A Service Oriented Architecture for Exploring High Performance Distributed Power Models

Yan Liu, Jared M. Chase, and Ian Gorton

Pacific Northwest National Laboratory
Richland WA 99352
{yan.liu,jared.chase,ian.gorton}@pnnl.gov

Abstract. Power grids are increasingly incorporating high quality, high throughput sensor devices inside power distribution networks. These devices are driving an unprecedented increase in the volume and rate of available information. The real-time requirements for handling this data are beyond the capacity of conventional power models running in central utilities. Hence, we are exploring distributed power models deployed at the regional scale. The connection of these models for a larger geographic region is supported by a distributed system architecture. This architecture is built in a service oriented style, whereby distributed power models running on high performance clusters are exposed as services. Each service is semantically annotated and therefore can be discovered through a service catalog and composed into workflows. The overall architecture has been implemented as an integrated workflow environment useful for power researchers to explore newly developed distributed power models.

Keywords: Service oriented architecture, high performance computing, power grid.

1 Introduction

Electrical power grids are increasingly incorporating high quality, high throughput sensor devices in the power distribution network. These devices are driving an unprecedented increase in the volume and the rate of information available to utilities. For example, a new Phasor Measurement Unit (PMU) sensor produces up to 60 samples per second, in contrast to existing conventional Supervisory Control And Data Acquisition (SCADA) measurements generated every five seconds or longer. The dramatic growth of these high quality sensors demands new power grid models for real-time predictions, with the time to solution in the range of ten milliseconds to one second. This is a radical reduction from the current ranges of two to four minutes.

Given the sheer size of the power grid, the solution of mathematical power models requires significant time to solve. With conventional power grid operations, the core power model of state estimation can only be updated in an interval of several minutes – much slower than the measurement cycle in seconds. However, a power grid could become unstable and collapse within seconds [5]. Therefore, the current state estimation is not fast enough to predict real-time power grid behavior and respond to emergencies such as the 2003 US-Canada Blackout [6].

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 748–762, 2012.

High Performance Computing (HPC) techniques are essential to handle such a dramatic increase in data size and rate and to meet the demand of the real-time predictions. Using the Western Electricity Coordinating Council (WECC) model as an example, for state estimation, the solution time is about five seconds [7], while for contingency analysis, about a 500-times speedup was achieved with 512 processors [8][9][10]. However, HPC capabilities usually can only support the computing demand at the regional scale; while real-time predictions model for power grids requires computing resources at the continental scale. Thus, the critical issue is how to build a power model that connects HPC-enabled regional-scale distributed power grids.

Many researchers have worked on hierarchical or distributed state estimation, such as [1-4]. Their work is mainly focused on the distributed state estimation algorithms and their efficiencies. There is no such a tool that has a capability to deploy the distributed power models in a HPC-based distributed computing environment.

At Pacific Northwest National Laboratory, our research aims to design a distributed system architecture that leverages parallel computing capacities to support the demand of real-time computing for distributed power models. In our previous work, we have investigated a method to partition the overall topology of a power grid into several connected sub-areas [20]. We have also implemented a distributed state estimation algorithm that allows each sub-area to compute its local state estimation on a HPC cluster. In this algorithm, adjacent sub-areas exchange their own state estimation results so that global state beyond the local area can be computed [21].

Our previous implementation has the peer-to-peer data exchange through middleware that connects state estimators through TCP sockets [20]. Therefore the data communication logic has been hard-coded in the state estimation code, which needs to know the destination of the intermediate results. This mechanism has limitations for validating the distributed state estimation algorithm, as to study the algorithm behavior, different partitions of the same power grid network should be tested [11]. The partition may change how the sub-areas are connected and hence the peer-to-peer interactions. Hardcoding the data communication in the state estimation code is certainly not sufficiently flexible to represent the procedural steps of distributed state estimation.

In this paper, we present our solution of exposing individual state estimators as a service and connecting distributed state estimation services into a workflow. The inputs and outputs of each service are semantically defined and therefore can be registered and discovered through a service catalog. Connected services are guaranteed to match the input and output types. The overall architecture has been implemented as an integrated workflow environment, including a workbench for an engineer to compose and monitor a workflow, a service repository to register and search a matching service, and job management for launching HPC jobs in a remote cluster. This workflow environment enables power grid researchers to explore newly developed distributed power models. In this paper, we present our engineering solution to realize this service oriented design. We discuss our experience and summarize the benefit and limitation of this solution.

