Domain-specific Metaware for Hydrologic Applications

Daniel Andresen, Mitchell Neilsen, Gurdip Singh, Prasanta Kalita [a,a,a,b,1]

[a] *Department of Computing and Information Sciences*
234 Nichols Hall, Kansas State University

[b] *Department of Agricultural Engineering*
University of Illinois, Urbana-Champaign

Abstract

The DHARMA domain-specific middleware system is intended to allow hydrologic field engineers to tackle water-management problems on a scale previously impossible without sophisticated computational management systems. DHARMA provides automatic data acquisition via the Internet; data fusion from online, local, and cached resources; smart caching of intermediate results; parallel process execution; automatic transformation and piping of data between different hydrologic simulation models; and interfaces with existing metacomputing systems. Our target watershed model, WEPP, is limited to very small watersheds with current computer technology. A revolutionary change in hydrologic modeling on the watershed scale will be brought about by applying various watershed models to large watersheds, such as the Lake Decatur Watershed which covers 925 square miles. Even with unlimited computing resources, the current version of WEPP cannot be directly applied to model the Lake Decatur Watershed because the current implementation of WEPP is limited to a maximum of 75 hillslopes. To address this issue, we have now integrated support for multiple simulation models in our software; for example, we can use WEPP to model individual hillslopes and another model, such as SITES, to model the connecting channels (reaches) and dams (structures), playing to the strengths of both models. In this paper we discuss the evolving software architecture of DHARMA and its interaction with the Web and Grid infrastructures.

Key words: middleware, hydrologic simulation, hydrology, distributed computing, XML

* Corresponding author
Email address: {`dan, neilsen, singh`}`@cis.ksu.edu`, `pkalita@sugar.age.uiuc.edu` (Daniel Andresen, Mitchell Neilsen, Gurdip Singh, Prasanta Kalita).

1 Introduction

The DHARMA project attacks two fundamental problems in today's hydrologic research environment. First, the inability of researchers to perform even simple investigations due to the inaccessibility and essential difficulty in acquiring and utilizing the necessary data. Often the data and simulation models are available via the Internet, but through a combination of obscurity, incompatibility, and inefficiency are essentially unusable. Second, local computing resources are inadequate to support modeling systems at the level desired. DHARMA addresses these problems through significantly easier access to the computational power and data acquisition capabilities of the Internet. The objective of the DHARMA project is to expand the applicability of the WEPP (Water Erosion Prediction Project) model and the SITES (Water Resource Site Analysis) model to large watersheds, specifically applying them to the 925 sq. mile Lake Decatur watershed in Illinois [7,9].

The DHARMA domain-specific middleware system is intended to allow hydrologic field engineers to tackle water-management problems on a scale previously impossible without sophisticated computational management systems. The system is intended for use not only by hydrologic researchers, but also by engineers working in the field on a daily basis. These engineers typically have older, inadequate local computing power for the increasingly complex models required.

DHARMA has the following major points of functionality: automatic data acquisition of geotemporal climatic data from online databases; data merging necessary for the computation to occur, from online, local, and cached resources; automatic transformation and piping of data between heterogeneous hydrologic simulations; smart caching of intermediate results to allow for reuse in future simulation cycles; task graph acquisition from the UI layer, and optimization through application-specific knowledge and dynamic results caching strategies; and, in the future, interfacing with computing resource managers, such as Globus and Condor [4,8].

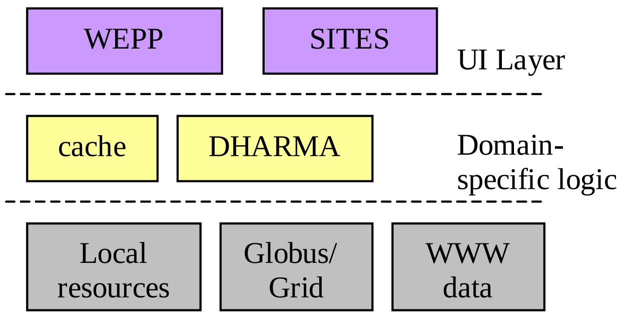


Fig. 1. DHARMA block diagram.

Major technical problems to be solved include: How do we automatically acquire the necessary data? How do we know which items can be cached? How can results be passed from simulation to simulation with dissimilar data formats? What Document Type Definition's (DTD) must be developed for information within this domain? What domain-specific task graph heuristics are needed for efficient computation?

To address these issues, we:

- use the eXtensible Markup Language (XML) and DTDs as a lingua franca between simulations, remote data sources, and DHARMA components.
- realize our primary goal is flexibility and ease in data acquisition and data transfer between simulations, which cannot be sacrificed for utility-limiting performance enhancements.
- allow DHARMA to have multiple interfaces to existing metacomputing systems, and use DHARMA to package up the computation into a tidy set for computation by these systems.
- develop heuristics for scheduling task graphs, interfaces for the various resources, and application-specific caching techniques.

The paper is organized as follows: Section 2 gives the background needed on WEPP, SITES and XML. Section 3 discusses the DHARMA software architecture and various implementation issues. Section 4 presents some preliminary experimental results, and Section 5 discusses related work and conclusions, as well as future directions.

2 Background

WEPP: The WEPP watershed model is a continuous simulation processes-based model which represents new soil erosion prediction technology based on fundamentals of stochastic weather generation, infiltration theory, hydrology, soil physics, plant science, hydraulics, and erosion mechanics [7]. Currently, all watershed scale hydrologic models are based on empirical relationship and seldom include process interactions among soils, hydrologic, plant, and atmospheric processes. Currently, WEPP is the only model which accounts for soils, sediment transport, runoff, channel flow, plant growth, decomposition, snow melt, freeze-thaw effects, and climatic conditions. The applicability of WEPP, however, is limited to very small watersheds (up to few hundred acres) due to currently available computer technology in handling input data requirements. If the WEPP model can be enhanced for application over larger watersheds, it will bring a revolutionary change in hydrologic modeling on the watershed scale as it will be the only such model which will provide accurate predictions and evaluate effects of alternative watershed management practices on watershed water quality.

