Constructing sequence diagram with existing classes

Written Date : June 14, 2010

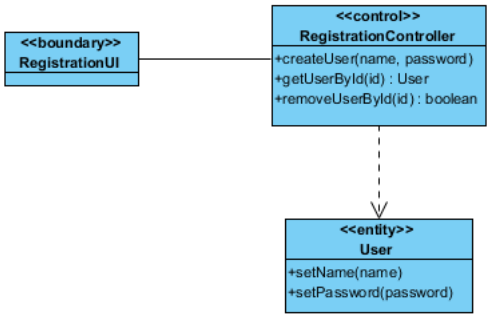
When you want to model the structure of a system or an application, you can make use of class diagram. When you want to model the interaction between objects in runtime, with the sequence of method invocation, you can make use of sequence diagram. Class diagram and sequence diagram can be related with each other. While a class in class diagram represents a blueprint of data, a lifeline in sequence diagram represents an instance of such blueprint. In this tutorial, we will start from a simple class diagram, and make use of a sequence diagram to model the dynamic method invocation related to a controller class modeled in class diagram.

---

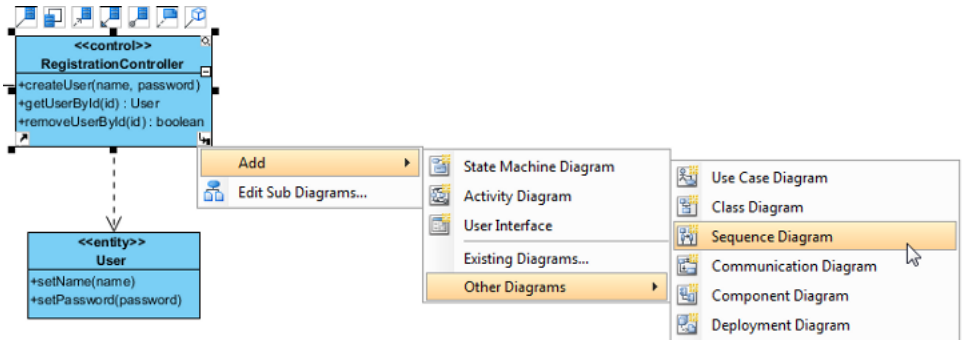
1. Download the project file *Simple-Registration.vpp* attached with this tutorial.

2. Start VP-UML in a new workspace.

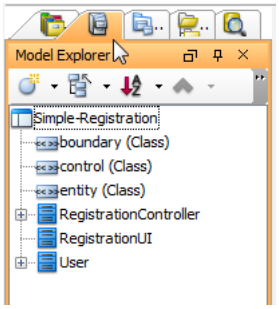
3. Open *Simple-Registration.vpp* and open class diagram *Registration*. Study the diagram content. We have three classes - *RegistrationUI*, *RegistrationController* and *User*.



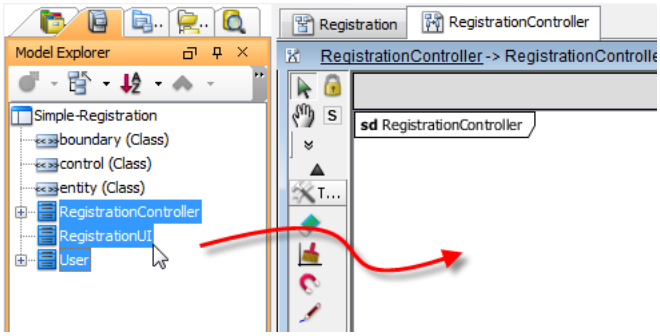
4. Now, we want to model the interaction between object instances of these classes in runtime. Since the controller class is responsible in controlling the registration process, add a sub-sequence diagram from it. Move the mouse pointer to *RegistrationController*. Click on the resource icon at the bottom right corner and select **Add** > **Other Diagrams** > **Sequence Diagram** from the popup menu.



