Contents(1)

- **Goal and Scope**
- **Basic Concepts on OOT**
  - Basic Concepts to represent the world
  - Basic Concepts for Reuse
  - Information Hiding Principle and Java Program
  - Superiority of OOT
- **Modeling Techniques**
  - Static Model: Class and Association
  - Dynamic Model: State Machine
  - Dynamic Model: Interaction Diagram
  - Concurrency Description: Active Object and Multi-thread Programming
  - Outline of UML2.0

Specific Features of Real-time Systems

- **Timeliness** is important. The system performs its function within specified time limits.
- **Reactive**. The system is continuously responding to events from the external environment that "drives" the execution of the system.
- **Concurrently executing threads of control**, where different parts of the software run in parallel.
- **Very high requirements on most of the nonfunctional requirements** such as reliability, fault tolerance, and performance.
- **Not deterministic**

H.E. Eriksson and M. Penker, "UML Toolkit", John Wiley & Sons, Inc.

Basic Concepts for real-time system modeling

- **time requirement**
- **asynchronous event handling**
- **communication**
- **concurrency**
  - process, thread
- **synchronization**

H.E. Eriksson and M. Penker, “UML Toolkit” John Wiley & Sons, Inc.

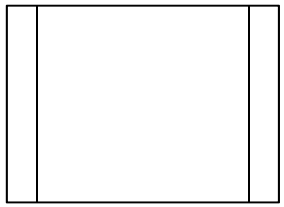
Concurrency in Object-Orientation

- **Explicit concurrency model**
  - Explicit concurrency model describes concurrency separately from the objects by defining processes at an early stage of analysis, and then treating processes and objects as separate modeling entities. The system is decomposed into a number of processes, and each process is internally modeled as an object-oriented system to design the internal structure.
- **Implicit concurrency model**
  - Implicit concurrency model delay the design of concurrency. The system is modeled as objects, where in an early analysis, all objects are considered to have their own execution threads; that is, be active objects. Gradually, through architecture and detailed design, the ideal analysis models are mapped onto an implementation with the support of services from the underlying real-time operating system.
- **UML can support both**
  - It has better support for the implicit concurrency model. Active classes, asynchronous communication, and synchronization can be modeled in early phases and gradually be translated into the services and capabilities of the implementation environment.

H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Active Class and Object

Active Class



UML2.0

<<Active Class>>
Communication Supervisor

Active Object

aninstance :
Communication Supervisor

An active class is a one that owns an execution thread and can initiate control activity

H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Thread of Control

- Thread: Thread of control
  - A sequence of control transfer in a program
- Multi Threads
  - Multiple threads exist together in a program and they can run concurrently

S. Oaks, H. Wong, "Java Threads", O'REILLY, 1997.

Thread using Thread class

```
public class OurClass {
  public void run() {
     for(int i = 0; i < 100; i++) {
        System.out.println("Hello");
     }
  }
}
```

The run() method that writes a string 100 times to standard output is defined in OurClass

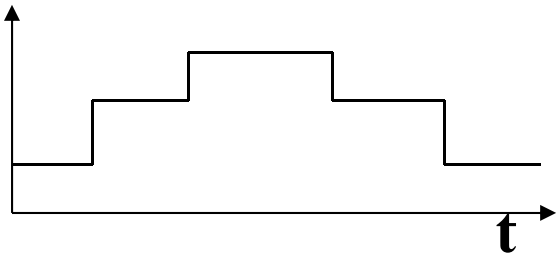
```
Import java.applet.Applet;
public class OurApplet extends Applet{
  public void init() {
     ourClass oc = new OurClass();
       oc.run();
  }
}
```

Applet thread works by executing run() method in Applet

Applet executes run()

Applet executes init()

Execution of Applet thread



This example simply shows a method invocation as shown in the left figure.

S. Oaks, H. Wong, "Java Threads", O'REILLY, 1997.

Execute run() method of OurClass concurrently with init() or the other Applet methods

```
public class OurClass extends Thread{
  public void run() {
     for(int i = 0; i < 100; i++) {
       System.out.println("Hello");
     } } }
```

Implementation of start() method is in a Thread class or its super class

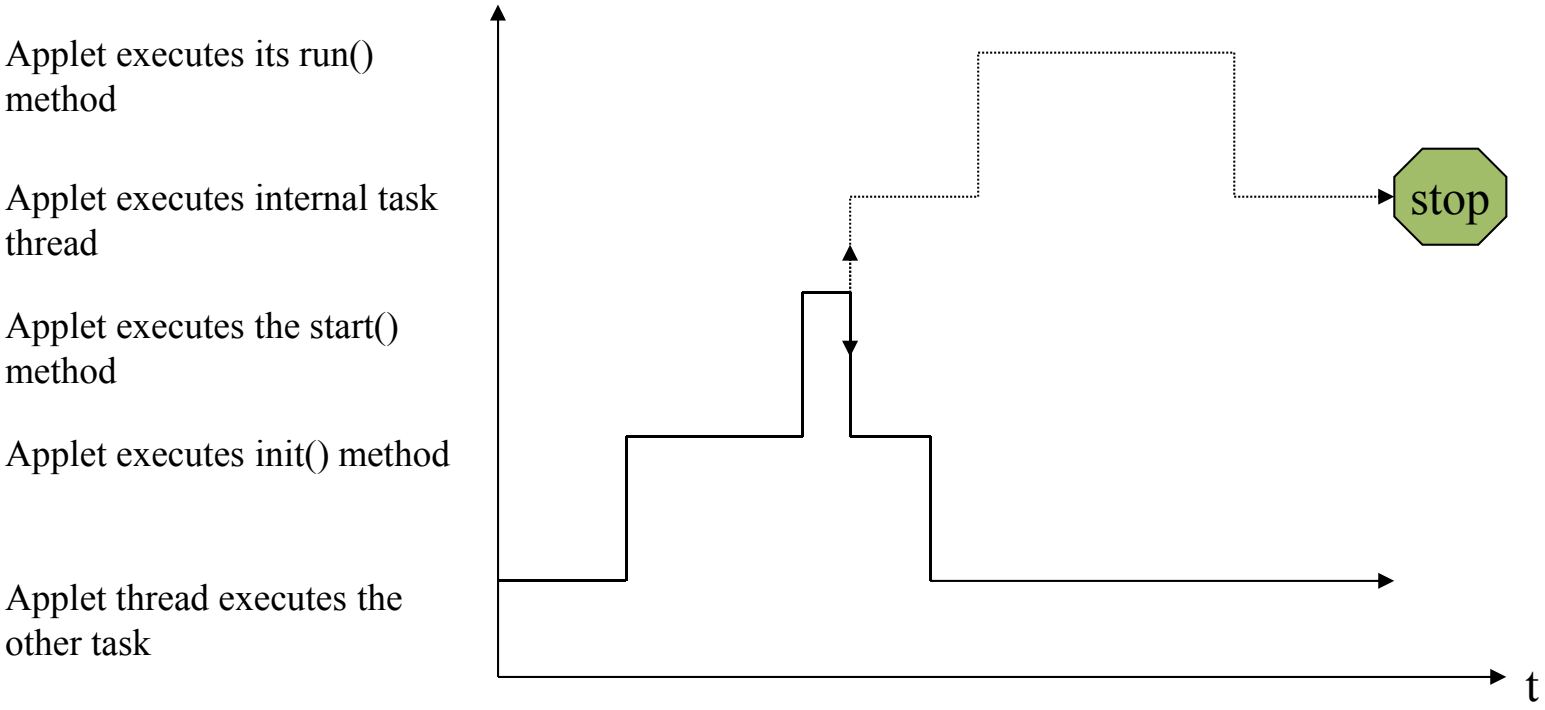
```
Import java.applet.Applet;
public class OurApplet extends Applet{
  public void init() {
     ourClass oc = new OurClass();
       oc.start();
  }
}
```

start() method invokes run() method directly or indirectly.

