ShipWare Field Session Group
Creating Shipping Comparison Software

Client: Kenneth Riskey, ShipWare
Team: Billy Duran, Kevin Raber, Peter Collins

Abstract

The problem that Ship Ware wanted solved was to have a downloadable program that compared shipping costs between the three large shipping companies (FedEx, USPS, and UPS). Once the user enters in details about their package, they should then be able to compare costs and transit times and pick a method that best suits their needs. After the user fills in all required information they should be able to print off the postage or prepaid label. The difficulty of this project was twofold, one in creating a GUI that was straightforward for the user and two in the retrieval of the shipping information from each of the three companies.

There were two options explored for this information retrieval, the first was to pull the necessary information from each website as the user requests the data. The second was to hard code the rate equations that each company uses into the program. These were the best choices after researching how current programs work given the availability of data provided by the three companies. The final design connects to each company's API servers and sends a request for shipping details. The servers then send responses to our program. Our program parses the response and outputs data for the user to read.

Introduction

The number of online sales has increased significantly due to the spread of the internet and rise in online companies. However, purchases online that are not digital require items to be shipped from the merchant to the consumer. When the number of items being shipped is fairly large, shipping costs become an important part of the associated business costs for companies. Therefore, companies which can reduce their shipping costs will save large sums of money.

Our client represents the firm ShipWare and the goal of the client is to have a program that retrieves the shipping costs from three main shipping companies, USPS, FedEx, and UPS, and to then allow a user to choose the best option that suits his/her needs. The problem currently is that there is not a program that can compare the shipping costs and transit times between the three companies. The benefit to the client is an easy-to-use program that sorts through the shipping data automatically.

Requirements

The requirements were very flexible, allowing different approaches to be tested. The only requirements given were in regards to the program’s appearance and the major pieces that needed to be included.

A. Functional Requirements

1. Retrieve data from three main shipping companies FedEx, USPS, and UPS
2. Take user input for return address, delivery address, size, weight, and type of package
3. Give user options for type of shipping times
4. Show best price among the three companies
5. Able to print off postage and pre-paid labels

The client gave access to their Stamps.com account, which is a program that provides price information and ability to prints off postage for USPS only. This program represented a model of what the client desired the program's appearance and functionally should be.

B. Non-Functional Requirements

1. Program should be downloadable
2. Program should be user friendly

The non-functional requirements were to ensure that the future users of the program could easily understand how to navigate through the program quickly and easily.

High-level Design

As shown in figure 1, the user interfaces with the GUI (graphical user interface) and enters required information such as shipping addresses, dimensions, weight, and type of package. The program then retrieves information from the big three shipping companies' shipping servers. From there, the user chooses which shipping option he/she would like and appropriate sample postage/pre-paid labels will be generated. The user then can print the sample postage/pre-paid labels.

