Control System: Traffic Management

The economics of software development have progressed to the point where many more kinds of applications are now automated than ever before, ranging from embedded microcomputers that control a myriad of automobile functions to tools that eliminate much of the drudgery associated with producing an animated film to systems that manage the distribution of interactive video services to millions of consumers. The distinguishing characteristic of all these larger systems is that they are extremely complex. Building systems so that their implementation is small is certainly an honorable task, but reality tells us that certain large problems demand large implementations. For some massive applications, it is not unusual to find software development organizations that employ several hundred programmers who must collaborate to produce millions of lines of code against a set of requirements that are guaranteed to be unstable during development. Such projects rarely involve the development of single programs; they more often encompass multiple, cooperative programs that must execute across a distributed target system consisting of many computers connected to one another in a variety of ways. To reduce development risk, such projects usually involve a central organization that is responsible for systems architecture and integration; the remaining work may be subcontracted to other companies or to other in-house organizations. Thus, the development team as a whole never assembles as one; it is typically distributed over space and—because of the personnel turnover common in large projects—over time.

Developers who are content with writing small, stand-alone, single-user, window-based tools may find the problems associated with building massive applications staggering—so much so that they view it as folly even to try. However, the actuality of the business and scientific world is such that

complex software systems must be built. Indeed, in some cases, it is folly not to try. Imagine using a manual system to control air traffic around a major metropolitan center or to manage the life-support system of a manned spacecraft or the accounting activities of a multinational bank. Successfully automating such systems not only addresses the very real problems at hand but also leads to a number of tangible and intangible benefits, such as lower operational costs, greater safety, and increased functionality. Of course, the operative word here is *successfully*. Building complex systems is plain hard work and requires the application of the best engineering practices we know, along with the creative insight of a few great designers.

This chapter tackles the development of such a problem.

9.1 Inception

To most people living in the United States, trains are an artifact of an era long past; in Europe and in many parts of Asia, the situation is entirely the opposite. Trains are an essential part of their transportation networks; tens of thousands of kilometers of track carry people and goods daily, both within cities and across national borders. In all fairness, trains do provide an important and economical means of transporting goods within the United States. Additionally, as major metropolitan centers grow more crowded, light rail transport is increasingly providing an attractive option for easing congestion and addressing the problems of pollution from internal combustion engines.

Still, railroads are a business and consequently must be profitable. Railroad companies must delicately balance the demands of frugality and safety and the pressures to increase traffic against efficient and predictable train scheduling. These conflicting needs suggest an automated solution to train traffic management, including computerized train routing and monitoring of all elements of the train system. Such automated and semiautomated train systems exist today in Sweden, Great Britain, West Germany, France, Japan [1], Canada, and the United States. The motivation for each of these systems is largely economic and social: Lower operating costs and more efficient use of resources are the goals, with improved safety as an integral by-product.

In this section, we begin our analysis of the fictitious Train Traffic Management System (TTMS) by specifying its requirements and the system use cases that further describe the required functionality.

Requirements for the Train Traffic Management System

Our experience with developing large systems has been that an initial statement of requirements is never complete, often vague, and always self-contradictory. For these reasons, we must consciously concern ourselves with the management of uncertainty during development, and therefore we strongly suggest that the development of such a system be deliberately allowed to evolve over time in an incremental and iterative fashion. As we pointed out in Chapter 6, the very process of development gives both users and developers better insight into what requirements are really important—far better than any paper exercise in writing requirements documents in the absence of an existing implementation or prototype. Also, since developing the software for a large system may take several years, software requirements must be allowed to change to take advantage of rapidly changing hardware technology.[1] It is undeniably futile to craft an elegant software architecture targeted to a hardware technology that is guaranteed to be obsolete by the time the system is fielded. This is why we suggest that, whatever mechanisms we craft as part of our software architecture, we should rely on existing standards for communications, graphics, networking, and sensors. For truly novel systems, it is sometimes necessary to pioneer new hardware or software technology. This adds risk to a large project, however, which already involves a customarily high risk. Software development clearly remains the technology of highest risk in the successful deployment of any large automated application, and our goal is to limit this risk to a manageable level, not to increase it.

