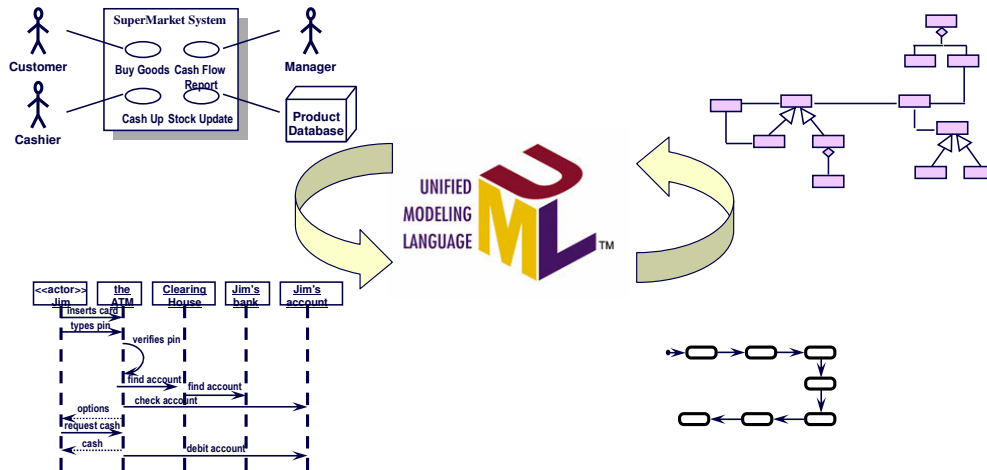




UML Diagram Relationships

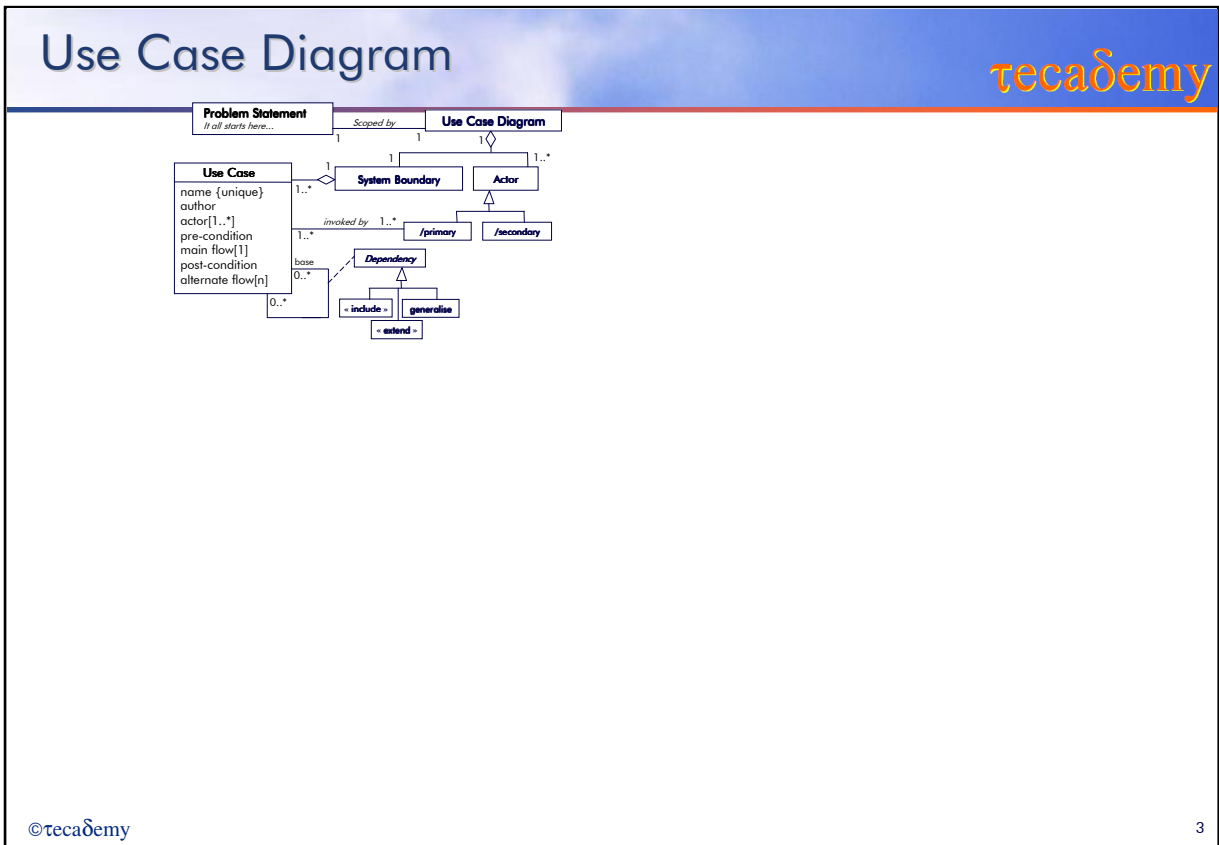
Objectives

- To show the components of the major UML diagrams
- To show how the major UML diagrams relate to each other



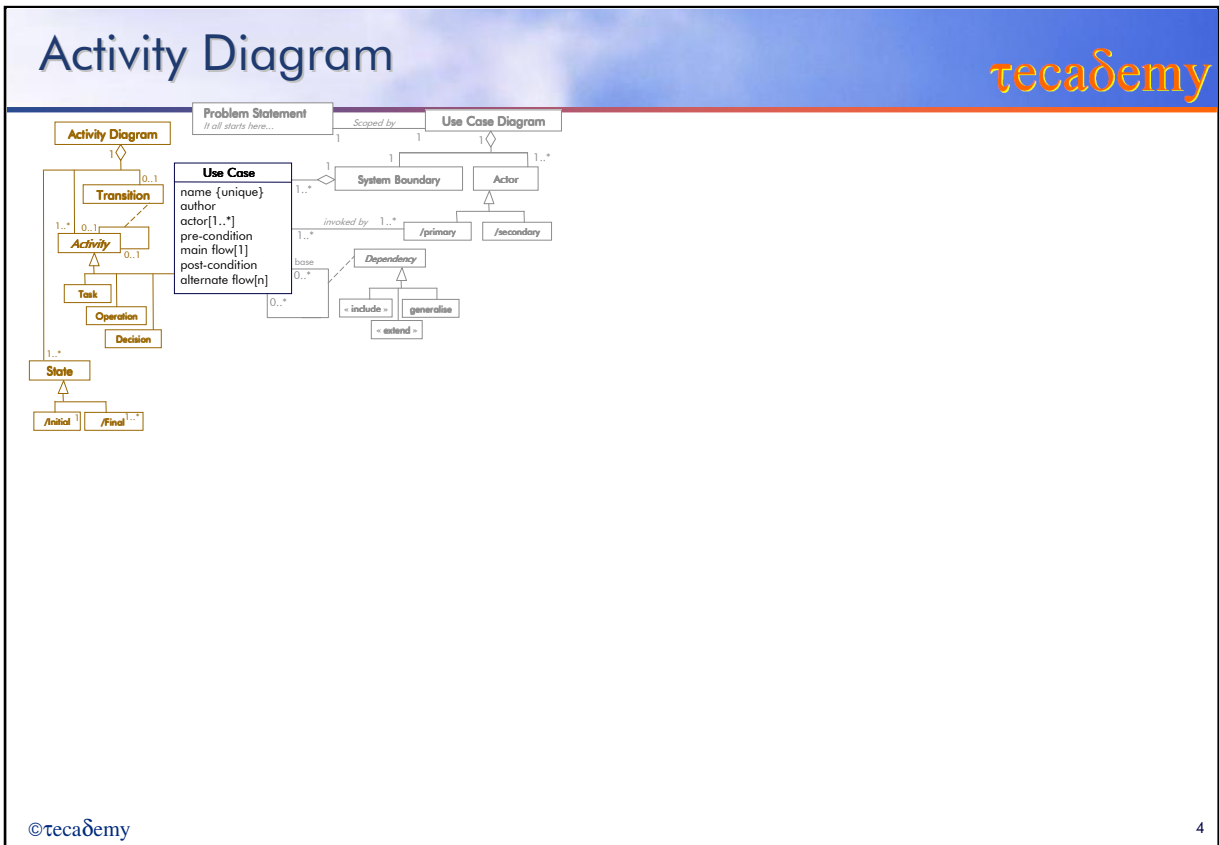
This session aims to show the way the different UML diagrams fit together. It uses UML class diagram notation to build a model to describe these interdependencies, adding one diagram type at a time.

