Control System: Traffic Management

The excerpt you've provided encapsulates the challenges and importance of developing large-scale, complex software systems, using examples ranging from traffic management to aerospace and banking. Let's break down some key points:

1. **Complexity of Modern Software Systems**: The passage highlights the increasing complexity of software systems due to technological advancements. These systems involve multiple components, distributed architectures, and millions of lines of code.
2. **Collaborative Development**: Developing such systems often requires collaboration among large teams of programmers distributed over space and time. This collaboration is essential for managing the complexity and scale of the project.
3. **Risk Management**: Large-scale software projects entail significant development risks, including evolving requirements and organizational challenges. Centralized architecture and integration teams help mitigate these risks.
4. **Necessity of Complex Systems**: Despite the daunting challenges, complex software systems are indispensable in various domains, from traffic management to banking. Automating these systems offers tangible benefits such as cost reduction, enhanced safety, and increased functionality.
5. **Engineering Practices and Creativity**: Successfully building complex systems demands a combination of rigorous engineering practices and creative insight from experienced designers. It's not just about writing code but also about designing robust architectures and scalable solutions.
6. **Real-world Examples**: The excerpt illustrates the significance of automated systems by highlighting real-world scenarios where manual operation would be impractical or unsafe, such as controlling air traffic or managing financial transactions.

Overall, the passage emphasizes the critical role of software engineering in addressing complex challenges and improving efficiency, safety, and functionality across various industries.

Software development has led to the automation of various applications, including embedded microcomputers, animated films, and video distribution systems. These systems are complex and require large implementations. Some large problems require large implementations, involving hundreds of programmers working together to produce millions of lines of code against unstable requirements. These projects often involve multiple, cooperative programs executed across a distributed target system. To reduce development risk, a central organization handles systems architecture and integration, while the remaining work may be subcontracted to other companies or in-house organizations. Developing complex software systems is challenging, but it can lead to benefits such as lower operational costs, greater safety, and increased functionality. Successfully automating such systems requires the application of best engineering practices and creative insight from designers.

1.INCEPTION

The excerpt discusses the significance of trains as a mode of transportation, particularly in regions like Europe and Asia, contrasting with their somewhat diminished role in the United States. Despite this, trains still play an important role in the US, especially for transporting goods and mitigating urban congestion and pollution.

However, running railroads as a business necessitates profitability, which involves balancing financial concerns with safety and efficiency. To achieve this balance, many countries have implemented automated or semi-automated train traffic management systems. These systems, found in various countries including

Sweden, Great Britain, and Japan, aim to reduce operating costs, optimize resource utilization, and enhance safety.

The excerpt sets the stage for the analysis of a fictional Train Traffic Management System (TTMS), outlining the importance of defining requirements and use cases to describe the system's functionality effectively. This introduction highlights the economic and social motivations behind implementing such systems and underscores their potential benefits in terms of cost reduction, resource efficiency, and safety enhancement.

1.1 Requirements for the Train Traffic Management System

The excerpt emphasizes the challenges associated with specifying requirements for large and complex systems like the Train Traffic Management System (TTMS). It highlights the inherent uncertainty and evolving nature of requirements during the development process. Due to the lengthy development timeline of such systems and the rapid pace of hardware technology evolution, the requirements must be allowed to change over time to adapt to new advancements.

To mitigate the risk associated with relying on potentially obsolete hardware technology, the excerpt suggests leveraging existing standards for communications, graphics, networking, and sensors in the software architecture. While pioneering new hardware or software technology may be necessary for truly novel systems, it also adds additional risk to an already high-risk project.

Given the enormity and complexity of the TTMS, attempting to specify every requirement in detail could lead to analysis paralysis. Therefore, the focus should be on identifying and prioritizing the most critical elements and prototyping candidate solutions within the operational context of the system under development. This iterative and incremental approach allows for better insight into the requirements and ensures that the development effort remains manageable.

The Train Traffic Management System (TTMS) is tasked with two **primary functions**: train routing and train systems monitoring. These functions encompass various related tasks such as traffic planning, failure prediction, train location tracking, traffic monitoring, collision avoidance, and maintenance logging. To address these functionalities effectively, the system defines eight distinct use cases:

1. **Route Train**: Establish a train plan specifying the travel route for a specific train.
2. **Plan Traffic**: Develop a traffic plan to guide the creation of train plans for a given time frame and geographic area.
3. **Monitor Train Systems**: Monitor the onboard systems of trains to ensure proper functioning.
4. **Predict Failure**: Analyze the condition of train systems to predict the likelihood of failures relative to the train plan.
5. **Track Train Location**: Monitor the real-time location of trains using resources from the TTMS and the Navstar Global Positioning System (GPS).
6. **Monitor Traffic**: Keep track of all train traffic within a designated geographic region.
7. **Avoid Collision**: Implement automatic and manual measures to prevent train collisions.
8. **Log Maintenance**: Provide functionality for logging maintenance activities performed on trains.

These use cases outline the core functionalities required for the effective operation of the Train Traffic Management System, addressing the diverse needs of train routing, monitoring, safety, and maintenance within a complex transportation network.

In addition to the functional requirements outlined by the use cases, the Train Traffic Management System (TTMS) must adhere to several **nonfunctional requirements** and constraints. These requirements specify how the system should perform and what standards it must meet. Here are the nonfunctional requirements and constraints:

1. **Safety**: Ensure the safe transportation of passengers and cargo.
2. **Speed**: Support train speeds of up to 250 miles per hour.
3. **Interoperability**: Interoperate with the traffic management systems of operators at the TTMS boundary.
4. **Compatibility**: Ensure maximum reuse of and compatibility with existing equipment.
5. **Availability**: Provide a system availability level of 99.99%.
6. **Redundancy**: Provide complete functional redundancy of TTMS capabilities to ensure system reliability.
7. **Accuracy of Position**: Ensure accuracy of train position within 10.0 yards.
8. **Accuracy of Speed**: Ensure accuracy of train speed within 1.5 miles per hour.
9. **Responsiveness**: Respond to operator inputs within 1.0 seconds to maintain real-time control and decision-making.
10. **Maintainability**: Have a designed-in capability to maintain and evolve the TTMS over time to adapt to changing requirements and technology advancements.

These nonfunctional requirements and constraints are essential for ensuring the safety, reliability, performance, and adaptability of the Train Traffic Management System while meeting the needs of its users and stakeholders. They provide additional criteria against which the system's effectiveness can be evaluated and verified.

Constraints:

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

Understanding the users and their roles within the Train Traffic Management System (TTMS) is crucial for designing an effective and user-friendly system. Here are the roles of the users and external system interfacing with 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.

1.2 Determining System Use Cases

Figure 9–1 illustrates the use case diagram for the Train Traffic Management System, depicting the system functionality and interactions with actors. It employs "include" and "extend" relationships to organize the relationships between various use cases. Specifically, the functionality of the "Monitor Train Systems" use case is extended by "Predict Failure," which involves conducting failure prediction analysis when abnormal operation or flagged issues arise, triggered at the Potential Failure extension point.

Additionally, the "Monitor Traffic" use case's functionality is extended by the "Avoid Collision" use case. Within this extension, actors have the optional capability to assist in collision avoidance during traffic monitoring, triggered at the Potential Collision extension point. This assistance can facilitate both manual and automatic interventions. Moreover, "Monitor Traffic" always includes the functionality of the "Track Train Location" use case to obtain precise information on the location of all train traffic, utilizing both TTMS resources and 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 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.