This is a very large and highly complex system that in reality would not be specified by simple requirements. However, for this chapter, the requirements that follow will suffice for the purposes of our analysis and design effort. In the real world, a problem such as this could easily suffer from analysis paralysis because there would be many thousands of requirements, both functional and nonfunctional, with a myriad of constraints. Quite clearly, we would need to focus our efforts on the most critical elements and prototype candidate solutions within the operational context of the system under development.

---

1. In fact, for many such systems of this complexity, it is common to have to deal with many different kinds of computers. Having a well-thought-out and stable architecture mitigates much of the risk of changing hardware in the middle of development, an event that happens all too often in the face of the rapidly changing hardware business. Hardware products come and go, and therefore it is important to manage the hardware/software boundary of a system so that new products can be introduced that reduce the system's cost or improve its performance, while at the same time preserving the integrity of the system's architecture.

The Train Traffic Management System has two primary functions: train routing and train systems monitoring. Related functions include traffic planning, failure prediction, train location tracking, traffic monitoring, collision avoidance, and maintenance logging. From these functions, we define eight use cases, as shown in the following list.

- `Route Train`: Establish a train plan that defines the travel route for a particular train.
- `Plan Traffic`: Establish a traffic plan that provides guidance in the development of train plans for a time frame and geographic region.
- `Monitor Train Systems`: Monitor the onboard train systems for proper functioning.
- `Predict Failure`: Perform an analysis of train systems' condition to predict probabilities of failure relative to the train plan.
- `Track Train Location`: Monitor the location of trains using TTMS resources and the Navstar Global Positioning System (GPS).
- `Monitor Traffic`: Monitor all train traffic within a geographic region.
- `Avoid Collision`: Provide the means, both automatic and manual, to avoid train collisions.
- `Log Maintenance`: Provide the means to log maintenance performed on trains.

These use cases establish the basic functional requirements for the Train Traffic Management System, that is, they tell us *what* the system must do for its users. In addition, we have nonfunctional requirements and constraints that impact the requirements specified by our use cases, as listed here.

Nonfunctional requirements:

- Safely transport passengers and cargo
- Support train speeds up to 250 miles per hour
- Interoperate with the traffic management systems of operators at the TTMS boundary
- Ensure maximum reuse of and compatibility with existing equipment
- Provide a system availability level of 99.99%
- Provide complete functional redundancy of TTMS capabilities
- Provide accuracy of train position within 10.0 yards
- Provide accuracy of train speed within 1.5 miles per hour
- Respond to operator inputs within 1.0 seconds
- Have a designed-in capability to maintain and evolve the TTMS

Constraints:

- Meet national standards, both government and industry
- Maximize use of commercial-off-the-shelf (COTS) hardware and software

Now that we have our core requirements defined, at least at a very high level, we must turn our attention to understanding the users of the Train Traffic Management System. We find that we have three types of people who interact with the system: `Dispatcher`, `TrainEngineer`, and `Maintainer`. In addition, the Train Traffic Management System interfaces with one external system, `Navstar GPS`. These actors play the following roles within the TTMS.

- `Dispatcher` establishes train routes and tracks the progress of individual trains.
- `TrainEngineer` monitors the condition of and operates the train.
- `Maintainer` monitors the condition of and maintains train systems.
- `Navstar GPS` provides geolocation services used to track trains.

Determining System Use Cases

Figure 9–1 shows the use case diagram for the Train Traffic Management System. In it, we see the system functionality used by each of the actors. We also see that we have «`include`» and «`extend`» relationships used to organize relationships between several of the use cases. The functionality of the use case `Monitor Train Systems` is extended by the use case `Predict Failure`. During the course of monitoring systems, a failure prediction analysis (`condition: {request Predict Failure}`) can be requested for a particular system that is operating abnormally or may have been flagged with a yellow condition indicating a problem requiring investigation. This occurs at the `Potential Failure` extension point.

