Getting started with UML on Bridgepoint

Introduction

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

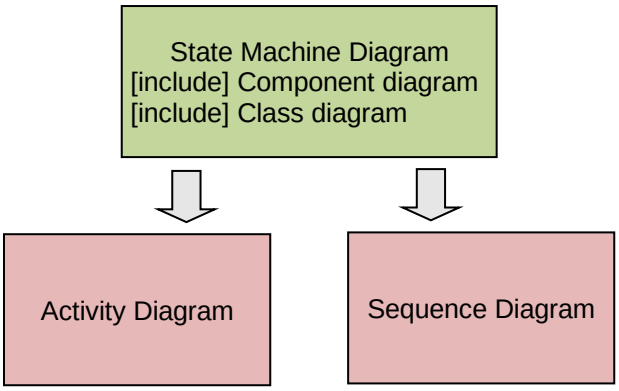
Objectives

After completing this lab, you will be able to know:

- What State Machine Diagram is included Component diagram and Class diagram.
- How to develop a State Machine diagram on Bridgepoint.
- How to draw an Activity diagram and Sequence diagram on Bridgepoint.

Procedure

This lab is separated into steps that consist of Bridgepoint programming for State Machine included the creation of a diagram for any element of Component, Class and attributes. Follow these detailed instructions to progress through the lab. After you complete this lab. You can design a State Machine diagram to control your space ship in EE-space wars game. We will find a winner by using your designed State Machine diagram to compete in a tournament or survival modes. After finishing your designed diagram. You need to draw an Activity diagram and Sequence diagram to describe how your State Machine works.



General Flow for this Lab

Bridgepoint programming

Action Language, OAL

xtUML supports modelling data, processing and control. Data and control are modelled with graphical diagrams such as component, class, and state machine diagrams. Processing is modelled with an action language called *Object Action Language* (OAL). OAL is similar in many ways to target programming languages such as Java, C++, Python or VHDL. However, it attempts to be:

- simple
- abstract
- model-aware
- translatable

OAL is simpler than most programming languages. It is meant to be somewhat minimal in its notation while being rich enough to model all necessary processing. Object Action Language is abstract and works at the level of detail of the model in which it operates. The instructions in the action language are aware of the model of which it is a part in both its syntax and its meaning. The names of model elements such as classes, attributes, events, messages, ports and parameters are parsed in the action language. Finally, the action language is target independent and can be translated into target-specific languages by a model compiler.

Operators

- Arithmetic: +, -, *, /, %
- Boolean: and, or
- Logical: ==, !=, <, <=, >, >=

Data Types

- Implicit Typing: All data items are implicitly typed by the value assigned to them on their first use within an action.
- Simple Data Types: Integer, Real, String, Boolean
- System Data Types: Date, Timestamp, Unique ID
- Reference Types: Timer Handle, Instance Handle, Instance Handle Set, Event Instance, Component Handle

Expressions

- a = 3; /* integer typed local variable */
- assign x = 3.14; /* floating point value (real) */
- y = 11.0; /* another real */
- done = false; // Boolean typed local variable
- z = x + y * x; /* Operator Precedence */
- b = a % 2; /* remainder operator */
- s1 = “Hello”; /* String Variable – dynamic size */
- s2 = “World!”; // C++ Comments also allowed
- s3 = s1 + “ “ + s2; // String Concatenation

IF Statement

```
if (serialNumber > 1000)
    // do something
elif (serialNumber > 2000)
    // do something
else
    // or something
end if;
```

Loops

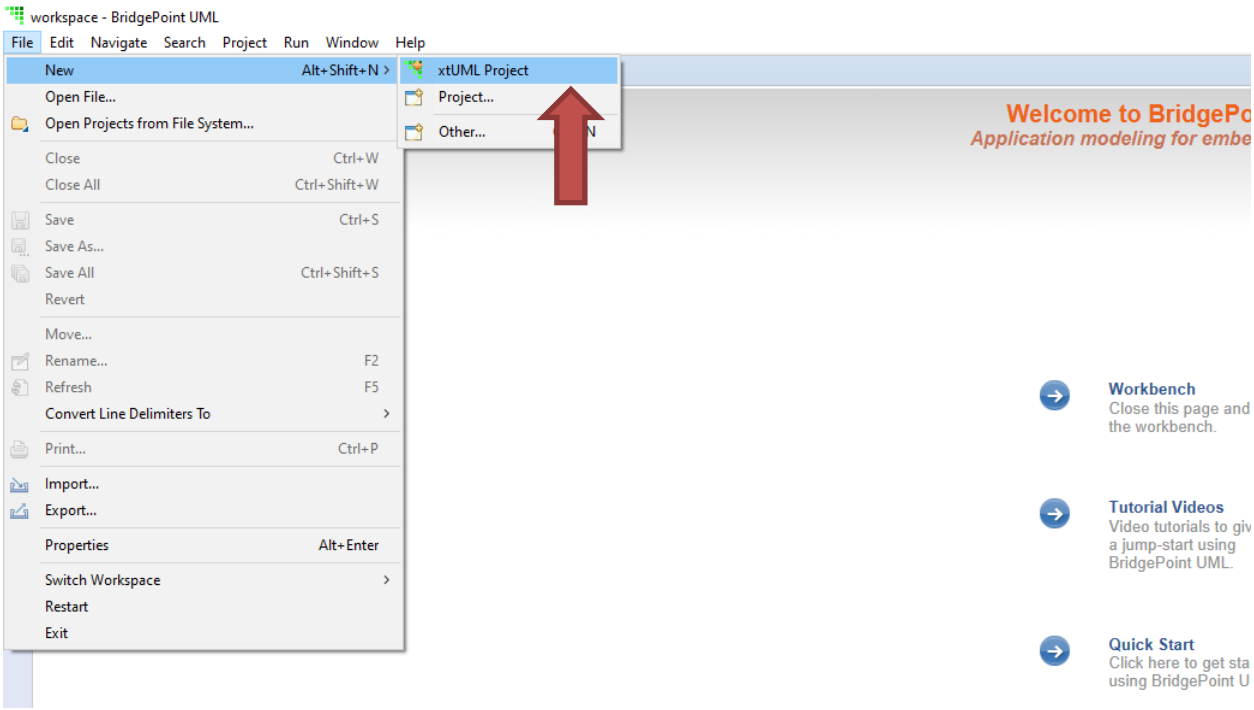
```
for each mobile in mobiles
    // do something
end for;

i = 0;
while (i < 4)
    // do something
    i = i + 1;
end while;
```