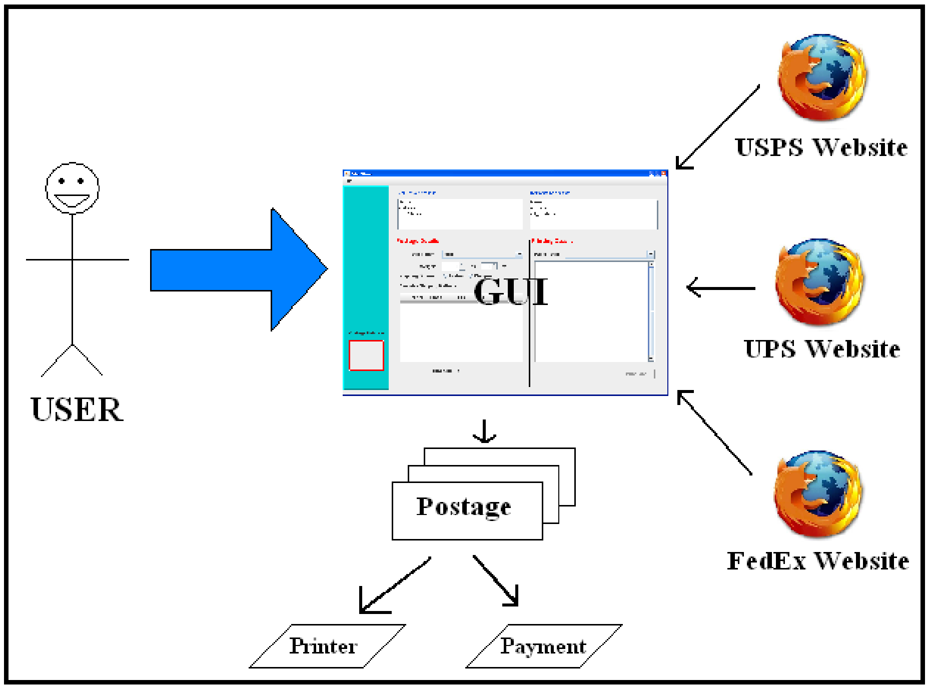


Figure 1: Architecture Diagram

Detail Design

A. <u>GUI</u>

The GUI provides the interface between the user and the program. It requires the user's inputs in order to display correct shipping information to the user. The GUI requires the user to enter both a delivery and return address, dimensions, weight, and the type of package it is. From there, the information is processed into three different XML (Extensible Markup Language) formats required by each of the big three and sent to their respective servers. Shipping information is received from the servers in XML format and parsed by the program getting shipping cost, method, and transit time. The shipping results are then displayed via the GUI. Once the user has selected a shipping method, a sample preview is shown in the printing details window, where the user can print it. The GUI also displays the total cost for sending the package for the chosen method. The GUI was designed using NetBeans IDE, which is a GUI generator, while most of the functionally was implemented using Eclipse, an open source Java IDE. The GUI uses Java action listeners to allow control over what each part of the program will do and to allow us to give large functional flexibility to the program. Our initial choice for IDE was Eclipse, but after being turned onto NetBeans by our fellow students, we found out that the drag and drop functionality for the GUI would allow us to create exactly what we wanted in much less time than with Eclipse. When trying to figure out how to display the shipping information during runtime, NetBeans became impossible to use due to the inability to edit the auto-generated GUI code. To solve this, we copied our project back into Eclipse, edited what we needed to and continued adding all functionality with Eclipse.

B. <u>Information Retrieval</u>

The program needed to pull information from the big three websites. Initially, we believed that there were only two available options for accomplishing this. The first of which was connecting to each of the big three websites, entering the users input into text fields on the websites, clicking the "calculate" button on the site, and then reading in the output from the webpages. The alternative option was to get access from the big three to their equations for calculating rates and hard coding these into our program. These two options, however, would not suffice for this project. The first would require the program opening up a web browser, controlling the computer’s mouse to click into text fields and to click the “calculate” buttons. This would be time consuming, cause the user to lose control of the computer, and not very good programming technique. The alternative would be more simple, but not maintainable. If any of the big three changed any of their calculations, a patch would have to be created for the software, which could become a tiresome task; depending on how often calculations were changed. In order to find out which of these options would be best, or to see if there was another option, we contacted the three companies hoping for some input and that is when they told us that each of them had APIs (Application Programming Interface) already made. These APIs are made to interface with other programs, exactly what we needed. The APIs allow us access to the online services of each of the big three. After receiving the delivery information from our user, the program converts it into XML using formats unique to each of the companies’ APIs and sends it to the respective server. These servers will then process the data and send back information on the different shipping options available for the inputted data. After our software receives this information, it will format it in a uniform way and display it to the user.

C. Price/Time Comparison

The purpose of the program is to compare the prices and transit times between the big three. So once the information is retrieved, the program sorts through all available shipping options and finds the best deal for the user. This means looking at transit times and shipping cost, doing a comparison, and outputting to the user the list of shipping methods, sorted by the cheapest/fastest method for their package. The user has the option of choosing between the fastest shipping method and the cheapest shipping method. This requires separate algorithms to find the sorted list based on which option the user chose. The algorithms for each had to generate a unique array containing the new sorted order because the façade we used would not allow us to sort the data without also deleting it.

