

# UNDERGRADUATE REPORT

## Web-based Scheduling System

*by Ryan Shows*

*Advisor: Jeffrey W. Herrmann, Edward Yi-tzer Lin*

**UG 2001-7**



*ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.*

*ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.*

**Web site <http://www.isr.umd.edu>**

## **Web-based Scheduling System**

By Ryan Shows  
University of Wisconsin  
Madison, WI

Advisors  
Dr. Jeffrey W. Herrmann  
Dr. Edward Lin

Research Experience for Undergraduates  
Institute for Systems Research  
University of Maryland  
College Park, Maryland

Funded by the  
National Science Foundation

**Aug 3, 2001**

# Table of Contents

1	Introduction .....	4
1.1	Motivation .....	4
1.2	Goal .....	4
2	Issues to Overcome .....	4
2.1	User Friendliness .....	4
2.2	Complexity .....	5
2.3	Flexibility .....	5
2.4	Scalability and Extensibility.....	5
2.5	Manageability .....	5
2.6	Multipart Requests .....	5
2.7	Consistent Input Files.....	5
2.8	XML Parsing .....	6
2.9	Thread Safety .....	6
3	One Implementation: ScheduleThis!.....	6
3.1	User Friendliness .....	6
3.2	Complexity .....	7
3.3	Flexibility .....	8
3.4	Scalability and Extensibility.....	8
3.5	Manageability .....	8
3.6	Multipart Requests .....	9
3.7	Consistent Input Files.....	9
3.8	XML Parsing .....	9
3.9	Thread Safety .....	11
4	Future Additions.....	11
4.1	Fetching Configuration Info.....	11
4.2	Endpoint Support.....	11
4.3	Import/export map links .....	11
4.4	Better Error Messages .....	12
5	Appendix A: ScheduleThis! User Guide.....	13
5.1	Outline .....	13
5.2	ScheduleThis! Overview .....	13
5.3	Tutorial .....	13
5.4	Creating your own input files from Microsoft Project.....	21
5.4.1	Installing the import/export maps .....	22
5.4.2	Exporting Project data.....	22
5.4.3	Importing schedule into project.....	22
5.4.4	Limitations on input files .....	22
6	Appendix B: Web Service Development Guide .....	24
6.1	Outline .....	24
6.2	Web Services Overview .....	24
6.3	Installing Web Services support to your web server.....	25
6.4	Tutorial: Building the sample Scheduler services.....	25
6.4.1	Setting up for the Scheduler services .....	25
6.4.2	Creating the solution and project .....	26

6.4.3	Importing the source code .....	26
6.4.4	Setting up the build class path.....	27
6.4.5	Generating the XML parsing classes (Optional).....	28
6.4.6	Creating and deploying the Scheduler Web services .....	28
6.5	Building your own Scheduler services .....	31
6.5.1	The scheduler-config.xml file .....	31
6.5.2	The interface.....	32
6.5.3	public String execute(String formdata) .....	32
6.5.4	public String getPerformance() .....	33
7	Appendix C: ScheduleThis! Installation Guide .....	34
7.1	Outline .....	34
7.2	Installing ScheduleThis!.....	34
7.3	Configuring ScheduleThis! .....	35
7.3.1	The scheduler-config.xml file .....	35
7.3.2	Proxy classes .....	36

# **1 Introduction**

## **1.1 Motivation**

Machine scheduling has long been a focus of engineering research. To stay competitive, industry recognized early that by streamlining their manufacturing processes, and automating scheduling, they could reduce costs dramatically. Although historically a mechanical engineering focus, today this kind of research is highly interdisciplinary due to the powerful role computers play. Most of the present day focus is on the general scheduling problem, opposed to the specific machine environments and constraints studied in the past. Another aspect of the scheduling problem, that may often be overlooked, is how to formulate the problem. That is the focus of this project. Most scheduling algorithms require very structured input files that are either created by hand or by a computer program. This is both tedious and difficult if unfamiliar with programming concepts.

## **1.2 Goal**

The goal of this project is to design a system to make advanced scheduling services available to a user with little or no programming experience, such as a project manager. Important aspects for the design of such a system include, but are not limited to, flexibility, scalability, and user friendliness. All these design aspects can be met with a web-based implementation using XML and Java. It is assumed a person using this system would be familiar with project management software such as Microsoft Project. It, therefore, makes sense to leverage this existing software to help formulate the scheduling problems.

# **2 Issues to Overcome**

There are many issues that will be encountered with any web-based approach to scheduling. All these issues must be overcome to create a feasible, manageable system. This section describes the common issues one will face when trying to implement an online scheduling system.

## **2.1 User Friendliness**

Ease of use is always a concern. Here, it is a primary goal. The system should be easy to navigate, yet still provide a rich interface.

## ***2.2 Complexity***

Building even a simple scheduling system becomes complex very quickly. This is especially evident when using many JSPs combined with servlets. Where should most of the code be placed, the JSPs or the servlets? How does one keep track of program flow? How does one easily collect data from the user and get it to where it is needed? These are just a few questions that come to mind.

## ***2.3 Flexibility***

Flexibility is important in any successful software product. Even more special attention must be paid with web-based applications. It is easy to get carried away, start hard-coding URLs and file accesses, then when the system must be put on a new server, or under a new context, it breaks.

## ***2.4 Scalability and Extensibility***

How well will the system function when the number of users doubles, or triples? Can more features be added without rebuilding the code base? A successful system must always be designed to grow and meet new unforeseen challenges.

## ***2.5 Manageability***

Code separated among many files can get out of hand fast, especially when trying to trace program flow across multiple files. Managing hierarchy and package structure also becomes an issue of importance. Reserving the ability to maintain and manage the code is imperative. This is impossible to do by using a simple text editor. A system of this magnitude necessitates the use of an Integrated Development Environment (IDE).

## ***2.6 Multipart Requests***

It is unavoidable that a system of this kind will have to handle multipart requests, since with any scheduling program there must be input files to define the problem, and to upload files to a web server via HTTP a multipart request must be used. Handling multipart requests is both tedious and error prone. The system must efficiently extract the files from the request and get them where they need to be processed. The same must be done with the other data in the request.

## ***2.7 Consistent Input Files***

It is assumed the user will export data from their project software to a file to be supplied to the system. The exported file could take many forms. The input files to the system must adhere to a certain structure to ensure that they are parsed and processed correctly. How can it be ensured that the input files conform to the proper structure, yet still make it easy for the user to export the data?

## **2.8 XML Parsing**

The system will be built on XML and Java technologies. It is, therefore, inevitable that the system will have to parse XML information. There are many XML tools and parsers available for Java. Which one is right for the job?

## **2.9 Thread Safety**

This is an issue unique to web applications. Servlet containers do not create a new instance of a requested servlet for every client. Instead, each client uses the same instance, thereby reducing the memory strain of the server. At the same time this introduces potential racing conditions. If two clients are accessing the same servlet at the same time, and modifying shared resources, an indeterminate state results. It must be ensured that only one client is allowed to access critical parts of the servlet at one time. The critical parts are sections of code when shared resources are being modified, such as instance variables.

# **3 One Implementation: ScheduleThis!**

ScheduleThis! is a system that has overcome the issues in the previous section to accomplish the project goals. What follows is an explanation of how ScheduleThis! overcame each of the issues.

## **3.1 User Friendliness**

The interface of ScheduleThis! is a series of forms, which collect information and help formulate the scheduling problem. First, the user chooses a scheduling problem to solve. Second, the user chooses an algorithm, the method by which to solve it. Finally, they enter additional information required by that algorithm and upload any required files, such as tasks and resources. This is a natural process of gathering the information. The “drill down” structure makes it easy to understand how the system is using the information it is gathering. There is a tutorial available with the online documentation that gives step-by-step instructions on using the system. See **Appendix A** for the tutorial.

### 3.2 Complexity

Complexity was overcome with the help of the Apache Struts Framework. Struts is an implementation of the Model-View-Controller (MVC 2) framework. With Struts there is only a single servlet, called the controller. All client requests first go to the controller. The controller then forwards the request to a custom Action class based on the HTML form action, and the configuration data stored in `struts-config.xml`. The Action class performs business logic functions. To perform its functions, the Action class may use JavaBeans or access databases. These components are the Model of the framework. Finally, when the Action class is done it forwards to a JSP that will be displayed as the response. The JSPs are the View of the framework. More information on Apache Struts can be found at <http://www.apache.org/struts>. See Figure 1 for see a visual representation of the architecture.