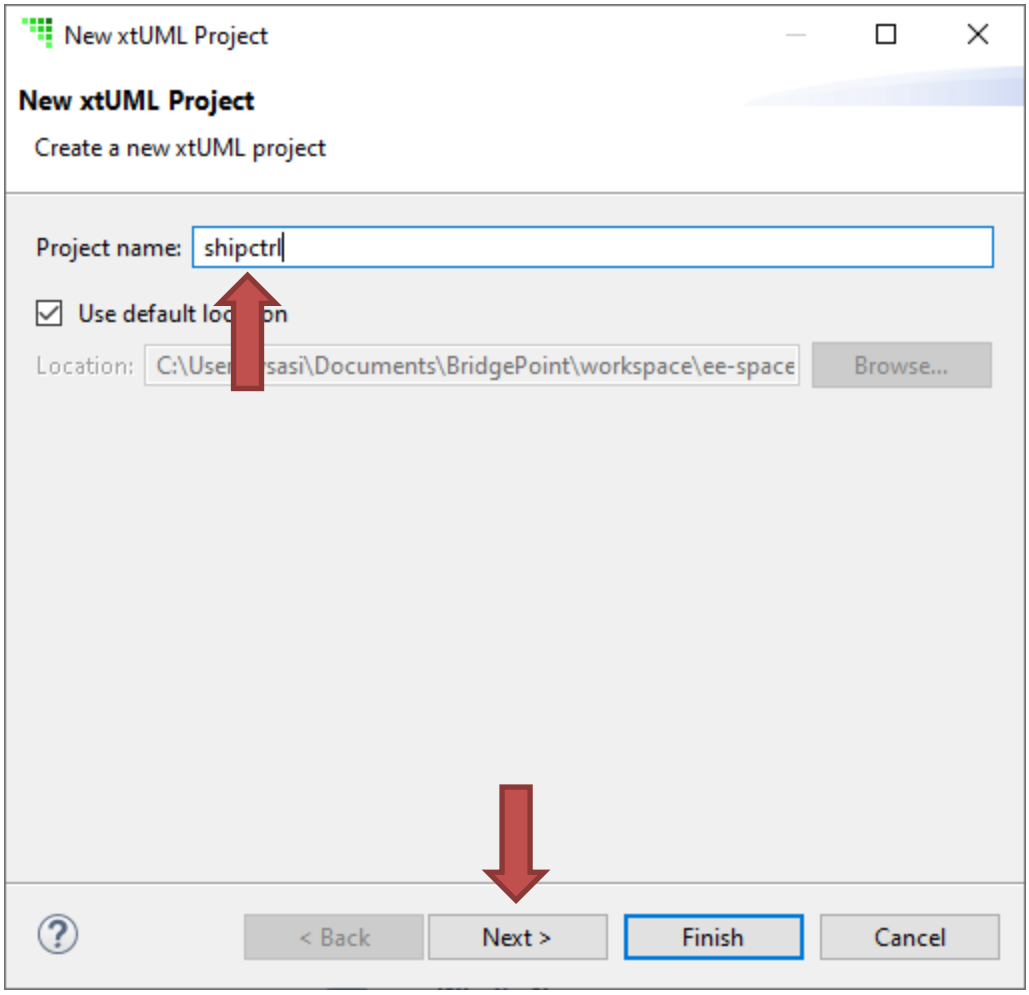
Starting project with Bridgepoint

Create a new project

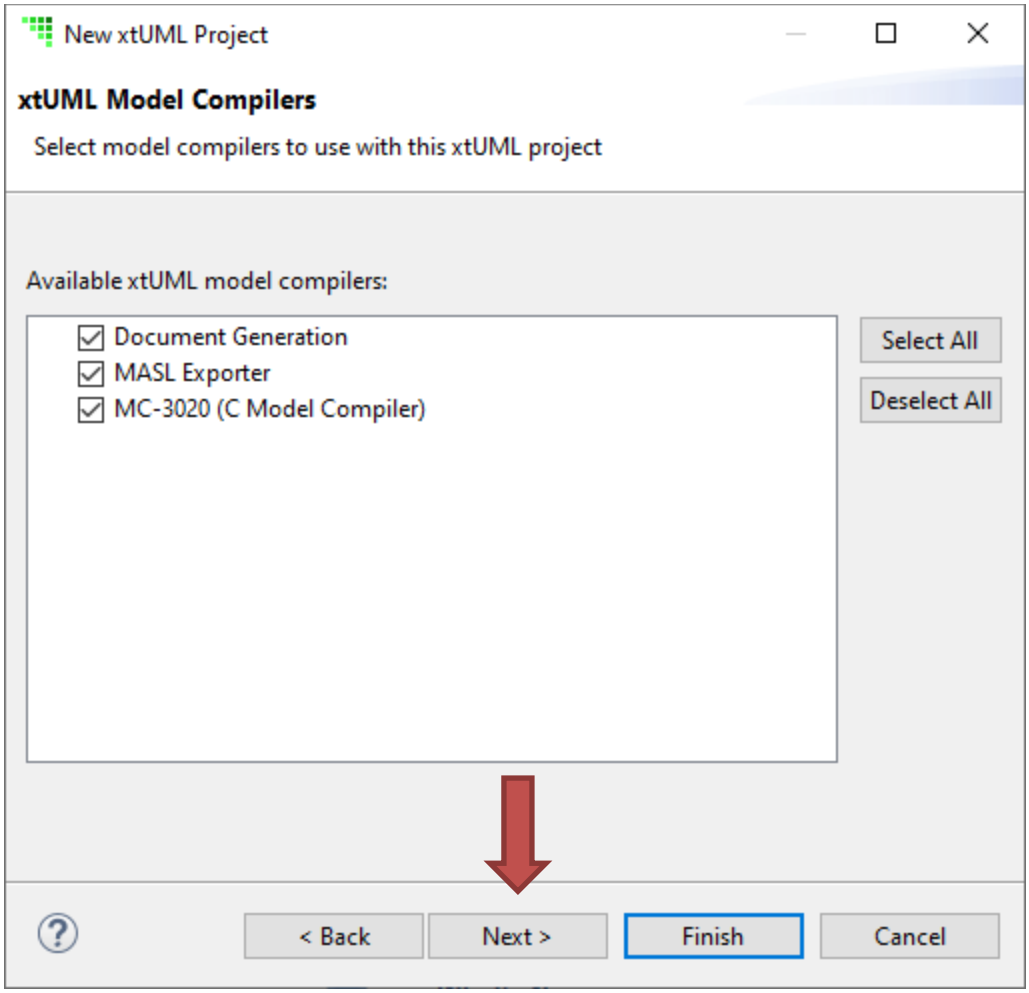
1. File => New => xtUML project



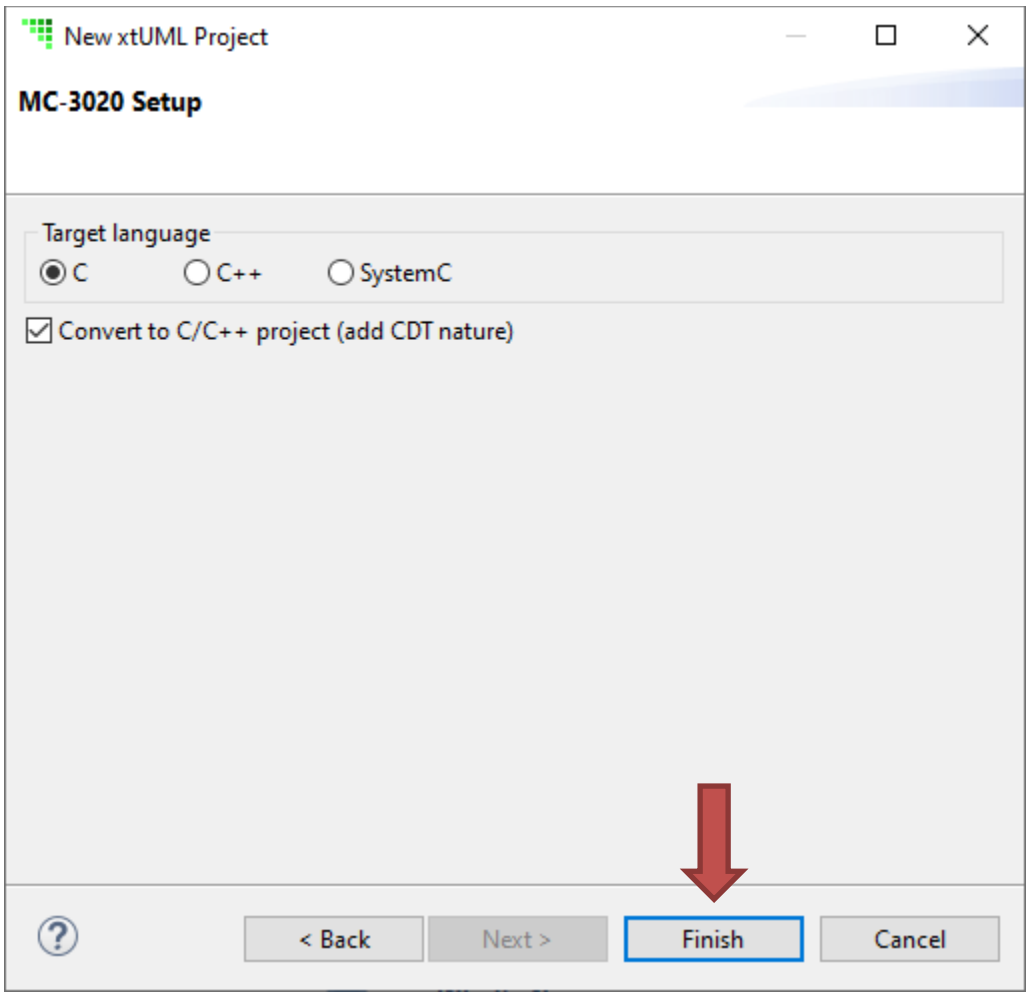
2. Give a name => Next



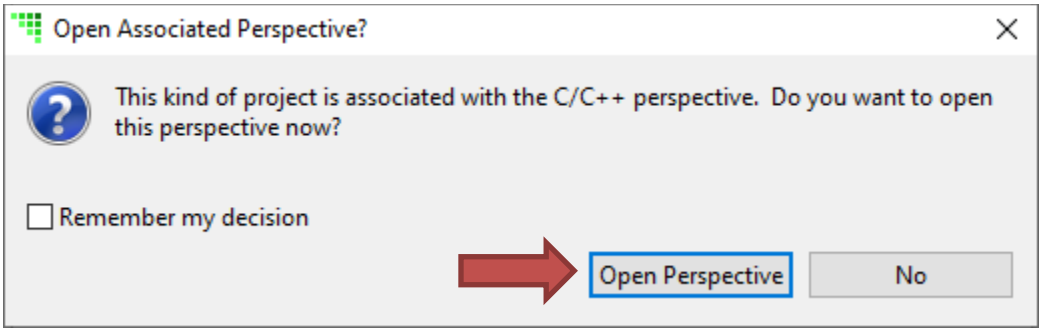
3. Check all model compilers



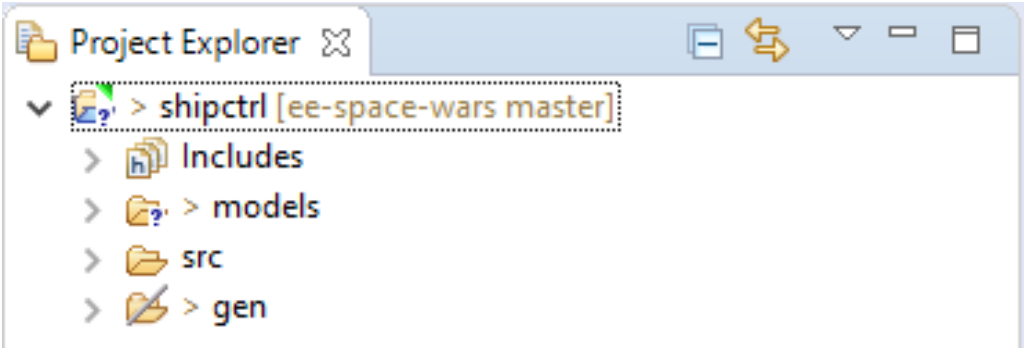
4. Select target C language and check ‘Convert to C/C++ project’ => Finish