The functionality of the `Monitor Traffic` use case is also extended, by that of the `Avoid Collision` use case. Here, when monitoring train traffic, an actor has optional system capability to assist in the avoidance of a collision—at the `Potential Collision` extension point. This assistance can support both manual and automatic interventions. `Monitor Traffic` always includes the functionality of the `Track Train Location` use case to have a precise picture of the location of all train traffic. This is accomplished by using both TTMS resources and the Navstar GPS.

We may specify the details of the functionality provided by each of these use cases in textual documents called use case specifications. We have chosen to focus on the two primary use cases, `Route Train` and `Monitor Train`

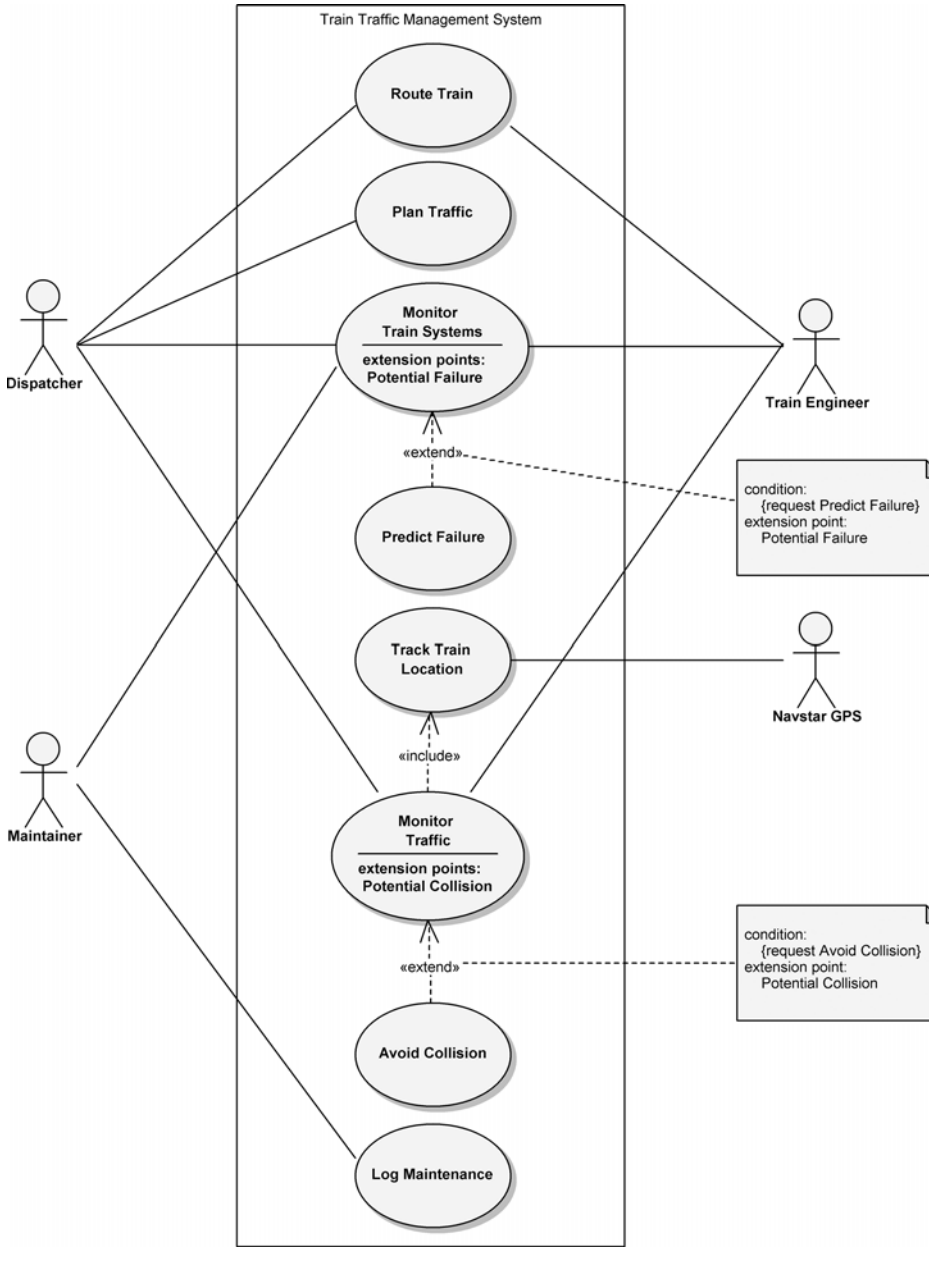

Figure 9–1 The Use Case Diagram for the Train Traffic Management System

`Systems`, in the following use case specifications. The format of the use case specification is a general one that provides setup information along with the primary scenario and one or more alternate scenarios.

It should be noted that these use case specifications focus on the boundary-level interaction between the users of the system and the Train Traffic Management System itself. This perspective is often referred to as a black-box view since the internal functioning of the system is not seen externally. This view is used when we are concerned with *what* the system does, not *how* the system does it.

Use case name: `Route Train`

Use case purpose: The purpose of this use case is to establish a train plan that acts as a repository for all the pertinent information associated with the route of one particular train and the actions that take place along the way.

Point of contact: Katarina Bach

Date modified: 9/5/06

Preconditions: A traffic plan exists for the time frame and geographic region (territory) relevant to the train plan being developed.

Postconditions: A train plan has been developed for a particular train to detail its travel route.

Limitations: Each train plan will have a unique ID within the system. Resources may not be committed for utilization by more than one train plan for a particular time frame.

