Int. J. Communications, Network and System Sciences, 2012, 5, 343-352
http://dx.doi.org/10.4236/ijcns.2012.56045 Published Online June 2012 (http://www.SciRP.org/journal/ijcns)

A Systematic Approach for Hydrological Model Couplings*

Daniel Salas[1], Xu Liang[1], Yao Liang[2]
[1]Department of Civil and Environmental Engineering, University of Pittsburgh, Pittsburgh, USA
[2]Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, Indianapolis, USA
Email: {das140, xuliang}@pitt.edu, yaoliang@iupui.edu

Received September 30, 2011; revised April 30, 2012; accepted May 10, 2012

ABSTRACT

It is of great importance to develop a systematic framework to integrate and coordinate software components to effectively and efficiently accomplish complex hydrological modeling tasks. In this paper, we examine the state-of-art information technologies including service-oriented architecture, and propose a systematic approach based on service-oriented architecture and scientific workflow to investigate the general model coupling problems. A prototype system, MoteWS, based on web services for publishing field measurement data from wireless sensor networks is developed to preliminarily explore and test our proposed architecture. Results and lessons learned are discussed and future recommendations in this direction are provided.

Keywords: Service-Oriented Architecture; Scientific Workflow; Web Services; Wireless Sensor Network; Data Retrieval

1. Introduction

The study of hydrological processes and their associated extreme events (e.g., floods and droughts) is of paramount importance to human lives, global climate change, a healthy and sustainable ecological environment, and the national/international economy. In today's scientific modeling for such very complex hydrological and environmental systems, coupling between different hydrological models and/or among hydrological models and climate models becomes a common practice.

A desirable framework for these models' coupling should support modularity, flexibility and interoperability, in the sense that individual models' development and implementation are independent with each other and have their own integrity and autonomy. Such a model coupling framework can not only greatly facilitate and fit in interdisciplinary and collaborative scientific work environment, but also dramatically increase the efficiency and flexibility of model couplings in research and operational practices.

In general, integration of hydrological models can be either a single point-to-point connection with a unique one-way interaction, or a multiple point interactive and coordinated set of collaborative activities. The former can be referred to as a simple coupling problem whereas the latter as a sophisticated coupling problem. In hydrological, environmental and climate fields, models, either physically-based or data-driven, are usually realized and simulated in terms of software systems. Due to the increasing complexity and heterogeneity of software packages employed and hardware and operating systems platforms used for individual models' development, we seek to develop a systematic approach and framework to facilitate such complex model coupling and integration. Service-Oriented Architecture (SOA) and scientific workflow have great potential to achieve our goal. Following the SOA approach, our idea is to encapsulate each individual model's functionality in services in addressing the needs of modularity, flexibility, autonomy, and interoperability of individual models in the coupling framework. As services can be distributed over Internet and reused, SOA can indeed promote remote collaborative and interdisciplinary team work and make different targeted models/systems' integrations efficient. On the other hand, SOA alone is not adequate to address the problem of automating coordinated set of collaborative interactions among models for sophisticated couplings. This leads to scientific workflow. The rest of the paper is organized as follows. Section 2 reviews and discusses how SOA, scientific workflow and other potential techniques could be used to integrate and coordinate systems that represent scientific models. Major advantages and disadvantages of these techniques, found in the scientific and commercial computing fields, are discussed. In Section 3, we propose a general architecture for model couplings based on SOA and scientific workflow. Section 4 describes MoteWS, a prototype web services-based system developed to publish field measurement data from

*Theory recompilation, analysis and practical application.

wireless sensor networks to illustrate and test preliminarily the main SOA component of our proposed architecture for model/system couplings. Finally, in Section 5, our learned lessons during our prototype development are discussed, recommendations and our future work along this research direction is provided.

2. Reviews and Analyses

2.1. Related Work

Some work dealing with the challenges of connecting collaborative hydrological models exists (e.g., [1,2]). For example, a theory-based analysis (e.g., [3]) is presented to show some solutions that can meet the needs of model integration. Some proofs-of-concept of scientific workflow or SOA architecture work, including [4-9], also exists. In [10], a scientific workflow analysis goes even further to find systematic ways to study results and to improve the design. One of the main goals of this paper is to show, in a condensed but easy-to-read manner, why SOA and scientific workflow are a better solution compared to other alternative techniques in order to couple, coordinate, collaborate and evolve hydrologic models in the process of hydrologic studies. While these reasons appear to be assumptions in other work, we believe it is important to establish them explicitly rather than implicitly to provide insights, and to pave the way to our proposal of a general architecture for hydrologic model coupling in the next section. In the following, our analyses are given regarding potential techniques for model coupling in a comprehensive manner, which is not available all together in the previous work.

2.2. Potential Techniques for Integration

The techniques can be classified into the following categories.

1) Data integration: The models or systems use a shared repository of data. Each system puts information in the repository, where others can find it and read it. The receiver can poll the repository continuously until it finds a message, or, the repository can provide an event-alert system. Advantages: Usually easy to implement. Disadvantages: Affects independence of each application. The integration is easily lost if any of the applications evolves.