2 Background on HPC-Based Distributed State Estimation

We briefly introduce the distributed state estimation algorithm. A state estimator basically solves the non-linear state estimation equations as:

$$z = h(x) + e \tag{1}$$

where z is the measurement vector of dimension; x is the state vector; e corresponds to the measurement errors, and h is a vector of non-linear functions which relate states to measurements [12]. Approximately, the state estimation problem can be solved by obtaining the solution to the following equation:

$$z = Hx + e \tag{2}$$

where H is the states to measurements matrix for the entire power system. The data resources include power flow-injections and voltage magnitudes. In the distributed version, the entire power system is decomposed into m non-overlapping sub-areas and each sub-area can run its local state estimation algorithm and also exchange data with neighboring sub-areas to reevaluate its local state estimation solution. Sub-areas are connected via tie lines. Correspondingly, the matrix H and the vector z are also partitioned into m parts as follows and each part is responsible for a subsystem of the entire power system.

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_m \end{bmatrix}, z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}, e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix}$$

Figure 1 shows the typical tie-line of the IEEE 118-bus system [13] that is used in this paper. The specific procedural steps are as follows, assuming Area 1 as internal and Area 2 being external.

1. Perform the state estimation in individual areas not directly connected to each other (i.e., Area 1 and Area 3) to create the pseudo-measurements of the internal boundary buses connected via tie-lines to other areas (i.e., real and reactive power injection P_{inj} and Q_{inj}, and voltage V). These pseudo measurements create a network equivalent attached to the external system.
2. Run the state estimator of the external connected areas (e.g. Area 2) with measurements obtained in step 1 to calculate new tie-line data. These new data represent an equivalent network attached to external areas. They are used to resolve the influence from the external states on the internal states.
3. Run the state estimator for internal areas. Verify that the boundary buses and tie line estimates are within tolerances; otherwise re-run the external state estimator with updated tie-line information (i.e. step 2).

The data exchanges between state estimators are depicted in Fig. 2 based on the partition shown in Fig. 3. Note that the partition of a bus system affects how sub-areas are connected by tie-line buses [11] and consequently it affects the interaction between the state estimators according to the distributed state estimation algorithm above.

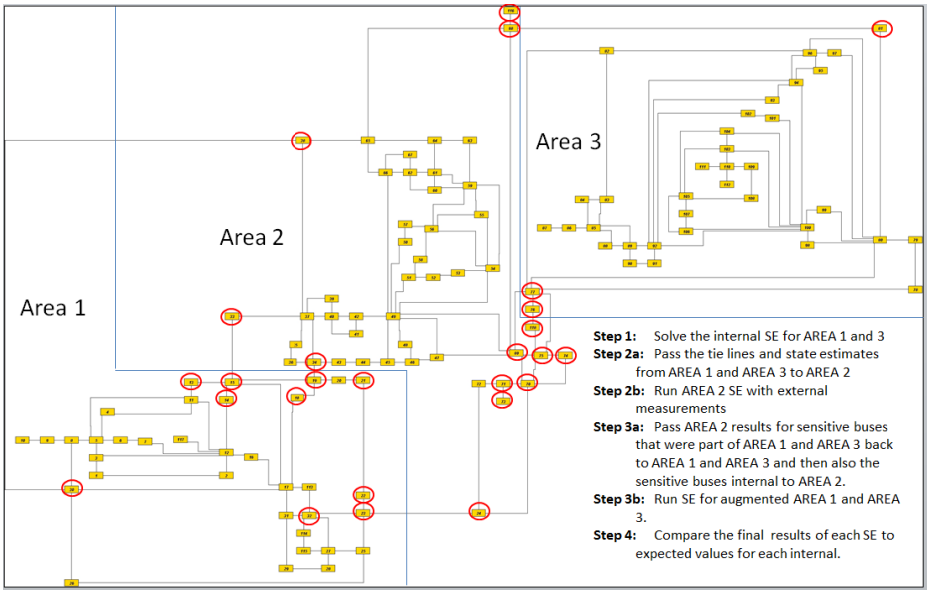

Fig. 1. Partitioning the IEEE 118-bus system into three areas

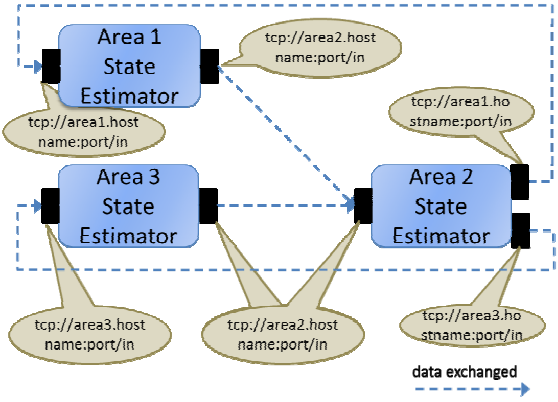

Fig. 2. Data communication between distributed state estimators

To understand the interaction, it is helpful for a power engineer to model the procedural steps of interaction as a workflow and observer the workflow execution. Thus a workflow environment allows the power engineer to examine in detail the step by step execution of the state estimation algorithm and understand not only final results but also the intermediate outputs.

