UMass

Electrical and Computer
Engineering Department

ECE 264, **Object-Oriented Software Development**

Lab 6:: 100 points (see Grading Notes for details) ::
Wednesday lab session (March 13) Due March 15, Friday by 5:00 pm
Monday lab session (March 25) Due March 27, Wednesday by 5:00 pm

1. Introduction and Acknowledgements

This lab assignment helps you to learn how to specify different types of software diagrams using Microsoft Visio™. This document, which was revised by Professor Honggang Wang, in spring 2013, is based heavily on the previous ECE 264 Lab Manual, written by by Makia Powell, an ECE 264 TA under Professor Hong Liu, in Spring 2008.

2. Deliverables

Repeat following the procedures and output all the similar figures from figure 1 to figure 5. To acquire these figures, you can do screen copy (using "PrintScr" key) and put all the screen copies (figures) in a single word file. Submit your word format file to your class directory on the M:\ ECE-264. Call your project *lab6_Visio*, and ensure that your source file name is ***lab6_*** *Visio****.docx***. You should place only the .docx file on the M:\ECE264. Failure to meet this specification will reduce your grade, as described in the ECE 264 lab grading handout, which you are strongly encouraged to read before starting the lab.

Table of Contents

Purpose 3
Getting Started – Launching Microsoft Visio ™ 3
Creating a UML Use-Case Diagram 4
Creating a UML Class Diagram 8
Creating a UML Sequence Diagram 15
Creating a UML Communication Diagram 18
Creating a UML Activity Diagram (Flowchart) 22
Creating a UML State Diagram 26
Visio™ Tips and Shortcuts 28

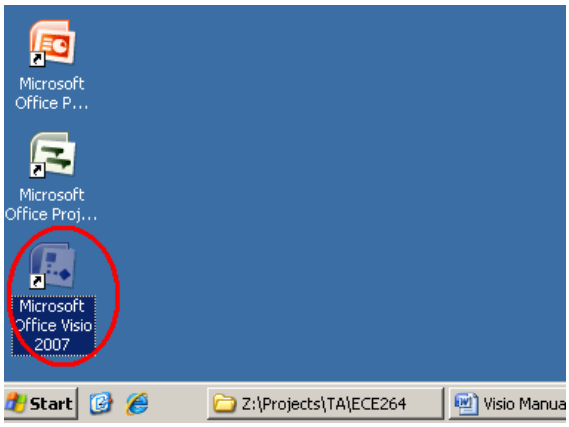
ECE 264, **Object-Oriented Software Development**

Purpose

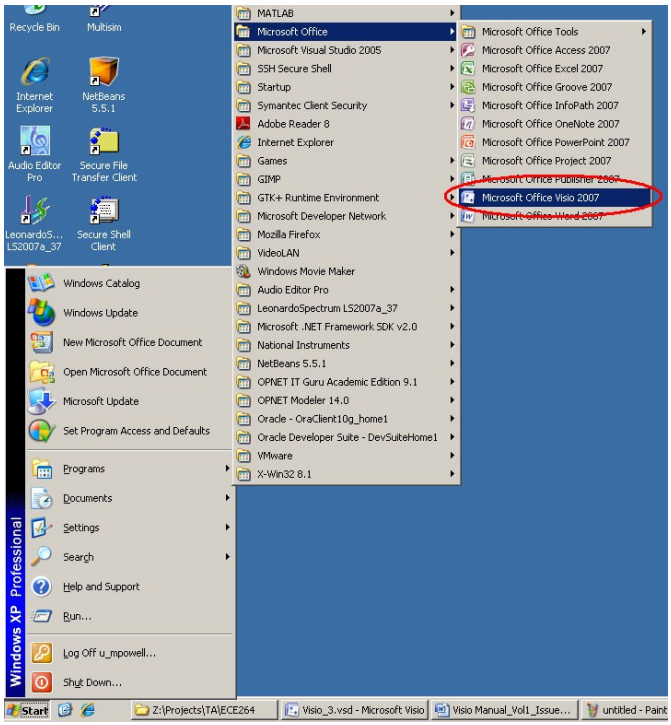
This manual covers how to specify different types of software diagrams using Microsoft Visio™

Getting Started – Launching Microsoft Visio ™

On most lab computers, Microsoft Visio can be found on the desktop. Double-Click on the Visio Icon to launch Microsoft Visio.



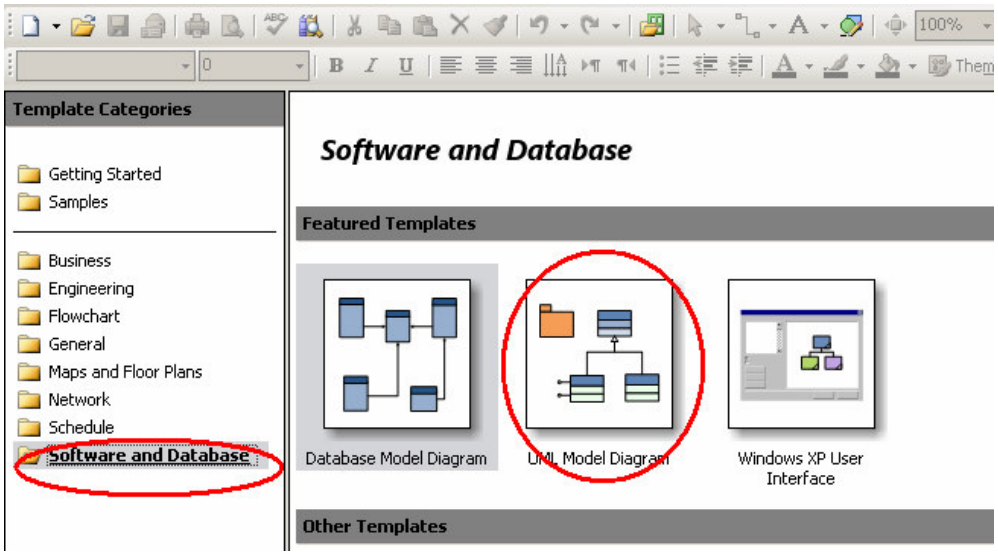
Alternatively, you can select **Start** **Programs** **Microsoft Office** **Microsoft Office Visio** from the main menu.



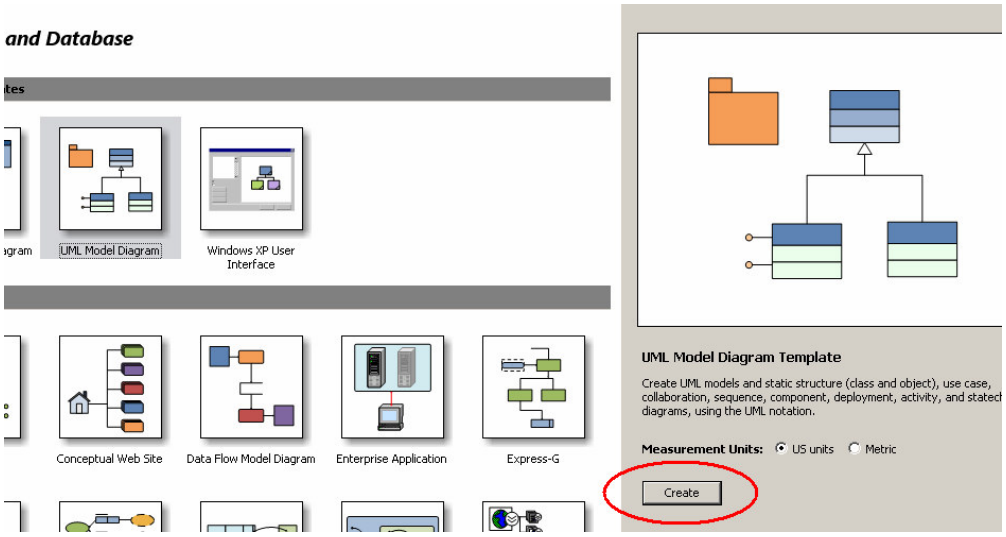
Next, we will demonstrate how to create a few types of software development diagrams: UML use-Case Diagrams, UML Class Diagrams, and UML Activity Diagrams (flowcharts).

Creating a UML Use-Case Diagram

When you open Visio™, there will be a few types of diagrams available from basic block diagrams, to engineering schematics. Since we are doing software development, click on the **Software and Database** on the left tab. This will bring up the software and Database templates. Select **UML Model Diagram** to open the UML Diagram Template Dialog on the right.



Next click **Create** to open the template.

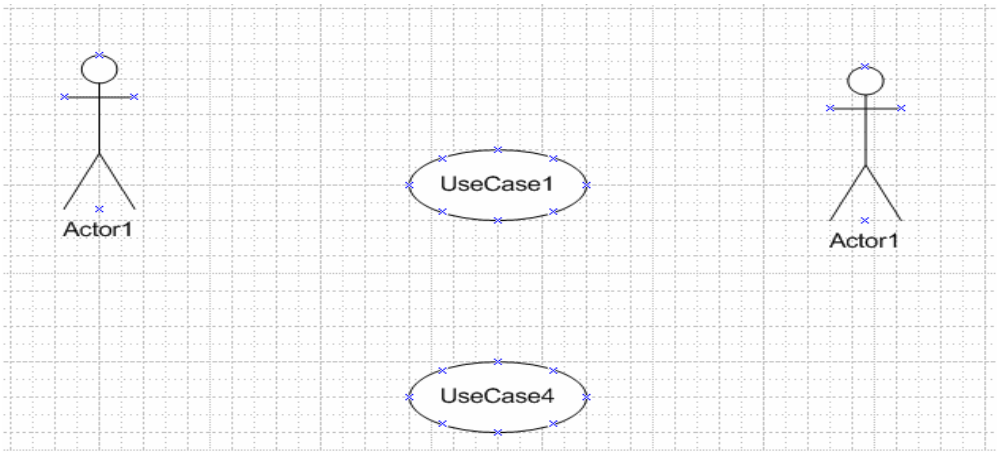


Now we can start creating diagrams. Since the first type of diagram is Use Case, we will create a fictitious Use Case for an application which greets the user, then exits. Depending upon the location of user, the application will greet the user in a different language. In short, an object-oriented version of "Hello World."

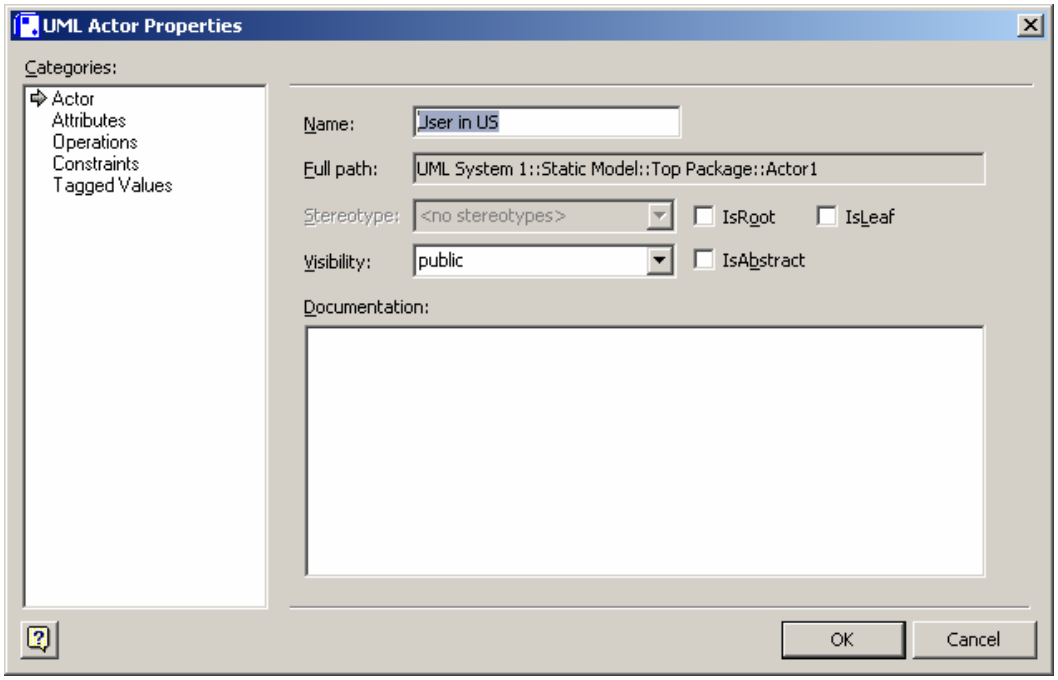
Each tab on the left of your screen opens symbols for different types of UML diagrams. Since we are doing a use-Case, click on the use case tab to the left of your screen.