2) Business integration: In this type of communication, one system sends a message from a core component (or a logic component) and the other system receives it in a core component (or a logic component). The business integration can be made using special sub-layer in the logic layer designed to process the messages (see **Figure 1**). Advantages: Each application keeps its independence. Disadvantages: Requires more work because some service is required to be developed. This service will contain the code to access the other layers, data, etc.

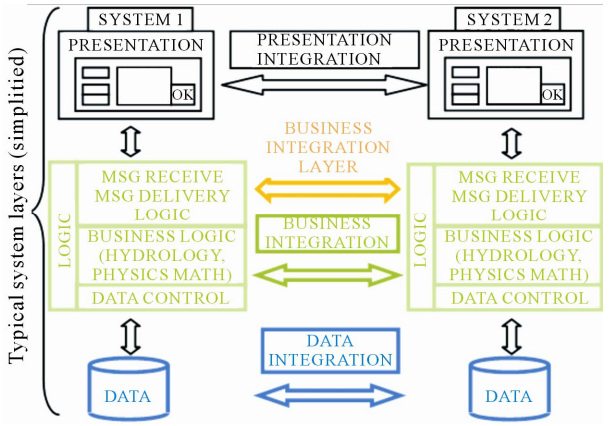

Figure 1. Illustration of different types of integration.

3) Presentation integration: The GUI of one system allows the user to access the GUI of another system. Advantages: Allows the end user to visually perceive the integration. Disadvantages: Most times it creates strong dependences between applications.

For the task of integration between hydrological models, the type of integration that best fits the requirements is the business integration because:

1) It makes it easier to keep models independent.

2) It makes it easier to keep low coupling[1] between systems.

3) Transparent to end users.

4) It is independent of data storage implementation.

To offer functionality to other systems, each system can publish:

1) An API[2]: This allows the client to access all objects, send, receive and update objects, and use services of these objects.

2) Services: The logic layer can publish independent services that are not part of the objects. It leads to Service Oriented Architecture (SOA).

For hydrological models' integration, the SOA architecture is chosen because the services keep things independent and allow lower coupling between systems. Also it helps to keep the systems transparent to the other side of the system developers in the sense that the developers in one system do not need to know the implementation details and the objects' structures of the other system.

The task of hydrological model integration requires a large amount of information coming in from multiple data sources and different models in a coordinated and collaborated way to obtain a solution. Thus, such a complex task leads to a next one, the flow control task (Coordination of activities).

[1]In computer science, coupling or dependency is the degree to which each program module relies on each one of the other modules.
[2]Application Program Interface.

2.3. Potential Techniques for Coordination of Activities

The techniques can be classified into the following categories.

2.3.1. By Design

- Description: In this technique, each model can send or receive messages following the rules established in the flow-design. The flow of activities can be centralized or decentralized without a real (automated) restriction. Then, in maintenance time, these systems tend to become decentralized, it means that each model will send and receive messages from another model.
- Problems:
  - If a developer breaks the rule and sends a message back to the original sender and this message asks to re-do computations, it can produce a dead-lock.
  - Each system must be able to implement communication protocols to any other system to be connected. For example:
    - System "A" in language C++ running on Windows.
    - System "B" running PHP on Linux.
    - System "D" running Java on Solaris.
    - System "E" running Pascal on Windows.
    - All systems need the code to communicate to each other's languages and operating systems.
  - The whole-integrated system works fine only if all the developers of each single system agree the flow-design paper and respect it always, though in reality, nothing ensures that.
  - If the flow-design changes, all the systems and interactions must be changed. It puts integrity in risk for all the individual systems.
  - Updating and improvement of the integrated system are expensive, risky and restricted.
- Advantages:
  - Requires no invest in any control tools.
- Where the control is: There is not automated control. There is only paper control.

2.3.2. By Central Control

- Description: In this technique, one model (or system) is named Controller. This Controller sends requests to other models, but the other models do not send messages between them. The Controller has the code to control the flow of information and activities that the other models perform. It can become decentralized in maintenance time.
- Problems:
  - It works okay only if the other models never send messages between each other.
  - It works okay only if the Controller implements a flow-design without mixing the flow code and the business (e.g., hydrology) code. Nothing ensures that.
  - The Controller must be able to implement communication protocols to any other system to be connected. This gives additional responsibilities to a system made for the hydrological modeling.
  - Updating and improvement of the system are very expensive and restricted.
- Advantages:
  - If the flow needs to change, only the Controller application must be reviewed and changed.
- Where the control is: The control is always implemented in the code of the Controller model.