Colour can be a useful device to show strongly related groups of classes and relationships – it is used in the lecturer's presentation to accentuate the elements of individual diagram types. For those of you watching in black and white, a colour print of the final model is available on request!

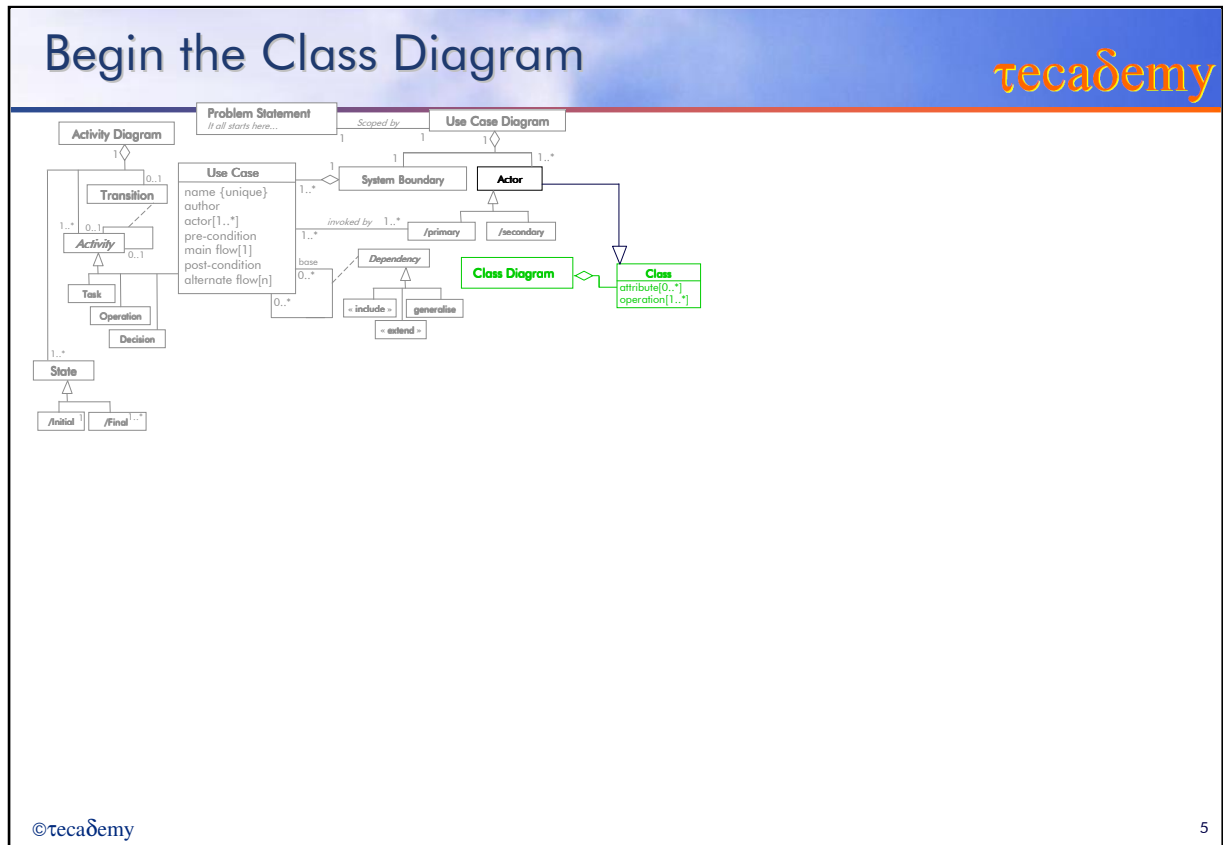


Before we can start on use cases, we need a statement of the problem. The *Use Case Diagram* defines the scope of the problem. Use cases that are in scope are contained within the system boundary, actors are outside. An actor that invokes a use case is *primary* with respect to that use case; one that is contacted by a use case is *secondary* with respect to that use case. It is possible for an actor to be primary with respect to one use case and secondary with respect to another.

A use case can have relationships with other use cases – a use case can *include* or *extend* a *base* use case, or be a generalisation of more specific use cases.