D. Printing

The postage and labels need to be printed off for the user after they decide on the best option for them. The plan for payment will work by linking to their account with ShipWare through XML. Then the user can either put a set amount on their account in amounts of 10$, 15$, etc. or pay as they go. Once the user chooses a specific shipping option, an XML request is sent to the server of that provider requesting the label. An XML response is received by our software and displays the image of the label for the user to see. They then choose the paper type, defaulted to 8.5”x11” letter and print the label.   The program will need to interact with the printer which is doable in Java as the Print command. The payment would require many more weeks of work as well as a certification process through the company and was out of scope for this project. The printing also required an entirely separate API specific to printing. However, we did include all the code necessary to print an image; the only thing needed is implementing the postage/label creation API.

E. UML

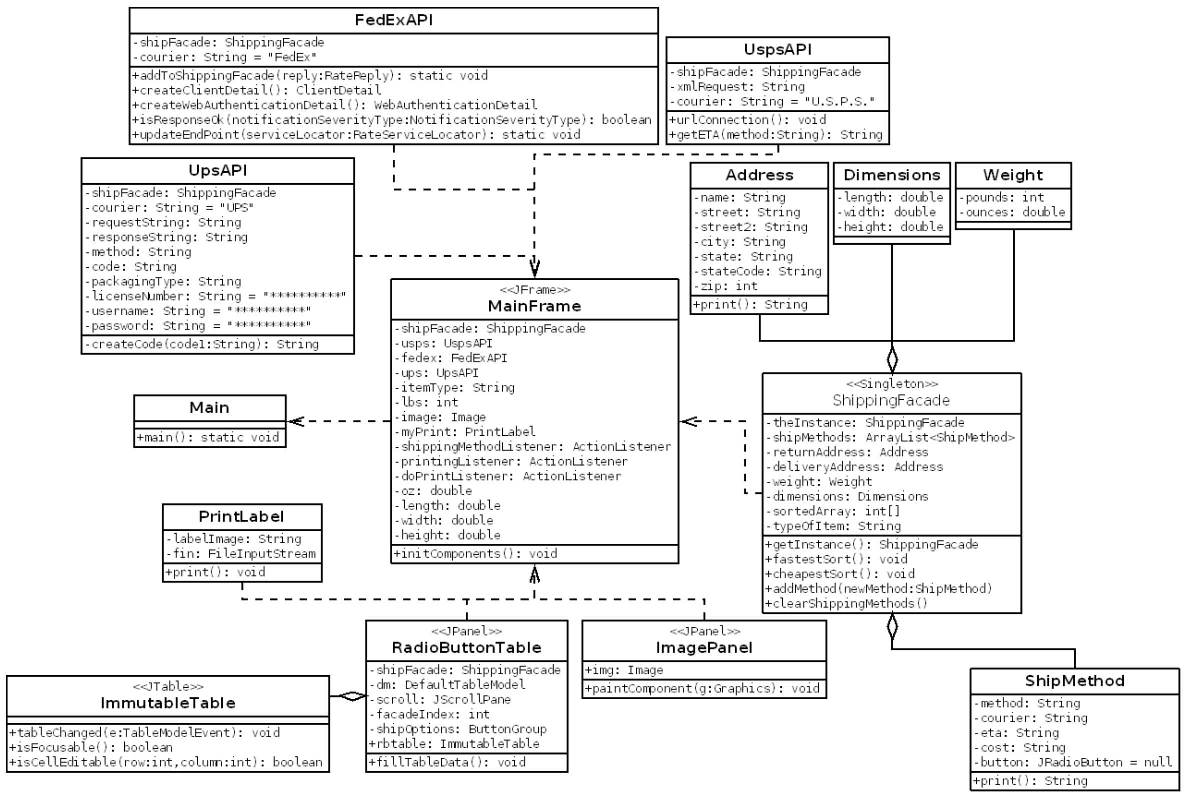


Figure 2: UML Diagram

Use Cases

D.1 Determine Shipping Options

Primary Flow:

1. Enter Return Address
2. Enter Delivery Address
3. Enter Package Weight
4. Enter Package Dimensions
5. Choose Package type
6. Choose Sort Option
7. Delivery info is Calculated
8. Delivery info is displayed in the Available Options box