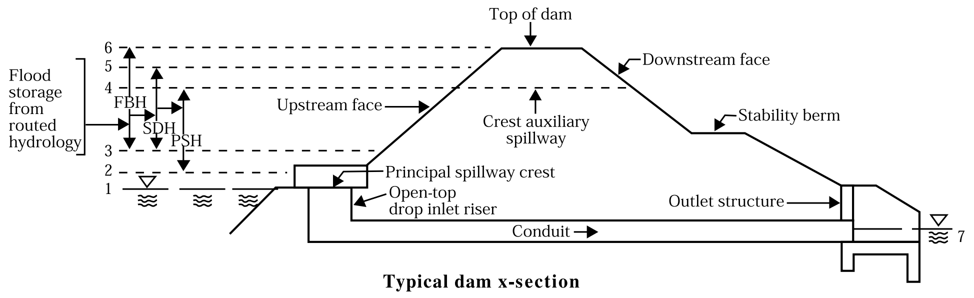


Fig. 2. Flood control structure components.

SITES: Vegetated earth (soil and rock) auxiliary or emergency spillways have been used extensively on reservoirs and ponds for flood control within the United States. These spillways generally consist of a trapezoidal channel cut through natural materials and vegetated as appropriate for the local area [12]. The USDA Soil Conservation Service (SCS) has constructed approximately 23,000 structures using this type of spillway [3]. Despite their widespread use, the processes by which these spillways erode during extreme events, and their effect on the environment, are only imperfectly understood. The SITES software has been used extensively by the SCS for the design and analysis of flood control structures since 1971. The SITES software was developed using Fortran, and consists of several thousand lines of source code [13]. SITES software evolved from the DAMS2 software, which has been used extensively by the Soil Conservation Service for the design and analysis of flood control structures since 1971. While SITES retains all of the capabilities of the DAMS2 software, it also adds a number of new features to better predict erosion. For example, it has been found that erosion of vegetated earth spillways can be divided into three phases: vegetal cover failure, concentrated flow erosion, and headcut advance [12]. Hydraulic and hydrologic analysis of structures (dams) designed using this new NRCS criterion for spillways and flood routing can be performed using SITES.

Input to SITES includes information about the structure being analyzed and information required to generate the inflow hydrograph(s) – functions that represent the amount of water flowing into the reservoir created by the structure (see Figure 2). Hydrographs, or the rainfall/watershed information required to generate them, may be derived from historical data for a given flow event, or generated automatically from standard design tables. Execution of SITES generates outflow hydrographs and other structural information, including predicted earth spillway erosion (Figure 3).

XML: We have chosen an XML-based data format for several reasons. Using XML leverages a huge wave of support from industry in terms of tools and source code. However, the primary advantage is the combination of device- and OS-independent data transfer, with the ability to retain its semantic structure [17]. This is a critical advantage over text files or HTML. Furthermore,

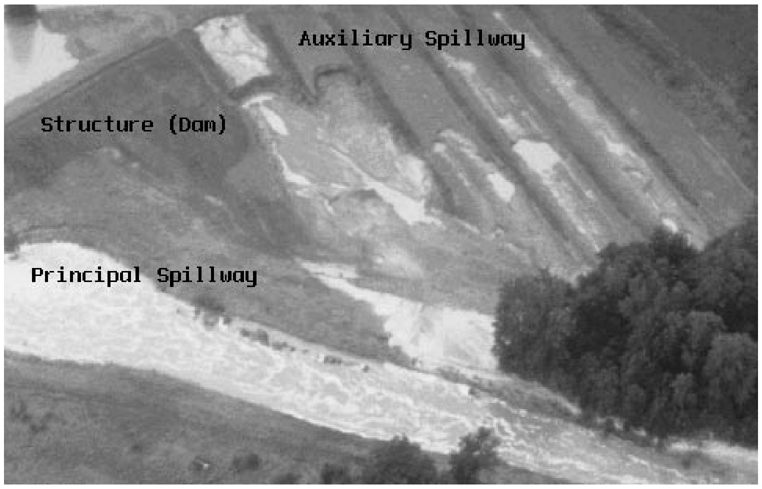


Fig. 3. Spillway erosion. (Image courtesy USDA)

XML allows a single information source to serve multiple applications. For example, our weather data server can easily be used by other weather-based simulations.

Alternative technologies to XML include CORBA, sockets, NetCDF, Java RMI, and other custom networking protocols. CORBA typically is more complex for developers, but provides an object-oriented interface [14,15]. NetCDF is designed for the exchange of scientific information, and offers an intrinsic Web interface, but has failed to achieve the same level of support across the entire Web industry as XML [11]. We hope to provide a gateway between our system and NetCDF data sources, due to the significant amount of scientific data available in that format. Java RMI provides an excellent mechanism for communicating between Java processes, but lacks the flexibility and genericity of XML [16]. Through the use of industry-standard information exchange mechanisms, we can provide for future expansion and compatibility with online information sources.

3 The DHARMA Architecture

We now give an overview of the DHARMA architecture, as illustrated in Figure 4. Our architecture generally follows the standard n-tier model with a user interface, task logic, and, differing from the standard three-tier model, a cluster-based computation/data layer. This includes the multiple user interfaces, the job manager, automated data acquisition, and execution engine.

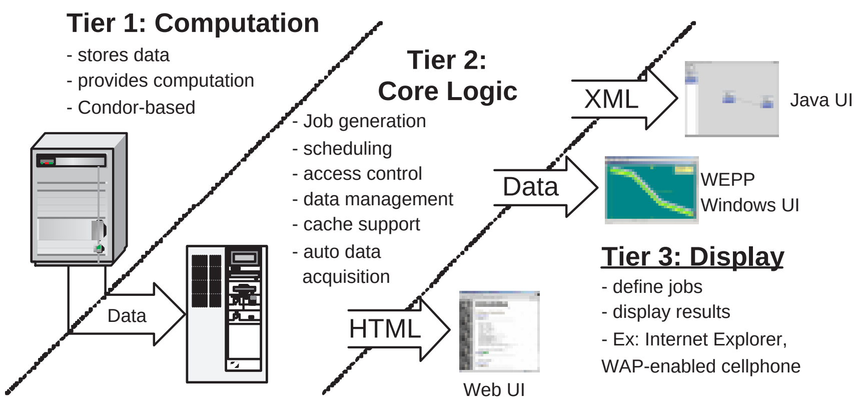


Fig. 4. DHARMA layers diagram.