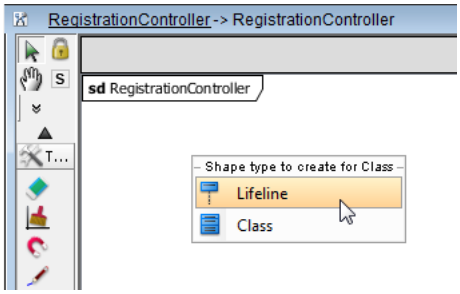
5. This creates an empty sequence diagram. Open the **Model Explorer**, which is the tab next to **Diagram Navigator**. You can see the three classes listed there.



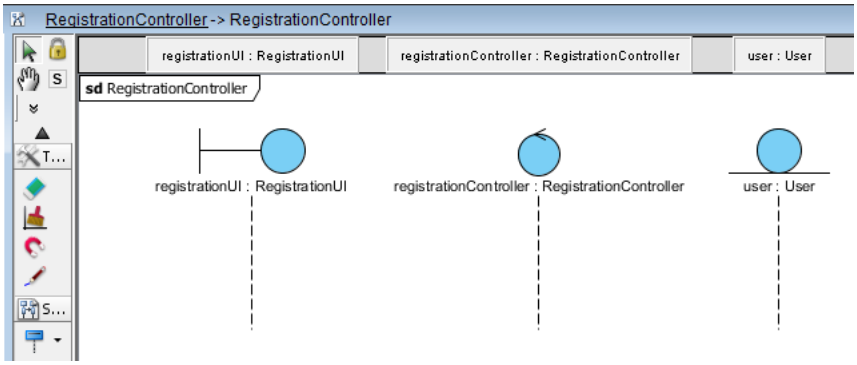
6. We want to make use of the classes in creating lifelines in sequence diagram. Select the three classes in **Model Explorer** and drag them onto the sequence diagram.



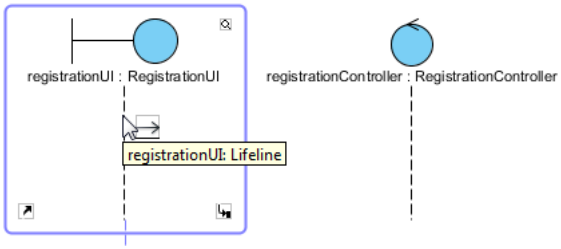
7. Select **Lifeline** in the popup menu. If you select **Class**, it will create class shapes in sequence diagram.



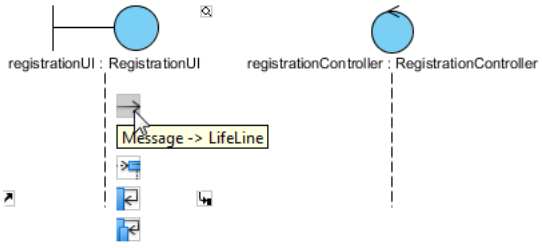
By selecting **Lifeline**, lifelines will be created, with the three classes set as classifiers.



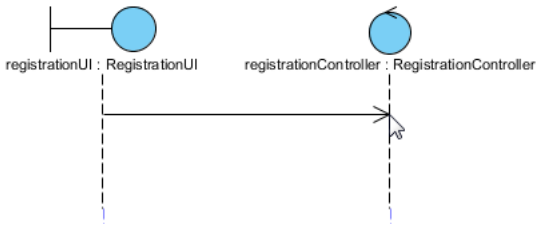
8. Let's model the method invocations between lifelines. Move the mouse pointer over lifeline *registrationUI*.



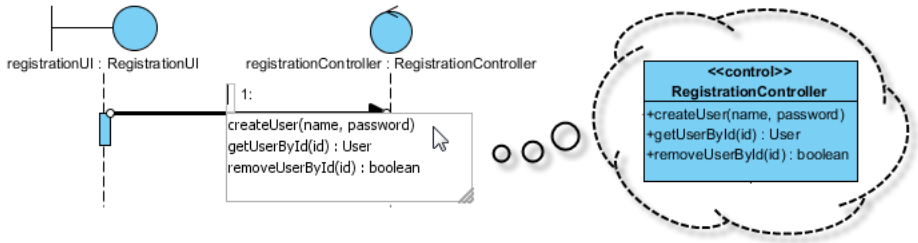
9. Press on the resource icon **Message -> Message**.