2.3.3. By Message Broker

- Description: All the involved systems send the messages to a component designed to intermediate communications only. The systems publish their services in the broker. Other systems can subscribe to the published services and then consume them. See **Figure 2**.
- Problems:
  - The flow control is embedded in the publish/subscribe design, in the implicit flow-design and in the connections that each system performs to the broker. Thus, the flow control becomes difficult to understand, maintain and change.
  - If a system does not respect the flow-design semantics, it could create an infinite loop.
- Advantages:
  - Models can send messages between them with almost no restrictions.
  - If there are infinite loops or errors, debugging is much easier than without the broker.
  - Models keep a high level of independency and low coupling. Thus, the capacity of each model to evolve is not affected, as is for the previous techniques.
  - Each system must be able to implement only one communication protocol: The one between its own language/platform and the broker.
  - In the category of central control, one of the models (*i.e.*, Controller model) was required to have code to communicate with all of the other systems. Here, the broker only establishes some communication protocols for some standard platforms (or maybe only one) and the others need to adapt to it.
  - The whole integrated system is ordered, traceable, repeatable. The flow-control is difficult to modify but is easier to analyze and debug.
- Where is the control: Mostly in the publish/subscribe protocol in the broker, but also in the flow-design semantics and some parts are in the models. See **Figure 3**.

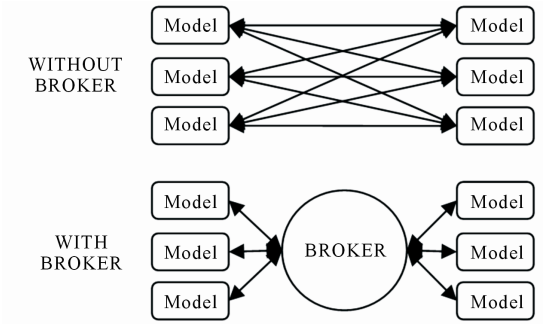

Figure 2. Integration using a message broker.

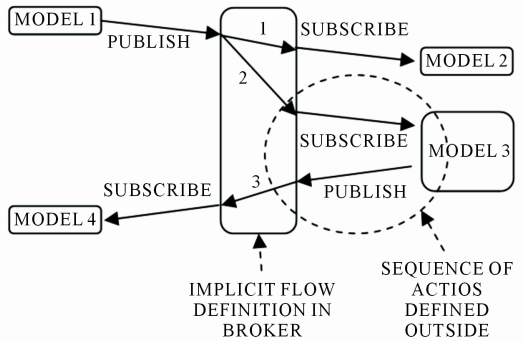

Figure 3. Flow definition in P/S protocols.

2.3.4. By Workflow

- Description: The flow controller is a component called workflow. This dedicated component has the full responsibility of controlling and coordinating all the processes required to complete the computations. Each time that the workflow requires to run specific models, it will call them. The workflow component can act also as either a broker or not. In this case we will assume the workflow is also a broker to achieve all the advantages described in both message broker control and workflow control.
- Problems:
  - It requires developers to have skills in workflow theory and integration.
  - It requires a thorough flow design.
  - It requires standards for processes, communication, units and composition of data.
- Advantages:
  - Each model can communicate to anyone else. Better yet, in this technique, each model does not need to communicate directly to others; the workflow component will send request to the other models and receive their responses.
  - Integration loops are controlled by only one component. Thus, infinite loops and dead-locks occur less and are easier to correct.
  - Each system keeps as much independency as possible with a lower coupling.
  - Each system must implement one communication protocol: The one between its own language and the workflow.
  - The flow-control code and the hydrological code are completely separated.
  - The whole integrated system is ordered, traceable, repeatable and the flow-control is much easier to understand, modify and debug.
- Where is the control: In the workflow component.

3. Models Coupling Architecture

We propose a general architecture for the hydrological model coupling, in which the selected communication technique is web services and the selected flow-control technique is workflow.

The driven criteria to choose the integration and flow-control technique is the maintainability. In commercial software, the maintenance cost represents between 60% and 80% of the cost of the life cycle. In scientific software, the maintenance (or evolution) is even more intensive due to the constant change of the concepts and ideas during ongoing research. One of the keys for maintainability is loose coupling because it implies that modifying one component does not affect others, reducing the complexity of the software system evolution. The communication technique that better fits these criteria is the business integration (see 0). It could be implemented through an API or services. Usually the use of an API would be more efficient but implies that the client component requires knowing the object structure of the host component, whereas the implementation in terms of services would be more transparent although a little bit more overheads. Those services also will be web services because: 1) The models are remotely reachable through a network; 2) This is a kind of services well known and very mature; 3) This is a standard and well defined way of interoperation; 4) The models are deployed on machines capable to communicate by SOAP on http servers.

The flow-control technique that better fits the maintainability criteria is by workflow because it keeps coupling as low as possible (see 0). The proposed architecture supports the requirement that models do not communicate to each other in any predefined way. Each model publishes some functionalities through services that are independent from other models. Moreover, each service uses WSDL to publish the metadata that allow others to find and use them. In that way the architecture fits the SOA paradigm. Nevertheless, SOA alone is not sufficient in the hydrological model coupling. This is because the publish/subscribe protocols that are typically used in SOA would define an implicit order for the model coupling interactions. **Figure 3** shows an example of a configuration in a publish/subscribe broker. It can be

seen clearly from **Figure 3** that Model 3 will first consume Model 1's service before it publishes its service to Model 4. Thus, it implicitly defines the service sequence.