Alternate Flow: Invalid zip Code/city/state

1. At 1 or 2 in "Determine Shipping Options" user uses an invalid zip Code/city/state then display dialog box with a zip code/city/state error message

Alternate Flow: Missing Entries

1. At 1 or 2 in "Determine Shipping Options" user misses a necessary entry display message to let user know.

1. At 3 and 4 if no values are entered display missing value message

Alternate Flow: Invalid Weight

3. At 3 in "Determine Shipping Options" user enters a weight less than or equal to 0 or greater than the maximum shipping weight then display dialog box with a invalid weight error

D.2 Print Postage

Primary Flow:

1. Determine Shipping Options
2. Select a Shipping Method
3. Choose Printing Paper Size
4. Click Print
5. Postage is printed

Alternate Flow: Missing Printer

1. At 4 if no printer is plugged into the computer

Implementation Details and Results

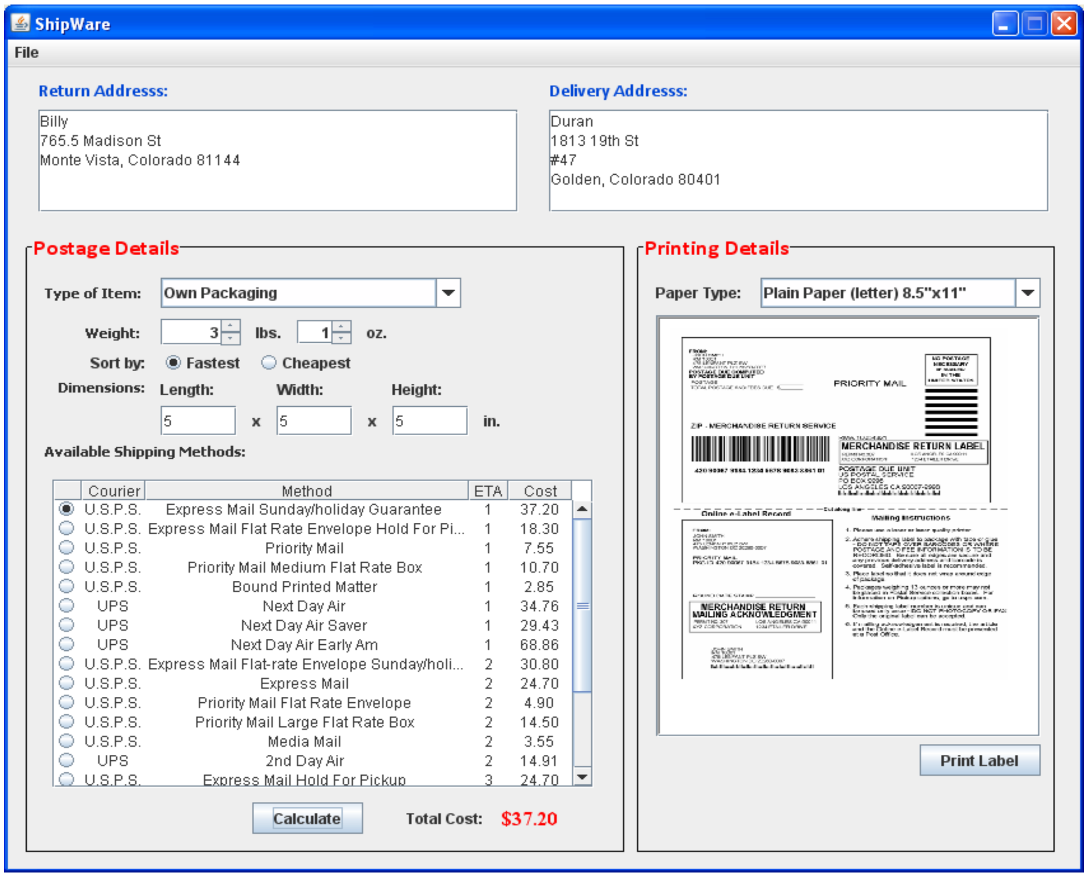
The project was initially coded using Eclipse and Java as the programming language. The reason for this was that the program was to be fairly GUI intensive and Java seemed the best language to work with the GUI in. However, it was slow going and would have taken a fair amount of time to complete just the GUI. That was when there was a switch to NetBeans as the program used. This is a drag and drop program using Java that makes creation of the GUI very easy and fast. It allowed more time to work on the data retrieval and comparison functionality. Once the GUI design was completed, we imported the code back over to Eclipse because Eclipse allowed us to edit the auto-generated GUI code that was locked in NetBeans. The other reason for using Java was that the three companies all use XML server requests in order for outside programs to retrieve information and Java already has a strong XML support in the language. This made it fairly easy to integrate the software of the companies with our own.