Assumptions: A train plan is accessible by dispatchers for inquiry and modification and accessible by train engineers for inquiry.

Primary scenario:

A. The Train Traffic Management System (TTMS) presents the dispatcher with a list of options.

B. The dispatcher chooses to develop a new train plan.

C. The TTMS presents the template for a train plan to the dispatcher.

D. The dispatcher completes the train plan template, providing information about locomotive ID(s), train engineer(s), and waypoints with times.

E. The dispatcher submits the completed train plan to the TTMS.

F. The TTMS assigns a unique ID to the train plan and stores it. The TTMS makes the train plan accessible for inquiry and modification.

G. This use case ends.

Alternate scenarios:

Condition triggering an alternate scenario:

Condition 1: Develop a new train plan, based on an existing one.

B1. The dispatcher chooses to develop a new train plan, based on an existing one.

B2. The dispatcher provides search criteria for existing train plans.

B3. The TTMS provides the search results to the dispatcher.

B4. The dispatcher chooses an existing train plan.

B5. The dispatcher completes the train plan.

B6. The primary scenario is resumed at step E.

Condition triggering an alternate scenario:

Condition 2: Modify an existing train plan.

B1. The dispatcher chooses to modify an existing train plan.

B2. The dispatcher provides search criteria for existing train plans.

B3. The TTMS provides the search results to the dispatcher.

B4. The dispatcher chooses an existing train plan.

B5. The dispatcher modifies the train plan.

B6. The dispatcher submits the modified train plan to the TTMS.

B7. The TTMS stores the modified train plan and makes it accessible for inquiry and modification.

B8. This use case ends.

Use case name: `Monitor Train Systems`

Use case purpose: The purpose of this use case is to monitor the onboard train systems for proper functioning.

Point of contact: Katarina Bach

Date modified: 9/5/06

Preconditions: The locomotive is operating.

Postconditions: Information concerning the functioning of onboard train systems has been provided.

Limitations: None identified.

