PSCMR COLLEGE OF ENGINEERING AND TECHNOLOGY

Case Study on Satellite Navigation System

BY
MR.KUMMARI ARUN KUMAR
Under the esteemed Guidance of
Dr.SK.Akbar
Professor & HoD

Table of Content

What is Case Study?

What is Case Study?

In **Object-oriented Analysis and Design (OOAD)**, a case study typically refers to a detailed investigation or examination of a particular **problem or scenario** that serves as an example for applying object-oriented principles and techniques.

what a case study involves in the context of OOAD?

1. Problem Description
2. Identification of Objects
3. Use of UML
4. Analysis and Design
5. Implementation insights
6. Validation and Iteration

Overall, a case study in OOAD provides a **practical, hands-on approach** to learning and applying object-oriented principles **to solve real-world problems.** It helps practitioners understand how to translate abstract concepts into concrete designs and implementations using object-oriented methodologies effectively.

what a case study involves in the context of OOAD?

1. Problem Description

A case study begins with a description of a **real-world problem or scenario** that needs to be analyzed and solved using object-oriented methods. This could be a business problem, a system requirement, or any other situation that can benefit from an object-oriented approach.

what a case study involves in the context of OOAD?

2. Identification of Objects

In OOAD, emphasis is placed on **identifying objects** (entities or concepts from the problem domain) and understanding their relationships and behaviors. A case study helps in identifying these objects relevant to the problem at hand.

what a case study involves in the context of OOAD?

3. Use of UML

Unified Modeling Language (UML) diagrams are often used in OOAD to visualize the system being studied. **A case study involves creating UML diagrams** such as class diagrams (to depict classes and their relationships), sequence diagrams (to show interactions between objects over time), and use case diagrams (to describe user interactions with the system).

what a case study involves in the context of OOAD?

4. Analysis and Design

Through the case study, analysts and designers explore how objects interact, how responsibilities are assigned, and how the system components should be designed to meet the specified requirements. This involves defining classes, their attributes, methods, and designing interfaces between objects.

what a case study involves in the context of OOAD?

5. Implementation insights

While a case study primarily focuses on analysis and design phases, it may also touch upon implementation considerations. This includes decisions on **programming languages, frameworks, and technologies** that will be used to build the system.

what a case study involves in the context of OOAD?

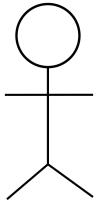
6. Validation and Iteration

Throughout the case study, validation of the design against the problem requirements is crucial. Iterative refinement based on feedback and additional analysis ensures that the designed system meets the expected functionality and performance criteria.

