A Service-Oriented Integrating Practice for Data Modeling, Analysis and Visualization

Dan Cheng¹, Zhibin Mai¹, Jianjia Wu¹, Guan Qin², Wei Zhao¹
¹Department of Computer Science
dcheng, zbmai, jianjiaw, zhao}@cs.tamu.edu
²Institute of Scientific Computation
guan.qin@tamu.edu
Texas A&M University

Abstract

Many large-scale scientific computing projects share a data archive and retrieval requirement, a call to perform customized modeling tasks, and a need to visualize data in a geospatially referenced manner. Today, researchers generally design and procure hardware and software to provide the above services for each individual project. The rocketing cost of duplicate development and implementation, and the difficulty to collaborate due to the lack of standardization for working process and data format often leave something to be desired. This paper presents a service-oriented architecture for integrating software and hardware resources in a unified manner in order to deliver highly reconfigurable, on-demand customized scientific computing tasks. A practical system has been developed to evaluate the proposed architecture.

1. Introduction

Many large-scale scientific computing projects, such as weather simulation and forecasting, reservoir simulation and visualization, fire risk analysis and forecasting and etc., share a data archive and retrieval requirement, a call to perform customized modeling tasks, and a need to visualize data in a geospatially referenced manner. Today, researchers generally design and procure hardware and software to provide these services for each individual project. This approach may result in different problems. First of all, since the data formats among different projects are varied widely, it is difficult to collaborate among multiple entities though the collaboration is superbly desired. Moreover, as each of the systems is developed separately, duplicate efforts cause waste of valuable resources such as time, manpower, and money. Additionally, the individually developed systems are often hard to maintain since each of them is customized to specific needs and may require special domain knowledge to support the system.

Recent years have seen an increasing recognition of the concept of service-oriented architecture (SOA) as a framework for future application developments. As an architectural style with the goal to achieve loose coupling among interacting software agents [1], SOA is quickly prompted as a promising paradigm for developing dynamic, adaptive and extensible on-demand applications [2][3][4].

In this paper, we present our practice of design and implementation of a SOA model to address the problems mentioned above. Specifically, our newly proposed service oriented architecture and implementation has the following features:

1) **Facilitating the reuse of existing software modules.** Software reuse is of great importance in the development of software applications. Besides savings in re-development time and costs, reusing tested software modules also helps improving the overall system quality. Our service oriented architecture and implementation facilitate software reuse by providing a set of supporting tools to ease the effort of integrating existing software modules into new scientific computing applications.

2) **Supporting the separation of concerns.** A typical scientific computing system involves different roles, e.g. module developers, application integrators, system administrators, and end users. Module developers are responsible for developing the scientific simulation modules, e.g. *MM5* atmospheric prediction model [5], or the *BEHAVE* fire spreading model [6]. Application integrators focus on integrating different
simulation models into new applications, e.g. linking MM5 and BEHAVE models together for near real-time fire risk forecasting. The responsibility of system administrators is managing the system resource, e.g. data, applications, and hardware. The end users are those who actually use the integrated scientific applications and data. In many current systems, these roles are often mixed together, e.g. an application integrator is also the module developer and system administrator, and in some extreme cases, the end user. One major problem of mixing different roles is the requirement of knowledge on different areas. For example, to integrate different simulation models, an application integrator may need extensive knowledge about the models as well as system software and hardware configurations. Ideally, application integrators should be shielded from the low-level details of the system implementation and configurations as well as the scientific logics of the models; the end users should be able to run the applications without the need to understand how models are linked together and executed in the backend; the system administrators should be able to maintain the system without detailed knowledge about the simulation models or integrated applications. To address this "mixing role" problem, our service oriented architecture and implementation explicitly separate these user roles during design phase and provide extensive supporting tools to ease such separation during implementation.

3) Providing real-time quality services. Many integrated scientific computing applications are used by emergency-response managers as decision support tools for risk management. For example, when large storms like Hurricane Katrina and Rita are expected to come, a flooding risk management application that integrates an MM5 atmospheric prediction model and a flooding model may be used to decide retreat strategies. The value of such applications depends critically on the ability to deliver the simulation results in a timely fashion and any delay of delivering results could lead to huge losses on economy and human lives. Unfortunately, to the best of our knowledge, little, if not none, of the existing systems provides such real-time quality service in an explicit way. In our practice, we proposed an elegant real-time management structure to provide this type of service.