start() method creates the new thread of control

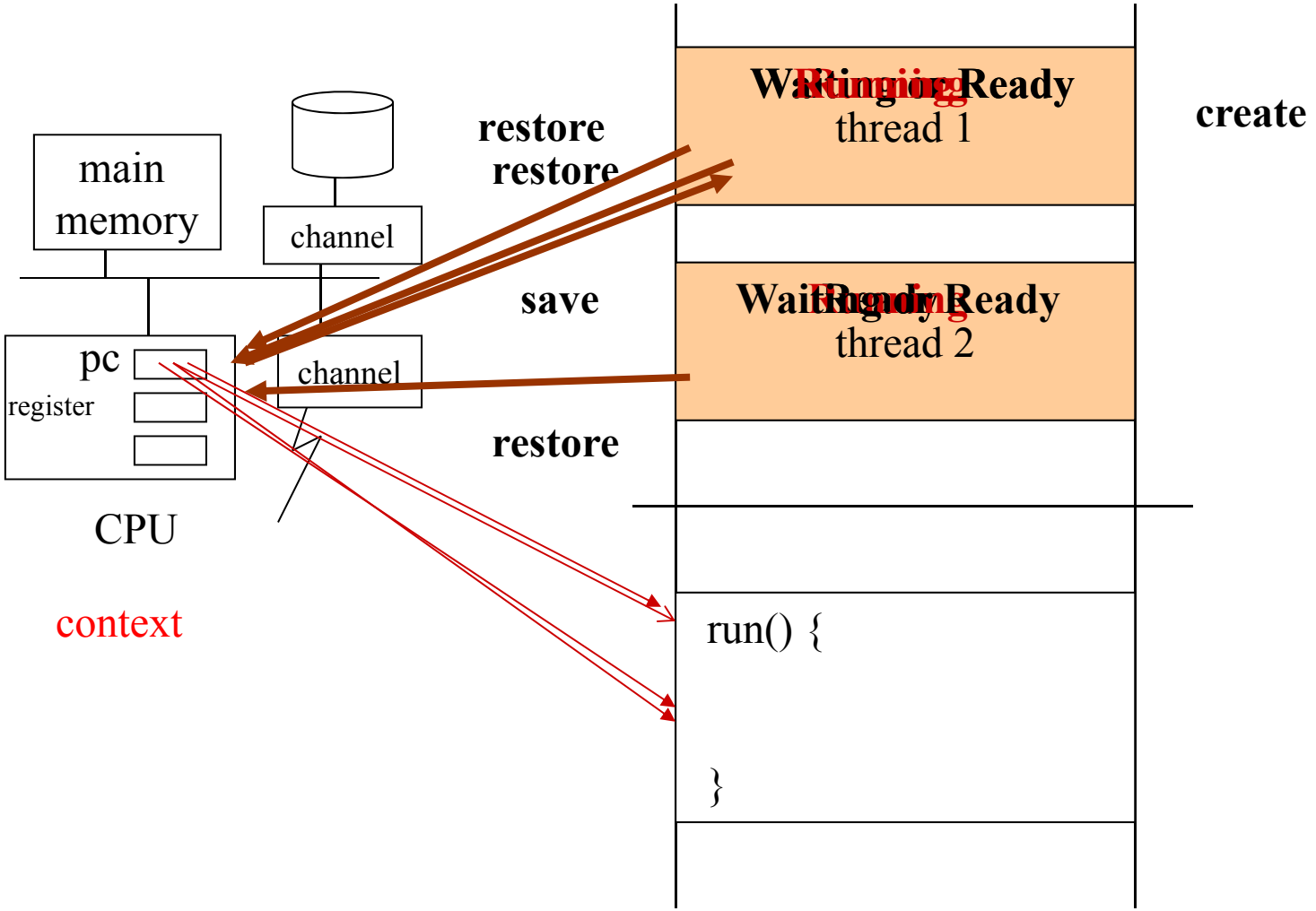
S. Oaks, H. Wong, "Java Threads", O'REILLY, 1997.

Execute run() method of OurClass concurrently with init() or the other Applet methods



S. Oaks, H. Wong, "Java Threads", O'REILLY, 1997.

Run-time environment view of threads

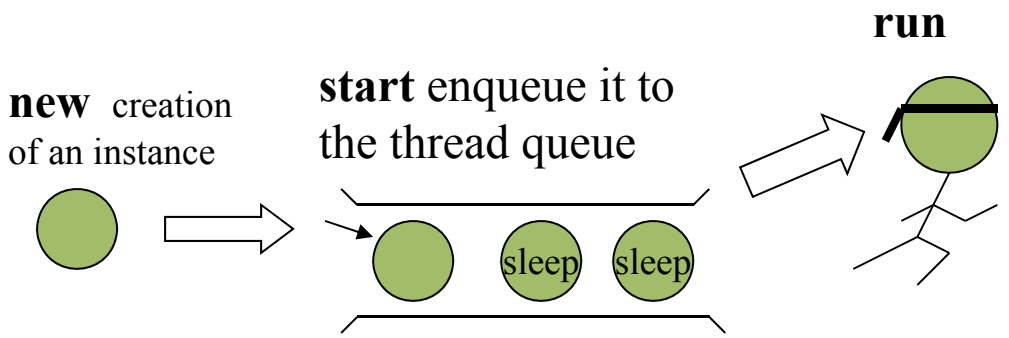


Methods of the Thread class

- **Thread()** : creates a thread object using default values for all option
- **void run()** : the method that is executed by the newly created thread. It can be considered as a main() of the newly created thread. This method should be over-ridded by the code that is executed by the new thread.
- **void start()** : creates a new thread and executes run() method that is defined in this thread class

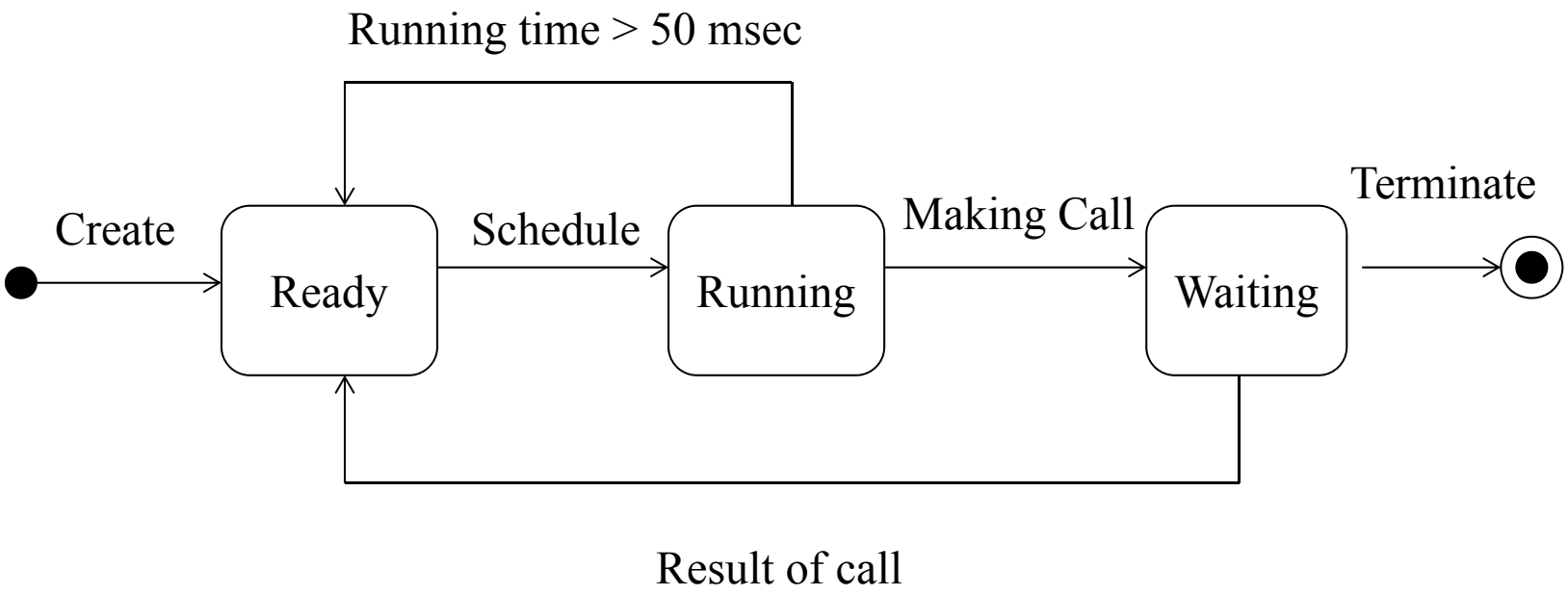
```
public void run() {
   if (target != null) {
       target.run();
}}
```

Default run() method of a Thread object



S. Oaks, H. Wong, "Java Threads", O'REILLY, 1997.

States of an active object



H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Thread using Runnable Interface

```
public interface Runnable{
      public abstract void run();
}
public class OurClass implements Runnable{
  public void run() {
     for(int i = 0; i < 100; i++) {
       System.out.println("Hellow");
     } }}
```

Runnable interface only have a run() method

```
Import java.applet.Applet;
public class OurApplet extends Applet{
  public void init() {
     Runnable ot = new OurClass();
     Thread th = new Thread(ot);
       th.start();
  }}
```

S. Oaks, H. Wong, "Java Threads", O'REILLY, 1997.

Synchronization

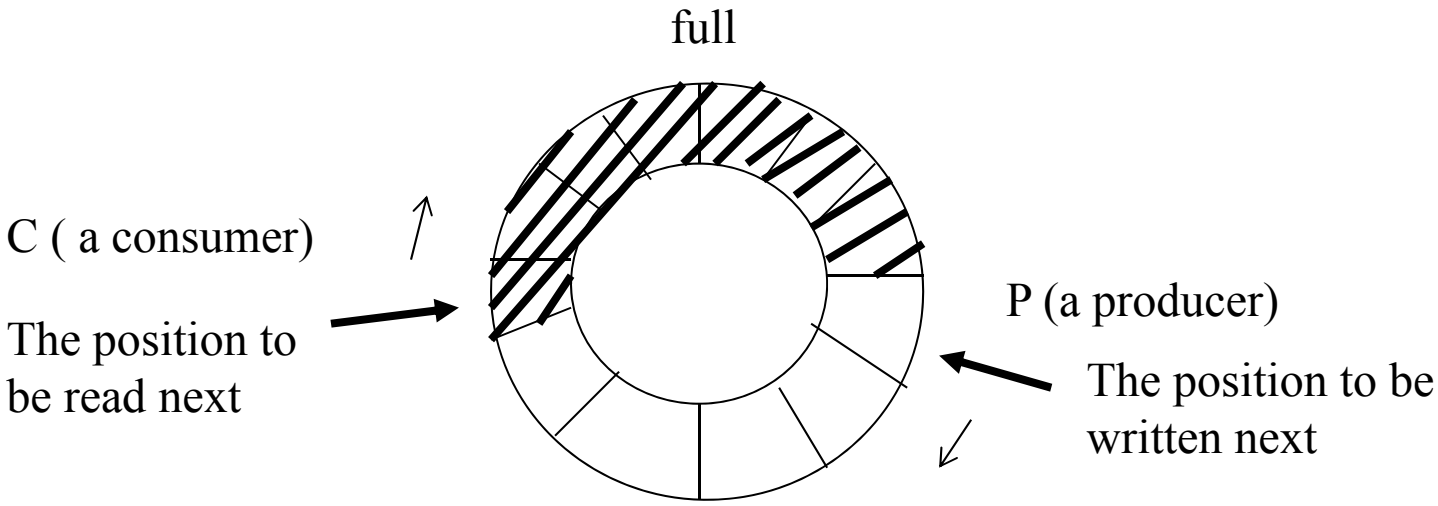
- Concurrent processes interact with each other
  - to exchange data
  - to share resources, referring variables that show the status of the resource
- It cause the occurrence of time-dependent errors
- Synchronization mechanisms are required to avoid errors.
  - Mutual exclusion: operation A and operation B should be executed exclusively
  - Message buffer: A producer of data can not send data more than the finite capacity of a buffer. A consumer of data can not receive data before it is produced.
  - Exchange of timing signals: semaphore invariance
  - **0 <= r(v) <= s(v) + c(v) <= r(v)＋ max(integer)**
  - s(v): number of signals sent, r(v): number of signals received, c(v): initial value of signals, max: maximum countable number of a counter

