UNIT-V

A Framework for Object-Oriented Data Flow Testing

MADHESWARI.K
AP/CSE
SSNCE

Topics to be covered

Event-/Message-Driven Petri Nets
Inheritance-Induced Data Flow

Message-Induced Data Flow

Slices

Data flow testing

DD-path testing is often insufficient; and in such cases, data flow testing is more appropriate.

The same holds for integration testing of object-oriented software; if anything, the

need is greater for two reasons: (1) data can get values from inheritance tree and (2) data can be defined at various stages of message passing.

Event-/Message-Driven Petri Nets

- Event-driven Petri nets used to express the message communication among objects
- Figure 15.14 shows the notational symbols used in an event and message-driven Petri net (EMDPN).
-

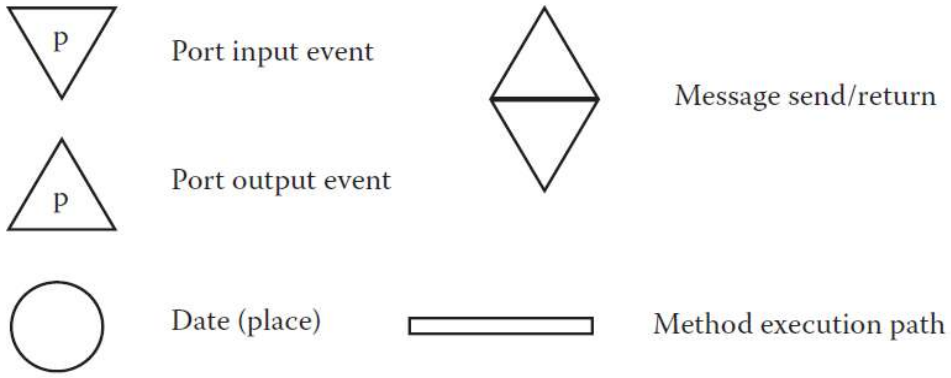


Figure 15.14 Symbols for event-/message-driven Petri nets (E/MDPN).

Event-/Message-Driven Petri Nets

- The fused triangle shape for messages is intended to convey that a message is an output of the sending method and an input to the destination method.

Definition

An *event- and message-driven Petri net (EMDPN)* is a quadripartite directed graph (P, D, M, S, In, Out) composed of four sets of nodes, P, D, M, and S, and two mappings, In and Out, where
P is a set of port events
D is a set of data places
M is a set of message places
S is a set of transitions
In is a set of ordered pairs from $(P \cup D \cup M) \times S$
Out is a set of ordered pairs from $S \times (P \cup D \cup M)$

Event-/Message-Driven Petri Nets

- We retain the port input and output events because these will certainly occur in event-driven, object-oriented applications.
- Obviously, we still need data places, and we will interpret Petri net transitions as method execution paths.

The new symbol is intended to capture the essence of inter object messages:

- They are an output of a method execution path in the sending object.
- They are an input to a method execution path in the receiving object.
- The return is a very subtle output of a method execution path in the receiving object.
- The return is an input to a method execution path in the sending object.

