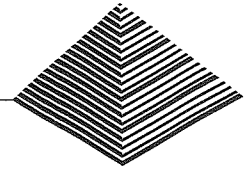
4



Case Study: FCAPS System

We now present a case study of using ADD 3.0 for a greenfield system in a mature domain. This case study details an initial design round composed of three iterations and is based on a real-world example. We first present the business context, and then we summarize the requirements for the system. This is followed by a step-by-step summary of the activities that are performed during the ADD iterations.

4.1 Business Case

In 2006, a large telecommunications company wanted to expand its Internet Protocol (IP) network to support "carrier-class services", and more specifically high-quality voice over IP (VOIP) systems. One important aspect to achieve this goal was synchronization of the VOIP servers and other equipment. Poor synchronization results in low quality of service (QoS), degraded performance, and unhappy customers. To achieve the required level of synchronization, the company wanted to deploy a network of time servers that support the Network Time Protocol (NTP). Time servers are formed into groups that typically correspond to geographical regions. Within these regions, time servers are organized hierarchically in levels or *strata,* where time servers placed in the upper level of the

hierarchy (stratum 1) are equipped with hardware (e.g., Cesium Oscillator, GPS signal) that provides precise time. Time servers that are lower in the hierarchy use NTP to request time from servers in the upper levels or from their peers.

Many pieces of equipment depend on the time provided by time servers in the network, so one priority for the company was to correct any problems that occur on the time servers. Such problems may require dispatching a technician to perform physical maintenance on the time servers, such as rebooting. Another priority for the company was to collect data from the time servers to monitor the performance of the synchronization framework.

In the initial deployment plans, the company wanted to field 100 time servers of a particular model. Besides NTP, time servers support the Simple Network Management Protocol (SNMP), which provides three basic operations:

- `set()` operations: change configuration variables (e.g., connected peers)
- `get()` operations: retrieve configuration variables or performance data
- `trap()` operations: notifications of exceptional events such as the loss or restoration of the GPS signal or changes in the time reference

To achieve the company's goals, a management system for the time servers needed to be developed. This system needed to conform to the FCAPS model, which is a standard model for network management. The letters in the acronym stand for:

- *Fault management.* The goal of fault management is to recognize, isolate, correct, and log faults that occur in the network. In this case, these faults correspond to traps generated by time servers or other problems such as loss of communication between the management system and the time servers.
- *Configuration management.* This includes gathering and storing configurations from network devices, thereby simplifying the configuration of devices and tracking changes that are made to device configurations. In this system, besides changing individual configuration variables, it is necessary to be able to deploy a specific configuration to several time servers.
- *Accounting.* The goal here is to gather device information. In this context, this includes tracking device hardware and firmware versions, hardware equipment, and other components of the system.
- *Performance management.* This category focuses on determining the efficiency of the current network. By collecting and analyzing performance data, the network health can be monitored. In this case, delay, offset, and jitter measures are collected from the time servers.
- *Security management.* This is the process of controlling access to assets in the network. In this case, there are two important types of users: technicians and administrators. Technicians can visualize trap information and configurations but cannot make changes; administrators are technicians who can visualize the same information but can also make changes to configurations, including adding and removing time servers from the network.

Once the initial network was deployed, the company planned to extend it by adding time servers from newer models that might potentially support management protocols other than SNMP.

The remainder of this chapter describes a design for this system, created using ADD 3.0.

4.2 System Requirements

Requirement elicitation activities had been previously performed, and the following is a summary of the most relevant requirements collected.

4.2.1 Use Case Model

The use case model in Figure 4.1 presents the most relevant use cases that support the FCAPS model in the system. Other use cases are not shown.

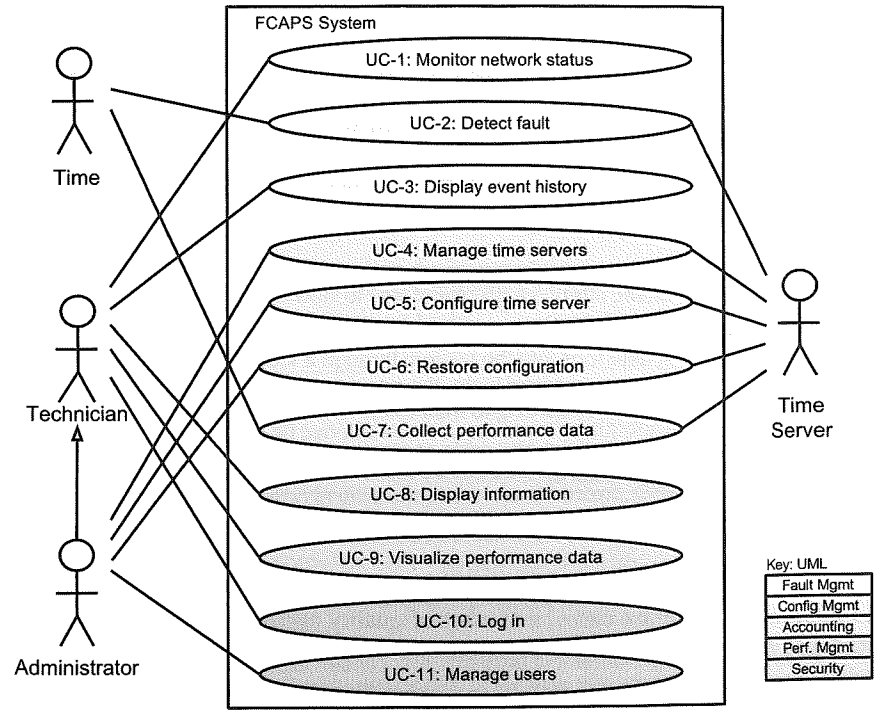


FIGURE 4.1 Use case model for the FCAPS system

Each of these use cases is described in the following table:

Use Case	Description
UC-1: Monitor network status	A user monitors the time servers in a hierarchical representation of the whole network. Problematic devices are highlighted, along with the logical regions where they are grouped. The user can expand and collapse the network representation. This representation is updated continuously as faults are detected or repaired.
UC-2: Detect fault	Periodically the management system contacts the time servers to see if they are "alive". If a time server does not respond, or if a trap that signals a problem or a return to a normal state of operation is received, the event is stored and the network representation observed by the users is updated accordingly.
UC-3: Display event history	Stored events associated with a particular time server or group of time servers are displayed. These can be filtered by various criteria such as type or severity.
UC-4: Manage time servers	The administrator adds a time server to, or removes a time server from, the network.
UC-5: Configure time server	An administrator changes configuration parameters associated with a particular time server. The parameters are sent to the device and are also stored locally.
UC-6: Restore configuration	A locally stored configuration is sent to one or more time servers.
UC-7: Collect performance data	Network performance data (delay, offset, and jitter) is collected periodically from the time servers.
UC-8: Display information	The user displays stored information about the time server—configuration values and other parameters such as the server name.
UC-9: Visualize performance data	The user displays network performance measures (delay, offset, jitter) in a graphical way to view and analyze network performance.
UC-10: Log in	A user logs into the system through a login/password screen. Upon successful login, the user is presented with different options according to their role.
U-11: Manage users	The administrator adds or removes a user or modifies user permissions.