Problems without synchronization

- **The problems that can occur if synchronization is not properly handled**
  - incorrect shared access: use mutual exclusion between the thread
  - inefficient resource usage: avoid busy-wait
  - dead locks: detection, protection, resolution
  - starvation: thread scheduling

H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Message Buffer



synchronization rule

(1) A consumer can not receive the message before the message is sent

0 <= r < s

(2) A producer can not put the new message into the position before the message In the position is read

0 <= s - r < max

The pointer P can not pass the pointer C and the pointer C can not pass the pointer

Synchronization Supports in UML

- **sequential**
  - The class/operation is intended only for use in a single thread of control
- **guarded**
  - The class/operation will work in the presence of multiple threads of control. The threads normally have to lock the object/operation before using it, and unlock it afterward.
- **synchronized**
  - The class/operation will work in the presence of multiple threads of control; the class itself handles this. Operation can be declared as synchronized.

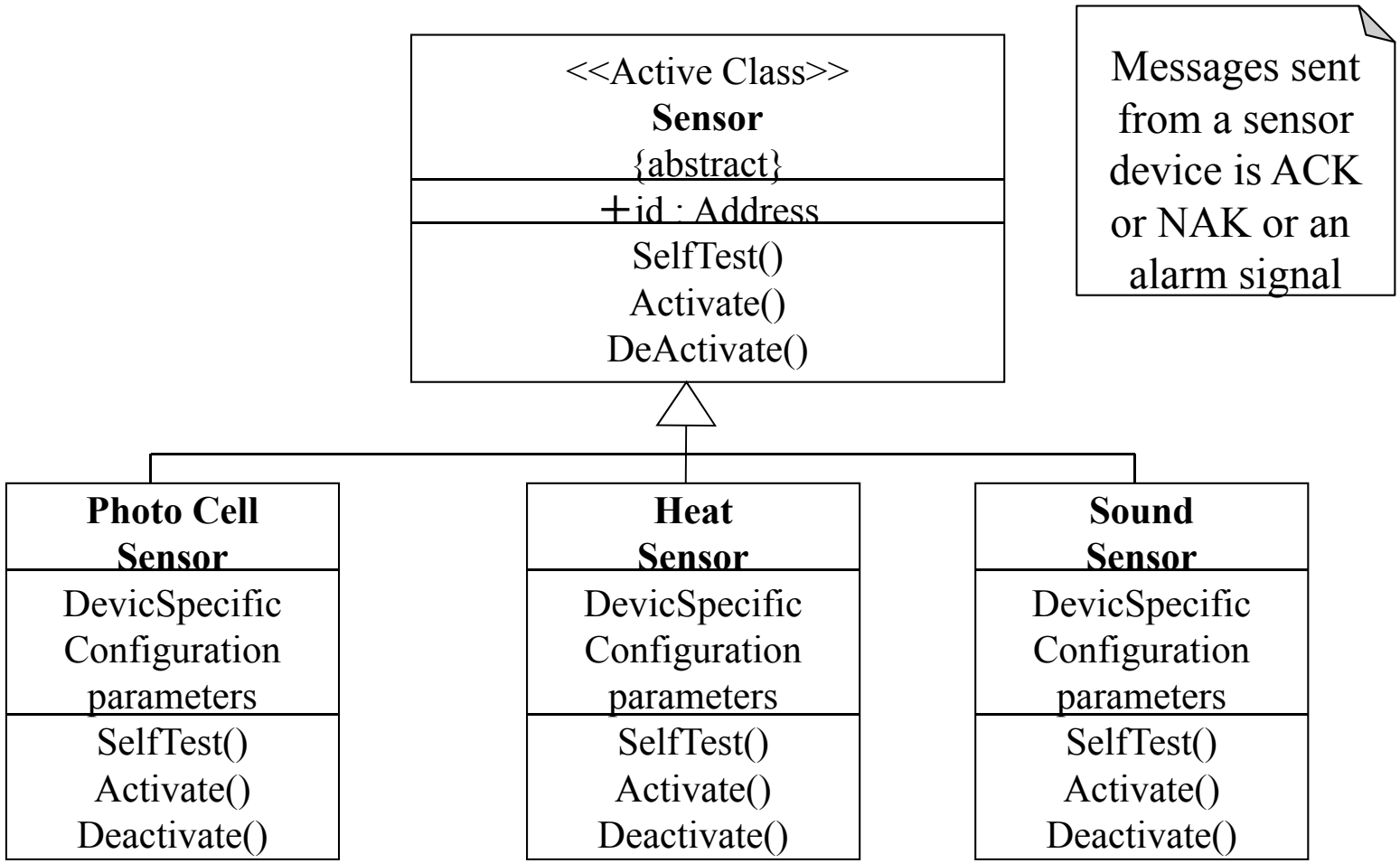
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Basic Mechanisms for Modeling Real-time Systems in UML - house alarm system -

- **Time:** time specifications and constraints are best defined in sequence diagrams
- **Concurrency:** is described as active classes
- **Asynchronous event:** asynchronous message
- **Synchronization:** can be described either as properties of classes or operations(concurrency properties) or as classes/stereotypes that define mechanism such as a semaphore, monitor, or critical region.
- **Distribution:** Thread deployed in a distributed system are described in a deployment diagram.

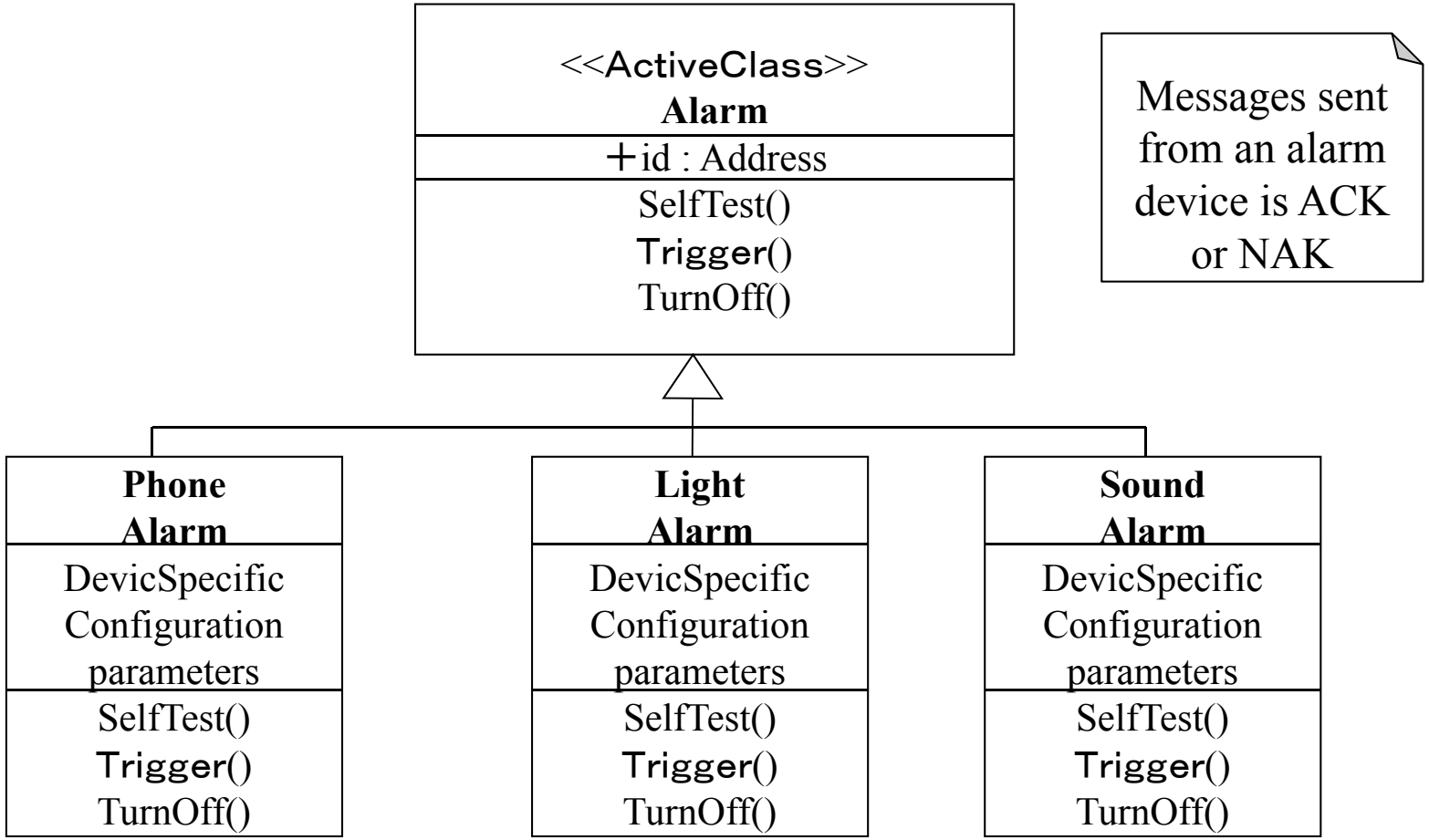
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Sensors