ECE 264, **Object-Oriented Software Development**

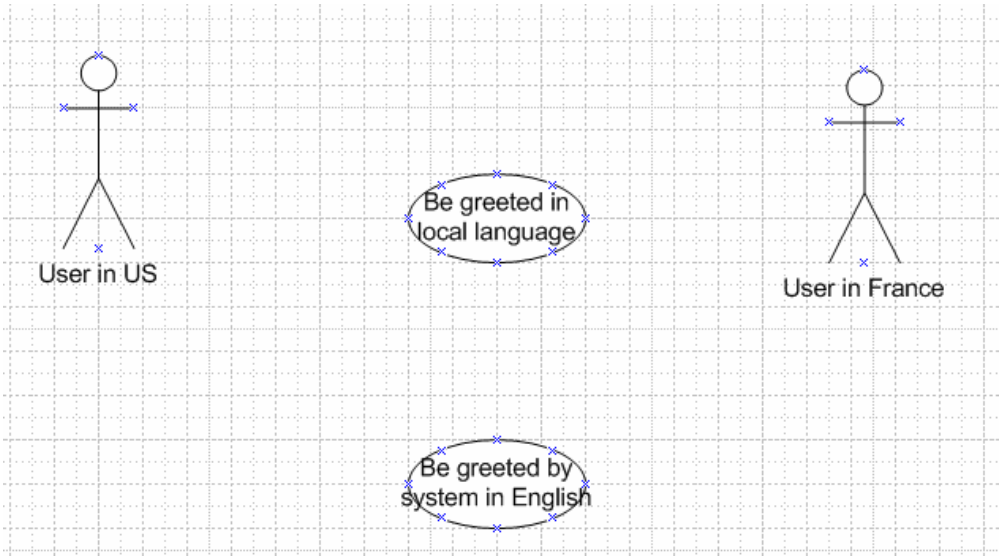
Drag, the following icons to the screen:



The stick icons represent actors, and the bubbles represent activities. Double click on one the actors to open up an options dialog. Each tab to the right contains different attributes that we can change. To change the name of the actor, enter a new name in the name dialog.

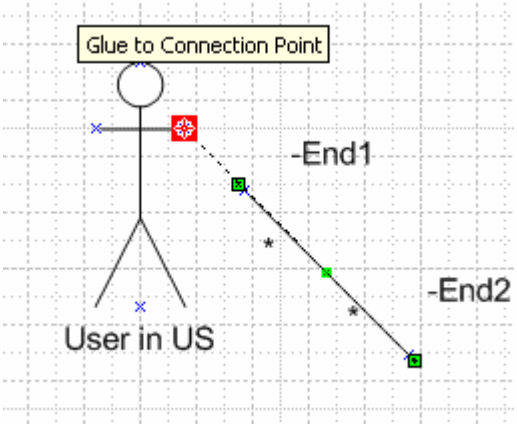


Label the diagram as shown:



Next, let's make connections between the actors. To demonstrate communication, select the straight line (click down with left mouse button) under the **UML Static Structure** palette, and drag it onto the diagram.

Next , drag both ends of the line to connect the user to the system. Notice that connection points have blue crosses, and when you hit a connection point, it is highlighted red.



ECE 264, Object-Oriented Software Development

Connect the diagram as follows. Notice that the last arrow is an ***Extension*** indicator. It shows that the local language greeting ***is an extension*** of the English language greeting. This usually indicates **inheritance** in C++ programming. If we used the **Uses** arrow instead, then this would indicate **association/aggregation**.

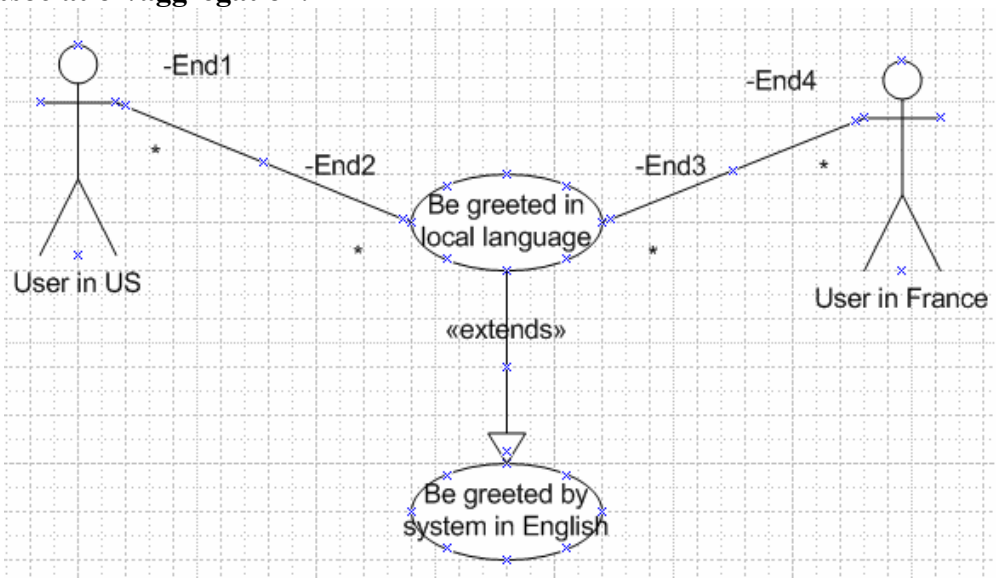
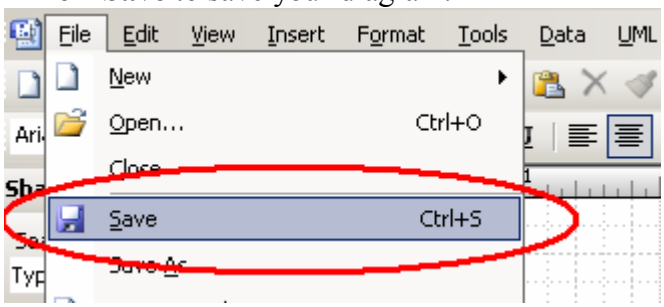
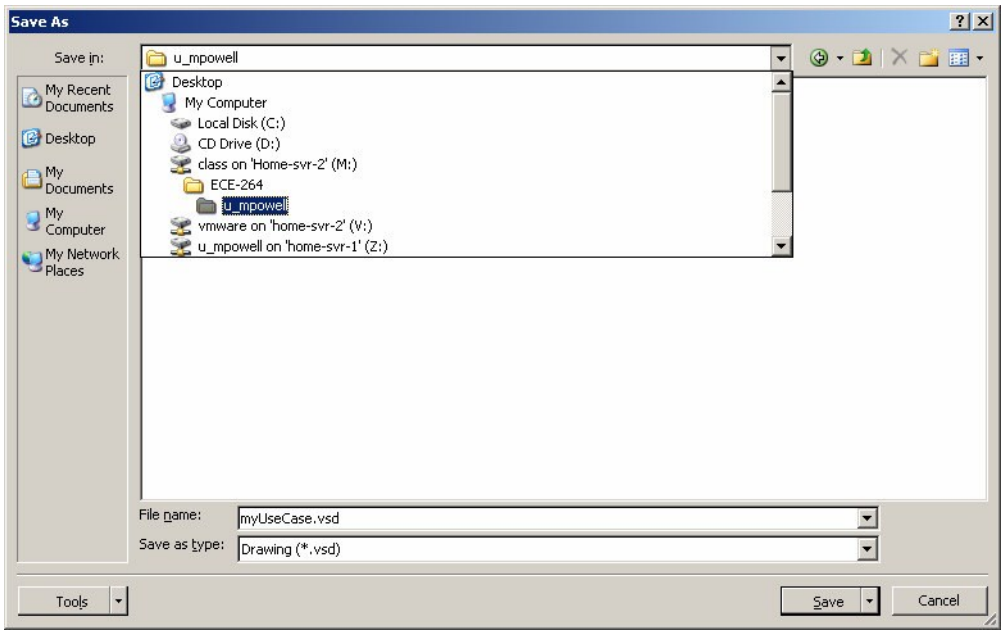


Figure 1: UML Use–Case Diagram

Now we are done. Click **File** **Save** to save your diagram.



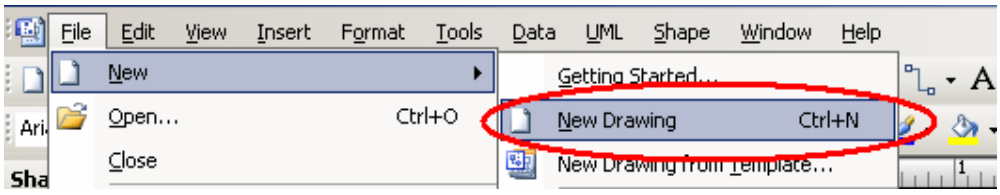

Save As
Save in:
u_mpowell
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
Desktop
My Computer
Local Disk (C:)
CD Drive (D:)
class on 'Home-svr-2' (M:)
ECE-264
u_mpowell
vmware on 'home-svr-2' (V:)
u_mpowell on 'home-svr-1' (Z:)
File name:
myUseCase.vsd
Save as type:
Drawing (*.vsd)
Tools
Save
Cancel

ECE 264, Object-Oriented Software Development

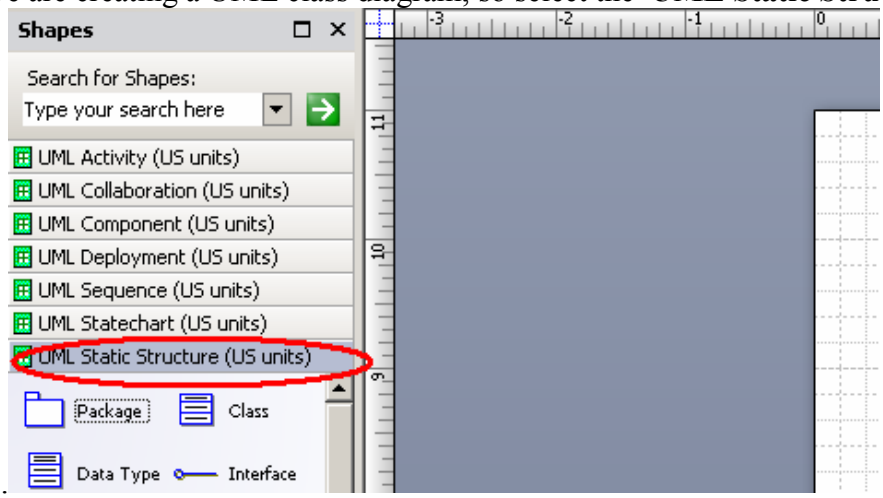
Creating a UML Class Diagram

Next, we will create a Class diagram for the same application, and also cover how to show association (aggregation), and inheritance relationships between classes. Since the local language greeting is a extension of the English language greeting, it appears that we will need at least two classes. One for the English language greeting, and a second for the local language greeting. We will also add a third class for a Translator.

Click, **File** **New Drawing** to start another UML diagram.

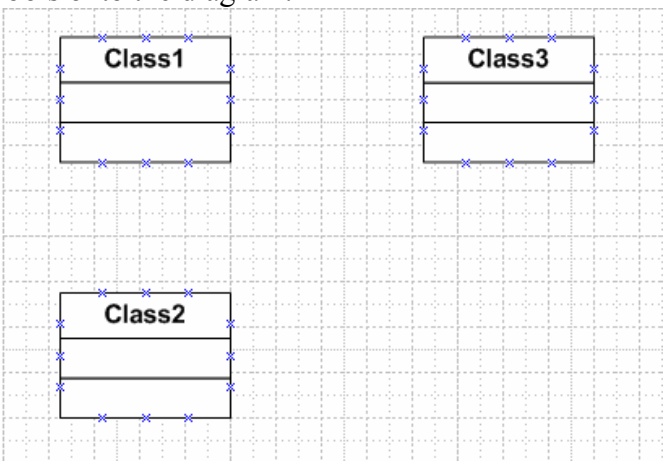


This time, we are creating a UML class diagram, so select the **UML Static Structure** palette.