3.1 The DHARMA user interfaces

There are currently four possible interfaces that can be used to run a simulation in our system. The four interfaces are the WEPP Windows Interface, the SITES Integrated Development Environment, the Dharma Web Interface and the Drag and Drop Interface. Each of these interfaces retrieve data from the user and interact with the middleware in a different manner. The primary interface that will be used when the project is fully functional will be the Drag and Drop Interface. The other three interfaces will still be supported at the end of the project.

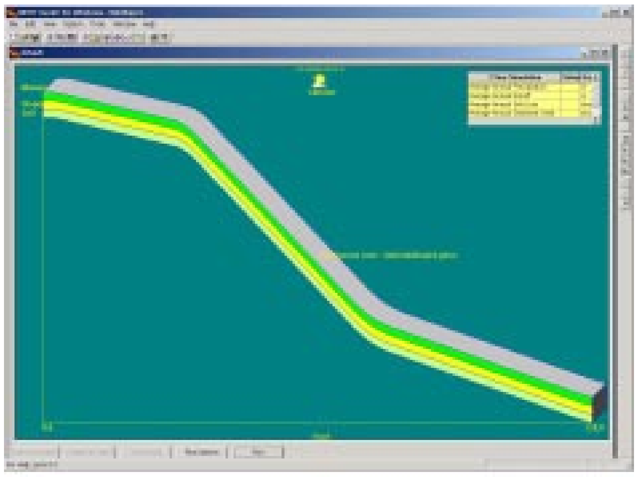


Fig. 5. The WEPP Windows Interface.

WEPP Windows Interface: The first of the four interfaces that can be used to interact with our system is the pre-existing WEPP Windows Interface (Figure 5). The designers of the WEPP simulation created a windows application that allows the users to run soil loss simulations on hill slopes and channels. This interface allows the user to input some basic attributes of the slope or channel that the simulation will model on. These inputs are then used to generate the files necessary to run the WEPP simulation.

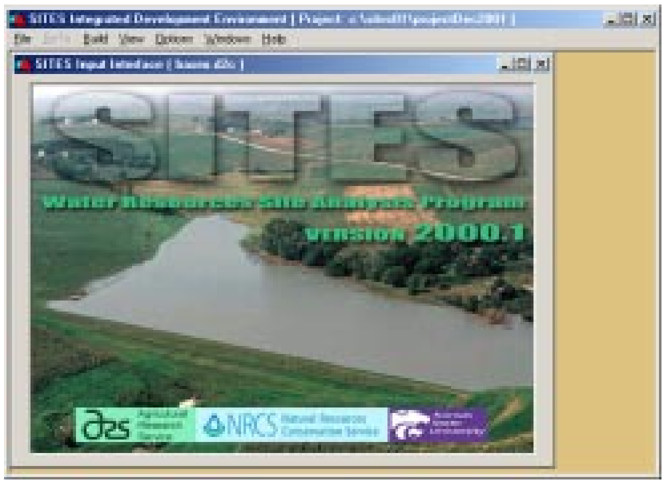


Fig. 6. SITES Integrated Development Environment.

SITES Integrated Development Environment: The SITES Integrated Development Environment (SITES 2000.1) is designed to allow the user to conveniently examine the impact of flood or design changes on the performance of the downstream (design) structure. A typical watershed schematic constructed using the SITES IDE is shown below in Figure 6. The environment also guides the user through the required input and assists in the interpretation of the output. It is designed to take advantage of the unique features of the SITES computational routine while operating in any contemporary Microsoft Windows environment.

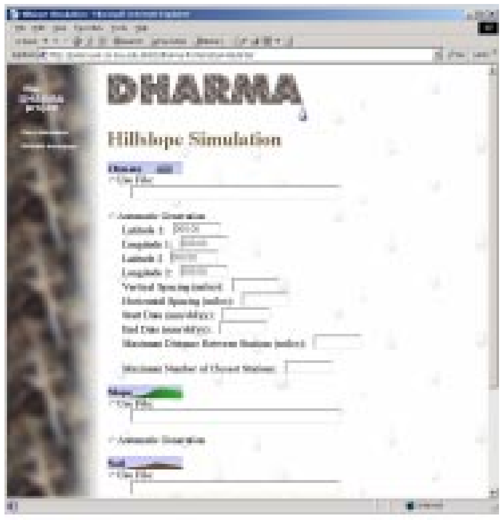
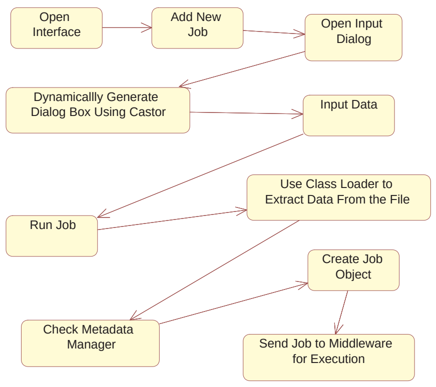


Fig. 7. Dharma Web Interface.

Dharma Web Interface: The third of the four interfaces that are at the disposal of the users of the Dharma system is the Dharma Web Interface (DWI). The DWI uses HTML, XML, and Java Servlets (Figure 7). The only models that the DWI currently supports are the WEPP and SITES simulation models. However, adding new simulations to the interface is relatively painless.

The overall structure of this interface is fairly straightforward. A user is presented with a dynamically list of simulations from which one can be selected. Depending on the simulation selected, our system dynamically generates a form from an XML document that describes the inputs necessary for that simulation. The user then enters the data and clicks submit. The submit button launches a Java Web Start application that checks to see if the Dharma servers have the data cached already. The application then bundles up the job and sends it to the middleware. When the user is inputting data, they are offered two options for each input file. They can submit a file or they can ask to have the file automatically generated. Currently, the user is only allowed to automatically generate climate files.