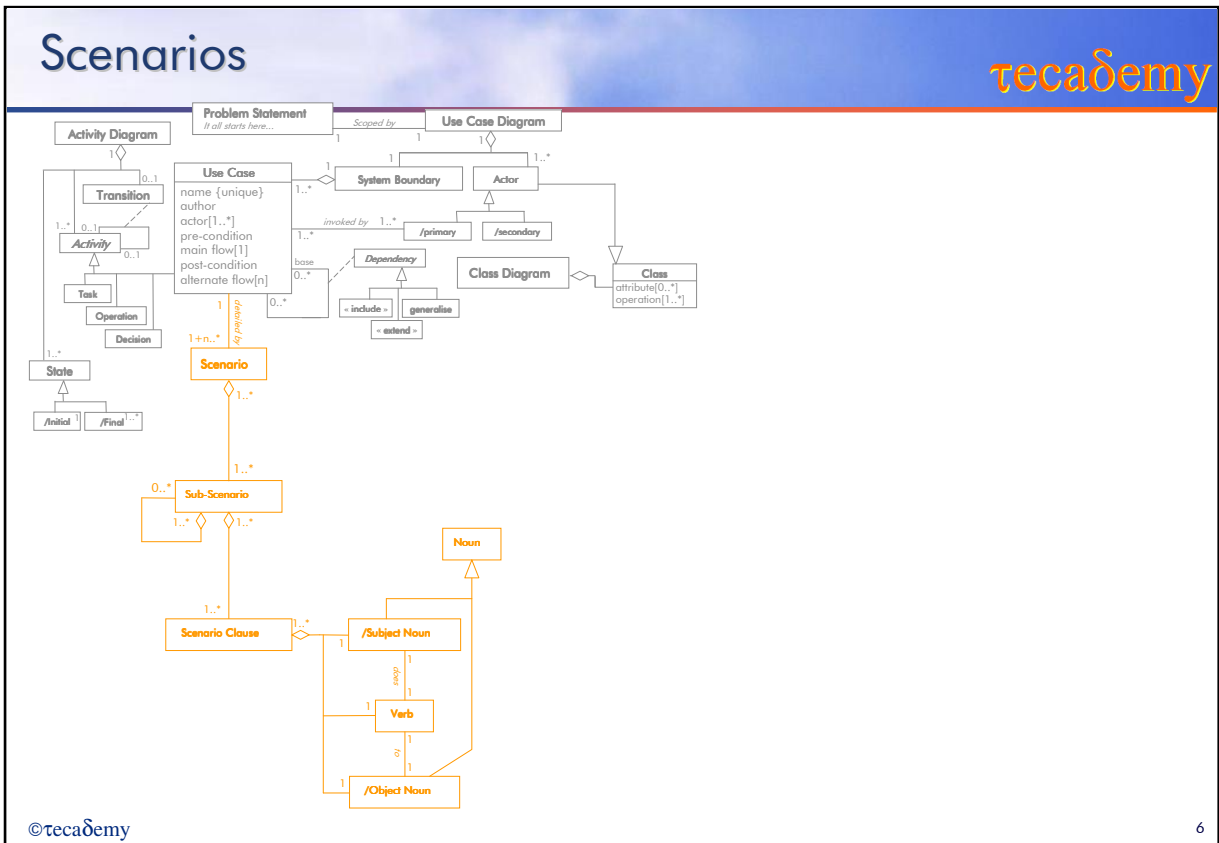


A use case diagram is *static* – there is no indication that one use case follows or precedes another, except perhaps in the case of inclusions or extensions. *Activity diagrams* are *dynamic* and used to show the temporal flow of the activities that comprise an overall process – in other words, an activity diagram represents a process' life history. A use case can be viewed as an activity, but an activity can be finer- or coarser-grained than a use case – for instance, an operation on a class can be modelled as an activity, as can a task or business process. Completion of one activity triggers a transition to the next.

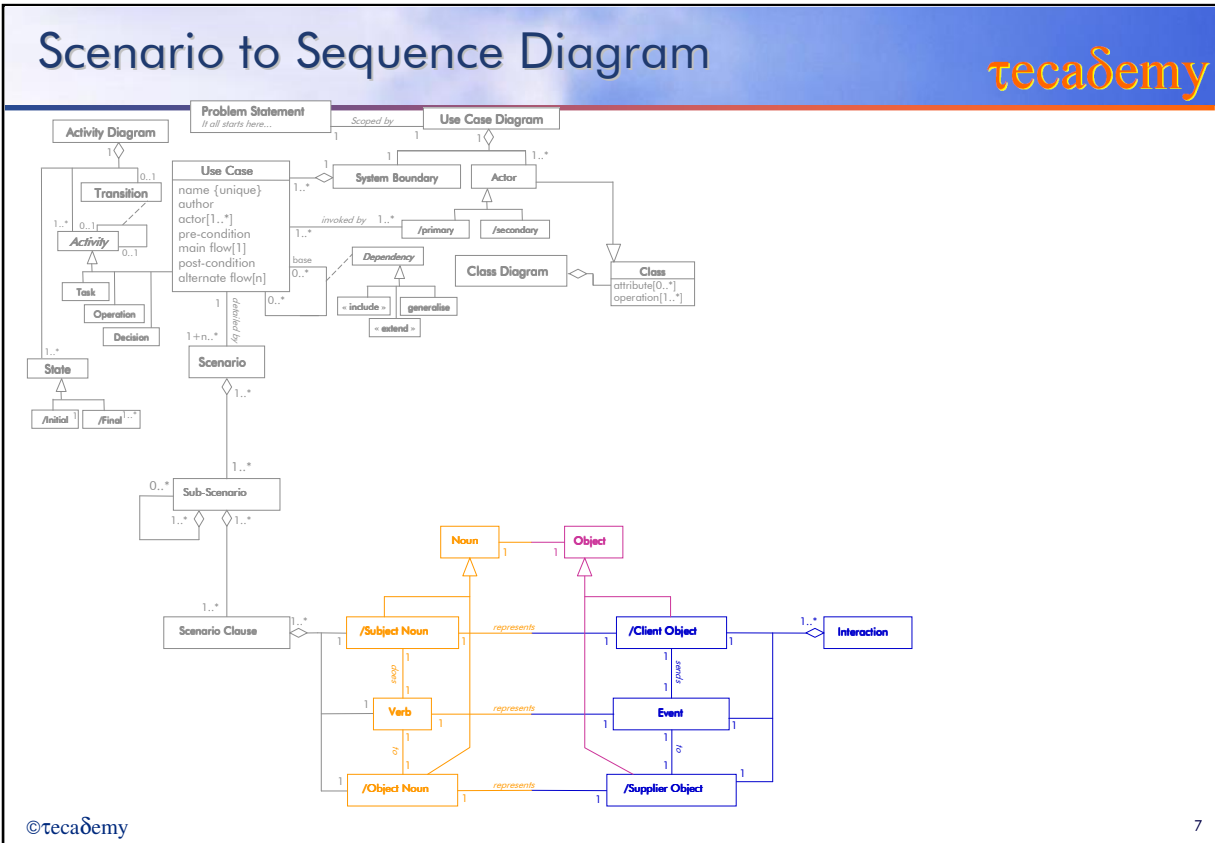


Use cases can be a rich source of domain-specific terms and concepts, some of which could be regarded as candidates for class status. Some terms might indicate properties of a class; an attribute by a noun, an operation by a verb. These can be shown on the initial *class diagram* and serve as the basis for discussions aimed at discovering other classes.

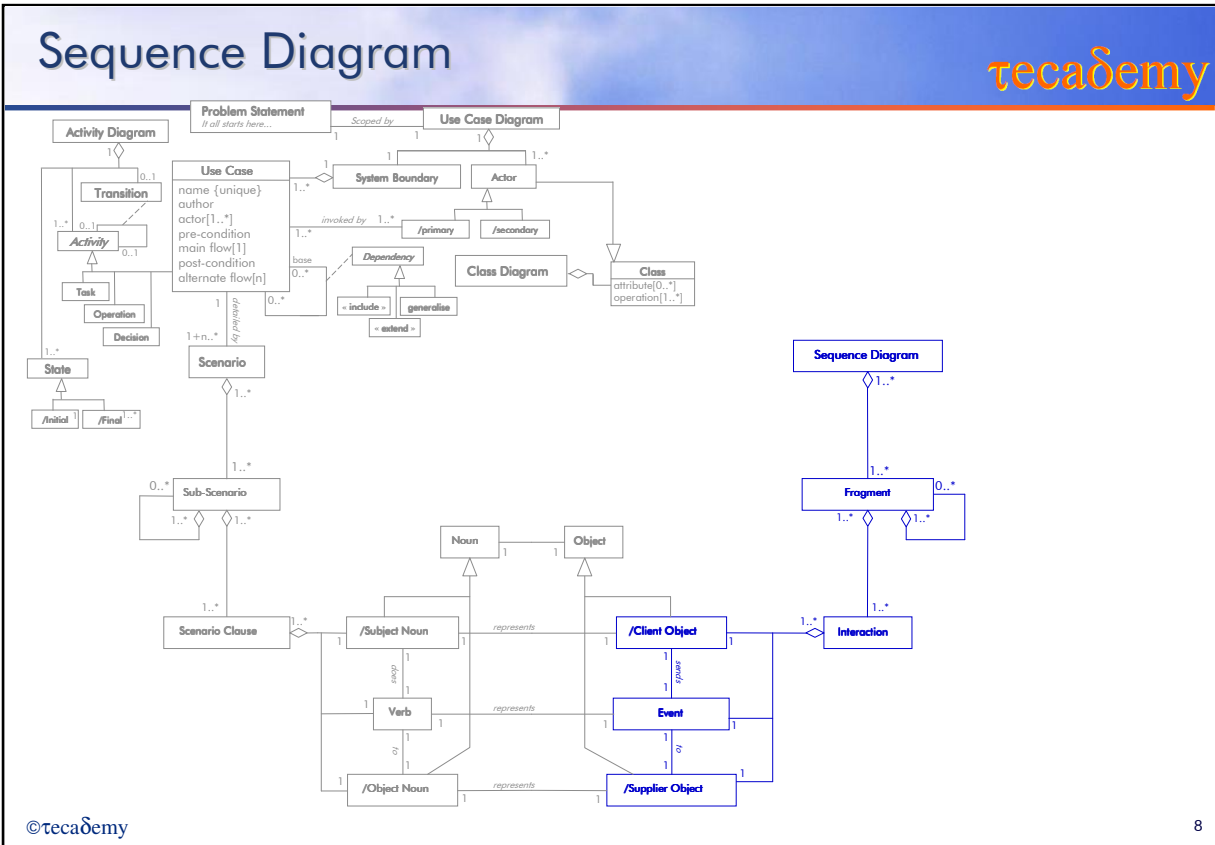
Actors are classes, too; but they are classes that exist outside of our system boundary and are therefore out of scope. They are not included in a class diagram except for contextual purposes.