4.2.2 Quality Attribute Scenarios

In addition to these use cases, a number of quality attribute scenarios were elicited and documented. The six most relevant ones are presented in the following table. For each scenario, we also identify the use case that it is associated with.

ID	Quality Attribute	Scenario	Associated Use Case
QA-1	Performance	Several time servers send traps to the management system at peak load; 100% of the traps are successfully processed and stored.	UC-2
QA-2	Modifiability	A new time server management protocol is introduced to the system as part of an update. The protocol is added successfully without any changes to the core components of the system.	UC-5
QA-3	Availability	A failure occurs in the management system during normal operation. The management system resumes operation in less than 30 seconds.	All
QA-4	Performance	The management system collects performance data from a time server during peak load. The management system collects all performance data within 5 minutes, while processing all user requests, to ensure no loss of data due to CON-5.	UC-7
QA-5	Performance, usability	A user displays the event history of a particular time server during normal operation. The list of events from the last 24 hours is displayed within 1 second.	UC-3
QA-6	Security	A user performs a change in the system during normal operation. It is possible to know who performed the operation and when it was performed 100% of the time.	All

4.2.3 Constraints

Finally, a set of constraints on the system and its implementation were collected. These are presented in the following table.

ID	Constraint
CON-1	A minimum of 50 simultaneous users must be supported.
CON-2	The system must be accessed through a web browser (Chrome V3.0+, Firefox V4+, IE8+) in different platforms: Windows, OSX, and Linux.
CON-3	An existing relational database server must be used. This server cannot be used for other purposes than hosting the database.
CON-4	The network connection to user workstations can have low bandwidth but is generally reliable.
CON-5	Performance data needs to be collected in intervals of no more than 5 minutes, as higher intervals result in time servers discarding data.
CON-6	Events from the last 30 days must be stored.

4.2.4 Architectural Concerns

Given that this is greenfield development, only a few general architectural concerns are identified initially, as shown in the following table.

ID	Concern
CRN-1	Establishing an overall initial system structure.
CRN-2	Leverage the team's knowledge about Java technologies, including Spring, JSF, Swing, Hibernate, Java Web Start and JMS frameworks, and the Java language.
CRN-3	Allocate work to members of the development team.

Given these sets of inputs, we are now ready to proceed to describe the design process, as described in Section 3.2. In this chapter, we present only the final results of the requirements collection process. The job of collecting these requirements is nontrivial, but is beyond the scope of this chapter.

4.3 The Design Process

We now ready to make the leap from the world of requirements and business concerns to the world of design. This is perhaps the most important job for an architect—translating requirements into design decisions. Of course, many other decisions and duties are important, but this is the core of what it means to be an architect: making design decisions with far-reaching consequences.

4.3.1 ADD Step 1: Review Inputs

The first step of the ADD method involves reviewing the inputs and identifying which requirements will be considered as drivers (i.e., which will be included in the design backlog). The inputs are summarized in the following table.

Category	Details
Design purpose	This is a greenfield system from a mature domain. The purpose is to produce a sufficiently detailed design to support the construction of the system.
Primary functional requirements	From the use cases presented in Section 4.2.1, the primary ones were determined to be: UC-1: Because it directly supports the core business UC-2: Because it directly supports the core business UC-7: Because of the technical issues associated with it (see QA-4)

Quality attribute scenarios	The scenarios were described in Section 4.2.2. They have now been prioritized (as discussed in Section 2.4.2) as follows:

Scenario ID	Importance to the Customer	Difficulty of Implementation According to the Architect
QA-1	High	High
QA-2	High	Medium
QA-3	High	High
QA-4	High	High
QA-5	Medium	Medium
QA-6	Medium	Low

	From this list, only QA-1, QA-2, QA-3, and QA-4 are selected as drivers.
Constraints	All of the constraints discussed in Section 4.2.3 are included as drivers.
Architectural concerns	All of the architectural concerns discussed in Section 4.2.4 are included as drivers.

4.3.2 Iteration 1: Establishing an Overall System Structure

This section presents the results of the activities that are performed in each of the steps of ADD in the first iteration of the design process.

4.3.2.1 Step 2: Establish Iteration Goal by Selecting Drivers

This is the first iteration in the design of a greenfield system, so the iteration goal is to achieve the architectural concern CNR-1 of *establishing an overall system structure* (see Section 3.3.1).

Although this iteration is driven by a general architectural concern, the architect must keep in mind *all* of the drivers that may influence the general structure of the system. In particular, the architect must be mindful of the following:

- QA-1: Performance
- QA-2: Modifiability
- QA-3: Availability
- QA-4: Performance
- CON-2: System must be accessed through a web browser in different platforms—Windows, OSX, and Linux
- CON-3: A relational database server must be used
- CON-4: Network connection to users workstations can have low bandwidth and be unreliable
- CRN-2: Leverage team's knowledge about Java technologies

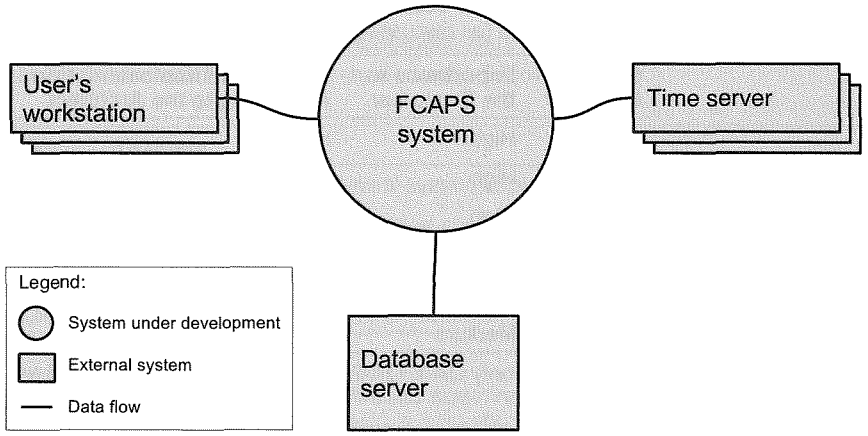


FIGURE 4.2 Context diagram for the FCAPS system

4.3.2.2 Step 3: Choose One or More Elements of the System to Refine

This is a greenfield development effort, so in this case the element to refine is the entire FCAPS system, which is shown in Figure 4.2. In this case, refinement is performed through decomposition.

4.3.2.3 Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

In this initial iteration, given the goal of structuring the entire system, design concepts are selected according to the roadmap presented in Section 3.3.1. The following table summarizes the selection of design decisions. Note that all of the design concepts used in this case study are also described in Appendix A.

Design Decisions and Location	Rationale
Logically structure the client part of the system using the **Rich Client Application** reference architecture	The Rich Client Application (RCA) reference architecture (see Section A.1.2) supports the development of applications that are installed in the users' PC. These applications support rich user interface capabilities that are needed for displaying the network topology and performance graphs (UC-1). These capabilities are also helpful in achieving QA-5, even if this design decision is not a driver. Although these types of applications do not run in a web browser (CON-2), they can be installed from a web browser using a technology such as Java Web Start.