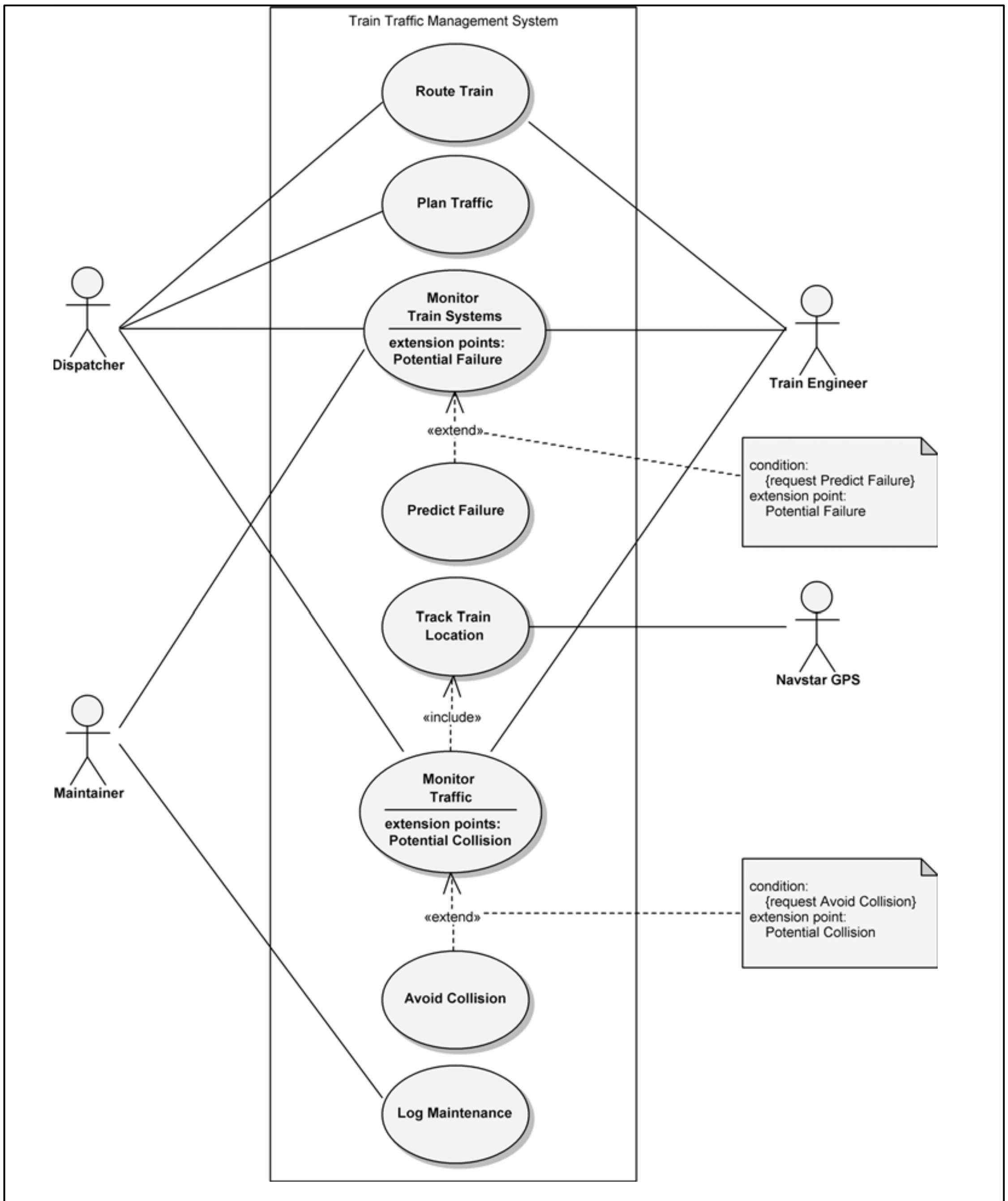
Train Traffic Management System
Route Train
Plan Traffic
Monitor Train Systems
extension points: Potential Failure
Dispatcher
Train Engineer
«extend»
condition: {request Predict Failure} extension point: Potential Failure
Predict Failure
Track Train Location
Navstar GPS
«include»
Maintainer
Monitor Traffic
extension points: Potential Collision
«extend»
condition: {request Avoid Collision} extension point: Potential Collision
Avoid Collision
Log Maintenance

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

That's a crucial point to highlight. By focusing on the boundary-level interaction between users and the Train Traffic Management System, these use case specifications adopt a black-box view. This perspective emphasizes what the system does, rather than how it achieves its functionality internally.

Use case name: Route Train

Use case purpose: This use case describes the process of establishing a train plan that defines the travel route for a particular train by the Dispatcher.

Point of Contact: Katarina Bach

Date Modified: 9/5/06

Preconditions:

- A traffic plan exists for the time frame and geographic region 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:

- The Train Traffic Management System (TTMS) presents the dispatcher with a list of options.
- The dispatcher chooses to develop a new train plan.
- The TTMS presents the template for a train plan to the dispatcher.
- The dispatcher completes the train plan template, providing information about locomotive ID(s), train engineer(s), and waypoints with times.
- The dispatcher submits the completed train plan to the TTMS.
- The TTMS assigns a unique ID to the train plan and stores it. The TTMS makes the train plan accessible for inquiry and modification.

Alternate Scenarios:

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

1. The dispatcher chooses to develop a new train plan, based on an existing one.
2. The dispatcher provides search criteria for existing train plans.
3. The TTMS provides the search results to the dispatcher.
4. The dispatcher chooses an existing train plan.
5. The dispatcher completes the train plan.
6. The primary scenario is resumed at step 5.

Condition 2: Modify an existing train plan.

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

2. The dispatcher provides search criteria for existing train plans.
3. The TTMS provides the search results to the dispatcher.
4. The dispatcher chooses an existing train plan.
5. The dispatcher modifies the train plan.
6. The dispatcher submits the modified train plan to the TTMS.
7. The TTMS stores the modified train plan and makes it accessible for inquiry and modification.

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:

1. The Train Traffic Management System (TTMS) presents the train engineer with a list of options.
2. The train engineer chooses to monitor the onboard train systems.
3. The TTMS presents the train engineer with the overview status information for the train systems.
4. The train engineer reviews the overview system status information.

Alternate Scenarios:

Condition 1: Request detailed monitoring of a system.

1. The train engineer chooses to perform detailed monitoring of a system that has a yellow condition.
2. The TTMS presents the train engineer with the detailed system status information for the selected system.
3. The train engineer reviews the detailed system status information.
4. The primary scenario is resumed at step 2.

Extension Point - Potential Failure:

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

1. The train engineer requests a failure prediction analysis for a system.
2. The TTMS performs a failure prediction analysis for the selected system.
3. The TTMS presents the train engineer with the failure prediction analysis for the system.
4. The train engineer reviews the failure prediction analysis.
5. The train engineer requests that the TTMS alert the maintainer of the system that might fail.
6. The TTMS alerts the maintainer of that system.
7. The maintainer requests the failure prediction analysis for review.
8. The TTMS presents the maintainer with the failure prediction analysis.
9. The maintainer reviews the analysis and determines that the yellow condition is not severe enough to warrant immediate action.
10. The maintainer requests that the TTMS inform the train engineer of this determination.
11. The TTMS provides the train engineer with the determination of the maintainer.
12. The train engineer chooses to perform detailed monitoring of the selected system.
13. The alternate scenario is resumed at step 3.

The Train Traffic Management System's requirements may seem simplified and somewhat vague, mirroring challenges encountered in real-world complex systems development. Effectively managing evolving requirements is crucial for success, defined as delivering the right functionality on time and within budget. Rather than stifling change, we embrace it, given the rapid advancements in hardware technology. Accommodating changing requirements over multi-year development cycles involves iterative and incremental processes, flexible architecture design, and leveraging commercial off-the-shelf (COTS) solutions, as guided by TTMS constraints. Our focus in this chapter will be on developing an adaptable architecture to meet these evolving needs.