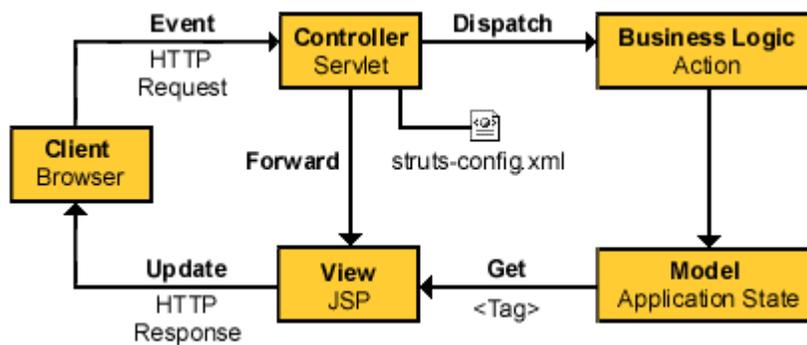


Figure 1 The Struts MVC Framework

Struts answers the question from before, where should the code go, JSPs or servlets? The answer is neither. Most of the code is put into the Action classes, which are invoked by the controller servlet when the user submits a form. Each form has an Action class to process it, except the last form (`upload.jsp`) because it uses a multipart request, which is not supported by Struts. Instead, `upload.jsp` POSTS directly to a servlet. Each form also has an `ActionForm` class associated with it, another feature of Struts. An `ActionForm` is a `JavaBean` with properties representing the form fields. When the form is submitted, the `ActionForm` class is automatically populated with the form data and passed to the Action class, making it easy to get to the form data. The `ActionForm` is also helpful for re-populating form fields when a validation error occurs, and the user must be returned to the form to correct the mistake. Any dynamic data the Action class would like to return is saved as an attribute to an implicit object such as `request`, or `session`. Finally, the Action class forwards a JSP to the client. The JSP displays dynamic content by accessing the stored attributes.

Using Struts reduces complexity for several reasons. First, the controller servlet makes managing the program flow much easier. The Action classes that are invoked, and the JSPs returned by them can easily be changed by modifying the `struts-config.xml` file. Second, it is much cleaner to create dynamic output. Writing HTML from servlets requires outputting using cryptic escape sequences. This is tedious, especially since most

HTML is static. It is natural to return a JSP that already consists of a static HTML template. The JSP can display any dynamic output using scriptlets.

### **3.3 Flexibility**

ScheduleThis! has the ability to be installed to any web server with Java servlet and JSP support, and under any context-root. The only change that must be made is the “context” initial parameter in the web.xml file under the ScheduleConfigServlet to reflect the context to which ScheduleThis! will be installed. The default is /ScheduleThis. When the servlet container is first started, ScheduleConfigServlet is invoked. It reads the “context” initial parameter and sets the “context” attribute of the ServletContext to that value. Every hyperlink is relative to the value of the “context” attribute. ScheduleThis! also needs to read and write files. In order to make file accesses flexible, the “real” path to the base of the current web context is found. “Real” path means the absolute path on the server machine. This is done using `ServletContext.getRealPath("/")`. Once this base path is obtained, all files can be accessed safely relative to it. Now, if the context, or server is switched, ScheduleThis! will use a new base path, and all is well.

### **3.4 Scalability and Extensibility**

The scheduling algorithms create a potential scalability problem. Some advanced algorithms are very processor intensive. If several clients are running these algorithms simultaneously, it could cripple the system, making it impossible for other clients to process a schedule. This was overcome by implementing the scheduling algorithms as web services. See **Appendix B** for an overview of web services. As web services each algorithm can run on a separate dedicated server, relieving ScheduleThis! to handle client requests. The advantages of web services do not end there, however. Anyone can develop scheduling services and easily integrate them with ScheduleThis! New scheduling web services are added by modifying the scheduler-config.xml file. See **Appendix C** for a summary of the scheduler-config.xml file. All that is needed from the web service provider is a Proxy class that will be used to communicate with the service.

The only other place scalability had to be considered was saving the completed schedules produced by the web services. These files must have a unique name, thus the file names are time stamped with the millisecond that the input file(s) was submitted for processing. If two clients submit files within the same millisecond, the first schedule to finish will be overwritten with the schedule produced for the second client. Since this scenario is highly unlikely, it is ignored.

### **3.5 Manageability**

The IBM XML and Web Services DE was used to create and build all ScheduleThis! and web services code. It is available from IBM AlphaWorks at <http://www.alphaworks.ibm.com/tech/wsde>. This tool was a natural choice as it supports web service development. The XML tools were also helpful in creating DTD files and parsing XML. See **Appendix B** for a tutorial on how to use the IBM XML and Web Services DE to create and deploy web services.

### **3.6 Multipart Requests**

As mentioned earlier a web-based scheduling application must be prepared to handle multi-part requests. ScheduleThis! uses the com.oreilly.servlet package available at <http://www.servlets.com/cos/index.html>. This package contains a class called MultipartRequest, which makes extracting and saving files, uploaded with a multipart request, extremely easy. Just instantiate the class like so:

```
MultipartRequest multi = new MultipartRequest(request, save-dir)
```

where request is the regular HttpServletRequest, and *save-dir* is the directory under the web server to save the uploaded files. The files in the request will automatically be saved to that directory with the same names as on the client machine. The MultipartRequest class also makes extracting parameters from a multipart request just as easy as with HttpServletRequest.

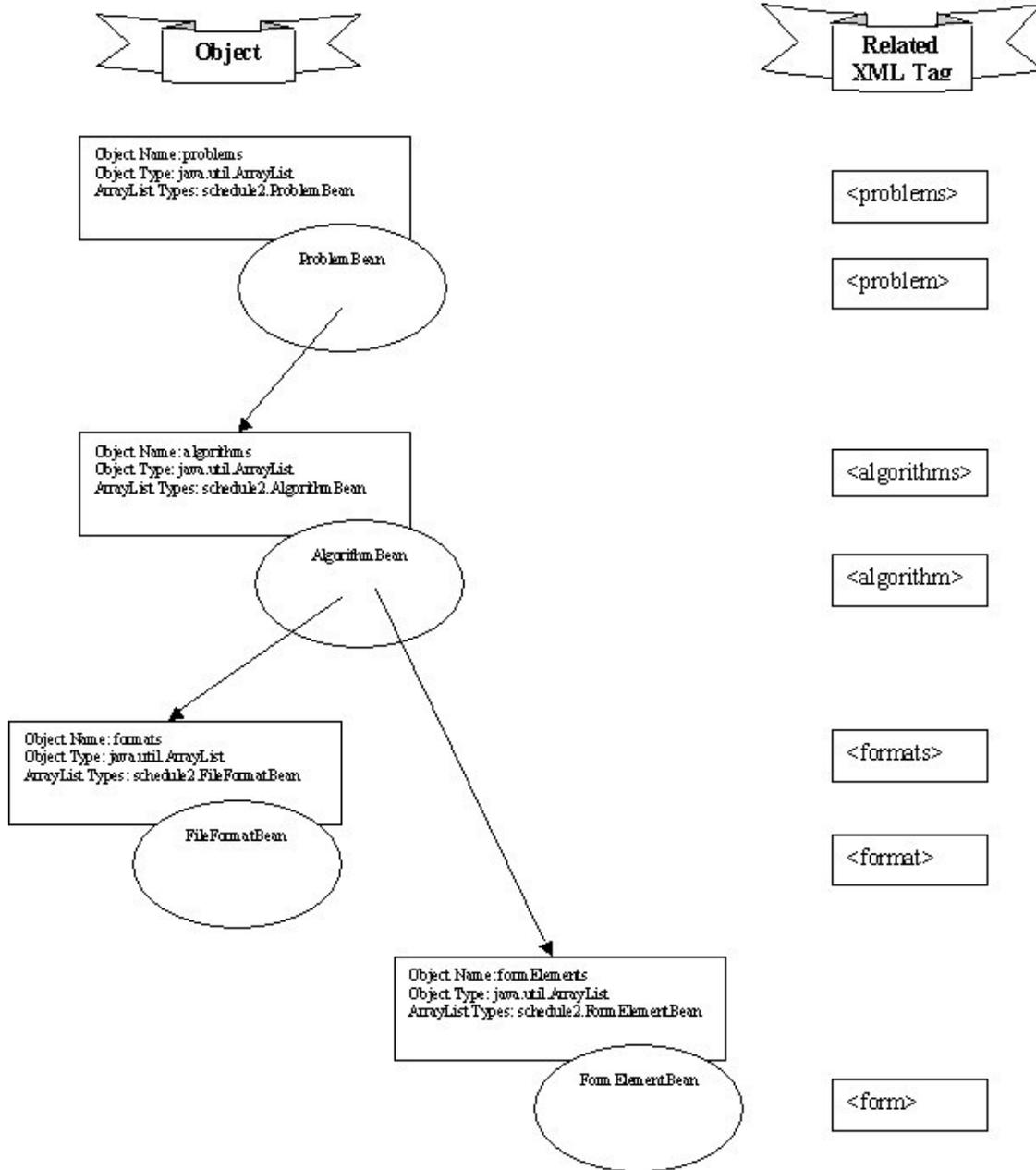
The only file that sends a multipart request in ScheduleThis! is upload.jsp because upload.jsp contains the form that has the input files. All other forms use Struts.

### **3.7 Consistent Input Files**

ScheduleThis! contains a format field in upload.jsp to indicate to the scheduling service which program the input file(s) are coming from. This may not be enough, however, because many times there is more than one way to export data from the same program. Microsoft Project allows the creation of import/export maps. These maps define what columns to export, in what order, and what character to use to delimit the fields. See **Appendix A** for how to use these maps.

