TABLE 13.10 Map-Reduce Pattern Solution

Overview	The map-reduce pattern provides a framework for analyzing a large distributed set of data that will execute in parallel, on a set of processors. This parallelization allows for low latency and high availability. The map performs the *extract* and *transform* portions of the analysis and the reduce performs the *loading* of the results. (*Extract-transform-load* is sometimes used to describe the functions of the map and reduce.)
Elements	*Map* is a function with multiple instances deployed across multiple processors that performs the extract and transformation portions of the analysis. *Reduce* is a function that may be deployed as a single instance or as multiple instances across processors to perform the load portion of extract-transform-load. The *infrastructure* is the framework responsible for deploying map and reduce instances, shepherding the data between them, and detecting and recovering from failure.
Relations	*Deploy on* is the relation between an instance of a map or reduce function and the processor onto which it is installed. *Instantiate, monitor, and control* is the relation between the infrastructure and the instances of map and reduce.
Constraints	The data to be analyzed must exist as a set of files. The map functions are stateless and do not communicate with each other. The only communication between the map instances and the reduce instances is the data emitted from the map instances as <key, value> pairs.
Weaknesses	If you do not have large data sets, the overhead of map-reduce is not justified. If you cannot divide your data set into similar sized subsets, the advantages of parallelism are lost. Operations that require multiple reduces are complex to orchestrate.

Multi-tier Pattern

The multi-tier pattern is a C&C pattern or an allocation pattern, depending on the criteria used to define the tiers. Tiers can be created to group components of similar functionality, in which case it is a C&C pattern. However, in many, if not most, cases tiers are defined with an eye toward the computing environment on which the software will run: A client tier in an enterprise system will not be running on the computer that hosts the database. That makes it an allocation pattern, mapping software elements—perhaps produced by applying C&C patterns—to computing elements. Because of that reason, we have chosen to list it as an allocation pattern.

Context: In a distributed deployment, there is often a need to distribute a system's infrastructure into distinct subsets. This may be for operational or business reasons (for example, different parts of the infrastructure may belong to different organizations).

Problem: How can we split the system into a number of computationally independent execution structures—groups of software and hardware—connected by some communications media? This is done to provide specific server environments optimized for operational requirements and resource usage.

Solution: The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a *tier*. The grouping of components into tiers may be based on a variety of criteria, such as the type of component, sharing the same execution environment, or having the same runtime purpose.

The use of tiers may be applied to any collection (or pattern) of runtime components, although in practice it is most often used in the context of client-server patterns. Tiers induce topological constraints that restrict which components may communicate with other components. Specifically, connectors may exist only between components in the same tier or residing in adjacent tiers. The multi-tier pattern found in many Java EE and Microsoft .NET applications is an example of organization in tiers derived from the client-server pattern.

Additionally, tiers may constrain the *kinds* of communication that can take place across adjacent tiers. For example, some tiered patterns require call-return communication in one direction but event-based notification in the other.

The main weakness with the multi-tier architecture is its cost and complexity. For simple systems, the benefits of the multi-tier architecture may not justify its up-front and ongoing costs, in terms of hardware, software, and design and implementation complexity.

Tiers are not components, but rather logical groupings of components. Also, don't confuse tiers with layers! Layering is a pattern of modules (a unit of implementation), while tiers applies only to runtime entities.

Table 13.11 summarizes the solution part of the multi-tier pattern.

Tiers make it easier to ensure security, and to optimize performance and availability in specialized ways. They also enhance the modifiability of the system, as the computationally independent subgroups need to agree on protocols for interaction, thus reducing their coupling.

Figure 13.15 uses an informal notation to describe the multi-tier architecture of the Consumer Website Java EE application. This application is part of the Adventure Builder system. Many component-and-connector types are specific to the supporting platform, which is Java EE in this case.

TABLE 13.11 Multi-tier Pattern Solution

Overview	The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a *tier*. The grouping of components into tiers may be based on a variety of criteria, such as the type of component, sharing the same execution environment, or having the same runtime purpose.
Elements	*Tier,* which is a logical grouping of software components. Tiers may be formed on the basis of common computing platforms, in which case those platforms are also elements of the pattern.
Relations	*Is part of,* to group components into tiers. *Communicates with,* to show how tiers and the components they contain interact with each other. *Allocated to,* in the case that tiers map to computing platforms.
Constraints	A software component belongs to exactly one tier.
Weaknesses	Substantial up-front cost and complexity.

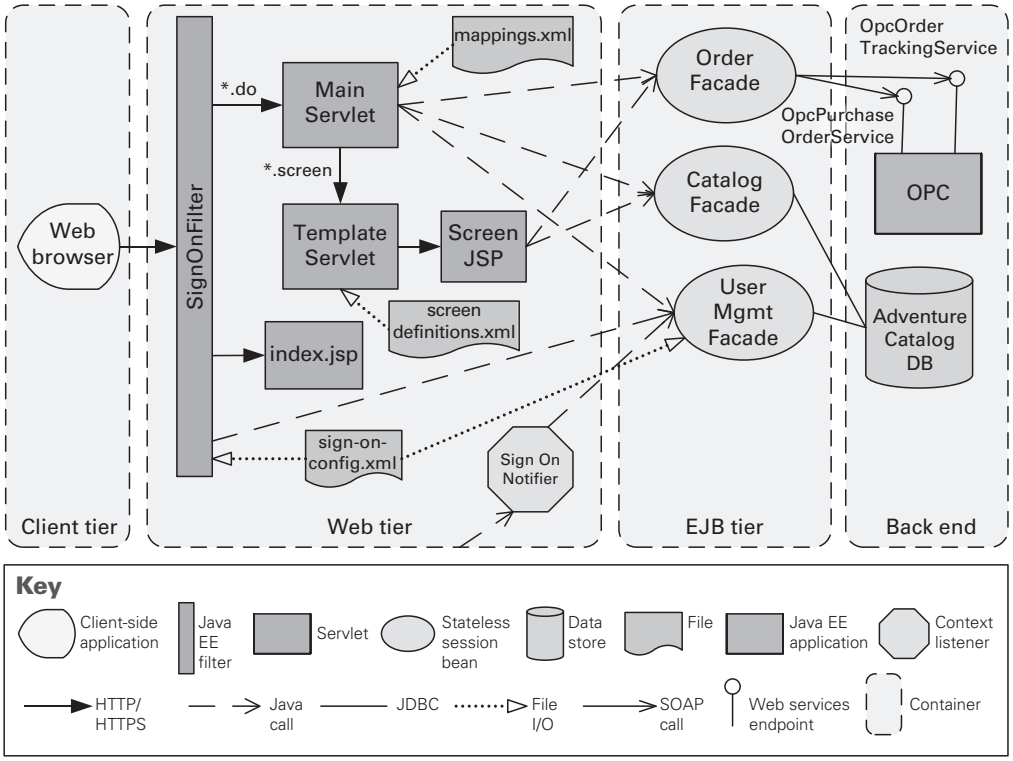

FIGURE 13.15 A multi-tier view of the Consumer Website Java EE application, which is part of the Adventure Builder system