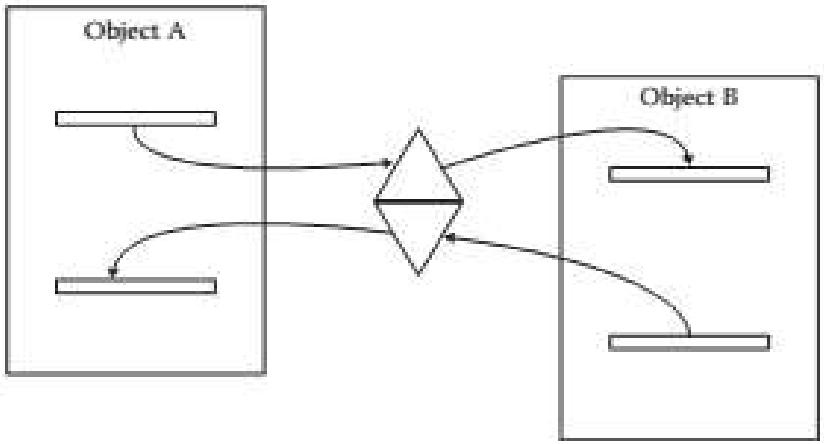


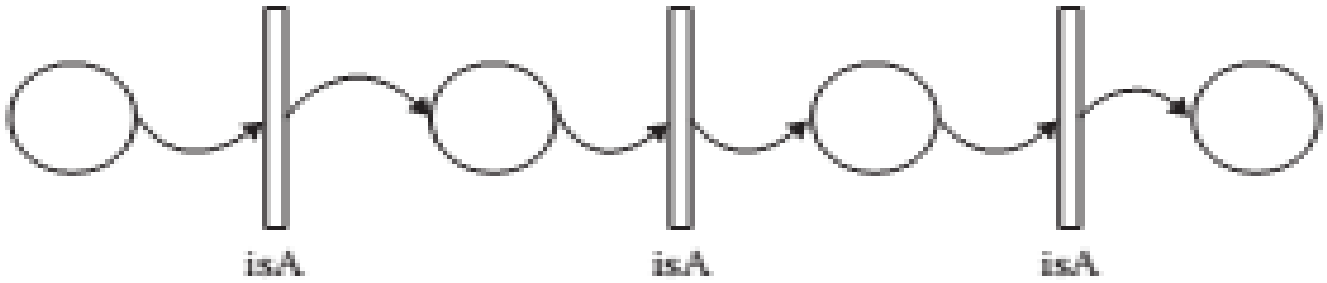
Figure 15.15 Message connection between objects.

Event-/Message-Driven Petri Nets

- The EMDPN structure, because it is a directed graph, provides the needed framework for data flow analysis of object-oriented software.
- Recall that data flow analysis for procedural code centers on nodes where values are defined and used.
- In the EMDPN framework, data is represented by a data place, and values are defined and used in method execution paths.
- A data place can be either an input to or an output of a method execution path; therefore, we can now represent the define/ use paths (du-path) in a way very similar to that for procedural code.
- Even though four types of nodes exist, we still have paths among them; so we simply ignore the types of nodes in a du-path and focus only on the connectivity.

Inheritance-Induced Data Flow

- Consider an inheritance tree in which the value of a data item is defined; in that tree, consider a chain that begins with a data place where the value is defined, and ends at the “bottom” of the tree.
- That chain will be an alternating sequence of data places and degenerate method execution paths, in which the method execution paths implement the inheritance mechanism of the object-oriented language.
- This framework therefore supports several forms of inheritance: single, multiple, and selective multiple.
- The EMDPN that expresses inheritance is composed only of data places and method execution paths, as shown in Figure 15.16.



Message-Induced Data Flow

- The EMDPN in Figure 15.17 shows the message communication among three objects. As an example of a define/use path, suppose mep3 is a Define node for a data item that is passed on to mep5, modified in mep6, and finally used in the Use node mep2. We can identify these two du-paths:

  du1 = <mep3, msg2, mep5, d6, mep6, return(msg2), mep4, return(msg1), mep2>

  du2 = <mep6, return(msg2), mep4, return(msg1), mep2>

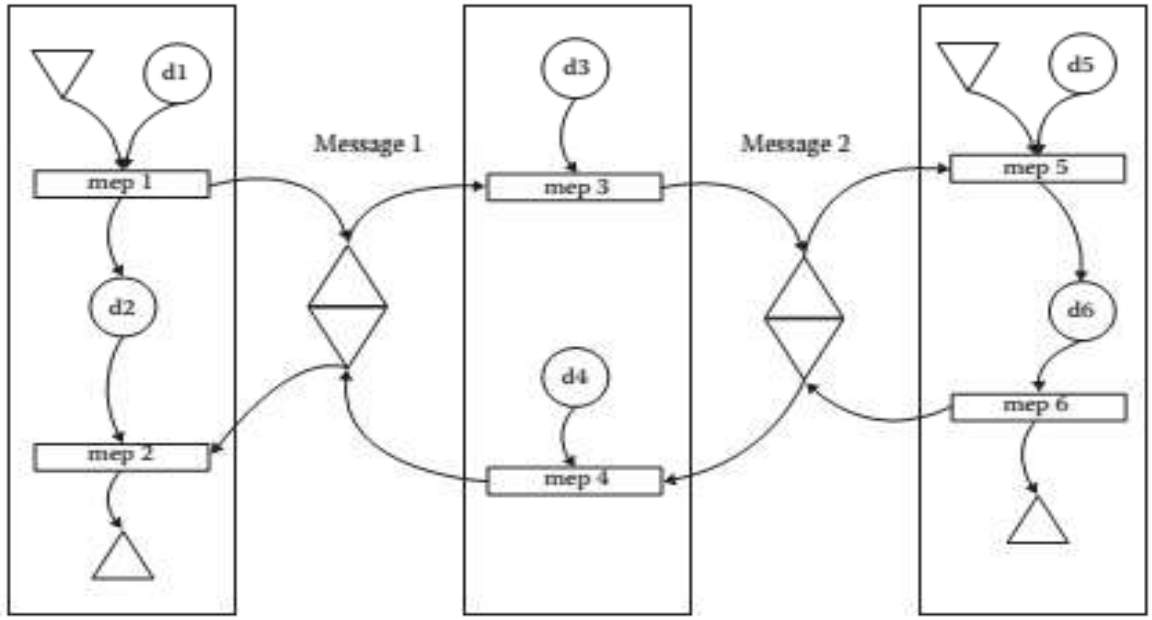


Figure 15.17 Data flow from message passing.