5. Click Open perspective

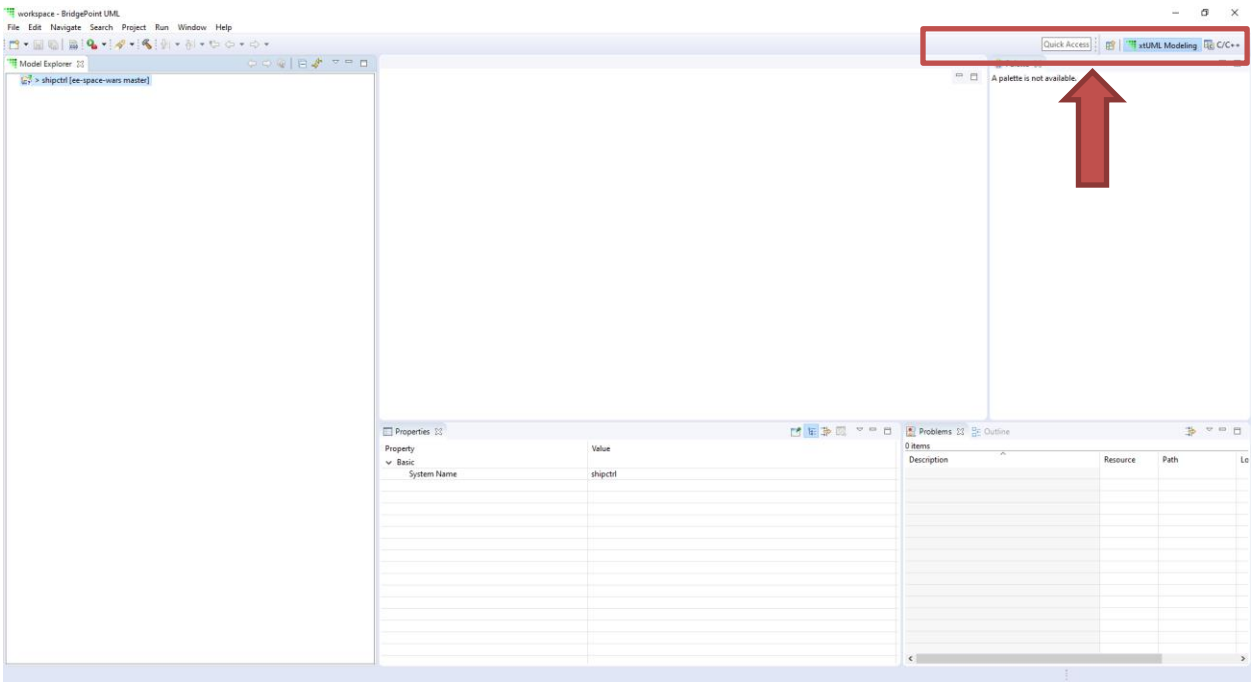


6. Finally, you should have project structure like this on C/C++ perspective

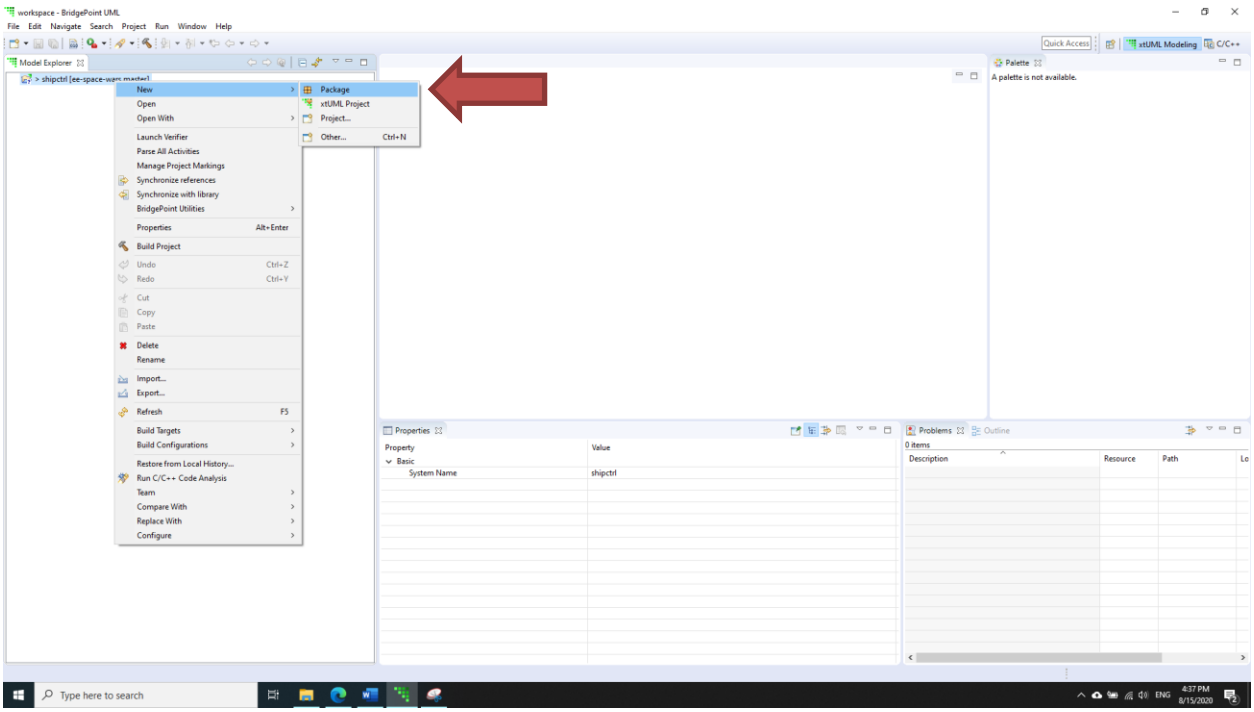


Create Package

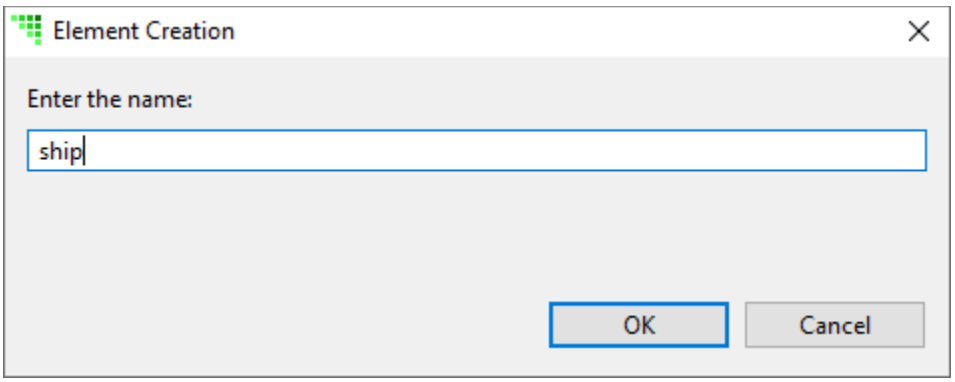
1. Change to xtUML Modeling perspective



2. Right click on project folder => New => Package

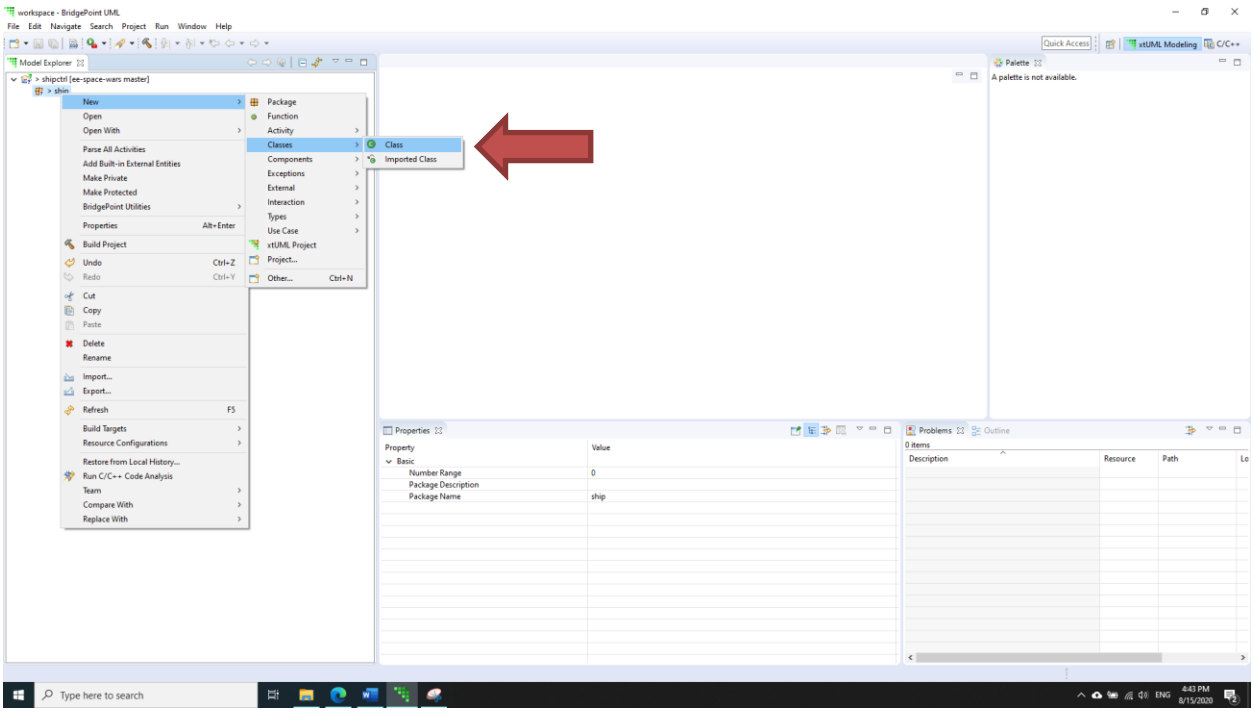


3. Give a name => OK

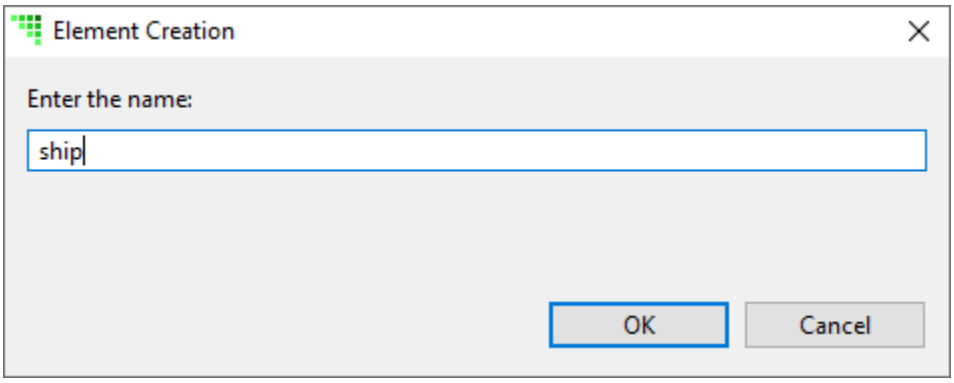


Create Controller

1. Right click on exist package => New => Classes => Class