Shapes used in UML

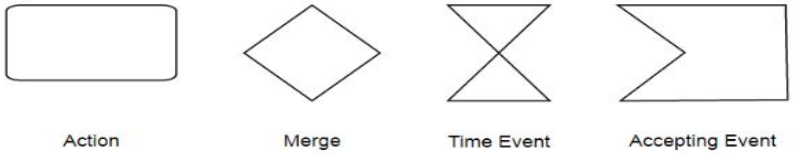
Shapes used in UML

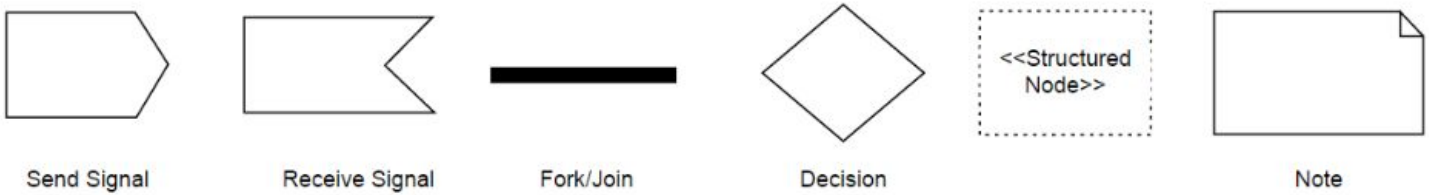
Initial State



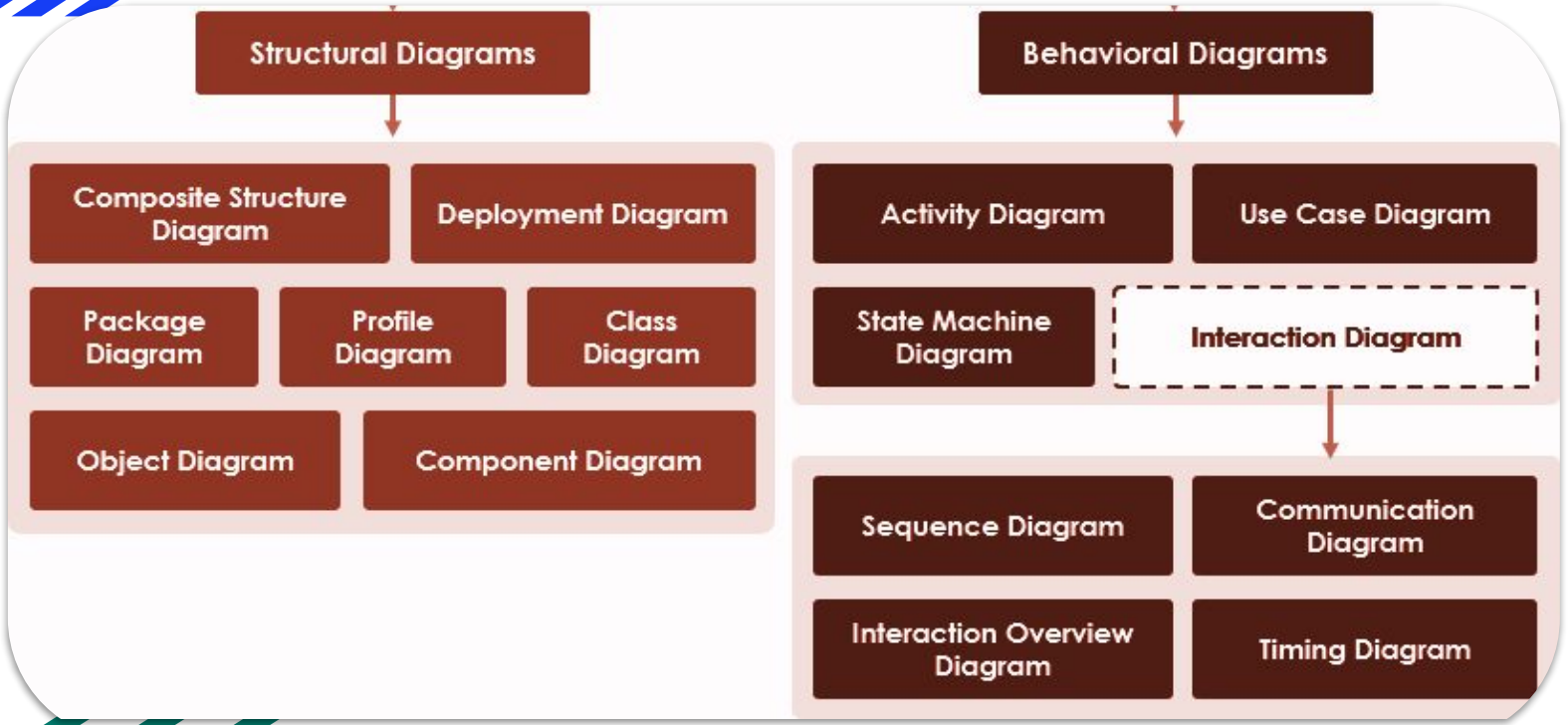
Actor

Final State





Types of UML Diagrams



Structural Diagrams

Class Diagram

A **class** is a collection of similar objects having similar attributes, behavior, relationships and semantics. Graphically class is represented as a **rectangle with three compartments.**

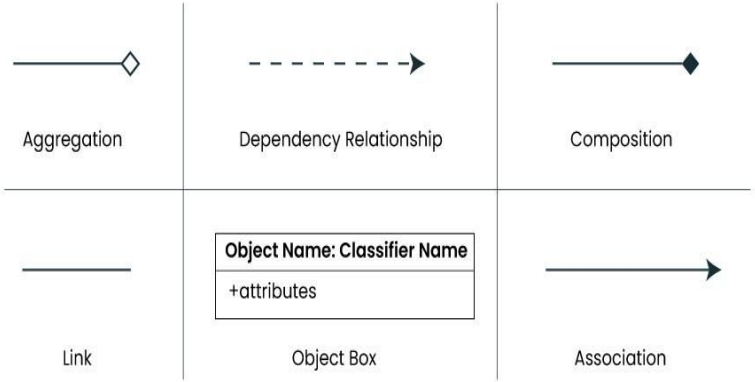
Graphical representation:

Example:

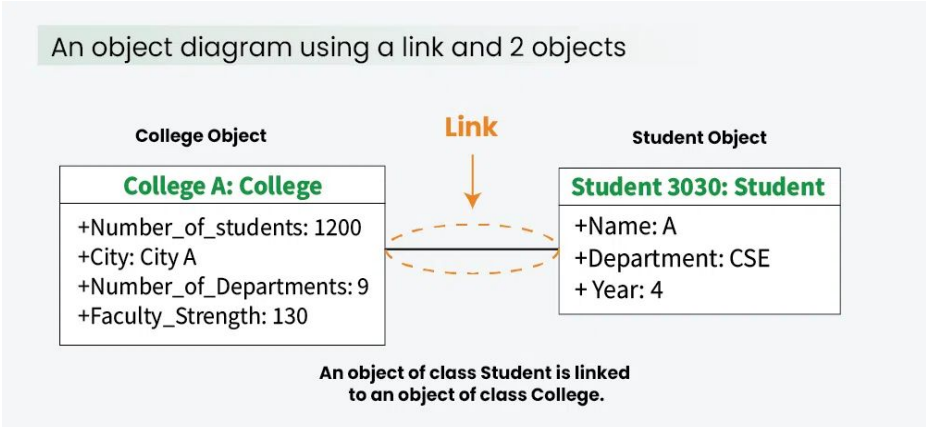
Object Diagram

An **object** refers to a specific instance of a class within a system. A class is a blueprint or template that defines the common attributes and behaviors shared by a group of objects. An object, on the other hand, is a concrete and individual occurrence of that class, possessing unique values for its attributes.