For each state estimator, a Message Passing Interface (MPI)-based code is run on a computational cluster to solve the estimation metrics in (1) and (2). A master processor fetches the data from the data communication infrastructure, partitions the data, and dispatches a block of data to a number of worker processors. All the processors

then run the computing tasks in parallel. The state estimation results are gathered after all the computing tasks are completed. More details can be found in our previous work [21].

3 The Service Oriented Architecture

The design of the architecture aims to make individual state estimators autonomous. This means the boundaries between state estimators are explicit, defined in terms of data inputs and outputs. In this architecture, the state estimator does not need to deal with the data communication with its adjacent neighbors. The data communication is managed by middleware that manages individual state estimators as registered services. Thus the state estimators can be cataloged and retrieved based on its semantic annotation (especially based on the types of inputs and outputs). The implementation of this design follows service oriented architecture principles. With this design, a power engineer who has the knowledge of the distributed state estimation algorithm can connect state estimators into a workflow aligned with the algorithm. The architecture allows the engineer to launch a state estimator on a HPC cluster and observe the data exchange between state estimators.

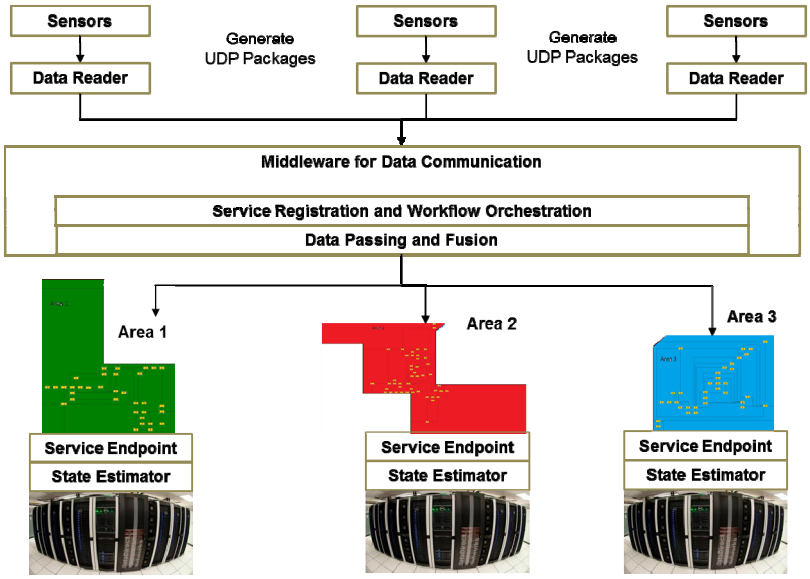

Fig. 3. Service oriented architecture of distributed state estimation

3.1 Architecture Overview

The architecture is depicted in Fig. 3. This architecture consists of three key parts: (1) reading and parsing data streams from the sensors at top of Fig. 3; (2) exposing state estimators as services and coordinating the distributed state estimation workflow in

the middle of Fig. 3; (3) launching HPC jobs to run state estimation at the bottom of Fig. 3. The first part has been implemented by scripts and Java applications following the standard IEEE C37.118 protocol to extract specific measurements of bus lines for a power system. In this section, we focus on the other two parts.

3.2 Semantic Service Definition and Registration

The state estimators have been implemented as Fortran MPI programs. To expose a state estimator as a service, we need to define the input and output ports for the state estimator and provide the data type of each port. Input and output ports are defined through constructing an OWL document as follows. In the example below, the semantic type for the input port of the service at area 1 is defined in an OWL Class called `Area1Input`. This OWL class has one property called `hasAREA1InputFilename`, and it is of the type `Area1InputFile`.

```
<owl:Class rdf:about="#Area1Input">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource=" #hasAREA1InputFilename"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:someValuesFrom rdf:resource="#Area1InputFile"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

List 1 Semantic definition of the input port

The semantic type of resource `Area1InputFile` further defines that its data type is string.

```
<owl:Class rdf:about="#Area1InputFile">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasFilename"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

List 2 Semantic definition of the data type

The purpose of defining the semantic type of a port such as `Area1InputFile` is to restrict the connection of two state estimator services unless they both have ports that are `Area1InputFile` strings. Therefore, the rules of connecting state estimators can be expressed by means of the semantic type of ports. Also, typed ports allow searching state estimator services that match the exact type of ports. For example, the resource class `Area1InputFile` can be used as an attribute to search with. Similarly, an output port can be defined.