We chose to use a Java servlet solution to implement the data gathering part of the interface. We did this because it is relatively easy and clean to get the data via a form and servlet. The servlet also allowed us to create a clean and aesthetically pleasing interface using HTML and XML style sheets. The servlets are also easy to write and extend later on. With a servlet solution, if anyone needs to modify the web interface in the future they can simply write a function or add an XML document.



Open Interface
Add New Job
Open Input Dialog
Dynamicallly Generate Dialog Box Using Castor
Input Data
Use Class Loader to Extract Data From the File
Run Job
Create Job Object
Check Metadata Manager
Send Job to Middleware for Execution

Fig. 8. Dharma GUI process.

Java Web Start was used to gather information about the files on the client. The Dharma system caches files that it receives so that they do not have to be later uploaded again. In order to ensure that we were not uploading files that were already on the servers, it was necessary to gather information about the files while they were still on the client side. Java Web Start was selected to do this because with Java Web Start we were able to launch a normal Java application on the client. This application was able to access the local file system on the client with the users permission. In order for the program to run, the users of the system must install the Java Web Start plugin and the Java plugin.

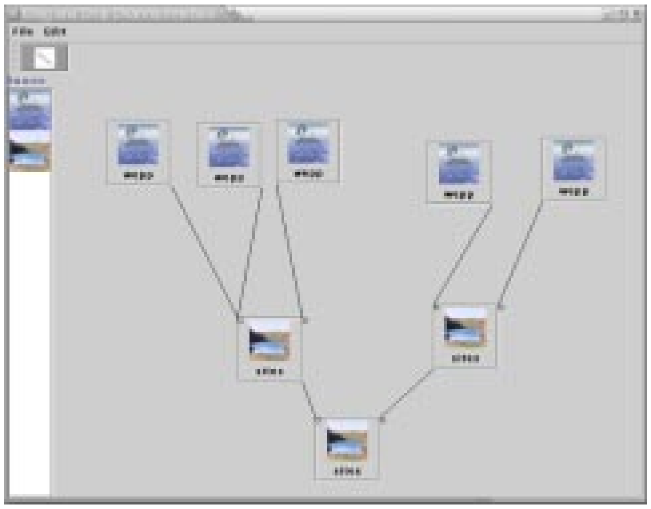


Fig. 9. Dharma Advanced Interface - Drag and Drop Interface with SITES and WEPP simulations.

Drag and Drop User Interface The Drag and Drop Interface (DND) is an interactive interface that allows the user to create a diagram that visually represents a simulation run (Figure 9). The interface is deployed on client machines using Java Web Start and written in Java.

The user is allowed to place buttons on the screen that represent watershed elements, such as hillslopes or channels. When a user clicks on the button for a simulation a dialog box will pop up that allows the user to input all of the data necessary to select and run a simulation. These dialog boxes are dynamically created from XML documents that describe the necessary inputs for a simulation. After a user has filled in the information on a simulation in the dialog box, an input icon will appear on the screen for each input into the simulation. Users are also allowed to chain simulations together and outputs are automatically fed into the dependent simulations, with the data formats converted as necessary.

The dynamic generation of XML Dialogs is accomplished by using an XML parser written for this project. The parser takes in an XML file name and return a JDialog dialog box. This is generic enough that it could be used in other projects needing the same type of functionality. The dynamic generation of dialog boxes will allow the system to be easily extended. All that will have to be added to the interface is an XML document detailing the necessary inputs for a simulation.

When a user starts a job, all simulations on the diagram will be executed. The interface grabs all of the data for the simulations on the screen and checks with the Metadata Manager (described in Section 3.3) to see if the files exist on

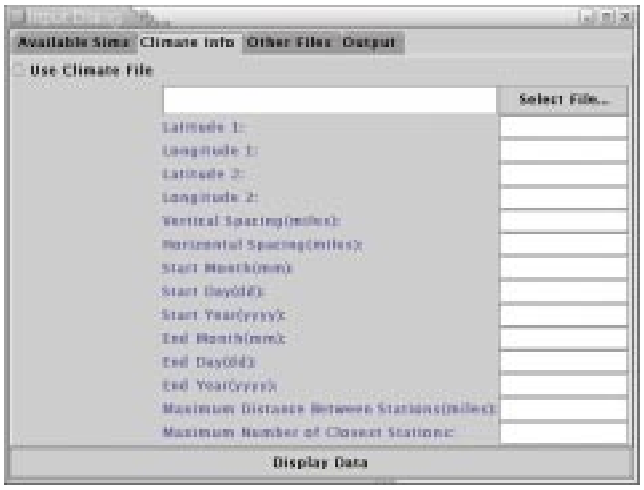


Fig. 10. Sample input dialog for WEPP simulation.

the Dharma servers. The application then bundles all of the information up into SOAP objects. These objects represent a simulation run. These objects are then sent to the middleware for processing and execution. If there are multiple simulations on the same diagram that are dependent on each other, the dependencies will be sent to the middleware as well so that scheduling can be done.

The Drag and Drop interface is a highly extensible user interface that allows the Dharma project to function for many different simulations. The only modifications that are necessary to extend the interface are an XML document detailing the inputs necessary for the creation of a dialog box for that simulation. Furthermore due to the utilization of Java Web Start it will not be necessary for users to acquire new releases of the software. With Java Web Start new versions are automatically downloaded every time there is a revision to the software. This process is completely transparent to the users of the system.

Simulation Handling Currently all handling of simulations is generic and ignorant of what a simulation needs for its inputs. The way the system currently runs, all that is required to add a simulation to the interface is an XML document specifying the inputs that are necessary for the new simulation model.

When the user clicks on a button on the workspace, a dialog box is automatically generated from XML using Castor and Swing. If the button has already been clicked then the box that was previously created is made visible again. The buttons in the workspace are custom buttons that were created specifically for the Dharma project. They store all of the data that is entered in the dialog box as well as a vector that contains information about dependencies.

After a user has entered all of the necessary information on a simulation, they are now ready to run their job. This is done by selecting the "Run Job" option from the file menu. When the user runs a job, the interface runs through each button on the workspace and prepares it for submission.