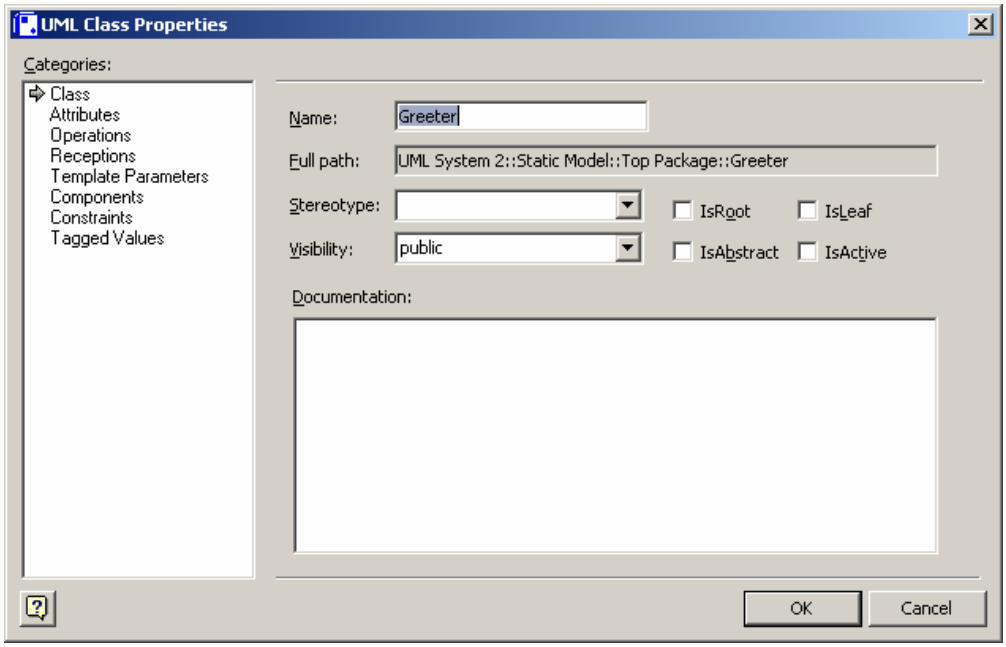


This opens the **UML Static Structure** palette (for UML Class Diagrams) .

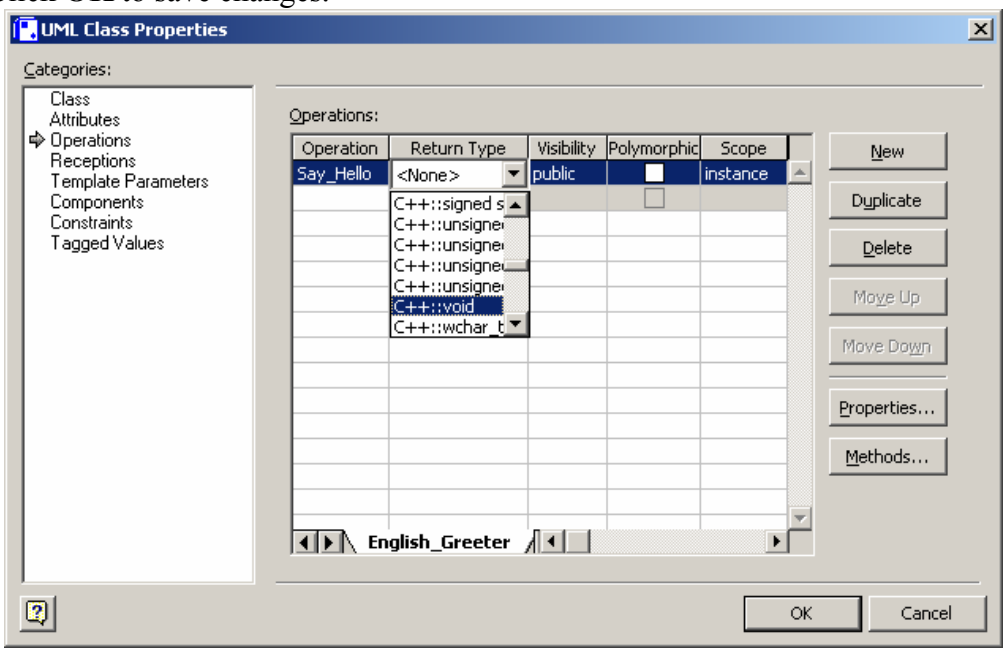
Drag the following symbols onto the diagram.



Next, double-click on class 2. This is the base class, which greets the user in English. Rename the class Greeter.



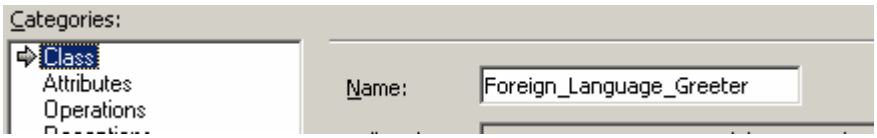
This class has no data members, but it does have one method which says "Hello World." Select **Categories:Operations** and enter this function. The return type is C++ void, and this is a public function Click **OK** to save changes.

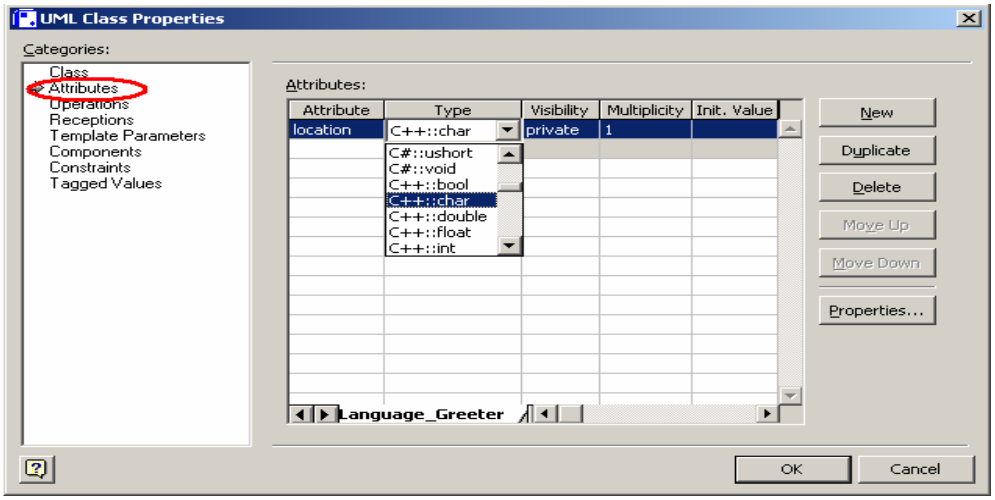


Next, click on **Class1**. This is our **Foreign_Language_Greeter**. It translates the English greeting depending upon location using a Translator. It has a location data member, and a function which sets the location and translates the English greeting.

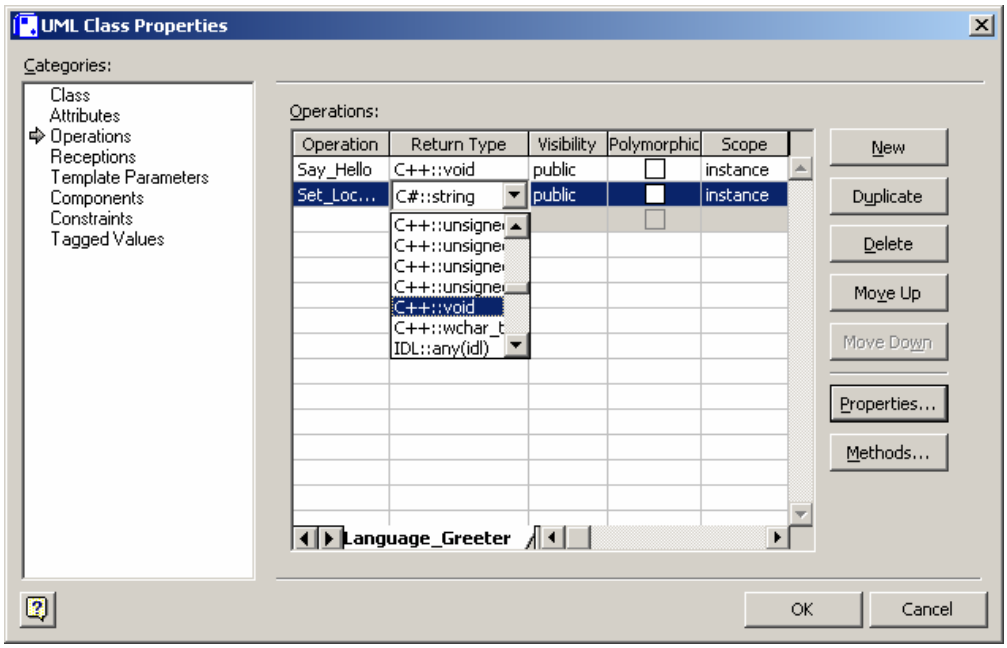
First Double-click on **Class1** and change the class name to **Foreign_Language_Greeter**. Next, select **Categories:Attributes** and add one data member, location. This is of type C++ char (string).

ECE 264, **Object-Oriented Software Development**

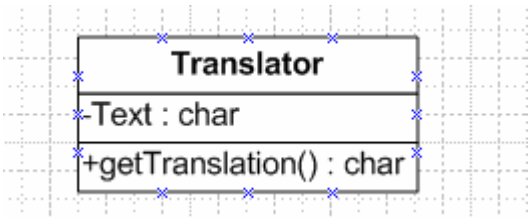




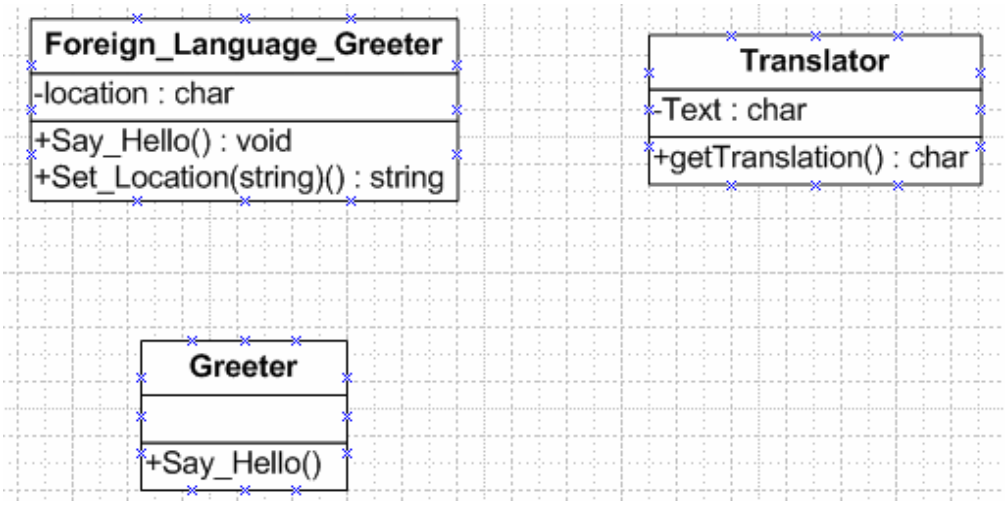
Now, select **Categories:Operations** and add a void function, **Say_Hello(),** and another function **Set_Location**.



Finally, change **Class3** to **Translator**, and add one method, **getTranslation()**, and one data member/attribute, **Text** using similar steps as shown for Foreign Language Greeter.



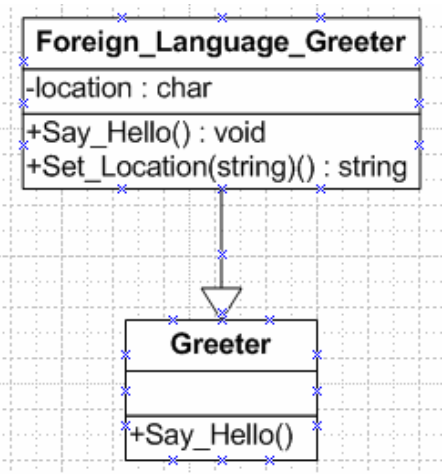
Notice that public members have a (+) to the left of the member name, and private members have a (–) next to the member name. Your diagram should now appear as shown below. Notice that the method **getTranslation()** returns a translated string.