Design Decisions and Location	Rationale
	Discarded alternatives:

Alternative	Reason for Discarding
Rich Internet applications (RIA)	This reference architecture (see Section A.1.3) is oriented toward the development of applications with a rich user interface that runs inside a web browser. Although this type of application supports a rich user interface and can be upgraded easily, this option was discarded because it was believed that plugins for executing RIA were less broadly available than the Java Virtual Machine.
Web applications	This reference architecture (see Section A.1.1) is oriented toward the development of applications that are accessed from a web browser. Although this reference architecture facilitates deployment and updating, it was discarded because it is difficult to provide a rich user interface experience.
Mobile applications	This reference architecture (see Section A.1.4) is oriented toward the development of applications that are deployed in handheld devices. This alternative was discarded because this type of device was not considered for accessing the system.

Logically structure the server part of the system using the **Service Application** reference architecture	Service applications (see Section A.1.5) do not provide a user interface but rather expose services that are consumed by other applications. No other alternatives were considered and discarded, as the architect was familiar with this reference architecture and considered it fully adequate to meet the requirements.
Physically structure the application using the **three-tier deployment pattern**	Since the system must be accessed from a web browser (CON-2) and an existing database server must also be used (CON-3), a three-tier deployment is appropriate (see Section A.2.2). At this point, it is clear that some type of replication will be needed on both the web/app tier and the database tier to support QA-3, but this will be addressed later (in iteration 3). Discarded alternatives include other *n*-tier patterns with $n != 3$. The two-tier alternative is discarded because an existing legacy database server needs to be incorporated into the system and this cannot be used for any other purpose, according to CON-3. All $n > 3$ alternatives are discarded because at this point no other servers are necessary for the solution.

(*continues*)

Design Decisions and Location	Rationale
Build the user interface of the client application using the Swing Java framework and other Java technologies	The standard framework for building Java Rich Clients ensures portability (CON-2) and it is what the developers were already familiar with (CRN-3). Discarded alternatives: The Eclipse SWT (Standard Widget Toolkit) framework was considered, but the developers were not as familiar with it.
Deploy the application using the Java Web Start technology	Access to the application is obtained via a web browser, which launches the installer (CON-2). This technology also facilitates updating because client code is reloaded only when a new version is available. As updates are not expected to occur frequently, this is beneficial for low-bandwidth situations (CON-4). The alternative would be the use of applets, but they need to be reloaded every time the web page is loaded, which increases the bandwidth requirements.

4.3.2.4 Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

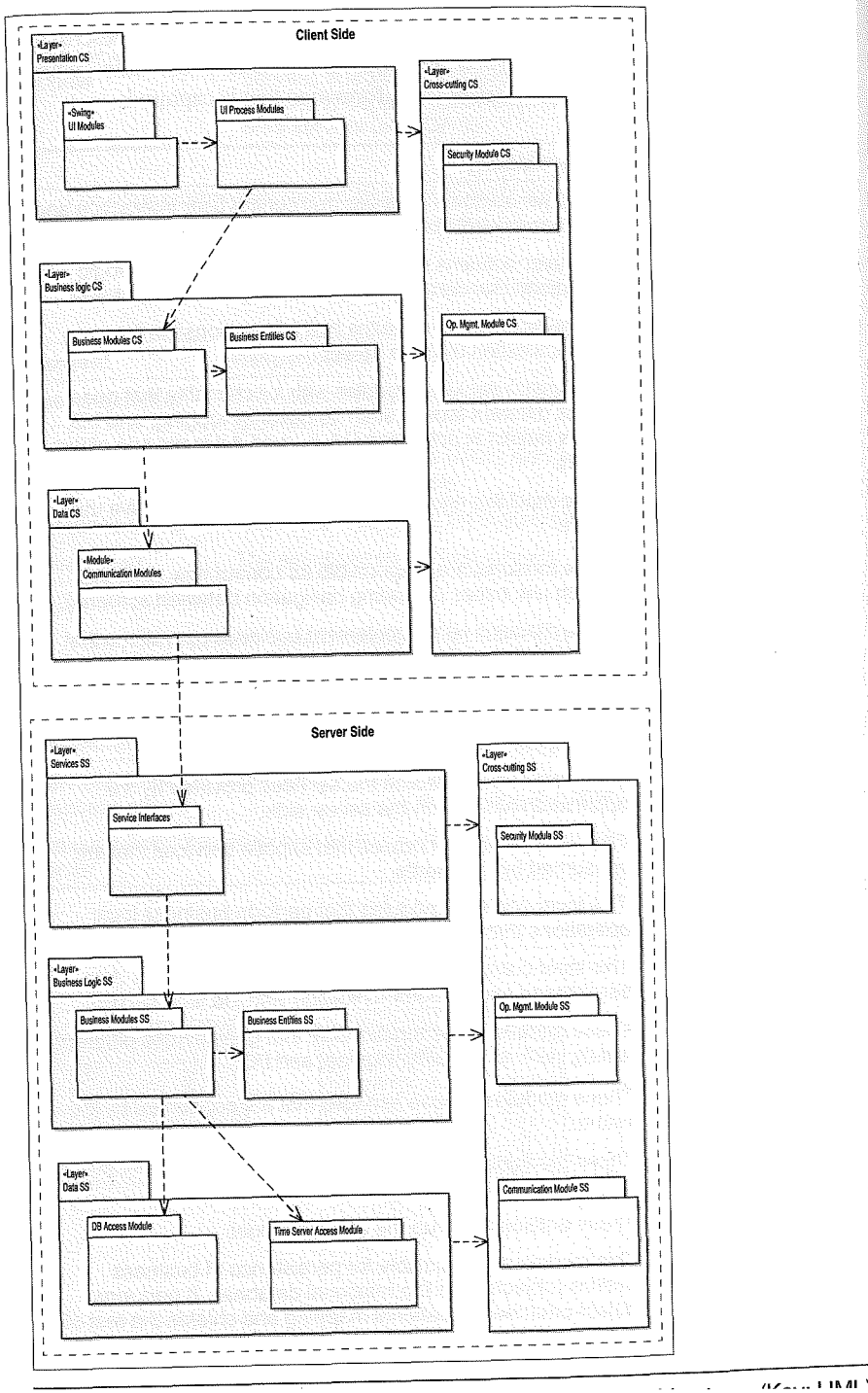
The instantiation design decisions considered and made are summarized in the following table:

Design Decision and Location	Rationale
Remove local data sources in the rich client application	It is believed that there is no need to store data locally, as the network connection is generally reliable. Also, communication with the server is handled in the data layer. Internal communication between components in the client is managed through local method calls and does not need particular support.
Create a module dedicated to accessing the time servers in the data layer of the Service Application reference architecture	The service agents component from the reference architecture is adapted to abstract the access to the time servers. This will further facilitate the achievement of QA-2 and will play a critical role in the achievement of UC-2 and UC-7.

The results of these instantiation decisions are recorded in the next step. In this initial iteration, it is typically too early to precisely define functionality and interfaces. In the next iteration, which is dedicated to defining functionality in more detail, interfaces will begin to be defined.