After the input and output ports are defined, the next step is to define the service itself. We use an open source system called SADI (Semantic Automated Discovery and Integration)[1] to generate the service stub code essential for wrapping a state estimator as a service. SADI is a framework for discovery of, and interoperability between, distributed data and analytical resources. It combines simple, stateless, GET/POST-based Web Services with standards from the W3C Semantic Web initiative. Using SADI, a Java Servlet class is generated and annotated given the service name specified. Previously defined input and output ports are now annotated as `@InputClass` and `@OutputClass` in the service definition class.

```
@Name("Area1Part1Service")
@Description("Area 1 Part 1 Service")
@ContactEmail("jared.chase@pnnl.gov")
@InputClass("http://neptune.pnl.gov:8090/powergrid2.owl#Area1Input")
@OutputClass("http://neptune.pnl.gov:8090/powergrid2.owl#Area1Output")

public class Area1Part1Service extends SimpleSynchronousServiceServlet {
…
```

List 3 Generated service code

In this generated service definition class, the logic for launching state estimation jobs on a remote cluster is implemented in the `processInput` method.

```
@Override
public void processInput(final Resource input, final Resource output)
{
    final int numCores = 1;
    String filename = input.getProperty(Vocab.hasAREA1InputFilename).getString();
    final String[] inputfiles = { filename };

  try {

    JobManagerService jm = new JobManagerService();
    String[] outputfiles = jm.launchJob("node_a_part1", "local", numCores, inputfiles);

    String outputfile = outputfiles[0];
    System.out.println("Area1Part1Service " + outputfile);
    output.addLiteral(Vocab.hasAREA1OutputFilename, outputfile);

  } catch (final IOException e) {
    e.printStackTrace();
  }
}
```

List 4 Service implementation to launching state estimation job

The input and output parameters for the `processInput` method are passed as an RDF formatted document. To parse the RDF document and extract data of interest, we have developed a Java class `Vocab` using the Apache JENA[2] API. JENA is an Apache framework for building Semantic Web applications that includes a Java API for reading and writing RDF documents.

[1] `http://sadiframework.org/content/`
[2] `http://jena.apache.org/`

```
@SuppressWarnings("unused")
private static final class Vocab
{
  private static Model m_model = ModelFactory.createDefaultModel();

  public static final Property hasAREA1OutputFilename =
m_model.createProperty("http://neptune.pnl.gov:8090/powergrid2.owl#hasAREA1OutputFilename");

  public static final Property hasAREA1InputFilename =
m_model.createProperty("http://neptune.pnl.gov:8090/powergrid2.owl#hasAREA1InputFilename");

  public static final Property hasFilename =
m_model.createProperty("http://neptune.pnl.gov:8090/powergrid2.owl#hasFilename");

  public static final Resource Area1Input =
m_model.createResource("http://neptune.pnl.gov:8090/powergrid2.owl#Area1Input");

  public static final Resource Area1Output =
m_model.createResource("http://neptune.pnl.gov:8090/powergrid2.owl#Area1Output");

  public static final Resource string =
m_model.createResource("http://www.w3.org/2001/XMLSchema#string");

  public static final Resource Area1OutputFile =
m_model.createResource("http://neptune.pnl.gov:8090/powergrid2.owl#Area1OutputFile");

  public static final Resource Area1InputFile =
m_model.createResource("http://neptune.pnl.gov:8090/powergrid2.owl#Area1InputFile");
}
```

List 5 Creation of property and resource mapping

The `Vocab` class is developed to create property and resource instance variables to tie the input and output data according to the semantic structure of the service. As shown in List 5, the property `hasAREA1InputFilename` in List 1 and the resource `Area1InputFile` in List 2 are tied to the URLs of the OWL file. The property or resource variables of `Vocab` can be used to extract the input data and package the output data using a well formed RDF document representation, for example

```
String filename =
  iput.getProperty(Vocab.hasAREA1InputFilename).
  getString()
```

3.3 Job Launching

The `processInput` method defined in previous section invokes the job launching software we have developed. This job launcher simplifies the remote job launch, monitoring, and results handling for the state estimation scripts that are located on a remote cluster. Fig. 4 below shows a high level view of the structure and the associated steps to configure the job launcher. The first step is to configure the machine registry. The configuration files include parameters of machine name, job scheduler, installation directory of simulation code(s), number of compute nodes to use, and so on. The second step invokes the job launching function, which writes the status of the job to log files.