2. Give a name => OK

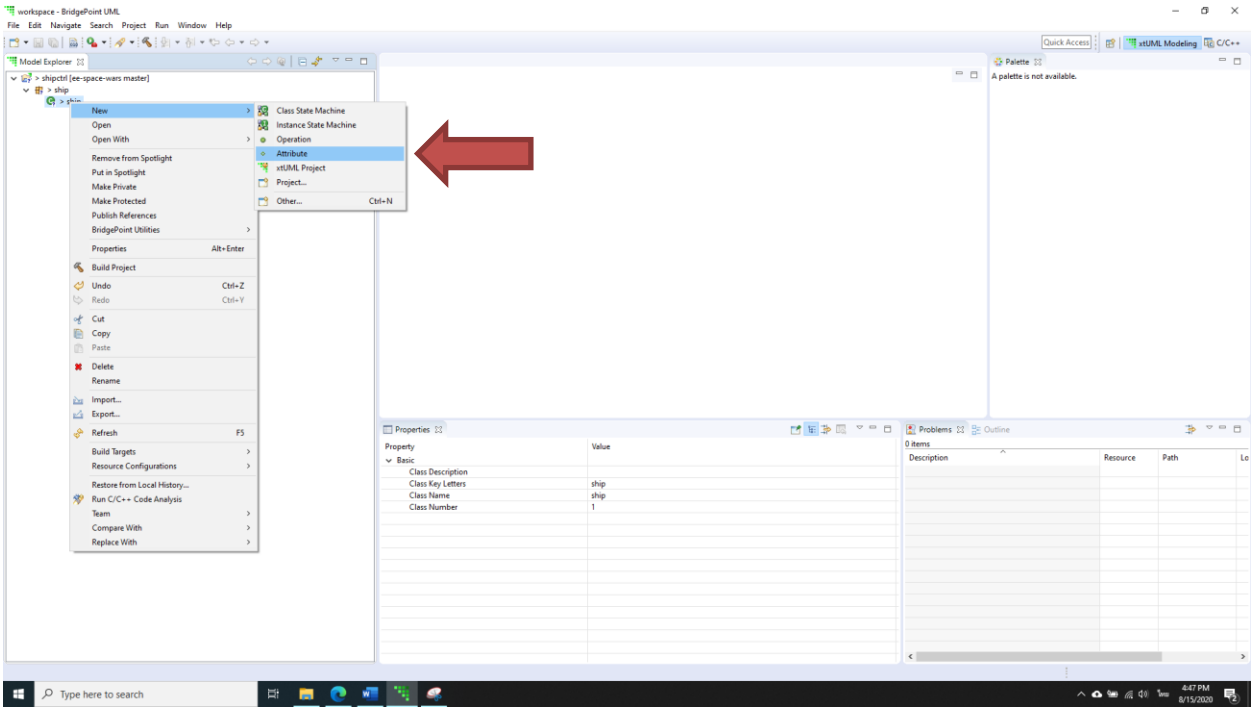


The attribute of a Class

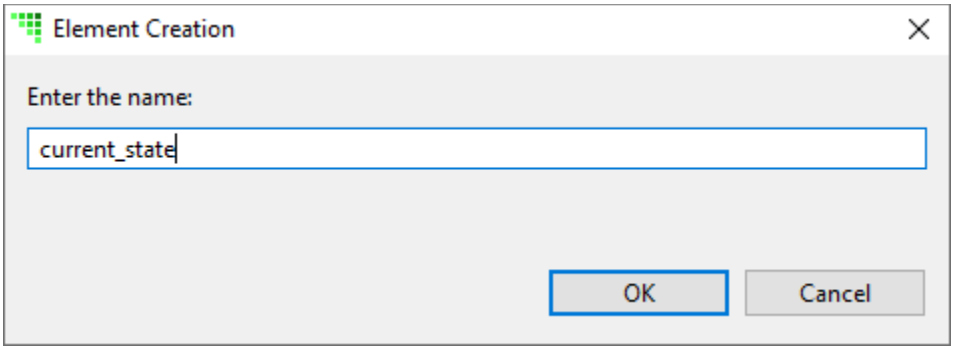
The class attribute is using for storing the value as a global variable.

Add an Attribute

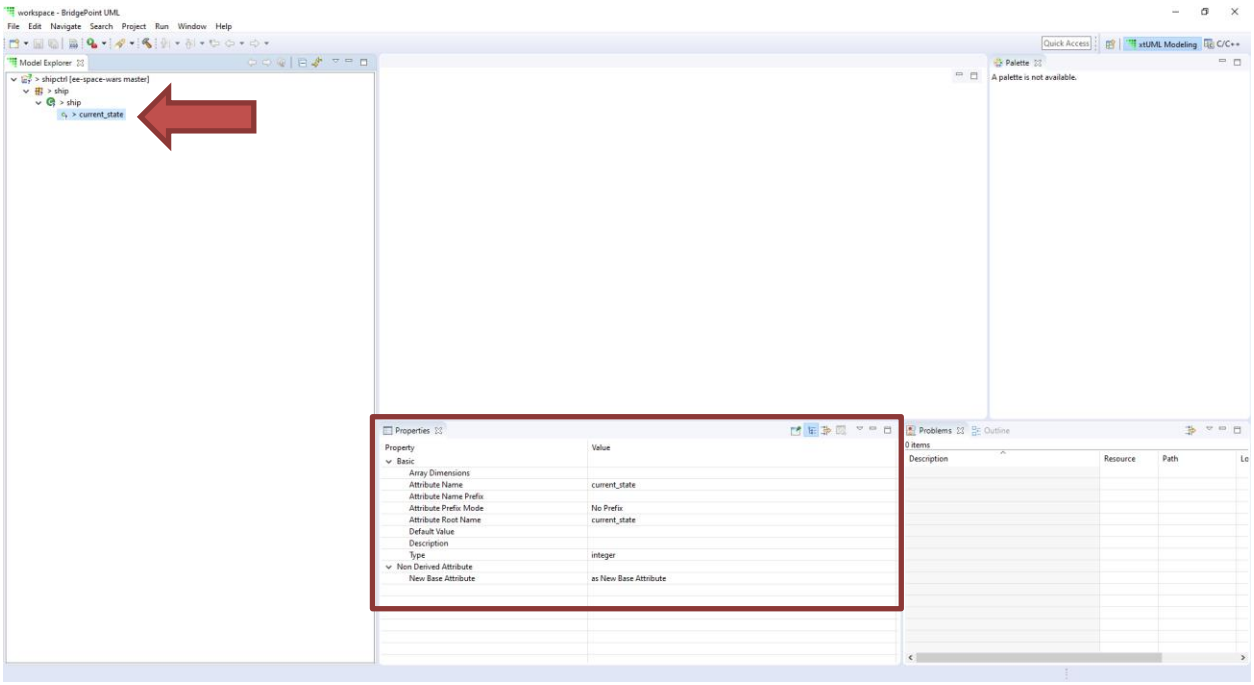
1. Open the class diagram to click a package
2. Right-click the class you want to add an attribute
3. New => Attribute



4. Give a name => OK



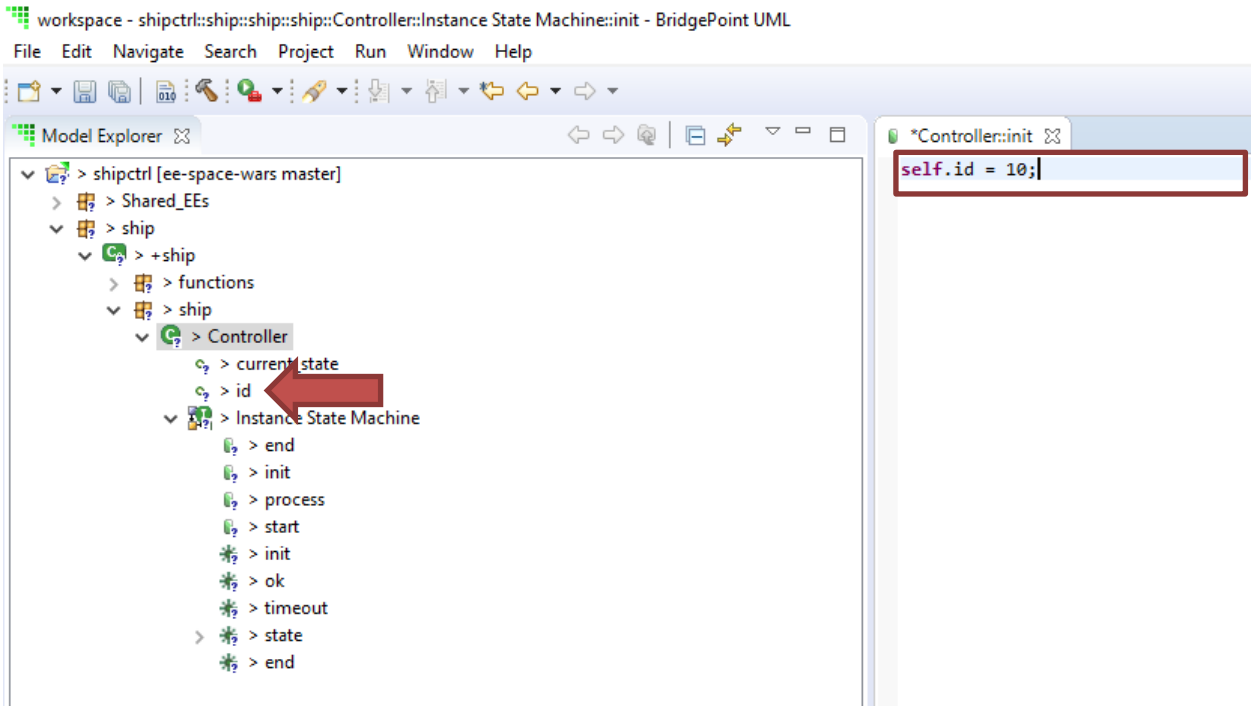
5. Set a type of the attributes in Properties view



