^2 for simple installation on the user's own machines. Another possibility is to make the code available through a cloud (Wakari) so that users do not need to install any software on their own machines, but they could use this free service to run HSP^2 . However, users might prefer a paid account to load large amounts of their data.

Long-term activities will include converting the water-quality algorithms and making the parameters, translation factors, and land use more dynamic within the simulation time period. Ultimately, all functionality of HSPF will be available but sometimes in a slightly different manner.

Some of the other long-term goals include the following:

- More data visualization tools
- A future GUI to directly create HSP^2 models without pervious UCI and WDM files

- An MPI-enabled version to use in clusters
- Tighter GIS integration
- Using satellite, radar, and Light Detection and Ranging (LiDAR) data directly by HSP^2
- Integrating data input from hydrologists by using mobile devices.

IMPACTS TO FUTURE DEVELOPMENT AND APPLICATION OF HPSF IN WATER RESOURCES

Converting HSPF to a more widely used language and manageable file storage will dramatically increase the number of water resources engineers who will be able to improve the science and engineering that drives the application of HSPF. The improved capability to link HSPF to external modeling and engineering software will provide for developing more efficient and defensible applications to use in improving the nation's water resources quality.

CONCLUSION

We believe that these early results and the detailed plans that were described will allow HSPF to continue to be a premiere watershed model into the foreseeable future.