Now, let's show the relationship between classes. **Foreign_Language_Greeter** is a type of Greeter. In fact, it **inherits** from Greeter. So, we use the **generalization** connector to indicate **inheritance**.

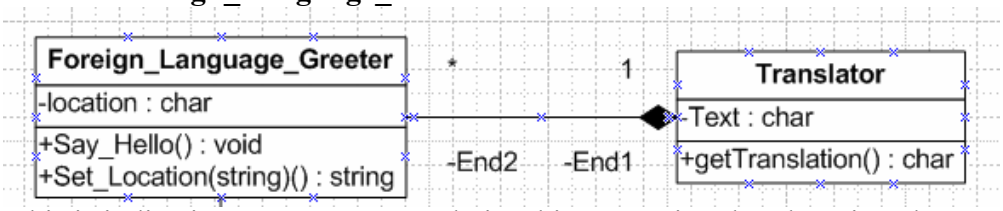
Connect both greeter classes as shown. The arrow always faces the base class.

ECE 264, Object-Oriented Software Development

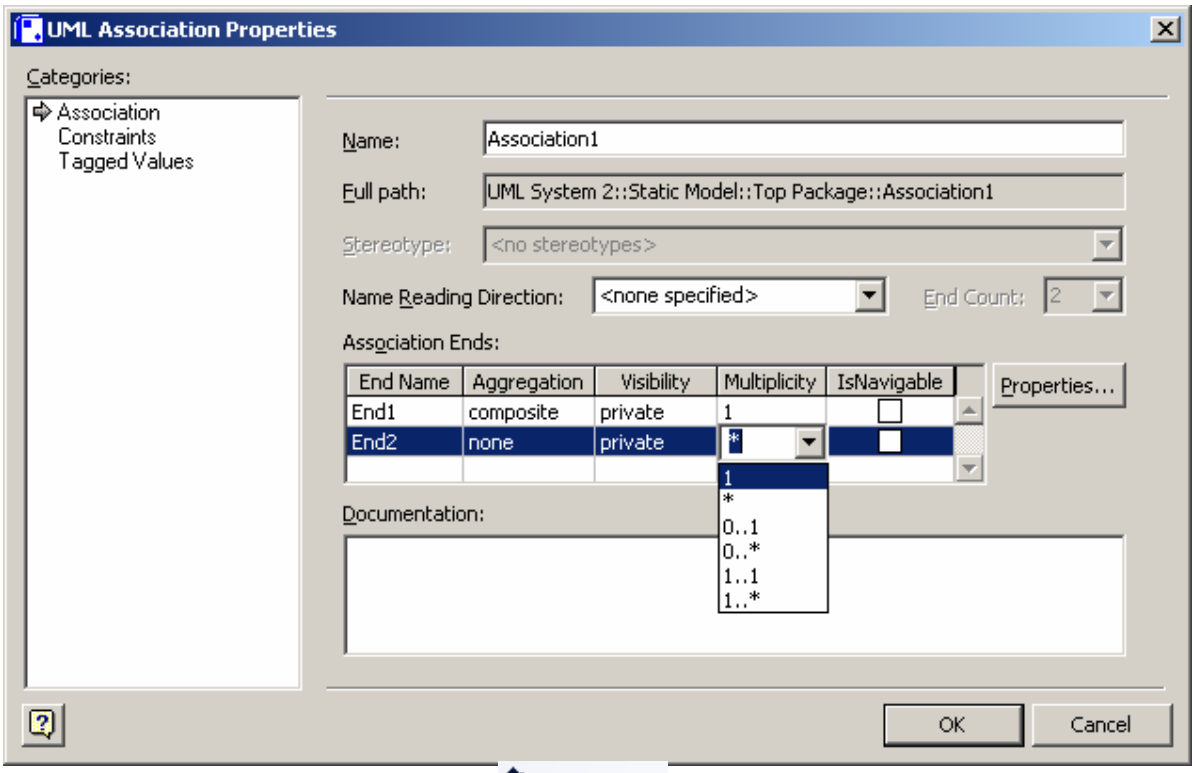


Foreign_Language_Greeter ***uses*** a Translator via **association**/***aggregation***. (Remember, this is when a class is declared as a data member to another class.) So, drag the ***Composition*** Arrow

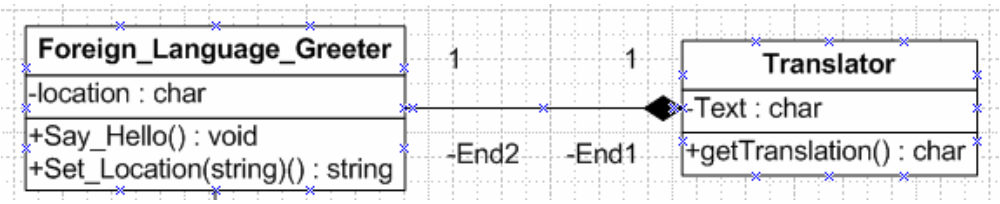
from **Foreign_Language_Greeter** to Translator.



Notice that this is indicating a many to one relationship, assuming that there is only ***one*** Translator for ***many*** Foreign_Language_Greeters. The many is indicated by the asterisk (*) , and the 1 is indicated by 1. In our case, there is only one translator for each Foreign_Language _Greeter, or a 1:1 relationship between each Translator and Greeter.

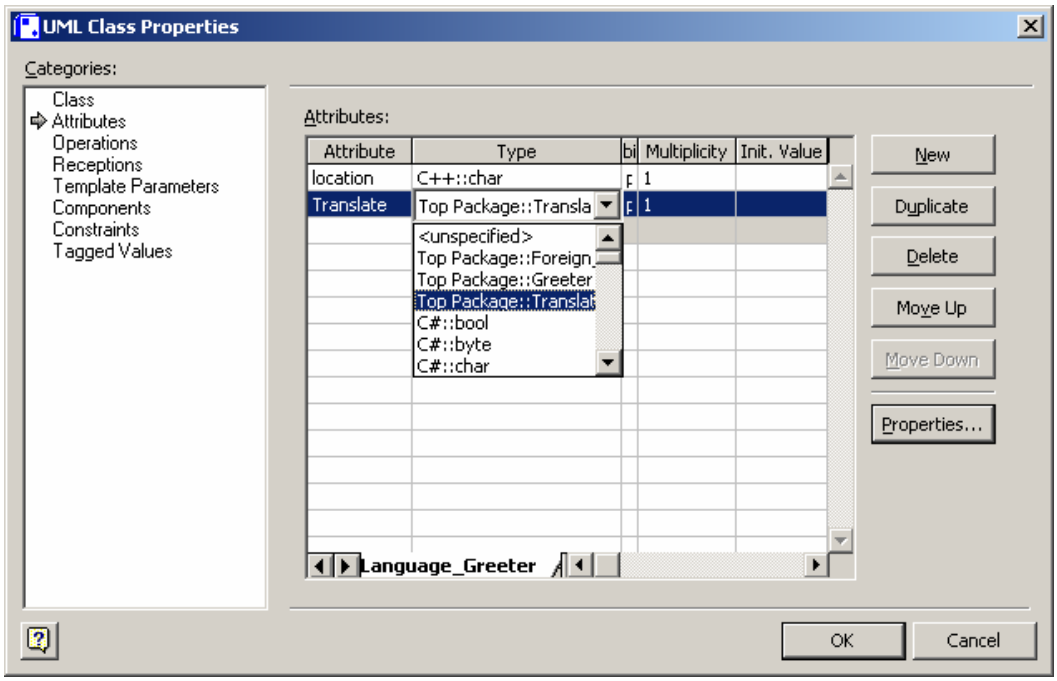


Double click on the **Composition arrow** to change the relationship. End 2 should be changed to 1, reflect a 1:1 relationship. Click OK to apply changes.

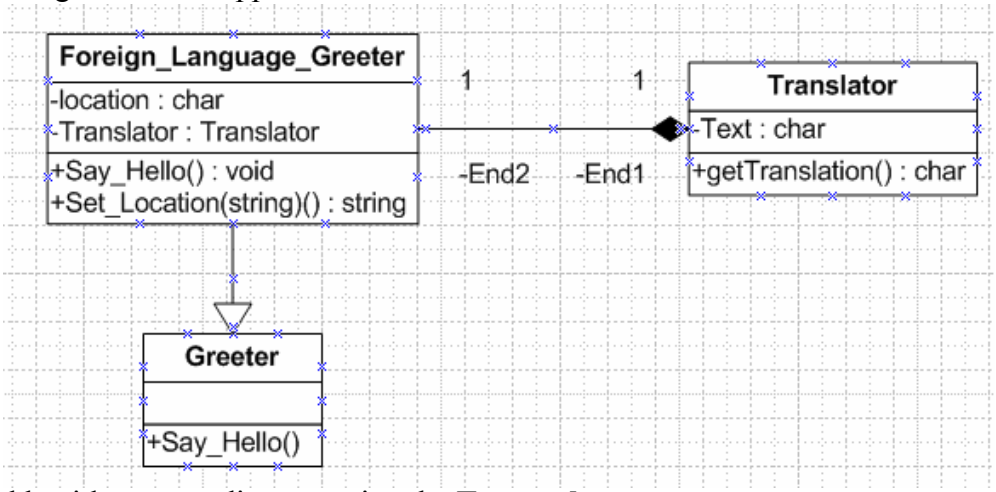


Next, we need to go back to **Foreign_Language_Translator** and modify it so that it uses a Translator called Translator. Double-click on **Foreign_Language_Translator**, and add the following under **Categories:Attributes**. Notice that now, Translator is also available as a data

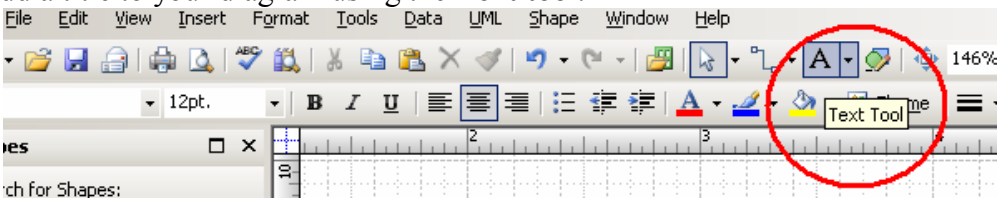
type. Visio™ automatically makes classes that you create available as data types in the form of **Top:Package::Your Classname**.



Now, your diagram should appear like this:



You can add a title to your diagram using the **Text tool**.



Click on the text tool, and then click where you would like to place your title.

Click anywhere else in the diagram to stop entering text. You can always double-click on text to change it.

Next, select the **Pointer Tool**, the drag the title to center it.