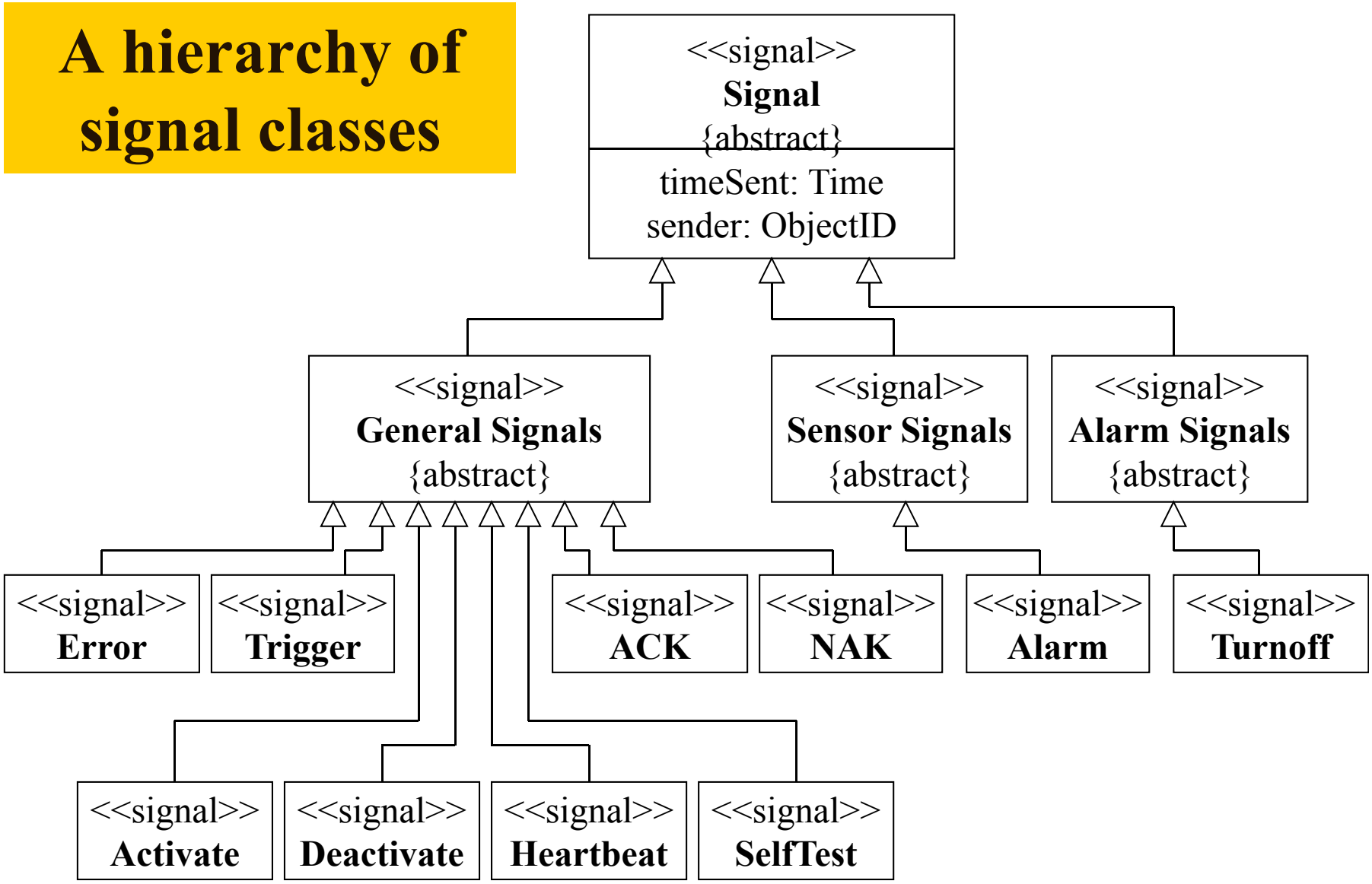
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Alarms



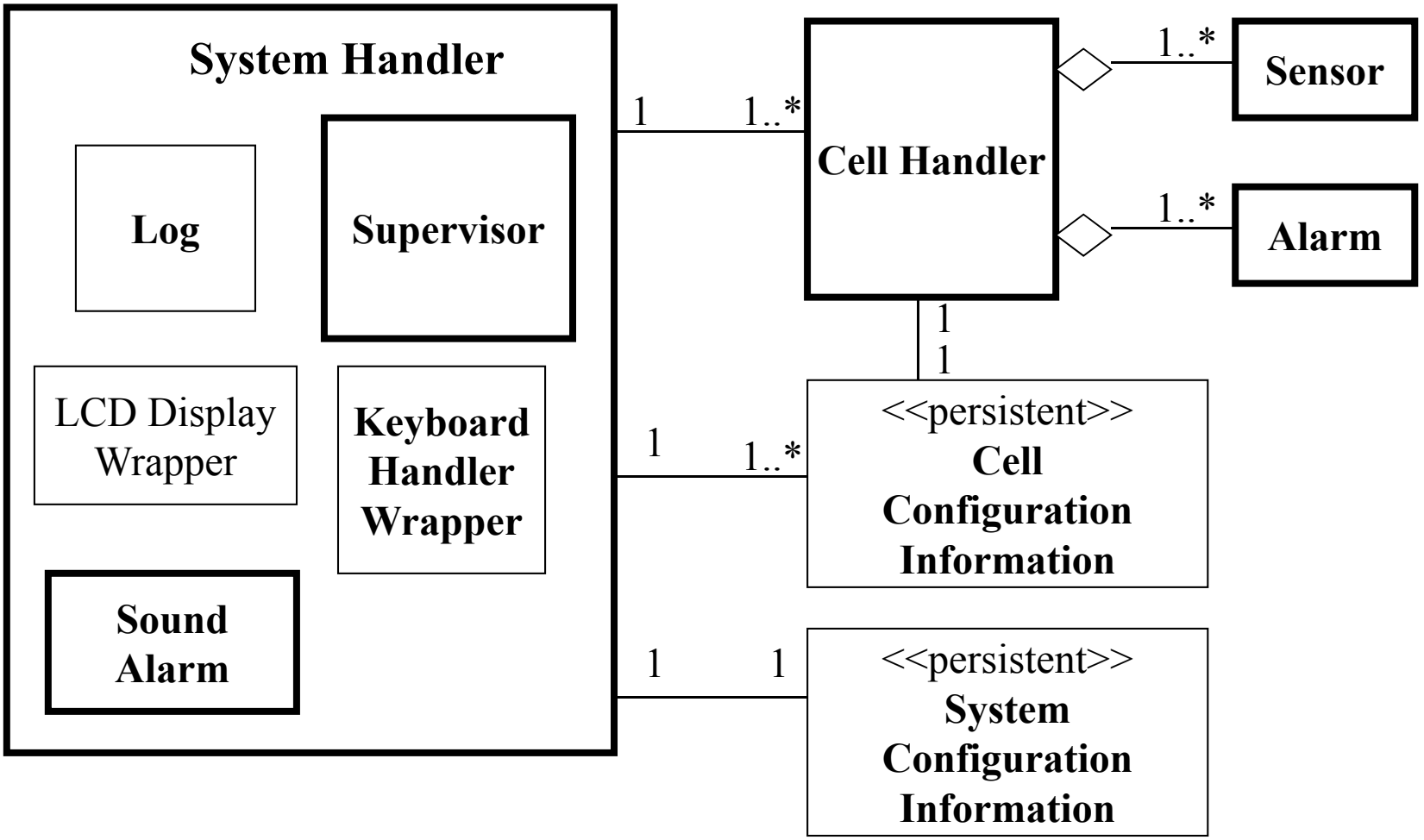
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

A hierarchy of signal classes



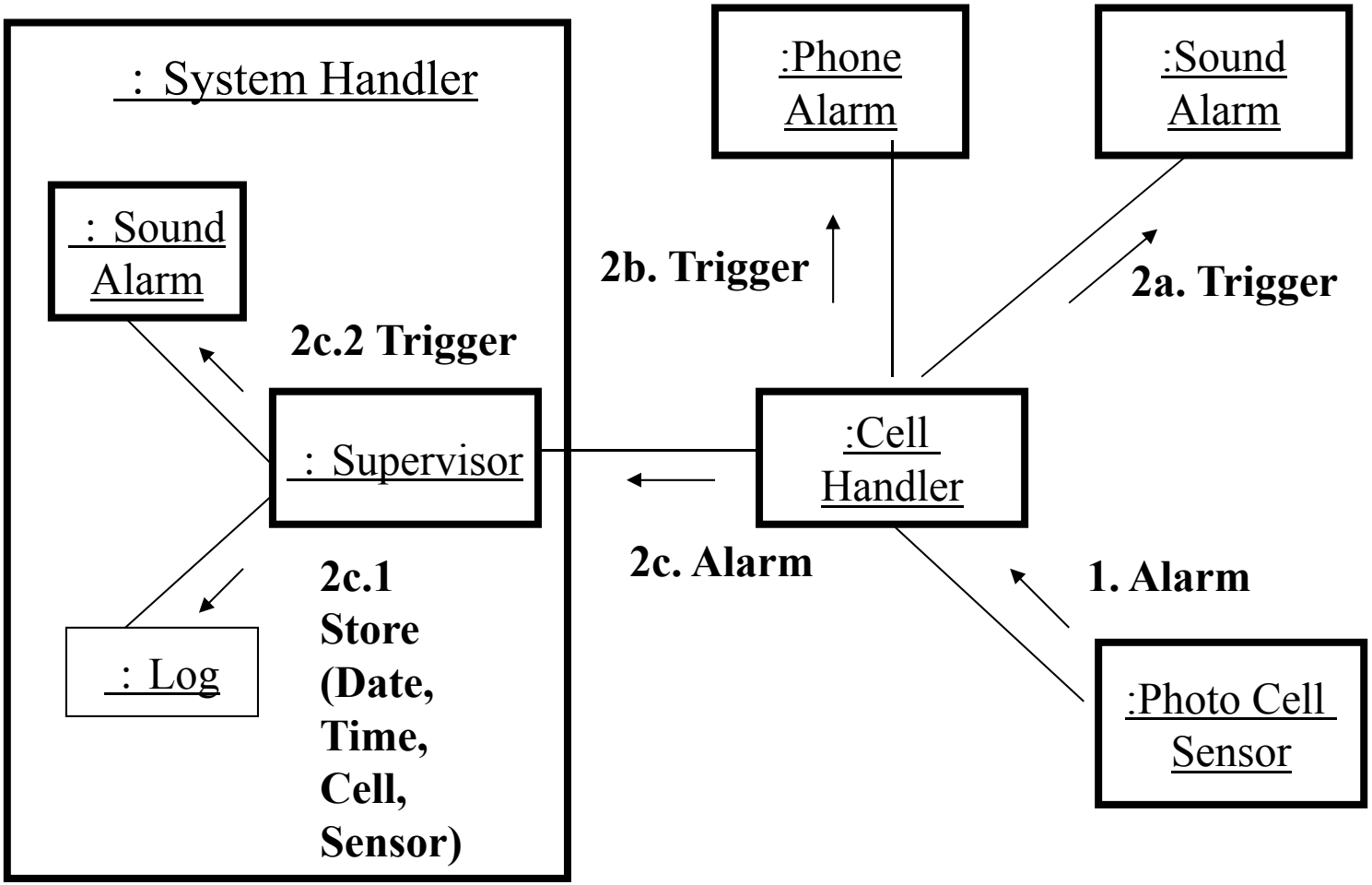
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Class diagram