4.3.2.5 Step 6: Sketch Views and Record Design Decisions

The diagram in Figure 4.3 shows the sketch of a module view of the two reference architectures that were selected for the client and server applications. These have now been adapted according to the design decisions we have made.



This sketch was created using a CASE tool. In the tool, each element is selected and a short description of its responsibilities is captured. Note that the descriptions at this point are quite crude, just indicating major functional responsibilities, with no details. The following table summarizes the information that is captured:

Element	Responsibility
Presentation client side (CS)	This layer contains modules that control user interaction and use case control flow.
Business logic CS	This layer contains modules that perform business logic operations that can be executed locally on the client side.
Data CS	This layer contains modules that are responsible for communication with the server.
Cross-cutting CS	This "layer" includes modules with functionality that goes across different layers, such as security, logging, and I/O. This is helpful in achieving QA-6, even if it is not one of the drivers.
UI modules	These modules render the user interface and receive user inputs.
UI process modules	These modules are responsible for control flow of all the system use cases (including navigation between screens).
Business modules CS	These modules either implement business operations that can be performed locally or expose business functionality from the server side.
Business entities CS	These entities make up the domain model. They may be less detailed than those on the server side.
Communication modules CS	These modules consume the services provided by the application running on the server side.
Services server side (SS)	This layer contains modules that expose services that are consumed by the clients.
Business Logic SS	This layer contains modules that perform business logic operations that require processing on the server side.
Data SS	This layer contains modules that are responsible for data persistence and for communication with the time servers.
Cross-cutting SS	These modules have functionality that goes across different layers, such as security, logging, and I/O.
Service interfaces SS	These modules expose services that are consumed by the clients.
Business modules SS	These modules implement business operations.
Business entities SS	These entities make up the domain model.
DB access module	This module is responsible for persistence of business entities (objects) into the relational database. It performs object-oriented to relational mapping and shields the rest of the application from persistence details.

Element	Responsibility
Time server access module	This module is responsible for communication with the time servers. It isolates and abstracts operations with the time servers to support communication with different types of time servers (see QA-2).

The deployment diagram in Figure 4.4 sketches an allocation view that illustrates where the components associated with the modules in the previous diagram will be deployed.

The responsibilities of the elements are summarized here:

Element	Responsibility
User workstation	The user's PC, which hosts the client side logic of the application
Application server	The server that hosts server side logic of the application and also serves web pages
Database server	The server that hosts the legacy relational database
Time server	The set of (external) time servers

Also, information about relationships between some elements in the diagram that is worth recording is summarized in the following table:

Relationship	Description
Between web/app server and database server	Communication with the database will be done using the JDBC protocol.
Between web/app server and time server	The SNMP protocol is used (at least initially).

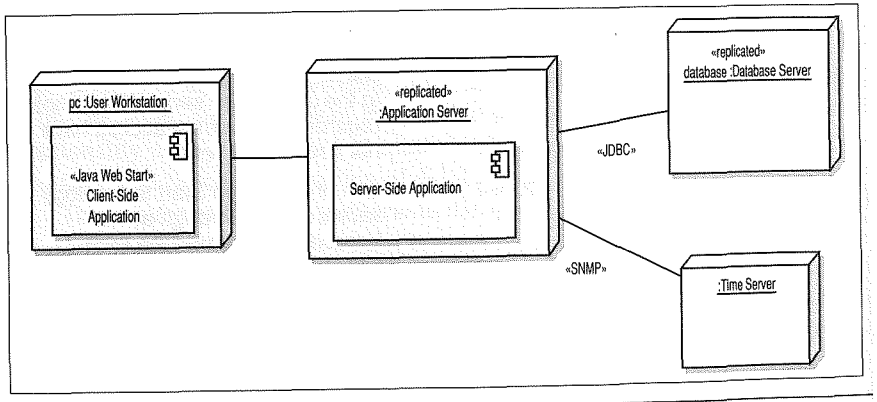


FIGURE 4.4 Initial deployment diagram for the FCAPS system (Key: UML)

4.3.2.6 Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

The following table summarizes the design progress using the Kanban board technique discussed in Section 3.8.2.

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
	UC-1		Selected reference architecture establishes the modules that will support this functionality.
	UC-2		Selected reference architecture establishes the modules that will support this functionality.
	UC-7		Selected reference architecture establishes the modules that will support this functionality.
QA-1			No relevant decisions made, as it is necessary to identify the elements that participate in the use case that is associated with the scenario.
	QA-2		Introduction of a time server access module in the data layer on the server application that encapsulates communication with the time servers. The details of this component and its interfaces have not been defined yet.
	QA-3		Identification of the elements derived from the deployment pattern that will need to be replicated.
QA-4			No relevant decisions made, as it is necessary to identify the elements that participate in the use case that is associated with the scenario.
	CON-1		Structuring the system using 3 tiers will allow multiple clients to connect to the application server. Decisions regarding concurrent access have not been made yet.
		CON-2	Use of Java Web Start technology allows access through a web browser to download the Rich Client. Since the Rich Client is being programmed in Java, this supports execution under Windows, OSX, and Linux.
		CON-3	Physically structure the application using the 3-tier deployment pattern, and isolate the database by providing database access components in the data layer of the application server.
	CON-4		Use of Java Web Start technology requires the client to be downloaded only the first time, and then when upgrades occur. This is helpful to support limited-bandwidth connections. More decisions need to be made regarding the communication between the presentation and the business logic layers.
CON-5			No relevant decisions made.
CON-6			No relevant decisions made.
		CRN-1	Selection of reference architectures and deployment pattern.
	CRN-2		Technologies that have been considered up to this point take into account the knowledge of the developers. Other technologies still need to be selected (e.g., communication with the time servers).
CRN-3			No relevant decisions made.

4.3.3 Iteration 2: Identifying Structures to Support Primary Functionality

This section presents the results of the activities that are performed in each of the steps of ADD in the *second* iteration of the design process for the FCAPS system. In this iteration, we move from the generic and coarse-grained descriptions of functionality used in iteration 1 to more detailed decisions that will drive implementation and hence the formation of development teams.

This movement from the generic to the specific is intentional, and built into the ADD method. We cannot design everything up front, so we need to be disciplined about which decisions we make, and when, to ensure that the design is done in a systematic way, addressing the biggest risks first and moving from there to ever finer details. Our goal for the first iteration was to establish an overall system structure. Now that this goal has been met, our new goal for this second iteration is to reason about the units of implementation, which affect team

formation, interfaces, and the means by which development tasks may be distributed, outsourced, and implemented in sprints.

4.3.3.1 Step 2: Establish Iteration Goal by Selecting Drivers

The goal of this iteration is to address the general architectural concern of *identifying structures to support primary functionality*. Identifying these elements is useful not only for understanding how functionality is supported, but also for addressing CRN-3—that is, the allocation of work to members of the development team.

In this second iteration, besides CRN-3, the architect considers the system's primary use cases:

- UC-1
- UC-2
- UC-7

4.3.3.2 Step 3: Choose One or More Elements of the System to Refine

The elements that will be refined in this iteration are the modules located in the different layers defined by the two reference architectures from the previous iteration. In general, the support of functionality in this system requires the collaboration of components associated with modules that are located in the different layers.