Object Notation:



Example:

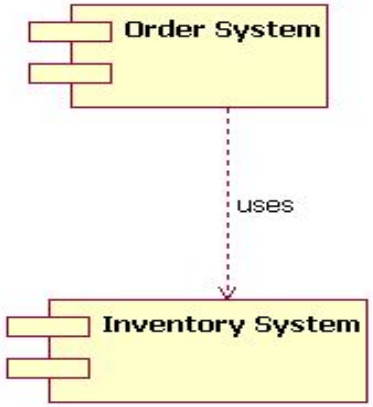


Component Diagram

A **component** is a physical and replaceable part of a system. Graphically component is represented as a **tabbed rectangle**. Examples of components are **executable files, dll files, database tables, files and documents.**

Graphical representation:

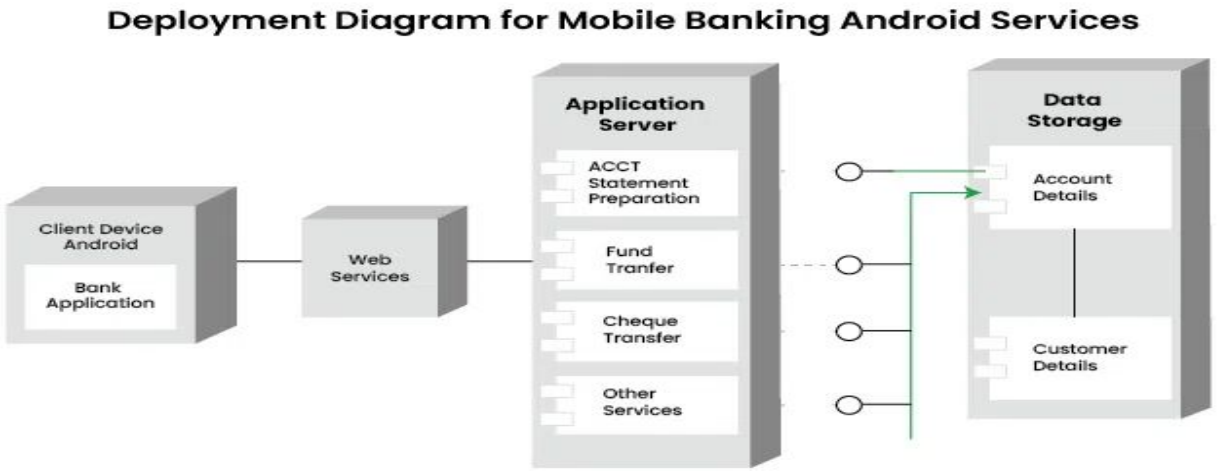
Example:



Deployment Diagram

A Deployment Diagram illustrates how software architecture, designed on a conceptual level, translates into the physical system architecture where the software will run as nodes. It maps out the deployment of software components onto hardware nodes and depicts their relationships through communication paths, enabling a visual representation of the software's execution environment across multiple nodes.

Example:

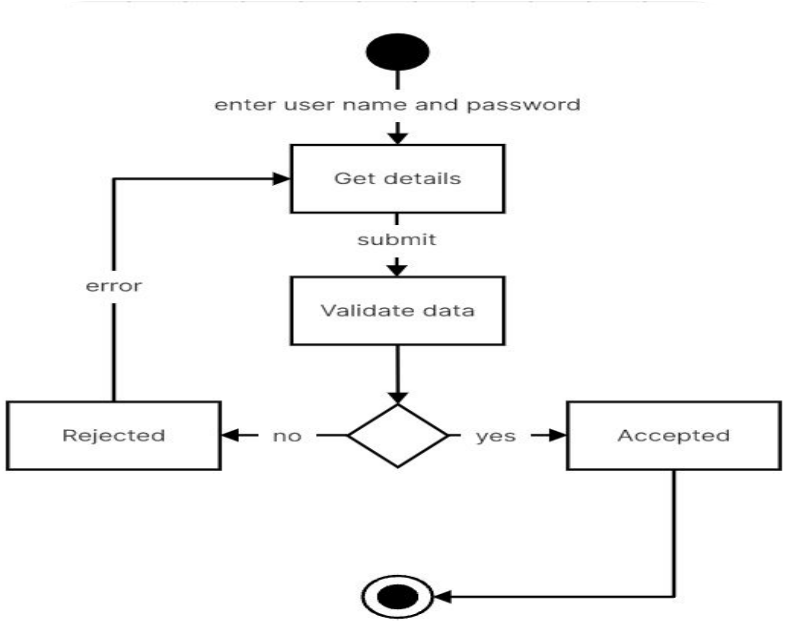


Behavioural Diagrams

Activity Diagram

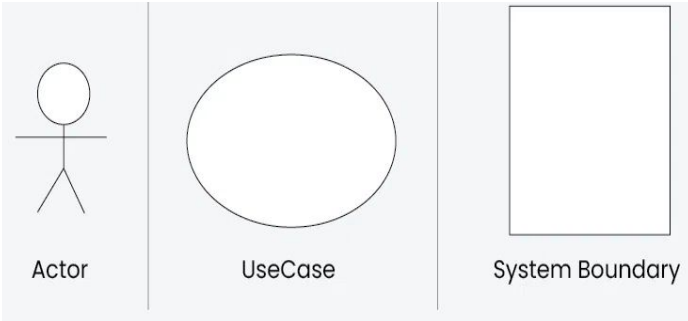
An activity diagram shows the **flow from activity to activity.**

Example:

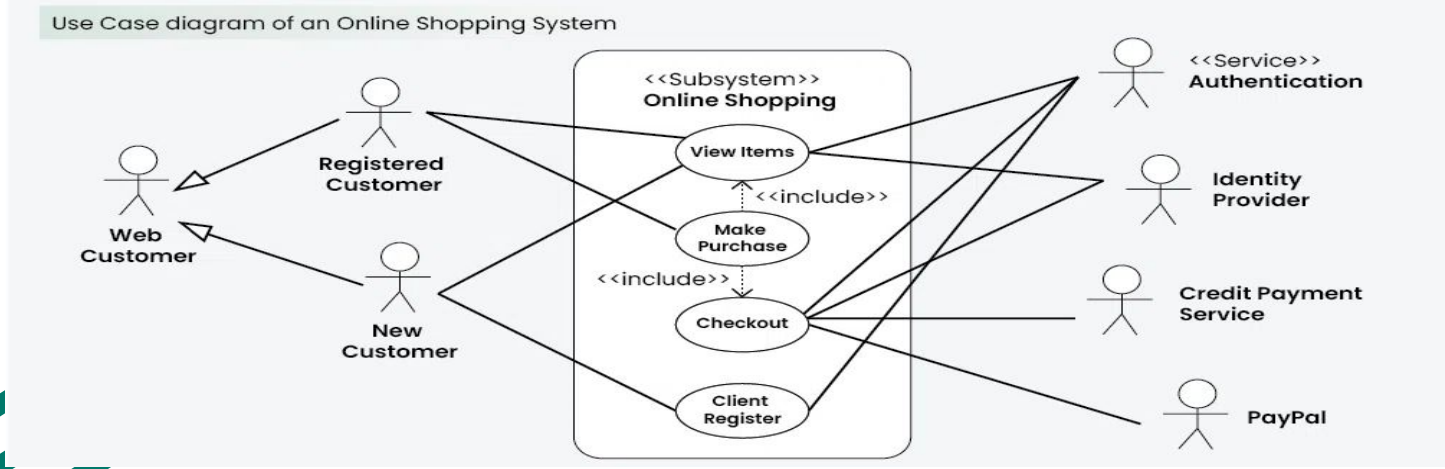


Use Case Diagram

A **Use Case Diagram** is a type of Unified Modeling Language (UML) diagram that represents the interaction between **actors** (users or external systems) and a system under consideration to accomplish specific goals. It provides a high-level view of the system's functionality by illustrating the various ways users can interact with it.

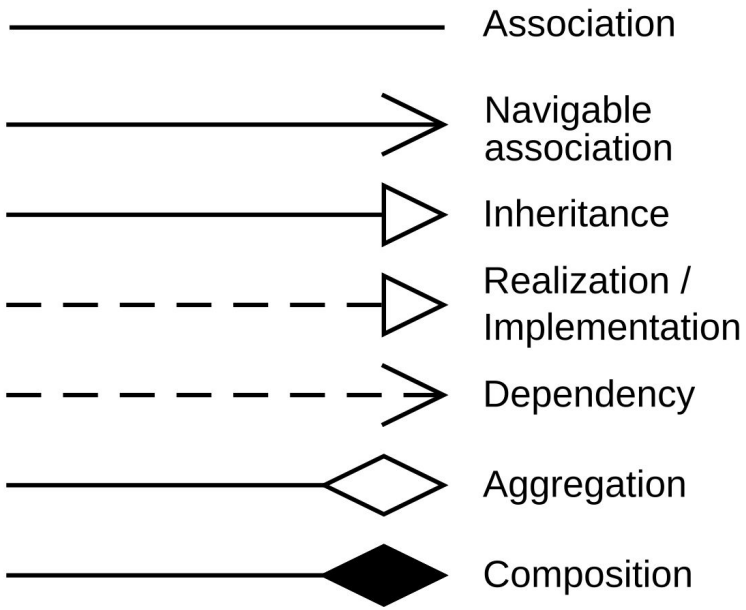


Example:



UML Relationships

UML Relationships



Case Study on
System Architecture:
Satellite-Based Navigation System

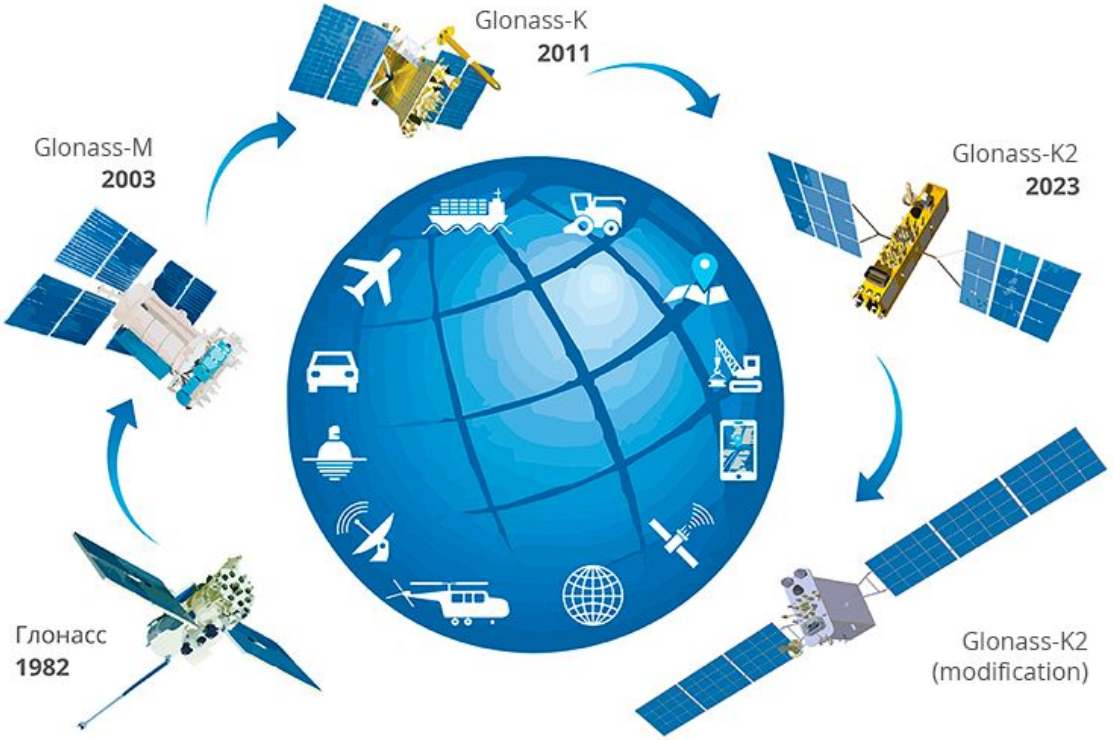
Introduction

Satellite-based navigation systems are essential tools that leverage networks of orbiting satellites to provide accurate **positioning and navigation** information worldwide. These systems are integral to aviation, maritime, and terrestrial transportation, offering precise location data, route planning, and synchronization capabilities. Their architecture includes satellites, ground control stations, user receivers, and data processing systems, designed to ensure reliability and real-time performance.

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.

Russian GLONASS



Inception Phase

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.

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.

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.

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.
- **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.

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

Requirements for the Satellite Navigation System

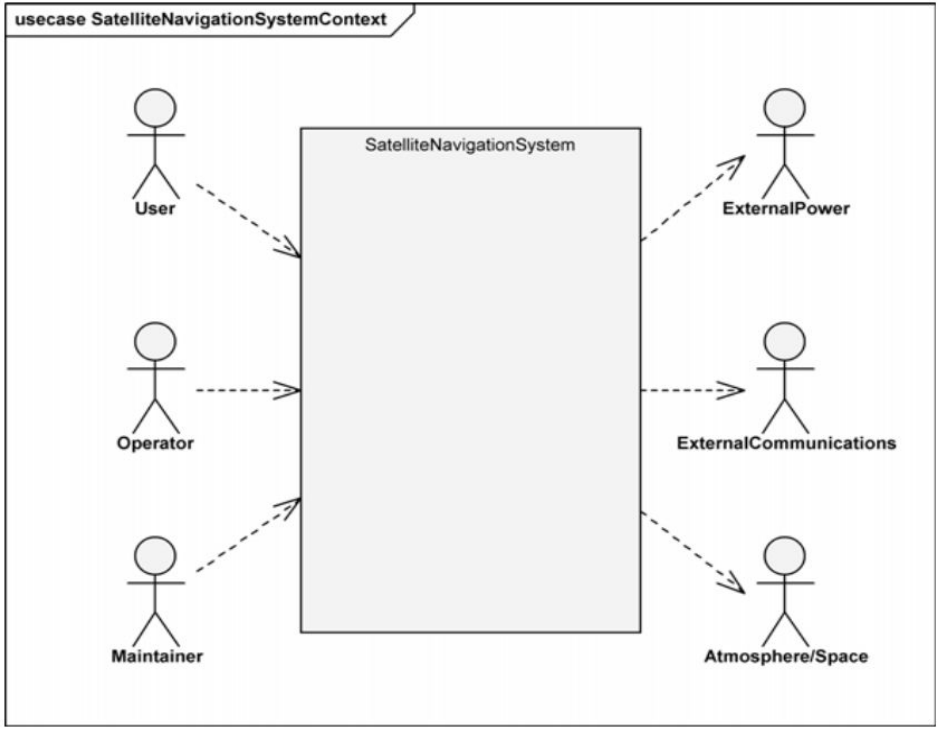
Non Functional requirements:

- 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:

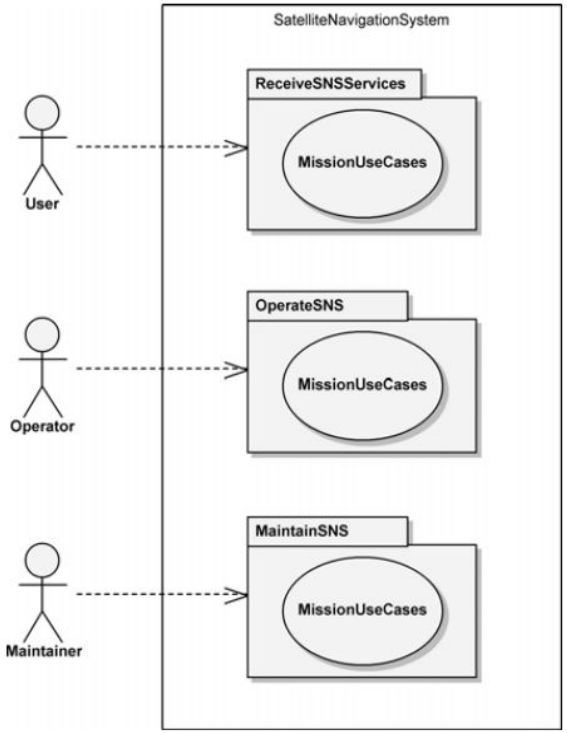
- Compatibility with international standards
- Maximal use of commercial-off-the-shelf (COTS) hardware and software.

Defining the Boundaries of the Problem



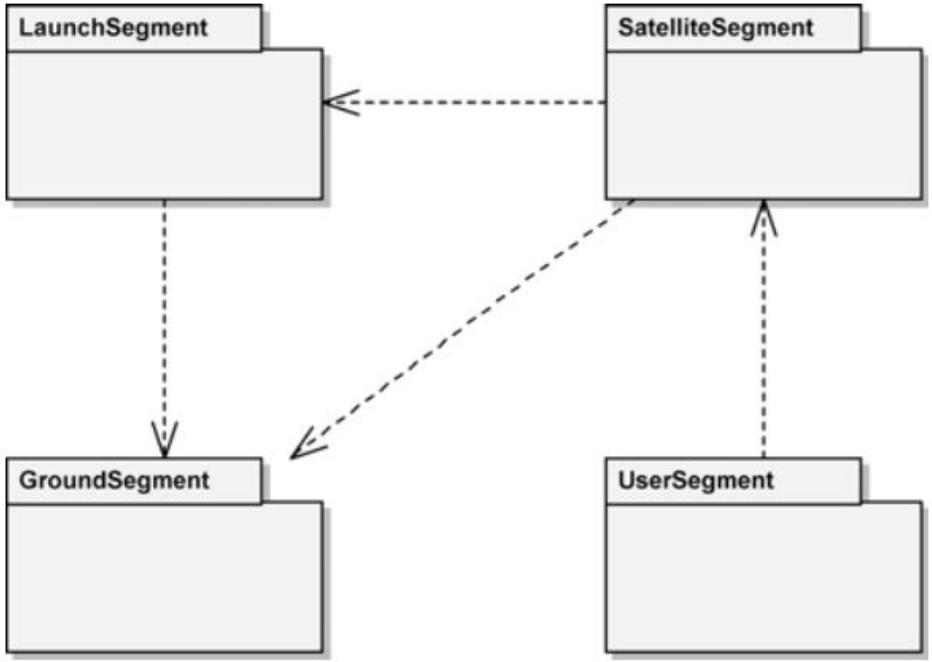
The Satellite Navigation System Context Diagram