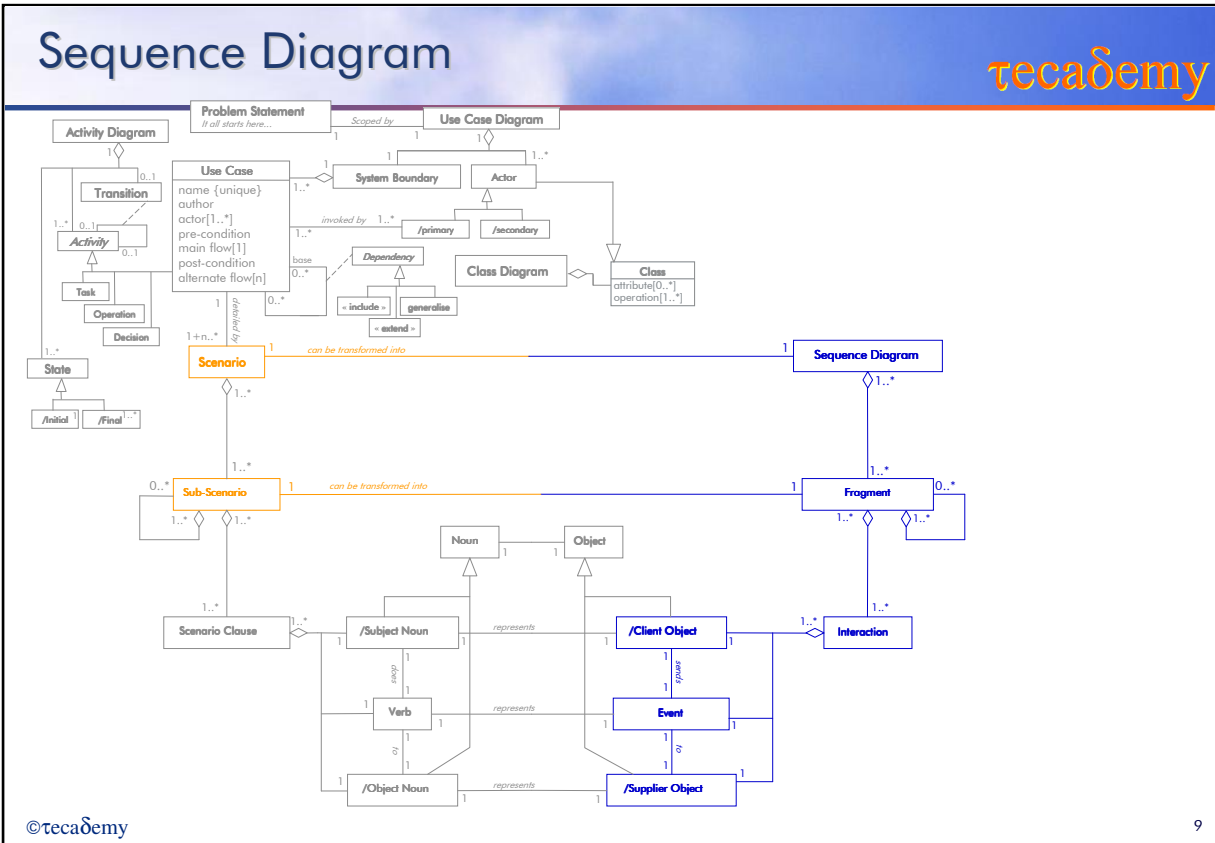
A *scenario* is a description of one instance of a use case in action, using a real example. There will be at least one successful scenario, possibly more; and several (n) error or exception scenarios. Each scenario is written as a sequence of subject-verb-object clauses. It is good practice to identify sub-scenarios – a sub-scenario being a piece of behaviour that could be repeated in other scenarios – as this can reduce the total number of scenarios that need to be written. A sub-scenario could map to an inclusion or extension use case, or could just be a piece of behaviour repeated in lots of scenarios.