It is sometimes necessary for the Metadata Manager to know certain information contained in the files that are being sent to it. In order to maintain the generic nature of the interface, we implemented a class loader that allows the interface to get the information from the files without actually knowing anything about the files. We store classes that can be used to extract the data from the files on the Dharma servers. When the UI is getting ready to check with the Metadata Manager to see if it should send a file, it downloads this class and calls the method that extracts the data. Once the data is extracted to the file it is added to the object that is sent to the Metadata Manager. The object is then sent to the Metadata Manager and the job continues in the same manner that it did previous to the changes.

By using the class loader we are able to maintain the generic nature of the interface. Had we not used the class loader, it would have been necessary for the interface to extract that data. This would have compromised the generic nature of the interface, because for each simulation we would have had to add code that extracted the fields. This would have made the work done to dynamically generate dialog boxes worthless because we would have had to add code anyway.

```
 <type>wepp.simulation.hillslope</type>
 <numelements>20</numelements>
 <tabpanes>
     <tabpanel>
       <tabname>Climate Info</tabname>
       <piecestype>
        <group>
         <type>wepp.types.climate</type>
         <heading>
          <type>radiotype</type>
          <radiotype>
           <rlabel>Use Climate File</rlabel>
           <rname>nac</rname>
...
```

Fig. 11. WEPP XML specification.

3.2 Middleware

The purpose of the middleware is to provide a flexible way of scheduling large computing jobs. In the context of Dharma, a job is composed of many different subtasks. The individual subtasks consist of a task type and resources. The task type is used to figure out what mechanism should be used to process the task, and the resources serve as the initial input data. The tasks are arranged into a directed acyclic graph (job graph) that describes the dependencies between the different tasks. If a task has any children, it means that those children cannot run until their parent is complete and it also means that they have access to any of the resources that their parent creates. The execution of a task follows the following steps: First, the cache is checked to see if the task has already been performed. If it is not in cache, the task is transformed according to the input format of the native simulation. For example, in the case of WEPP, this would mean making a control file. When this "staging" is complete, the task is enqueued in the backend's queue. Once the simulation has been executed by the backend, any resources produced are cached.

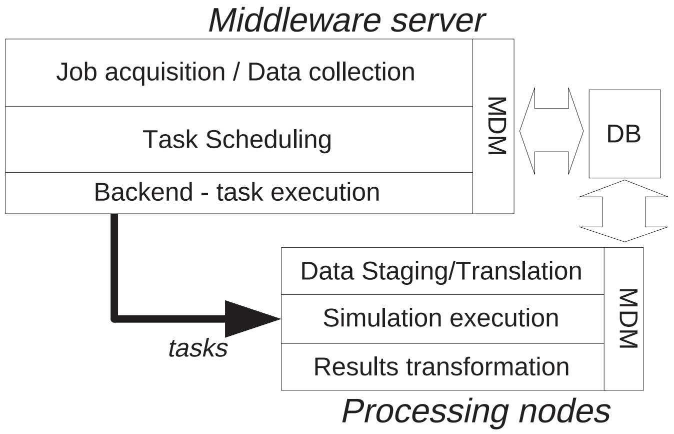


Fig. 12. Dharma middleware overview.

The middleware provides an interface for adding jobs, checking on the status of jobs (and individual tasks within a job), and getting the results of jobs (and tasks). Status objects have two attributes, a status code and a status string. They work similar to HTTP error codes where the status codes have well known predefined meanings, and the status string provides further, detailed information of the fault to the user. The resources that move around in the system also are not simply raw data. Attached to each resource is information that defines the data type; that is, metadata that describes the data, and either a URL that describes where the data is or the raw data itself. The advantage of using URLs instead of raw data is that it reduces network overheard when passing resources from component of the system to another.

In our earlier version of DHARMA, all steps to run a task, except running the simulation itself, were done on a single machine. This scheme did not scale well as the staging activities were expensive in comparison to the actual simulation itself. The current version of DHARMA incorporates a distributed version of the middleware. We have several middleware worker nodes that perform all of the activities associated with preparing a task. There exists a master middleware server that is responsible for handing out the tasks. Once the front-end has generated a job graph, it uses the interface to submit the graph. After the master middleware server has received the graph, it is responsible for making sure that all of the tasks are executed in the proper order (i.e. no child

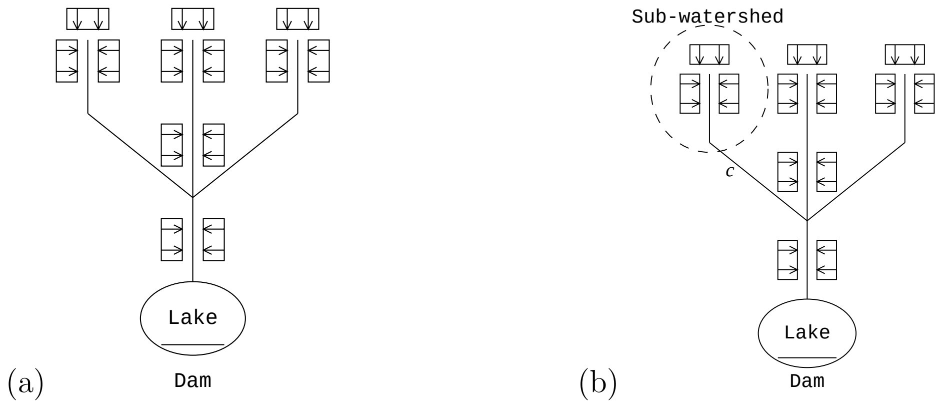


Fig. 13. A typical WEPP watershed job divided into autonomous hillslopes, with dependencies shown (a) and sub-watershed selected for independent computation (b). The blocks represent hillslopes, connecting edges represent channels.

is executed before its parent). When a worker node starts up, it communicates with the master server. If there is an available task, the task along with all the resources necessary to run it are returned. It is important to note that only the URLs and not the actual resources are moved from one machine to another.

As shown in Figure 13(a), a multiple-channel, multiple-hillslope simulation is called a *watershed* in WEPP. We have isolated each *hillslope* element of the watershed as an individual unit of computation. We consider a hillslope to be an ideal atomic unit of parallel computation because–by definition–runoff from one hillslope does not flow into another.