### **3.8 XML Parsing**

There were two places where XML parsing was used. First, ScheduleConfigServlet parses the configuration file scheduler-config.xml. This parsing is performed using the XML Digester class available as part of Struts. It is easy to set-up events with the Digester class. The Digester can create a JavaBean object when it sees a start element tag, and automatically fill in the bean's properties with the element's attributes. ScheduleConfigServlet uses the Digester to create an object tree out of the configuration file. This object tree is depicted in the following image:



**Figure 2 Object tree representing configuration information**

The second place where XML parsing takes place is in the scheduling services. These services must parse the XML String to obtain the data and files uploaded by the user. Here, Java parsing classes were generated using the IBM XML and Web Services DE. See **Appendix B** for a tutorial on generating Java parsing classes from a DTD file.

### **3.9 Thread Safety**

There is no shared state in ScheduleThis!, and therefore no racing conditions to consider. The only class that has instance variables is ExecuteThread. Since a new instance of ExecuteThread is created for every schedule submitted, there are no problems.

Each of the scheduling web services must consider racing conditions more closely because they have shared state. The methods of the Proxy classes that ScheduleThis! uses to invoke web services are synchronized, so only one call may be processed at a time, therefore ScheduleThis! cannot break the web service alone. The methods of the web service itself, however, are NOT synchronized creating a potential problem. For instance, if more than one client is using the web service, the clients could access the service at the same time and a racing condition would result. This issue is something that must be taken into account by the service provider if they plan on the service being used by multiple clients. The sample scheduling services provided with the ScheduleThis! distribution are not thread safe, and therefore may only be used with one instance of ScheduleThis!

## **4 Future Additions**

Due to time constraints on the project, some planned features were not implemented. Below is a description of each of the main features that will hopefully be implemented in the future.

### **4.1 Fetching Configuration Info.**

Currently, the files and other data required by the web service must be added to the scheduler-config.xml file. It would be relatively straightforward to instead fetch this information from the web service.

### **4.2 Endpoint Support**

If the web services are deployed to a new server, the Proxy classes must be re-generated so ScheduleThis! knows where to find them. An attribute to the <algorithm> tag in the scheduler-config.xml file could be added that defines the endpoint of the web service. Before any methods of the web service are invoked, setEndpoint() would be called and passed the value of the attribute. Now, when web services change location, only a small change to the scheduler-config.xml file is needed.

### **4.3 Import/export map links**

Right now ScheduleThis! assumes the user knows which map to use for each algorithm. It would be nice to have a way for the web service to provide a link to a map, or project template file containing the map, needed to generate the input files.

#### **4.4 Better Error Messages**

It would also be helpful if the web service could convey more detailed error messages when there is a problem generating a schedule. One possible solution is throwing a custom exception.

## 5 Appendix A: ScheduleThis! User Guide

This guide is meant to be viewed by a web browser. Depending on how you are viewing it you may not be able to follow the links that are provided. It is available with the ScheduleThis! online documentation.

Welcome to the ScheduleThis! portal family. You have made an excellent choice for your machine scheduling needs. This guide offers an overview and tutorial on how to use ScheduleThis! with the sample scheduling services provided with the distribution.

---

### 5.1 Outline

1. [Schedule This Overview](#)
  2. [Tutorial](#)
  3. [Creating your own input files from Microsoft Project](#)
    - [Installing import/export maps](#)
    - [Export Project data](#)
    - [Import schedule into Project](#)
    - [Limitation on input files](#)
- 

### 5.2 ScheduleThis! Overview

ScheduleThis! is designed to be used with project management software such as Microsoft project. First, the project data is exported to a text file, or multiple text files. These files are uploaded to ScheduleThis!, which schedules the tasks. Then, the schedule can be downloaded and imported back into the project software to reflect the schedule. This procedure is outlined in the tutorial using a sample input file.

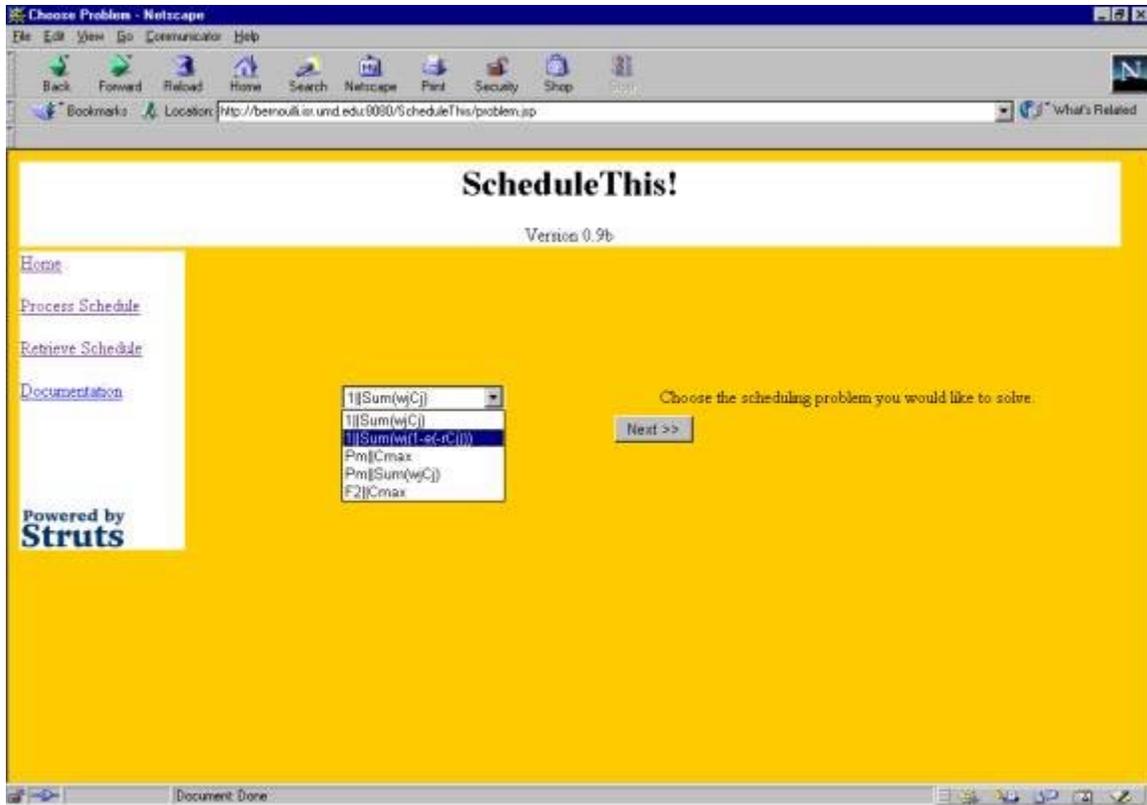
---

### 5.3 Tutorial

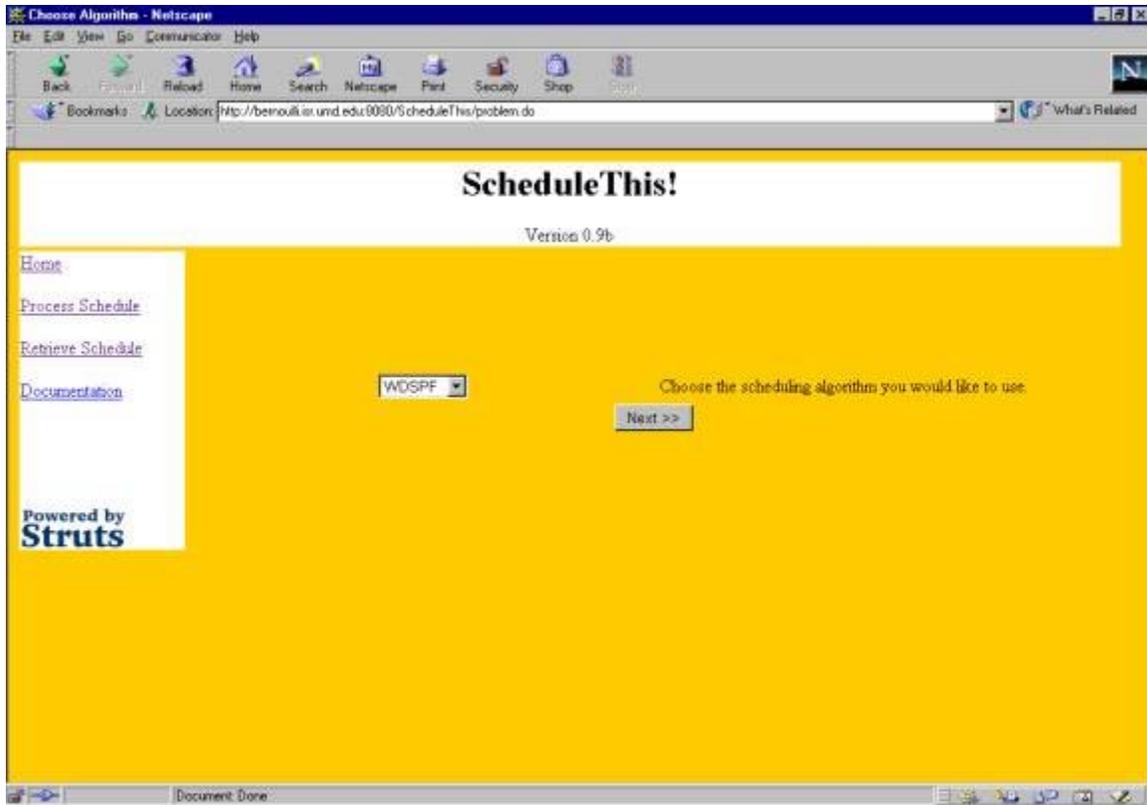
This tutorial gives step-by-step instructions on processing a schedule from a sample input file. In order to create a schedule from your own custom input files, please read [Creating your own inputfiles from Microsoft Project](#).

