System Architecture: Satellite-Based Navigation

a detailed overview of the process of developing a system architecture for a satellite-based navigation system, focusing on principles of object-oriented analysis and design. Here's a breakdown of key points:

1. **Object-Oriented Principles**: The passage emphasizes that the principles of object-oriented analysis and design, as well as UML notation, are applicable not just to software development but also to the development of system architecture. This means that the approach involves understanding system requirements and partitioning the system into segments or subsystems.
2. **Abstraction and Scope**: System architecture concerns are described as abstract, large in scope, and impactful, but not involved in implementation or technology details. The focus is on creating a system with long-term viability, operability, maintainability, and extensibility.
3. **Satellite Navigation System Example**: The example used in the text is the development of a Satellite Navigation System (SNS). The choice of this domain is based on its technical complexity and interest, compared to simpler invented examples. The text mentions existing satellite-based navigation systems like the U.S. GPS, Russian GLONASS, and the European Galileo system.
4. **Inception Phase**: The first steps in developing the system architecture are categorized as systems engineering rather than software engineering. The focus is on defining the problem boundary, determining mission and system use cases, developing functional and nonfunctional requirements, and identifying constraints.
5. **Systems Engineering Approach**: The International Council on Systems Engineering (INCOSE) defines systems engineering as an interdisciplinary approach to enabling successful systems realization. System architecture, within this context, involves arranging elements and subsystems and allocating functions to meet system requirements.

This passage sets the stage for the subsequent development of the system architecture, highlighting the importance of understanding requirements, abstraction, and the interdisciplinary nature of systems engineering.

1. INCEPTION

The inception phase in the development of system architecture, emphasizing its alignment with systems engineering principles.

1. **Systems Engineering Focus**: The passage underscores that the initial steps in developing system architecture are fundamentally systems engineering activities. This implies a broad, interdisciplinary approach to problem-solving, rather than focusing solely on software engineering aspects.
2. **INCOSE Definition**: The International Council on Systems Engineering (INCOSE) is referenced for its definitions of systems engineering and system architecture. Systems engineering is characterized as an interdisciplinary approach aimed at realizing successful systems. System architecture is defined as the organization of elements and subsystems to meet system requirements.
3. **Objectives of Inception Phase**: The primary goal of the inception phase is to determine what needs to be built for the customer. This involves several key activities:
    - Defining the problem boundary: Establishing the scope and context of the system.
    - Determining mission use cases: Identifying the primary tasks or functions the system needs to support.
    - Analysing mission use cases: Breaking down mission use cases to derive system use cases, which represent specific interactions and functionalities of the system.

4. **Requirements Development**: The inception phase involves developing both functional and nonfunctional requirements, as well as identifying any constraints that may impact system design and development.

1.1 Requirements for the Satellite Navigation System

For the Satellite Navigation System, the initial step in building solutions involves utilizing the provided documentation, including the vision statement and associated high-level requirements and constraints.

Vision:

- Provide effective and affordable Satellite Navigation System services for our customers.

Functional requirements:

- Provide SNS services
- Operate the SNS
- Maintain the SNS

Nonfunctional requirements:

- Reliable enough to guarantee good service.
- Accurate enough to meet user needs now and in the future.
- Backup systems in place for critical functions.
- Lots of automation to keep operating costs low.
- Easy to maintain to reduce maintenance expenses.
- Able to be expanded to add more features later on.
- Ensure that space-based elements have a long lifespan.

Constraints:

- Compatibility with international standards
- Maximal use of commercial-off-the-shelf (COTS) hardware and software.

1.2 Defining the Boundaries of the Problem

The requirements and constraints do permit us to take an important first step in the design of the system architecture for the Satellite Navigation System.

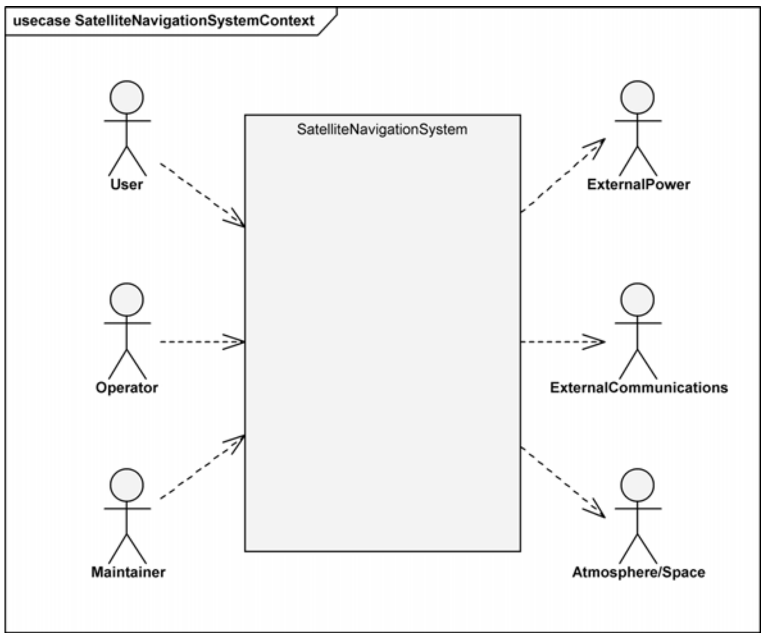

Figure 8–1 The Satellite Navigation System Context Diagram

This context diagram provides us with a clear understanding of the environment within which the SNS must function. Actors, representing the external entities that interact with the system, include people, other systems that provide services, and the actual environment. Dependency arrows show whether the external entity is dependent on the SNS or the SNS is dependent on it.

Users, operators, and maintainers rely on the SNS for navigation services, operation, and maintenance, respectively. While the SNS can generate its own power as a backup, primary power services come from an external source, represented by the ExternalPower actor. Similarly, ExternalCommunications provides communication services to the SNS, serving as a primary or backup option, distinct from internal system communications. Prefixing these actors with "External" distinguishes them from internal system services.

The Atmosphere/Space actor, though seemingly unusual, serves as the transmission medium for communications between the Satellite Navigation System's ground-based and space-based assets, making it a crucial service provider. Its condition directly impacts the quality of these communications. Additionally, considering the constraint of "Compatibility with international standards." Numerous national and international regulations and treaties govern satellite transmissions; thus, we have important reasons to specify this actor.

A critical point about our context diagram is the actual boundary of the system, that is, what is inside our system and what is not. Placing the Operator and Maintainer actors outside the boundary reflects the viewpoint of a specific stakeholder—the customer—whose focus is on providing navigation information to users. While users perceive operators and maintainers as part of the system, the customer's perspective prioritizes the system's core functionality. However, in scenarios where the system includes operation and maintenance services, the Operator and Maintainer actors would be within the system boundary. This highlights the importance of considering different perspectives when defining system boundaries.

the variability in presenting context diagrams, ranging from elaborate to simple formats. Elaborate diagrams offer detailed information on the flow of information between actors and the system. In mature development environments, this information is often known earlier in the cycle and may be included in the diagram. Regardless of complexity, clarity and consistency are crucial. The preferred approach emphasizes simplicity, conveying the system as a container interacting with external entities, providing and receiving services. This understanding is vital at the outset of development.

High-level nonfunctional requirements like reliability and accuracy, along with design constraints, are provided for the Satellite Navigation System (SNS), documented in a supplementary specification. This document also includes functional requirements applicable to multiple use cases. Establishing a glossary ensures consistent understanding of terms among the development team.

The development process for the Satellite Navigation System must allow the architecture to evolve over time and rely on existing standards during implementation. While a comprehensive analysis or design isn't possible in a single chapter, the focus is on designing the first and second levels of the architecture. Functional requirements serve as containers for mission-level use cases, providing a high-level functional context for the system. Object-oriented analysis and UML notation are used to depict these use cases, ensuring clarity in understanding the system's functionality.

1.3 Determining Mission Use Cases

The vision statement prioritizes providing effective and affordable Satellite Navigation System services, requiring the architect to focus on developing practical mission use cases to avoid analysis paralysis and ensure solvability.