A branch of the dependency tree has been isolated in Figure 13(b). Note that there are only three hillslopes feeding the channel in the sub-watershed. This owes to two factors:

(1) WEPP itself only supports up to three hillslopes which empty into any one channel.
(2) Because we control the size of the sub-watersheds (i.e., they are defined to meet our purposes), we elect to maintain a relatively small size in order to maximize parallelization.

We will invoke the WEPP binary upon the results of the hillslopes inside the ellipse of Figure 13(b). The cumulative water runoff from the “watershed” will be used as the displacement through the channel c when this channel is used as an input to a later watershed.

One of the future improvements planned for the middleware is a new scheduling algorithm. As previously stated, it immediately executes any runnable task on any machine that is available. Examining the graph and figuring out better times and places to run tasks could optimize the scheduling. For instance, if there is a series of tasks that create large amounts of data, it would be time consuming and a waste of network bandwidth to try to move that data from machine to machine as tasks are executed. Instead, that series of task should be run on one set of machines.

3.3 Metadata Manager (MDM)

In order to make the system more efficient, we cache files that are submitted to us by the users when they submit a job. We also cache the output files that are created by simulation runs. To solve the problem of caching we created the Metadata Manager (MDM). The MDM is responsible for maintaining information on all of the files that have been cached on the Dharma servers. The MDM stores information on all of the files in the system in a Postgres database.

The primary keys stored by the MDM are file name, file size, checksum of the file, file type, the lat- long box that the file stores data on, location on servers, and other specialized attributes. The manager was designed to be easily extensible. For each type of file in the system there is a Java class that represents the data stored on that file. These classes contain information that is stored about the files. There are also methods in these classes that tell a database how to create tables for this type of file and create queries to get information about these file types. All that is required to add a new file type to the system is a class that tells the manager how to set up and access data on files of this type.

To access data in the MDM you must create a SOAP request that the MDM will process and return the results of the query. When looking for a file matching some specific parameters, the user will be returned a list of URLs to where the files are located on the Dharma servers. Users of the system are not allowed to directly access the database that stores the information on the files for security reasons. The abstraction added by the MDM provides a security buffer between the metadata and the outside world. This will also allow us to later implement the idea of file ownership. However, we also had to have an efficient way to store information on files that we have cached so that our system would run efficiently. In our test system, execution times typically drop over 70% compared to uncached execution indicating the overhead introduced by our system is small and the caching is effective.

3.4 Automatic weather data acquisition

Automatic data acquisition over the Internet is one of the major points of functionality of Dharma. The data acquisition module has the following goals:

(1) Automatic data acquisition via the Internet for geotemporal climatic data from online databases and digital libraries.
(2) Merging the data necessary for computation to occur, and storing it in an XML annotated form, to facilitate easily conversion to different formats.
(3) Conversion of the XML annotated data to different formats as may be needed. More specifically, this module performs the conversion for WEPP and SITES. The conversion is performed by using an XSLT stylesheet specific to each simulation.

Currently, DHARMA uses CLIGEN (a Climate Generator program) to provide WEPP with climatic data over a given time span in a per-point-per-day format. The WEPP interface takes cligen-formatted files as input. This approach has two disadvantages:

(1) Cligen can generate the data only for weather stations for which past data is available.
(2) Since cligen requires historical data, it cannot generate per-point-per-day data for any arbitrary point.
(3) Cligen also cannot interpolate data from nearby stations to give an approximation of the weather at a particular point during a given time span.
(4) Cligen can generate data only for one weather station at a time. However, often we may need data for a set of points.

This module can effectively be used as a replacement for CLIGEN, to provide data gathered in real-time to the WEPP interface. Figure 14 depicts the architecture of this module and its environment.

The Online Weather Database/Data Library collects data from various weather stations, and serves out this data in the form of XML. This data is translated by simulation-specific modules into the appropriate input files. When requesting for data, users need to specify the set of points needed as a rectangle or box, in terms of the latitude/longitudes of the topleft and bottom-right corners of the box. More specifically, requests for data must have the following parameters:

- Latitude/longitude of top-left corner
- Latitude/longitude of bottom-right corner
- Horizontal spacing between points in the rectangle specified above
- Horizontal spacing between points in the rectangle specified above

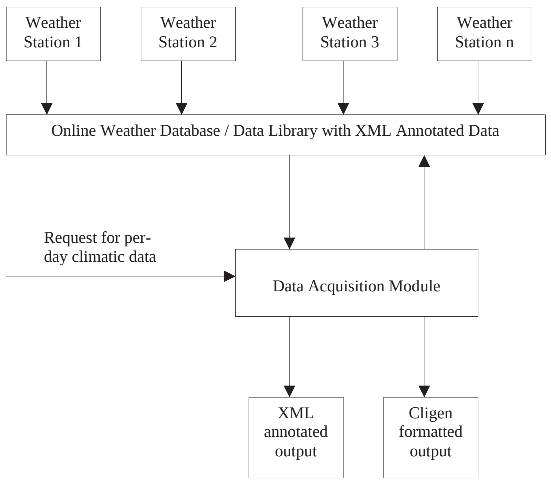


Fig. 14. Automatic weather acquisition overview.

- Starting day, month and year for the climatic data
- Ending day, month and year for the climatic data
- Maximum distance of any point from stations to be used for interpolating data
- Maximum number of stations to be used for interpolating data.

On receiving a request, the module performs the following actions:

(1) Break the input request into a set of requests for individual points. For this, it interpolates the individual latitude/longitude for each point in the specified grid.
(2) Fetch the data for each point for each day in the specified time span, from the Online Weather Database/Data Library. The fetching is done in a multithreaded fashion, with one thread being used for each point in the grid
(3) Convert the fetched data into an XML annotated format.
(4) Convert the per-point climatic data into cligens output format using XSLT. The conversion to the cligen format is again done in a multi-threaded way.
(5) Merge the XML data for all points into a single file, and store the file to disk
(6) Merge the cligen formatted data for all points into a single file, and store the file to disk

The user can use the files created above directly, or they may be used as input to the WEPP interfaces.

Performance is very acceptable, averaging approximately one minute to retrieve two years worth of weather data for over 700 points from a local weather web server. This information is cached, so repeated invocations are very efficient.