However, implicit design tends to obscure operational logic and flow control among services and thus is not always adequate, especially for a complex model coupling system. For example, in **Figure 3** the sequence from the subscription to the publishing of Model 3 (*i.e.*, Arrow 2-Model 3-Arrow 3) is defined outside the flow definition. Implicit definition is insufficient and error-prone also if the problem is complex or flow-control changes permanently. Scientific workflow solves this requirement through explicit definition of the interaction flow.

The workflow component uses the functionality that is already published by the models as services.

It can be located in a separate machine, so that its performance will not be affected by individual model runs. The workflow component will have the control of the process and will be responsible for using the models as required. It is possible to use a model multiple times or establish loops with recurrent calling to models (time-steps). The models will have no direct communication. Models will only response to requests made by the workflow. The models will publish their services as web services. Our proposed architecture is illustrated in **Figure 4**.

Models located in different facilities can work together through remote connections. The interaction starts when the controlling machine starts a computation (see LOCATION 2 in **Figure 4**). The workflow residing at LOCATION 2 knows which and how the activities should be made. For example, this workflow can execute some activities and after that it determines the next execution offered by Model 1.

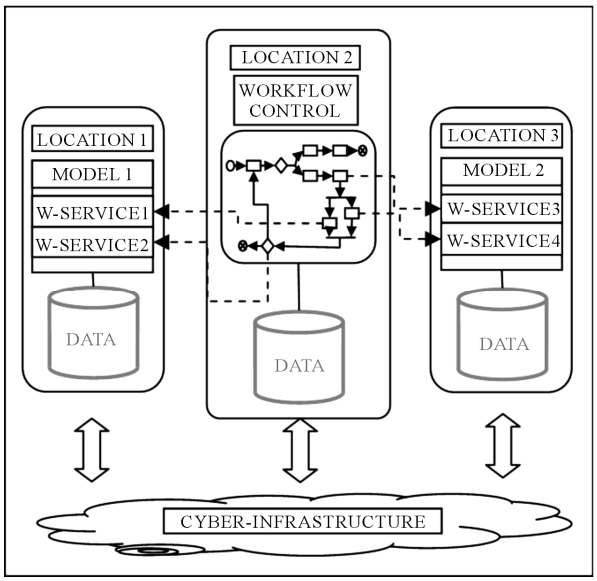

Figure 4. General architecture proposed.

At this point, the workflow will establish a connection with, for example, web service 2 (WS2) of Model 1. The workflow will call this web service to send required parameters and will receive a response from Model 1. Model 1 will then use the parameters received to execute hydrological computations and obtain results.

Responses from Model 1 can be used for decision making, or can be used as parameters for sending to another model, say, Model 2. For example, assume the response from Model 1's WS2 will be used as parameters for Model 2's WS3. In this case, the work-flow will establish a connection to web service 3 of Model 2. After having a response from Model 2, the workflow can finish its work or can start a new iteration calling again the models as many times as required, and so on.

As the models and workflow illustrated in **Figure 4** will be available through Internet, they will become part of the Cyber-infrastructure. We adopt the following simple definition for Cyber-infrastructure: The set of all the services and resources available through a network to work in a scientific-collaborative way.

For example, as illustrated in **Figure 4**, for people working on Model 1, cyber-infrastructure is the cloud that helps them work with a workflow and the collaborating model—Model 2. On the other hand, for people working on Model 2, cyber-infrastructure is the cloud that helps them work with a workflow and the collaborating model—Model 1.

4. Motews: A Web Services Based Prototype System

To examine and test out proposed architecture, we started implementing the communication part—the web services—by developing an online system to publish real-world field measurements from wireless sensor network in near real-time.

4.1. Description of the Prototype System

The field measurements are being collected through a wireless sensor network, made up with devices that have sensors and transmitters (we call them motes for simplicity). Each wireless mote transmits all the observations to a data sink gateway called the net-bridge. There is one net-bridge for each field network in our two testbeds. The net-bridge, connecting to both wireless base station and Internet, is a gateway with Linux operating system and wall power. In addition to collecting data from the mote network, the net-bridge is also able to publish network services (like web services). That is, a web service was deployed in the net-bridge to publish the observations collected from the fields. This web service will dispatch a tar file including all the files collected from a given date.

The web service has been also deployed in a central data server at the hydrology lab at the University of Pittsburgh that collects information from all the net-bridges. This central server has client software to poll each hour from all of the net-bridges. If new files are found, they are added to the data repository. The central server publishes the similar web service that the net-bridges do. But in this case, the user can send a parameter indicating the source (*i.e.*, site ID) of data to be retrieved. The final result is that one can retrieve either consolidated data by using the web service provided at the central data server with one-hour delay at most, or single-site data by using the web service deployed at each gateway with little delay. Due to the fact that the field measurement data are published through web services, other applications, hydrological models, workflows or desktop programs can be easily integrated with these sources of data.