Large projects like this often have a centralized team for system architecture, while development tasks are subcontracted. The system architecture divides into four segments: Ground, Launch, Satellite, and User segments for satellite-based systems.

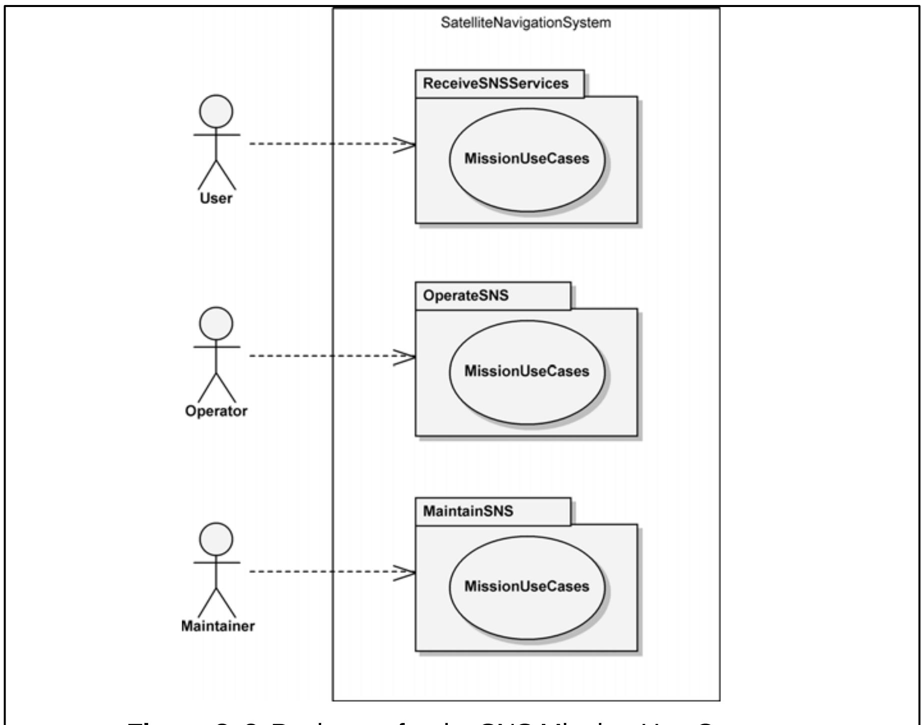

Figure 8–2 Packages for the SNS Mission Use Cases

While some may argue that defining a logical architecture is design rather than analysis, but it's crucial to start constraining the design space. At this stage, the system architecture is principally object-oriented, with complex objects representing major system functions. Refining these objects during analysis resembles the process during design.

Even before conceptualizing a package diagram for the architecture, analysis begins by articulating primary mission use cases with domain experts to understand system behaviour.

This black-box perspective avoids unnecessarily constraining the architecture, focusing on use case functionality before allocating it to individual segments. Activity diagrams are used to illustrate expected system behaviour, emphasizing success scenarios of mission use cases, while alternate scenarios are addressed later.

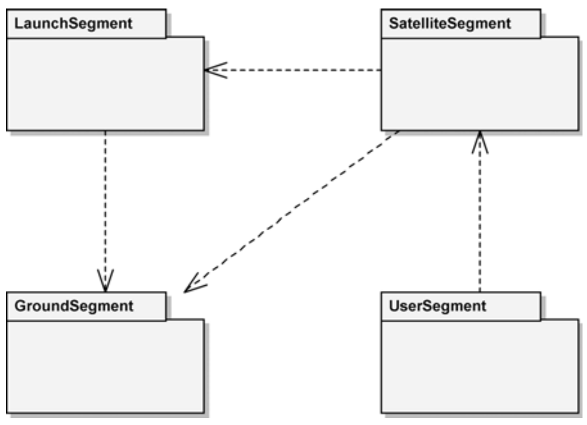

Figure 8–3 The SNS Logical Architecture

In this context, success scenarios represent the main objectives of system interactions, as exemplified by the ATM's "Withdraw Cash" use case.

The Withdraw Cash use case, like all others, comprises various scenarios, with the primary scenario representing successful execution. Alternate or secondary scenarios branch off the primary scenario, addressing situations like exceeding withdrawal limits. Real-time systems, such as the Satellite Navigation

System, often rely heavily on secondary scenarios, which are critical but receive less attention. Analyzing these scenarios is crucial for system development efforts to ensure complete and safe operation.

We define four mission use cases for the Operate SNS package:

- Initialize Operations
- Provide Normal Operations
- Provide Special Operations
- Terminate Operations

These are based on our analysis of the overall SNS operation and rely on domain expertise, past experience, and possibly simulations or prototypes. Activity diagram modeling aids in developing these mission use cases, with Figure 8–4 illustrating the results for OperateSNS. Our focus now shifts to analyzing the Initialize Operations mission use case to identify necessary system activities for enabling SNS initialization by the operator.

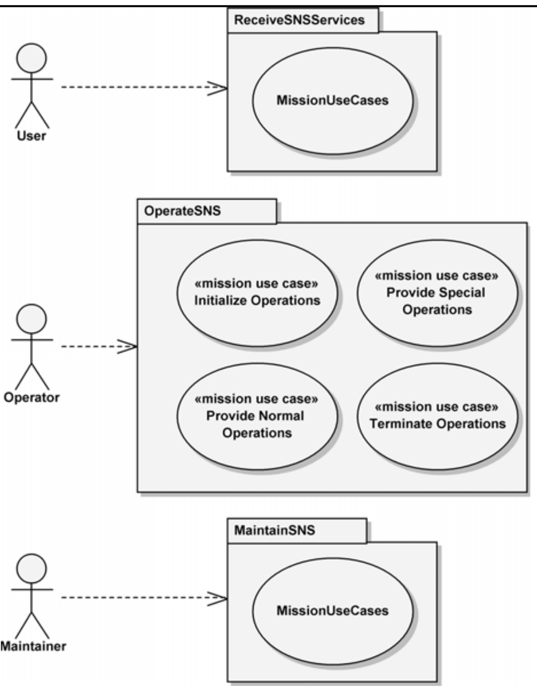
ReceiveSNSServices
User
MissionUseCases
OperateSNS
«mission use case» Initialize Operations
«mission use case» Provide Special Operations
Operator
«mission use case» Provide Normal Operations
«mission use case» Terminate Operations
MaintainSNS
Maintainer
MissionUseCases

Figure 8–4 Refining the `OperateSNS` Mission Use Case Package

1.4 Determining System Use Cases

We construct an activity diagram for the Initialize Operations mission use case to identify system use cases without considering SNS segments. This approach avoids constraining our analysis by predefining architectural solutions and treats the SNS as a black box. Our focus is on understanding the control flow between the operator and the SNS, emphasizing high-level execution behaviour.

The activity diagram focuses on SNS activities rather than messaging, simplifying its representation. To define all SNS activities, we would analyse each mission use case, considering the continuous operations required for system functionality. For instance, visualizing the multitude of activities needed for 24/7 system operation is daunting. However, in this instance, we concentrate on the Initialize Operations mission use case, as depicted in Figure 8–5.