Hillslopes	*Single events (s)*	*2-yr. Cont. (s)*
136	8	17
946	13	67
4999	46	319
10002	90	624

Table 1
Dharma run times (10 processors).

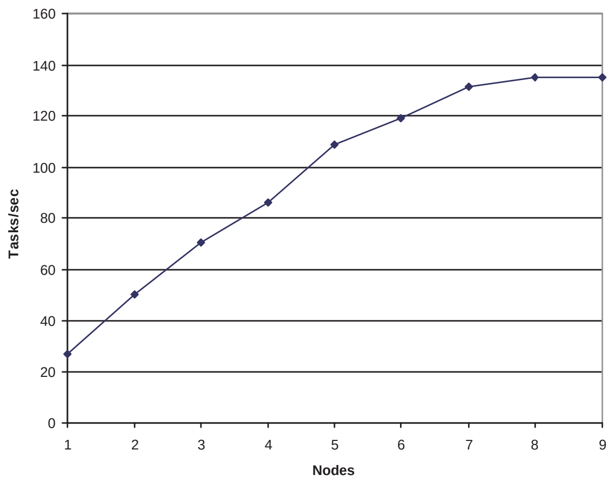


Fig. 15. Scalability for single storm simulation (5K tasks, 2-18 processors).

4 Experimental Results

Initial experimental results are positive. We estimate a total of approximately 10,000 hillslopes will be required to adequately model the Lake Decatur watershed. The project to gather this data is underway at UIUC, and will be completed later. In the meantime, we are using a subset of the Lake Decatur watershed (the 38-sq. mi. Big Ditch watershed) and randomly generated task graphs with WEPP hillslopes as leaf nodes and SITES structures as interior nodes. We have obtained convincing results indicating the scalability of our system.

Overhead for the system is somewhat steep, given its distributed nature and the degree of data conversion/translation necessary for the heterogeneous simulations. We offset this by exploiting caching, fast hardware, and parallelism. All computation nodes and our MySQL server are dual-processor Athlon PR1800-1900+ rackmount servers, with 2GB RAM running Linux 2.4.18 and IBM's JVM 1.3.1, connected via switched Fast Ethernet. A similarly-configured dual Athlon MP serves as our NFS and middleware server.

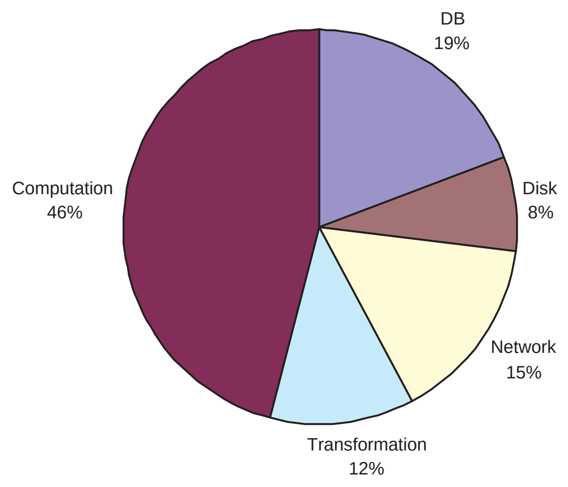


Fig. 16. Time spent per task in processing nodes (event simulation).

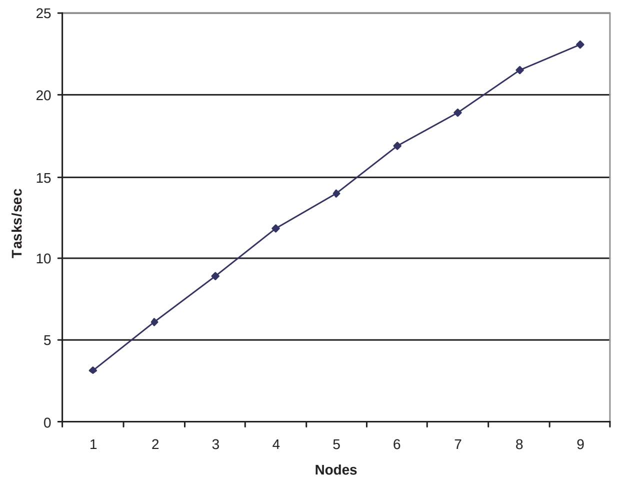


Fig. 17. Scalability for 2-year continuous simulation (5K tasks, 2-18 processors).

Single-storm event simulation. We use WEPP and SITES compiled into a Linux binary and a standard single-event 10-year storm. The current single-event task has a rather low computation/communication ratio (Figure 16), making our goal of high performance more difficult.

We also see in Figure 15 and Table 1 we are achieving near-linear speedups through approximately 4 nodes (8 processors), though we are currently bottlenecked by our NFS server. For the 10K-node single event simulation, approximately 1.2GB of data is generated. Potential solutions include using a parallel filesystem or using a higher-performance server.

Two-year continuous simulations. For continuous simulations (Figure 17 and Figure 18), with their higher computational demands (Table 1), the computatation/communication ratio is significantly better. Here, despite the fact that over 10x the amount of total data is generated, we achieve near-linear

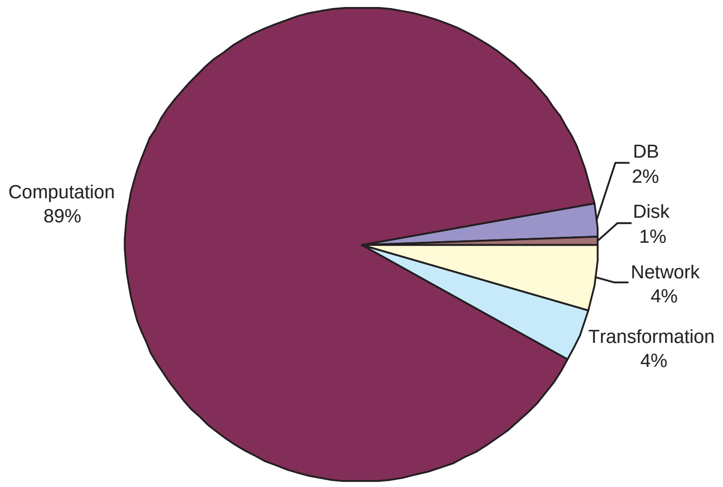


Fig. 18. Time spent per task in processing nodes (2-year continuous.

speedup through 18 processors and are CPU limited. For each graph node, approximately 500KB of data is transferred to the fileserver to be cached and returned to the client.