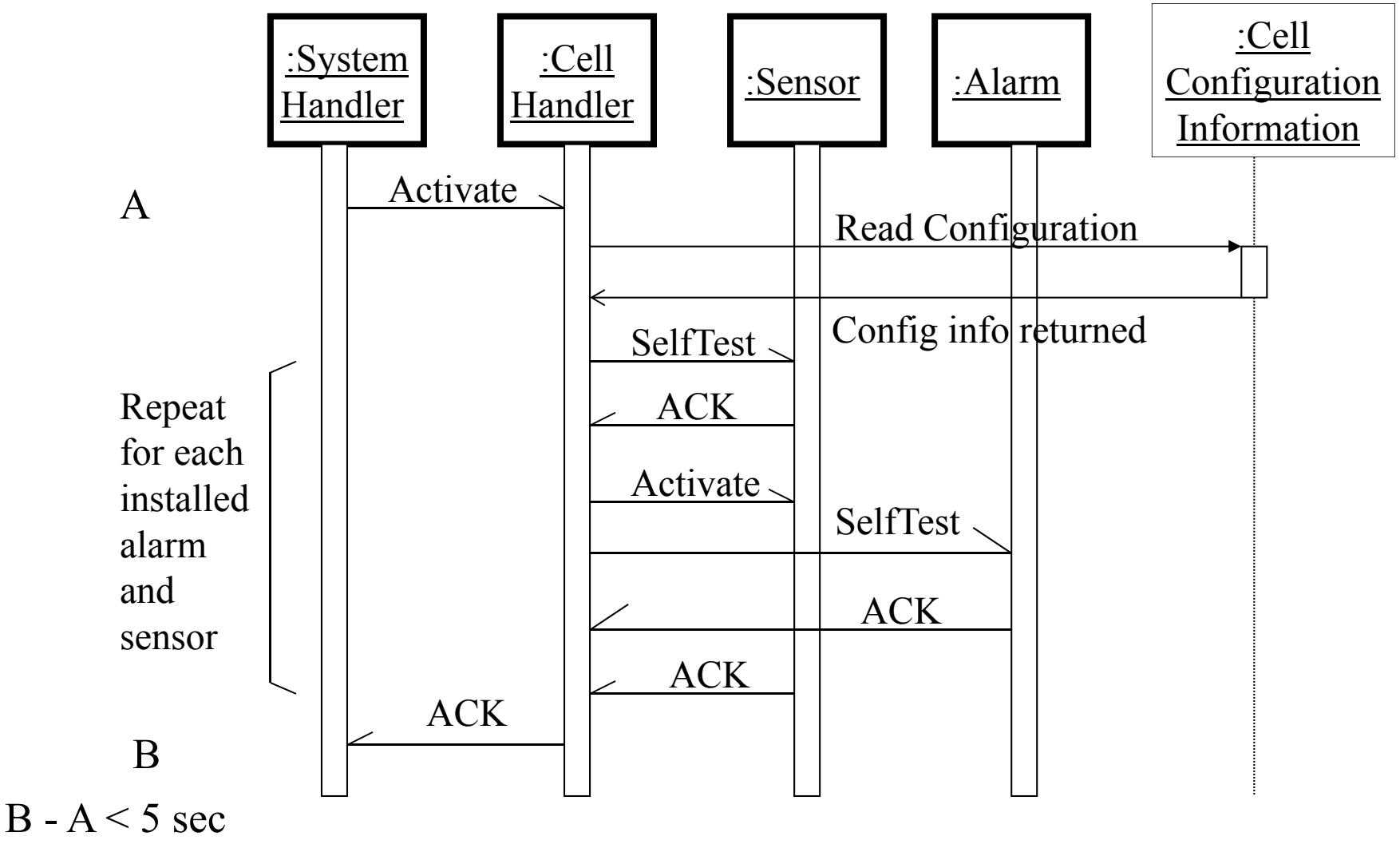
H.E. Eriksson and M. Penker, “UML Toolkit” John Wiley & Sons, Inc.

Collaboration diagram



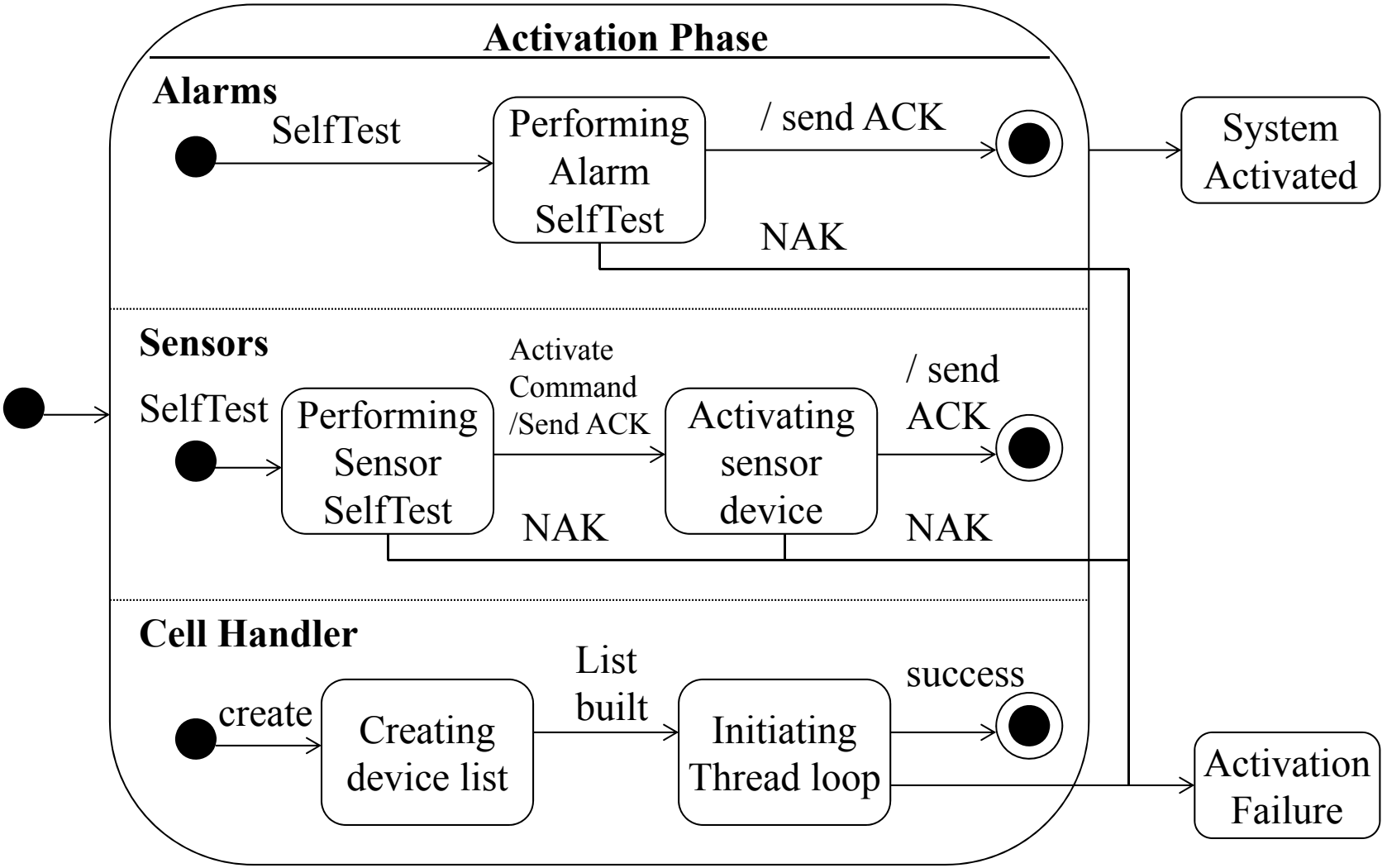
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Sequence diagram