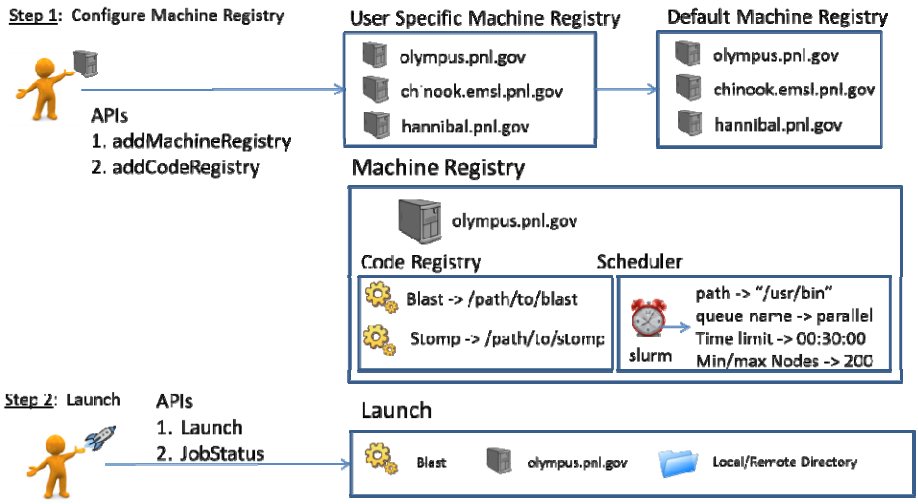

Fig. 4. Structure of job launching

The API to invoke the job launcher is the `launchJob` method in the `JobManagerService` class. The `launchJob` method takes 4 parameters.

```
JobManagerService jm = new JobManagerService();
String[] outputfiles = jm.launchJob("node_a_part1", "local", numCores, inputfiles);
```

The first parameter is Code Registry, a string that is an identifier for retrieving information from a code registry; in this case it is `node_a_part1`, a Perl script to submit an job to a cluster queue. The information that `JobManagerService` retrieves includes the submit/launch script as well as the names for the code's input and output files.

The second parameter is Machine Registry, a string that is an identifier for retrieving information from a machine registry; in this case "local". The information the `JobManagerService` retrieves includes the name of the machine, user account to launch the job with, ssh keys (if required), number of nodes, queuing system, installation location of code, time limit, and other required parameters.

The third parameter is the number of cores to use to run a simulation. This parameter is only used when launching a job through a queuing system.

The fourth parameter is an array of input files that will be used to run the job. These files are transferred from the machine that is running the SADI services to the remote cluster machine. In our example the input file is passed in this list as an argument into the state estimator.

3.4 Service Registration

Once a service is defined and implemented, it is deployed to a web application server such as Tomcat, and then registered using the SADI service registry. The SADI service registry first validates the service to make sure semantic types are correctly

defined. Then the SADI service registry registers the semantic types inside the service catalog. Fig. 5 below shows the services that are registered within the SADI service registry along with the semantic types of their input and output.

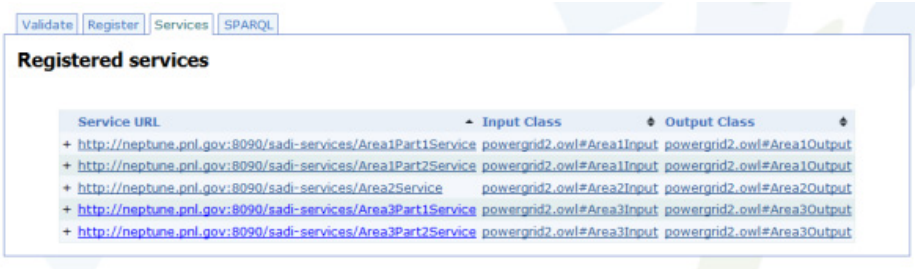

Fig. 5. Service registry

4 Composing a Workflow

Once services are created and registered for state estimators, they can be composed using workflow tools that support Web Services. In our work, we use the Taverna