First, type in the URL where ScheduleThis! is located. You should then see the home page. In this example, ScheduleThis! is installed on bernoulli.isr.umd.edu under the web context /ScheduleThis.

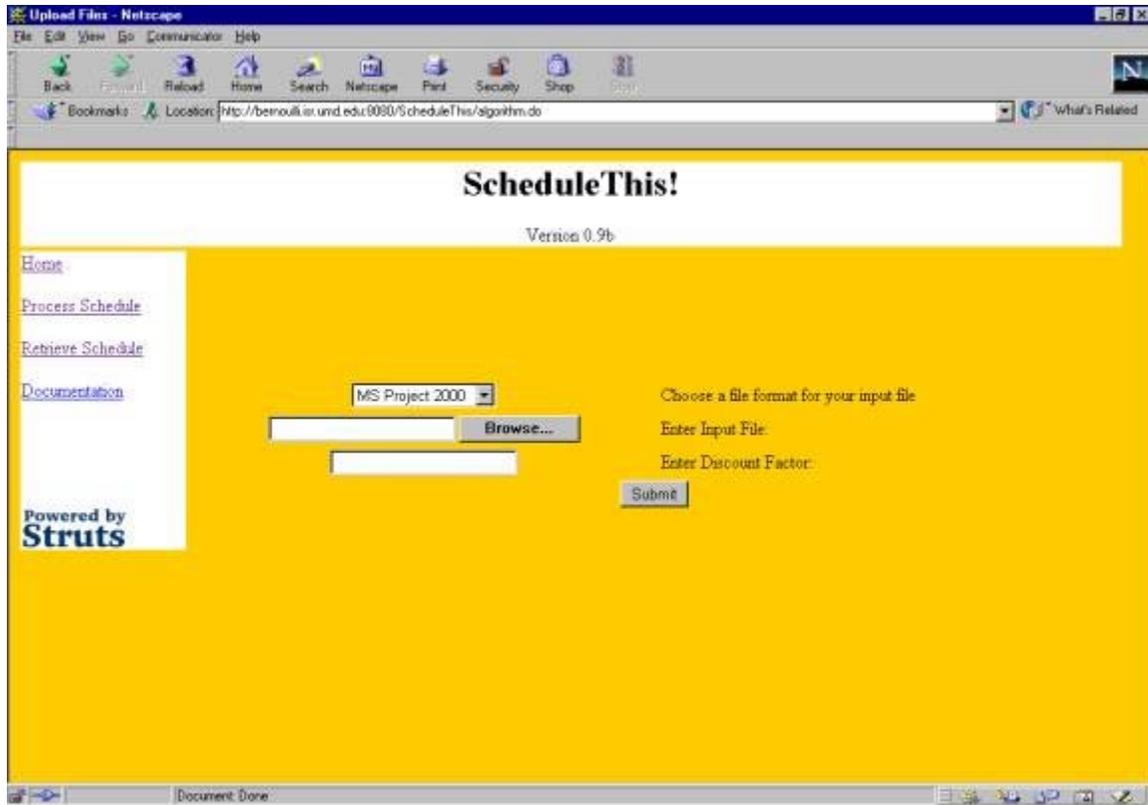




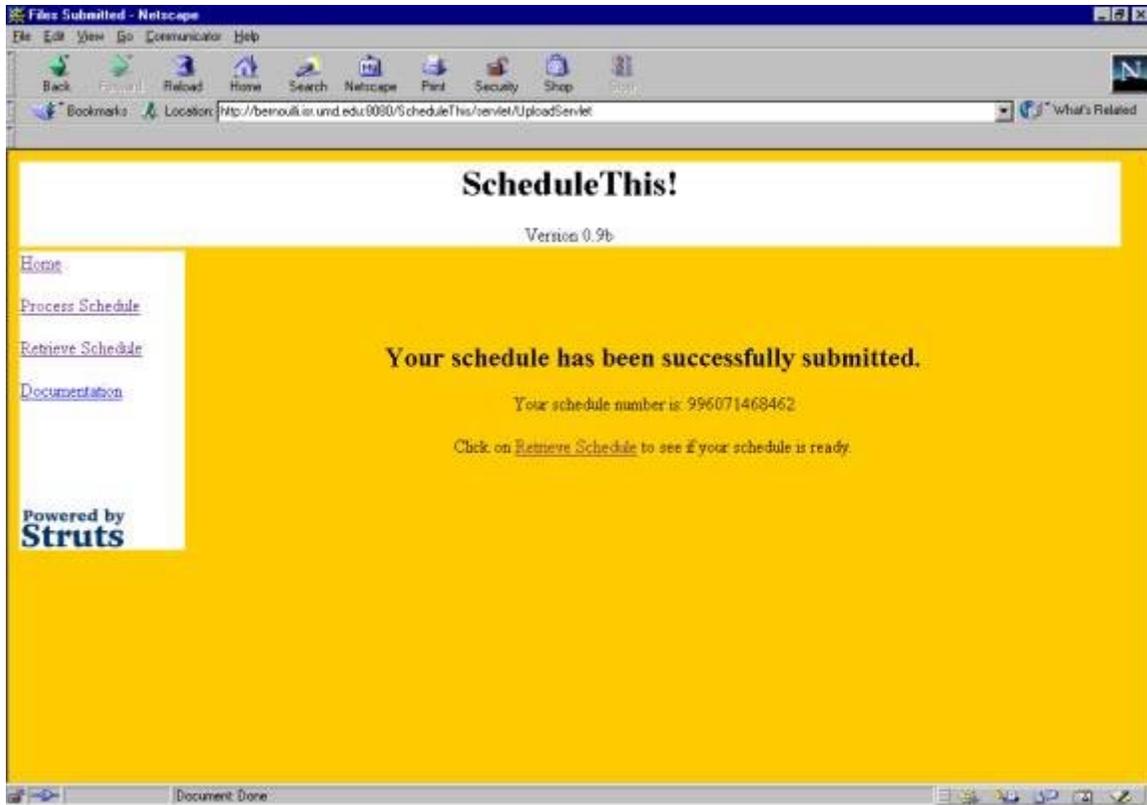
Here you choose a scheduling problem you want to solve. The machine environment, constraints, and objective are all included with each problem. The exact problem you want to solve may not be listed, but try to choose the problem closest to your actual machine environment and desired objective. More problems can be added by modifying the scheduler-config.xml file. In this example choose the problem  $1||\text{Sum}(w_j(1-e(-rC_j)))$ . Click **Next**.



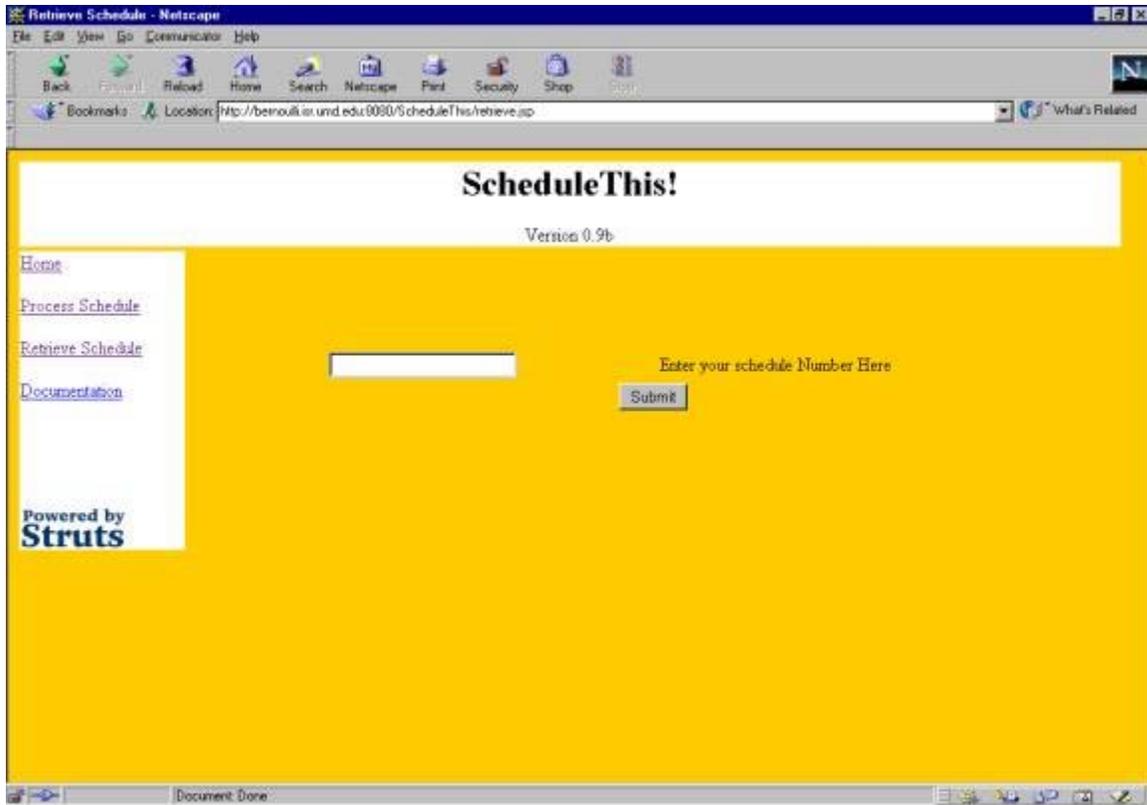
Now you chose the specific algorithm that you would like to use to create the schedule. A single problem may have more than one algorithm. Trying the different algorithms is a good way to compare different heuristics. In this case, only the WDSPT algorithm is available, and it is optimal for  $1||\text{Sum}(w_j(1-e^{-rC_j}))$ . Choose this algorithm by clicking **Next**.