4.2. Design and Implementation

Figures 5 and **6** show the schematic view of the sensor data collection and dissemination through web services, respectively. The motes have sensors that take measures from the field, and send the observations through the multiple hop wireless networks to the net-bridges. The net-bridges collect the observations and store them in files. Each net-bridge publishes a web service that dispatches these files on demand. The central server uses the client software designed to access the web service published by the net-bridge and to retrieve the files. The central server also publishes a web service of its own that can be accessed by users to get the data from multiple test bed fields (e.g., two different sites at present). Through such web services of sensor data gathering from multiple testbeds, the architecture and communications of the wireless mote networks themselves do not need to be modified. **Figure 7** shows the structural view of the web services framework. Different kinds of clients access to the web services through a framework that offers a unique portal at the central server. The framework uses part of the URL as a parameter to determine which web service will response to a given request. These components (the web services) should implement some functions with given names and parameters, and can be connected to form the web services framework.

In our prototype system, the framework chosen is Axis2C, and it provides (among others):

- Transport of the SOAP messages.
- Queuing messages in pipes.
- Locate and dispatch incoming messages to the service that will response it.
- Validate messages.
- Embeddable in C applications.

Figure 8 shows a modular view of web service. The first group of functionalities has the responsibility of

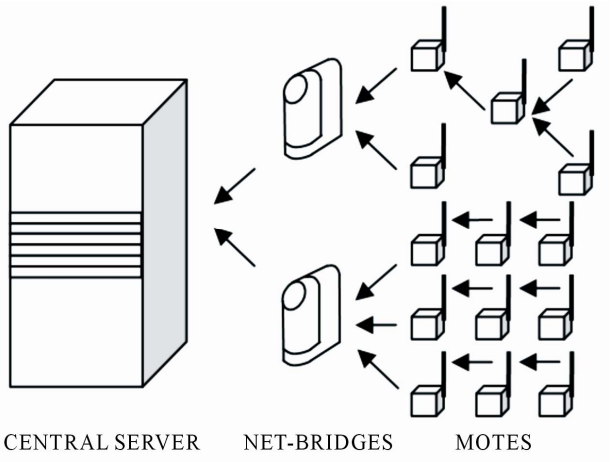

Figure 5. A schema of information collection setup from two different remote testbeds to the University of Pittsburgh campus.

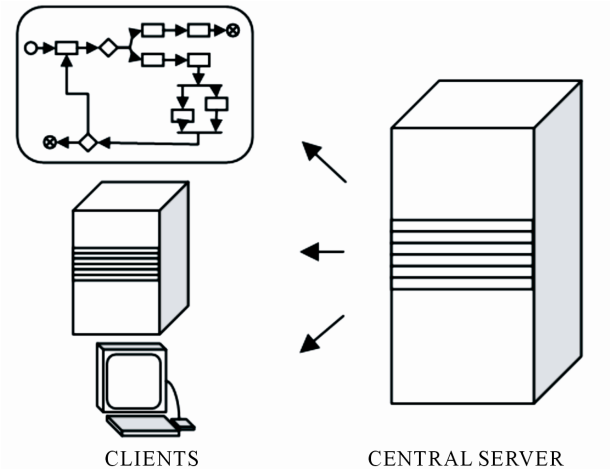

Figure 6. A schema for dispatching information.

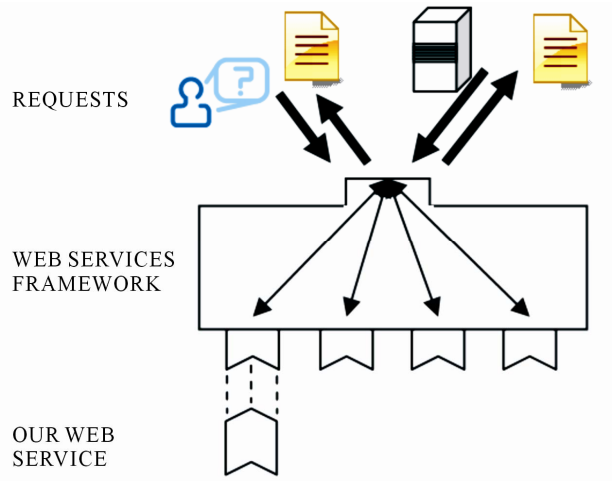

Figure 7. Schema of the web services framework.

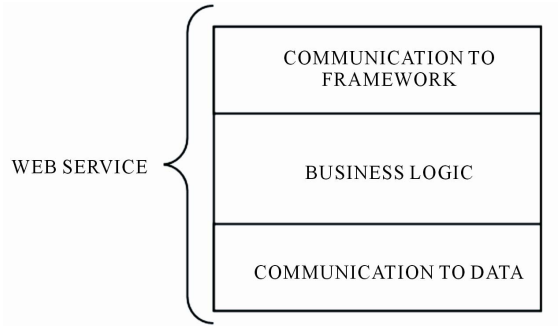

Figure 8. Web service modules.

fulfilling certain requirements to be connected with the framework. It includes functions to create, delete, receive the XML message and respond to an XML message among others.

Several frameworks in different languages are available for people to develop, deploy and publish web services. Axis2C was chosen because it is an open source tool and it is in C. Most of our computational models are written in C. Thus, it would be easier to build web services with them.