4) Promoting community collaboration. Large-scale scientific computing projects are usually distributed by nature with collaborators spreading over country or even world. It is very common that one collaborator needs data from others and meanwhile generates data needed by others. However, collaboration is often found to be difficult for the lack of standardization in data format and working process. Our service oriented architecture and implementation support collaboration through data related services, e.g. data discovery, data subscriptions, data retrieval, data integration, as well application related services, e.g. remote application execution.

The rest of this paper is organized as follows. Section 2 introduces our design strategies and the service-oriented architecture of our proposed system for data modeling and analysis. Section 3 presents the design and implementation details of the main system components. Section 4 illustrates the working process of the system via a use case, followed by related works in Section 5. Summary and future works are given in Section 6.

2. Service Oriented Architecture

In this section, we will introduce our design strategies and system architecture.

2.1 Design Strategies

To achieve the goals identified in the last section, we take the following strategies.

1). To facilitate the reuse of existing software modules, we use the service oriented architecture concept to encapsulate existing software modules into individual service elements. To ease the integration of the encapsulated service elements, we further utilize the concept of "workflow" to handle complex relationship of different service elements in a consistent way. Specifically, a workflow is composed of a set of workflow service elements. A workflow service element could be a wrapper of an existing simulation module, data retrieval module, or visualization module. A workflow specification defines the relationship among the elements, e.g. input/output relationship, execution sequence, resource requirement, and some other parameters, e.g. time interval of a simulation.

2). To support the separation of concerns, we explicitly classify users into different roles in our system, and design different interfaces and supporting tools to facilitate the separation.

3). To provide real-time quality service as well as other commonly used services, we model application workflows as real-time tasks with requirements on execution deadline and computing resources. We employ a priority based scheduler to arbitrate execution sequences of the workflows and we adopt an admission control component to avoid resource over provision.
4). To promote community collaboration on data and applications, we follow the service oriented architecture concept to wrap both data and applications as service objects. Data service objects hide the internal format of the data and export standardized public interfaces accessible by other collaborators. Through the interfaces, collaborators can perform many data related services in a consistent way, e.g. querying, subscribing, and retrieving data. Application service objects hide the internal implementation logics of the applications and provide collaborators with a set of application programming interfaces. This service oriented approach helps ease the collaboration effort. For example, changing of internal data format or application logics does not affect the collaborators as along as the service interfaces are intact.

2.2 System Architecture

Following the design strategies identified in the last section and the loose coupling principle of the SOA model, we design a five-layer service-oriented architecture as shown in Figure 1.

![Figure 1. Service Oriented Architecture](image)

The uppermost “User Interface Layer” provides users with data inquiry, data visualization, workflow orchestration, workflow execution, and other services, through different kinds of interfaces, e.g. web portal, java applet and web service. To allow flexible access control and "separation of concerns", the users are classified into different roles. For each service provided, we assign different level of access permissions for each role using declarative, role-based access-control mechanisms.

The “Application Enable Layer” provides services to "User Interface Layer". Services provided by this layer include various enables for data modeling, analysis and visualization, e.g., numerical models for data analysis, toolkits for data translation and mapping, visualization packages for data visualization, etc. Another large category of enables are the workflow orchestration tools. These workflow orchestration tools are responsible for wrapping the legacy scientific modules as basic service elements and for orchestrating the interactions among the different service elements.

The “Management Layer” contains a set of core services components. Major components in this layer include data management components, real-time management components, workflow management components, resource management components, and cross-cutting service components. Data management components include a data catalog service component and a data archive service component. Data catalog service component is responsible for the maintenance of the meta-data information about the archived data. Data archive service component is responsible for data retrieval and storage. Real-time management components are composed of an admission control service component and a real-time scheduler. Admission control service component is introduced here to guarantee the timely completion of the tasks in the system and the scheduler is the unit that resolves the resource usage conflicts. As mentioned earlier, we model the different applications as workflows in our system. The workflow management component is responsible for maintaining a catalog of the available workflows. The actual execution of the workflow is controlled by the workflow engine service component. Similar to the workflow management component which manages the "software" resources, the resource management component maintains the configuration information of hardware resources through the resource catalog service component. The cross-cutting service components include security, user management monitoring and reporting, and others.