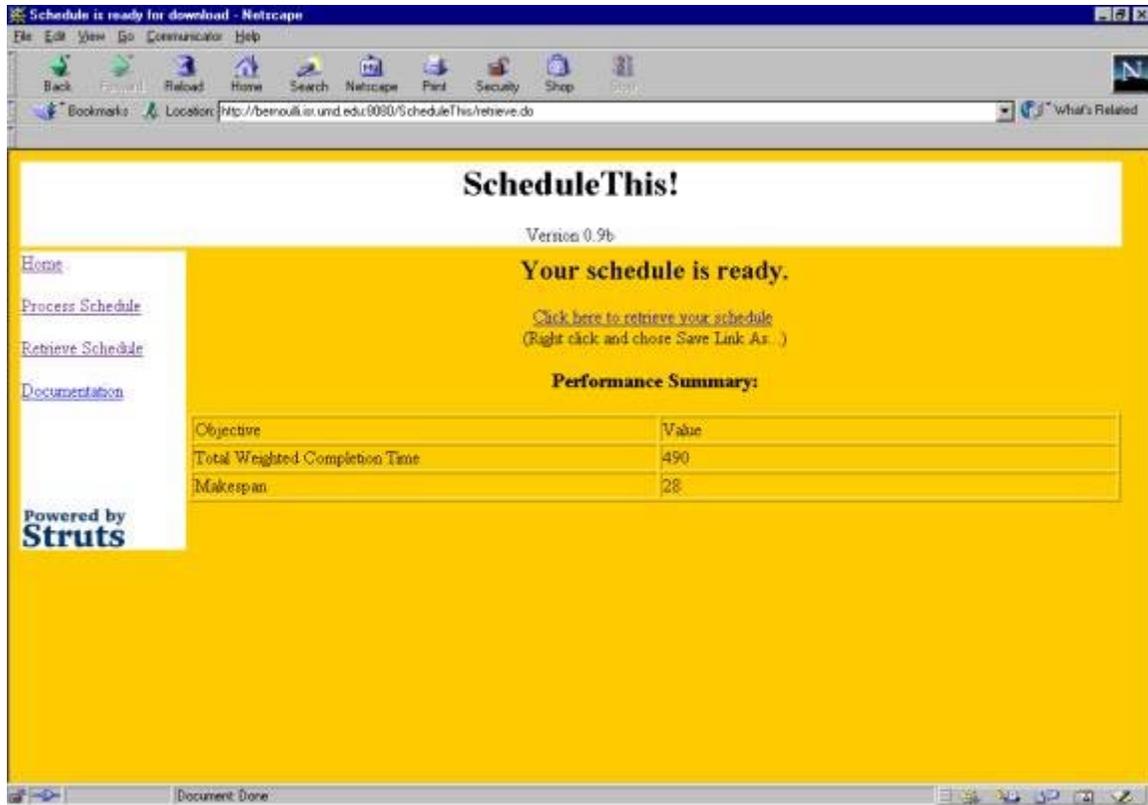
Now you must specify the input file(s) to be processed, and any additional information specific to the chosen algorithm. This page will not always look the same. The form that is displayed depends on the algorithm you chose on the previous page. The file format pull down box, however, is always present. The algorithm must support at least one file format. The file formats supported by different algorithms may be different. In this example the problem is  $1 || \text{Sum}(w_j(1 - e^{-rC_j}))$ . The only file format available is Microsoft Project. Since this is a single machine environment only one input file is required which contains a task list. Click [here](#) to download the sample input file. Click **Browse** and select the input file you downloaded. The WDSPT algorithm requires a discount factor (the "r" in the objective). Enter 0.1 in the discount factor field. Click **Submit**. You should now see the following page.



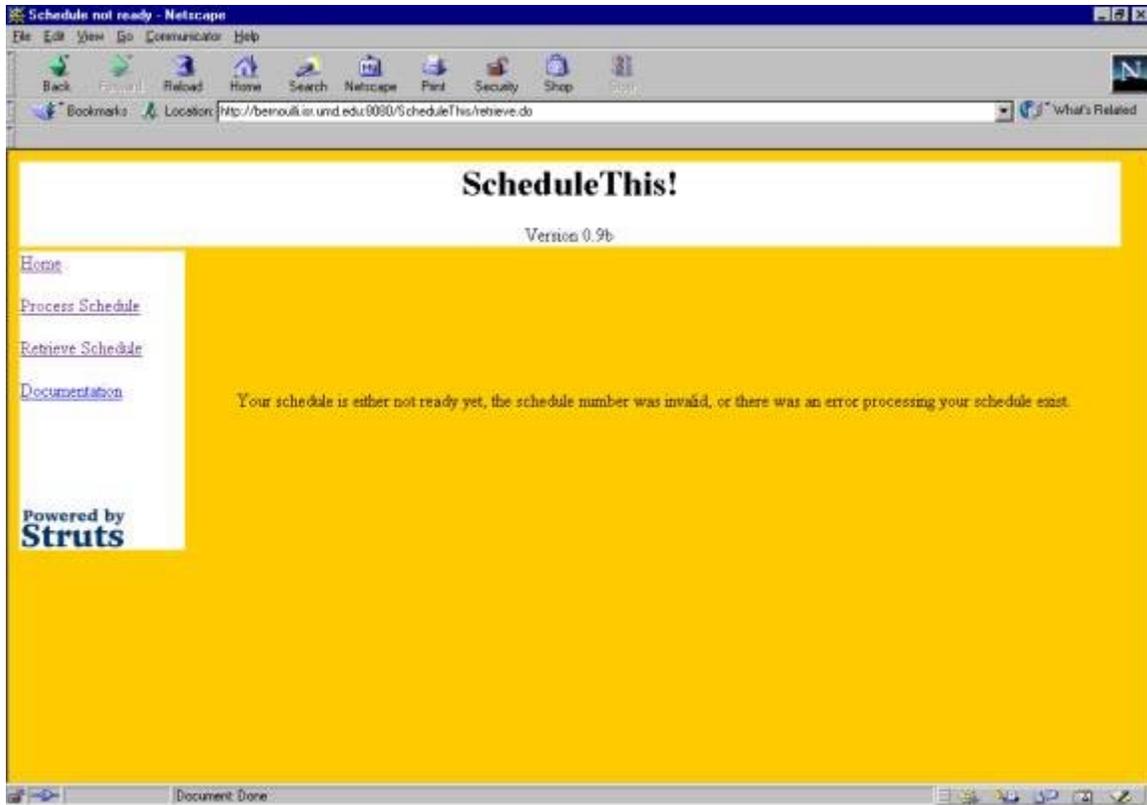
This means your files have been successfully uploaded to the ScheduleThis! server. This does not necessarily mean your schedule is ready yet. Some scheduling algorithms may run for a long time, so instead of waiting until the algorithm is done, ScheduleThis! runs the scheduler in a separate process and returns you to this page. This way you can start processing other schedules while the last one is running. To check if the schedule is ready, click on **Retrieve Schedule** on either the navigation bar to the left, or the link provided below the schedule number. Remember your schedule number because it will be needed to retrieve your schedule.



Type in your schedule number into the text field. Click **Submit**. If your schedule is ready you should see the following page.



You can now download the completed schedule from the link provided. The performance information differs from algorithm to algorithm. If the schedule is not ready you would see...



This can mean a few things. First, the schedule is still processing, and you should come back and check again later. Second, you may have entered the schedule number incorrectly. Check the number and try again. If you have lost the number re-submit your files. Lastly, there may have been an error processing your input files. This is mostly likely because the input format was unrecognized by the algorithm. If you are using Microsoft project make sure you have exported the file with the correct import/export map. More specific error messages will be provided in the future.

---

## ***5.4 Creating your own input files from Microsoft Project***

Eventually, you are going to want to create you own list of tasks/resources and have ScheduleThis! produce a schedule for you. As explained in the overview, input files are expected to come from project management software. Currently, the only project software supported by the scheduling algorithms is Microsoft Project. You need to export data from Microsoft Project using a map. The map defines what fields from project will be exported and in what order. This is important because the algorithm expects the input file to have a certain structure in order to parse it correctly. Below is a table listing the maps required for each of the sample algorithms included with ScheduleThis!.