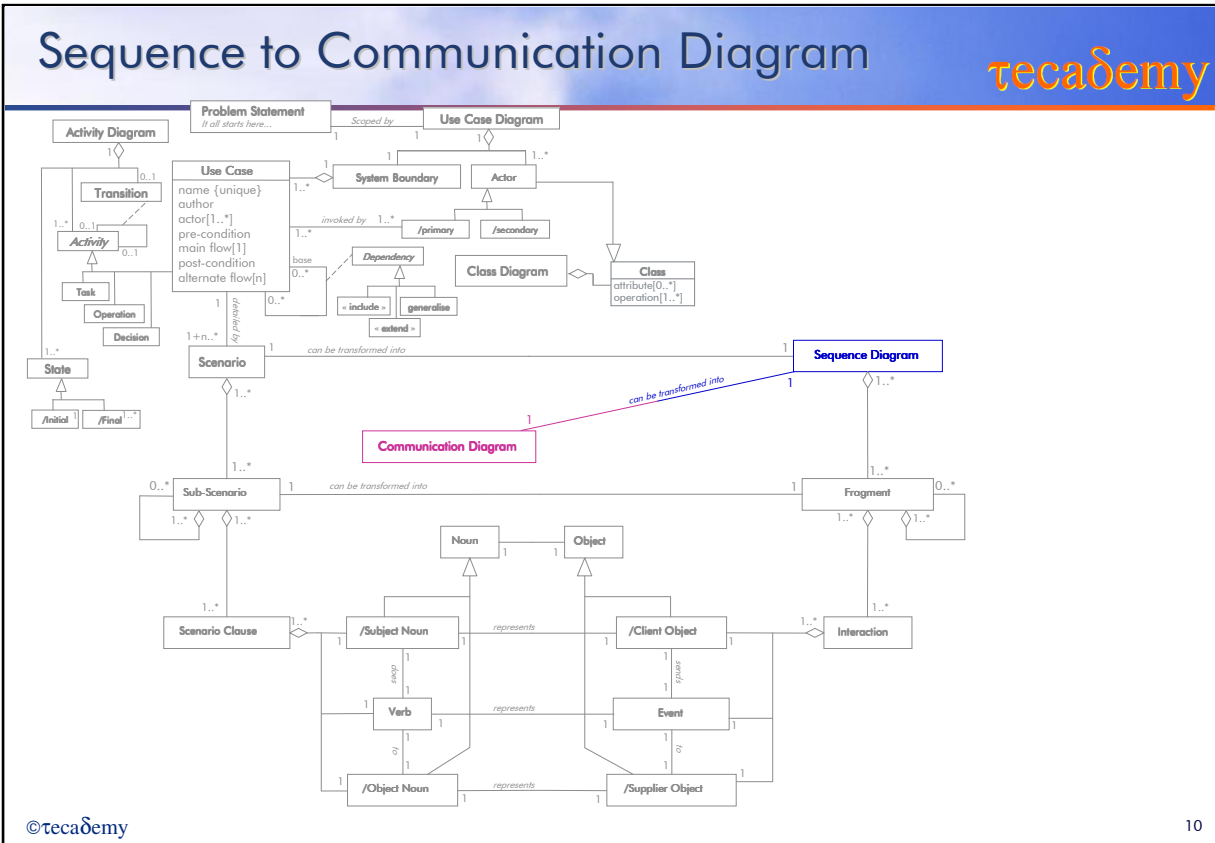
The subject noun does something 'verb' to the object noun (using 'object' in the grammatical sense). If we map nouns to *objects*, the subject noun represents the *client* object, the object noun the *supplier*, and the verb (preferably in the active rather than passive form) shows a message passing from client to supplier. This represents a single interaction between two objects.



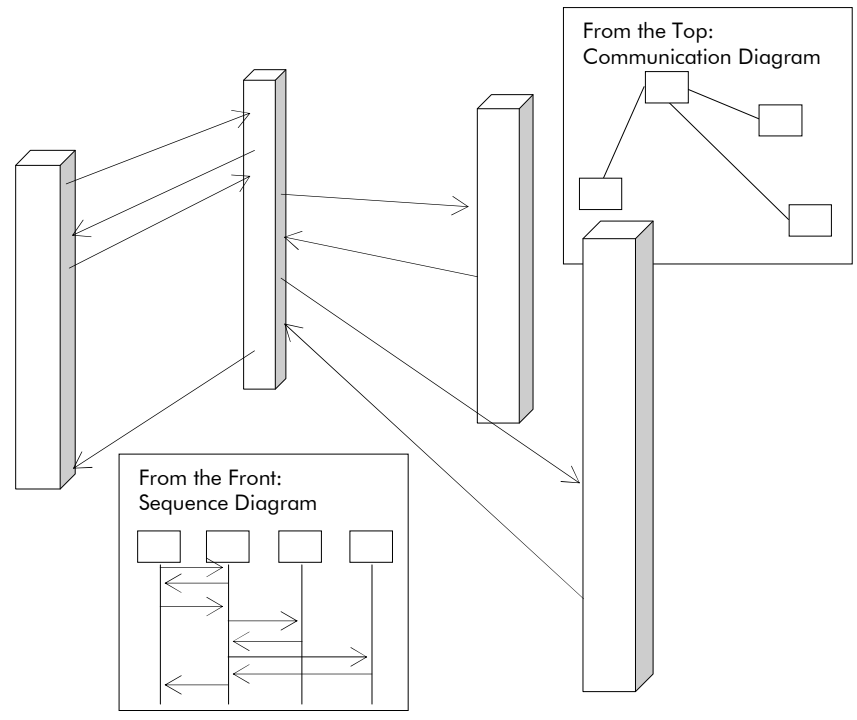
The *sequence diagram*, as the diagram name suggests, shows the sequence of object interactions.

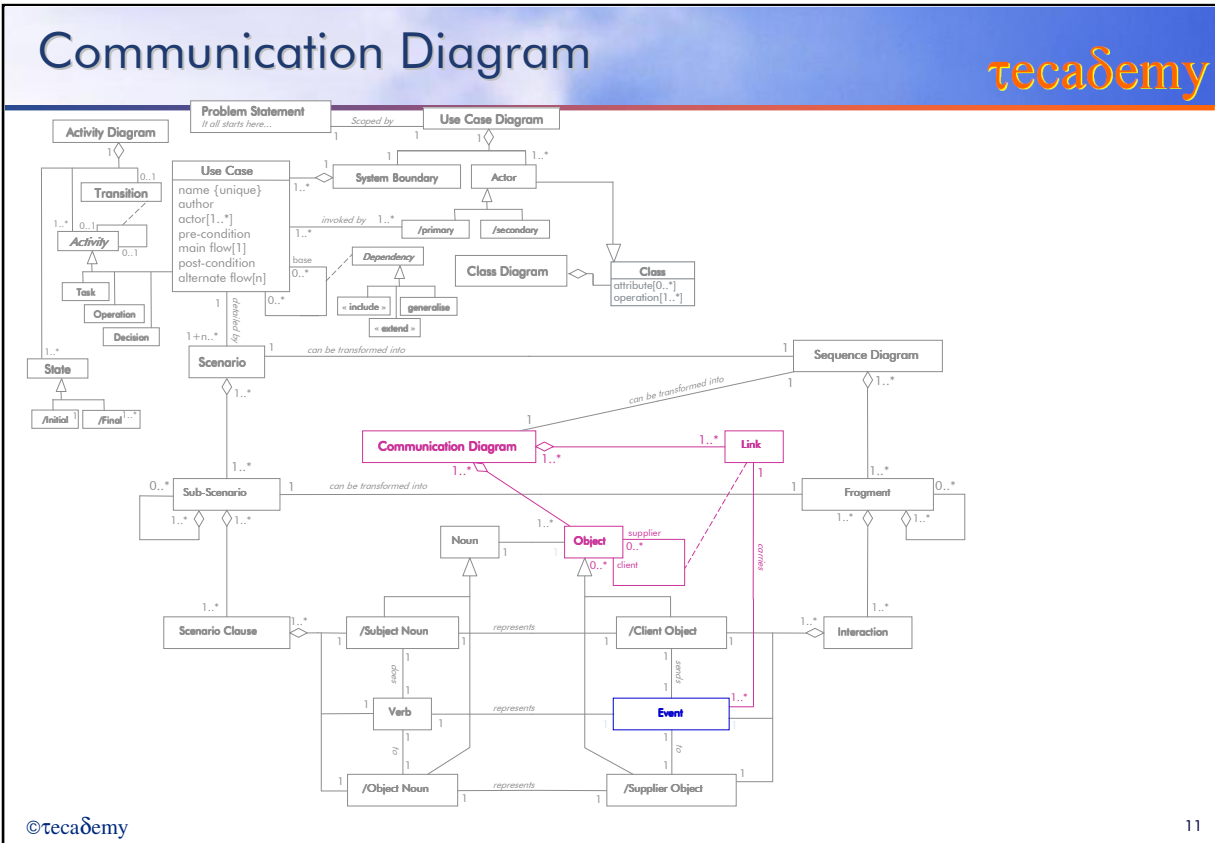


And just as a scenario clause can map to a single object interaction, a scenario can be represented as one sequence diagram. They are simply different representations of the same information – one textual, one graphical.