Assumptions: Monitoring of onboard train systems is provided when the locomotive is operating. Audible and visible indications of system problems, in addition to those via video display, are provided.

Primary scenario:

A. The Train Traffic Management System (TTMS) presents the train engineer with a list of options.

B. The train engineer chooses to monitor the onboard train systems.

C. The TTMS presents the train engineer with the overview status information for the train systems.

D. The train engineer reviews the overview system status information.

E. This use case ends.

Alternate scenarios:

Condition triggering an alternate scenario:

Condition 1: Request detailed monitoring of a system.

E1. The train engineer chooses to perform detailed monitoring of a system that has a yellow condition.

E2. The TTMS presents the train engineer with the detailed system status information for the selected system.

E3. The train engineer reviews the detailed system status information.

E4. The primary scenario is resumed at step B..

Extension point—`Potential Failure`**:**

Condition 2: Request a failure prediction analysis for a system.

E3-1. The train engineer requests a failure prediction analysis for a system.

E3-2. The TTMS performs a failure prediction analysis for the selected system.

E3-3. The TTMS presents the train engineer with the failure prediction analysis for the system.

E3-4. The train engineer reviews the failure prediction analysis.

E3-5. The train engineer requests that the TTMS alert the maintainer of the system that might fail.

E3-6. The TTMS alerts the maintainer of that system.

E3-7. The maintainer requests the failure prediction analysis for review.

E3-8. The TTMS presents the maintainer with the failure prediction analysis.

E3-9. The maintainer reviews the analysis and determines that the yellow condition is not severe enough to warrant immediate action.

E3-10. The maintainer requests that the TTMS inform the train engineer of this determination.

E3-11. The TTMS provides the train engineer with the determination of the maintainer.

E3-12. The train engineer chooses to perform detailed monitoring of the selected system.

E3-13. The alternate scenario is resumed at step E3.

Even though the requirements for the Train Traffic Management System are very simplified, we still have not completely specified them, and they are somewhat vague. This is not unlike what we've encountered while developing large, complex systems in the real world. As we've discussed previously, effectively managing ever-changing requirements is critical to having a successful development process, which we should all define as providing the right functionality, on time, and within budget. But don't think that our goal is to stop requirements from changing; we can't and we shouldn't want to do this. We can understand this if we focus on the rapid pace of functional enhancements made to hardware technology that, usually along with decreased cost, provide ever more solutions to our software development problems. Just look at the incredibly capable and complex software that can be run on today's personal computers, with their processors running at multigigahertz speeds and sporting gigabytes of random access memory (RAM).

So, how do we accommodate changing requirements, especially over development time frames that may encompass several years? We've found that using an iterative and incremental development process is one of the key means to managing the risks associated with changing requirements in such a large automated system. Another is designing an architecture that remains flexible throughout the development. Yet another is maximizing the use of COTS hardware and software, as one of the TTMS constraints directs us to do. As we proceed through this chapter, our prime focus will be on developing an architecture that can accommodate change.

9.2 Elaboration

Our attention now turns to developing the overall architecture framework for the Train Traffic Management System. We begin by analyzing the required system functionality that leads us into the definition of the TTMS architecture. From there, we begin our transition from systems engineering to the disciplines of hardware and software engineering. We conclude this section by describing the key abstractions and mechanisms of the TTMS.

Analyzing System Functionality

Now that the requirements for the Train Traffic Management System have been specified, our focus turns to *how* the system's aggregate parts provide this required functionality. This perspective is often referred to as a white-box view since the internal functioning of the system is seen externally. We use activity diagrams to analyze the various use case scenarios to develop this further level of detail.

Let's begin by looking at Figure 9–2, which analyzes the primary scenario of the `Route Train` use case. This activity diagram is relatively straightforward and follows the course of the use case scenario. Here we see the interaction of the `Dispatcher` actor and the `RailOperationsControlSystem`, which we've designated as the primary command and control center for the TTMS, as the `Dispatcher` creates a new `TrainPlan` object.