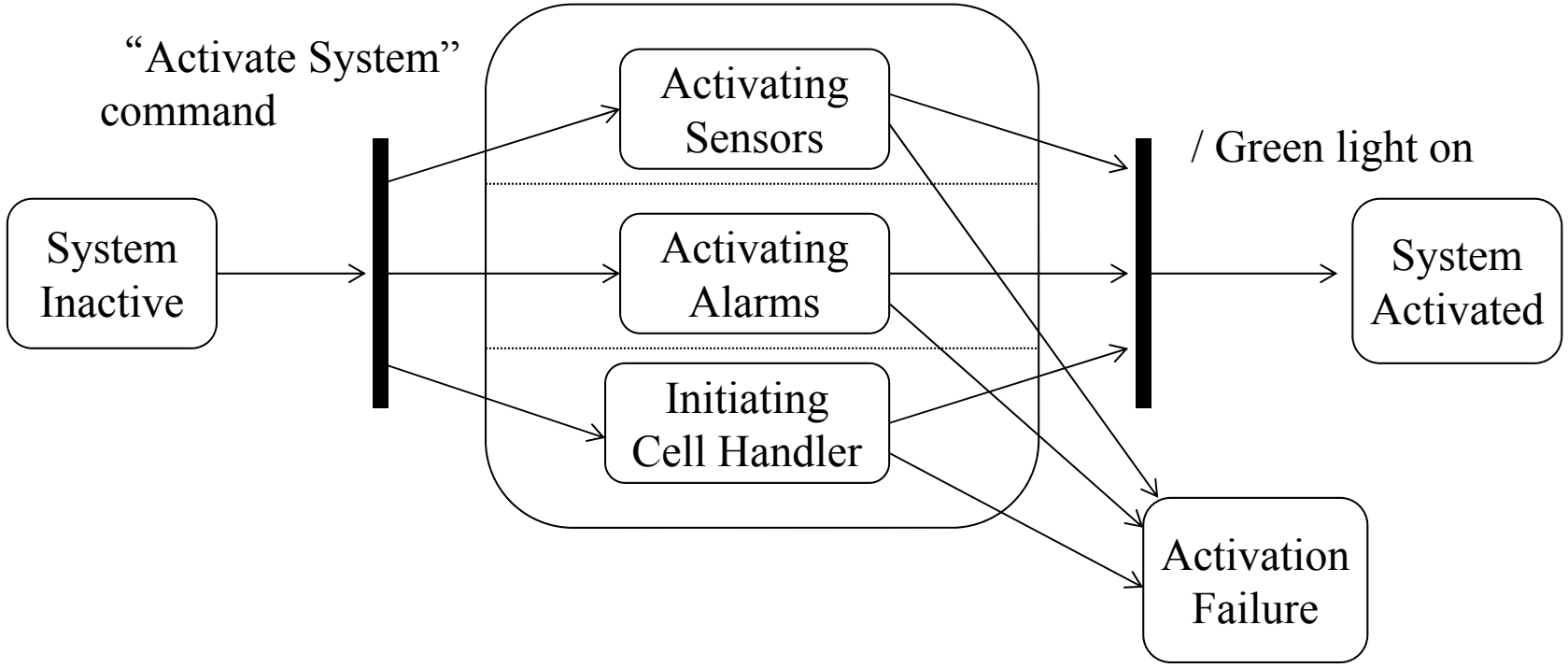
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

State diagram



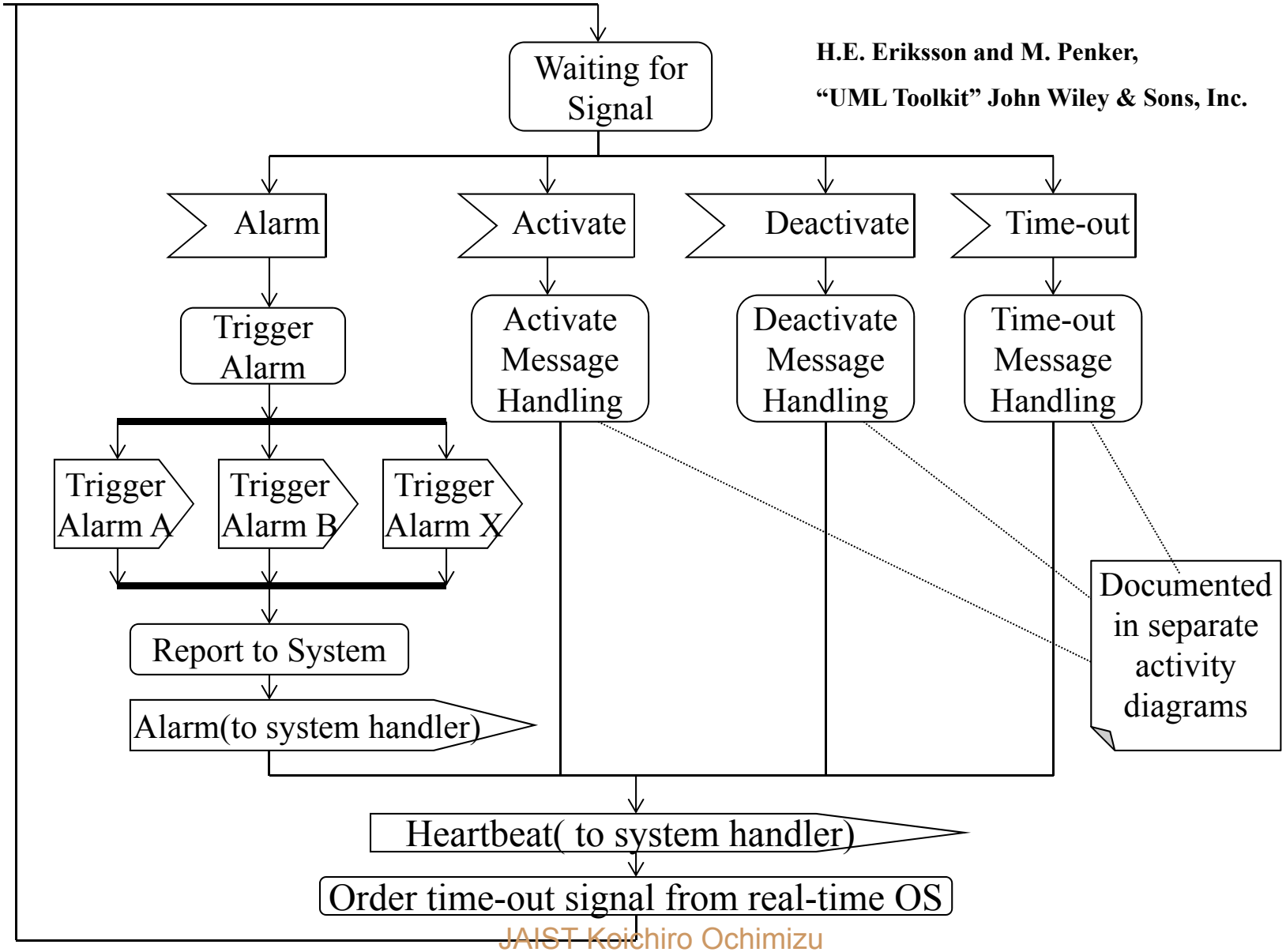
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Control flows are divided into concurrent threads that run in parallel and later become synchronized again



H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

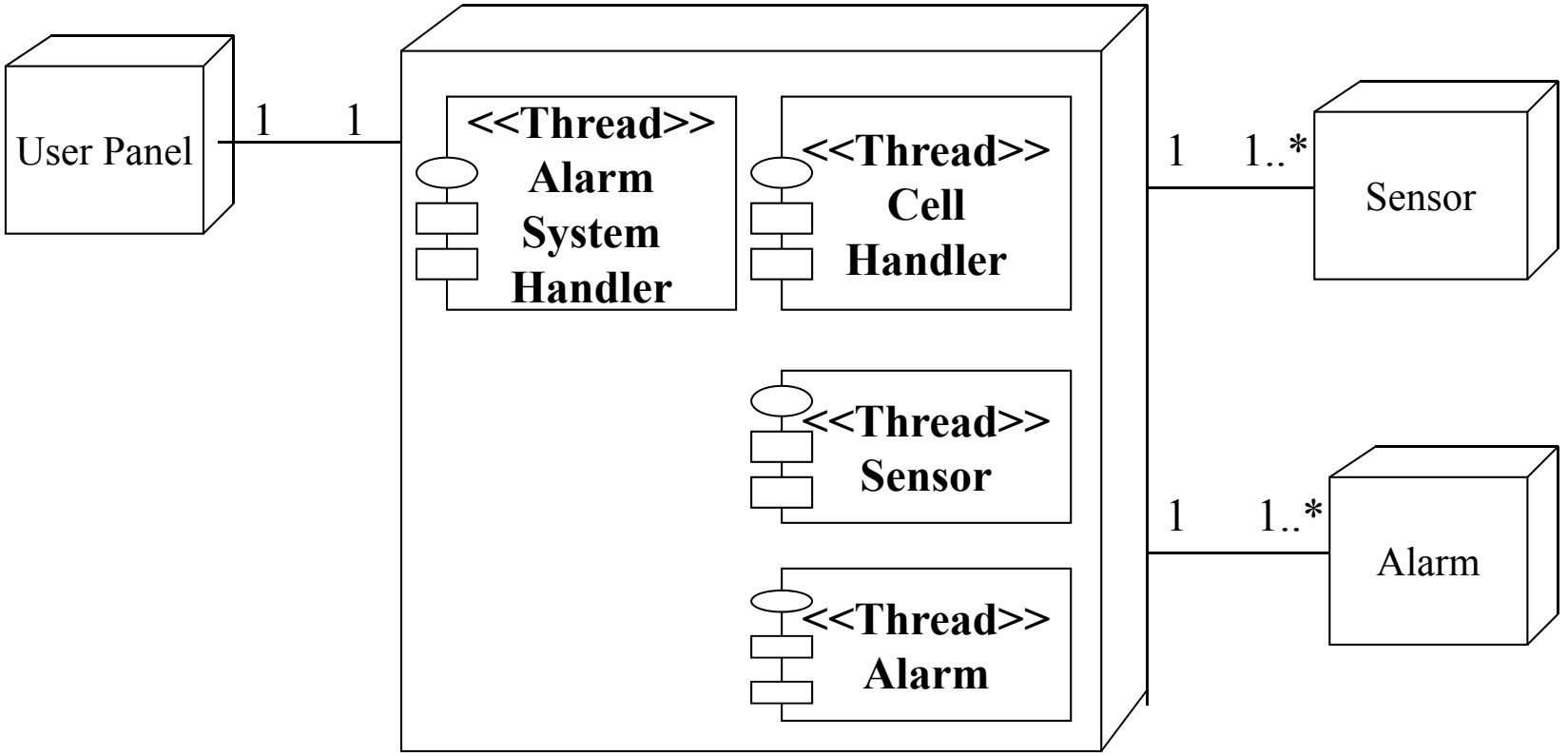
The run operation defined for the active Cell Handler class