The big three companies already had APIs created that were exactly what was needed to get the information for the program. Therefore, when it came to data requests we were able to view sample code and modify it in order to both parse the information and get the correct data. There were also a few interesting libraries used that helped the process. In the GUI aspect, we used javax.swing.table because it allowed the shipping methods to be outputted in a clear format for the user. We struggled trying to get this part of the GUI to work until we found this library. Two other major libraries were the DocumentBuilder library and the NodeList library. The DocumentBuilder is what allowed us to parse the responses we received from the companies. NodeList also helped with this by allowing us to break down the document that was created according to what pieces of information were being looked for. Without these libraries the amount of time to complete the requirements would have to great within this time span.

The biggest problem we ran into came from working with the companies’ code. There were many times when the requests being sent were incorrect by very minor details, therefore, the server couldn't respond and the error was very hard to debug. This happened for each company and the way we were able to solve it was by contacting the technical support group of each company. Another major issue was trying to integrate their code with ours. There were a lot of factors that caused problems; the main one being that FedEx's code is mostly stand alone. This means that we had to fix the build path and refactor the code which was very large in order to integrate it into our program.

Results

The project asked for us to retrieve information from a server and then parse it. The following shows both requests and replies to each company. There were two tests to verify the accuracy. The first was that the companies gave sample inputs and outputs to use to see if our program would produce an output that matched the output provided. Then, once we switched from the test data to actual data, we went online to each company’s website and compared the results. Figure 3 shows the final GUI with sample output.



Return Addresss:
Billy
765.5 Madison St
Monte Vista, Colorado 81144

Delivery Addresss:
Duran
1813 19th St
#47
Golden, Colorado 80401

Type of Item: Own Packaging

Weight: 3 lbs. 1 oz.

Sort by: [✓ Fastest] Cheapest

Dimensions: Length: 5 x Width: 5 x Height: 5 in.

Available Shipping Methods:

	Courier	Method	ETA	Cost
●	U.S.P.S.	Express Mail Sunday/holiday Guarantee	1	37.20
○	U.S.P.S.	Express Mail Flat Rate Envelope Hold For Pi...	1	18.30
○	U.S.P.S.	Priority Mail	1	7.55
○	U.S.P.S.	Priority Mail Medium Flat Rate Box	1	10.70
○	U.S.P.S.	Bound Printed Matter	1	2.85
○	UPS	Next Day Air	1	34.76
○	UPS	Next Day Air Saver	1	29.43
○	UPS	Next Day Air Early Am	1	68.86
○	U.S.P.S.	Express Mail Flat-rate Envelope Sunday/holi...	2	30.80
○	U.S.P.S.	Express Mail	2	24.70
○	U.S.P.S.	Priority Mail Flat Rate Envelope	2	4.90
○	U.S.P.S.	Priority Mail Large Flat Rate Box	2	14.50
○	U.S.P.S.	Media Mail	2	3.55
○	UPS	2nd Day Air	2	14.91
○	U.S.P.S.	Express Mail Hold For Pickup	3	24.70

Total Cost: $37.20

Paper Type: Plain Paper (letter) 8.5"x11"

Figure 3: Final GUI with sample output

USPS

The following is the request format that USPS requires for their server to respond with the correct data. The first line is the code that allows access to their production server. The following lines given are always the ZIP codes and weight of the package. This information is pulled from the user given inputs.