4.3.3.3 Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

In this iteration, several design concepts—in this case, architectural design patterns—are selected from the book *Pattern Oriented Software Architecture, Volume 4*. The following table summarizes the design decisions. The words in **bold** in the following table refer to architectural patterns from this book, and can be found in Appendix A.

Design Decisions and Location	Rationale and Assumptions
Create a **Domain Model** for the application	Before starting a functional decomposition, it is necessary to create an initial domain model for the system, identifying the major entities in the domain, along with their relationships. There are no good alternatives. A domain model must eventually be created, or it will emerge in a suboptimal fashion, leading to an ad hoc architecture that is hard to understand and maintain.
Identify **Domain Objects** that map to functional requirements	Each distinct functional element of the application needs to be encapsulated in a self-contained building block—a domain object. One possible alternative is to not consider domain objects and instead directly decompose layers into modules, but this increases the risk of not considering a requirement.

Design Decisions and Location	Rationale and Assumptions
Decompose **Domain Objects** into general and specialized **Components**	Domain objects represent complete sets of functionality, but this functionality is supported by finer-grained elements located within the layers. The "components" in this pattern are what we have referred to as modules. Specialization of modules is associated with the layers where they are located (e.g., UI modules). There are no good alternatives to decomposing the layers into modules to support functionality.
Use Spring framework and Hibernate	Spring is a widely used framework to support enterprise application development. Hibernate is an object to relational mapping (ORM) framework that integrates well with Spring. An alternative that was considered for application development is JEE. Spring was eventually selected because it was considered more "lightweight" and the development team was already familiar with it, resulting in greater and earlier productivity. Other ORM frameworks were not considered, as the development team already was familiar with, and happy with the performance of, Hibernate.

4.3.3.4 Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation design decisions made in this iteration are summarized in the following table:

Design Decisions and Location	Rationale
Create only an initial domain model	The entities that participate in the primary use cases need to be identified and modeled but only an initial domain model is created, to accelerate this phase of design.
Map the system use cases to domain objects	An initial identification of domain objects can be made by analyzing the system's use cases. To address CRN-3, domain objects are identified for all of the use cases in Section 4.2.1.
Decompose the domain objects across the layers to identify layer-specific modules with an explicit interface	This technique ensures that modules that support all of the functionalities are identified. The architect will perform this task just for the primary use cases. This allows another team member to identify the rest of the modules, thereby allocating work among team members. Having established the set of modules, the architect realizes the need to test these modules, so a new architectural concern is identified here: CRN-4: A majority of modules shall be unit tested. Only "a majority of modules" are covered by this concern because the modules that implement user interface functionality are difficult to test independently.

(*continues*)

Design Decisions and Location	Rationale
Connect components associated with modules using Spring	This framework uses an inversion of control approach that allows different aspects to be supported and the modules to be unit-tested (CRN-4).
Associate frameworks with a module in the data layer	ORM mapping is encapsulated in the modules that are contained in the data layer. The Hibernate framework previously selected is associated with these modules.

While the structures and interfaces are identified in this step of the method, they are captured in the next step.

4.3.3.5 Step 6: Sketch Views and Record Design Decisions

As a result of the decisions made in step 5, several diagrams are created.

- Figure 4.5 shows an initial domain model for the system.
- Figure 4.6 shows the domain objects that are instantiated for the use case model in Section 4.2.1.
- Figure 4.7 shows a sketch of a module view with modules that are derived from the business objects and associated with the primary use cases. Note that explicit interfaces are not shown but their existence is assumed.

The responsibilities for the elements identified in Figure 4.7 are summarized in the table that begins on page 95.

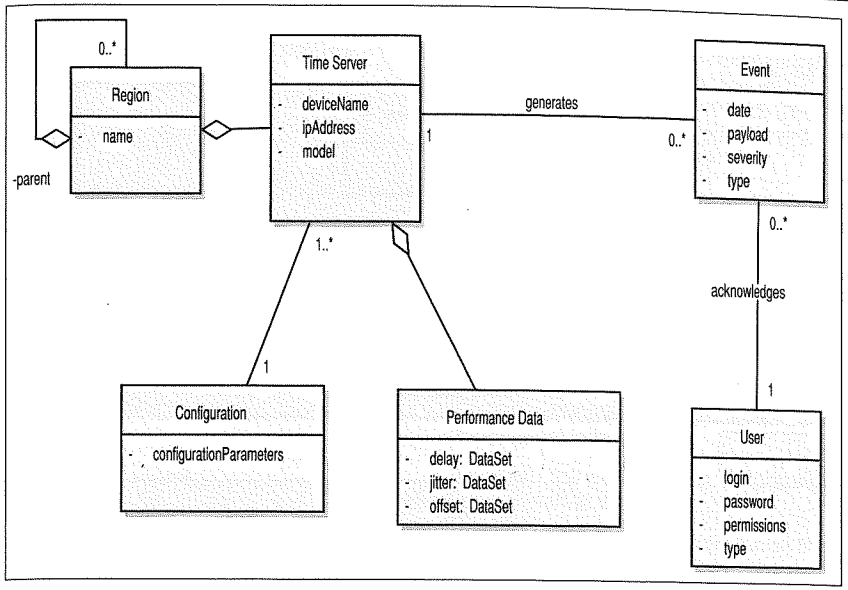


FIGURE 4.5 Initial domain model (Key: UML)

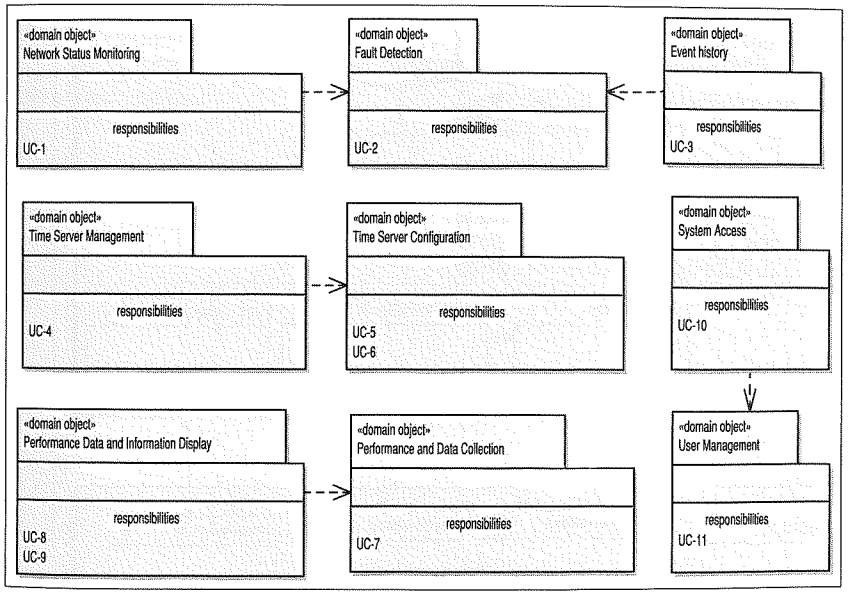


FIGURE 4.6 Domain objects associated with the use case model (Key: UML)