2. ELABORATION

In the Elaboration phase, we delve into the Train Traffic Management System's required functionality, laying the groundwork for defining its architecture. This transition bridges systems engineering with hardware and software engineering disciplines. Throughout this process, we articulate the TTMS's key abstractions and mechanisms to ensure a comprehensive understanding of its architecture.

2.1 Analysing System Functionality

In analyzing the Train Traffic Management System's functionality, we employ a white-box perspective, scrutinizing its internal workings through activity diagrams. Figure 9–2 illustrates this approach, detailing the primary scenario of the Route Train use case. Here, the Dispatcher interacts with the RailOperationsControlSystem to create a new TrainPlan object. Balancing functional and nonfunctional requirements alongside technical considerations, we aim for modular design, emphasizing cohesion and loose coupling among system elements to ensure scalability and flexibility.

In contrast to Figure 9–2, the activity diagram in Figure 9–3 delves into the first alternate scenario of the Monitor Train Systems use case. Here, the TrainEngineer opts for detailed monitoring of the LocomotiveAnalysisandReporting system under a yellow condition. This scenario highlights the constituent elements of the TTMS responsible for this capability: the OnboardDisplay system, the LocomotiveAnalysisandReporting system, the EnergyManagement system, and the DataManagement unit. Balancing the need to encapsulate abstractions with the requirement for visibility among elements, we strive for modularity, creating cohesive and loosely coupled system components.

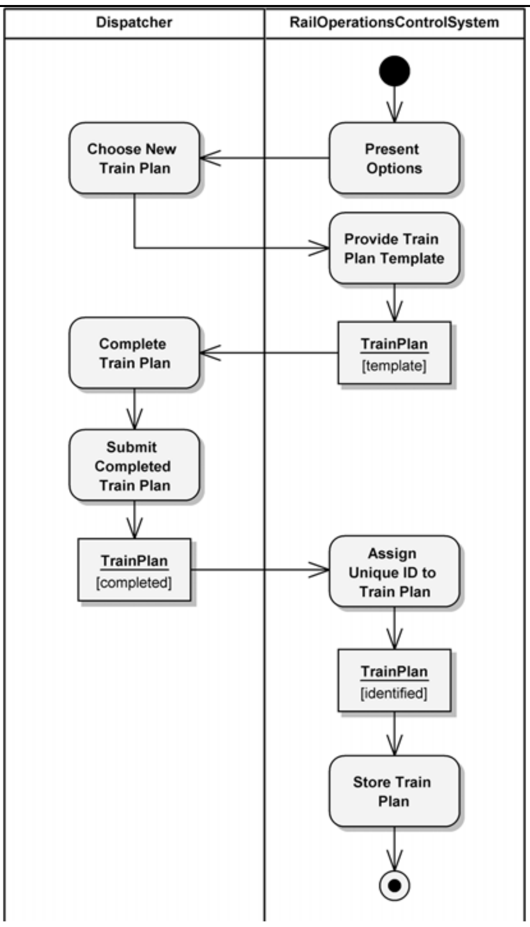

Figure 9–2 The `Route Train` Primary Scenario

We observe that the OnboardDisplay system acts as the intermediary between the TrainEngineer and the TTMS. It receives the TrainEngineer's request to monitor the train systems and subsequently solicits the necessary data from the other three systems. Initially, it provides the TrainEngineer with an overview of the system status for review. At this juncture, the TrainEngineer could conclude the interaction, completing the primary scenario. However, in the alternate scenario, the TrainEngineer opts for a more detailed examination prompted by the yellow condition reported by the LocomotiveAnalysisandReporting system. In response, the OnboardDisplay system retrieves detailed data from the system for presentation. Following the review, the TrainEngineer reverts to monitoring the overview level of system status information.

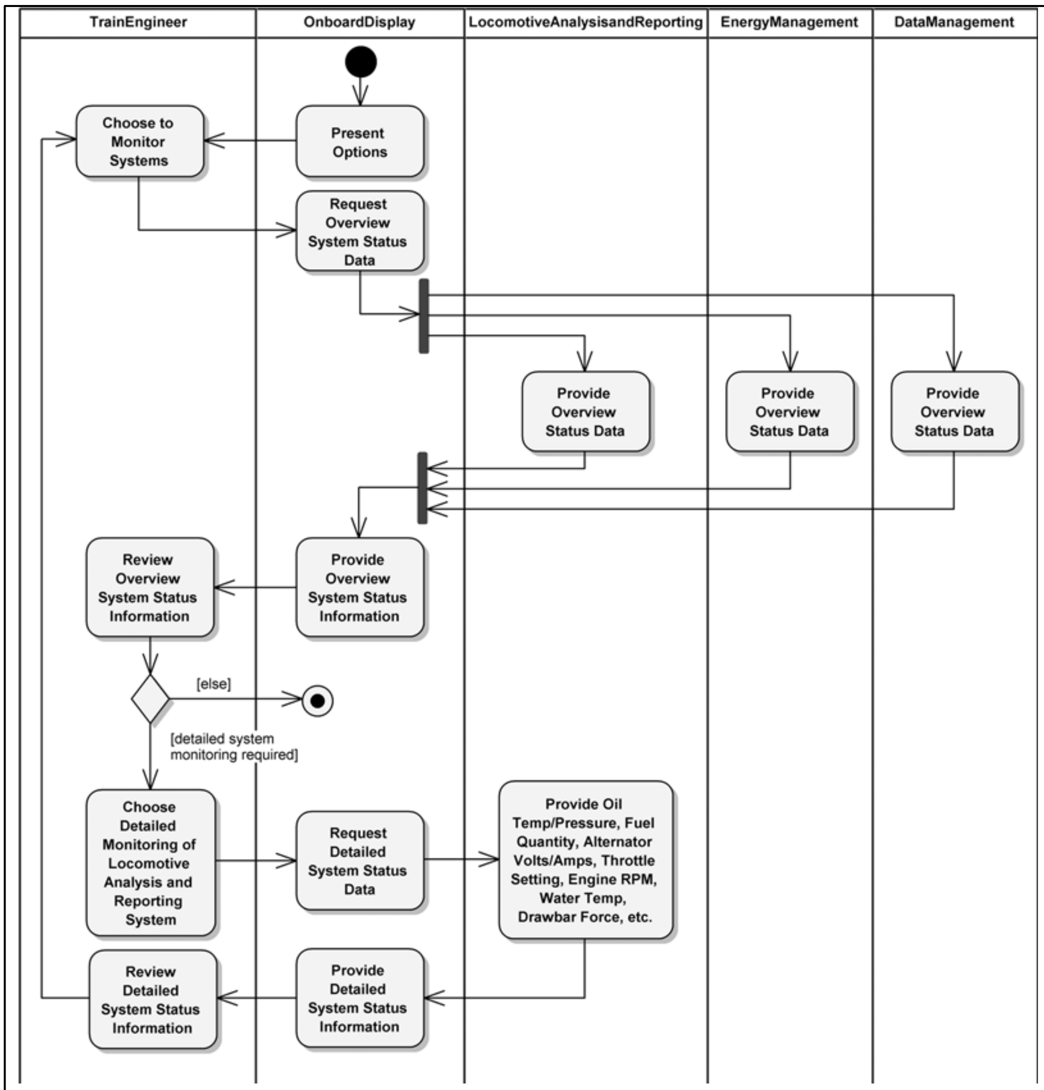
TrainEngineer
OnboardDisplay
LocomotiveAnalysisandReporting
EnergyManagement
DataManagement
Present Options
Choose to Monitor Systems
Request Overview System Status Data
Provide Overview Status Data
Provide Overview Status Data
Provide Overview Status Data
Provide Overview System Status Information
Review Overview System Status Information
[else]
[detailed system monitoring required]
Choose Detailed Monitoring of Locomotive Analysis and Reporting System
Request Detailed System Status Data
Provide Oil Temp/Pressure, Fuel Quantity, Alternator Volts/Amps, Throttle Setting, Engine RPM, Water Temp, Drawbar Force, etc.
Provide Detailed System Status Information
Review Detailed System Status Information

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

It's indeed a matter of project convention whether we consider the activity diagram in Figure 9–3 as depicting one alternate scenario or two separate scenarios (primary and alternate). The second alternate scenario we discussed earlier illustrates the extension of the Monitor Train Systems use case functionality with that of the Predict Failure use case. This scenario could be added to Figure 9–3 to offer a more comprehensive view of system capability, outlining the steps through which the TrainEngineer requests a failure prediction analysis (condition: {request Predict Failure}) for the problematic system. In fact, we present this perspective in the Interaction Overview Diagram sidebar.

