An Integrated Framework for Heterogeneous Decision Support Systems

Presented by Helen S. Du, Supervised by Dr. Xiaohua Jia

Computer Science Department

The World Wide Web has become a prime media for sharing and exchanging information from all over the world.

There is a growing need of an open architecture for the integtration and interoperation among multiple data sources and hetreogeneous decision models.

When a Web DSS is designed and implemented based on a closed architecture, it would have little flexibility in terms of add or modifying data and models.

Our open architecture privides more flexibility for Web DSS from two perspectives:

- Allows implementation fo models to be apart from application logic and data management.
- Any changes of source data will not affact application logic and models implementation.

We present a framework that integrates XML and Java™RMI technologies to support the open architecture for Web-based *Decision Support Systems* (DSS), where multiple source data and heterogeneous decision models are independent from each other, and yet openly interchangeable in a distributed network.

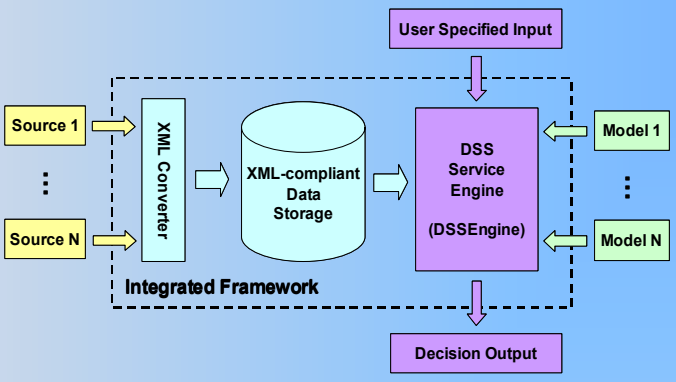


Figure 1. Open Architecture for Web DSS

Frome a system designe point of view, Web-based DSS consists of three mian components:

- **Data Management** – collects and pre-processes a large number of source data to privide historical and summarized information for analytical or decision making spupose.
- **Decision Models Management** – maintains and operates heterogeneous decision models residing locally and/or remotely in the netowrk.
- **User Interface** – provides user-specific requirements for selecting appropriate decision models, and presents to the user the results from the decision making process.

Under this design,

Multiple Source Data are collected from different Web sites, converted and stored in XML-compliant foramt.

Heterogeneous Decision Models are maintained locally as well as remotely in a distriubted fashion.

DSS Service Engine acts as a server for all data and models to interact, and produces final decision results.

Since our deigne handles multiple source data, in the implementation, we define a simple XML *Document Type Definition* (DTD) to specify the source data obtained from various Web sites. This file is referenced by the XML Conveter whenever it needs to translate source data into XML format.

A Sample XML DTD for Stocks:

```
<!ELEMENT SimpleStock(country,exchange,(stock)*)>
<!ELEMENT country CDATA>
<!ELEMENT exchange CDATA>
<!ELEMENT stock(date,price,name?,volume?,turnover?,dividend?)>
    <!ATTLIST stock symbol ID #REQUIRED>
<!ELEMENT date CDATA>
<!ELEMENT price CDATA>
<!ELEMENT name CDATA>
<!ELEMENT volume CDATA>
<!ELEMENT turnover CDATA>
<!ELEMENT dividend CDATA>
```

We define an XML DTD for all models to follow:

```
<!ELEMENT model (name, parameter+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT parameter (name, type)>
    <!ATTLIST parameter ptype (SI|UI|O) #REQUIRED)>
<!ELEMENT type (#PCDATA)>
```

Based on this, external model providers can define their own XML-compliant profiles regardless of what language or system they uses.

```
<!-- Sample XML profile for portfolio
     optimization model -->
<?xml version="1.0" standalone="yes">
<model>
<name>GetOptimalAssetAllocation</name>
<parameter ptype="SI">
<name>MeanReturn</name>
<type>float</type>
<name>Covariance</name>
<type>Covariance_type</type>
</parameter>
<parameter ptype="UI">
<name>MinReturn</name>
<type>float</type>
<name>RiskLevel</name>
<type>RiskLevel_type</type>
</parameter>
<parameter ptype="O">
<name>OptimalAssetAllocation</name>
<type>OptimalAssetAllocation_type</type>
</parameter>
</model>
```