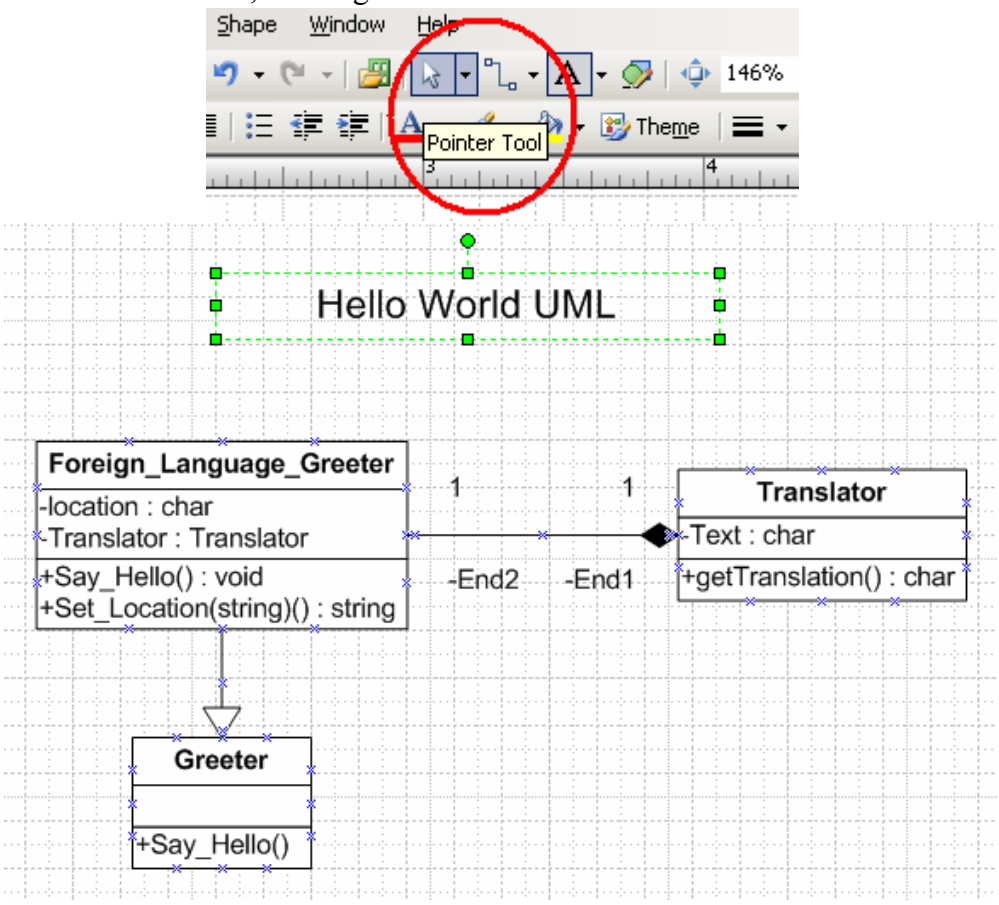


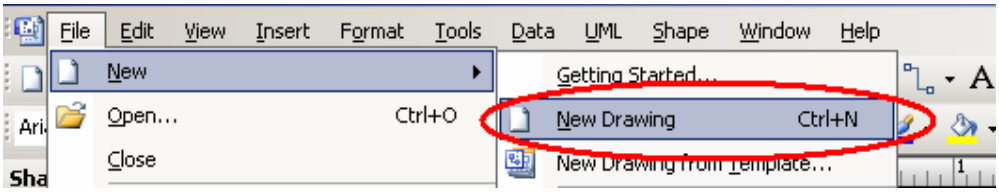
Figure 2: UML Class Diagram

Now, we are done, Select **File** **Save** to save your diagram.

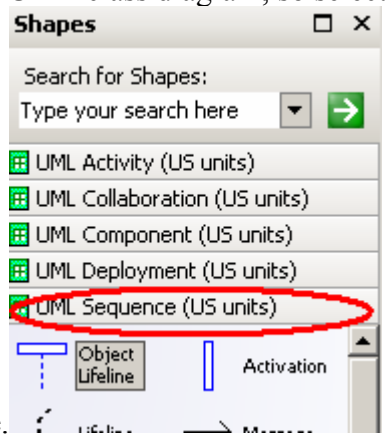
Creating a UML Sequence Diagram

Next, we will show how these three classes interact with each other to greet the user using a sequence diagram. A sequence diagram shows how the program flows between objects in order to complete a task. In short, it is similar to a flowchart involving classes and the instances of these classes; objects. It also uses numbers to indicate program sequence.

Click, **File** **New Drawing** to start another UML diagram.

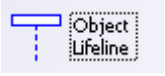


This time, we are creating a UML class diagram, so select the **UML Sequence** palette.



Since sequence diagrams show the interaction between classes/objects, we need to place a symbol for each class on the diagram. There are three classes, **Greeter**, **Foreign_Language_Greeter**, and **Translator**.

Following are some symbols, and an explanation of each:



Symbolizes a Class/Object we have 3 classes, so we need 3 Object lifelines.

This is used to indicate an event.

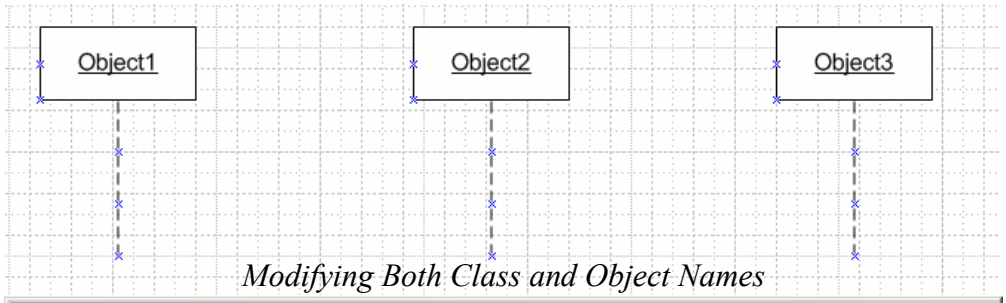
Messages are used to connect events (what happens next?)

Good software practice to provide notes.

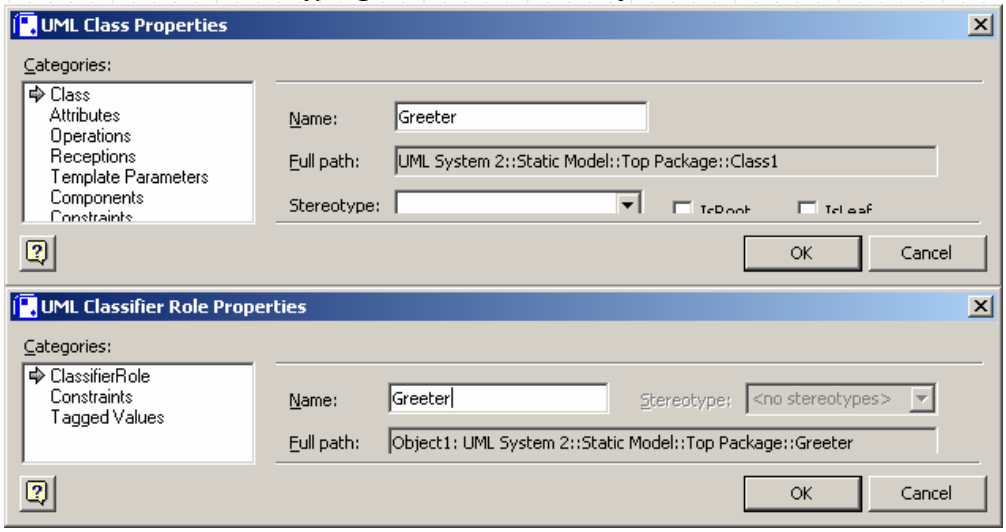
Draw the following diagram. Double click on **Object1**, to change both the object (instance) and class name. For simplicity, name both the object and class name **Greeter**.

ECE 264, **Object-Oriented Software Development**

All Three Objects/Classes

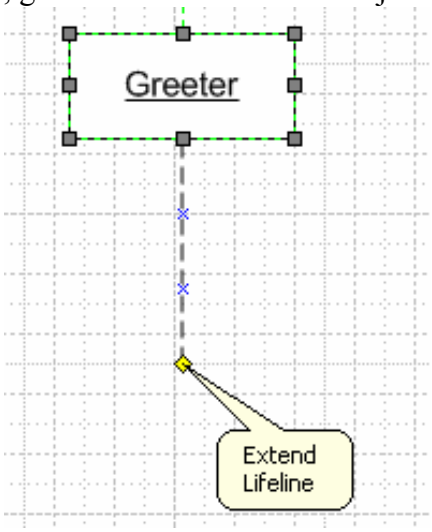


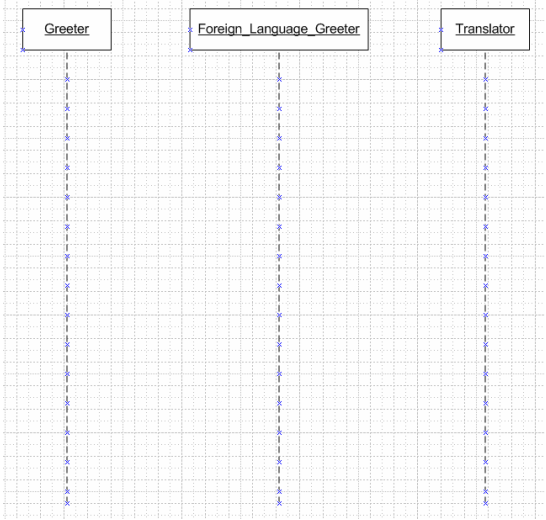
Modifying Both Class and Object Names



Name Objects 2 and 3 as shown. (For the other 2 classes).

Now, go to the bottom of each Object lifeline, and drag the line down.





Now, complete the following diagram. You will have to get the user

symbol from the UML Use-Case

Palette in Visio ™.

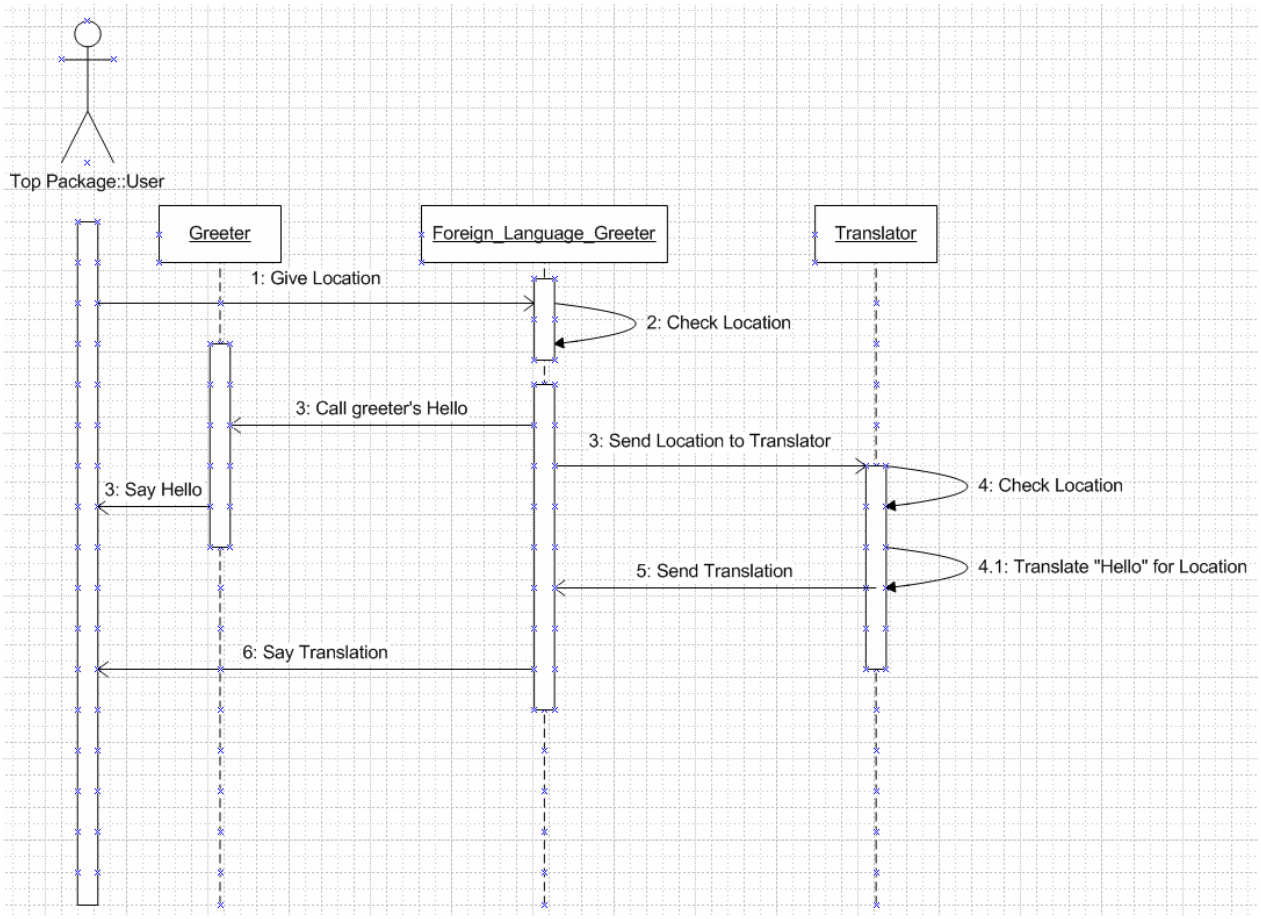


Figure 3: UML Sequence Diagram

When you double-click on a message, you can add text. The number before each message indicates sequence. For example, **1:Give Location** comes before **2:Check location**. Give location is from the user to the Foreign Language Greeter. Depending upon the location, **Foreign_Language _Greeter** will either call Greeter's English hello (3: Call greeter's hello), or Send the location to the translator (3: Send location to translator.)