2.2 Defining the TTMS Architecture

The locomotive analysis and reporting system is a crucial component of the Train Traffic Management System, comprising various digital and analog sensors for monitoring locomotive conditions. These sensors track parameters such as oil temperature, oil pressure, fuel quantity, alternator volts and amperes, throttle setting, engine RPM, water temperature, and drawbar force. The sensor data is relayed to the train engineer via the onboard display system and to dispatchers and maintainers elsewhere on the network. Any deviations from normal operating ranges trigger warning or alarm conditions, and a log of sensor values is maintained to facilitate maintenance and fuel management.

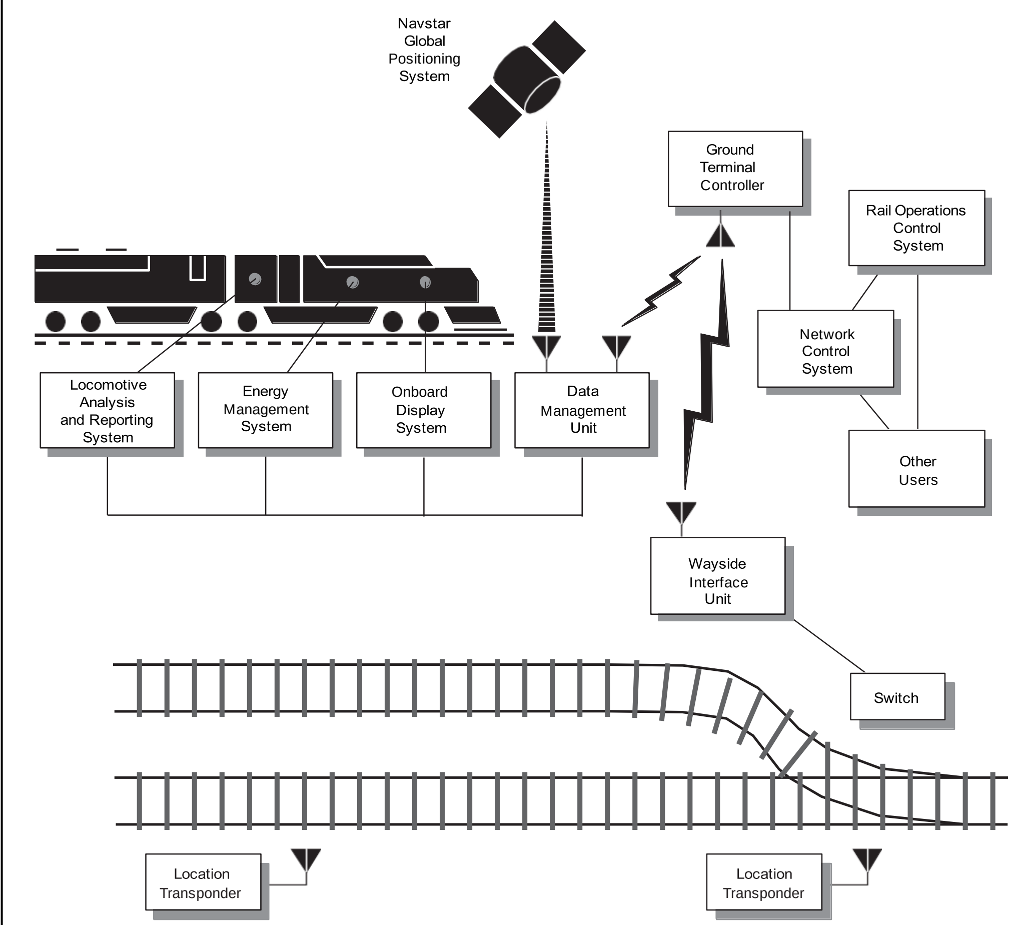

Figure 9–5 The Train Traffic Management System

The energy management system advises the engineer on optimal throttle and brake settings based on factors like track conditions and schedule, displayed on the onboard system. The onboard display acts as an interface, showing data from various systems and allowing navigation. The data management unit facilitates communication between onboard systems and the network. Train location tracking uses transponders and GPS, with the onboard system showing estimated positions and receiving data from transponders and GPS receivers for precise location information.

The wayside interface units control track devices and report their status, interfacing with ground terminal controllers, which relay data to passing trains and manage the network's health. Ground terminal controllers connect to a central network control system, which monitors the network and routes information in case of failures. Dispatch centers manage train routes, track progress, and ensure safety, with dispatchers controlling different territories. The system must accommodate changing track layouts, equipment, and train routes, requiring flexibility and adherence to national standards while maximizing the use of commercial off-the-shelf hardware and software.

2.3 From Systems Engineering to Hardware and Software Engineering

Up to this point, our focus has been on systems engineering, analyzing use case scenarios to define functional requirements and develop a candidate architecture for the Train Traffic Management System (TTMS). As we progress, the architecture of lower-level hardware and software elements will evolve, guided by our system architects. Eventually, we'll determine which parts of the system will be implemented in hardware, software, or manual operations, requiring collaboration among engineering teams. The candidate architecture, depicted in Figure 9–5, reflects an object-oriented approach, emphasizing component-based design with cohesive and loosely coupled elements. Further analysis will involve activity diagrams for system-level perspectives and more detailed interactions, utilizing sequence diagrams, class diagrams, and prototypes as we delve into lower architectural levels.

Figure 9–6 depicts a sequence diagram illustrating the automated processing of a daily train order, offering more detailed insights into the Train Traffic Management System (TTMS) than the previous activity diagram. This scenario assumes the creation of a new train plan, as concluded in Figure 9–2. Major events and system element interactions are highlighted. As we progress in development, we'll delve into detailed documentation of element attributes, operation signatures, and association specifications. Our systems engineering analysis must allocate requirements to hardware, software, and operational elements, recognizing that some functionality, particularly in safety-critical areas, may necessitate human-in-the-loop design. Allocation decisions depend on various factors, including reuse considerations, commercial availability, and architectural preferences. Commercially available items, such as sensors, may be allocated to vendor engineers, while software provides flexibility and hardware ensures performance where essential.

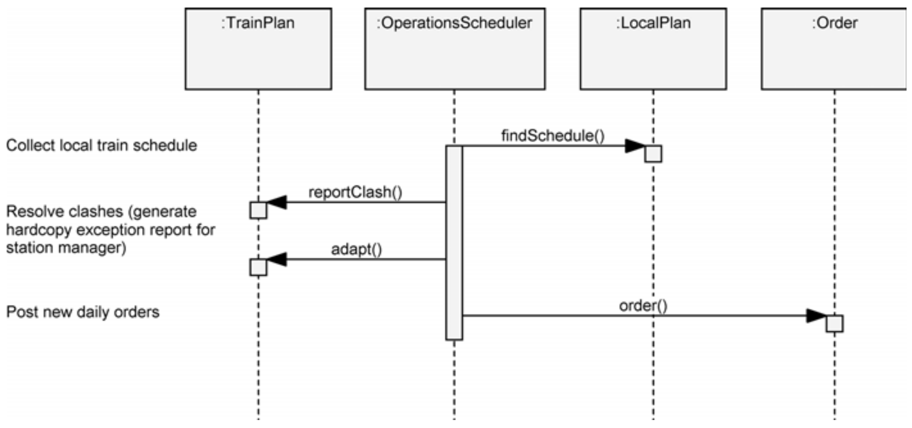

Figure 9–6 A Scenario for Processing Daily Train Orders

The initial hardware architecture for the Train Traffic Management System has been chosen by system architects, providing a starting point for allocating software requirements. As we progress with analysis and design, we retain the flexibility to trade off between hardware and software solutions based on evolving needs. Figure 9–7 illustrates the deployment hardware using deployment diagrams, mirroring the system's block diagram in Figure 9–5. Each train hosts a computer handling various systems, while location transponders and wayside devices are controlled by separate computers. Communication between devices occurs via transmitters and receivers using various mediums. Ground terminal controllers connect to local area networks at dispatch centers, each equipped with primary and backup computers for uninterrupted service.