The “Resource Access Layer” contains service components needed to access lower level computing resources, e.g. high performance computing clusters, graphic servers, database servers, storage, network connectivity. Major components in this layer include resource management middleware, data query service component, data transport service component, and remote access service component.

The lowermost “Resources Layer” refers to all the physical resources, such as computing, graphic, database, storage, network connectivity and so on. For many large-scale scientific computing projects, the physical resources might be widely dispersed over the Internet.
3. Implementation Details

Based on the service oriented architecture, a practical system has been implemented and deployed in the Advanced Data Center at Texas A&M University to serve the computational science community through web portal [7]. The system development is based on Java 2 Platform, Enterprise Edition (J2EE) framework [8]. We use Enterprise Java Bean (EJB) application server, JBoss AS [9] with web service enabled, as our service container and JSP/Servlet server, Tomcat [10] with Struts [11], as our web portal container. The backend support system includes two Portal Batch System (PBS) [12] clusters with 64 nodes of dual-AMD 64-bits processors for intensive computing, tens of high performance computers as system application servers, database servers, data transport servers, and etc., 40 terabytes storages for huge volumes of data archive and databases, and Gigabit Ethernet network.

Figure 2. Relationship of Service Components

Due to space limitations, in the following paragraphs, we will focus on the implementation of the main service components in the application enable layer and management layer. Figure 2 gives a detailed view on the relationship of the components. Workflow catalog manages the information related to the workflows in the system, e.g. available workflows models, access control permissions, resource requirements. Resource catalog maintains current system hardware configurations and their utilization information. Data catalog holds the meta-data about the archived data in the system. These different catalog services are used by the workflow orchestration, admission control, workflow engine, scheduler, resource access and data archive services which will be discussed in detail in the following subsections.

3.1 Workflow Orchestration

Workflow orchestration is used to wrapping existing software modules as service elements and to handle the dependency relationship of these service elements in complex workflows. The workflow orchestration includes two sub-components: application wrapper and Directed Acyclic Graph (DAG) modeler. Application wrapper wraps the existing software modules into service elements by on-the-fly creation of uniform access interfaces and dynamic generation of input parameters, e.g. from web user input. The DAG modeler models the relationship of the service elements as directed acyclic graph. The construction and updating of the DAG graph is done using a drag & drop workflow GUI editor in user interface layer. The constructed DAG graphs are stored in the workflow catalog in a standard xml format.

3.2 Admission Control

Admission control is a component for real-time service guarantee. Before any workflow can be executed in the system, a user needs to submit a request to the admission control service with desired deadline requirement, i.e. when to finish the workflow. Admission control is needed since the system resource is limited and accepting any incoming workflow may lead to deadline missing of previously admitted workflow. In our system, an admission request includes three kinds of information: 1) timing information, such as start time, deadline, expiration time, and execution periods; 2) parameters needed for estimating the execution time of the workflow; and 3) other general information, e.g. network bandwidth and data storage demands. Upon the admission request of a workflow, the admission control service component checks whether the new workflow and the admitted ones can meet their deadlines. If yes, the admission control will return success. The workflow will be added to the admitted workflow table through workflow catalog service component. A timing event will also be registered to invoke the workflow according to the admission request information. If not, the admission control returns failure and the admission request is rejected. Therefore all the accepted
workflows in the system will be guaranteed to finish before their required deadlines.

3.3 Workflow Engine

The workflow engine is the component responsible for interpreting the workflow specification and releasing of individual jobs. The workflow engine is a multi-instance service. A workflow engine instance is created at the time a workflow is invoked and is destroyed after the workflow is finished or removed. In the rest of this paper, we use workflow engine or workflow engine instance interchangeably to represent an instance of workflow engine service. After a workflow engine instance is created, it generates a serial of jobs based on the workflow specification. Workflow engine instance monitors the execution progress of the jobs and releases jobs based on the specification as well as dynamic execution status. Most of the time, the workflow engine instance will be in sleep stage and is waken up by the resource access service component when a job is finished normally or terminated unexpectedly. Once awaken, the workflow engine instance decides which job should be released next or start the termination process if all the jobs have been finished.