Algorithm	Map
WSPT	Schedule Task Map Simple
WDSPT	Schedule Task Map Simple
LPT	Schedule Task Map for task file and Schedule Resource Map for resource file
WSPT for Parallel Machines	Schedule Task Map for task file and Schedule Resource Map for resource file
Johnson	Schedule Johnson Map for task file and Schedule Resource Map for resource file

#### 5.4.1 Installing the import/export maps

All the above maps are available in the [ScheduleThisTemplate](#) file. To install the maps, download the template file. Open the template file in Microsoft Project. Click on **Tools->Organizer** and click on the maps tab. Copy all the maps from the ScheduleThisTemplate to GLOBAL.MPT. Now the maps should be available when you want to export your project data.

#### 5.4.2 Exporting Project data

When you are ready to export your data click **File->Save As...** In the **Save as type** box chose Text(Tab Delimited). Enter a file name and click the **Save** button. The **Export mapping** dialog box will appear. Choose the appropriate map for the algorithm you are going to use by checking the table above. Click the **Save** button.

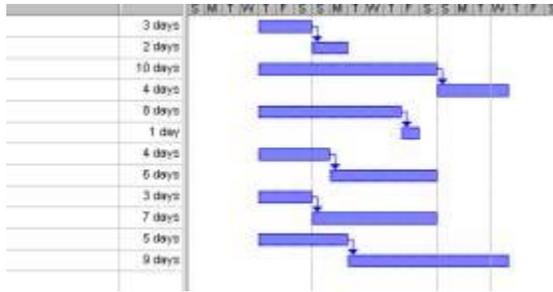
#### 5.4.3 Importing schedule into project

Download the schedule from ScheduleThis! as described in the tutorial. Use the same map that you used to export the data to import the new schedule back into project. The map will merge the data from the import file into the current project file.

#### 5.4.4 Limitations on input files

The sample scheduling web services have some restrictions on the input files. In order for your input files to be processed correctly they must adhere to these limitations. These limitations are due to how the scheduling services were implemented.

- All tasks must have the same start date except when using the Johnson algorithm.
- With the Johnson algorithm the tasks should be grouped into pairs, with the first task preceding the second. Each pair represents a job. The first task in the job represents the required processing time on the first machine, and the second task represents the processing time required on the second machine. The following image should help clarify this structure.



- The Microsoft Project factory calendar must be changed to include weekends as work days. This can be seen in the above image.
- Task durations must be in days.
- For multiple machine environments you must define at least one resource in the resource sheet. The resources listed in the resource sheet are considered the available machines for scheduling.

## 6 Appendix B: Web Service Development Guide

This guide is meant for people who are trying to develop machine scheduling web services that can be used with ScheduleThis! This document has step-by-step instructions on how to build and deploy the sample scheduling web services included with the ScheduleThis! distribution, using the IBM XML and Web Services DE. Also included are instructions on how to install web services support to your web server.

---

### 6.1 Outline

[Web Services Overview](#)

[Installing Web Services support to your web server](#)

[Tutorial: Building the sample Scheduler services](#)

[Setting up for the Scheduler services](#)

[Creating the solution and project](#)

[Importing the source code](#)

[Setting up the build classpath](#)

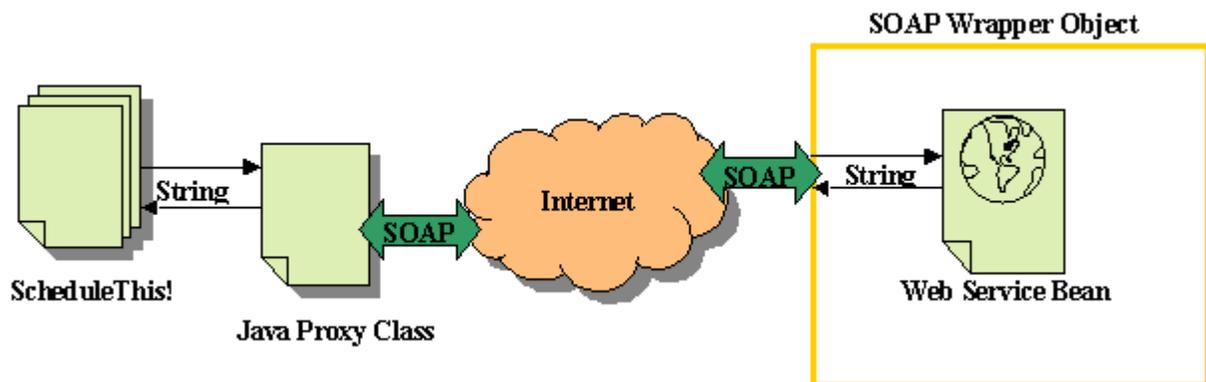
[Creating and deploying the Scheduler Web services](#)

[Building your own Schedule services](#)

[The scheduler-config.xml file](#)

[The interface](#)

### 6.2 Web Services Overview



Very quickly what are web services? Web services are basically java bean classes you create that are wrapped in a SOAP (Simple Object Access Protocol) object. Web services are accessed by clients through the use of a Proxy class. The Proxy class can be generated when the web service is created. Once the client has the proxy class it can execute methods on the remote web service. Web services basically take a normal class that may access business logic or databases using proprietary technologies such as Java RMI, wrap it in a SOAP (Simple Object Access Protocol) object, which makes that class accessible over the Internet via standard SOAP messages. SOAP messages are universal

because they are XML. The java proxy class generates these SOAP messages when the client calls a method, and they are sent over the Internet to the web service, which causes the corresponding method on the remote web service to be invoked. The client application doesn't need to know anything about SOAP, they just execute a method on the proxy class, which reflects the interface of the remote class they want to run, and all the communication details are taken care of for them.

---

### **6.3 Installing Web Services support to your web server**

Before ScheduleThis! can invoke any web services those services must be up and running on a server somewhere. To do this you must first install Apache SOAP to your web server. Instructions for installing SOAP are available from <http://xml.apache.org/soap/docs/index.html>. Click on the **Installation** button on the left. I suggest using Tomcat 3.2.2 because most of the documentation refers to it. Follow the server-side instructions. Once SOAP is installed you will be able to deploy your web services to your web server using a web-based deployment tool called the SOAP admin tool.

---

### **6.4 Tutorial: Building the sample Scheduler services**

This tutorial explains how to get the sample web services running included with ScheduleThis!. To test them you will need to run ScheduleThis!. Read the **ScheduleThis! Installation Guide** to learn how to install ScheduleThis! Included with the distribution are 5 sample scheduling services that were developed to test ScheduleThis! These services implement the `schedule2.Scheduler` interface. This interface is included with the distribution. It is a good idea to implement this interface because it forces your classes to implement the `execute()` and `getPerformance()` methods that are called on by ScheduleThis! This tutorial explains how to build and deploy these sample web services using the IBM XML and Web Services DE available from <http://www.alphaworks.ibm.com/tech/wsde>.

\*\* To get the services working quickly on localhost, you may skip much of the tutorial. Since the class files are already provided with the distribution in `/Schedulers/servlets` you may jump ahead to **Creating and deploying the Scheduler Web services**, just copy the class files to `soap-home/WEB-INF/classes` on Tomcat, and following the deploy instructions. This only works for installing the services to localhost. In order to get the services working on another server you must re-generate the Proxy classes. To use services deployed to localhost, ScheduleThis! will also have to be running on the localhost.

#### **6.4.1 Setting up for the Scheduler services**

Before you can create and deploy the sample services, you need to do the following:

- Create the solution and project
- Import the source code
- Set up the build class path
- Generate XML parsing classes (Optional)

- Deploy the services

The following sections provide details about each of these tasks.

## 6.4.2 Creating the solution and project

Before you can develop the sample services you need to create a solution, and a project that will contain the code for the services.

To create the solution, do the following:

1. From the Development Environment desktop, click **Perspective > Switch to > Java Perspective**.
2. From the desktop, click **File > New > Solution**. The New Solution wizard opens.
3. In the **Solution name** entry field, type **ScheduleThis!**, and click **Finish**.

To create the project, do the following:

1. From the desktop, click **File > New > Web Project**. The Web Project wizard opens.
2. In the **Project name** entry field, type **Schedulers**, and click **Finish**.

## 6.4.3 Importing the source code

We provide the source code for the Scheduler services in the distribution. You need to import the source code into the Schedulers project by doing the following:

1. In the Packages pane on the desktop, select the **Schedulers** project.
2. From the desktop, click **File > Import**. The Import wizard opens.
3. In the wizard, select **File system** from the **Select import source** list, then click **Next**.
4. In the **Directory** text field, specify the following location of the source code. Use the **Browse** button if necessary:

```
install-dir\Schedulers
```

Where *install-dir* is the directory where you installed the ScheduleThis! distribution.

5. Select the **All types** radio button.
6. In the **Container** text field, ensure the following is specified:  
`/ScheduleThis!/Schedulers`. If the correct container is not specified, use the **Browse** button to specify it.