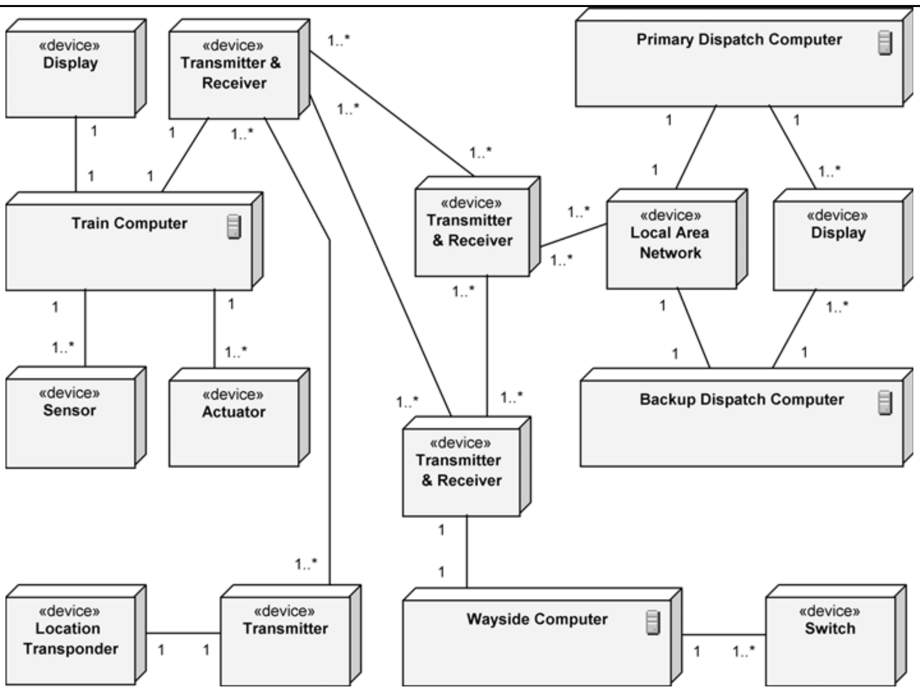
«device» Display
«device» Transmitter & Receiver
Primary Dispatch Computer
Train Computer
«device» Transmitter & Receiver
«device» Local Area Network
«device» Display
«device» Sensor
«device» Actuator
Backup Dispatch Computer
«device» Transmitter & Receiver
«device» Location Transponder
«device» Transmitter
Wayside Computer
«device» Switch
1
1..*

Figure 9–7 The Deployment Diagram for the Train Traffic Management System

In operational mode, the Train Traffic Management System involves numerous computers, with one per train, wayside interface unit, and two at each dispatch center. While the deployment diagram displays only a subset due to redundancy, maintaining clarity during development hinges on well-defined interfaces among system elements. Initially, interfaces can be loosely defined but should swiftly become formalized to enable parallel development, testing, and release. This approach facilitates hardware/software trade-offs without disrupting completed parts of the system and accommodates diverse developers' understanding within a large, potentially globally distributed, development organization. The specification of key abstractions and mechanisms is entrusted to experienced architects for consistency and coherence.

2.4 Key Abstractions and Mechanisms

A study of the requirements for the Train Traffic Management System suggests that we really have four different subproblems to solve:

1. **Networking**: The system relies heavily on a distributed communications network for the transmission of messages between various components. Ensuring timely and reliable communication across this network is crucial for safe and efficient operation.
2. **Database**: The system requires robust database management to store and retrieve critical information related to train schedules, routes, maintenance records, and more. Any issues with database functionality could disrupt the entire system's operations.
3. **Human/machine interface**: The interface between human operators and the system is essential for effective monitoring, control, and decision-making. Designing intuitive and efficient interfaces that facilitate user interaction without causing confusion or errors is vital.
4. **Real-time analog and digital device control**: The system must interact with a variety of analog and digital devices onboard trains and at wayside locations. This involves real-time control of sensors, actuators, switches, and other equipment to ensure safe and efficient train operations.

Failure to address any of these subproblems adequately could result in system inefficiencies, safety risks, or operational disruptions, hence the need to manage their development with careful attention and priority.

Managing real-time train tracking and routing involves maintaining consistent data across the network, akin to a distributed database. This ensures accurate train location and route information despite simultaneous updates and queries.

Designing user interfaces for train engineers and dispatchers requires simplicity and intuitiveness, catering to users who may not have advanced computer skills. Interfaces must prioritize ease of use and domain-specific usability for optimal efficiency and reduced errors.

To address these challenges, the system must handle sensor data and control mechanisms consistently across various devices. Each of the four subproblems—networking, database management, user interface design, and real-time device control—requires focused attention from experts in parallel with each other. This iterative and incremental approach ensures that analysis informs architecture, and design uncovers new aspects requiring further refinement.

In summary, our domain analysis reveals three fundamental abstractions:

1. **Trains:** comprising locomotives and cars
2. **Tracks:** encompassing profile, grade, and wayside devices
3. **Plans:** including schedules, orders, clearances, authority, and crew assignments

Each train has a location on the tracks and is associated with exactly one active plan, while each point on the tracks may have zero or one trains.

“1”Basically, we've pinpointed four main strategies to tackle the biggest challenges in development.
“1”In essence, we've identified four key mechanisms for addressing the main development risks:
“1”Continuing, we may devise a key mechanism for each of the four nearly independent subproblems:

1. Message passing
2. Train schedule planning
3. Displaying information
4. Sensor data acquisition

These mechanisms are central to our system's functionality, so it's crucial to engage our top system architects to explore various approaches and establish a framework for subsequent development by junior team members.

3. CONSTRUCTION

In construction, we lay down the foundational structure of the system by defining its core classes and the interactions between them. By focusing on the key mechanisms early on, we address the most critical aspects of the system and solidify the vision of the system's architects. These initial products form the basis upon which the rest of the system is developed. We delve into the details of each key mechanism, such as message passing and sensor data acquisition, followed by a discussion on release management to facilitate our iterative development process. Finally, we explore how building the system architecture aids in specifying the subsystems of the TTMS.

3.1 Message Passing

Message passing in the Train Traffic Management System doesn't involve method calls like in programming languages. Instead, it's about communication at a higher level of abstraction within the problem domain. For instance, messages could be signals to control devices along the tracks, notifications of trains passing certain points, or instructions from dispatchers to train engineers.These messages are exchanged at two main levels: between computers and devices, and among different computers within the system.

Our focus lies on the second level of message passing within the Train Traffic Management System, where communication occurs among different computers. Given the geographically distributed nature of our system, we need to address concerns like noise, equipment failure, and security. To identify these messages, we can analyze each pair of communicating computers, as depicted in our previous deployment diagram (refer back to Figure 9–7). For each pair, we should consider three questions:

1. What data does each computer handle?
2. What data needs to be exchanged between these computers?
3. At what level of abstraction should this data be communicated?