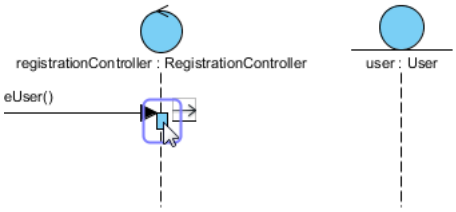
10. Move to lifeline *registrationController* and release the mouse button.



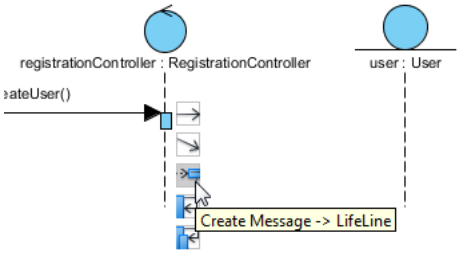
11. This pops up a list of name that you can choose for the new sequence message. You can see that those are operation of class *RegistrationController*. Select *createUser(name, password)*.



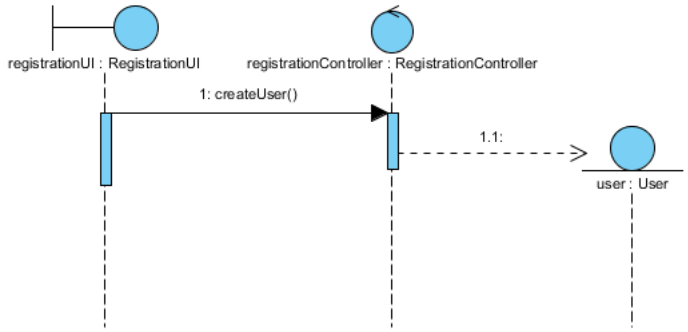
12. Relate lifeline *registrationController* and user. We say that *registrationController* creates the user lifeline. Therefore, we need to relate them with a create message. Move the mouse pointer over the activation in lifeline *registrationController*.



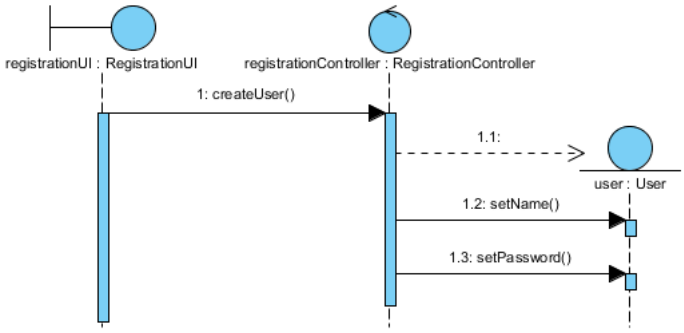
13. Select the resource icon Create *Message -> Lifeline*.



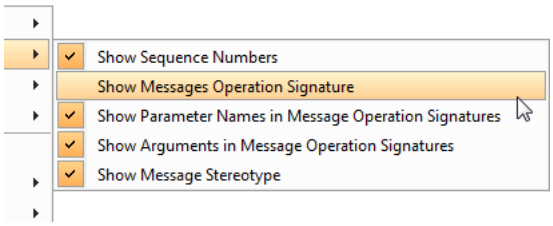
14. Move to lifeline user and release the mouse button.



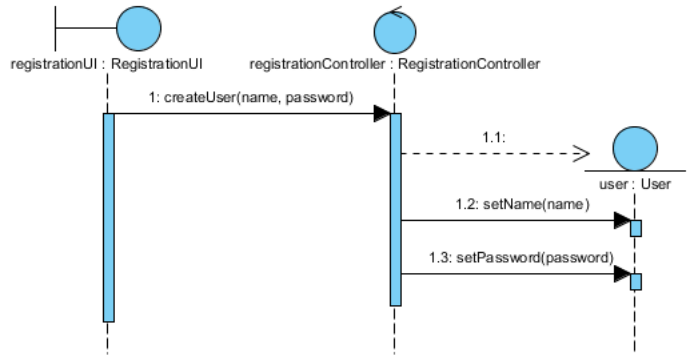
15. Create messages *setName* and *setPassword* from lifeline *registrationController* to user. Up to now, the diagram become:



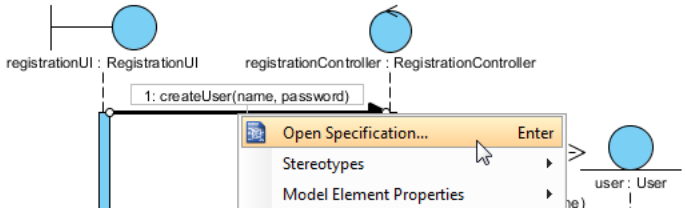
16. The operations' parameters are not presented by default. To show them, right click on the background of diagram and select **Presentation Options** > **Message Display Options** > **Show Messages Operation Signature** from the popup menu.