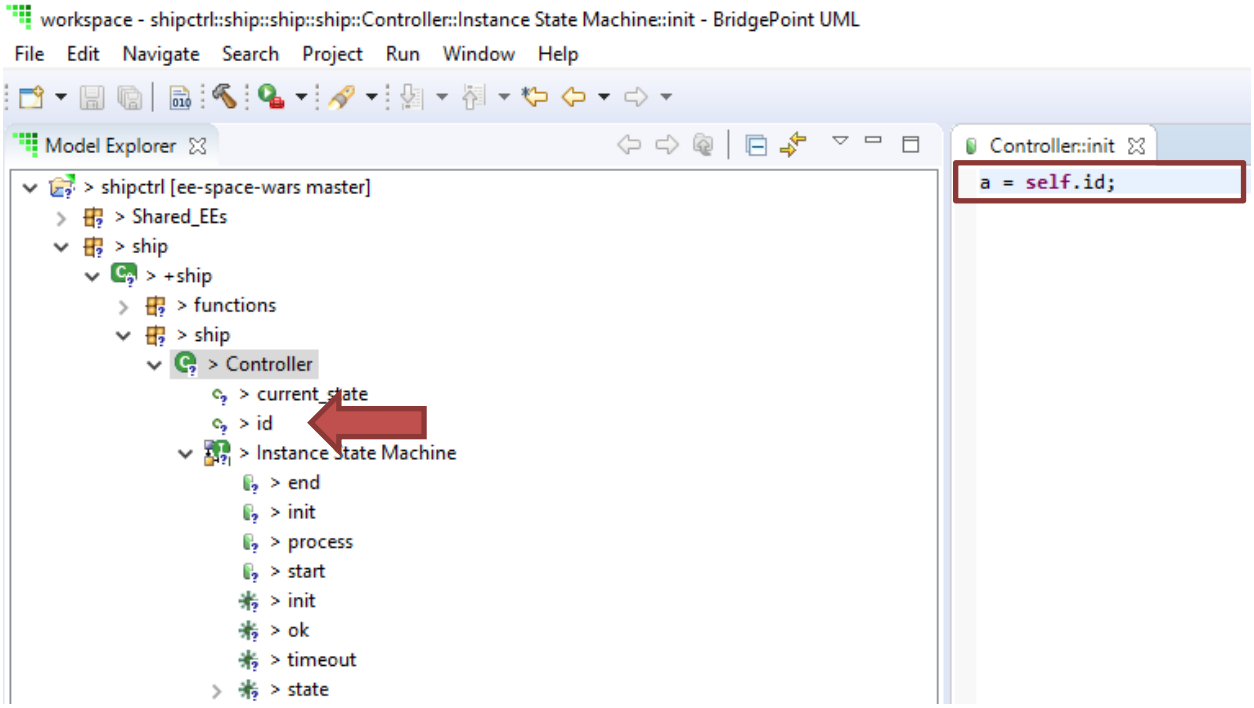
Set or Get a Value to Attribute

Implement in **states** or **functions**, for an example, we denote the name of an attribute is 'id'.

- Set: self.id = 10;



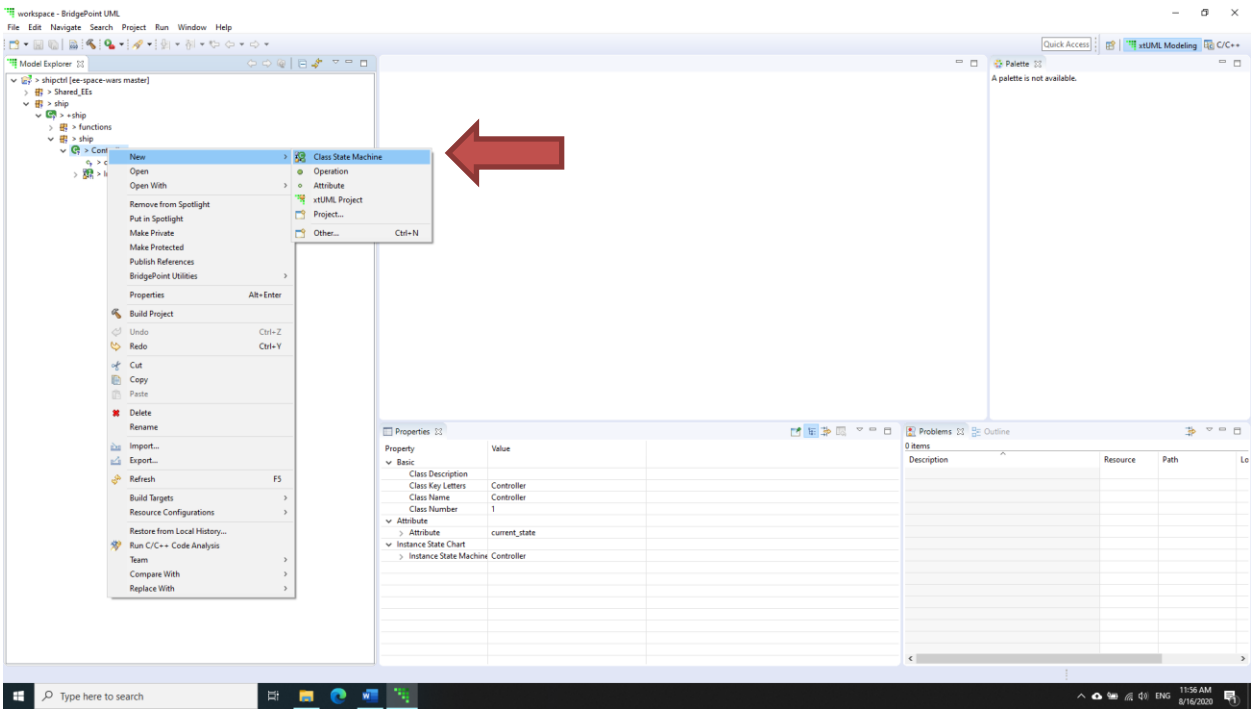
- Get: a = self.id;



Instance state machine class

Add an instance state machine class

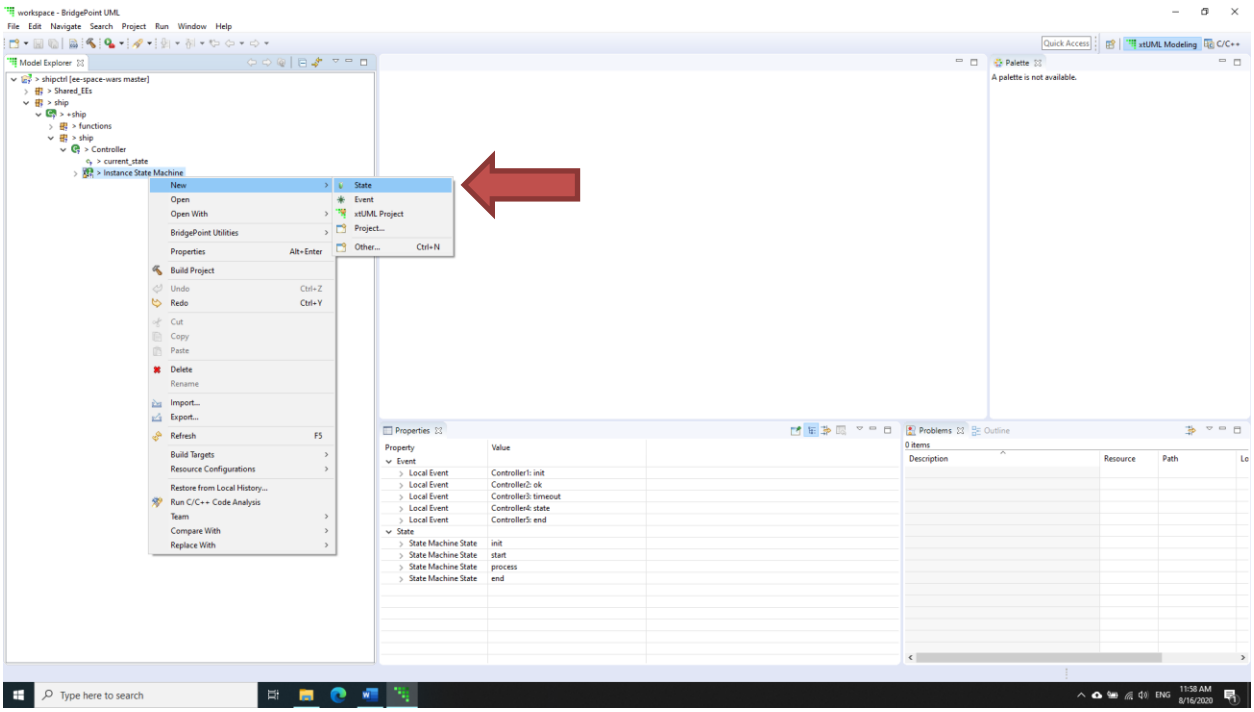
1. Right click on a class => New => Instance state machine class