4.3. Detailed Design

The current prototype is just using an existing framework to test the connectivity and performance of C applications through webservices, without the workflow part yet. Hence, the detailed design will include the following: 1) the setup required to make the framework work in the specific platforms; 2) the component that defines the specific web-service; and 3) the client component that connects to the web-service. They are described as follows.

1) Setup at the server:

a) Install the Axis2C libraries using the usual open source commands: configure, make and make install. It implies that the framework is compiled IN THE NETBRIGDE. For Windows, the binary files are available.

b) Ensure that OpenSSL is installed. In Linux it can be obtained through automatic package managers. In Windows it is required to download and install manually.

Setup at the client:

a) Connectivity to the server.

2) Component that defines the specific web-service:

This component needs to be located in the framework folder as a library which offers the following functions:

- Constructor: It allocates memory for the skeleton, which is a container with the framework information and the list of names for the rest of functions.
- Init: It loads the name of the web-service so it is prepared to response requests.
- Invoke: It receives the information corresponding to a specific request, process the information and it is responsible for generating a response. For the purposes of this prototype, the process is performed in another function. But, for a real full integration model, the Invoke function should call some component of the business layer in its host application.
- OnFault: This function defines the actions taken when an error is detected.
- Free: It releases the memory initially allocated.

3) Client component:

It can be developed either using the Axis2C framework, or just independently following the steps required to connect to any web-service.

- In the former case (*i.e*., this prototype), the client is compiled using the Axis2C headers and libraries. A payload object is created and filled with the information to be sent to the server. Then, using the Axis2C libraries, the payload is transformed to XML and sent to the server through the http. The Axis2C libraries help receiving synchronously the response. Then, the client extracts the results from the XML and shows them to the user.
- In the latter case, most development tools can extract the WSDL by connecting to the web-service through the http. The WSDL is the descriptor of the web-service. Once it is obtained, the tools provide APIs to create a kind of payload, send the message, receive the response and extract the results.

This specific prototype was developed without a WSDL, but following prototypes that will actually connect to workflow, will also offer a WSDL.

4.4. Performance Testing

The MoteWS has been tested manually and automatically with both serial load and parallel load. We have conducted two phases of testing. In the first phase of testing (*i.e*., preliminary test) the unitary test used is a request that retrieves all the field measurements for one month of data. In the second phase of testing (*i.e*. detailed test), data files with different sizes are retrieved in experiments with increasing number of files.

First group of tests: The serial load test was performed to detect memory leaks or other misbehaviors that may lead to performance deterioration. **Figure 9** shows the time required to get the response from the web service for each of the 297 requests conducted at the central server. The average is 8.6 seconds. The time required increases slightly on each experiment: About $0.00000003 \times 24 \times 60 \times 60 = 0.0026$ seconds in average. Although it is a minor decrease in performance for the purposes of this tool, the behavior should be analyzed for a future project with a more intensive demand from clients. It is considered a successful test.

The parallel test was made in two parts. The first part

was to determine the number of concurrent clients that could be accommodated by the web service and how critic it would be to increase that number. It was found that the web service can accommodate up to 10 concurrent clients. That is an encouraging number given our current inexpensive hardware using very limited resources. If this number increases, for example, up to 11 concurrent clients, the ratio of fails drops abruptly from 0% (for 10 clients) to 50% (for 11 clients). The performance drops more if more concurrent clients are present.

The second part in the parallel test was to examine the web service performance in terms of response time up to 10 concurrent clients, and the results are shown in **Figure 10**, the average response time needed per request for a given number of concurrent clients.

As we can see, for the case of 10 concurrent clients, each one would observe in average about 55 seconds in response delay. It also seems that the exponential regression fits better than the linear one. The parallel testing was also successful.

Second phase of testing: Serial tests: The serial tests have been run for different file sizes from 60 Kb to 330 Mb. The file is requested, and once it is done, the file is downloaded again and stored with a different name. This process is repeated several times (1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 times).

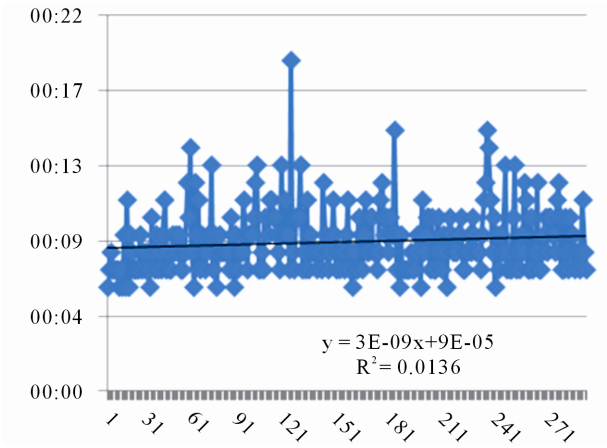

Figure 9. Serial load results of phase 1 testing.