The solution to these questions isn't fixed; rather, we need to iterate until we're confident in our message definitions and ensure there are no communication bottlenecks. At this stage, it's crucial to focus on the essence of messages rather than their technical format. Prematurely choosing a low-level representation can lead to disruptions later. Instead, we should concentrate on the roles and operations of each message.

The class diagram in Figure 9–8 captures our design decisions regarding important messages in the Train Traffic Management System. Messages are instances of a generalized abstract class named Message, with subclasses like TrainStatusMessage and TrainPlanMessage. Stabilizing the interfaces of these key message classes early on is wise. We can then build programs to simulate message creation and reception for testing.

While the class diagram may be incomplete initially, an object-oriented architecture allows incremental addition of messages without major disruptions. Ultimately, the message-passing mechanism should ensure reliable delivery while abstracting away implementation details for clients. This enables simplification in handling message transmission and reception.

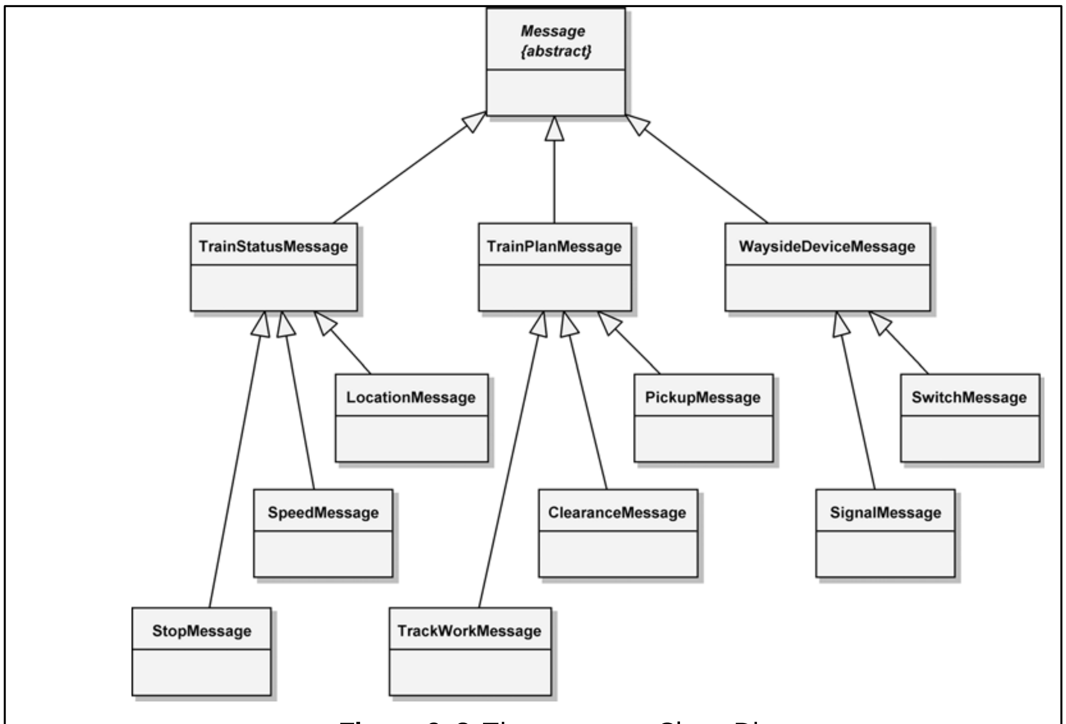

Figure 9–8 The `Message` Class Diagram

Figure 9–9 illustrates our message-passing mechanism design. To send a message, a client creates a new message and sends it to its node's message manager for queuing. The message manager then uses a Transporter object to broadcast the message across the network. This process is asynchronous, indicated by the open-headed arrow, to avoid client waiting due to radio link delays.

Upon receiving the message, a Listener object presents it to its node's message manager, which creates a parallel message and queues it. A receiver can wait for the next message to arrive using the nextMessage() operation, a synchronous operation, at the head of its message manager's queue.

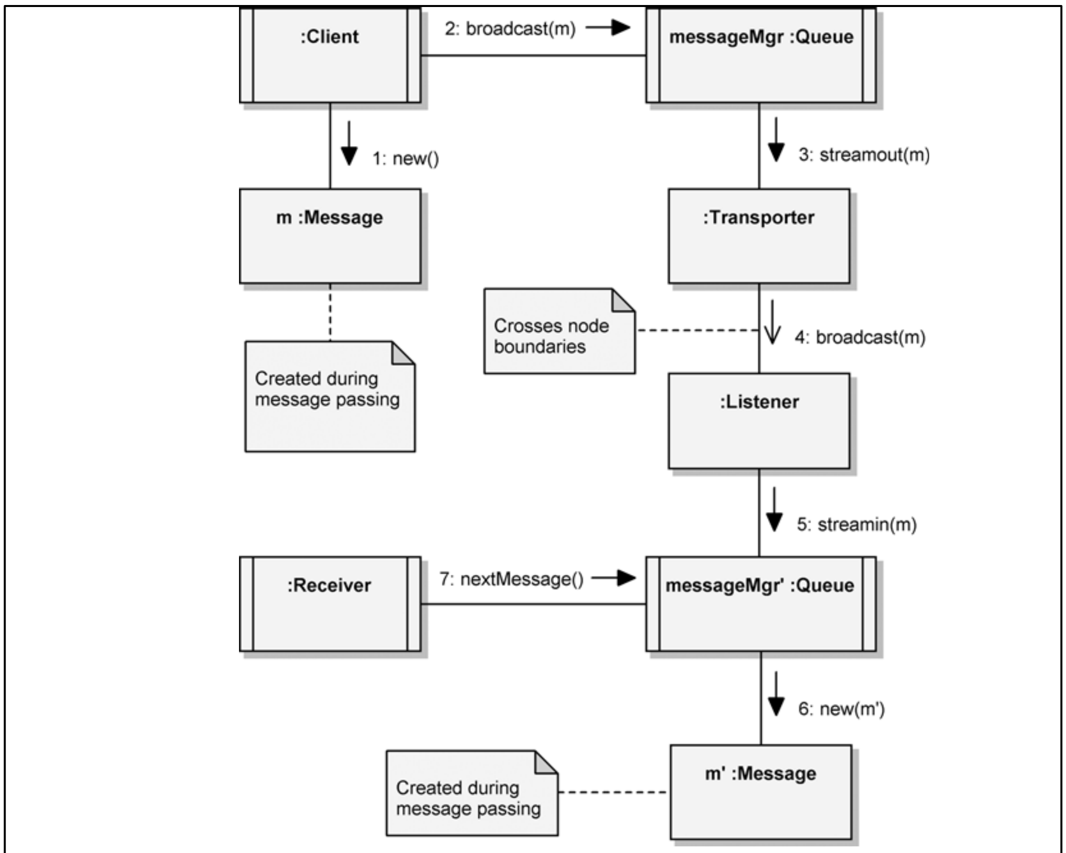

Figure 9–9 The Message-Passing Mechanism

Our message manager design situates it at the application layer in the OSI model for networks, enabling clients to interact with application-specific messages. However, the final implementation may involve additional complexities, such as encryption, decryption, and error detection and correction codes, to ensure reliable communication in noisy or error-prone environments.

3.2 Train Schedule Planning

The Train Schedule Planning phase involves defining the components of a train plan, considering various client needs for plan creation, modification, and access. The TrainPlan class diagram outlines these components, indicating that each plan includes a crew, multiple general orders, and ordered actions with details like time, location, speed, and authority. For instance, a train plan could include actions such as departure, arrival, and speed changes, each with specific parameters.