7. Click **Finish**. The source code is imported into the Schedulers project.

#### 6.4.4 Setting up the build class path

The Scheduler services need to know the location of some supporting classes contained in external .jar and .zip files. You specify the location of these files by setting up the build class path. To set up the class path, do the following:

1. Click the **Navigator** tab, and right-click the **Schedulers** project. From the pop-up menu, click **Properties**.
2. In the Properties dialog box, click **Add External Jar**.
3. In the Open dialog box, specify the following .jar file, then click **Open**:

itp\plugins\com.ibm.etools.b2b.wsd\runtime\ivjdab.jar

4. Repeat the previous two steps to add the following files to the class path:

itp\plugins\xmlschemamodel\jars\xsdbeans.jar

itp\plugins\sqltoxml\runtime\sqltoxml.jar

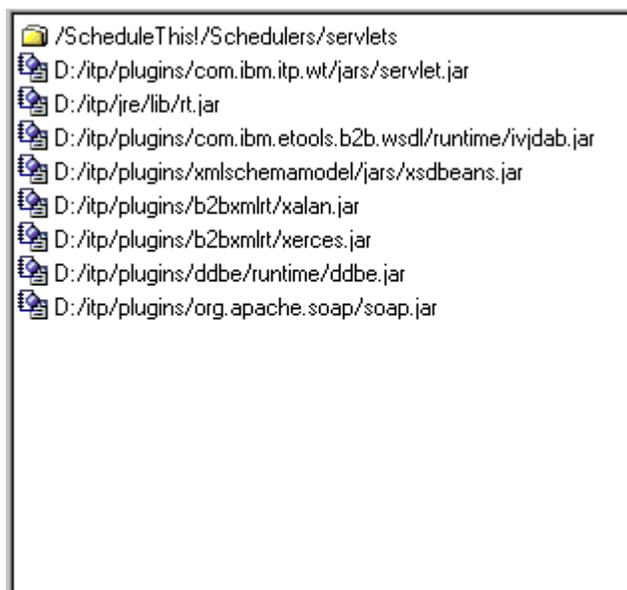
itp\plugins\b2bxmlrt\xalan.jar

itp\plugins\b2bxmlrt\xerces.jar

itp\plugins\ddbe\runtime\ddbe.jar

itp\plugins\org.apache.soap\soap.jar

When you're done, the class path should look similar to this:



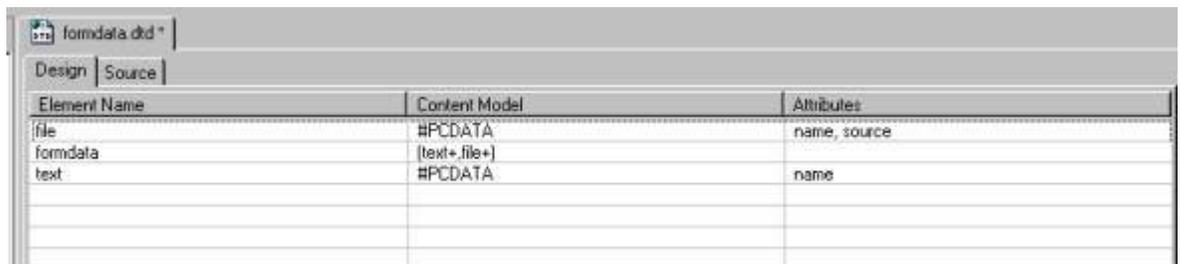
5. Click **OK** to close the Properties dialog box.
6. If any error messages appear in the All Tasks window of the desktop, click **Workbench > Rebuild All**, then **OK**.

### 6.4.5 Generating the XML parsing classes (Optional)

This section is optional because the parsing classes are already provided, and by generating them again you will just be overwriting the ones already there. You can complete this section if you wish to learn about how to generate XML parsing classes from a DTD file.

To generate the parsing classes, do the following:

1. In the **Navigator** pane on the desktop, expand the **ScheduleThis!**, **Schedulers**, and **servlets** folders.
2. Double-click on **formdata.dtd**. This launches the DTD Editor, and the contents of the formdata.dtd file appear in the center pane of the desktop like this:



Element Name	Content Model	Attributes
file	#PCDATA	name, source
formdata	(text+.file+)	
text	#PCDATA	name

3. Click **DTD Editor > Generate Java**. The Generate Java Beans dialog box opens.
4. In the **Package Name** entry field, enter the following package:  
schedule2.generated
5. For the **Root Element Name**, select **formdata**, then click **Finish**.
6. To see the generated classes, go to the Packages pane, expand the **ScheduleThis!**, **Schedulers**, and **servlets** folders, and then expand the **schedule2.generated** package.

### 6.4.6 Creating and deploying the Scheduler Web services

To create a Web service, you start with an existing Java bean that contains the functions a client application will invoke. You then create a Web Services Description Language (WSDL) file that describes the Web service. The IBM XML and Web Services DE provides a wizard to automate this process. We will use the wizard to create the Proxy class needed by ScheduleThis!, but not to deploy the service. Instead, we will use the admin tool provided by SOAP for deployment.

First, you must generate the Proxy classes that will be used by ScheduleThis! to invoke methods on the web service. Anytime you switch the server the web service is installed to, you must regenerate the Proxy classes because the classes contain a URL to the web

service. These steps show you how to generate the Proxy for WSPTScheduler. Follow the same steps to generated the Proxy classes for the other Scheduler beans.

1. From the desktop, click the **Navigator** tab. In the Navigator pane, select the **Schedulers\servlets** folder.
2. Click **File > New > WSDL Web service**. The Web Service Definition wizard opens.
3. Take all the defaults, including **Create a Web service (WSDL) from an existing Java bean**, and click **Next**. The Java bean Selection dialog box opens.
4. Select the **schedule2/algorithms/WSPTScheduler.class** file, and click **Next**. The bean classes for all the sample services end in Scheduler. The Method Selection dialog box opens.
5. In the dialog box, the **execute()** and **getPerformance()** methods should be selected. Click **Next**. The Web Service Definition dialog box opens.
6. For the **Deployment Environment**, just accept the default because we will not be using this wizard to deploy anyway. In the host field enter the host to where you will be serving the web service. Take the defaults in the remaining fields. Notice that the Web Service ID is **urn:WSPTScheduler**. This will be need when we deploy the service using the SOAP admin tool. Click **Next**. The Java bean Web Service Deployment dialog box opens.
7. Click **Next**. We will not be deploying here.
8. Click **Next**. The Java Client Proxy Generation dialog box opens. The Container field should contain **/ScheduleThis!/Schedulers/servlets**, append **/schedule2/generated** to the end of it. This will package the Proxy class into **schedule2.generated**, which is required by **ScheduleThis!**. The proxy class name field should be **WSPTSchedulerProxy**. Click **Next** and then **Finish**.
9. Open up the generated Proxy class by double clicking on **WSPTSchedulerProxy.java**. Make the class implement the **SchedulerProxy** interface. Type Ctrl-S to save.
10. Repeat steps for the other 4 sample services.

Before the services can be deployed, the code must be exported to SOAP. To export the code to Tomcat 3.2.2, do the following:

1. From the desktop, click the **Navigator** tab. In the Navigator pane, select the **ScheduleThis!\Schedulers\servlets** folder.
2. From the desktop, click **File > Export**. The Export wizard opens.
3. In the wizard, select **File system** from the **Select import source** list, then click **Next**.
4. In the **Directory** text field, specify the following path. Use the **Browse** button if necessary:

*soap-home*\WEB-INF\classes

where *soap-home* is the directory under Tomcat where Apache SOAP is installed..

5. Click **OK**, then click **Finish**.

Finally, copy the Proxy classes from *soap-home\WEB-INF\classes\schedule2\generated* to *ScheduleThis\WEB-INF\classes\schedule2\generated*, where *ScheduleThis* is the document base on Tomcat where ScheduleThis! is installed.

To deploy WSPTScheduler to Tomcat 3.2.2, do the following:

1. Enter <http://server:port/soap/admin> into your web browser and press enter. The Apache SOAP Admin tool should appear. Click on **Deploy** to deploy a new service.
2. Fill in the fields as shown below to deploy WSPTScheduler, and leave all other fields blank. For the other services you have to replace everywhere you see WSPTScheduler with the name of that service bean.

Property	Details										
ID	<input type="text" value="urn:WSPTScheduler"/>										
Scope	<input type="text" value="Application"/>										
Methods	<input type="text" value="execute getPerformance"/> (Whitespace separated list of method names)										
Provider Type	<input type="text" value="Java"/>										
	For User-Defined Provider Type, Enter FULL Class Name: <input type="text"/>										
	Number of Options: <input type="text"/>										
	<table border="1"><thead><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td><input type="text"/></td><td><input type="text"/></td></tr></tbody></table>	Key	Value	<input type="text"/>							
Key	Value										
<input type="text"/>	<input type="text"/>										
<input type="text"/>	<input type="text"/>										
<input type="text"/>	<input type="text"/>										
<input type="text"/>	<input type="text"/>										
Java Provider	Provider Class: <input type="text" value="schedule2.algorithm.WSPTScheduler"/>										
	Static?: <input type="text" value="No"/>										

3. Now scroll down to the bottom and click **Deploy**. You can view details about all the deployed web services by clicking on the **List** button, or remove deployed services by clicking on **Un-deploy**.
4. Repeat for the other 4 sample services.