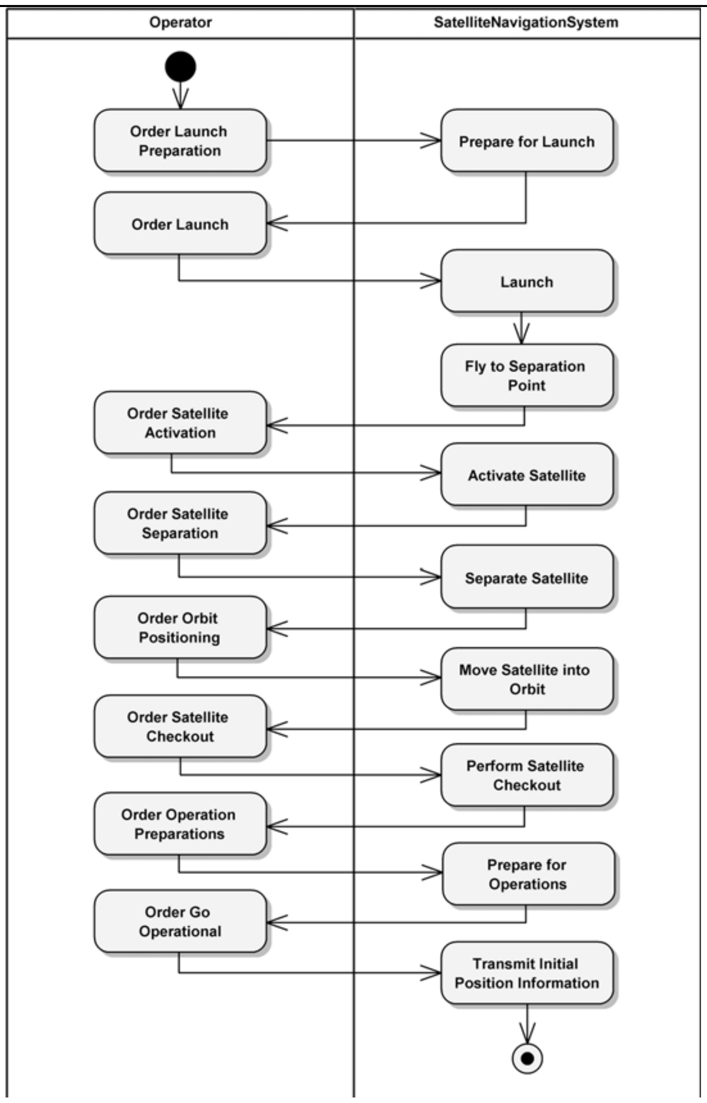

Figure 8–5 The Black-Box Activity Diagram for `Initialize Operations`

The activity diagram informs the creation of system use cases through experienced systems engineering judgment. Actions like Prepare for Launch and Launch are merged into one use case, Launch Satellite, while other actions, deemed to embody significant system functionality, become individual use cases. This process yields the system use cases for the Initialize Operations mission, detailed in Table.

Figure 8–6 illustrates an updated use case diagram featuring system use cases from Table 8–1. The InitializeOperations package encompasses system use cases derived from the Initialize Operations mission use case, while other SNS operation-related mission use cases are labeled «mission use case». This modeling approach is deemed effective, though each team should document their preferred techniques.

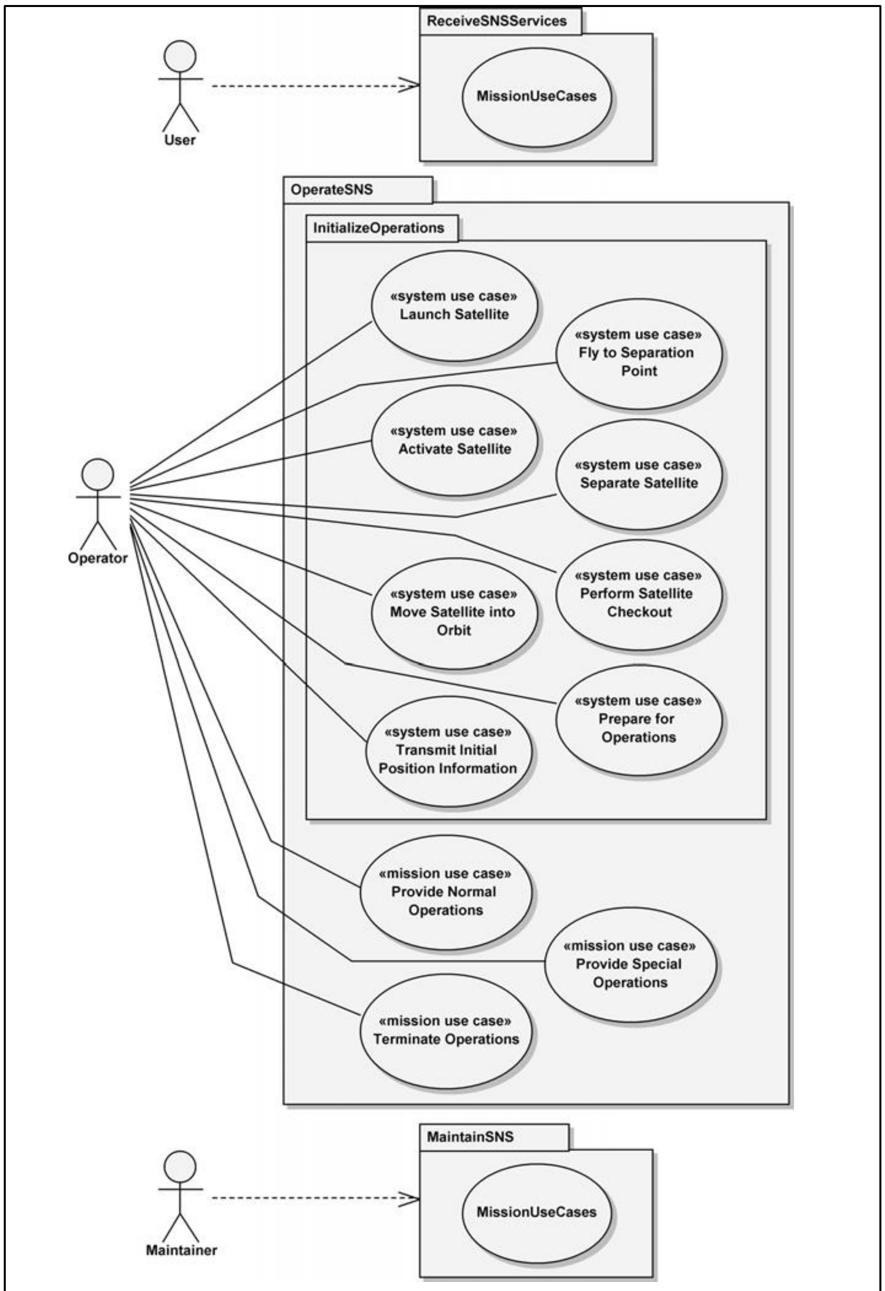
ReceiveSNSServices
User
MissionUseCases
OperateSNS
InitializeOperations
«system use case» Launch Satellite
«system use case» Fly to Separation Point
«system use case» Activate Satellite
«system use case» Separate Satellite
Operator
«system use case» Move Satellite into Orbit
«system use case» Perform Satellite Checkout
«system use case» Prepare for Operations
«system use case» Transmit Initial Position Information
«mission use case» Provide Normal Operations
«mission use case» Provide Special Operations
«mission use case» Terminate Operations
MaintainSNS
Maintainer
MissionUseCases

Figure 8–6 System Use Cases for `Initialize Operations`

Table 8–1 System Use Cases for *`Initialize Operations`*

System Use Case	Use Case Description
`Launch Satellite`	Prepare the launcher and its satellite payload for launch, and perform the launch.
`Fly to Separation Point`	Fly the launcher to the point at which the satellite payload will be separated. This involves the use and separation of multiple launcher stages.
`Activate Satellite`	Perform the activation of the satellite in preparation for its deployment from the launcher.
`Separate Satellite`	Deploy the satellite from the launcher.
`Move Satellite into Orbit`	Use the satellite bus propulsion capability to position the satellite into the correct orbital plane.
`Perform Satellite Checkout`	Perform the in-orbit checkout of the satellite's capabilities.
`Prepare for Operations`	Perform the final preparations prior to going operational.
`Transmit Initial Position Information`	Go operational and transmit initial position information to the users of the SNS.

2. ELABORATION

The Elaboration Phase focuses on establishing the system architecture to meet the developed system use cases. It begins with addressing architectural concerns and activities, followed by validating the proposed system architecture and allocating nonfunctional requirements. This macro-level analysis precedes segment decomposition and subsystem specification, ensuring alignment with system use case functionality.

2.1 Developing a Good Architecture

Good architectures are typically object-oriented and exhibit organized complexity. They feature well-defined layers of abstraction with clear interfaces, allowing for easy modification without disrupting client assumptions. Additionally, they prioritize simplicity, achieving common behaviour through shared abstractions and mechanisms. Effective communication of the architecture to stakeholders is crucial for its success, as outlined in the Creating Architectural Descriptions sidebar.