When determining the constituent elements of the TTMS, we must of course consider the requirements, both functional and nonfunctional, and the constraints. But we also have two competing technical concerns: the desire to encapsulate abstractions and the need to make certain abstractions visible to other elements. In other words, we must strive to design elements that are cohesive (by grouping logically related abstractions) and loosely coupled (by minimizing the dependencies among elements). Therefore, we define modularity as the property of a system that has been decomposed into a set of cohesive and loosely coupled elements.

In contrast to Figure 9–2, the activity diagram of Figure 9–3 is a bit more complicated because we've illustrated the first alternate scenario of the `Monitor Train Systems` use case, where the `TrainEngineer` chooses to perform a detailed monitoring of the `LocomotiveAnalysisandReporting` system, which has a yellow condition. Here we see that the constituent elements of the TTMS providing this capability are the `OnboardDisplay` system, the `LocomotiveAnalysisandReporting` system, the `EnergyManagement` system, and the `DataManagement` unit.

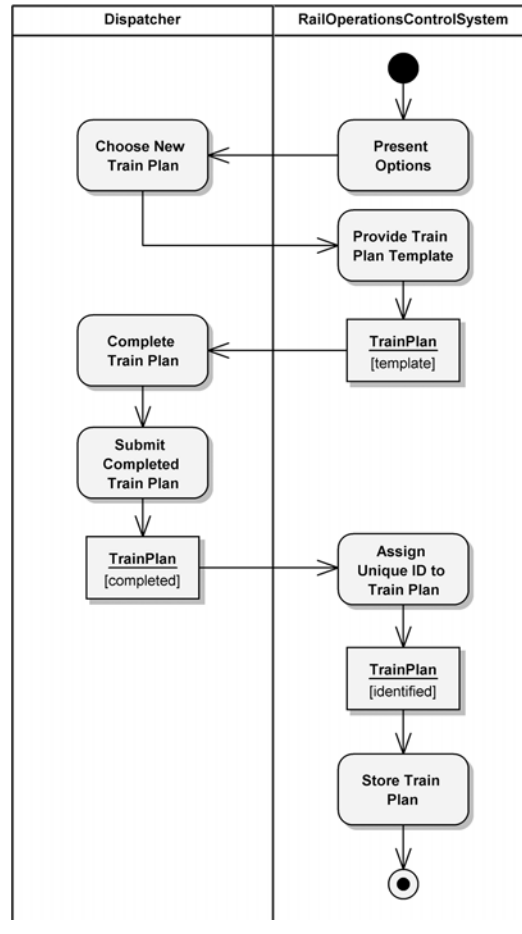

Figure 9–2 The `Route Train` Primary Scenario

We see that the `OnboardDisplay` system is the interface between the `TrainEngineer` and the TTMS. As such, it receives the `TrainEngineer`'s request to monitor the train systems and then requests the appropriate data from each of the other three systems. The overview level of status information is provided to the `TrainEngineer` for review. At this point, the `TrainEngineer` could remain at the overview level, which would end the primary scenario. In the alternate scenario, however, the `TrainEngineer` requests a more detailed review from the `LocomotiveAnalysisandReporting` system because it has presented a yellow condition indicating some type of problem that requires attention. In response, the `OnboardDisplay` system retrieves the detailed data from the system for presentation. After reviewing this information, the `TrainEngineer` returns to monitoring the overview level of system status information.