Message-Induced Data Flow

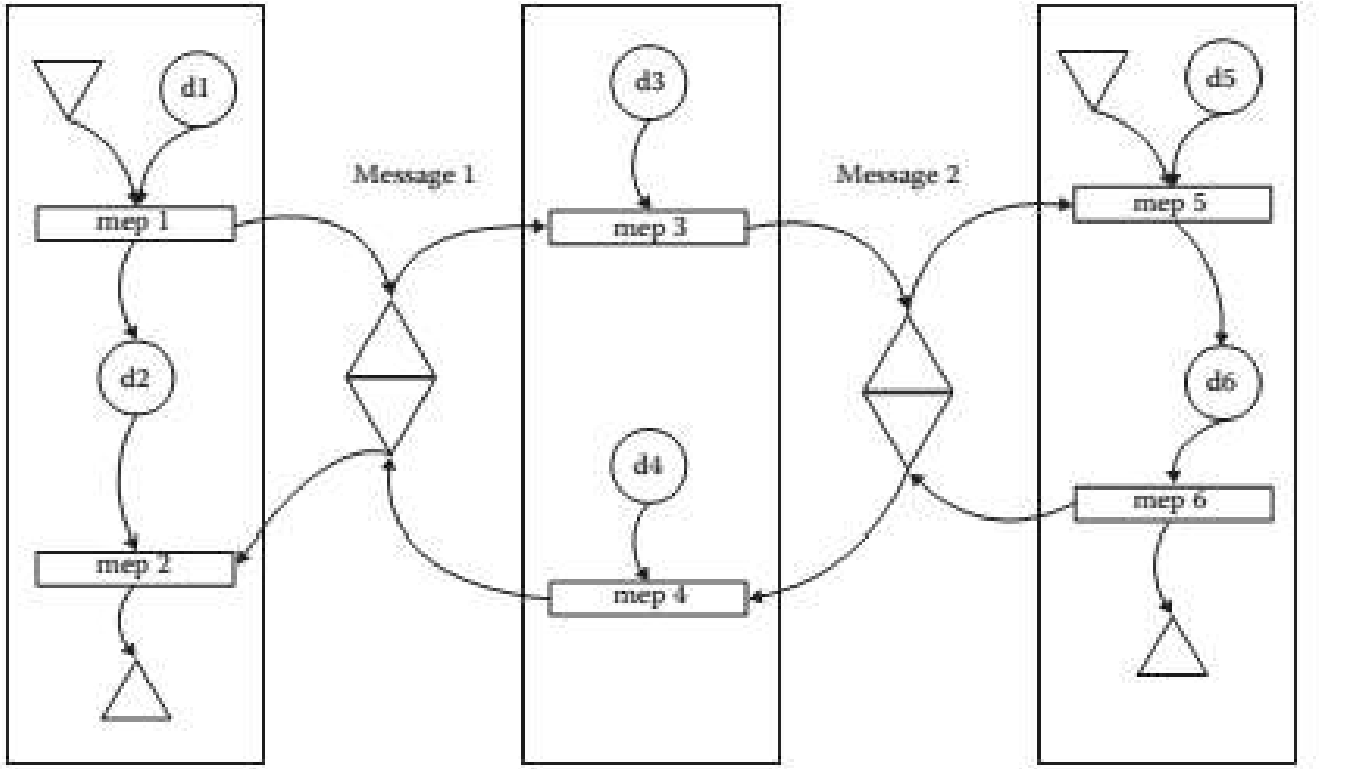


Figure 15.17 Data flow from message passing.

In this example, du2 is definition clear; du1 is not. Although we do not develop data flow testing for object-oriented software here, this formulation will support that endeavor.

Slices

It is tempting to assert that this formulation also supports slices in object-oriented software. The fundamentals are there, so we could go through the graph theory motions. Recall that the more desirable form of a slice is one that is executable. This appears to be a real stretch, and without it, such slices are interesting only as a desk-checking approach to fault location.