2.2 Defining Architectural Development Activities

Architectural development activities for the Satellite Navigation System involve:

1. Identifying architectural elements to establish problem boundaries and initiate object-oriented decomposition.
2. Defining the behaviour and attributes of the identified elements.
3. Establishing relationships among elements to delineate boundaries and collaborations.
4. Specifying interfaces and refining elements for analysis at the next abstraction level.

These activities focus on defining segments, their responsibilities, collaborations, and interfaces, providing a framework for evolving the architecture and exploring alternative designs, often conducted concurrently rather than sequentially.

2.3 Validating the Proposed System Architecture

During the Elaboration Phase, system architects experiment with alternative decompositions to ensure robust global design decisions, often through modeling or prototyping. Macro-level analysis validates assumptions and decisions, particularly focusing on problematic areas like Initialize Operations. Caution is advised against skipping validation, as early architectural changes are less costly. Redundancy requirements drive strategic architecture decisions, such as hot-swappable hardware for critical components across segments.

In allocating system functionality to segments, we ensure each segment contributes to the overall functionality of the Satellite Navigation System (SNS). This allocation aligns with object-oriented principles, including abstraction, encapsulation, modularity, and hierarchy. Segments are delineated with clear boundaries, compartmentalizing subsystems while maintaining cohesion and loose coupling.

Performing a comprehensive analysis of system functionality allocation to segments, especially for complex systems like the SNS, may require several months. To validate architectural decisions efficiently, a quick proof-of-concept is recommended before diving into detailed analysis. Throughout this process, questions regarding segment responsibilities, knowledge, delegation, and potential issues must be continuously addressed, resembling those in software engineering. Object-oriented principles and UML notation are equally applicable at the system architecture level as they are in software development. Now, transitioning from black-box to white-box analysis, we delve into the internal structure of the SNS, examining segment functionality and collaboration to meet system requirements.

In developing the system architecture, individual activity diagrams would be created for each of the eight SNS system use cases, with actions apportioned across segments to adhere to object-oriented principles. However, in this macro-level analysis, all eight system use cases are analyzed on one activity diagram to provide a broader perspective of functionality. With the four logical SNS segments identified, functionality allocation begins, aiming for loosely coupled segments with functional cohesion, akin to designing good classes.

Analysing the Launch Satellite system use case, we construct the white-box activity diagram in Figure 8–7. For instance, actions from the Launch Satellite use case are allocated to the GroundSegment and LaunchSegment. GroundSegment initiates preparations, while LaunchSegment executes them, reflecting the collaborative effort in the Satellite Navigation System's operation.

Analysing the Launch Satellite system use case, we allocate actions to the GroundSegment and LaunchSegment. The GroundSegment prepares for launch and commands the LaunchSegment to do the same. Upon completion, the GroundSegment initiates the launch. Continuing this process, we develop the complete activity diagram for Initialize Operations, focusing solely on this portion of functionality. Remaining actions leading up to and following this point are contained within other mission use cases but are not our current focus in this macro-level analysis.

In addition to preparing for launch, the Ground Segment engages in various activities during normal operations, such as monitoring system status, managing satellite flight dynamics, handling alarms, and optimizing satellite operations. This analysis provides a glimpse into the functionality of the Ground Segment, but further analysis of other mission use cases is required to fully define the architecture of the Satellite Navigation System. However, for our current quick look, we'll focus on analyzing the Initialize Operations capability.

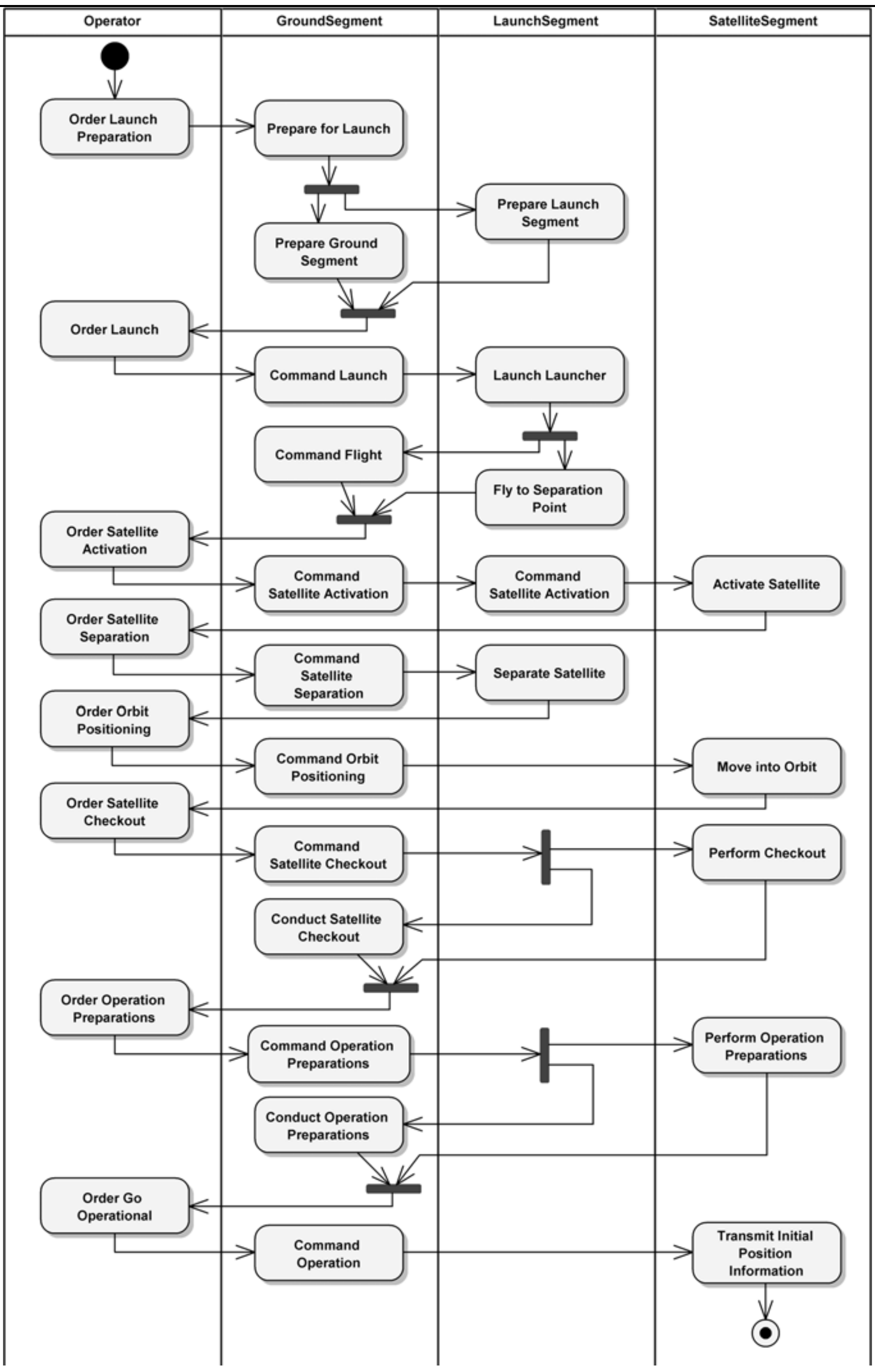
Operator
GroundSegment
LaunchSegment
SatelliteSegment
Order Launch Preparation
Prepare for Launch
Prepare Launch Segment
Prepare Ground Segment
Order Launch
Command Launch
Launch Launcher
Command Flight
Fly to Separation Point
Order Satellite Activation
Command Satellite Activation
Command Satellite Activation
Activate Satellite
Order Satellite Separation
Command Satellite Separation
Separate Satellite
Order Orbit Positioning
Command Orbit Positioning
Move into Orbit
Order Satellite Checkout
Command Satellite Checkout
Perform Checkout
Conduct Satellite Checkout
Order Operation Preparations
Command Operation Preparations
Perform Operation Preparations
Conduct Operation Preparations
Order Go Operational
Command Operation
Transmit Initial Position Information

Figure 8–7 The White-Box Activity Diagram for `Initialize Operations`

In the GroundSegment partition, actions like "Prepare for Launch," "Prepare Ground Segment," and "Command Launch" form the "Control Launch" use case. Following a similar approach, the "Command Flight" action constitutes the "Control Flight" use case. This method is repeated for all partitions to define segment use cases, as summarized in Table 8–2.