Each one of the

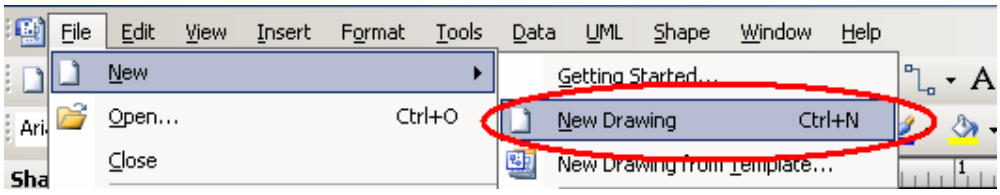
boxes represents a function. Notice that **Foreign_Language_Greeter** has two activation boxes since it has two functions, **setLocation()** *(2: Check Location),* and **Say_Hello()** *(3: Send Location to Translator and 5: Send Translation).* Sequences within the same function such as Translator's getTranslation have the form **n1.n2**, where n1 and n2 are both numbers. For example instead of 4, and 5, seqences within getTranslation are labeled **4.1** and **4.2** to indicate the same function.

ECE 264, Object-Oriented Software Development

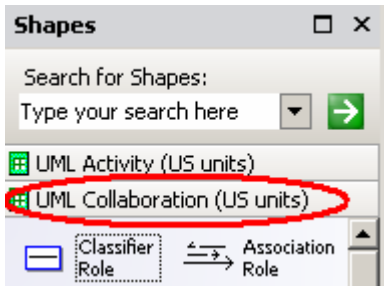
Creating a UML Communication Diagram

The UML sequence diagram is useful for showing how the classes communicate in time, but what about the bigger picture? Or, what if we are unsure about the exact order of events? A UML Communication diagram shows the messages passed between calls in order to accomplish a task with an emphasis on topology, not time. This means, that it will indicate the relationship between classes, or functions called while they accomplish a task.

Click, **File** **New Drawing** to start another UML diagram.

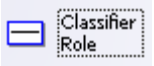


This time, we are creating a UML Collaboration diagram, so select the **UML Collaboration** palette.



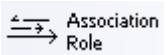
Since we have already created a UML Sequence diagram showing how the classes will communicate in sequence, we will need to translate these messages into actual function calls. This is since we're more interested in the structure of the communication than the exact order. For example, Foreign_Language_Greeter's **2: Check Location**, becomes **setLocation()**, since the **setLocation()** function both set's the user's location and checks whether it is in the US.

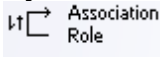
Following are some symbols, and an explanation of each:



Symbolizes a Class/Object.

Symbolizes Multiple Classes/Objects.





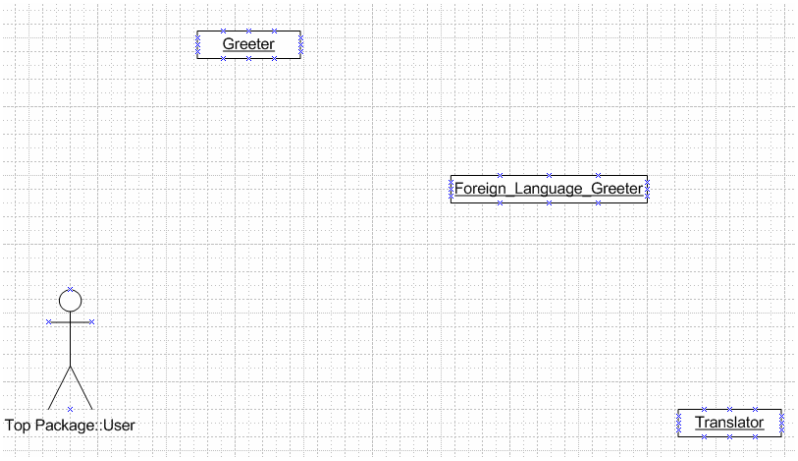
Messages are used to connect events (what happens next?)

Good software practice to provide notes.

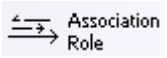
Using

Draw the following diagram. You will have to get the **user** symbol from the **UML Use Case**

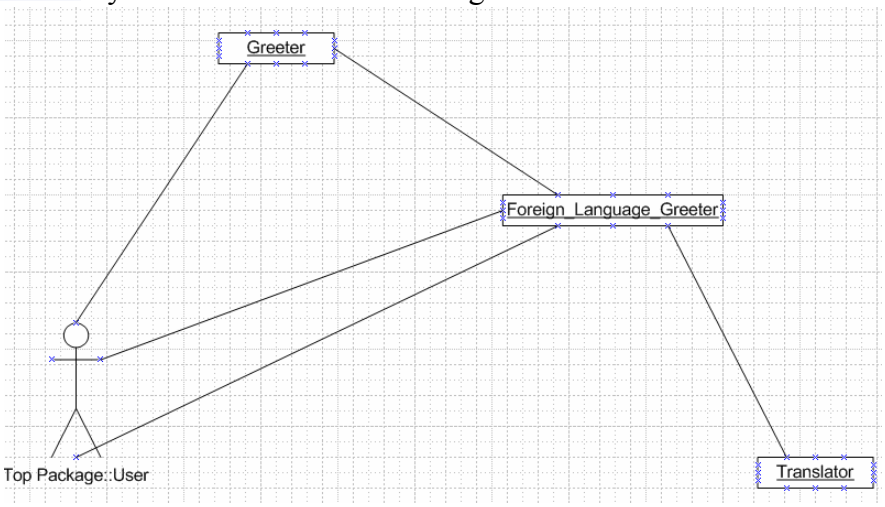
Palette.



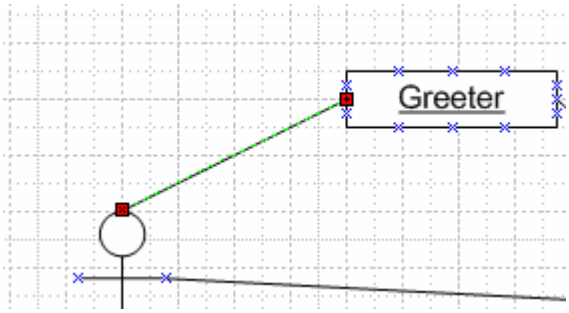
Now, we need to translate the messages in the sequence diagram into function calls. But first, we need to show the function call flow. This is the same as the message flow between objects in the sequence diagram.

Use the 

symbol to create the following:

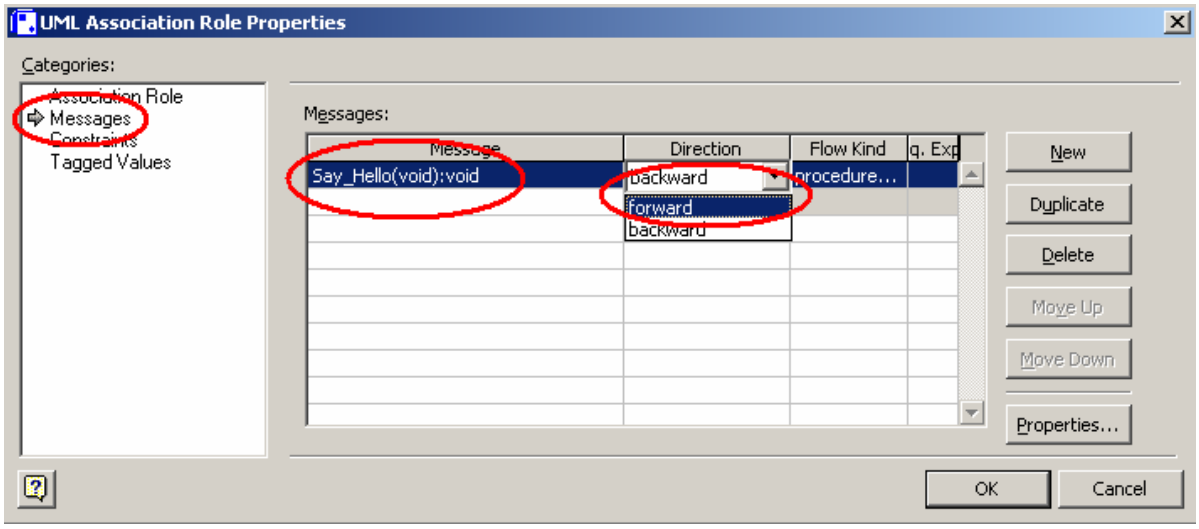


Next, we will indicate direction and functions. Double click on the arrow connecting the User's head, to Greeter.



Since the **Greeter** communicates with the user via **Say_Hello(),** lets modify the link so that it points from Greeter to the user, and specifies the function **Say_Hello()**.

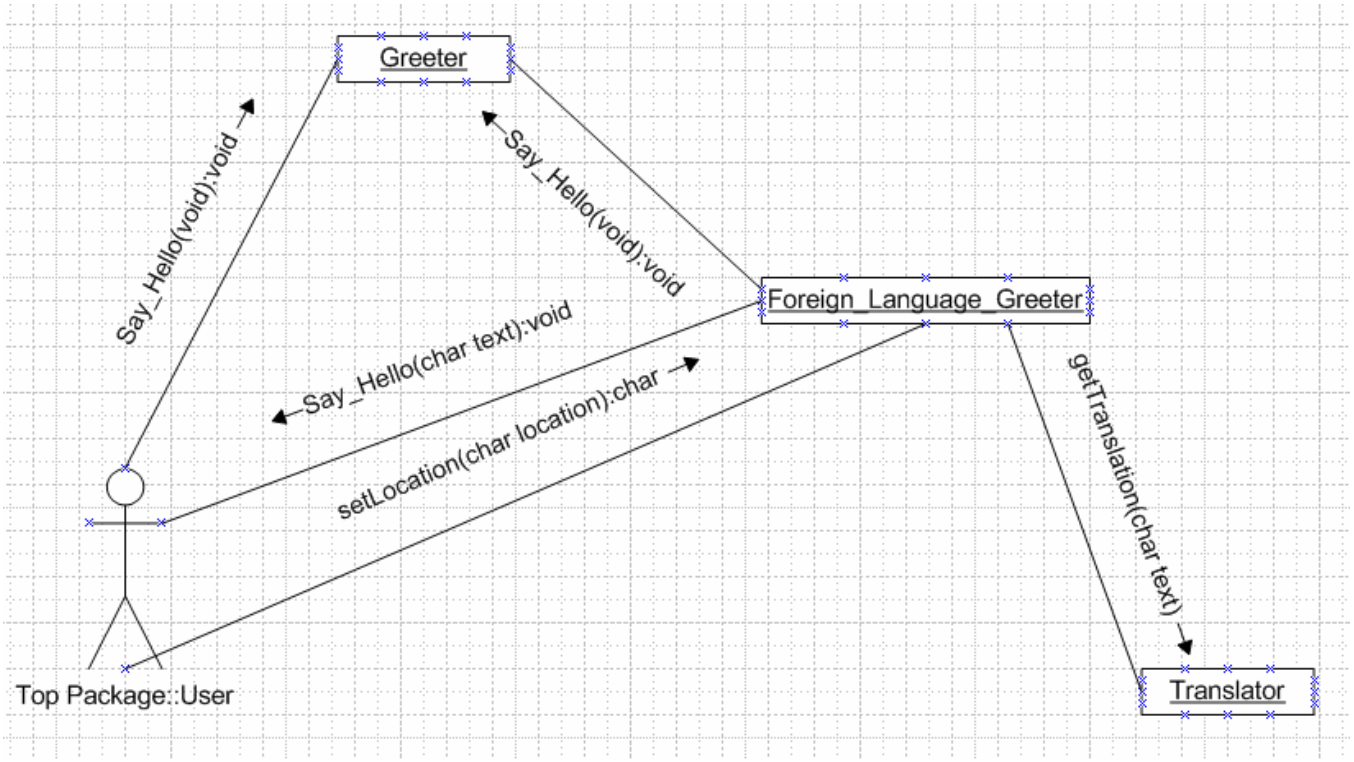
Under **Categories:Messages**, enter **Say_Hello(void): void** as the message name, and leave the direction forward. This indicates that the User ***uses*** Greeter's **Say_Hello()** function to be greeted in English.