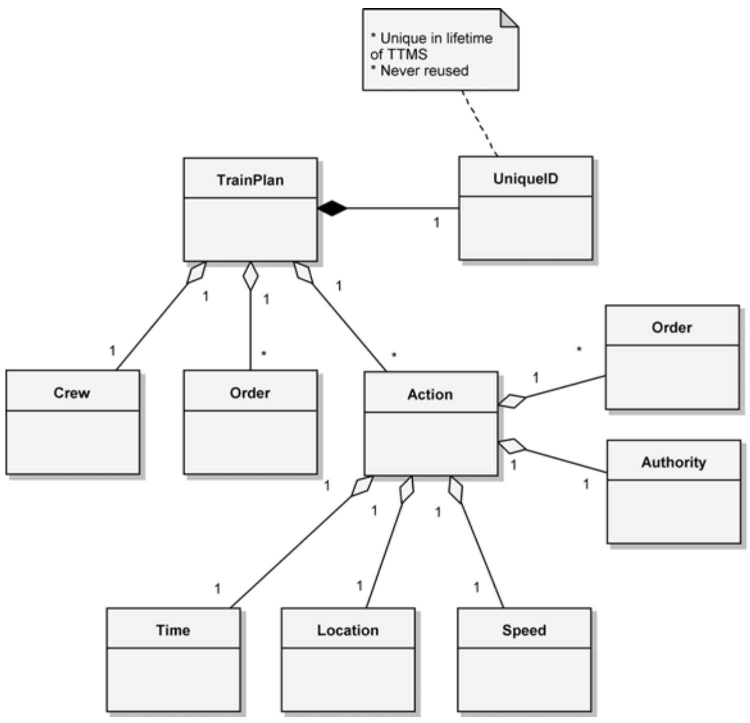

Figure 9–10 The `TrainPlan` Class Diagram

The TrainPlan class includes a UniqueId attribute for identifying each instance uniquely. Certain classes in the diagram, like UniqueID and Crew, are standalone entities due to their complexity and specific regulations they must adhere to. For instance, Crews have location and speed restrictions imposed on their work. Similar to the Message class, the key elements of a train plan are designed early, with details evolving as plans are applied to different client needs over time.

In a hypothetical scenario without communication issues or resource constraints, a centralized database would store all train plans, ensuring unique instances. However, practical constraints like communication delays and limited computing resources make this approach unrealistic. Accessing plans from a dispatch center to a train wouldn't meet real-time needs.

To create the illusion of a single, centralized database, we store train plans on dispatch center computers and distribute copies as needed across the network. Each train computer maintains a copy of its current plan for efficient onboard querying. Changes to plans, initiated by dispatchers or train engineers, trigger updates across all copies. The train schedule planning mechanism, illustrated in Figure 9–11, manages this process. It involves a centralized database holding primary versions of plans and distributes mirror-image copies. Requests for specific plans prompt cloning from the primary version, with updates synchronized across all copies using network messages and a record-locking mechanism for consistency.

Table 9–1 Actions a Train Plan Might Contain

Time	Location	Speed	Authority	Orders
0800	Pueblo	As posted	See yardmaster	Depart yard
1100	Colorado Springs	40 mph		Set out 30 cars
1300	Denver	45 mph		Set out 20 cars
1600	Pueblo	As posted		Return to yard

The train schedule planning mechanism, depicted in Figure 9–11, relies on a centralized database at dispatch centers containing primary versions of train plans, along with distributed mirror-image copies across the network. When a client requests a plan, the primary version is cloned and delivered, with the copy's network location recorded in the database. Changes made by clients trigger updates to the primary version and broadcast messages to synchronize changes across all copies. This ensures consistency via a record-locking mechanism. Whether initiated by onboard clients or dispatch center users, changes propagate similarly across the network using the message-passing mechanism. Leveraging commercial database management systems on dispatch computers addresses backup, recovery, audit trails, and security needs.

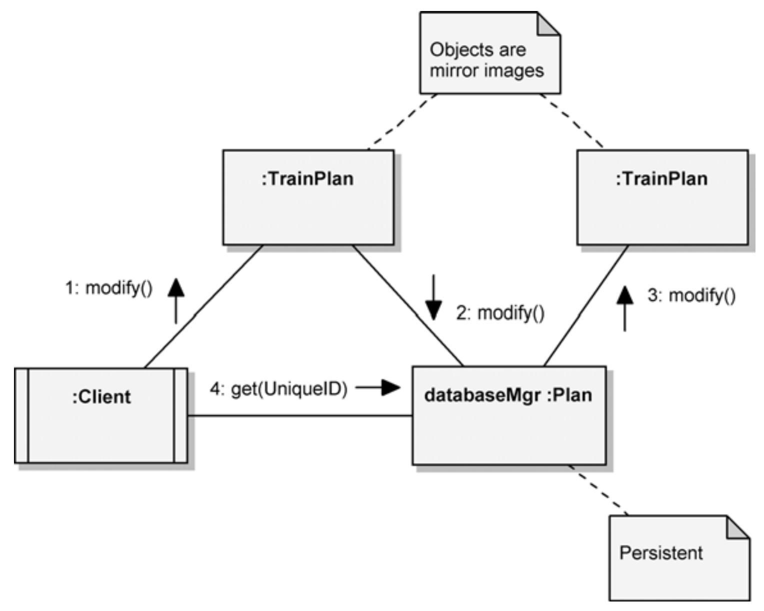

Figure 9–11 Train Schedule Planning

3.3 Displaying Information

Using off-the-shelf technology for our database needs allows us to concentrate on the domain-specific aspects of our problem. Similarly, we can leverage standard graphics facilities for our display requirements. This approach raises the level of abstraction in our system, sparing developers from pixel-level manipulation of visual elements. However, it's crucial to encapsulate design decisions regarding the visual representation of objects. For instance, when displaying track profiles and grades, which may occur at dispatch centers or onboard trains, we can choose between a display manager object that queries object states for visual representation or having displayable objects encapsulate their own display logic. The latter approach is preferred for its simplicity and alignment with the object model.

While encapsulating display logic within individual objects simplifies implementation, it can lead to redundant code and inconsistency across different displayable objects. To address this, it's advisable to conduct a domain analysis of all displayable objects to identify common visual elements. Subsequently, an intermediate set of class utilities can be devised to provide display routines for these shared elements. These utilities can then leverage lower-level graphics packages, promoting code reuse and maintainability.

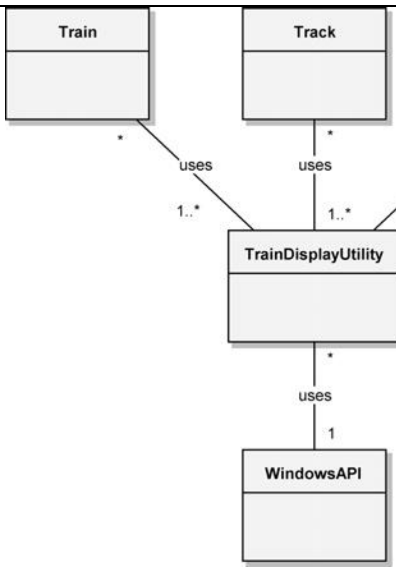

Figure 9–12 Class Utilities for Displaying

Figure 9–12 illustrates a design where the implementation of all displayable objects shares common class utilities, which build upon lower-level Windows interfaces. While this diagram simplifies the architecture, in reality, multiple peer class utilities may be needed for the Windows API and train display utilities. The primary advantage of this approach is its resilience to lower-level changes resulting from hardware or software adjustments. For instance, updating display hardware would only necessitate reimplementing routines in the TrainDisplayUtility class, rather than modifying every displayable object's implementation.

3.4 Sensor Data Acquisition

The Train Traffic Management System relies on various sensors, monitoring parameters like oil temperature, fuel quantity, and switch positions. Despite different sensor outputs, their processing shares similarities. Most sensors require periodic sampling, with normal values triggering notifications and extreme values prompting alarms or actions.