Table 8–2 Segment Use Cases for *`Initialize Operations`*

SNS Segment	Segment Use Case	Segment Use Case Action
`GroundSegment`	`Control Launch`	`Prepare for Launch`
		`Prepare Ground Segment`
		`Command Launch`
	`Control Flight`	`Command Flight`
	`Command Satellite Activation`	`Command Satellite Activation`
	`Command Satellite Separation`	`Command Satellite Separation`
	`Control Orbit Positioning`	`Command Orbit Positioning`
	`Command Satellite Checkout`	`Command Satellite Checkout`
		`Conduct Satellite Checkout`
	`Conduct Operation Preparations`	`Command Operation Preparations`
		`Conduct Operation Preparations`

Table 8–2 Segment Use Cases for *`Initialize Operations`* *(Continued)*

NS Segment	Segment Use Case	Segment Use Case Action
aunchSegment	aunch	repare Launch Segment
		Launch Launcher
	Fly to Separation Point	Fly to Separation Point
	Command Satellite Activation	Command Satellite Activation
	Separate Satellite	eparate Satellite
SatelliteSegment	.ctivate Satellite	.ctivate Satellite
	Maneuver to Orbit	Iove into Orbit
	Prepare for Operations	Perform Checkout
		Perform Operation Preparations
	Transmit Initial Position Information	Transmit Initial Position Information

2.4 Allocating Nonfunctional Requirements and Specifying Interfaces

Given the new nonfunctional requirement, we must allocate the 7-day timeline among the use cases identified in Table 8–2, ensuring the entire process from launch preparation to satellite navigation transmission spans fewer than 168 hours. This involves careful planning and coordination to meet the customer's specified timeline and operational needs, potentially impacting the scheduling and sequencing of various activities within the system.

To allocate the nonfunctional requirement of 168 hours among the segment use cases, domain expertise is crucial. We rely on the knowledge of domain experts and development teams, supplemented by techniques like simulation, to determine the impact of different allocation schemes. In this example, 48 hours are allocated to Initialize Operations segment use cases, while the remaining 120 hours are allocated to preparatory activities such as activating the Ground Segment and checking satellite integrity.

Documenting the results of allocating nonfunctional requirements depends on the tools and process used by the team. While many tools offer ways to document results, such as in a requirements database, we prefer a visual representation. Here, we've utilized Table 8–3 to clearly present the allocation of 48 hours across segment use cases. These techniques extend recursively from the system to segment levels, down to their subsystems, ensuring comprehensive allocation of requirements.

Constraints such as "Compatibility with international standards" and "Maximal use of COTS hardware and software" impact system design significantly. For instance, the former necessitates interaction with regulatory agencies for airwave usage, influencing the external interfaces of segments like Ground, Launch, and Satellite. It's vital to address constraints early in the development cycle to ensure compliance and avoid overlooking critical nonfunctional requirements.

Table 8–3 Launch Time Allocations for `Initialize Operations`[a]

SNS Segment	Segment Use Case	Allocated Time (hours:minutes)
`GroundSegment`	`Control Launch`	11:22
	`Control Flight`	0:17
	`Command Satellite Activation`	0:01
	`Command Satellite Separation`	0:01
	`Control Orbit Positioning`	0:05
	`Command Satellite Checkout`	16:30
	`Conduct Operation Preparations`	4:30
	`Command Operation`	0:01
`LaunchSegment`	`Launch`	11:30
	`Fly to Separation Point`	0:17
	`Command Satellite Activation`	0:01
	`Separate Satellite`	0:04
`SatelliteSegment`	`Activate Satellite`	0:03
	`Maneuver to Orbit`	13:45
	`Prepare for Operations`	21:29
	`Transmit Initial Position Information`	0:03[b]

Developing and documenting interface specifications for the Satellite Navigation System involves analyzing its functionality and considering various actors such as User, Operator, and Maintainer. Human/machine interface specialists play a crucial role in this task. Interfaces with actors like ExternalPower and ExternalCommunications can be specified early due to existing standards, while interfaces with the Atmosphere/Space actor are largely governed by regulations and treaties set by national and international agencies.

2.5 Stipulating the System Architecture and Its Deployment

The logical architecture of the Satellite Navigation System (SNS) depicted in Figure 8–3 has been validated through behavioral prototyping efforts. This architecture is represented at the highest level by a component diagram (Figure 8–8), which hierarchically decomposes the system into segments and illustrates their relationships. Interface between segments is shown using both the ball-and-socket notation (e.g., LaunchSupport interface) and dashed dependency lines connecting required and provided interfaces (e.g., PositionInformation interfaces).

In Figure 8–1, three system actors, ExternalPower, ExternalCommunications, and Atmosphere/Space, are not represented in the interfaces shown in Figure 8–8, which depict the logical architecture of the Satellite

Navigation System (SNS). Although these actors provide crucial services to the SNS, they are not the primary focus of our efforts in developing its logical architecture.

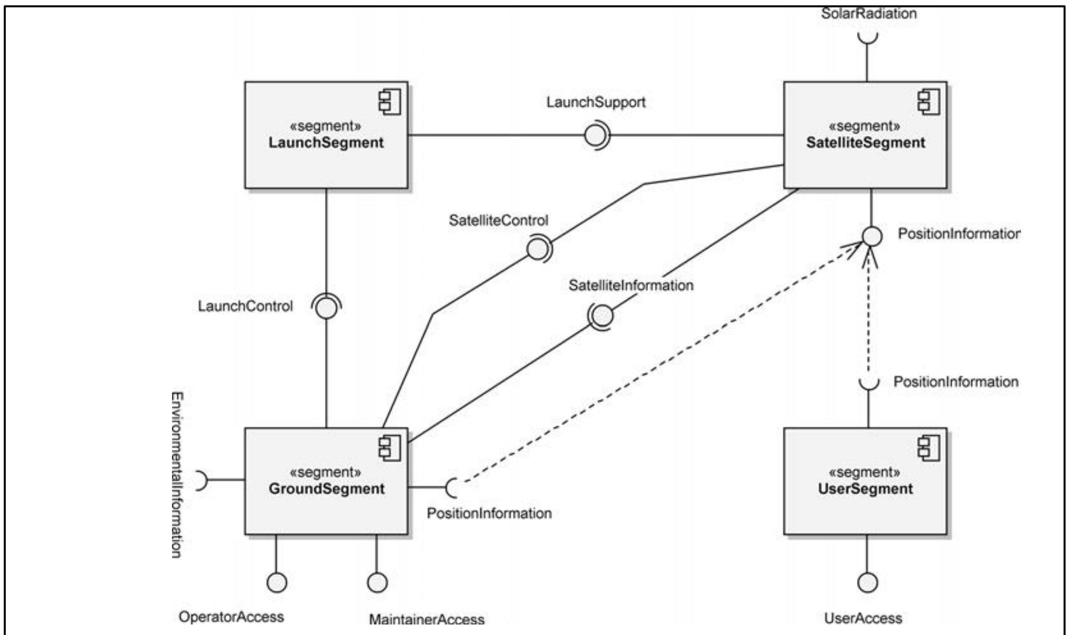

Figure 8–8 The Component Diagram for the Satellite Navigation System

Figure 8–9 illustrates the deployment of components from Figure 8–8 onto the architectural nodes of the system. While this usage deviates from typical UML 2.0 notation, it effectively communicates the information. The interfaces facilitating interaction with the Satellite Navigation System by the Operator, Maintainer, and User actors are encapsulated within its segments, depicted through dependencies.