H.E. Eriksson and M. Penker,

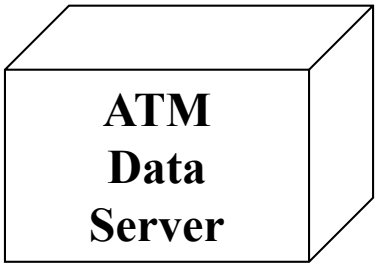
"UML Toolkit" John Wiley & Sons, Inc.

The deployment diagram for the house alarm system



H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

UML notation

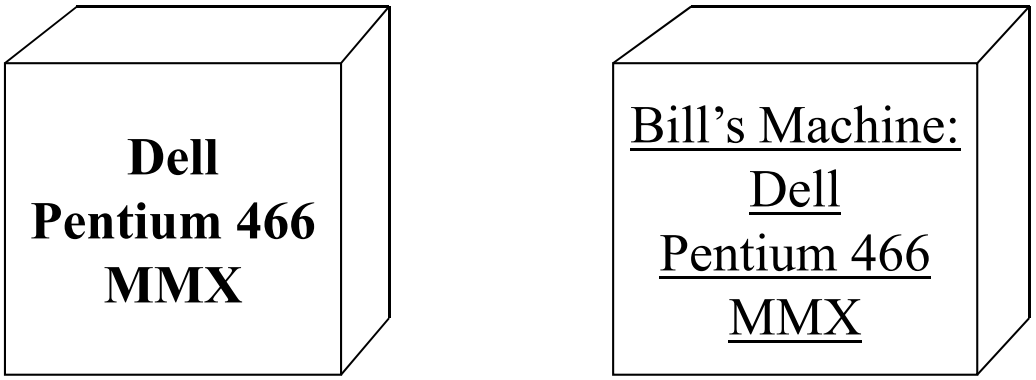


Node

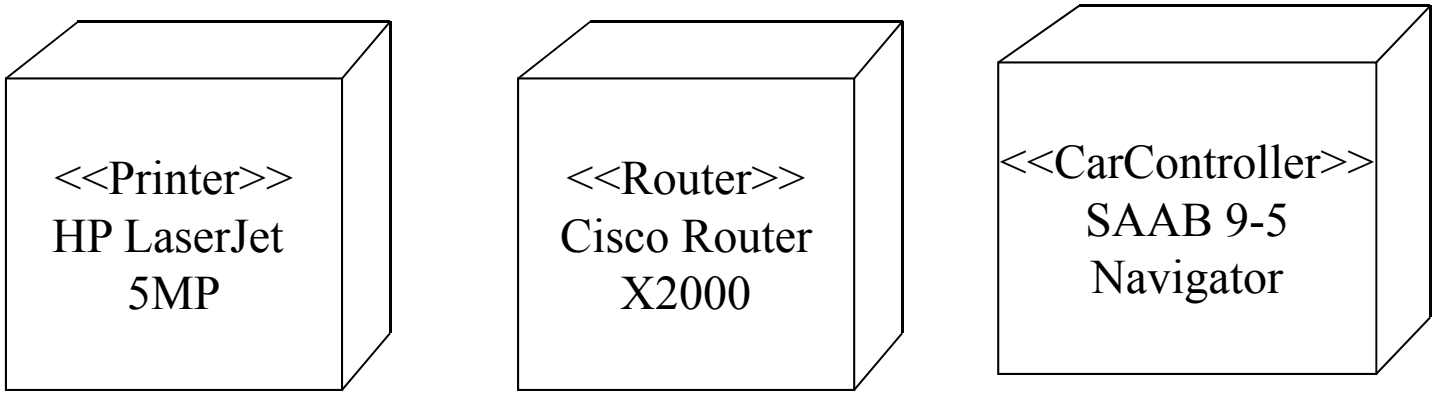
A physical element that exists at run time and that represents a computational resource, generally having at least some memory and, often times, processing capability

G.Booch, J.Rumbaugh, I. Jacobson , ”The Unified Modeling Language User Guide”, Addison Wesley, 1999.