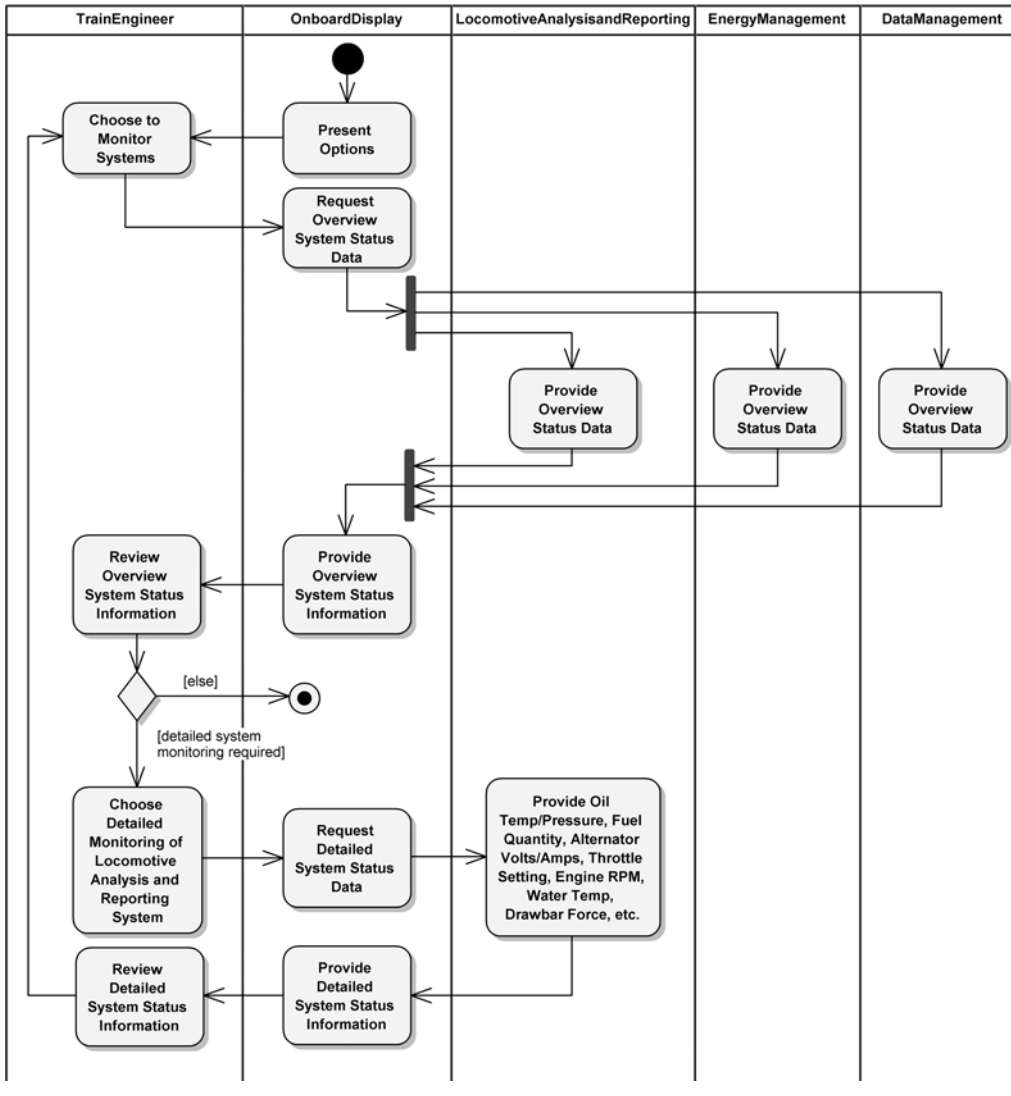

Figure 9–3 A `Monitor Train Systems` Alternate Scenario

It is a matter of project convention whether we regard the activity diagram in Figure 9–3 as representing one (alternate) or two (primary and alternate) separate scenarios. The second alternate scenario we described earlier details the extension of the `Monitor Train Systems` use case functionality with that of the `Predict Failure` use case. This scenario could be appended to Figure 9–3 to provide a more complete picture of system capability by detailing the actions whereby the `TrainEngineer` requests a failure prediction analysis (`condition: {request Predict Failure})` be run on the problematic system. In fact, we show this perspective in the Interaction Overview Diagram sidebar.