State

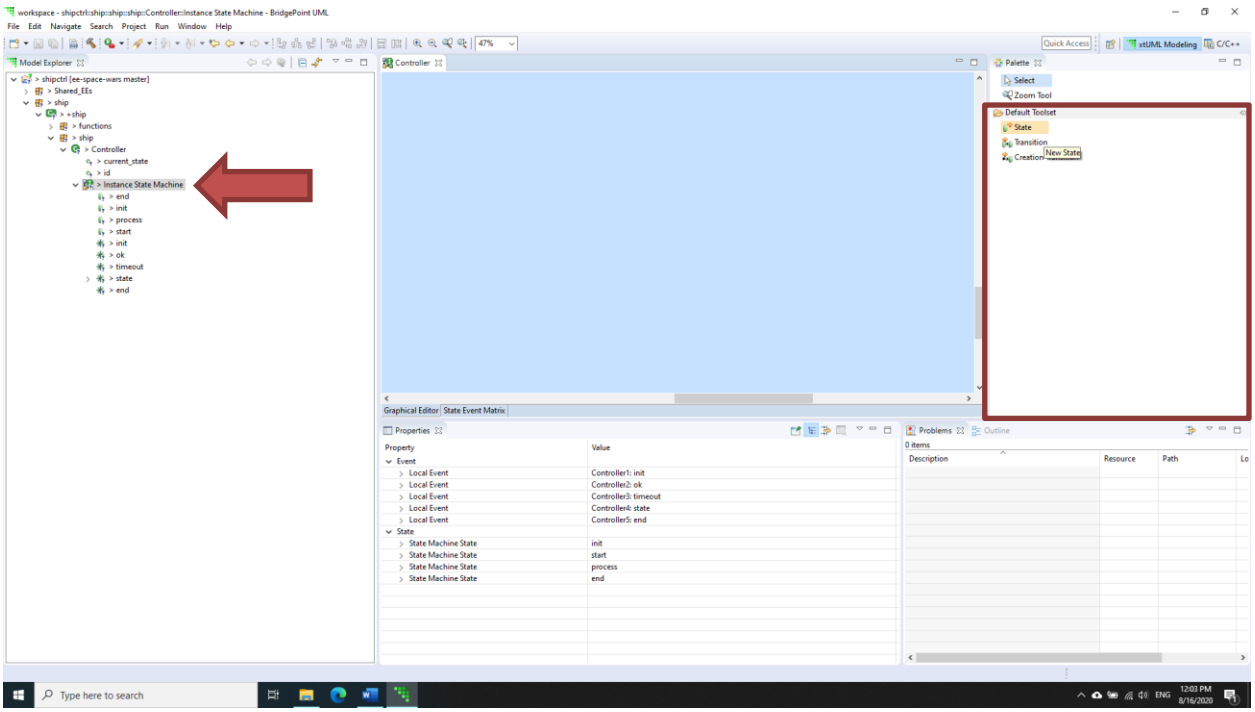
Add state element to machine class

1. Right click on a state machine class => New => state

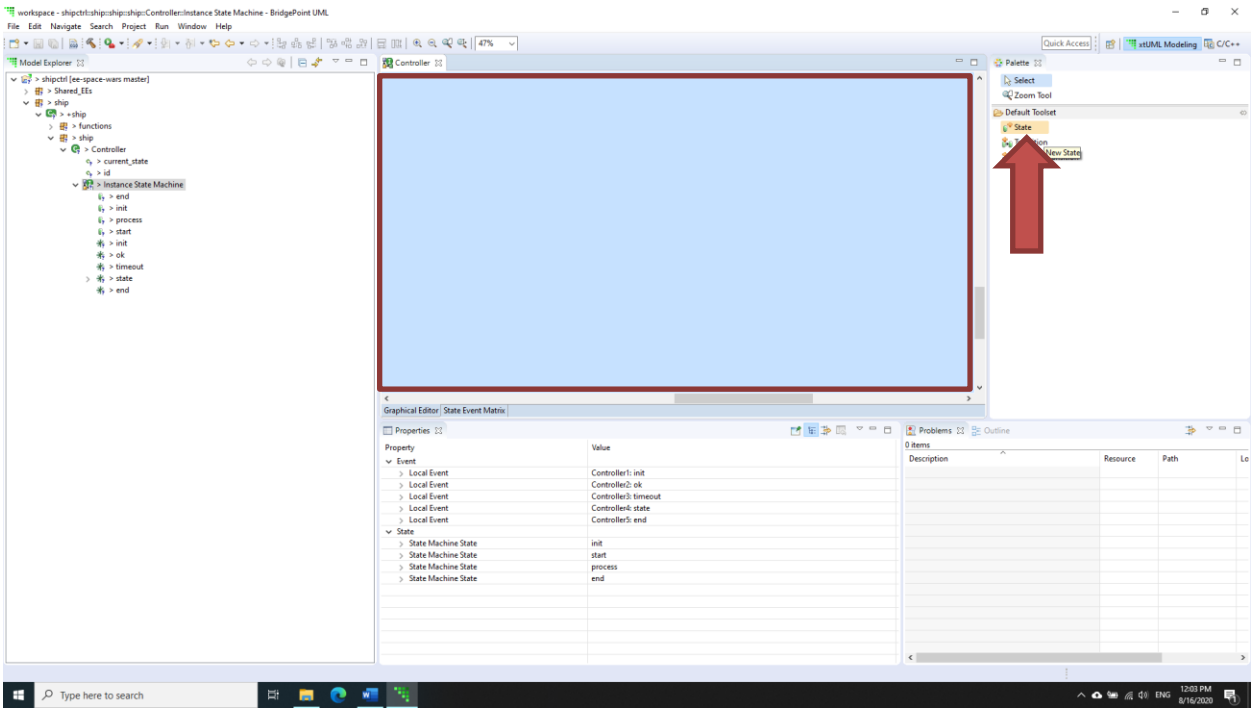


Or another way to add a new state

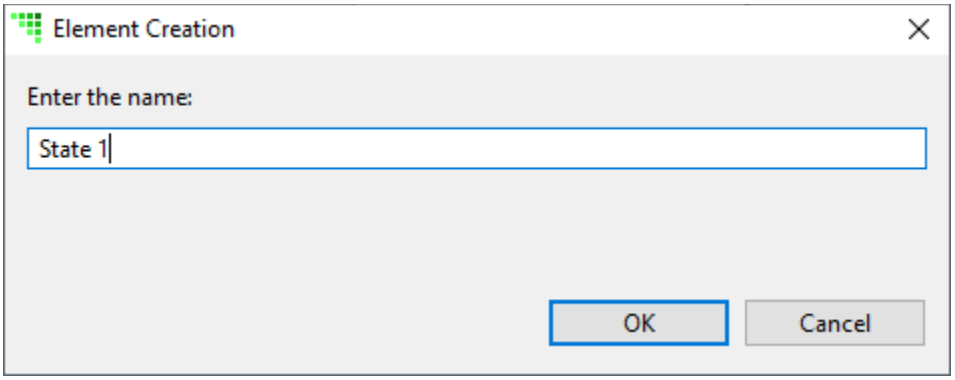
1. Double click on Instance machine class => Looking on Default toolset



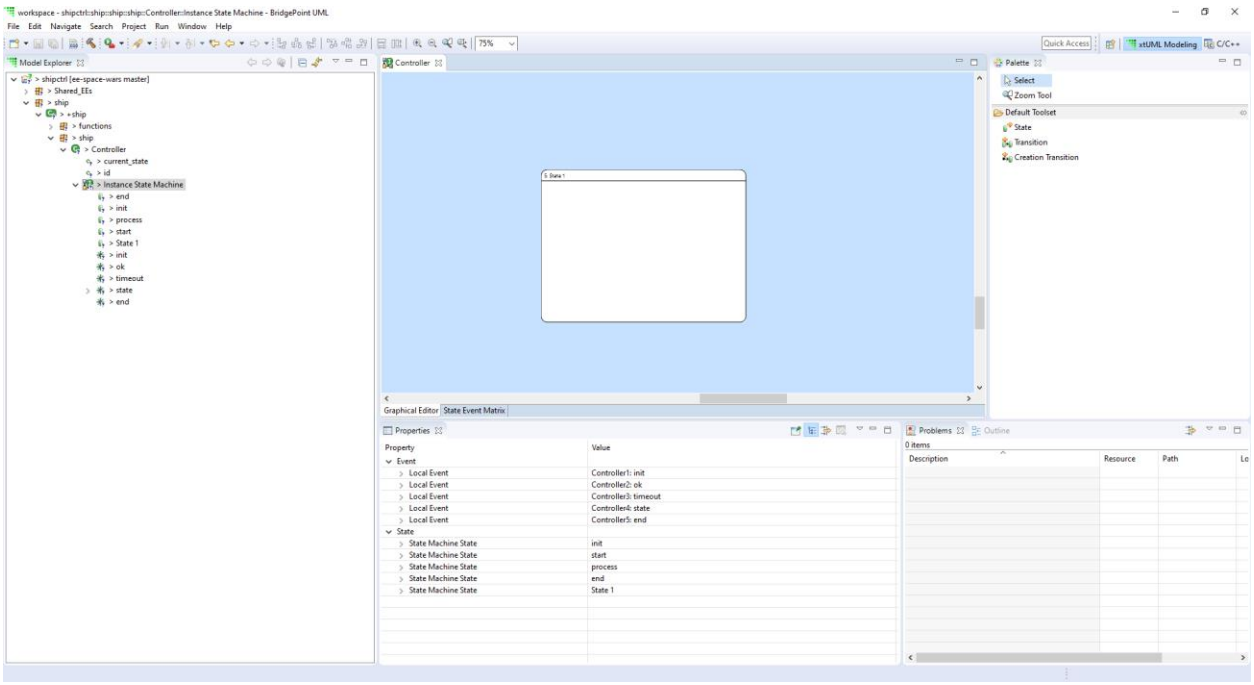
2. Click on State and place on a blue space by single click