Request:

```
<RateV3Request%20USERID="**********">
  <Package%20ID="1ST">
    <Service>ALL</Service>
    <ZipOrigination>81144</ZipOrigination>
    <ZipDestination>80401</ZipDestination>
    <Pounds>40</Pounds>
    <Ounces>3.0</Ounces>
    <Size>REGULAR</Size>
    <Machinable>true</Machinable>
  </Package>
</RateV3Request>
```

Response:

This is how the server responds to the request. The data is then parsed using the tags MailService and Rate and that information is displayed to the user.

```
<Postage CLASSID="23">
      <MailService>Express Mail Sunday/Holiday Guarantee</MailService>
      <Rate>158.40</Rate>
</Postage>
<Postage CLASSID="25">
     <MailService>Express Mail Flat-Rate Envelope Sunday/HolidayGuarantee</MailService>
      <Rate>30.80</Rate>
</Postage>
<Postage CLASSID="2">
      <MailService>Express Mail Hold For Pickup</MailService>
      <Rate>145.90</Rate>
</Postage>
```

UPS

Request:

UPS requires more data to be sent than USPS does. It needs the return address and the delivery address. Using that information it generates the associated costs and shipping methods.

```
<?xmlversion="1.0"?>
<AccessRequestxml:lang="enUS">
	<AccessLicenseNumber>**************</AccessLicenseNumber>
	<UserId>ShipWare.com</UserId>
	<Password>********</Password>
</AccessRequest>
<?xml version="1.0"?>
<RatingServiceSelectionRequestxml:lang="enUS">
	<Request>
		<RequestAction>Rate</RequestAction>
	</Request>
	<PickupType>
		<Code>01</Code>
	</PickupType>
	<Shipment>
		<Shipper>
			<Address>
				<AddressLine1>765.5 Madison st.</AddressLine1>
				<AddressLine2></AddressLine2>
				<City>MonteVista</City>
				<StateProvinceCode>CO</StateProvinceCode>
				<PostalCode>81144</PostalCode>
				<CountryCode>US</CountryCode>
			</Address>
		</Shipper>
```

Response:

The response is very large and contains a large quantity of data that is unneeded for our purposes. The following is the data that is output to the user and is parsed from the response using the tags CurrencyCode and MonetaryValue.

```
<TotalCharges>
	<CurrencyCode>USD</CurrencyCode>
	<MonetaryValue>30.47</MonetaryValue>
</TotalCharges>
<GuaranteedDaysToDelivery>1</GuaranteedDaysToDelivery>
```

FedEx

Request:

The request contains all the same detail that UPS needed. The following is the partial request string that is sent to the FedEx servers.

```
<v8:RequestedShipment>
        <v8:ShipTimestamp>2010-06-20T11:32:55.958-04:00</v8:ShipTimestamp>
        <v8:DropoffType>REGULAR_PICKUP</v8:DropoffType>
        <v8:ServiceType>GROUND_HOME_DELIVERY</v8:ServiceType>
        <v8:PackagingType>YOUR_PACKAGING</v8:PackagingType>
        <v8:TotalWeight>
          <v8:Units>LB</v8:Units>
          <v8:Value>20.0</v8:Value>
        </v8:TotalWeight>
        <v8:Shipper>
          <v8:AccountNumber>#########</v8:AccountNumber>
          <v8:Address>
            <v8:StreetLines>1059 Valley View</v8:StreetLines>
            <v8:City>Hurst</v8:City>
            <v8:StateOrProvinceCode>TX</v8:StateOrProvinceCode>
            <v8:PostalCode>76053</v8:PostalCode>
            <v8:CountryCode>US</v8:CountryCode>
            <v8:Residential>true</v8:Residential>
          </v8:Address>
        </v8:Shipper>
```

Response:

The responses vary depending on what flags you send. Flags meaning type of shipping options you want. In this case we asked for all return flags to be sent. The sample response was fairly easy to retrieve since it was all in the following format. The tag for FedEx to parse is Amount.

```
<v8:TotalNetCharge>
              <v8:Currency>USD</v8:Currency>
              <v8:Amount>13.19</v8:Amount>
</v8:TotalNetCharge>
```

Glossary

GUI – Graphical user interface

API – Application programming interface

XML – Extensible mark-up language, set of rules for encoding documents in machine readable form

Tag – The keyword in the XML response that matches the searched for word