---

## 6.5 Building your own Scheduler services

Eventually you are going to want to build your own scheduling web services. This section describes what you need to know so your web service will function properly with ScheduleThis!

### 6.5.1 The scheduler-config.xml file

This file is used by ScheduleThis! to add your scheduling service to its interface, and to load the Proxy class you provided to invoke your service. Your service will not be accessible unless added to this file. The scheduler-config file is also used to specify what files and additional data the user must provide to your web service. The file is stored in *ScheduleThis/WEB-INF/* where *ScheduleThis* is the context to which the ScheduleThis! application is installed. As a web service developer you would not normally have access to modify this file since most likely it is on another server administered by someone else. Instead, you would tell the administrator about your web service and provide him with the Proxy class and he would make the proper adjustments to the configuration file. An explanation of the structure of scheduler-config.xml is provided here, so you as a developer will understand better what information needs to be provided to the ScheduleThis! administrator.

You can click [here](#) for the default scheduler-config.xml file. This is the way the file looks when you first install ScheduleThis! The `<scheduler-config>` and `<problems>` elements must always be present and should not be changed.

#### `<problem>`

The `<problem>` element describes a certain problem to be solved (i.e. machine environment, constraints, and objective). The "label" attribute will be the text displayed in the drop down box to the user. The "value" attribute for all problems must be an integer, and the integers must start at 0 with the first problem and increment by one for each new problem IN ORDER. The order matters because ScheduleThis! uses an array to search for a problem, and the index to the array is the value attribute.

#### `<algorithms>`

Nested within each `<problem>` is an `<algorithms>` element. The purpose of the `<algorithms>` tag is to contain `<algorithm>` tags. Each `<problem>` must have one and only one `<algorithms>` element.

#### `<algorithm>`

The `<algorithm>` element is used to describe a specific web service that has been provided to solve the problem. There may be multiple algorithms for each problem. This means new services may be added under an existing `<problem>`. The tag must appear within an `<algorithms>` element. The "value" attribute for the `<algorithm>` element contains the fully qualified class name of the Proxy class. Java reflection will use this

name to load the class. The "label" attribute will be the text supplied to the user through a drop down menu. The "fileSource" attribute can take two values, either "file" or "url" indicating whether the web service will be provided with the content of the input files itself, or a URL to find the files.

#### **<format>**

Nested within each <algorithm> element are the supported file formats and additional form fields required for the algorithm. The <format> element adds a file format to the formats drop down menu. Each <format> must be nested within a <formats> element. This feature is available because the service could know how to parse information from more than one program. The "value" attribute for <format> will be passed to the service, and should be used to determine how to parse the input file.

#### **<form>**

There are currently two types of form elements supported. The type "file" creates an HTML file element. Use this for files that must be uploaded. The second is the "text" type, which creates an HTML text field. Use this for additional information required by the service, such as a discount factor or number of iterations. The "name" attribute specifies what the "name" parameter in the HTML <input> element will be. The "label" attribute is the text that will be displayed beside the form element telling the user what to enter.

## **6.5.2 The interface**

Your web service must implement and make visible two methods: execute() and getPerformance(). ScheduleThis will invoke these methods! to create the schedule and gather objective results respectively. It is suggested that you implement the schedule2.Scheduler interface.

### **6.5.3 public String execute(String formdata)**

The execute() method is the meat of your web service. It must parse all the input files and additional info. provided from ScheduleThis!, create a schedule, and return a String representation of the completed schedule in the same format as the input files. Since you expect the input files to have a certain structure you may wish to create a map for users of ScheduleThis! to use to export files. See the **ScheduleThis! User Guide** to learn about maps with Microsoft Project. The input parameter to execute(), formdata, is an XML String. Basically, an XML file stored as a String. Click [here](#) for a sample formdata String. The XML String contains the files uploaded by the user and the additional information required by the algorithm. The XML String can contain two types of elements, <file> and <text>. There will be a <file> element for each file uploaded by the user. The attribute "name" is the same as the "name" attribute of the HTML <file> element. The "source" attribute tells your service whether the content of this input file is provided by the <file> element, or a URL to the file. There is a <text> element for every

text field that was specified in the scheduler-config.xml file. The only attribute of <text> is "name" which again is just the name used in the HTML form.

#### **6.5.4 public String getPerformance()**

The getPerformance() method returns objective results on your algorithm such as flow time, makespan, etc. The objectives you want to calculate and return are up to you. You can choose just to return null, and ScheduleThis will display no results! The format of the String is name=value pairs separated by commas. Click [here](#) to see an example of a String that would be returned from this method.

---

## 7 Appendix C: ScheduleThis! Installation Guide

This guide is meant to be viewed by a web browser. Depending on how you are viewing it you may not be able to follow the links that are provided. It is available with the ScheduleThis! online documentation.

This guide is meant for someone who is trying to install and/or configure ScheduleThis! For ScheduleThis! to do anything the sample web services must be installed. Please read the **Web Service Development Guide** to learn how to create and deploy the sample services provided with the ScheduleThis! distribution.

---

### 7.1 Outline

1. [Installing ScheduleThis!](#)
  2. [Configuring ScheduleThis!](#)
    - [The scheduler-config.xml file](#)
    - [Proxy classes](#)
- 

### 7.2 Installing ScheduleThis!

This section explains how to install ScheduleThis! to the Tomcat server. It should work on any web server that has JSP and servlet support, but Tomcat 3.2.2 is the only server that its been tested on.

Take the **ScheduleThis** folder from the distribution and copy it into the TOMCAT\_HOME/webapps folder. You must restart the server for the changes to take effect. You should now be able to access ScheduleThis! by entering <http://server:port/ScheduleThis> where server and port are the server name and port number of your Apache Tomcat installation.

Before ScheduleThis! will work correctly the sample web services must be deployed. The **Web Service Developers Guide** has a tutorial that explains how to create and deploy the sample services provided with the distribution in the **Schedulers** folder.

You may want to install ScheduleThis! to a different context than /ScheduleThis. No problem. Just rename the folder to whatever context you wish. In addition, you must also modify the web.xml file. Change the <param-value> of the initial parameter "context" in ScheduleConfigServlet to whatever you renamed the folder to. You must include the preceding slash. After restarting Tomcat you can access ScheduleThis! under the new context.

---

## 7.3 Configuring ScheduleThis!

When first installed, ScheduleThis! is already configured to use the 5 sample web services provided with the distribution. Eventually, you will want to remove the samples and add more services. This is done through the scheduler-config.xml file located in /ScheduleThis/WEB-INF. Changes in this file are reflected by changes in the pull down menus of the ScheduleThis! interface. What follows is a description of the structure of the file and the meaning of each element and attribute.

### 7.3.1 The scheduler-config.xml file

You can click [here](#) for the default scheduler-config.xml file. This is the way the file looks when you first install ScheduleThis! The <scheduler-config> and <problems> elements must always be present and should not be changed.

#### <problem>

The <problem> element describes a certain problem to be solved (i.e. machine environment, constraints, and objective). The "label" attribute will be the text displayed in the drop down box to the user. The "value" attribute for all problems must be an integer, and the integers must start at 0 with the first problem and increment by one for each new problem IN ORDER. The order matters because ScheduleThis! uses an array to search for a problem, and the index to the array is the value attribute.

#### <algorithms>

Nested within each <problem> is an <algorithms> element. The purpose of the <algorithms> tag is to contain <algorithm> tags. Each <problem> must have one and only one <algorithms> element.

#### <algorithm>

The <algorithm> element is used to describe a specific web service that has been provided to solve the problem. There may be multiple algorithms for each problem. This means new services may be added under an existing <problem>. The tag must appear within an <algorithms> element. The "value" attribute for the <algorithm> element contains the fully qualified class name of the Proxy class. Java reflection will use this name to load the class. The "label" attribute will be the text supplied to the user through a drop down menu. The "fileSource" attribute can take two values, either "file" or "url" indicating whether the web service will be provided with the content of the input files itself, or a URL to find the files.

#### <format>

Nested within each <algorithm> element are the supported file formats and additional form fields required for the algorithm. The <format> element adds a file format to the formats drop down menu. Each <format> must be nested within a <formats> element. This feature is available because the service could know how to parse information from more than one program. The "value" attribute for <format> will be passed to the service, and should be used to determine how to parse the input file.

### **<form>**

There are currently two types of form elements supported. The type "file" creates an HTML file element. Use this for files that must be uploaded. The second is the "text" type, which creates an HTML text field. Use this for additional information required by the service, such as a discount factor or number of iterations. The "name" attribute specifies what the "name" parameter in the HTML <input> element will be. The "label" attribute is the text that will be displayed beside the form element telling the user what to enter.

### **7.3.2 Proxy classes**

You must obtain the Proxy class from a web service before it can be invoked. The Proxy class needs to implement the SchedulerProxy interface and be packaged schedule2.generated. If both those conditions are met, place the class in *ScheduleThis*/WEB-INF/classes/schedule2/generated, where *ScheduleThis* is the document base on Tomcat where ScheduleThis! is installed. If you haven't already added the service to the scheduler-config file see [The scheduler-config.xml file](#).

---