3.4 Scheduler

All the individual jobs generated by the workflow engine instances are appended to the incoming job queue managed by the scheduler. The scheduler picks the jobs in the job queue one by one and inserts them into the outgoing job queue. The jobs in the outgoing job queue are arranged based on the scheduling policy, e.g. high priority job is in the front of the queue. We use static priority scheduler in current implementation. The scheduler maintains a job queue for each cluster and the jobs in the queue are arranged based on their priorities. After admission, each workflow is assigned to a specific cluster for execution. For each new job, the scheduler finds the cluster on which it will run and inserts the job to the outgoing job queue corresponding to the cluster. Since the queue is ordered based on their priorities, the insert operations is very efficient using the binary search algorithm with a complexity of O(log(n)).

3.5 Resource Access

The resource access service is responsible for monitoring the cluster and picking the jobs from the outgoing job queue and dispatching them to the cluster once the cluster is free. The resource access service also monitors the progress of the jobs running on the cluster. The resource access component is implemented as an active process that periodically interacts with the resource management middleware, e.g. PBS. The resource access service first checks the job status when it is waken up. It notifies the workflow engine if any job is finished. Then the resource access service checks the cluster resource status. If the cluster is free (empty or partially empty), it will perform the following two operations: (1) notify the scheduler and the workflow engine; (2) get the next job from the scheduler. Note that the resource access service may not get a job from the scheduler. If this is the case, the resource access service will change to sleep mode until the next polling time in order to save CPU cycle.

3.6 Data Archive

Data archive service is responsible for dynamic data retrieval and storage. It prepares the data required by the workflows in an accessible format, processes the output data from the workflows, and archives them back to the data storage.

In the next section, we will use an atmospheric prediction and visualization example to illustrate how the different components discussed above are integrated together to accomplish a complex workflow with the data archive, analysis, and visualization.

4. Case Study

In this section, we use an example, an integrated workflow of MM5 atmospheric prediction and visualization as shown in Figure 3, to illustrate how our service oriented architecture and implementation support software model reuse, separation of concerns, real-time services, and collaboration.

Before going further, let us briefly introduce the software modules of integrated workflow. The MM5 is a limited-area, nonhydrostatic, terrain-following sigma-coordinate model designed to simulate or predict mesoscale atmospheric circulation. The model is supported by several pre- and post-processing programs, such as terrain, regrid, interp, etc. The output of the MM5 prediction model is gigabyte volume of digital data. It is very difficult for users, if not impossible, to discover the knowledge by reading the MM5 output data directly. The data need to be converted to intuitive images using a visualization package, GEMPAK [13]. This conversion process is called MM5 visualization.
The first step needs to be done by the application integrator is to create the basic service elements that are wrappers of existing software modules or scripts. After all the service elements are created, the application integrator can build a workflow to connect all the service elements together. The application integrator creates the basic service elements and workflow using a workflow editor as shown in Figure 4. Once built, the workflow will be stored into the workflow catalog.

When a workflow is invoked, a workflow engine instance will be created to parse the workflow specification, create job(s) and handle the execution sequence of the job(s). As shown in figure 3, three jobs are created by the MM5 workflow engine instance. Each job can include one or more applications. For example, the MM5 preprocess job includes three sequential computing applications: terrain, regrid and interp; while MM5 prediction analysis job only includes one parallel computing application: mm5. After the workflow specification is parsed and its jobs are created, the workflow engine instance will start submitting the created jobs to the scheduler. The scheduler prioritizes all the jobs submitted by the workflow engine in its job queues according to certain priority policies. In the case of the MM5 workflow, the first job passed to the scheduler is the MM5 preprocess job. The resource access service notifies the scheduler if it discovers free resource; then it fetches the highest
priority job from the scheduler’s job queues and starts it via the resource management middleware.

After the MM5 preprocess job is submitted to run via the resource management middleware, the resource access service monitors the execution of the job until it is finished or terminated unexpectedly. Once the MM5 preprocess job is finished, the resource access service notifies the MM5 workflow engine instance. The MM5 workflow engine instance updates its status upon receiving the notification and checks which job should be released next. In this case, the MM5 prediction analysis is the next one and the workflow engine instance will submit it to the scheduler. The rest of the process is the same as the MM5 preprocess job. When all the jobs of the MM5 workflow instance are finished, the workflow instance will be terminated. Before the termination, the instance performs the following steps: 1) notifying the admission control; 2) archiving result if required; 3) storing log files and information about instance, such as final status (success or failed), start time, finished time etc., into database; 4) notifying user if required. A few samples of the MM5 visualization data are shown in Figure 5.