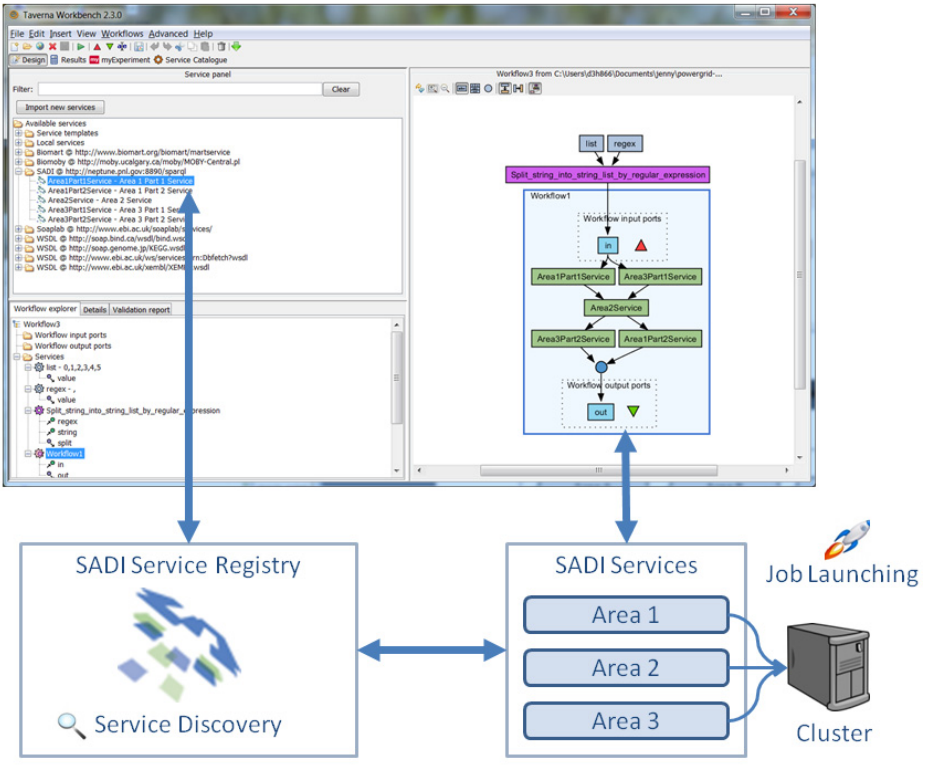

Fig. 6. Integrated Workflow Environment

Workflow Management System[3] to compose a workflow. The overall integrated workflow environment is shown in Fig. 6. This environment consists of the Taverna workbench, SADI service registry, state estimation services registered in SADI, and the job launcher embedded in the service implementation. The details of how to build and run a workflow are discussed below.

Taverna supports a SADI plugin, therefore the services that are created and registered to SADI (as described in section 3) are available in Taverna as shown in Fig. 6. Since Area 1 and Area 3 run state estimation in two separate steps: first, running its own local area state estimation, and sending its estimation of measurement on the tie-line buses to Area 2; and second, running state estimation again when Area 2 sends back its estimation of measurement on the tie-line buses. Therefore, Area 1 and Area3 have two separate services, such as `Area1Part1Service`, `Area3Part1Service` and `Area1Part2Service`,`Area3Part2Service` respectively.

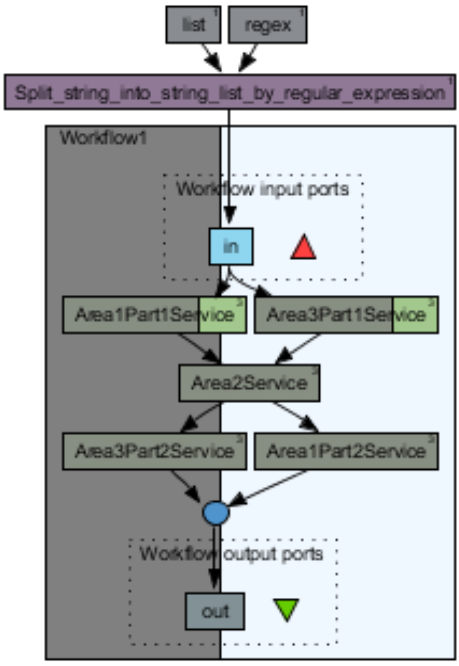

Fig. 7. Workflow composition and execution

Taverna allows SADI services to be used as drag-and-drop components to build workflows. Each workflow component has semantically typed input and output ports that can be used to perform service discovery. Service discovery guides users by helping them find components with an output port that has the same semantic type as the input port of an existing component.

To model the distributed state estimation workflow in Taverna, a power engineer needs to compose the workflow components according to procedural steps of distributed state estimation. In this paper, we focus on modeling the steps of distributed state estimation introduced in Section 2. These steps are run iteratively as time passes. This means for every time step of interest, these steps are invoked. Therefore, we introduce a hierarchy to the workflow. The top level of the workflow contains a list of values such as the timestamps to run the workflow steps. For each value in list is parsed, a sub-level workflow is invoked. The sub level workflow models the steps of distributed state estimation.