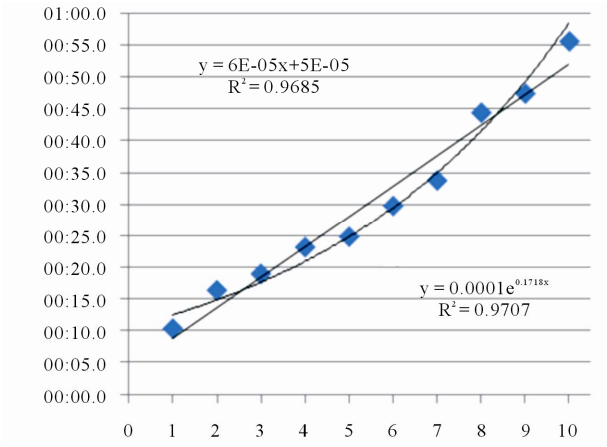

Figure 10. Parallel load results of phase 1 testing.

Each test has been given a name according to the size of the file (A: 60 Kb, B: 2.6 Mb, C: 8.1 Mb, D: 21 Mb, E: 330 Mb) and the number of file downloads in series (e.g. A1, A500, C5, etc.). Each test trial is also repeated 2 or 3 times to check consistency of the behavior. A total of 112 tests trials have been run for a total of 17256 files retrieved. All together accumulate almost 133 Gb. For each test, the total time required to download *n* files of size *m* has been recorded to compute the transmission speed.

The speed is averaged for each file size and for the number of each file's downloads. The results are shown in **Figures 11** and **12** for different number of downloads and file sizes, respectively. A trend line and its equation are shown for each figure. It can be seen that the change in the trend for the test range is much smaller than the variability of the results. We can conclude that the speed is not significantly affected by the continuous use of the service (number of file downloads) or the size of the files. The number of failures was less than 1% for every test trial.

Parallel tests: The first parallel test was performed with a series of 100 files of 60 Kb each. The series was run in parallel for 2, 5 and 10 threads. For that size it was hard to find a trend in the results (see **Figure 13** experiment A100).

Figure 12 shows that the optimum speed is obtained for files of size 10 Mb. For smaller files, the starting and completing processes for each download consume considerable time, compared to the actual retrieval of the files. For larger files, the system may become more stressed processing the download. The number of failures was below 1%, just as the serial test. For the 2.6 Mb files, the average speed was computed for each download (Experiment B100). It can be seen in **Figure 13** that the speed decreases rapidly when parallel threads are added.

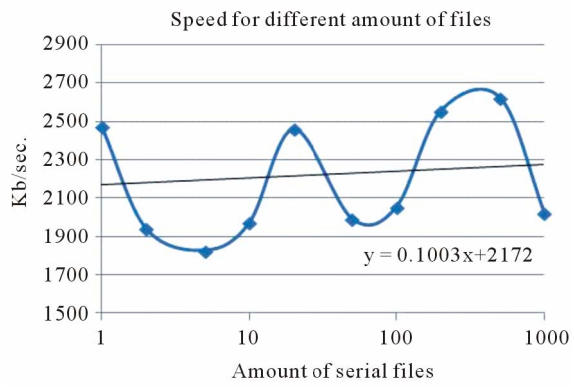

Figure 11. Serial load results of phase 2 testing for different amount of files.

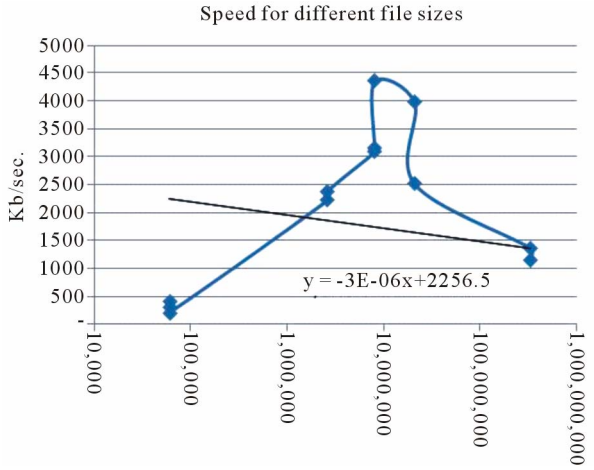

Figure 12. Serial load results of phase 2 testing for different file sizes.

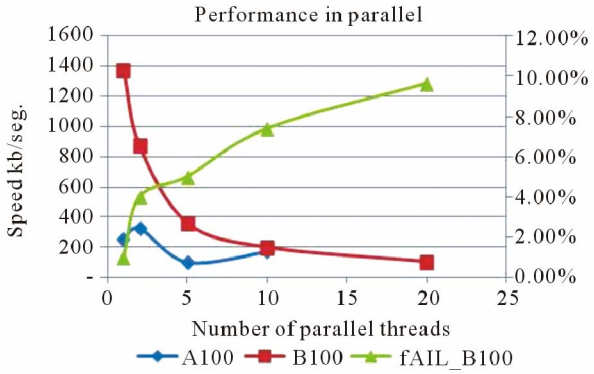

Figure 13. Serial load results of phase 2 testing.