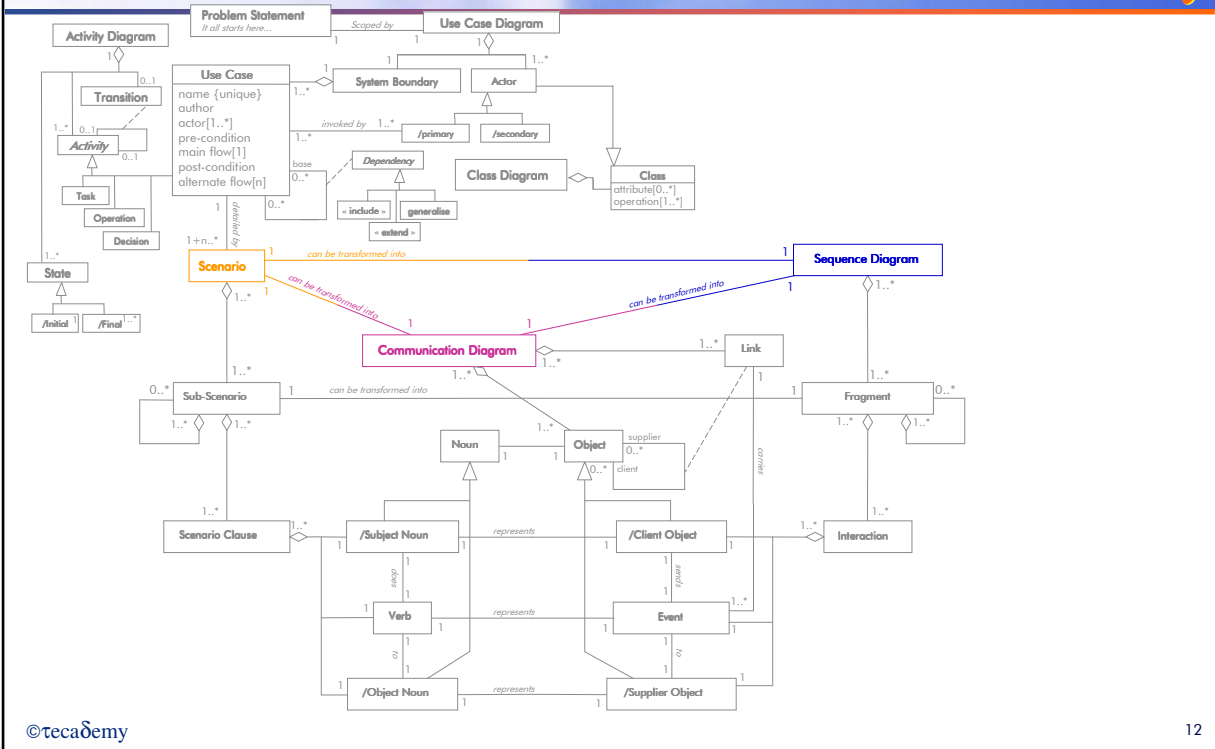
If a sequence diagram represents the temporal view of a scenario, a *Communication Diagram* shows the structural view. They are both different projections of an underlying three-dimensional model (*pace* the Fence-post Model – the sequence diagram is the fence viewed from the front, the Communication Diagram the fence viewed from the top).



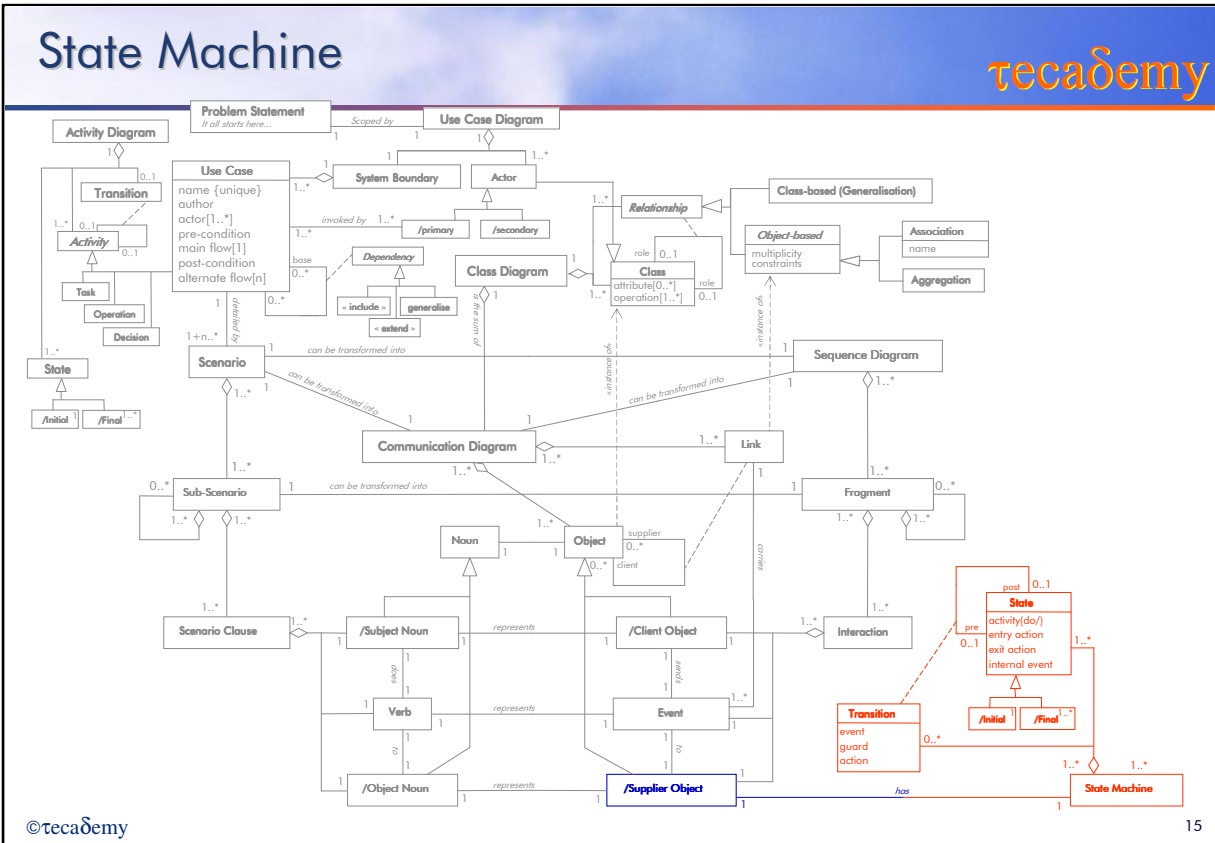


The Communication Diagram shows objects and the links between them. The links carry messages between the objects.

