

Sequence diagrams

Modeling interactions between objects

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Interaction diagrams

- Interaction diagrams are models that describe how groups of objects interact to realise some behaviour.
- Typically we use an interaction diagram to capture the realisation of a single use case scenario.
- The diagram shows a subset of the objects of the system and the messages that are passed between these objects during the scenario execution.
- Two isomorphic syntaxes exist :
 - Sequence diagrams : Emphasize message sequence
 - Collaboration diagrams : Emphasize object structure

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Recap : Procedural execution model

- In the procedural model, program execution is decomposed into a hierarchy of procedures and sub-procedures.
- A thread of control is described by the sequential execution of statements and nested procedure calls in which the calling procedure is blocked until the called procedure terminates.
- In a single-thread program, only one procedure executes at any given moment.
- In a multi-thread program, we use asynchronous signals to call a procedure, without blocking the caller. The caller continues executing concurrently with the called procedure.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Object-oriented execution model

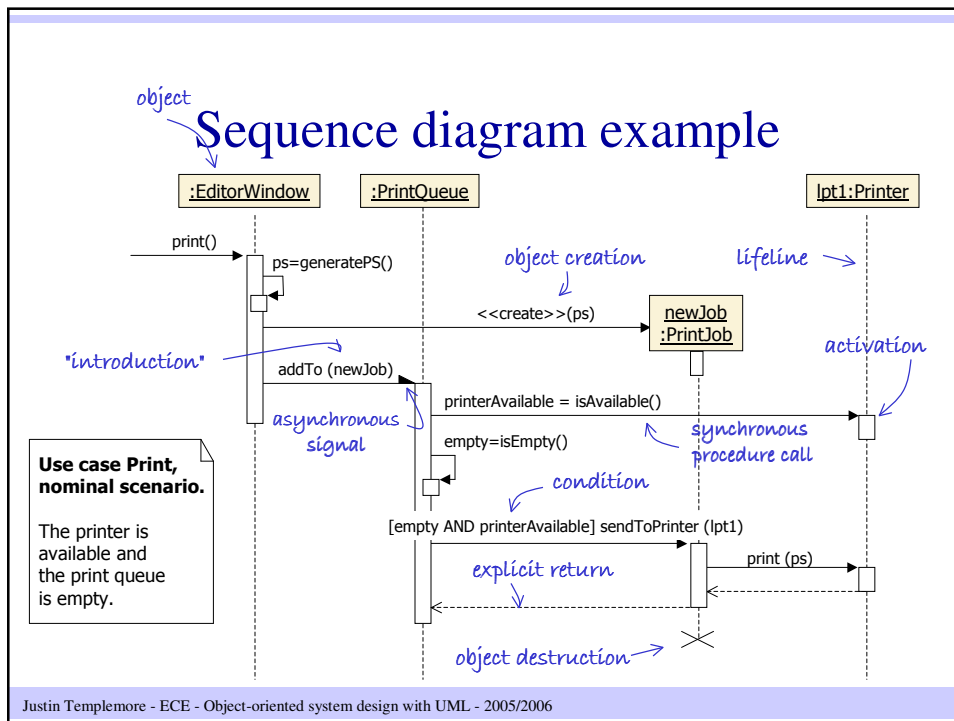
- Object-oriented execution is similar to the procedural execution model, with some differences:
 - Procedures are **structurally distributed** amongst objects according to responsibility.
 - A procedure call is performed by **sending a message** to the object which "owns" a procedure.
 - For two objects to be able to exchange messages, they must be structurally associated, or "introduced" during the interaction.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Activations

- Calling the procedure of an object **activates** that object. The duration of the procedure execution is called an **activation**.
- In a single-thread model, only one object is activated and executing at any given moment
 - although other objects may be activated and waiting
- In a multi-thread model, multiple objects may be active and executing simultaneously
 - leading to multiple activations of a single object.
 - A synchronised object may only have one activation at any given time.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006



Object

- An **object** is shown as a box at the top of a dashed vertical line called its **lifeline**.
- The box contains a label of the form **instanceName:className**.
 - **Must** be underlined to distinguish from a class declaration.
 - The instance name is a reference that can be passed between objects in messages, and may be omitted if the precise identity of an object is unimportant
- The **lifeline** shows the object's life during the interaction
 - objects may be created and destroyed during the interaction