Also, the number of failures increases fast. For 20 threads, the ratio of failures is almost 10% which is very high. A ratio of failures of 5% should be the maximum acceptable, and it is obtained between 5 and 10 threads. It confirms that, with the given hardware, the system should be used to accommodate at most 10 users retrieving files simultaneously.

4.5. Discussions

By publishing the observations from field through web services, clients do not need to have direct access to the machines. Thus, server's risk of accidental damage or misconfiguration would be reduced.

The web services can be easily embedded into existing model software because they are all written in C. The code also can be compiled in Windows or Linux. The data can be integrated in processes or iterations managed by others, who can access the data through the web services.

In Axis2C it is possible to define and develop a web service starting from a WSDL or not.

Trying to manually create the WSDL after the web service is made can be difficult. Thus, for the future web services made in Axis2C, our recommendation is to generate the WSDL first with a latest generation development tool.

After that, scripts offered by Axis2C to generate part of the web service skeleton should be used. The rest of the development can be proceeded in the same way.

5. Conclusions

Web services and workflow provide a standard solution for communication and integration of hydrological models. They should be made in the business (hydrology) layer to provide transparency to end users of each independent model and to ensure independency of the data models. The web services also can be deployed to provide connectivity to every component that requires to be connected to the cyber-infrastructure, like the field measurement sensor networks where web services provide a solution to get automated and remote access to the field measurements online in hydrological studies.

To preliminarily examine and test our proposed general architecture for hydrological model couplings, we deployed web services for collecting and publishing field measurement data from two wireless sensor networks in near real-time. Our ultimate goal is to thoroughly examine and test the proposed general architecture for hydrological model couplings, as shown in **Figure 4**. In the future work, various hydrological model components will be equipped with web services, and a central coordinator to realize workflow will be developed to control the model couplings and iterations within and between institutions. It will also make it possible for direct use of the observation data from field measurements to the coupled-modeling system in near real-time. Thus, this work can directly lead to and support for a future framework/ mission of realizing a real-time online hydrologic monitoring, control, and forecast system.

6. Acknowledgements

This work was supported in part by NSF grant CNS-0721474 to the University of Pittsburgh.

REFERENCES

[1] E. Delman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, *et al.*, "Examining the Challenges of Scientific Workflows," *Computer*, Vol. 40, No. 12, 2007, pp. 24-32. doi:10.1109/MC.2007.421

[2] S. M. Guru, M. Kearney, P. Fitch and C. Peters, "Challenges in Using Scientific Workflow Tools in the Hydrology Domain," *The* 18*th Worlds IMACS MODSIM Congress*, Cairns, 13-17 July 2009, pp. 3514-3520.

[3] A. Goderis, C. Brooks, I. Altintas, E. A. Lee and C. Goble, "Heterogeneous Composition of Models of Computation,"

Future Generation Computer Systems, Vol. 25, No. 5, 2009, pp. 552-560. doi:10.1016/j.future.2008.06.014

[4] M. J. Fairman, A. R. Price, G. Xue, M. Molinari, D. A. Nicole, T. M. Lenton, *et al.*, "Earth System Modeling with Windows Workflow Foundation," *Future Generation Computer Systems*, Vol. 25, No. 5, 2009, pp. 586-597. doi:10.1016/j.future.2008.06.011

[5] Q. H. Shao, P. Sun and Y. Chen, "Efficiently Discovering Critical Workflows in Scientific Exploration," *Future Generation Computer Systems*, Vol. 25, No. 5, 2009, pp. 577-585. doi:10.1016/j.future.2008.06.005

[6] Cyberinfrastructure for Environmental Research and Education, "Report from a Workshop Held at the National Center for Atmospheric Research," 30 October 2002.

[7] F. C. Delicato, P. F. Pires, L. Pirmez and L. F. Rust da Costa, "A Flexible Web Service Based Architecture for Wireless Sensor Networks", *Proceedings of the 23rd International Conference on Distributed Computing Systems*, Providence, 19-22 May 2003, pp. 730-835.

[8] P. Taylor, H. Neuhaus, Y. F, Shu, D. Smith and A. Terhorst, "Hydrological Sensor Web for the South Esk Catchment in the Tasmanian State of Australia," *Fourth IEEE International Conference on eScience*, Indianapolis, 7-12 December 2008, pp. 432-433. doi:10.1109/eScience.2008.89

[9] J. Leguay, M. Lopez-Ramos, K. Jean-Marie and V. Conan, "An Efficient Service Oriented Architecture for Heterogeneous and Dynamic Wireless Sensor Networks," *The 33rd IEEE Conference on Local Computer Networks*, Bonn, 14-17 October 2008, pp. 740-747. doi:10.1109/LCN.2008.4664275

[10] R. S. Govindaraju, B. Engel, D. Ebert, B. Fossum, M. Huber, C. Jafvert, *et al.*, "Vision of Cyberinfrastructure for End-to-End Environmental Explorations," *Journal of Hydrologic Engineering*, Vol. 14, No. 1, 2009, pp. 53-64. doi:10.1061/(ASCE)1084-0699(2009)14:1(53)