3. Give a name => OK



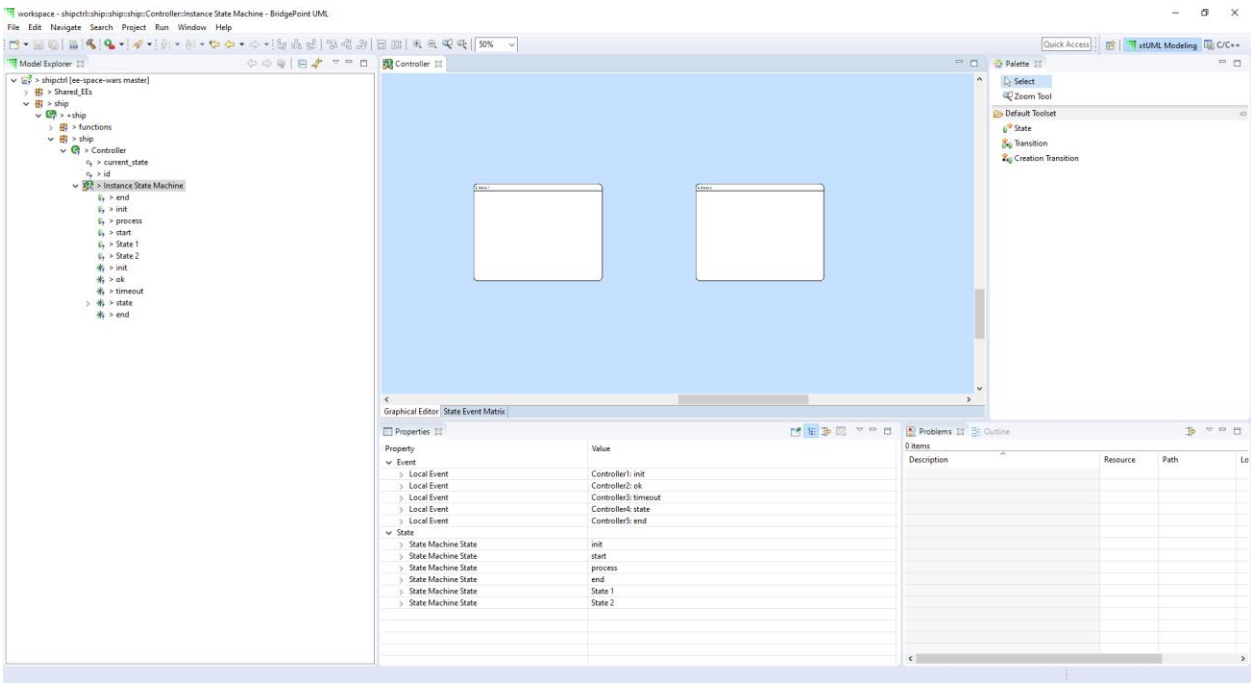
4. Finally, you should get this



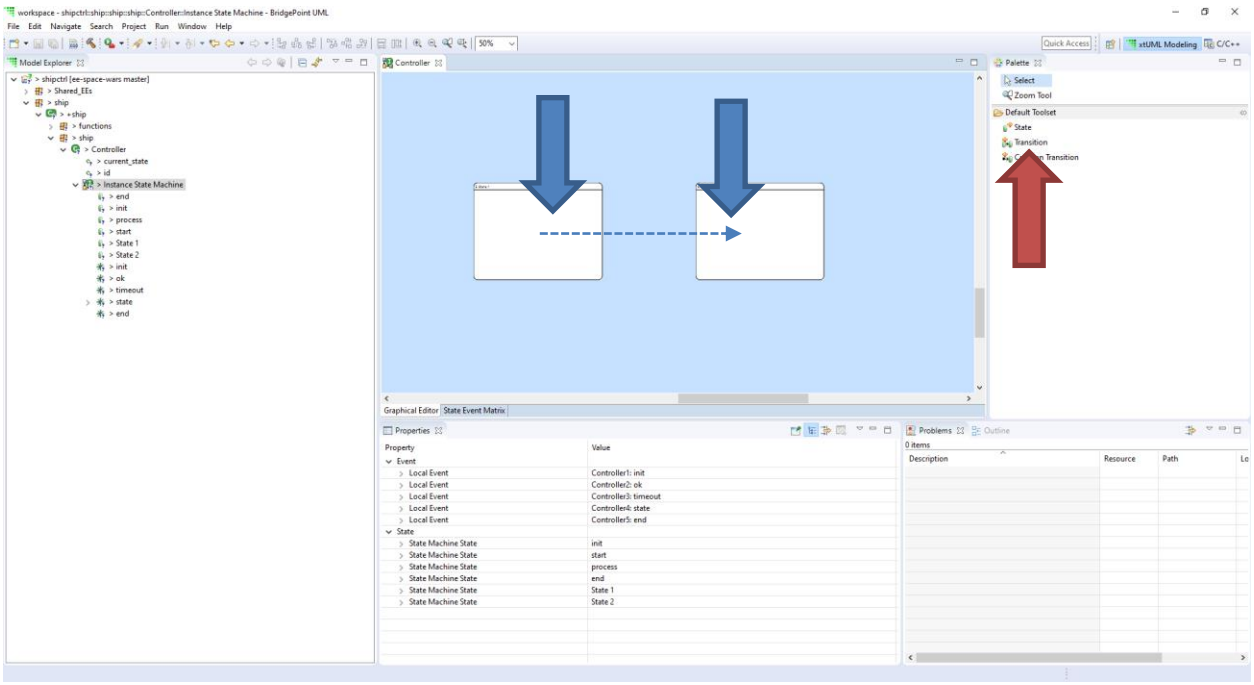
Transition

Link Transition between 2 state elements

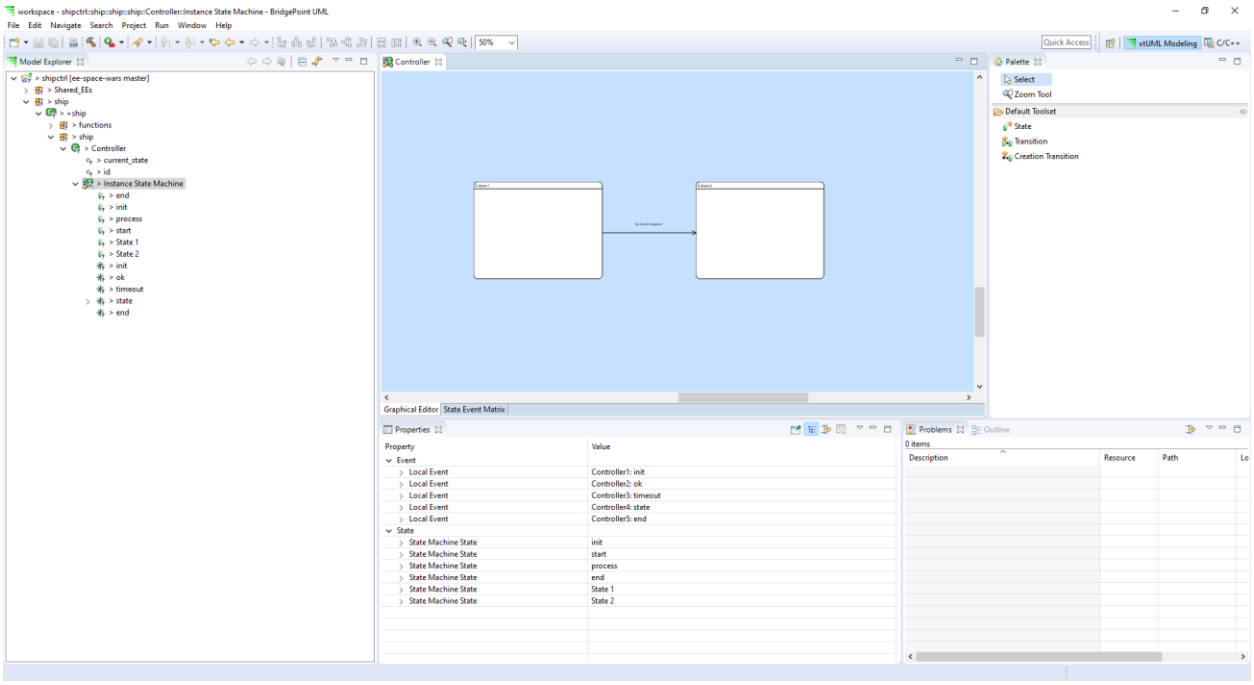
1. Create 2 state element on blue space



2. Click on Transition in the default toolset => Drag from source element to destination element.



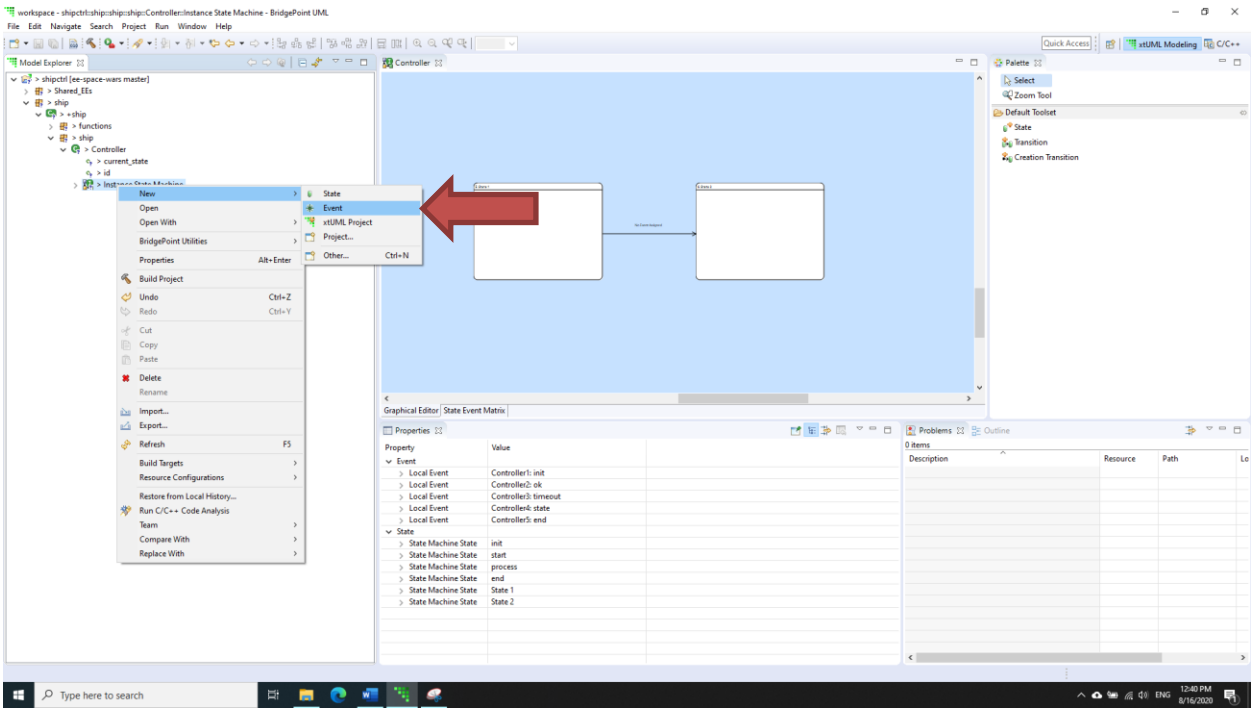
3. Finally, you should got this



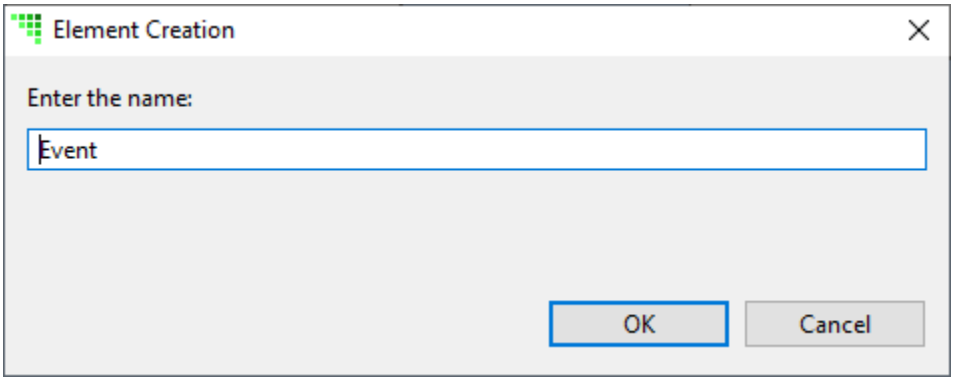
Event

Define an Event

1. Right click on a state machine class => New => Event

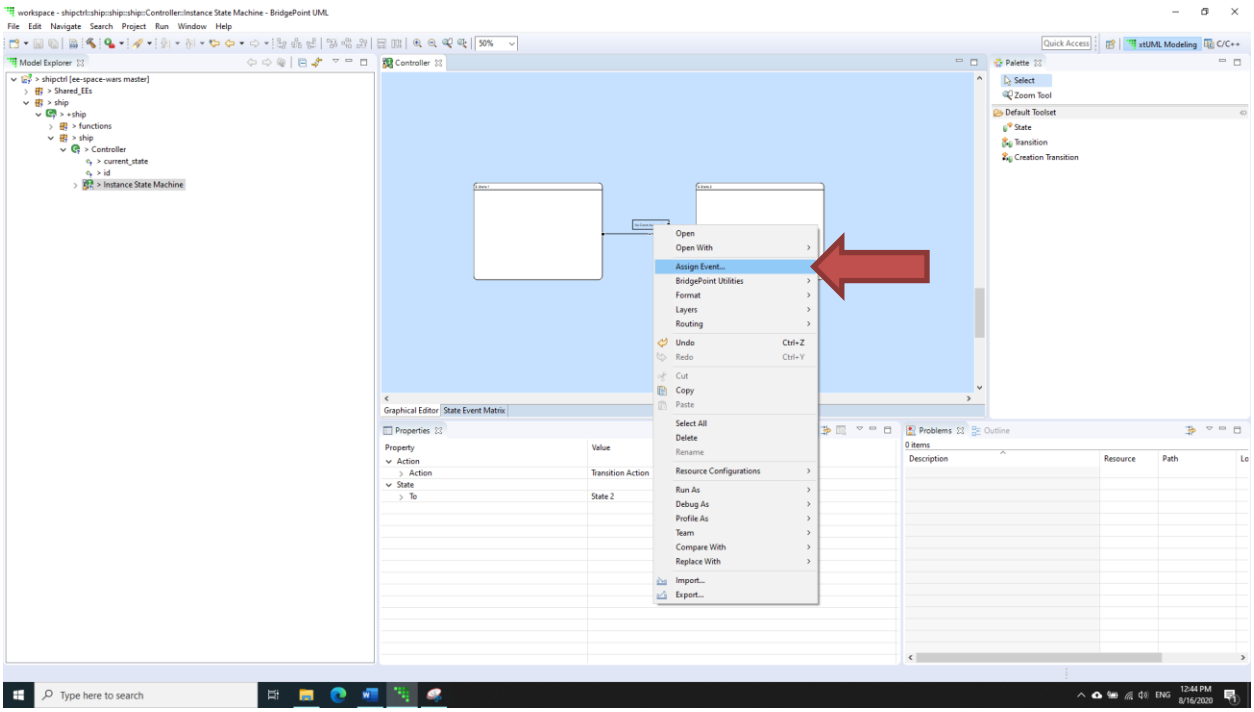


2. Give a name => OK

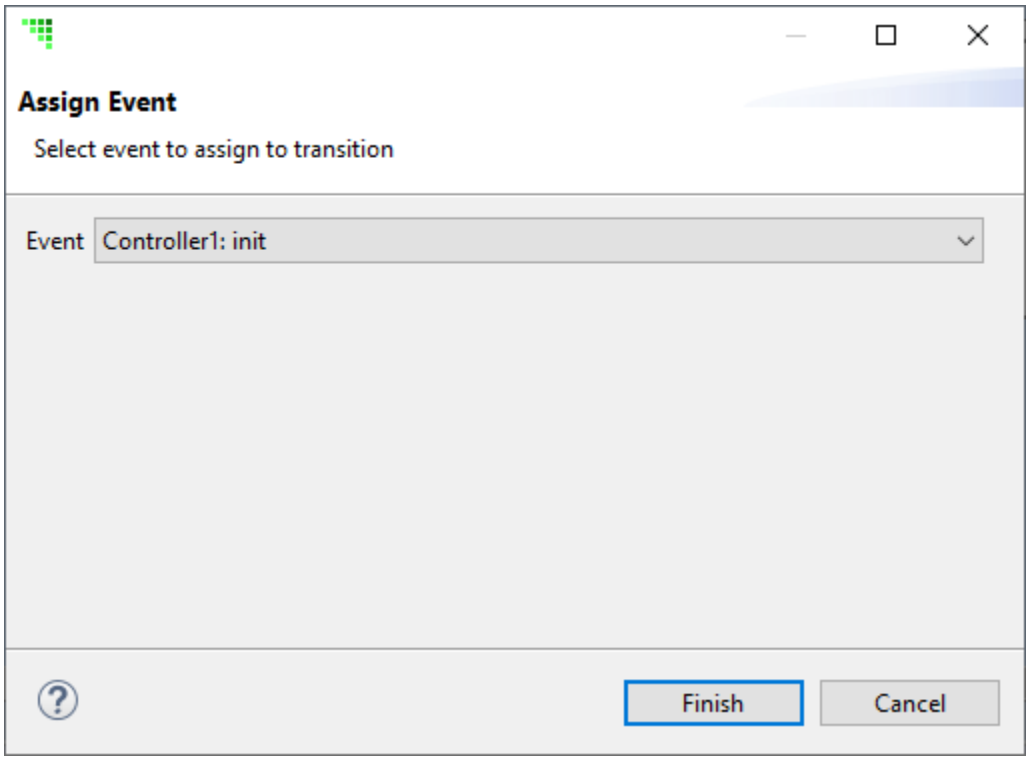


Assign an Event to Transition link

1. Right-click on Transition links => Assign event



2. Select an event => Finish



Getting Parameters in an Event

'rcvd_evt' is a special variable representing received events parameters.

Generate Event in a State

Syntax: generate [event name]:[comment] to [receiver]

```
select any ctrl from instances of Controller;
alive = rcvd_evt.alive;
if(alive == 1)
    Ship::send(str:"angle 180");
    Ship::send(str:"fire");
    generate Controller2:'ok' to ctrl;
else
    generate Controller5:'end' to ctrl;
end if;
```

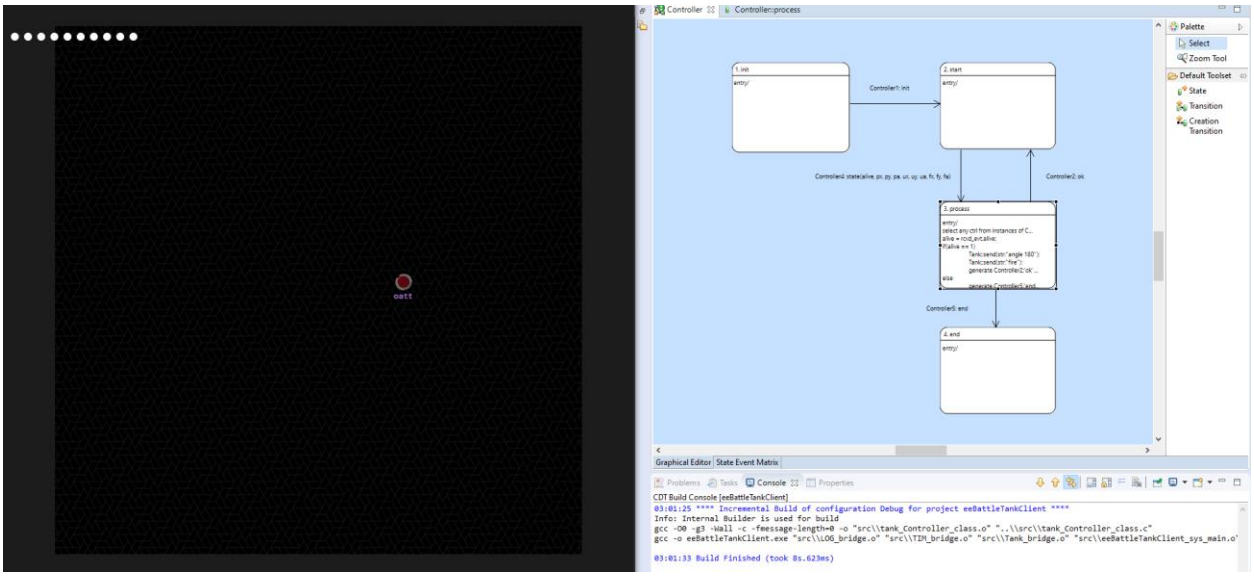
Get parameter from Event

Generate Event to specific state

State machine diagram Assignment 1

State machine diagram typically is used to describe state-dependent behaviour for an object. An object responds differently to the same event depending on what state it is in. State machine diagrams are usually applied to objects but can be applied to any element that has behaviour to other entities such as actors, use cases, methods, subsystems systems, etc. and they are typically used in conjunction with interaction diagrams (usually sequence diagrams).

EE space wars Instance state machine rules



….

Exercise

- Explain and Draw State machine diagram of your state machine diagram using Bridgepoint.

Activity diagram — Assignment 2

Activity diagram is another important behavioral diagram in UML diagram to describe dynamic aspects of the system. Activity diagram is essentially an advanced version of flow chart that modeling the flow from one activity to another activity.

Exercise

- Create a new Package and Draw an activity diagram of your state machine diagram using Bridgepoint.

Sequence diagram — Assignment 3

Sequence Diagrams show elements as they interact over time and they are organized according to object (horizontally) and time (vertically)

Exercise

- Create a new Package and Draw a sequence diagram of your state machine diagram using Bridgepoint.