Object with lifeline

lpt1:Printer

Class

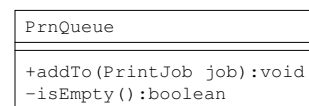
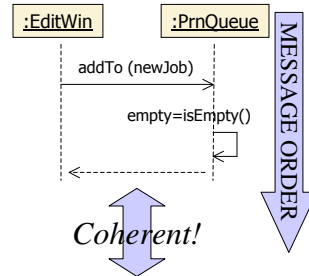
Printer

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Messages

- A **message** is shown by an **arrow** between the lifelines (or activations) of two objects.
- The message is declared as an **operation** in the receiving object's class.
- A message may specify :
 - the procedure name (obligatory)
 - the message arguments
 - a return value
- The **order** in which messages are sent are shown from top to bottom on the page.
- A **self call**, shown by a message returning to the sending object, indicates that an object calls one of its own procedures
 - shows local procedural decomposition

Sequence diagram



Class diagram

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

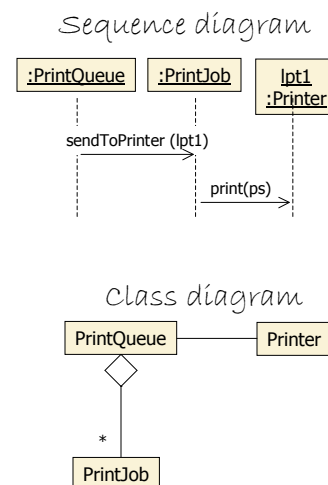
Messages

- UML defines three basic message types
 - **Simple message** : Message execution model unspecified.
 - **Procedure call** : A synchronous message that blocks the caller while the called procedure executes.
 - **Signal** : An asynchronous message that does not block the caller. The caller continues to execute concurrently with the called procedure.
- **Return** : Shows the return of a procedure call.
 - Not a new message.
 - Signals return immediately with no result.
 - Returns are implicit at the end of a procedure call, and should only be shown if they add clarity to the diagram.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Reference parameters

- Procedure call arguments may be **value** or **reference** parameters
 - A reference parameter serves to "introduce" one object to another,
 - allowing the receiving object to send messages to the introduced object for the duration of the interaction,
 - even if these two object's classes are not associated.



Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Message control information

- A message may specify optional message information.
- A **condition** shows when a message is sent. The message is sent only if the condition is true (if).

[empty AND printerAvailable] sendToPrinter (lpt1) →

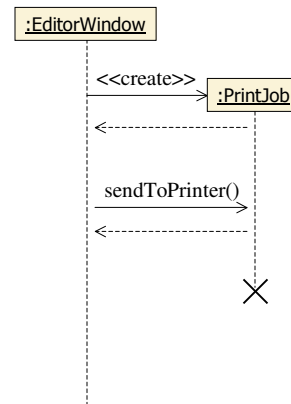
- An **iteration marker** shows that a message is sent many times to multiple receiver objects (loop).

*[all PrintJobs in queue] cancel () →

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Object creation and destruction

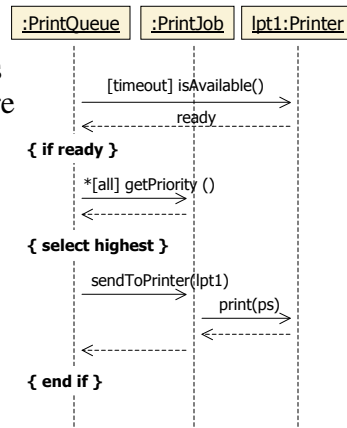
- Most objects exist prior to and continue to exist after an interaction.
- **Creation** of an object during the interaction is shown by a **<<create>>** message directed at the object box.
 - Stereotyped message that represents a constructor; may have parameters.
- **Deletion** of an object during the interaction is shown by a large **X** interrupting the lifeline.
 - An object may commit suicide, or be destroyed by a message.



Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Lifeline control annotation

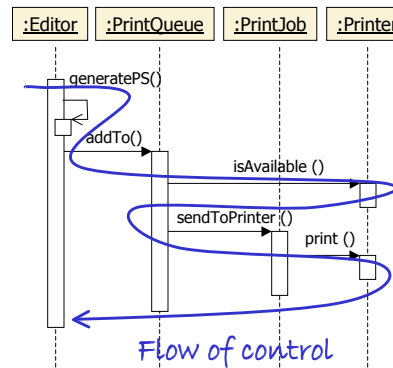
- We can describe the internal decisions and control mechanisms of a procedure using comments on the lifeline of the object
 - { **this is a comment** }
- This practice is **discouraged** because it complicates the diagram.
 - A sequence diagram should emphasize message order and control flow
- The **algorithm** of an individual procedure is more appropriately described on an **activity diagram**.



Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Activations and flow of control

- An **activation** is shown as a white box on an object's life-line. It shows when that object is active.
- Reception of a message activates an object, which remains active while processing or waiting for nested procedures call to return.
- A procedure implicitly returns control to the calling object at the end of its activation.
- Activations are useful for showing **nested procedure calls**: The message arrow leaves the activation rectangle and not the lifeline.



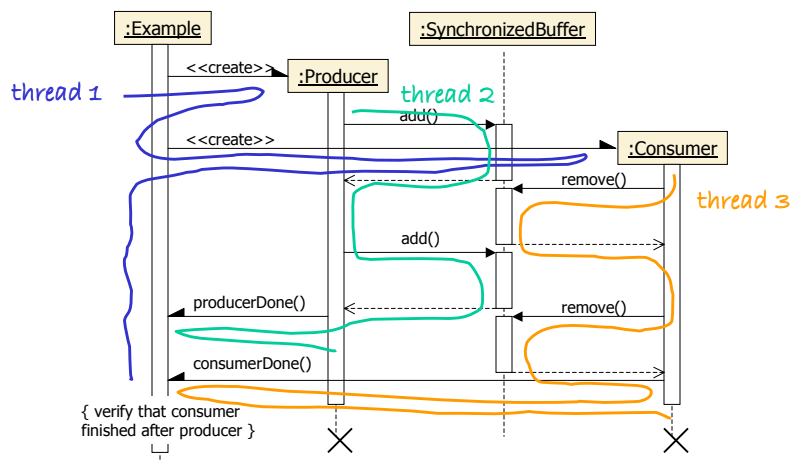
Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Signals and multi-threading

- A signal does not block the caller, so it can continue with its own processing while the called object is also activated.
- A signal can do one of 3 things:
 - Create a new activation, which will execute concurrently with the sending object's activation.
 - Shown by a signal linked to the top of an activation;
 - Communicate with an activation that is already running.
 - Shown by a signal arriving on an existing activation.
 - Create a new active object
 - Shown by a signal pointing to an object box.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Multi-thread example



Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Collaboration diagrams

An interaction diagram notation
emphasizing structure.

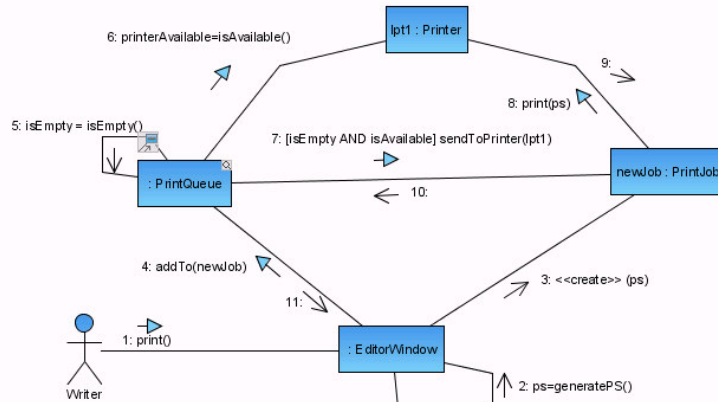
Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Collaboration diagram

- Collaboration diagrams are a second form of interaction diagram equivalent to sequence diagrams,
- But placing the emphasis on structure, rather than message sequence.
- A collaboration diagram shows both objects and instantiated associations between objects.
- Messages are shown as arrows placed along the instantiated associations.
- Message order is shown by a numbering scheme.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Collaboration diagram example



Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Message numbering

- Messages can be simply numbered in ascending order (as in example).
 - Does not show message nesting
- The decimal numbering scheme uses a new decimal point when a message is nested.
 - 1 {1.1, 1.2}, 2, 3 {3.1, 3.2 {3.2.1, 3.2.2}, 3.3}, 4, etc.
- Message order is never as clear as in a sequence diagram.

Justin Templemore - ECE - Object-oriented system design with UML - 2005/2006

Conclusion

- Sequence and collaboration diagrams are equivalent; use whichever suits you best (try both).
 - If object structure is important : Collaboration diagram
 - If message sequence is important : Sequence diagram
- The strength of these diagrams is the clear way in which they show message and control flow between objects, so keep them uncomplicated
 - Show one use case scenario per diagram
 - Detail operation algorithms on activity diagrams
 - Minimise your usage of message control structures
- Don't forget to declare messages used in the sequence diagram on your class diagram.