Computation breakdown. In Figures 16 and 18 we show the breakdown of the total times on our computational nodes. Efficiency suffers for event simulation, given the small amount of computation relative to the amount of data transferred, but we see this problem disappearing for continuous runs. We expect an increase in performance after moving to gigabit Ethernet sometime this summer, based on the total time spent in network communication. We are happy to see the time spent converting the data from one format to another is relatively small (using SAX helped considerably to keep the overhead down). The database has the potential to become a bottleneck, since each node in the task graph generates approximately ten database queries to determine when data already exists or has been created. We are investigating the use of a clustered database server.

5 Related work and conclusions

Presenting a simple interface to the researcher and educator, with its seamless and transparent access to distributed databases, is a difficult technological challenge. Various other distributed computation systems have previously been developed, but all are aimed at automating the distribution of existing scientific programming and data. Systems such as SWEB++, Ninf, AppLeS, Globus, and Legion all will schedule computation quite competently [1,10,2,4,6]. However, they fail to address the most fundamental difficulty in conducting any sort of operation on distributed data sets – the acquisition and matching of data to the models being used. Furthermore, the interface presented to the user is often more suited to the professional programmer rather than a researcher in the physical sciences. Systems devoted to universal data access, such as the WebHLA and DATORR projects, lack the domain-specific knowledge to acquire and optimally use the necessary hydrologic data [5].

In this paper we have discussed the DHARMA system, which, through the use of standards from the computational Grid and the WWW, meets our goals of increasing user capabilities while maintaining simplicity of use. We show how we have implemented automatic weather data acquisition, multi-level caching, connecting multiple simulations, and an extensible set of user interfaces to bring hydrologists into a new era.

As shown in Table 1, a 136 node task graph covering the Big Ditch watershed takes 8s. on ten processors (five nodes), while 10,002 task nodes take 90 seconds for a single storm event, and simulating two years takes only around five minutes. Given this performance, we are quite confident in our ability to simulate the entire Lake Decatur watershed with an unprecedented degree of precision when our full dataset becomes available.

In the near future we are extending DHARMA in two directions. First, we are working to incorporate the HEC-RAS river modeling tool. This will significantly increase the accuracy of our channel simulation, and, due to the fact that HEC-RAS allows for backflows where two channels meet, forcing us to extend our dependency graphs to include cycles. Second, we are developing the tools to automatically aggregate and partition jobs to increase our overall execution efficiency. We plan to begin using CONDOR-G to test performance over Globus and the Grid.

Acknowledgments

This work was supported in part by startup funds from Kansas State University, and NSF ITR-0082667. All opinions, findings, conclusions and recommendations in the material are those of the author(s), and do not necessarily reflect the views of the National Science Foundation. We also wish to thank Michael C. Hirschi, Ryan Porter, Jesse Greenwald, Dan Lang, Matt Hoosier, Mahesh Kondwilka, T.J. Rothwell, and Doug Armknecht for their efforts and enthusiasm.

References

[1] D. Andresen and T. Yang. Multiprocessor scheduling with client resources to improve the response time of WWW applications. In *Proceedings of the 11th ACM/SIGARCH Conference on Supercomputing (ICS'97)*, Vienna, Austria, July 1997.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing'96*, Pittsburgh, PA, November 1996.

[3] K. Cato and C. Mathewson. Rock material performance during emergency spillway flows. In *Proceedings of the 1989 ASAE Annual International Meeting*, St. Joseph, Michigan, ASAE Paper No. 89-2649, 1989.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[5] G. C. Fox, W. Furmanski, G. Krishnamurthy, and H. T. Ozdemir. Using WebHLA to integrate HPC FMS modules with Web/commodity based distributed object technologies of CORBA, Java, COM and XML. In A. Tentner, editor, *High performance computing: HPC '99: Conference; 7th — April 1999, San Diego, CA*, PROCEEDINGS OF THE HIGH PERFORMANCE COMPUTING SYMPOSIUM 1999; 7th, pages 273–278, San Diego, CA, USA, 1999. Society for Computer Simulation.

[6] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds, Jr. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, Department of Computer Science, University of Virginia, June 08 1994.

[7] J. Laflen, L. Leonard, J. Lane, and G. Foster. WEPP: A New Generation of Erosion Prediction Technology. *Journal of Soil and Water Conservation*, 1991.

[8] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference on Distributed Computer Systems*, pages 104–111. IEEE, June 1988.

[9] M. Neilsen and D. Temple. A distributed simulation environment for water resource site analysis. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 560–566, 1999.

[10] Ninf, a network based information library for global world-wide computing infrastructure, 1996. http://hpc.etl.go.jp/NinfDemo.html.

[11] R. Rew and G. Davis. NetCDF: An interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82, July 1990.

[12] D. Temple and G. Hanson. Headcut development in vegetated earth spillways. *Applied Engineering in Agriculture*, 10(5):677–682, 1994.

[13] D. Temple, H. Richardson, H. Moody, H. Goon, M. Lobrecht, J. Brevard, G. Hanson, E. Putnam, D. Woodward, and N. Miller. Water Resource Site Analysis Computer Program (SITES) – User's Guide. Technical report, USDA, 1996.

[14] S. Vinoski. Distributed Object Computing with CORBA. *C++ Report*, 5(6), Aug. 1993.

[15] M. Weiss, A. Jhonson, and J. Kiniry. *Distributed Computing: Java, CORBA, and DCE*. Open Software Foundation Version 2.1, Feb. 1996.

[16] A. Wollrath and J. Waldo. Trail: RMI. *The Java Tutorial*, July 1999. http://java.sun.com/docs/books/tutorial/rmi/index.html.

[17] Extensible Markup Language (XML). *W3C*, July 1999. http://www.w3.org/XML/.