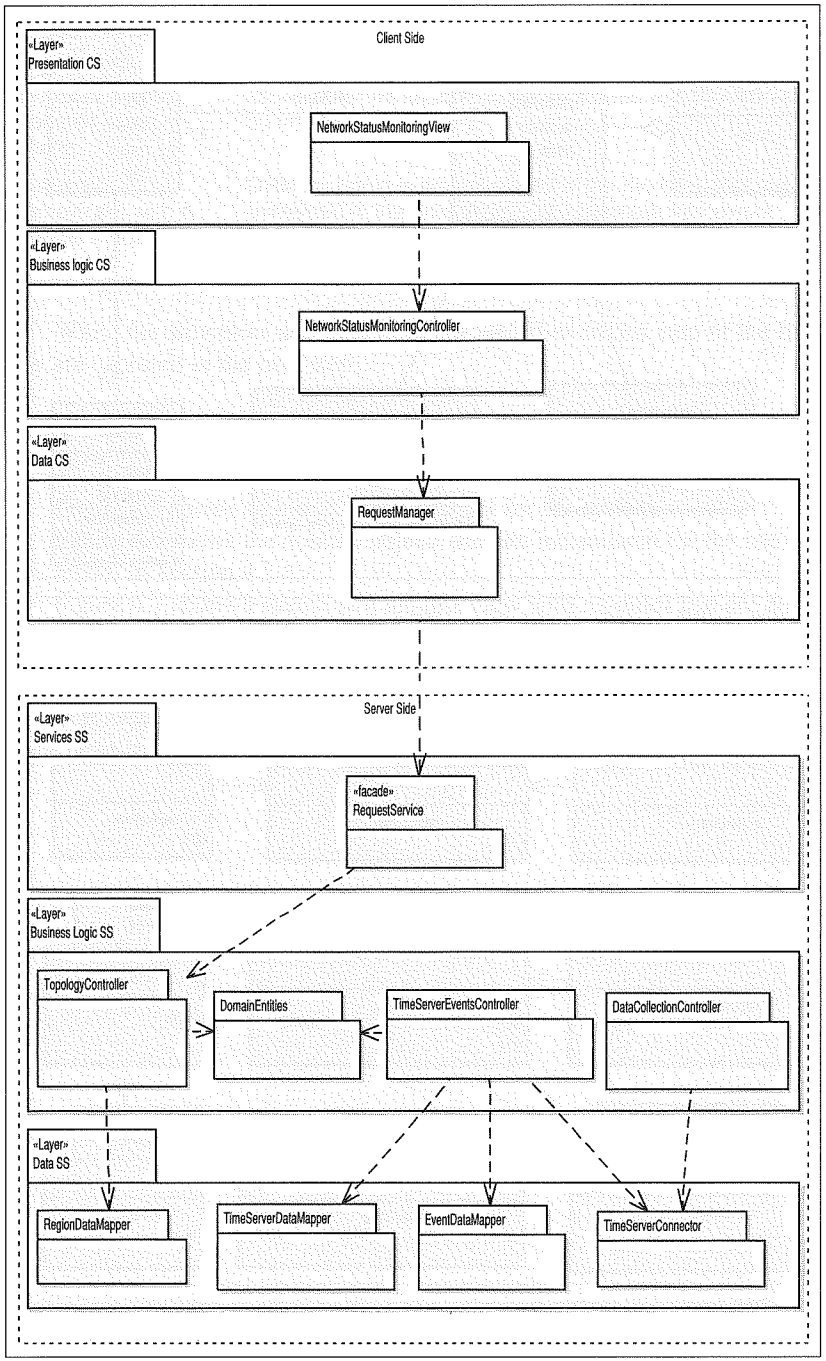


FIGURE 4.7 Modules that support the primary use cases (Key: UML)

Element	Responsibility
`NetworkStatusMonitoringView`	Displays the network representation and updates it when events are received. This component embodies both UI components and UI process components from the reference architecture.
`NetworkStatusMonitoringController`	Responsible for providing the necessary information to the presentation layer for displaying the network representation.
`RequestManager`	Responsible for communication with the server-side logic.
`RequestService`	Provides a facade that receives requests from the clients.
`TopologyController`	Contains business logic related to the topological information.
`DomainEntities`	Contains the entities from the domain model (server side).
`TimeServerEventsController`	Contains business logic related to the management of events.
`DataCollectionController`	Contains logic to perform data collection and storage.
`RegionDataMapper`	Responsible for persistence operations (CRUD) related to the regions.
`TimeServerDataMapper`	Responsible for persistence operations (CRUD) related to the time servers.
`EventDataMapper`	Responsible for persistence operations (CRUD) related to the events.
`TimeServerConnector`	Responsible for communication with the time servers. It isolates and abstracts operations with the time servers to support communication with different types of time servers (see QA-2).

The following sequence diagrams for UC-1 and UC-2 were created in the previous step of the method to define interfaces (as discussed in Section 3.6). A similar diagram was also created for UC-7 but is not shown here due to space limitations.

UC-1: Monitor Network Status

Figure 4.8 shows an initial sequence diagram for UC-1 (monitor network status). It shows how the user representation of the topology is displayed on startup (after the user has successfully logged into the system). Upon launch, the topology is requested from the `TopologyController` on the server. This element retrieves the root region through the `RegionDataMapper` and returns it to the client. The client can then populate the view by traversing the relationships within the `Region` class.

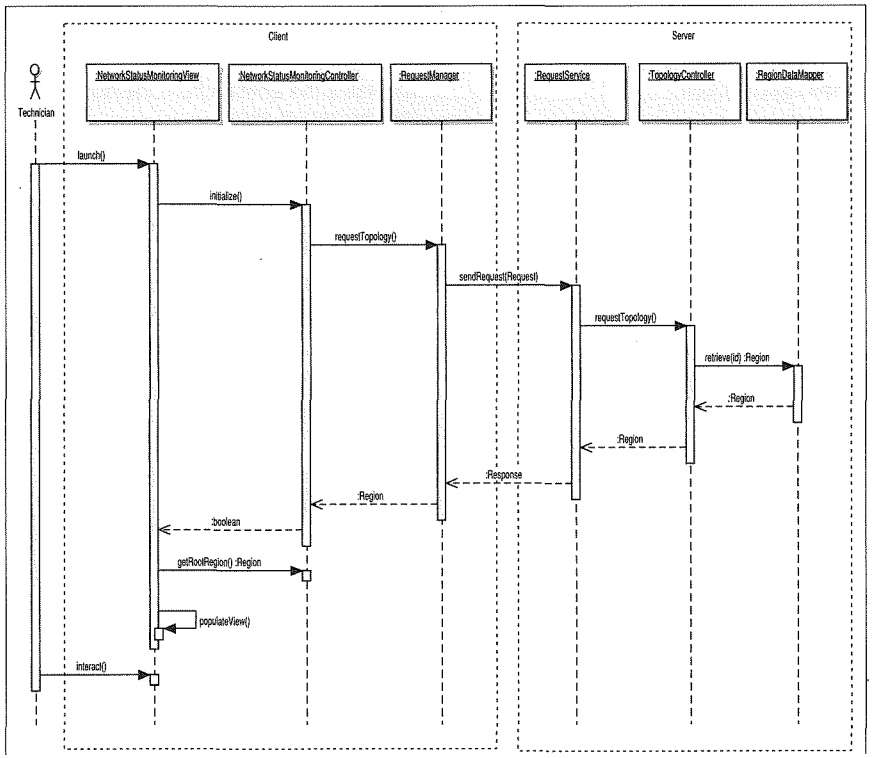


FIGURE 4.8 Sequence diagram for use case UC-1 (Key: UML)