The decision to enhance functional redundancy by distributing the GroundSegment and LaunchSegment across two geographically dispersed sites is depicted in the architecture. Each segment now has backup sites ready to assume primary roles if needed, safeguarding against complete loss due to events like natural disasters. This is symbolized by the multiplicities of 2 on the communication association between the GroundSite and LaunchSite nodes, ensuring resilience in critical operations.

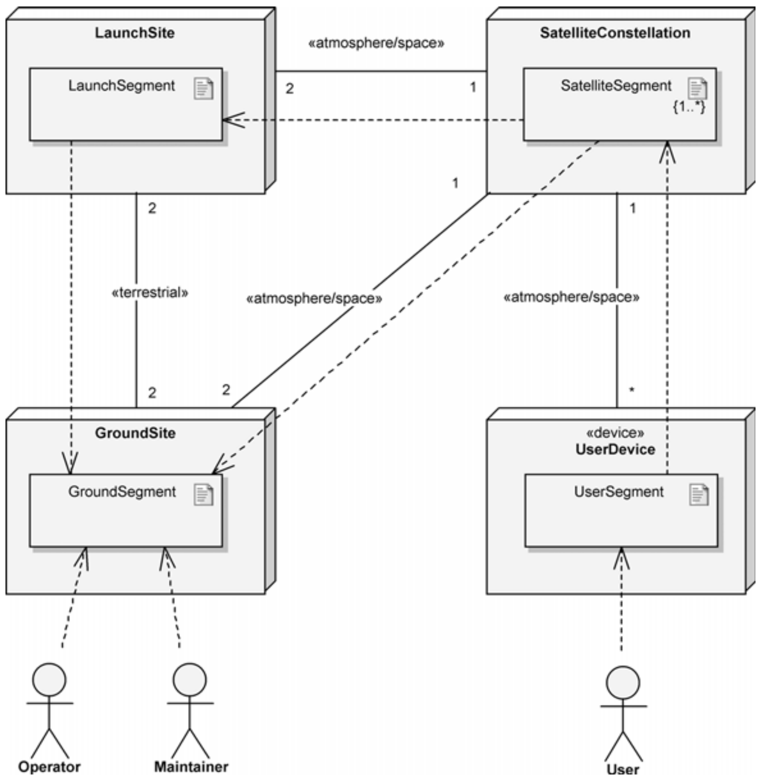

Figure 8–9 The Deployment of SNS Segments

The SatelliteConstellation node represents the satellites comprising the Satellite Navigation System, aiding the SatelliteSegment artifacts by providing necessary support, such as gravity for orbit stability and the atmosphere and outer space for communication. The {1..*} multiplicity on SatelliteSegment indicates the minimum number of satellites in the constellation. The specific coverage area is pending determination by the customer to ascertain the required number of satellites for optimal service provision to SNS users.

2.6 Decomposing the System Architecture

With the validation of our assumptions and decisions regarding the Satellite Navigation System's architecture for the Initialize Operations use case, we're ready to detail its segments and subsystems. Any encountered issues would prompt architecture adjustments. It's crucial to note that our behavioral prototype has fulfilled its purpose and should be discarded, similar to how prototype code isn't the basis for final software.

To refine the SNS architecture into manageable components, we need to delve deeper into each segment, breaking them down into nested subsystems. This involves applying comprehensive analysis techniques, similar to those used in prototyping the architecture for the Initialize Operations functionality. We repeat these steps across all levels of abstraction within the system, allocating nonfunctional requirements to each component accordingly.

The analysis techniques employed for developing the SNS architecture are outlined as follows:

1. Conduct black-box analysis for system use cases to identify their actions.

2. Conduct white-box analysis to allocate these actions across segments.

3. Define segment use cases based on the allocated system actions.

4. Perform black-box analysis for each segment use case to determine its actions.

5. Conduct white-box analysis to allocate segment actions across subsystems.

6. Define subsystem use cases based on the allocated segment actions.

These steps provide a comprehensive framework for analyzing and decomposing the system architecture into manageable components.

It's important to emphasize that our approach to decomposing the system architecture should be applied horizontally across each architectural level rather than vertically from one system use case to subsystems and so on. This ensures a comprehensive view of the system that can be validated at any point. Each step should be completed across the entire architectural level before proceeding to the next step.

We continued our analysis—not shown here—by performing the following activities:

- Performed black-box analysis for the Launch Satellite system use case to determine its actions
- Performed white-box analysis of these system actions to allocate them across segments
- Defined GroundSegment use cases from these allocated system actions
- Performed black-box analysis for the GroundSegment's Control Launch use case to determine its actions
- Performed white-box analysis of the GroundSegment's Control Launch actions to allocate them across its subsystems

Figure 8–10 provides a detailed breakdown of the actions required from each GroundSegment subsystem to fulfill its role in controlling the launch. This analysis allows us to develop the architecture for each segment of the Satellite Navigation System. Subsequent pages will showcase the resulting architectures for each segment.

The architecture of the GroundSegment comprises five subsystems:

1. **ControlCenter**: Provides command and control functionality for the entire Satellite Navigation System, supported by TT&C and SensorStation.
2. **TT&C (tracking, telemetry, and command)**: Monitors and controls the SatelliteSegment.
3. **SensorStation**: Provides position information from the SatelliteSegment and environmental data.
4. **Gateway**: Facilitates communication between the ControlCenter, LaunchSegment, and SatelliteSegment for launch activities and satellite operations control.
5. **UserInterface**: Grants access to GroundSegment functionality for the Operator and Maintainer actors.

This breakdown ensures effective coordination and functionality within the GroundSegment of the Satellite Navigation System.

The logical architecture of the LaunchSegment consists of three subsystems:

1. **LaunchCenter**: Offers command and control functionalities for the LaunchSegment, akin to the ControlCenter of the GroundSegment.
2. **Launcher**: Provides the necessary capabilities to deploy the SatelliteSegment into its initial orbit.
3. **Gateway**: Facilitates communication between the LaunchCenter and the GroundSegment, enabling launch control support from the GroundSegment and providing launch assistance to the Launcher.

This architecture ensures seamless coordination and operation within the LaunchSegment of the Satellite Navigation System.