Determining Mission Use Cases



Packages for the SNS Mission Use Cases

Determining Mission Use Cases



The SNS Logical Architecture

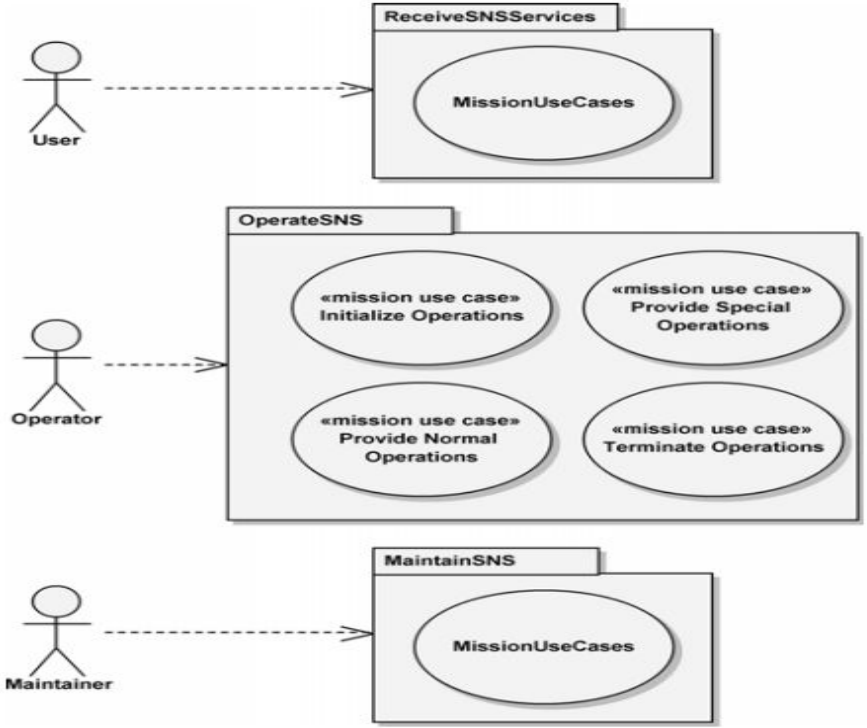
Determining Mission Use Cases

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:

1. Initialize Operations
2. Provide Normal Operations
3. Provide Special Operations
4. Terminate Operations

Determining Mission Use Cases



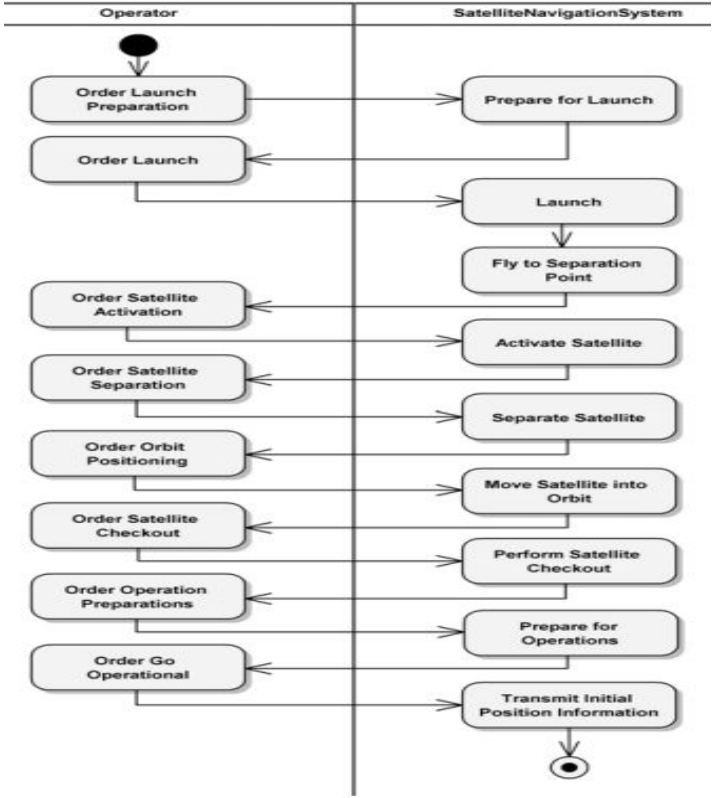
Refining the OperateSNS Mission Use Case Package

Determining Mission 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.

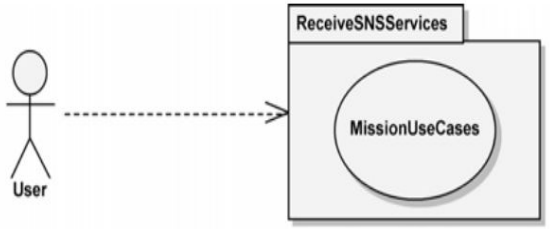
Our focus is on understanding the control flow between the operator and the SNS, emphasizing high-level execution behaviour.