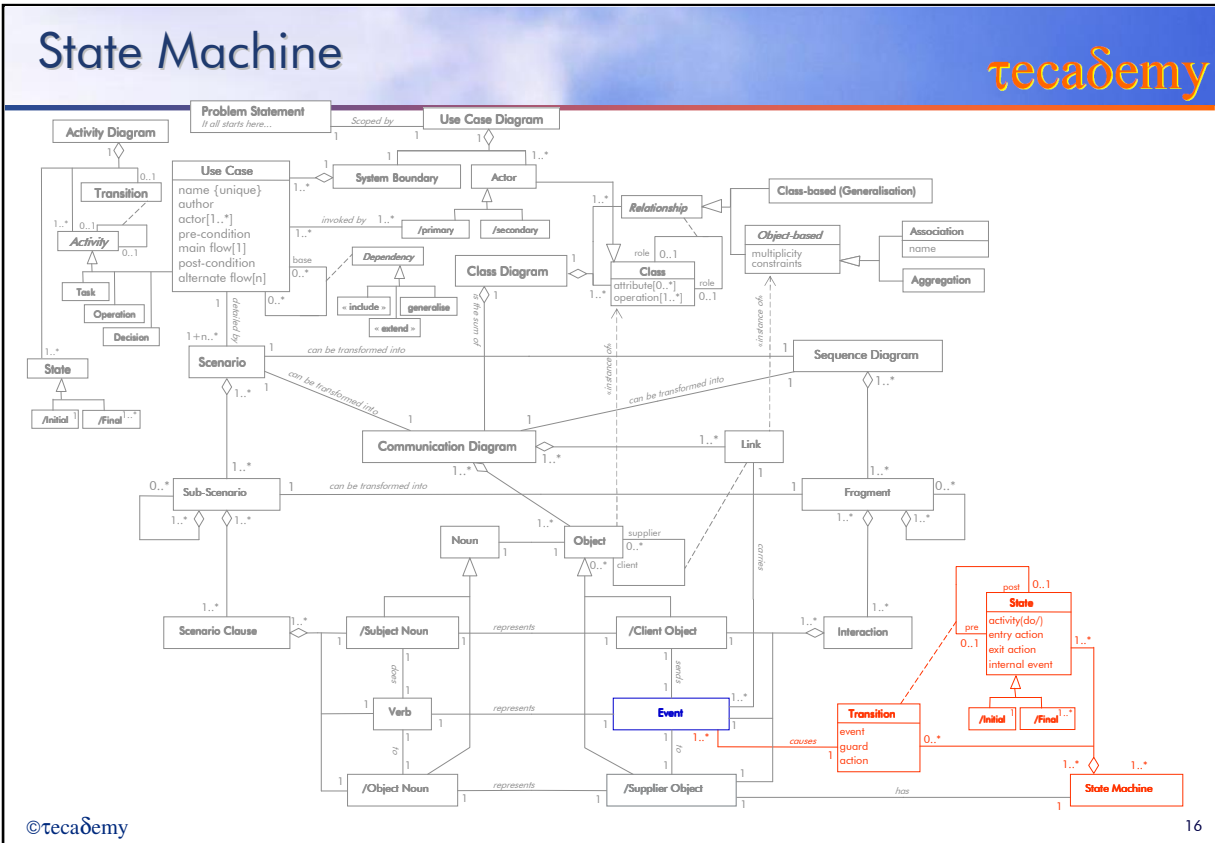
Three of a Kind



Scenarios, sequence diagrams and communication diagrams are three different views of the same thing – i.e. one specific instance of a use case.

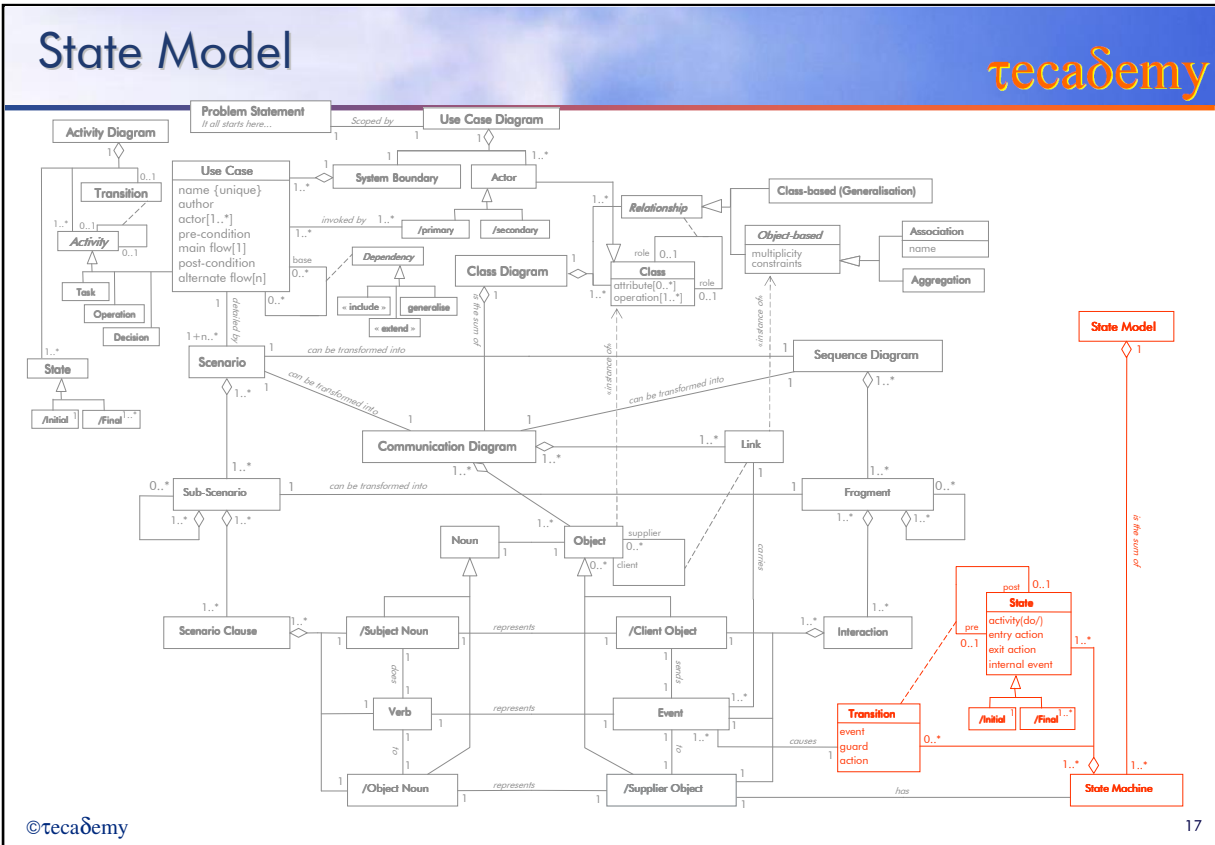


Every object responds to message events in some way – every object has a life-cycle. An object is born, it does stuff, it dies. The object’s life cycle can be shown as a *state machine* – a diagram showing the states the object occupies and the transitions from one state to another.