From the interactions identified in the sequence diagram, initial methods for the interfaces of the interacting elements can be identified:

Method Name	Description
Element: `NetworkStatusMonitoringContoller`	
`boolean initialize()`	Opens up the network representation so that users can interact with it.
`Region getRootRegion()`	Returns a reference to the root region and the neighbors of this object (excluding traps).
Element: `RequestManager`	
`Region requestTopology()`	Requests the topology. This method returns a reference to the root region from which it is possible to navigate through the complete topology.
Element: `RequestService`	
`Response sendRequest(Request req)`	This method receives a request. Only this method is exposed in the service interface. This simplifies the addition of other functionality in the future without having to modify the existing service interface.
Element: `TopologyController`	
`Region requestTopology()`	Requests the topology. This method returns a reference to the root region from which it is possible to navigate through the complete topology.
Element: `RegionDataMapper`	
`Region retrieve(int id)`	Returns a `Region` from its `id`.

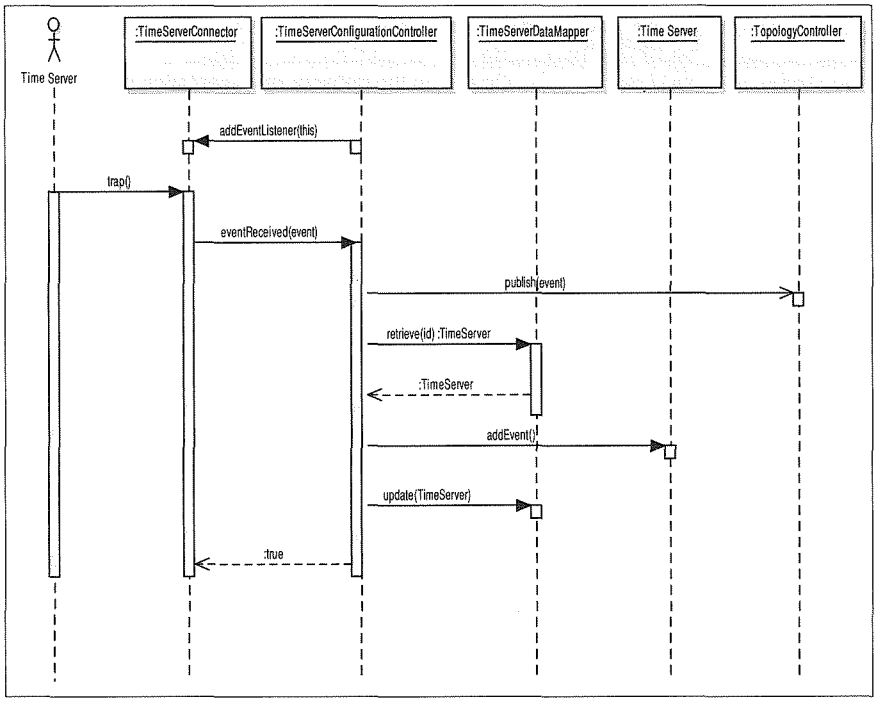


FIGURE 4.9 Sequence diagram for use case UC-2 (Key: UML)

UC-2: Detect Fault

Figure 4.9 shows an initial sequence diagram for UC-2 (detect fault) shows only the components on the server side. The interaction starts with a `TimeServer` sending a trap, which is received by the `TimeServerConnector`. The trap is transformed into an `Event` and sent to the `TimeServerConfigurationController`. The `Event` is sent asynchronously to the `TopologyController` for publication to the clients and is then persisted.

From this interaction, initial methods for the interfaces of the interacting elements can be identified:

Method Name	Description
Element: `TimeServerConnector`	
`boolean addEventListener (EventListener el)`	This method allows components from the business logic to register themselves as listeners to events that are received from the time servers.
Element: `TimeServerConfigurationController`	
`boolean eventReceived(Event evt)`	This callback method is invoked when an event is received.
Element: `TopologyController`	
`publish(Event evt)`	This method notifies the clients that a new event has occurred.
Element: `TimeServerDataMapper`	
`TimeServer retrieve(int id)`	Retrieves a `TimeServer` identified by its `id`.
`boolean update(TimeServer ts)`	Persists changes in a `TimeServer`.

4.3.3.6 Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

The decisions made in this iteration provided an initial understanding of how functionality is supported in the system. The modules associated with the primary use cases were identified by the architect, and the modules associated with the rest of the functionality were identified by another team member. From the complete list of modules, a work assignment table was created (not shown here) to address CRN-3.

Also, as part of module identification, a new architectural concern was identified and added to the Kanban board. Drivers that were completely addressed in the previous iteration are removed from the table.

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		UC-1	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-2	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-7	Modules across the layers and preliminary interfaces to support this use case have been identified.
	QA-1		The elements that support the associated use case (UC-2) have been identified.
	QA-2		The elements that support the associated use case (UC-5) have been identified.
	QA-3		No relevant decisions made.
	QA-4		The elements that support the associated use case (UC-7) have been identified.
	CON-1		No relevant decisions made.
	CON-4		No relevant decisions made.
	CON-5		Modules responsible for collecting data have been identified.
	CON-6		Modules responsible for collecting data storage been identified.
	CRN-2		Additional technologies were identified and selected considering the team's knowledge.
		CRN-3	Modules associated with all of the use cases have been identified and a work assignment matrix has been created (not shown).
	CRN-4		The architectural concern of unit-testing modules, which was introduced in this new iteration, is partially solved through the use of an inversion of control approach to connect the components associated with the modules.

4.3.4 Iteration 3: Addressing Quality Attribute Scenario Driver (QA-3)

This section presents the results of the activities that are performed in each of the steps of ADD in the third iteration of the design process. Building on the fundamental structural decisions made in iterations 1 and 2, we can now start to reason about the fulfillment of some of the more important quality attributes. This iteration focuses on just one of these quality attribute scenarios.

4.3.4.1 Step 2: Establish Iteration Goal by Selecting Drivers

For this iteration, the architect focuses on the QA-3 quality attribute scenario: A failure occurs in the management system during operation. The management system resumes operation in less than 30 seconds.

4.3.4.2 Step 3: Choose One or More Elements of the System to Refine

For this availability scenario, the elements that will be refined are the physical nodes that were identified during the first iteration:

- Application server
- Database server

4.3.4.3 Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

The design concepts used in this iteration are the following:

Design Decisions and Location	Rationale and Assumptions
Introduce the **active redundancy** tactic by replicating the application server and other critical components such as the database	By replicating the critical elements, the system can withstand the failure of one of the replicated elements without affecting functionality.
Introduce an element from the **message queue** technology family	Traps received from the time servers are placed in the message queue and then retrieved by the application. Use of a queue will guarantee that traps are processed and delivered in order (QA-1).