The sub workflow receives a value from the list in the top level workflow and triggers the `Area1Part1Service` and the `Area3Part1Service`. When a service is invoked within Taverna, it calls the `JobManagerService` to launch the state estimation jobs in a remote cluster. After both these components complete, they send

[3] `http://www.taverna.org.uk/`

their output to the `Area2Service`. After the Area2Service receives both inputs it executes and sends a separate result back to `Area1Part2Service` and `Area3Part2Service`. When both these services are finished a message is sent to the output port of the top level workflow to indicate that one iteration of the workflow is finished.

During the execution of a workflow, the result view of the Taverna workbench shows the progress of the workflow by changing colors as the workflow components execute (see Fig. 7). Once the entire workflow is finished executing, the lower result view appears populated with results data as shown in Fig. 8. Intermediate inputs and outputs for each component are available to navigate within a tree view. The separate iterations for each sub workflow execution are available.

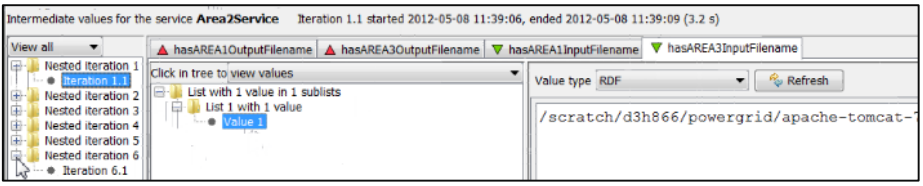

Fig. 8. Monitor workflow execution

By viewing the output file from the output port of each workflow component, a user is aware of the status of interactions between distributed state estimators. The output files in the result view allow a user to validate the distributed state estimation algorithm and identify errors of the algorithm itself.

5 Reflection and Discussions

The major benefit of using semantic services is that semantic rules for connecting services can be enforced and service discovery through the SADI service repository can be executed within an integrated workflow environment. This helps guide the user through the process of workflow composition and execution, especially when the power model evolves and consequently the data flow changes. For example, each site of the distributed state estimation involves 4 to 7 different input datasets, 2 scripts for control actions and 2 to 4 exchanges of intermediate data. For any distributed state estimation involving more than 2 sites, it is impractical to manually execute the scripts and coordinate the data flow. With this simple sematic enhancement, the workflow environment allows changing of the process through workflow composition and any violation against the composition rule is prevented.

Some of the drawbacks of using SADI services include that they use a Servlet protocol as opposed to SOAP or REST which are more common protocols used in service oriented architecture. SADI does include documentation for handling integration with SOAP services using custom code inside the SADI servlet. Another limitation is that SADI services use RDF structure data to send and receive messages. In our solution, we have developed custom code to unpackage and repackage the data specific to measurements of power grid to send to a state estimation service.

6 Related Work

The power grid is evolving to the future power grid that encompasses a business strategy within the electric utility industry for incorporating intelligence in the power distribution network. This evolution creates major challenges in the increasing complexity of the bulk power grid to respond to demand growth, support renewable energy sources and satisfy the requirements for enhanced, adaptive service quality. One of those challenges is concerned with data exchange. A service oriented architecture (SOA) has been recently adopted and reported to cope with comprehensive data exchange in layers between different stakeholders [14-16]. According to [17], SOA and Web services offer a flexible and extensible approach to integration of multiple, often autonomous, data sources and analysis procedures. Service oriented technologies support interoperability with other power grid frameworks (e.g., SCADA) through the use of emerging standards including Common Interface Model (IEC 61970/61968) and OPC Unified Architecture (IEC 62541). CIM is domain-specific data model and can be seen as a domain ontology for the utility domain. The OPC UA can be viewed as a platform for interoperability and service-based communication.

Within this background, semantic web services facilitate automation of service discovery and execution. However, the discovery could fail even if there are matching web services within the repository. For this purpose, the provision and consumption of a service should be described in a much more specific way in terms of semantics [18].

7 Conclusion

In this paper, we present a service oriented architecture for connecting distributed power models (e.g. state estimation) in a workflow environment. We define sematic services with input and output types to constrain the data exchange rules. The semantic services are mapped to state estimators running on high performance clusters. These services are registered in service repository, and discovered and composed in a workflow. Overall this integrated workflow environment separates the concerns of running individual state estimators and orchestrating their interactions in a workflow. Our future work will enhance the semantic representation of power models and support standards of CIM (IEC 61970/61968) in the service oriented architecture.

