- Non-functional requirements may demand different request characteristics.
- The non-synchronous requests are oneway requests, deferred synchronous requests and asynchronous requests. Oneway requests are never synchronized. Deferred synchronous requests are synchronized by the client using polling. Asynchronous requests are synchronized by the server.
- All these synchronization primitives can be achieved regardless of the middleware using synchronous requests and multiple threads.
- CORBA provides dedicated implementations for oneway, deferred synchronous and asynchronous communication that may execute more efficiently and may be easier to use than multiple threads.
- Event notifications can be done in an anonymous way using group requests, in which an event producer posts events into a channel that notifies those consumers that have registered an interest in the events.
- Sending multiple requests at once can improve the performance of requests because the communication overhead between the client and the middleware is reduced and clients reduce the waiting time for results as they process them in the order in which they become available.
- Multiple requests can be implemented using tuple spaces, synchronous requests and multiple threads in all forms of object-oriented middleware.
- CORBA provides multiple requests as part of the dynamic invocation interface and they may be easier to use and execute more efficiently than multi-threaded implementations of multiple requests.
- Middleware supports different degrees of request reliability. Highly reliable requests, such as exactly-once, totally ordered or atomic requests, tend to be slow while less reliable requests that are executed at-most-once or maybe can be handled much faster.

Self Assessment

7.1 Which non-functional requirements can be addressed using CORBA's dynamic invocation interface?

7.2 Is there any dependency between dynamic and multiple requests?

7.3 What are the differences between multiple requests and group requests?

7.4 How could group requests be implemented using message queues?

7.5 Asynchronous requests seem much more efficient than synchronous requests. Why are they not used all the time?

7.6 How do the CORBA DII operations `send` and `invoke` differ?

7.7 When would you use threads and when would you use message queues to implement asynchronous requests?

7.8 Can asynchronous requests implemented with message queues perform faster than those implemented with threads?

7.9 Given that exactly-once provides the highest degree of reliability, why should designers make do with weaker reliability?

Further Reading

The multiple requests that are available in CORBA are defined as part of the Dynamic Invocation Interface that is specified in Chapter [Object Management Group, 1998c]. The CORBA group communication primitives that we sketched in this chapter are specified in Chapter 4 of [Object Management Group, 1998a].

In this chapter, we have discussed how multi-threading can be used to implement asynchronous communication. While doing so, we have assumed that the reader is familiar with the concepts and implementation of concurrency. A good introduction to the theory and practice of concurrent programming is provided by [Magee and Kramer, 1999]. In particular, they discuss models and Java implementations for message passing, which can be used for asynchronous communication between distributed objects. [Lea, 1997] gives another good introduction to concurrent programming in Java.

The group communication primitives provided by CORBA's Event service are just one example. More recently, the OMG has adopted a Notification service [Object Management Group, 1998b] that enables clients to adjust quality of service attributes for the group communication. There is currently a lot of research interest into push technology for the Internet. A good conceptual framework for Internet-scale event notification is provided by [Rosenblum and Wolf, 1997].

There is a large body of work on coordination primitives and languages. This research community has a conference series in which many approaches to communication in distributed systems based on tuple spaces have been presented. The proceedings have been edited by [Ciancarini and Hankin, 1996], [Garlan and LeMetayer, 1997] and [Ciancarini and Wolf, 1999]. With a more industrial orientation, there are various prototypes of communication systems based on tuple spaces. JavaSpaces [Freeman et al., 1999] has been developed by Sun Microsystems and TSpaces [Wyckoff et al., 1998] is a prototype developed at IBM Almaden Research Center.

There are several message-oriented middleware products in the market, which support asynchronous and group communication. One of the market leading products is IBM's MQSeries. A good overview of MQSeries is provided by [Gilman and Schreiber, 1996].