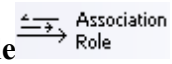


Now, we have an arrow indicating that the Greeter greets the user using its **Say_Hello** function.

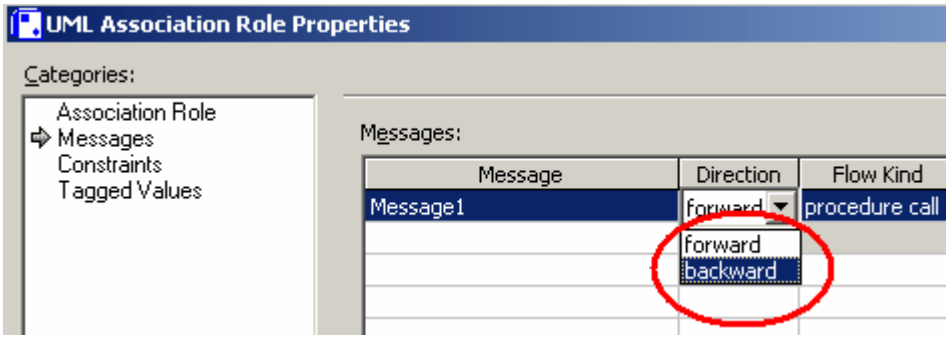
ECE 264, Object-Oriented Software Development

Repeat similar steps to complete the diagram:



You may have to change the direction of the **Association Role** 

arrows. You can change the direction by double-clicking on the arrow and selecting **backward/forward** under **Categories:Messages.**

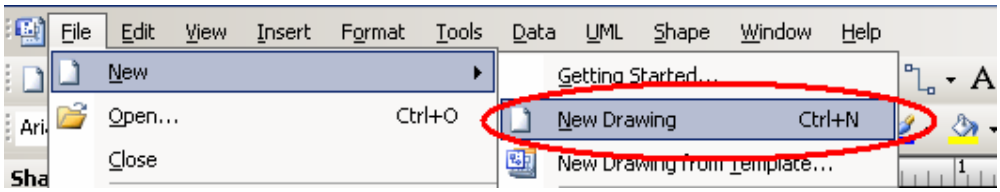


Also note that the user uses Two **Say_Hello**() s. The Greeter's **Say_Hello(void):void** takes no arguments and returns void since it merely says Hello in English. The Foreign_language_Greeter's **Say_Hello(char text):void** takes an argument text, since it Says whatever translation text was given to it by the Translator's getTranslation() function. getTranslation returns a translation called **Text**, which is then sent to the user.

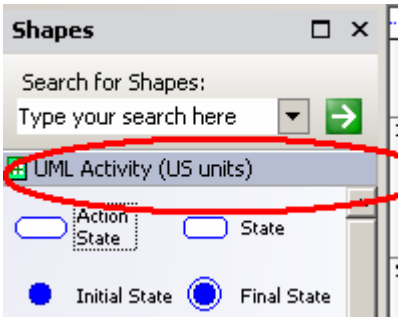
Creating a UML Activity Diagram (Flowchart)

Next, we will create a flowchart for the **Say_Hello()** method in **Foreign_Language_Greeter**.

As before, click, **File** **New Drawing** to start another UML diagram.



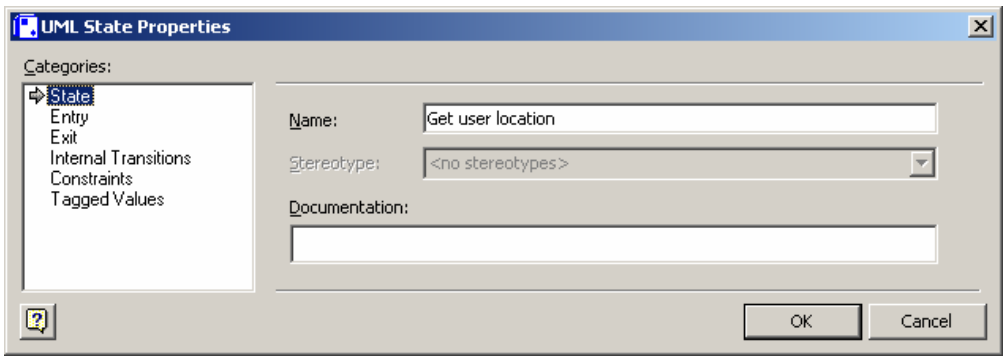
This time, we are creating a UML Activity Diagram (flowchart), so select the **UML Activity** palette.



Next, drag the following symbols onto the diagram.

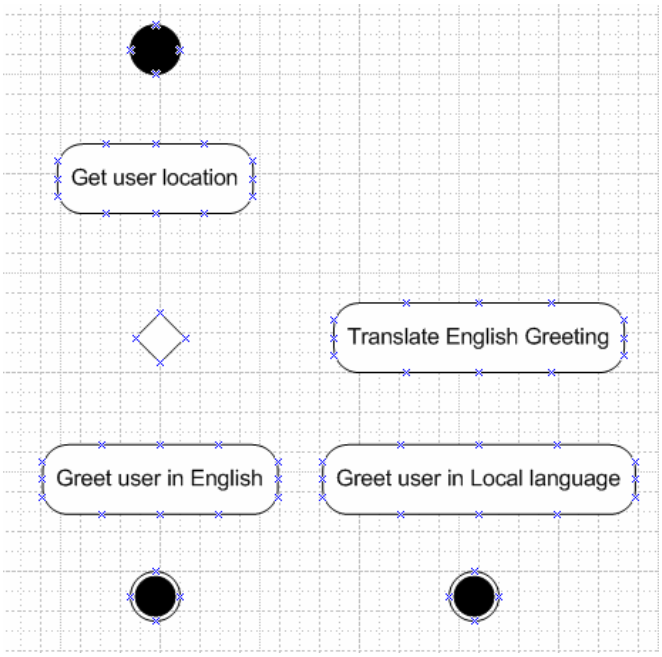
Notice that the rounded rectangles represent states, the dark circle represents the initial program state (beginning of program), and the outline circle represents the final state (end of program). The diamond represents a conditional (if) statement.

Just like before, you can change the names of boxes by double clicking on them. For example, double-click on state 1. The program starts by getting the user's location, so the name of this state will be **get user location**. Click **OK** to apply changes.

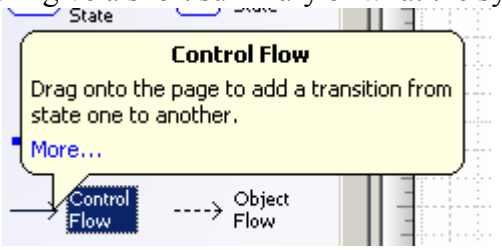


ECE 264, Object-Oriented Software Development

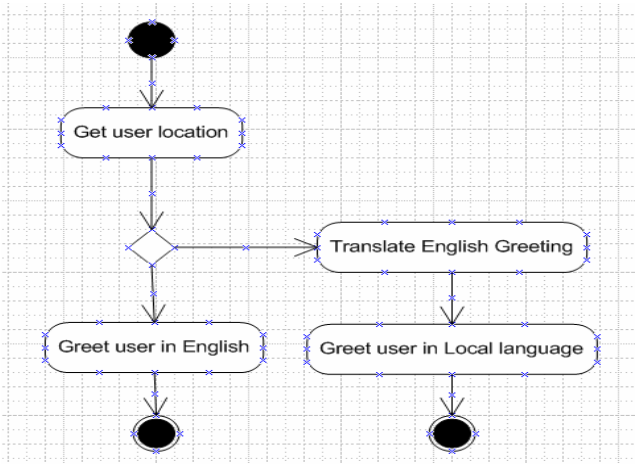
Modify the diagram to look like this:



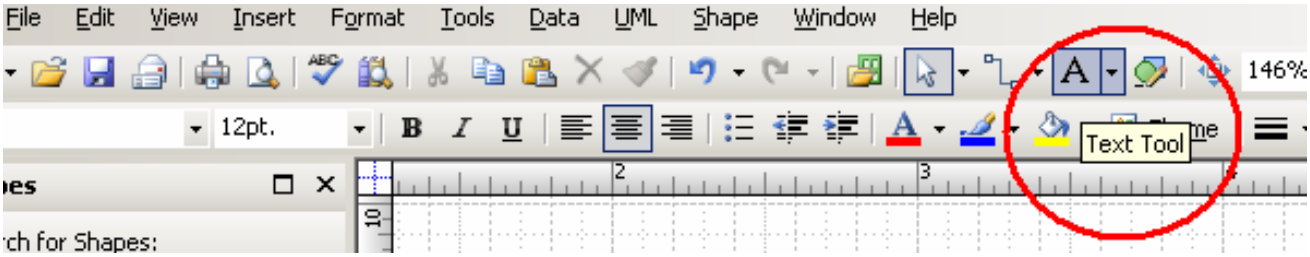
Next, connect the points on the flowchart. Usually, you will use the Control Flow Arrow. If you mouse-over a symbol, Visio will give a short summary of what the symbol represents.

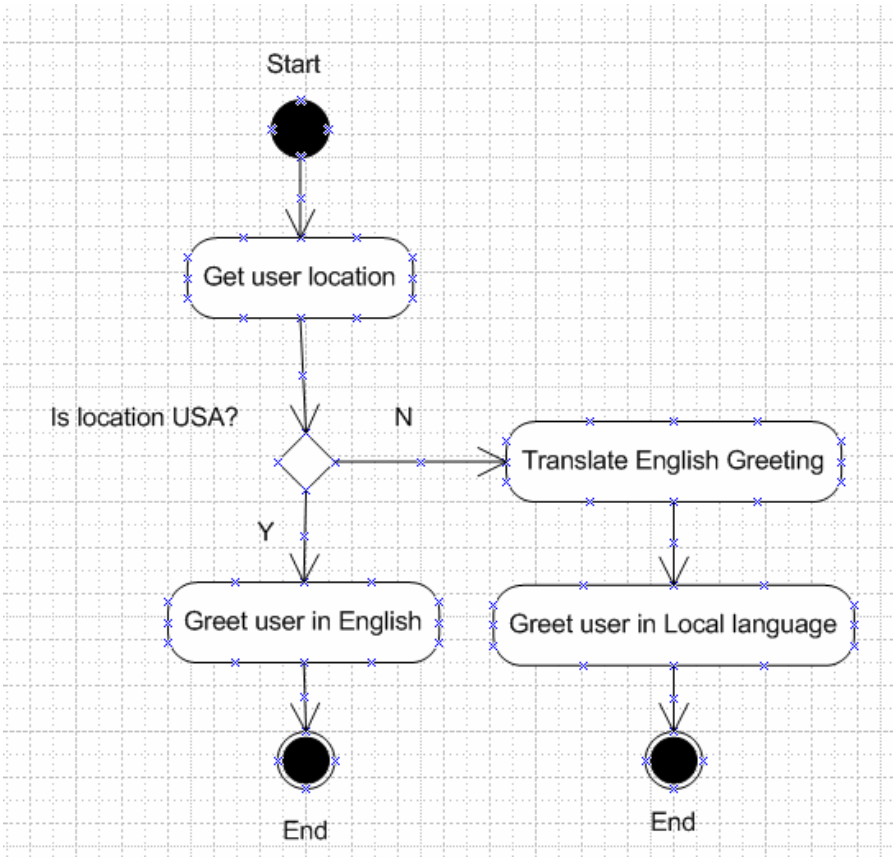


Modify the diagram as shown.

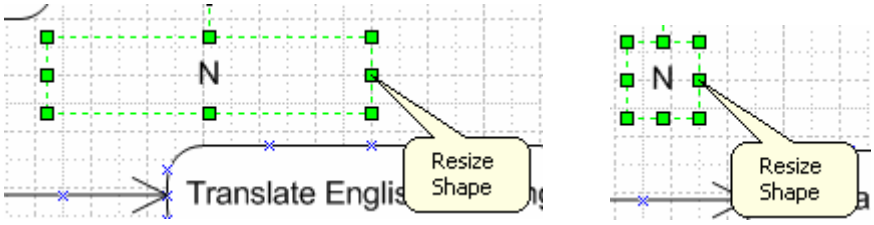


Next, we need to indicate what the conditional (if) statement decides. Use the text tool. To add text next to the conditional statement as shown:





You may have to click on one of the green boxes surrounding the text and drag it to resize the textbox.



