The design decisions addressed in 3.1.5 are reflective of those encountered when applying the architectural style familiarly described as “highly distributed.” The guidelines provided could be turned into specific design decisions applicable to the application scenario of building a medical appointments scheduler after assumptions concerning the underlying intent of the design decisions as regards the architecture is presumed. Regardless of the mechanism used in the application of the ‘highly-distributed’ style, specific choices affecting properties associated with individual components and the system as a whole are to be faced.

At a high level, the physical means for separating software components which provide services from those which request them of the application can be implemented in terms of distributing these components amongst distinctive locations. The logical components that provide the algorithm to effect the most efficient scheduling approach may be stored on a machine or environment, separately, in terms of hardware, in this case, in other words. Components that initiate scheduling requests are not a part of the application proper as one might encounter in a web application running on a server that gets all its data from the database connector and database instance on the same machine as the app server, itself, in the physically and logically separated design prototype, for example.

1. In terms of identity agnosticism, TCP’s means are a suitable medium through which to carry out such a decision (in contrast to the web application that accepts and responds to all logic, itself and ties requests to a process specifically associated with each requester) as the REST principle can be prescribed or adhered to using TCP.
2. By providing a key or index into the schedulers’ queue or results, in this situation the scheduler can maintain an open line for inbound communication from requesters, and perhaps upon receiving the request the receiver can immediately return an isAccepted() handle that allows isScheduledYet() calls to be made by the requester thereafter and a retrieveScheduleDetails() which first validates using an isScheduledYet() call inside it.
3. To keep requesters as entities separate from each other, the requesters are insulated by using the ‘handler’ technique described in (ii) and the data that is generated in association with each request is reflective of this idea in terms of its relational data storage process and structure. The requester thus becomes dependent on the response mechanism that the server delivered to it, which may remain physically separated. The handler technique is implemented using an implementation of connectors, similar to as is described in detail on pg.71, paragraph 2, in association with the ‘procedure call’ approach.
4. Multiple service providers can emerge to service the growing or extant requests to the server(s) by peeking into a queuing server that managers the active handles into the scheduler and can return scheduling details to each (active) requester based on the mapping between the handle-ID’s and scheduling objects with which they are associated