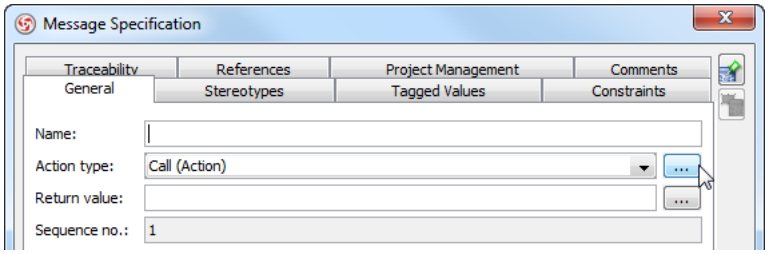
The parameters are then presented.



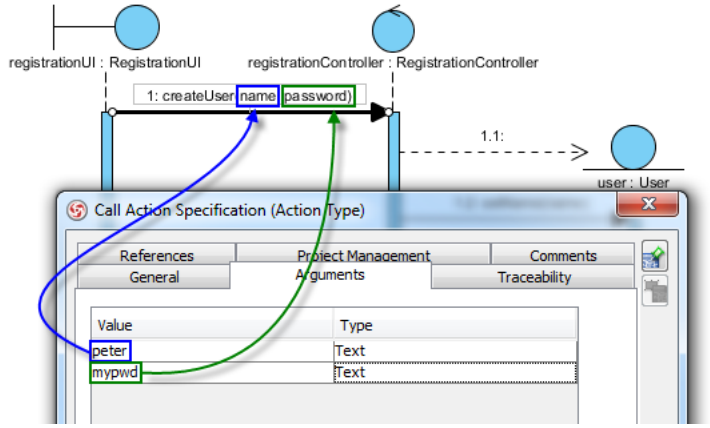
17. We can specify the arguments as well. Take the message *createUser(name, password)* as example. Right click on it and select **Open Specification...** from the popup menu.



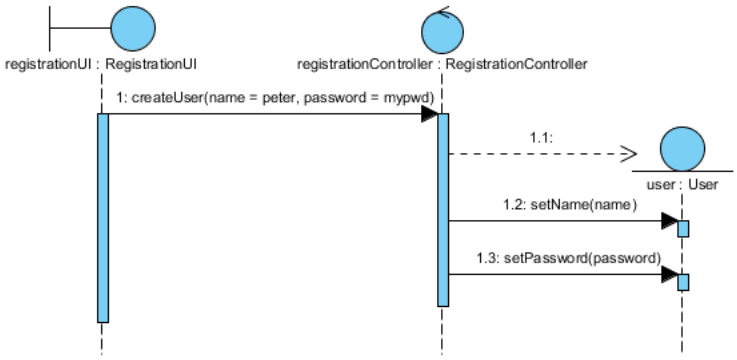
18. Edit the action type property by clicking on the button with dotted caption, next to **Action type**.



19. In the **Call Action Specification** dialog box, click **Add** to add an argument. In this example, click **Add** to add argument *peter*. Click **Add** again to add argument *mypwd*. Note that the two arguments are actually referring to the two parameters given by the operation. If you add the third argument here, it will be ignored (as there are only 2 operations defined).



20. Click **OK** to close the dialog boxes and go back to diagram. The arguments are added and presented on diagram. Finally the diagram become:



Visual Paradigm for UML home page
(http://www.visual-paradigm.com/product/vpuml/)

UML tutorials
(http://www.visual-paradigm.com/product/vpuml/tutorials.jsp)

Visual Paradigm home page
(http://www.visual-paradigm.com/)