ECE 264, Object-Oriented Software Development

It is good programming practice to have only one entry and one exit point for the program.

However, we have two exit points. In order to correct this, use the Transition Join

symbol to connect the endpoints as shown below:

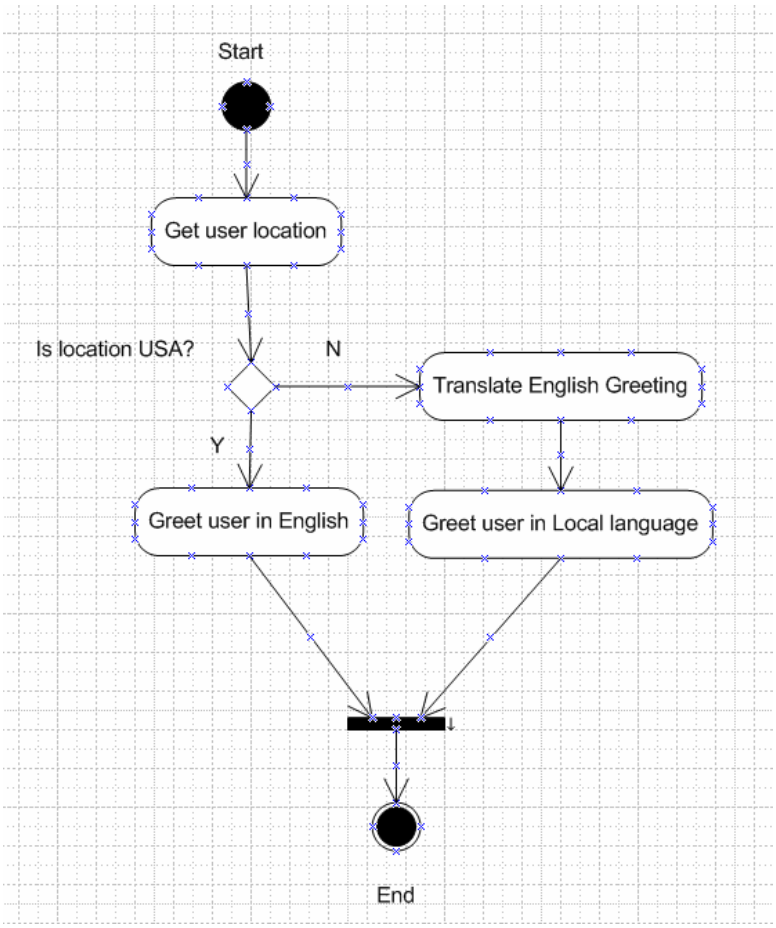
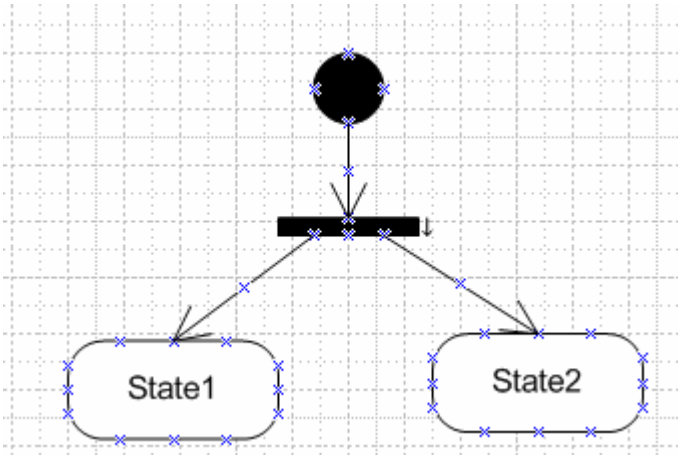


Figure 4: UML Communication Diagram

Likewise, you can use the

for branches:



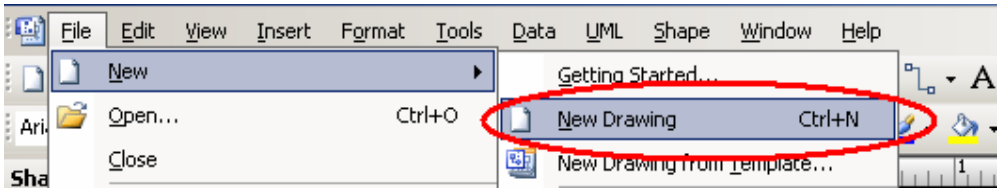
Next, you can add a title if you choose, then select **File** **Save** to save your work.

ECE 264, Object-Oriented Software Development

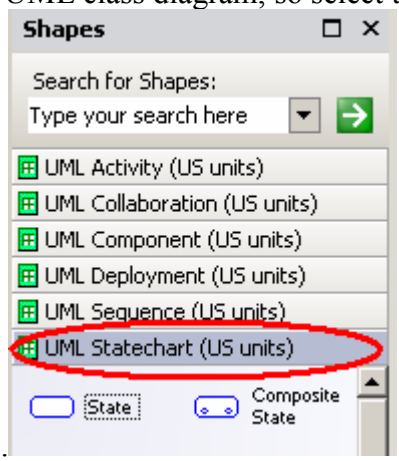
Creating a UML State Diagram

Next, we will create a UML State Diagram for the **Foreign_Language_Greeter** class. This is very similar to the Activity Diagram/Flowchart shown in the previous section, but instead of describing the state of a function, it describes the behavior of entire class. A state diagram shows the state of an object, as it is being used. In short, it is similar to a flowchart for a class.

Click, **File** **New Drawing** to start another UML diagram.



This time, we are creating a UML class diagram, so select the **UML Statechart** palette.



This opens the **UML Statechart** palette (for UML State Diagrams) .

Notice that the palette is very similar to the UML Activity palette that we worked with previously.

Given this fact, let's look at the state diagram for Foreign_Language_Greeter, and examine the differences between this diagram, and the previous section's Activity diagram.

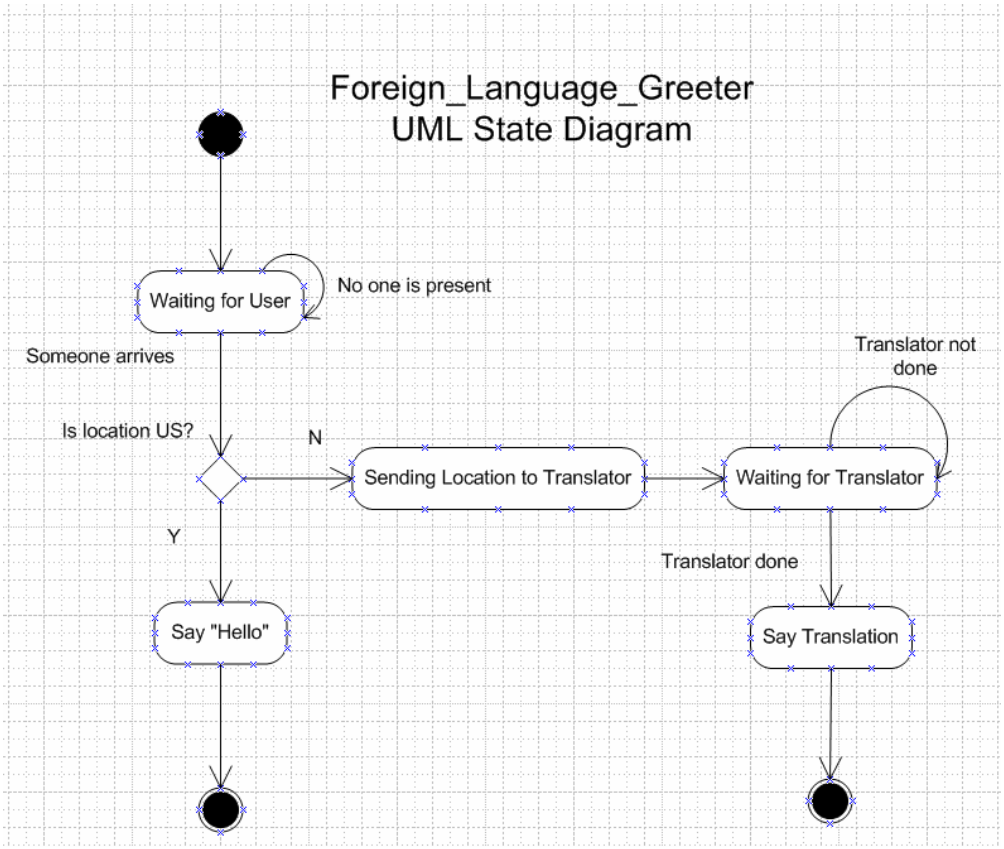
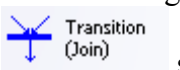


Figure 5: UML State Diagram

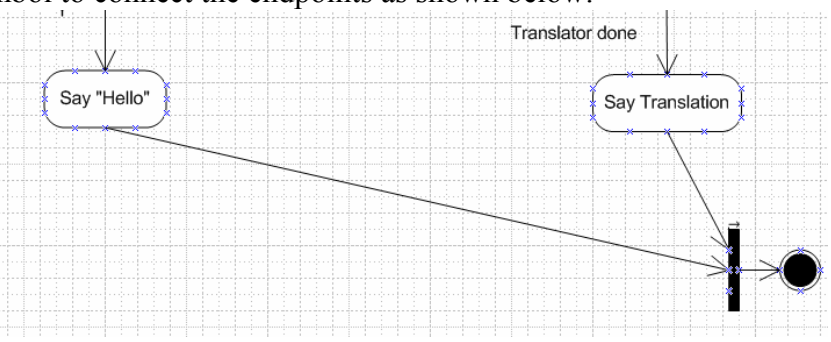
The **Foreign_Language_Greeter** first waits for the user, and continues waiting if no one is present. As soon as someone arrives, it checks whether their location is within the US and if not, then it sends the location to the translator, and waits for a translation. The **Foreign_Language_Greeter** then ends by either saying hello in English (US), or using the Translator's greeting.

The difference in this diagram is that the states *"Sending Location to Translator",* and *"Waiting for translator"* implies that the Foreign Language greeter uses a Translator, and that translator is a separate entity/object. This is much different than in the state was "Translating Greeting," which would imply that the **Foreign_Language_Gree**ter has a translation function.

Notice that again, we have two endpoints. In order to correct this, use the Transition Join



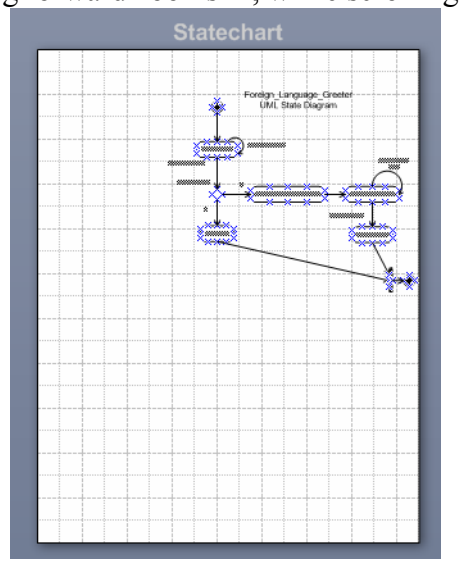
symbol to connect the endpoints as shown below:

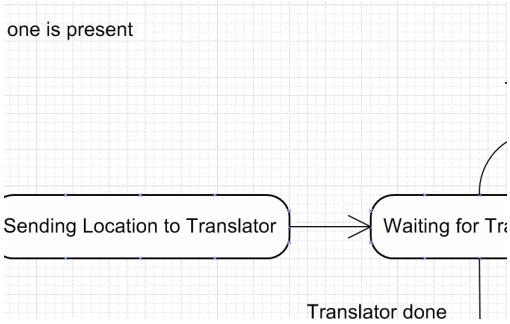


ECE 264, **Object-Oriented Software Development**

Visio™ Tips and Shortcuts

To Zoom In/Out: Hold the Ctrl key, and use your middle mouse button to zoom in/out. Scrolling forward zooms in, while scrolling backward zooms out.





Now you have the tools necessary to start using Visio™ to create UML diagrams for your team project. Good luck!