From the above example, we can notice that compared to the traditional integration process that requires significant human involvement, the new solution greatly eases the effort of reusing of existing software module through the workflow orchestration tool. It is easier for the collaboration among scientists dispersed all over the country and even world through web portal or web service. Moreover, with the new proposed service oriented architecture, the application integrators can wrap existing software modules without the need to understand the detailed logics of the modules; the system administrators can easily maintain system without worrying about how the workflows are integrated; and end users can customize, execute the workflows and view the results by simply clicking buttons through the web portal without the need to worry about the complex backend processing. Furthermore, by employing the admission control service, the system can provide guarantees on the completion time of the applications. Additionally, solving a new weather modeling task is simply a creation of a new workflow with its parameters being appropriately set. This significantly reduces the development cycle and accordingly development cost as compared to the traditional coding approach. Meteorology scientists are also allowed to monitor and control the execution of an MM5 workflow in a real-time manner which is rather difficult if not impossible in the traditional systems. Finally, the new solution is more maintainable and reliable as the MM5 integrators are solely working on the level of modeling process which is composed by previously approved and bug-free applications.

Figure 5. MM5 Visualization Data

5. Related Works

LEAD [14] is a large scale effort to build infrastructure that allows atmospheric science researchers to dynamically and adaptively respond to weather patterns to produce better-than-real time predictions of tornadoes and other “mesoscale” weather events.

LEAD has some similarities with our proposed system. Both use workflow technology to reuse the existing software modules to construct and auto-handle larger scale scientific analysis. Both also support event driven to dynamically and adaptively respond to the predefined external events to conduct real time analysis. However, the difference lies in the explicit separation of user roles and real-time guarantee mechanism. Our proposed system explicitly classifies users into different roles and provides different interfaces and supporting tools to facilitate the separation. Our proposed system uses scalable admission control mechanism to resolve resource competition issues in order to guarantee the admitted workflows to be completed by their deadlines.

GEON [15] is developing cyberinfrastructure for integrative research to enable transformative advances in geoscience research and education. Similar to LEAD, GEON is also based on service-oriented architecture, with an emphasized focus on supporting geoscience. Different from LEAD and GEON, our proposed system is a generic system for scientific research. It imposes no constraints on the kinds of the supported scientific applications and data even though it is currently deployed in Advance Data Center to support atmospheric research.

GAP [16] is a system to enable legacy scientific applications on computing grids using a service-
oriented architecture. It provides approach to automatically turn an application into a service and generate the user interface from a XML description for the application without the need of coding. Like our proposed system, GAP explicitly classifies users into different roles, such as application specialist, system administrator and users. Similar to GAP, our system can automatically wrap the application to service element and generate user interface without the application integrator writing any codes. However, our system frees the application integrator from writing tedious xml descriptions, which describe how to turn the application to service element, via the workflow module editing features in the workflow GUI editor. In addition, GAP currently does not support real-time guarantee service and workflow mechanism although workflow is identified in their future plan.

6. Final Remarks

The concept of service-oriented architecture is both an outgrowth of the early advances in information technology and a promising framework for future evolution of the application development. This paper presents a practice of designing and developing a system for data modeling, analysis and visualization under the framework of service-oriented architecture. Following an initial discussion of the SOA model, a five-layer service oriented architecture is proposed. Based on the general architecture, the design and implementation of some core service components are presented in detail. Finally, we evaluate the proposed solution with a practical example MM5 weather prediction workflow which illustrates the interactions between components within a single layer and from different layers. We plan to continue on the research about service-oriented integration and collaboration in the specific large-scale scientific computing and modeling area. Besides weather forecasting, many other applications such as environmental forecasting, reservoir simulation share the similar business processing models but with different emphasizes. Moreover, another interesting research topic might be the relationship between SOA and contemporary software architectures, the effects of SOA on the domain of software engineering and methodologies.

References