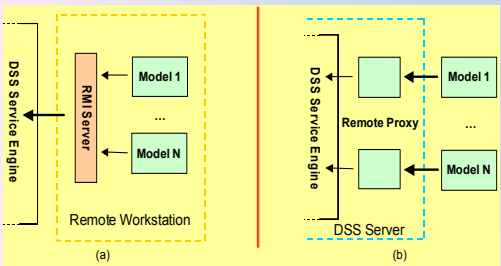


Figure 2. Two Approaches for Distributed Models

In first approach, each model is treated as a single executable model. The second approach splits the models into distributed parts

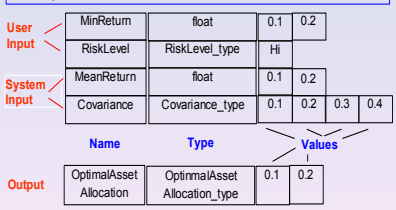
(a) Easy to code, simple to implement;
Java only environment, platform-independent

(b) Platform- and language-independent;
Need to deal with interoperation b/w two parties.

Data preparation for distribution:

All input/opuput parameters are packed as simple vector of DataPacket class before distribution.

```
public Class DataPacket implements Serializable {
    public String   name;
    public String   type;
    public Vector   value_array;
    }
```



Class Defining I/O Message and Sample I/O for Models

```
/*  Model interface that all models must implement  */
interface myModel  {
    public Vector execute(Vector input);
    ...
}

/*  Remote model interface that DSS Service Engine
    implements to run distributed models  */
public interface remoteModel extends Remote  {
    Vector remoteRunModel(String ModelName, Vector input)
    throws RemoteException;
    ...
}

/*  DSS Service Engine invokes remote model execution
    method: remoteModel.remoteRunModel() */
public class DSSEngine  {
    public static void main(String[] args) throws Exception  {
        System.setSecurityManager(new RMISecurityManager());
        String url = "rmi server's site address eg.//hostname:port/";
        remoteModel r = (remoteModel)Naming.lookup(url+"RM");

        DataPacket msg =new DataPacket();
        msg.name=....;      msg.type=....;
        msg.num=...;        msg.value_array=...;

        Vector inMsg=new Vector();      inMsg.add(msg);

        System.out.println("Start to execute remote models");
        Vector outMsg = r.remoteRunModel("Model1",inMsg);
        System.out.println("Get Model1 results");
        ...
    }
}
```

```
/*  Sample implementation of remote model server  */

public class remoteModelSvr extends UnicastRemoteObject
implements remoteModel  {
    public remoteModelSvr() throws RemoteException {}

    public Vector remoteRunModel(String ModelName,
    Vector input) throws RemoteException  {
        return doModel(ModelName, input);
    }

    public Vector doModel(String ModelName, Vector input)
    throws RemoteException  {
        myModel m = null;
        System.out.println("Start the model-->" + ModelName);
        try {
            m =
              (myModel)(Class.forName(ModelName).newInstance());
        } catch(Exception e) {e.printStackTrace();}

        System.out.println("Get results from the model");
        Vector outMsg = m.execute(input);
        ...
        return outMsg;
    }
    ...
    static public void main(String args[]) throws Exception  {
        System.setSecurityManager(new RMISecurityManager());
        remoteModelSvr rms = new remoteModelSvr();
        String url = "rmi server's site address eg.//hostname:port/";
        Naming.rebind(url + "RM", rms);
        System.out.println("Remote Model Server Started");
    }
}
```

Java RMI Codes for Remote Model Execution

Department of Computer Science

香港城市大學 City University of Hong Kong