Distributed Applications

Some applications by their very nature are distributed across multiple computers because of one or more of the following reasons:

- The *data* used by the application are distributed
- The *computation* is distributed
- The *users* of the application are distributed

Data are Distributed

Some applications must execute on multiple computers because the data that the application must access exist on multiple computers for administrative and ownership reasons.

The owner may permit the data to be accessed remotely but not stored locally. Or perhaps the data cannot be co-located and must exist on multiple heterogeneous systems for historical reasons.

Computation is Distributed

- Some applications execute on multiple computers in order to take advantage of multiple processors computing in parallel to solve some problem.
- Other applications may execute on multiple computers in order to take advantage of some unique feature of a particular system.

Users are Distributed

Some applications execute on multiple computers because users of the application communicate and interact with each other via the application. E.g. a banking application where ATMs and PCs share access to the bank’s database.

Each user executes a piece of the distributed application on his or her computer, and shared remote objects, typically execute on one or more servers.

Prior to designing a distributed application, it is essential to understand some of the fundamental realities of the distributed system on which it will execute.
Fundamental Realities of Distributed Systems

Distributed application developers must address a number of issues that can be taken for granted in a local program where all logic executes in the same operating system process. The following table summarizes some of the basic differences between objects that are co-located in the same process, and objects that interact across process or machine boundaries.

<table>
<thead>
<tr>
<th></th>
<th>Co-located</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>Failures</td>
<td>Objects fail together</td>
<td>Objects fail separately</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network can partition</td>
</tr>
<tr>
<td>Concurrent access</td>
<td>Only with multiple threads</td>
<td>Yes</td>
</tr>
<tr>
<td>Secure</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Communication

- The communication between objects in the same process is orders of magnitude faster than communication between objects on different machines.

The implication of this is that you should avoid designing distributed applications in which two or more distributed objects have very tight interactions (i.e. have significant communication between them).

- If two objects have tight interactions, they should be co-located.

Failures

- When two objects are co-located, they fail together; if the process in which they execute fails, both objects fail.

The designer of the objects need not be concerned with the behavior of the application if one of the objects is available and the other one is not since that will not happen.

- But if two objects are distributed across process boundaries, the objects can fail independently. In this case, the designer of the objects must be concerned with each of the object's behavior in the event the other object has failed.
• So also, in a distributed system the network can partition and both objects can execute independently assuming the other has failed.

Concurrent Access

• The default mode for most local programs is to operate with a single thread of control.

Single threaded programming is easy. Objects are accessed in a well-defined sequential order according to the program's algorithms, and you need not be concerned with concurrent access.

If you decide to introduce multiple threads of control within a local program, you must consider the possible orderings of access to objects and use synchronization mechanisms to control concurrent access to shared objects.

• In a distributed application, there are necessarily multiple threads of control. Each distributed object is operating in a different thread of control. A distributed object may have multiple concurrent clients.

As the developer of the object and the developer of the clients, you must consider this concurrent access to objects and use the necessary synchronization mechanisms.

Security

• When two objects are co-located in the same process, you need not be concerned about security. When the objects are on different machines, you need to use security mechanisms to authenticate the identity of the other object.

Distributed Object Systems

• Distributed object systems are distributed systems in which all entities are modeled as objects.

Distributed object systems are a popular paradigm for object-oriented distributed applications. Since the application is modeled as a set of cooperating objects, it maps very naturally to the services of the distributed system.

In spite of the natural mapping from object-oriented modeling to distributed object systems, do not forget the realities of distributed systems described above. Process boundaries really do matter and they will impact your design.