By conducting a domain analysis of periodic, nondiscrete sensors, we can establish a common sensor mechanism to avoid redundancy and ensure consistency. This architecture could involve a hierarchy of sensor classes and a frame-based system for data acquisition from various sensors.

3.5 Release Management

In our incremental development approach, we employ release management techniques to analyze the system architecture and specify its subsystems. We begin by choosing a few key scenarios, implementing them to create an executable product that simulates their execution. This vertical slice through our architecture allows us to iteratively build and refine the system.

For instance, we might start by implementing the primary scenarios of three key use cases: Route Train, Monitor Train Systems, and Monitor Traffic. These scenarios touch critical architectural interfaces, allowing us to validate our assumptions. Subsequently, we can follow a release sequence:

1. Create or modify a train plan.
2. Request system monitoring and failure prediction analysis.
3. Avoid collisions manually or with automated assistance, and track train traffic using TTMS resources or Navstar GPS.

With a 12- to 18-month development cycle, we aim for stable releases every 3 months, gradually building functionality until all system scenarios are covered.

Success in this strategy hinges on effective risk management, where we prioritize addressing the highest development risks with each release. For control system applications like this, it involves early testing of positive control to identify any system control gaps promptly. Our release sequence emphasizes selecting scenarios across the system's functional elements, mitigating unforeseen gaps in our analysis.

3.6 System Architecture

The software design for large systems often starts before the hardware is finalized, as software design typically takes longer. To enable independent progress, software should minimize dependencies on specific hardware and be built with replaceable subsystems. In systems like the Train Traffic Management System, taking advantage of evolving hardware tech during software development is crucial. Early and intelligent software decomposition facilitates parallel work by subcontractors, often driven by non-technical factors like team assignments and subcontractor relationships established early in the system's development.

When selecting a suitable subsystem decomposition, we often start by clustering the highest-level objects around functional lines. These functional lines represent outwardly visible and testable behaviors resulting from logical collections of objects working together. These high-level abstractions and mechanisms serve as good candidates to organize our subsystems. Initially, we may assert the existence of these subsystems and then refine their interfaces over time as the design evolves.

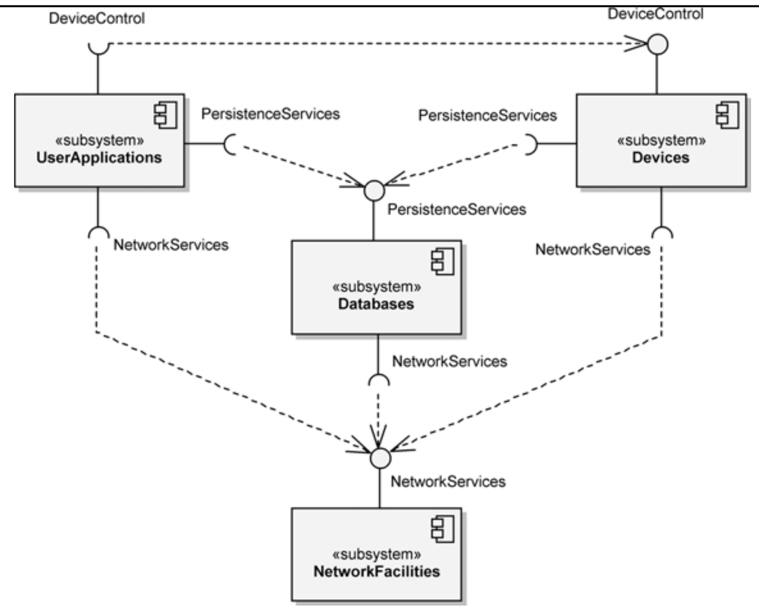

Figure 9–13 The Top-Level Component Diagram for the Train Traffic Management System

The component diagram depicted in Figure 9–13 illustrates our design choices for the top-level system architecture of the Train Traffic Management System. This architecture adopts a layered approach that integrates the functionalities of the four identified subproblems: networking, database management, human-machine interface, and real-time device control.

3.7 Subsystem Specification

In the subsystem specification of the Train Traffic Management System (TTMS), each subsystem behaves like an object, possessing a unique identity, state, and complex behavior. Subsystems serve as repositories for other subsystems and classes, defined by the resources they expose through provided interfaces.

The top-level component diagram in Figure 9–13 serves as a foundational blueprint for the TTMS subsystem architecture. However, this diagram is just the starting point; each top-level subsystem requires further decomposition into nested subsystems across multiple architectural levels. For instance, the NetworkFacilities subsystem decomposes into private RadioCommunication and public Messages subsystems.

Similarly, the Databases subsystem, building upon NetworkFacilities, implements the train plan mechanism and decomposes into TrainPlanDatabase and TrackDatabase, alongside a private DatabaseManager subsystem.

Within the Devices subsystem, software related to wayside devices and onboard locomotive actuators and sensors are grouped into separate subsystems. These subsystems utilize resources from TrainPlanDatabase and Messages, facilitating the implementation of the sensor mechanism.

The UserApplications subsystem is divided into EngineerApplications and DispatcherApplications, catering to different user roles. EngineerApplications handle interactions for train engineers, while DispatcherApplications manage interactions for dispatchers. Both share resources from the Displays subsystem. This decomposition results in four top-level subsystems, each containing smaller ones, housing all key abstractions and mechanisms.

Developers can create their own stable releases before integration into the team's software, with carefully engineered subsystem interfaces guarding them rigorously. For example, the TrainPlanDatabase subsystem relies on Messages, TrainDatabase, and TrackDatabase subsystems, serving multiple clients like WaysideDevices, LocomotiveDevices, EngineerApplications, and DispatcherApplications. While it appears as a monolithic database externally, internally, it operates as a distributed system, leveraging the message-passing mechanism provided by the Messages subsystem.

The TrainPlanDatabase subsystem provides typical database operations such as adding, deleting, modifying, and querying records. These functionalities will be encapsulated within classes that declare these operations. As the design progresses, each subsystem's interface will evolve over time, with most changes expected to be upwardly compatible if the initial characterization of behavior is well-done in an object-oriented manner.

4 TRANSISTION

Adapting a control system like the Train Traffic Management System to evolving business needs is essential for maintaining its effectiveness. While technical risks are predominant, political and social factors also play a role. A resilient object-oriented architecture offers flexibility in responding to regulatory changes and market dynamics.

For instance, integrating payroll processing into the system addresses a critical risk posed by outdated hardware. Although initially disparate, upon closer examination, combining these functions offers significant benefits. Leveraging information from train plans, such as crew assignments, enables more accurate and timely payroll calculations, enhancing operational efficiency.

“1”Adding payroll processing would involve creating a new subsystem within UserApplications, leveraging existing mechanisms. Integrating expert system technology, like a dispatcher's assistant, would require more substantial modifications, potentially affecting networking, data management, and human-machine interface subsystems.

“1”Adding payroll processing functionality to our existing design would involve creating a new subsystem within the UserApplications subsystem. This subsystem would leverage existing mechanisms, as it would have visibility to important components required for its operation. In well-structured object-oriented systems, accommodating significant additions in requirements can often be managed by building new applications on top of existing mechanisms.

“1”However, injecting expert system technology, such as a dispatcher's assistant, would have a more substantial impact on the system's architecture. This addition would likely require significant modifications to accommodate the specialized functionality and interactions with existing components. The dispatcher's assistant would need access to various data sources, decision-making algorithms, and communication channels, potentially leading to adjustments in subsystems related to networking, data management, and human-machine interfaces.

Adding an expert system for the dispatcher's assistant would involve inserting a new subsystem between TrainPlanDatabase and DispatcherApplications, utilizing similar knowledge bases. The presentation of advice to users may require new mechanisms like a blackboard architecture. Well-engineered architectures tend to accommodate unforeseen user needs by allowing the reuse of existing mechanisms, preserving design integrity.