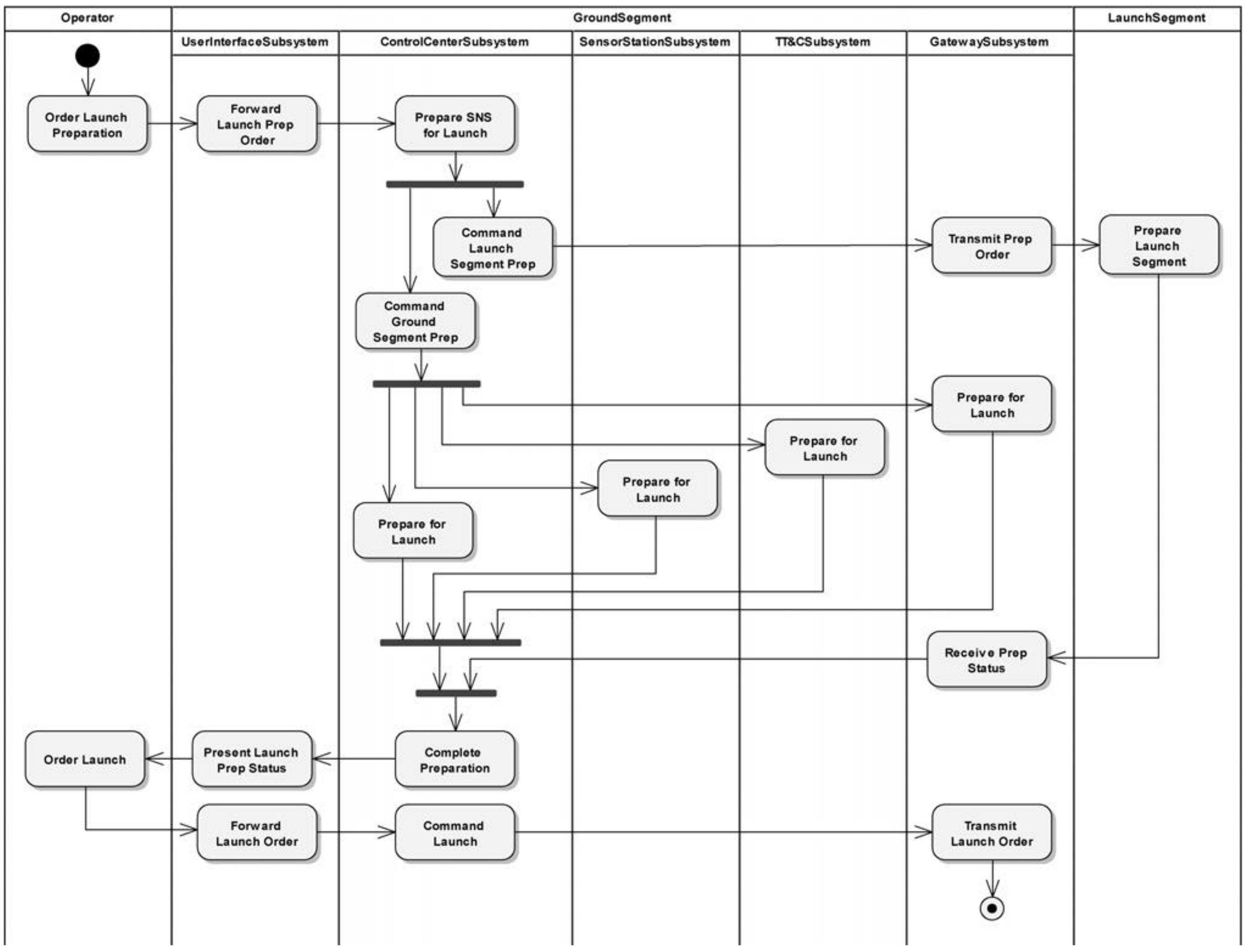

Figure 8–10 The White-Box Activity Diagram for `Control Launch`

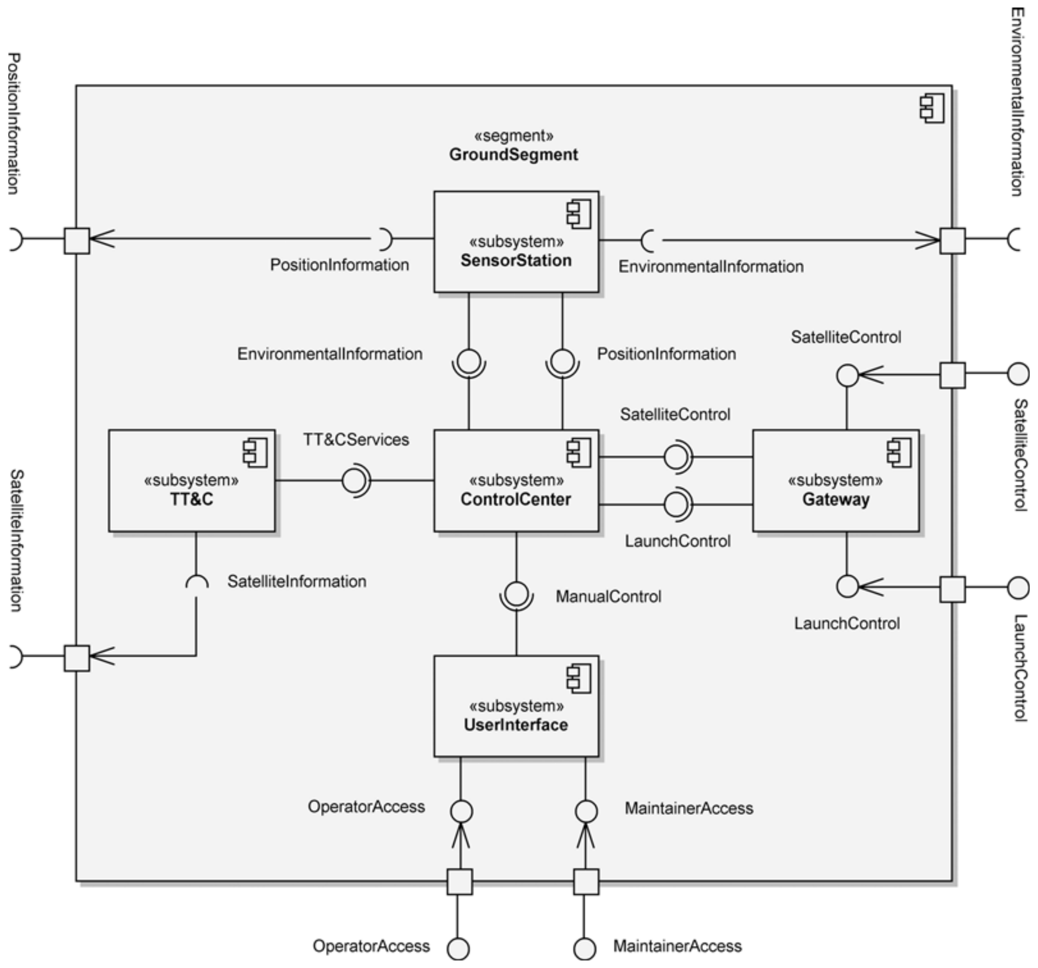

Figure 8–11 The Logical Architecture of the `GroundSegment`

The SatelliteSegment breaks down into two subsystems:

1. **SatelliteBus**: Offers infrastructure support for the NavigationPayload subsystem. It hosts equipment for power, attitude control, propulsion, and other services.
2. **NavigationPayload**: Contains equipment such as a high-accuracy clock and position signal generation, enabling the provision of position information to users of the Satellite Navigation System.

This architecture ensures that the SatelliteSegment efficiently delivers navigation services by separating infrastructure support from navigation payload functionalities.

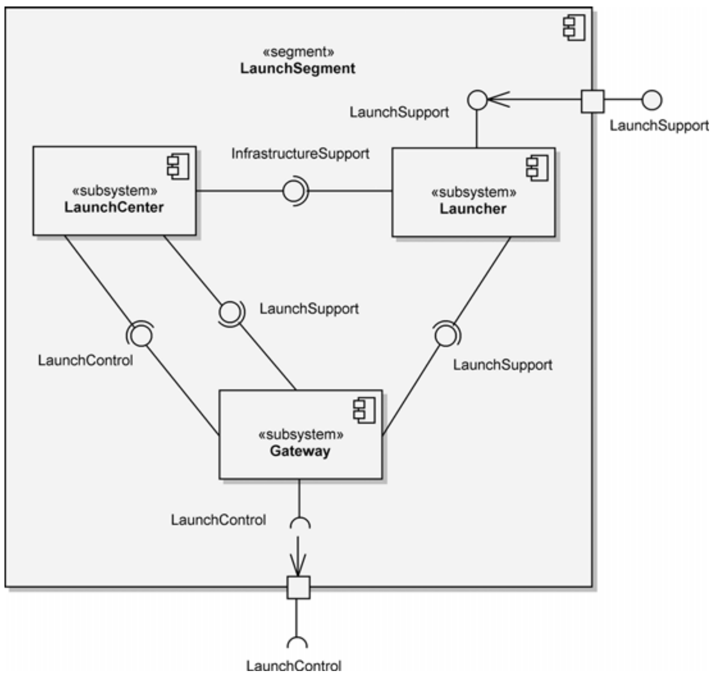

Figure 8–12 The Logical Architecture of the `LaunchSegment`

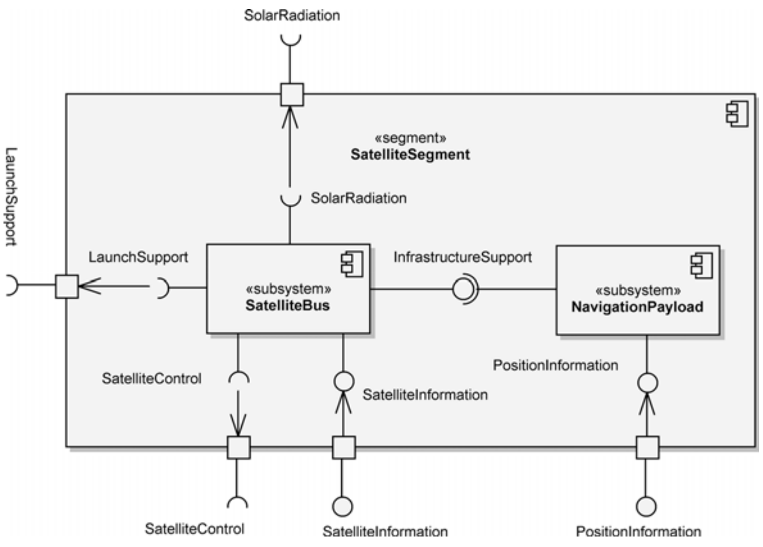

Figure 8–13 The Logical Architecture of the `SatelliteSegment`

The UserSegment is composed of several subsystems:

1. **Receiver**: Obtains position information from the SatelliteSegment and delivers it as position data.
2. **Processor**: Translates received position data into navigation information.
3. **UserInterface**: Facilitates user access and utilization of navigation services through various specialized interfaces like push buttons, touch screens, and audible alerts.

This decomposition ensures efficient processing and delivery of navigation information to users through a well-structured interface system.

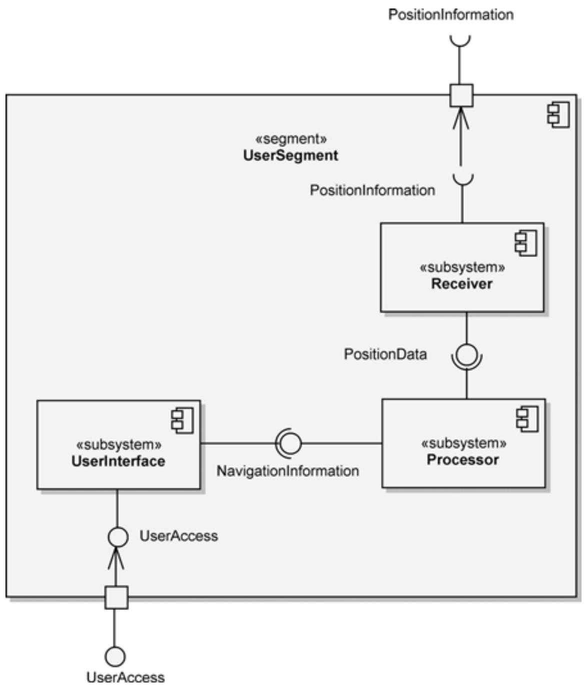

Figure 8–14 The Logical Architecture of the `UserSegment`

Decomposing the system architecture into segments and subsystems creates manageable units for work assignments, configuration management, and version control. Each segment or subsystem is owned by an organization, team, or individual, facilitating detailed design and implementation while managing interfaces effectively. This approach enables the management of complex development programs by breaking down the problem into smaller, more manageable parts. Thus, it demonstrates the applicability of object-oriented analysis and design principles, along with UML 2.0 notation, to both software and system architecture development.

3. CONSTRUCTION

At the end of the Elaboration phase, as we pointed out in Chapter 6, a stable architecture of our system should have been developed. Any modifications to the system architecture that are required as a result of activities in the Construction phase are likely limited to those of lower-level architectural elements, not the segments and subsystems of the Satellite Navigation System that have been our concern. In line with the stated intent of this chapter—to show the approach to developing the SNS system architecture by logically partitioning the required functionality to define the constituent segments and subsystems—we do not show any architectural development activities in this phase.

In the Construction phase, modifications to the system architecture typically focus on lower-level architectural elements rather than segments and subsystems, which should have been stabilized during the Elaboration phase. Therefore, our focus remains on implementing and refining the previously defined segments and subsystems without significant architectural changes.

4. TRANSISTION

In the post-transition phase, we evaluate how well the Satellite Navigation System's design meets the requirements of extensibility and long service life. As new users and implementations emerge, we assess the system's ability to accommodate new functionality and adapt to changes in target hardware while ensuring reliable operation. This evaluation helps us gauge the effectiveness of our design in meeting evolving needs and sustaining the system over its intended lifespan.

4.1 Adding New Functionality

Adding the capability to use position information from other systems like GPS, GLONASS, and Galileo is feasible with minimal impact on the Satellite Navigation System. Since the change is isolated to the User Segment, existing components can be upgraded, such as the Receiver subsystem and firmware in the Processor subsystem. This illustrates the flexibility of well-structured object-oriented systems, where new requirements can often be accommodated by building upon existing mechanisms.

The introduction of the capability to support search and rescue (SAR) missions by receiving distress beacons would primarily impact the Satellite Segment, with some minor impact on the Ground Segment. Ideally, if we had anticipated this requirement, we would have already incorporated the necessary capability into the Satellite Segment, resulting in no design impact but only operational considerations. Otherwise, we would need to add an additional subsystem to the Satellite Segment to support this functionality. The impact on the Ground Segment would likely involve minor software or operational modifications to relay information about distress beacon reception to civil authorities.

Indeed, despite the challenges posed by modifications to space-based assets, adding the SAR capability to the Satellite Segment would still have minimal impact on the overall SNS architecture. Even in the worst-case scenario, integrating this capability into future Satellite Segment elements would entail minimal disruption to the existing architecture or functionality.

4.2 Changing the Target Hardware

The pace of hardware technology advancements often outstrips our capacity to develop software, and early decisions driven by political or historical factors can lead to choices that may later become regrettable. Consequently, the target hardware for large systems tends to become obsolete much sooner than the software it supports.

“1”For example, after several years of operational use, we might decide we need to replace the entire ControlCenter subsystem of the Ground Segment. How might this affect our existing architecture? If we have kept our subsystem interfaces at a high level of abstraction during the evolution of our system, this hardware change would affect our software in only minimal ways. Since we chose to encapsulate all design decisions regarding the specifics of the ControlCenter subsystem, no other subsystem was ever defined to depend on the specific characteristics of a given workstation, for example; the subsystem encapsulates all such hardware secrets. This means that the behavior of workstations is hidden in the ControlCenter subsystem. Thus, this subsystem acts as an abstraction firewall, which shields all other clients from the intricacies of our particular computing technology.

“1”Maintaining high-level abstraction in our subsystem interfaces ensures minimal impact on existing software when replacing hardware components like the ControlCenter subsystem. By encapsulating design decisions and hiding hardware specifics within the subsystem, other subsystems remain independent of workstation characteristics. This approach establishes an abstraction firewall, shielding clients from the intricacies of specific computing technology.

“1” Replacing the entire ControlCenter subsystem of the Ground Segment after several years of operational use would likely have minimal impact on the existing architecture if the subsystem interfaces have been maintained at a high level of abstraction throughout the system's evolution.

By encapsulating all design decisions regarding the specifics of the ControlCenter subsystem and keeping subsystem interfaces abstract, other parts of the system are shielded from the intricacies of the ControlCenter hardware. This means that dependencies on specific characteristics of the hardware, such as workstations, are hidden within the ControlCenter subsystem.

Therefore, when replacing the ControlCenter subsystem, as long as the new hardware adheres to the same abstract interfaces, the software in other subsystems should require only minimal changes, if any. The abstraction firewall provided by the ControlCenter subsystem ensures that the behavior of the workstations remains transparent to other components of the system, thus preserving the system's overall stability and reducing the risk of disruptions during the replacement process.

Maintaining a design where only the Gateway subsystem interfaces with network communications ensures minimal impact from radical changes in telecommunications standards. Higher-level clients remain unaffected, shielded from the complexities of networking by the Gateway subsystem. This design approach provides resilience against shifts in real-world telecommunications technology.

None of the changes we have introduced rends the fabric of our existing architecture. This is indeed the ultimate mark of a well-architected, object-oriented system.

Overall, the fact that your architecture can accommodate changes without disrupting its fundamental structure underscores its robustness and the effectiveness of the design decisions made during its development.