4.3.4.4 Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation design decisions are summarized in the following table:

Design Decisions and Location	Rationale
Deploy message queue on a separate node	Deploying the message queue on a separate node will guarantee that no traps are lost in case of application failure. This node is replicated using the tactic of active redundancy, but only one copy receives and treats events coming from the network devices.
Use active redundancy and load balancing in the application server	Because two replicas of the application server are active at any time, it makes sense to distribute and balance the load among the replicas. This tactic can be achieved through the use of the Load-Balanced Cluster pattern (see Section A.2.3). This introduces a new architectural concern, CRN-5: Manage state in replicas.
Implement load balancing and redundancy using technology support	Many technological options for load balancing and redundancy can be implemented without having to develop an ad hoc solution that would be less mature and harder to support.

The results of these instantiation decisions are recorded in the next step.

4.3.4.5 Step 6: Sketch Views and Record Design Decisions

Figure 4.10 shows a refined deployment diagram that includes the introduction of redundancy in the system.

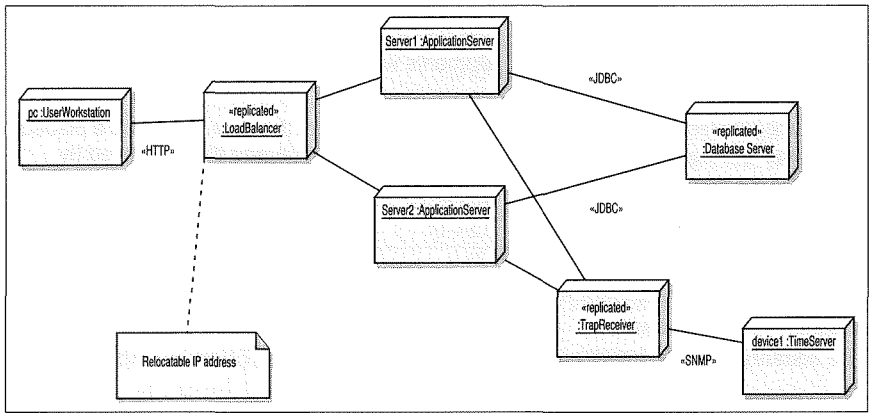


FIGURE 4.10 Refined deployment diagram (Key: UML)

The following table describes responsibilities for elements that have not been listed previously (in iteration 1):

Element	Responsibility
`LoadBalancer`	Dispatches (and balances the load of) requests coming from clients to the application servers. The load balancer also presents a unique IP address to the clients.
`TrapReceiver`	Receives traps from network devices, converts them into events, and puts these events into a persistent message queue.

The UML sequence diagram shown in figure 4.11 illustrates how the `TrapReceiver` that was introduced in this iteration exchanges messages with other elements shown in the deployment diagram to support UC-2 (detect fault), which is associated with both QA-3 (availability) and QA-1 (performance).

As the purpose of this diagram is to illustrate the communication that occurs between the physical nodes, the names of the methods are only preliminary; they will be refined in further iterations.

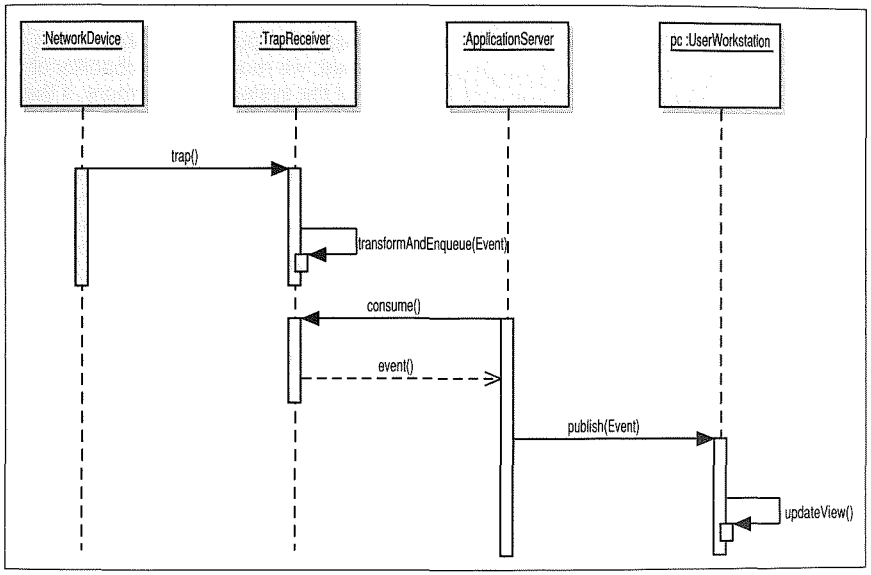


FIGURE 4.11 Sequence diagram illustrating the messages exchanged between the physical nodes to support UC-2 (Key: UML)

4.3.4.6 Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

In this iteration, important design decisions have been made to address QA-3, which also impacted QA-1. The following table summarizes the status of the different drivers and the decisions that were made during the iteration. Drivers that were completely addressed in the previous iteration have been removed from the table.

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
	QA-1		The introduction of a separate replicated trap receiver node can help ensure 100% of the traps are processed, even in the case of a failure of the application server. Furthermore, because trap reception is performed in a separate node, this approach reduces application server processing load, thereby helping performance. Because specific technologies have not been chosen, this driver is marked as "partially addressed".
	QA-2		No relevant decisions made.
	QA-3		By making the application server redundant, we reduce the probability of failure of the system. Furthermore, if the load balancer fails, a passive replica is activated within the required time period. Because specific technologies have not been chosen (message queue), this driver is marked as "partially addressed".
	QA-4		No relevant decisions made.
	CON-1		Replication of the application server and the use of a load balancer will help in supporting multiple user requests.
	CON-4		No relevant decisions made.
	CON-5		No relevant decisions made.
	CON-6		No relevant decisions made.
	CRN-2		No relevant decisions made.
	CRN-4		No relevant decisions made.
CRN-5			This new architectural concern is introduced in this iteration: manage state in replicas. At this point, no relevant decisions have been made.

4.4 Summary

In this chapter, we presented an example of using ADD to design a greenfield system in a mature domain. We illustrated three iterations with different foci: addressing a general concern, addressing functionality, and addressing one key quality attribute scenario.

The example followed the roadmap discussed in Section 3.3.1. It is interesting to observe that in the first iteration, two different reference architectures were used to structure the system. Also, the selection of externally developed components—in this case, frameworks—was carried out across the different iterations. Finally, the example illustrates how new architectural concerns appear as the design progresses.

This example demonstrates how architectural concerns, primary use cases, and quality attribute scenarios can be addressed as part of architectural design. In a real system, more iterations would be necessary to create a complete architecture design by addressing other scenarios with high priority.

In this example, we assumed that the architect is using a CASE tool during design, so diagrams were produced using UML. This is certainly not mandatory, as we will see in the case study presented in Chapter 5. Also, note that it is relatively simple to generate preliminary view sketches by using the information that is generated as part of the design process.

4.5 Further Reading

Appendix A provides descriptions and bibliographical references of all the design concepts used in this case study.