Node



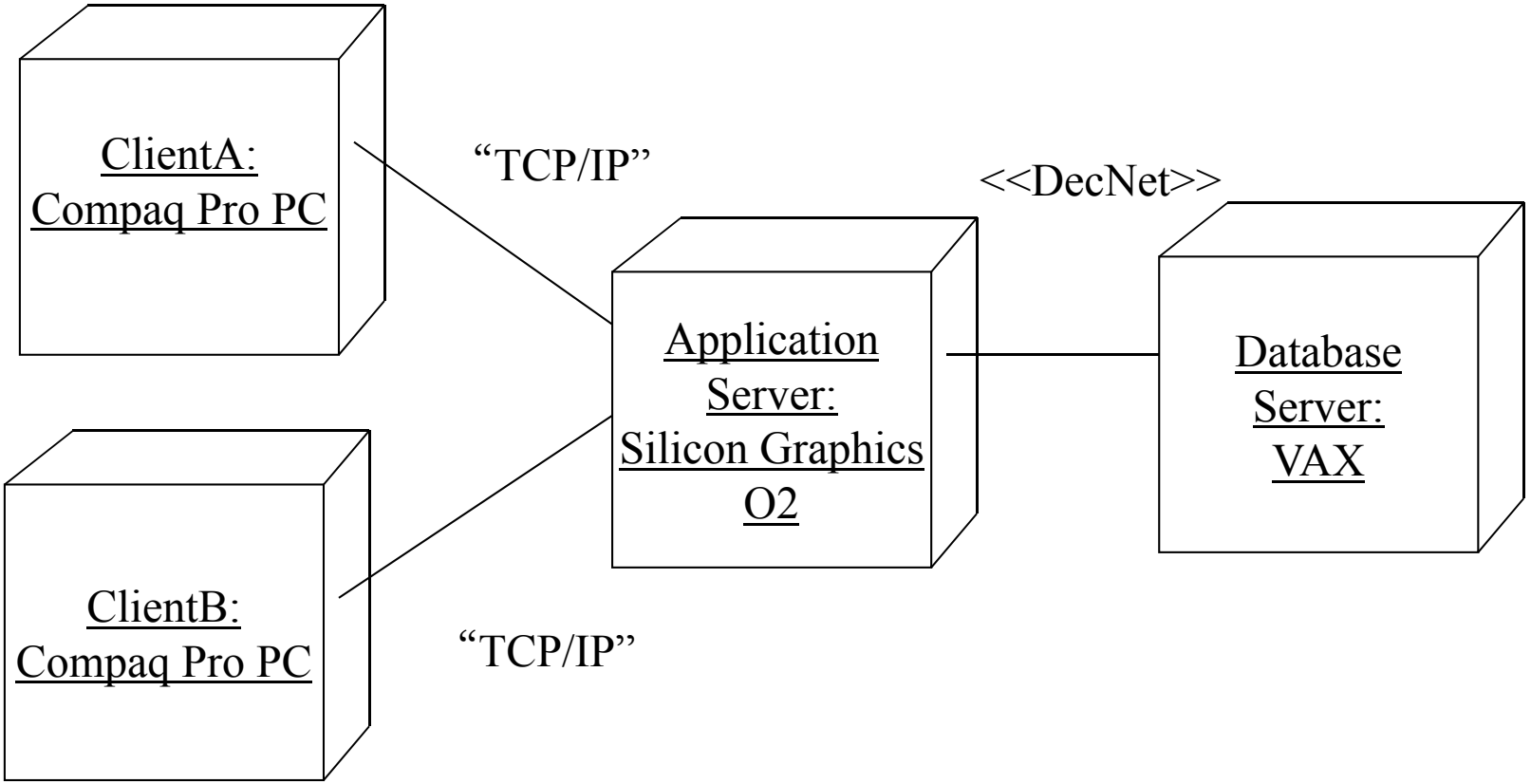
node type **an instance**



Device nodes and possible stereotype

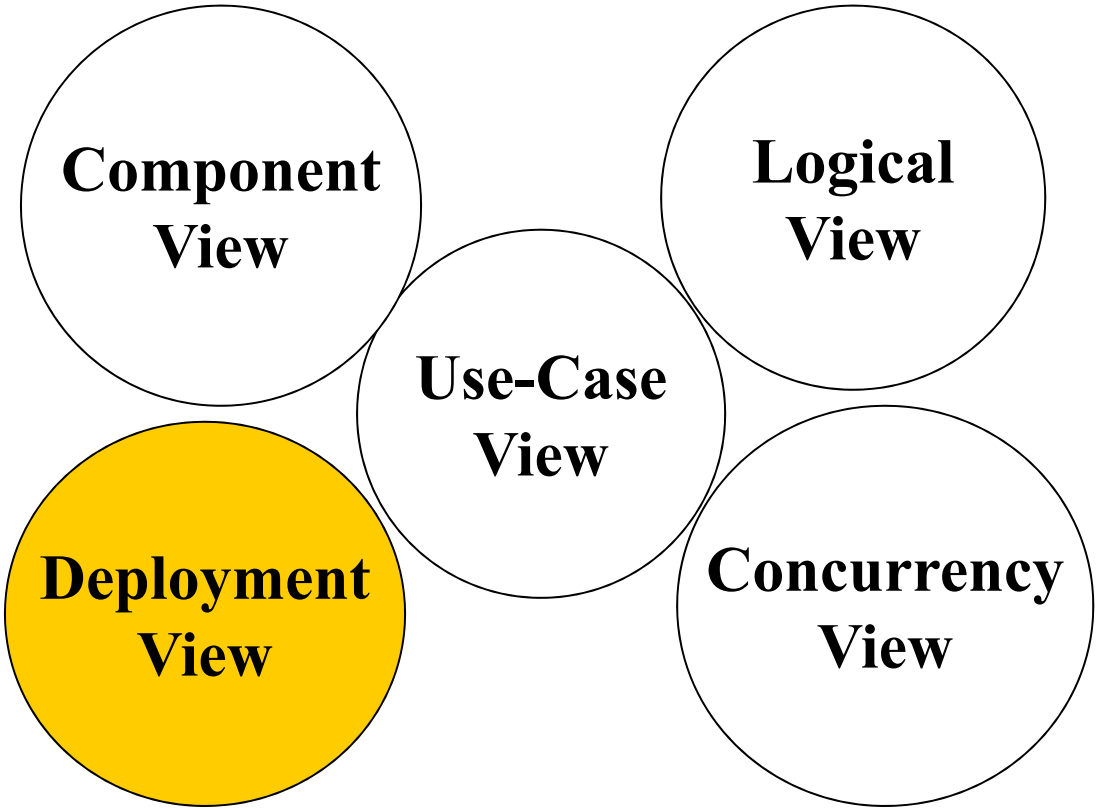
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Communication Association between nodes



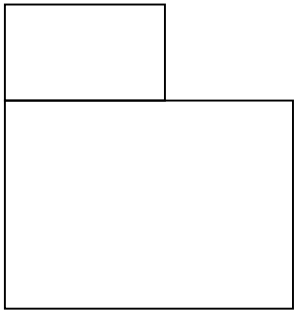
H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Deployment View

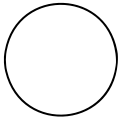


H.E. Eriksson and M. Penker, “UML Toolkit” John Wiley & Sons, Inc.

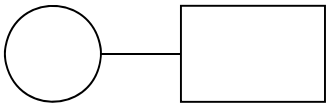
UML notations



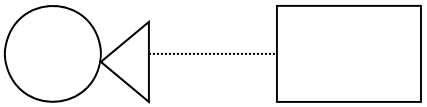
Package: Subsystem, Framework



Interface



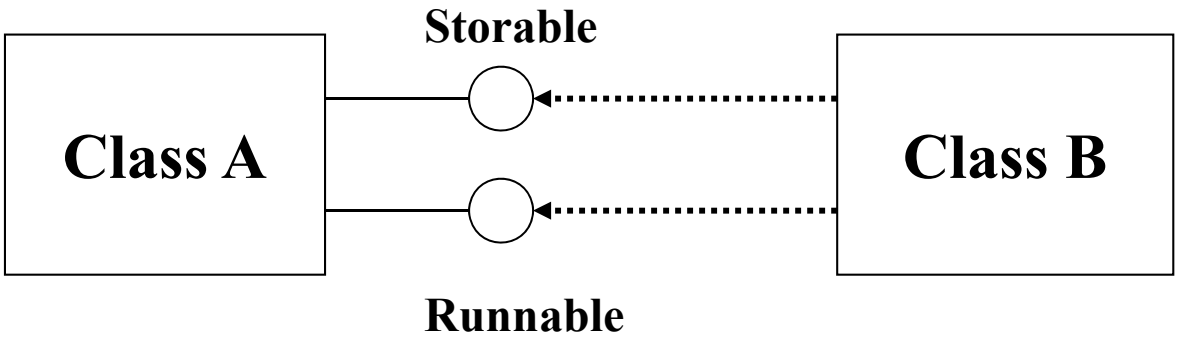
realization (simple form)

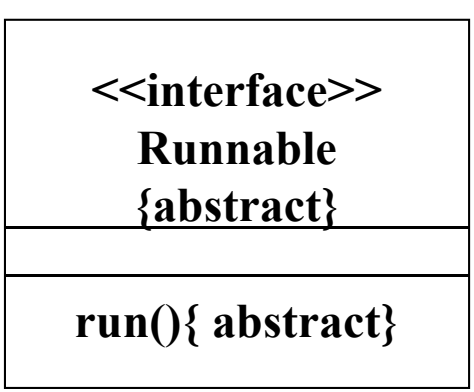


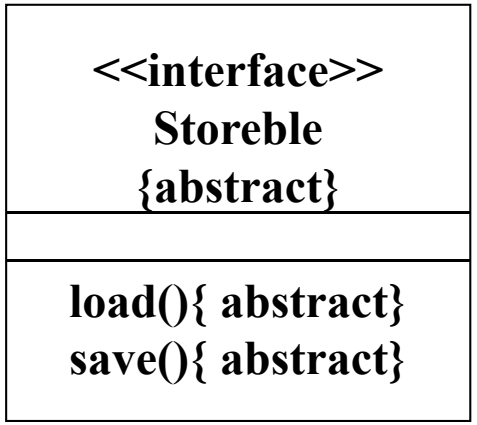
realization (expanded form)

G.Booch, J.Rumbaugh, I. Jacobson , "The Unified Modeling Language User Guide", Addison Wesley, 1999.

Interface







H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.

Java Implementation

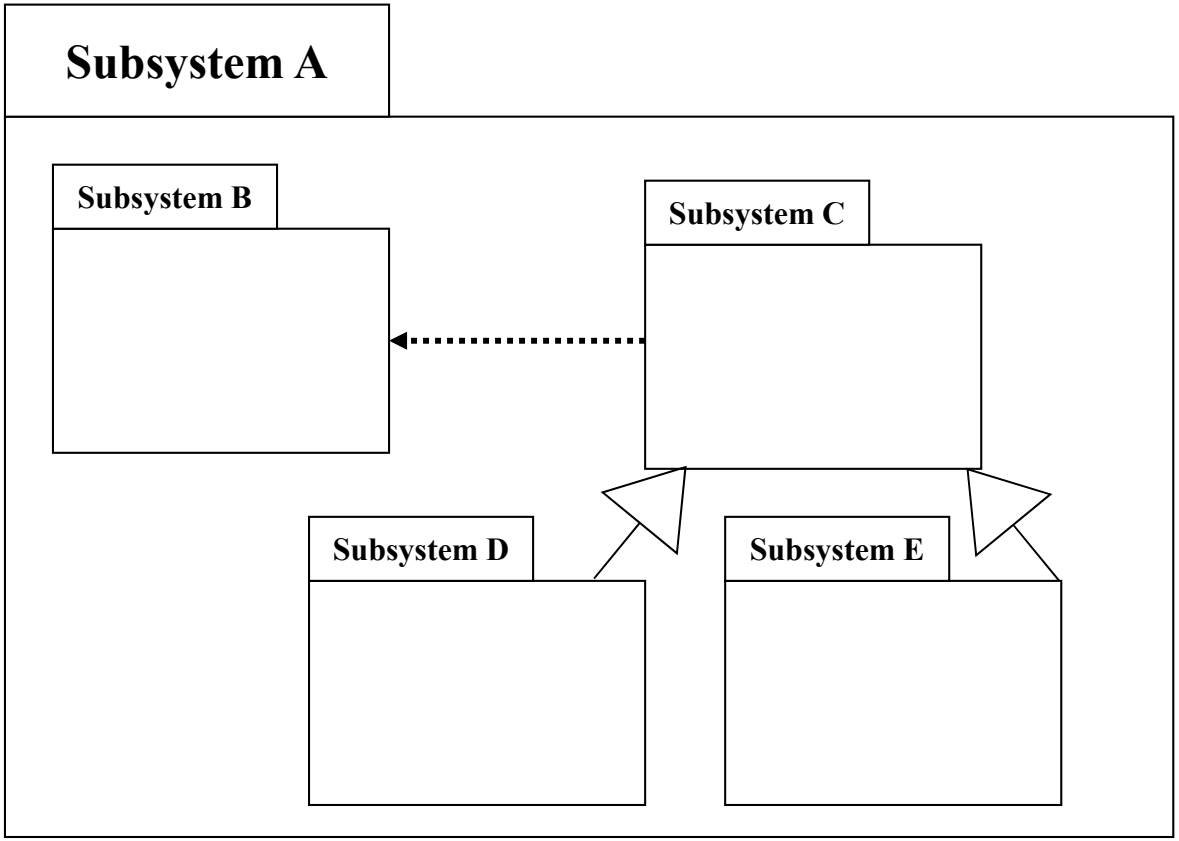
```
interface  Storable
{
   public void save();
   public void load();
};

 public class Person  implements  Storable
{
 public void save();
    {
      /* Implementation of save operation for Person */
    } public void save();
    {
      /* Implementation of  load operation for Person */
    }
};
```

H.E. Eriksson and M. Penker, “UML Toolkit” John Wiley & Sons, Inc.

Package

A pakage is a grouping mechnism, whereby all kinds of model elements can be linked



H.E. Eriksson and M. Penker, "UML Toolkit" John Wiley & Sons, Inc.