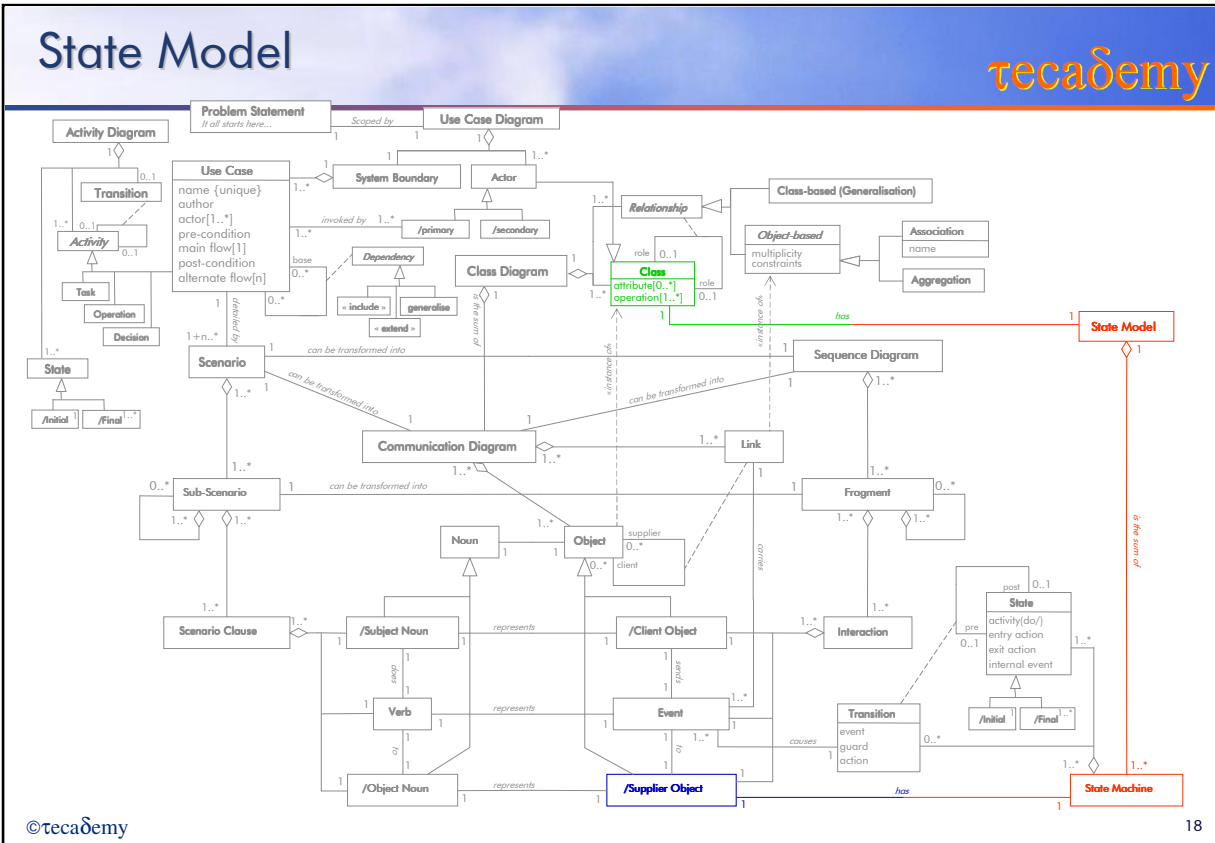


A transition from one state to another occurs because of an event (such as a message or response) received by the object of interest.

If the whole system is considered as a single object, the events that cause it to transition from one state to another are the use case invocations; the states represent the pre- and postconditions for those use cases.



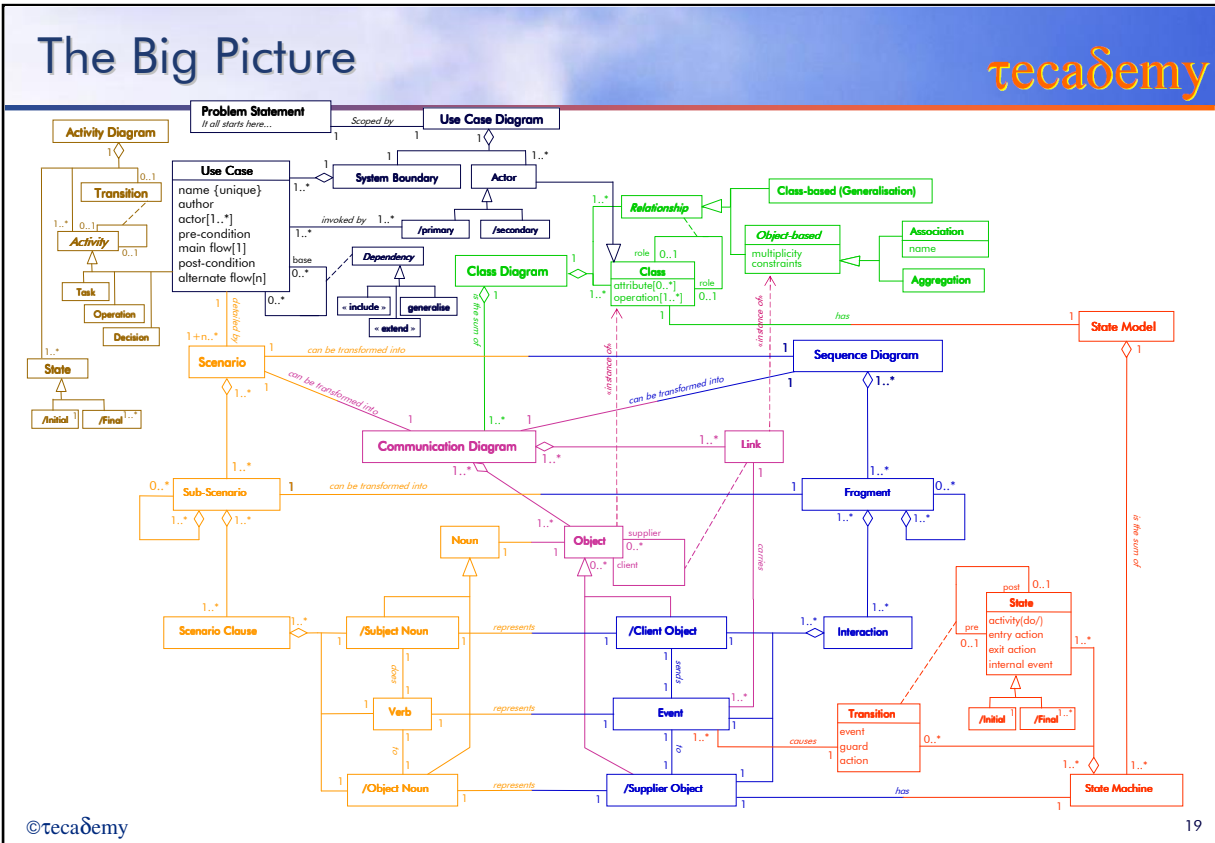
A state model shows all possible lifecycles for an object or a system.



The state *machine* shows the lifecycle for one particular object – the state *model* shows all possible lifecycles for a whole class of objects.

All classes have a state model. We model only those with interesting or complex behaviour.

Every state is defined in terms of the values of an object’s attributes. State modelling helps to identify the attributes a class must provide for its objects in order for them to ‘know’ what state they are in. It also helps to identify the operations a class must provide in order for an object to respond to all of the message events it might receive.



In the interests of simplicity (!!!) this diagram is not complete – but it covers the most common UML elements and models. Please feel free to add to it at your leisure!

The model above demonstrates how the UML can be used to represent knowledge – even fairly complex knowledge. The use of colour can help to make a complex model more accessible. Unfortunately, the limitations of the production process for this handout precludes the use of colour - ask your lecturer how to get the full-colour version!

Notes