References

1. Cutsem, T.V., Howard, J.L., Ribben-Pavalla, M., El-Fattah, Y.M.: Hierachical State Estimation. International Journal of Electric Power and Energy Systems 2, 70–81 (1980)
2. Cutsem, T.V., Howard, J.L., Ribben-Pavalla, M.: A Two-level Static State Estimator for Electric Power Systems. IEEE Trans. on PAS 100, 3722–3732 (1981)

3. Cutsem, T.V., Ribbens-Pavella, M.: Critical Survey of Hierarchical Methods for State Estimation of Electrical Power Systems. IEEE Trans. on Power Apparatus and Systems 102(10), 3415–3424 (1983)
4. Habiballah, I.O., Irving, M.R.: Multipartitioning of Power System State Estimation Networks Using Simulated Annealing. Electric Power Systems Research 34, 117–120 (1995)
5. Huang, Z., Guttromson, R.T., Nieplocha, J., Pratt, R.G.: Transforming Power Grid Operations. Scientific Computing 24(5), 22–27 (2007)
6. U.S.-Canada Power System Outage Task Force, Final Report on the August 14, 2003 Blackout in the United State and Canada: Causes and Recommendations (April 2004), https://reports.energy.gov/
7. Chen, Y., Huang, Z., Liu, Y., Rice, M., Jin, S.: Computational Challenges for Power System Operation. Accepted by Hawaii International Conference on System Sciences (2012)
8. Huang, Z., Chen, Y., Nieplocha, J.: Massive Contingency Analysis with High Performance Computing. In: Proceedings of the IEEE Power Engineering Society General Meeting 2009, Calgary, Canada, July 26-30 (2009)
9. Gorton, I., Huang, Z., Chen, Y., Kalahar, B., Jin, S., Chavarría-Miranda, D., Baxter, D., Feo, J.: A High-Performance Hybrid Computing Approach to Massive Contingency Analysis in the Power Grid. In: Proceedings of the 2009 Fifth IEEE International Conference on E-Science, E-SCIENCE, December 09-11, pp. 277–283. IEEE Computer Society, Washington, DC (2009), doi: http://dx.doi.org/10.1109/e-Science.2009.46
10. Chen, Y., Huang, Z., Chavarría-Miranda, D.: Performance Evaluation of Counter-Based Dynamic Load Balancing Schemes for Massive Contingency Analysis with Different Computing Environments. IEEE PES General Meeting. PNNL-SA-69878, Pacific Northwest National Laboratory, Richland, WA (2009)
11. Bose, A., Poon, K., Emami, R.: Implementation Issues for Hierarchical State Estimators. Final Project Report (September 2010)
12. Abur, A., Expósito, A.G.: Power System State Estimation Theory and Implementation. CRC Press (2004)
13. IEEE 118-bus teset case, http://www.ee.washington.edu/research/pstca/pf118/pg_tca118bus.html (accessed November 2011)
14. Pathak, J., Li, Y., Honavar, V., McCalley, J.: A Service-Oriented Architecture for Electric Power Transmission System Asset Management. In: Georgakopoulos, D., Ritter, N., Benatallah, B., Zirpins, C., Feuerlicht, G., Schoenherr, M., Motahari-Nezhad, H.R. (eds.) ICSOC 2006. LNCS, vol. 4652, pp. 26–37. Springer, Heidelberg (2007)
15. Lalanda, P.: An E-Services Infrastructure for Power Distribution. IEEE Internet Computing 9(3), 52–59 (2005)
16. Marin, C., Lalanda, P., Donsez, D.: A MDE Approach for Power Distribution Service Development. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 552–557. Springer, Heidelberg (2005)
17. Postina, M., Rohjans, S., Steffens, U., Uslar, M.: Views on Service Oriented Architectures in the Context of Smart Grids. In: First IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 25–30. IEEE Computer Society (2010)
18. Rohjans, S., Uslar, M., Juergen Appelrath, H.: OPC UA and CIM: Semantics for the smart grid. In: 2010 IEEE PES Transmission and Distribution Conference and Exposition, pp. 1–8 (2010)

19. Buyya, R., Murshed, M.: GridSim: a Toolkit for the Modeling and Simulation of Distributed Resouce Management and Scheduling for Grid Computing. Concurrency Computat. Pract. Exper. 14, 1175–1220 (2002)
20. Liu, Y., Jiang, W., Jin, S., Rice, M., Chen, Y.: Distributing Power Grid State Estimation on HPC Clusters A System Architecture Prototype. Accepted to the IEEE IPDPS Workshop on Parallel and Distributed Scientific and Engineering Computing, Shanghai, China (May 2012)
21. Jin, S., Chen, Y., Rice, M., Liu, Y., Gorton, I.: A Testbed for Deploying Distributed State Estimation in Power Grid. Accepted to IEEE Power & Energy Society General Meeting, Orlando, FL, USA (May 2012)