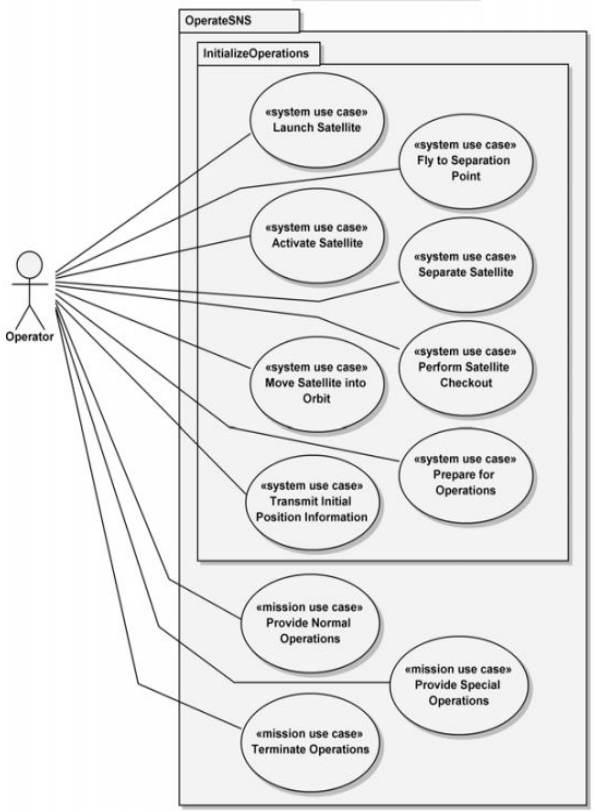
Determining System Use Cases

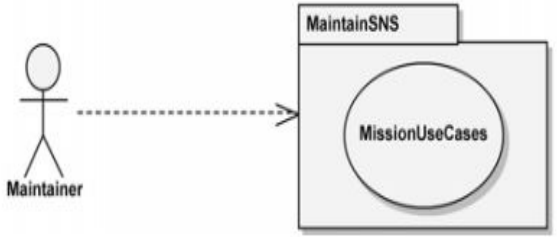


The Black-Box Activity Diagram for Initialize Operations

Determining System Use Cases







System Use Cases for Initialize Operations

Determining System Use Cases

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.

System Use Cases for Initialize Operations

Elaboration Phase

Elaboration Phase

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.

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.

Defining Architectural Development Activities

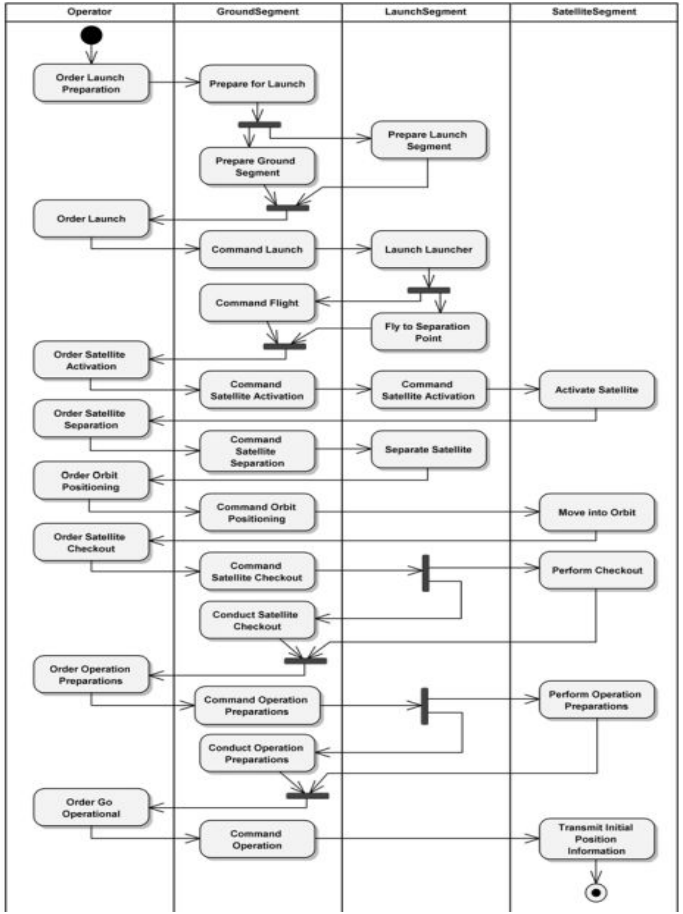
Architectural development activities for the Satellite Navigation System involve:

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.

Validating the Proposed System Architecture

The first step is to review the results of our previous work, assess where we stand, and plan the path forward. With our domain experts, we evaluate the SNS logical architecture and the black-box activity diagram for the Initialize Operations mission use case, from both functional and nonfunctional perspectives. We believe that we've captured the functionality correctly.



The White-Box Activity Diagram 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
	Command Operation	Command Operation

Segment Use Cases for Initialize Operations

SNS Segment	Segment Use Case	Segment Use Case Action
LaunchSegment	Launch	Prepare Launch Segment
		Launch Launcher
	Fly to Separation Point	Fly to Separation Point
	Command Satellite Activation	Command Satellite Activation
	Separate Satellite	Separate Satellite
SatelliteSegment	Activate Satellite	Activate Satellite
	Maneuver to Orbit	Move into Orbit
	Prepare for Operations	Perform Checkout
		Perform Operation Preparations
	Transmit Initial Position Information	Transmit Initial Position Information

Segment Use Cases for Initialize Operations (Continued)

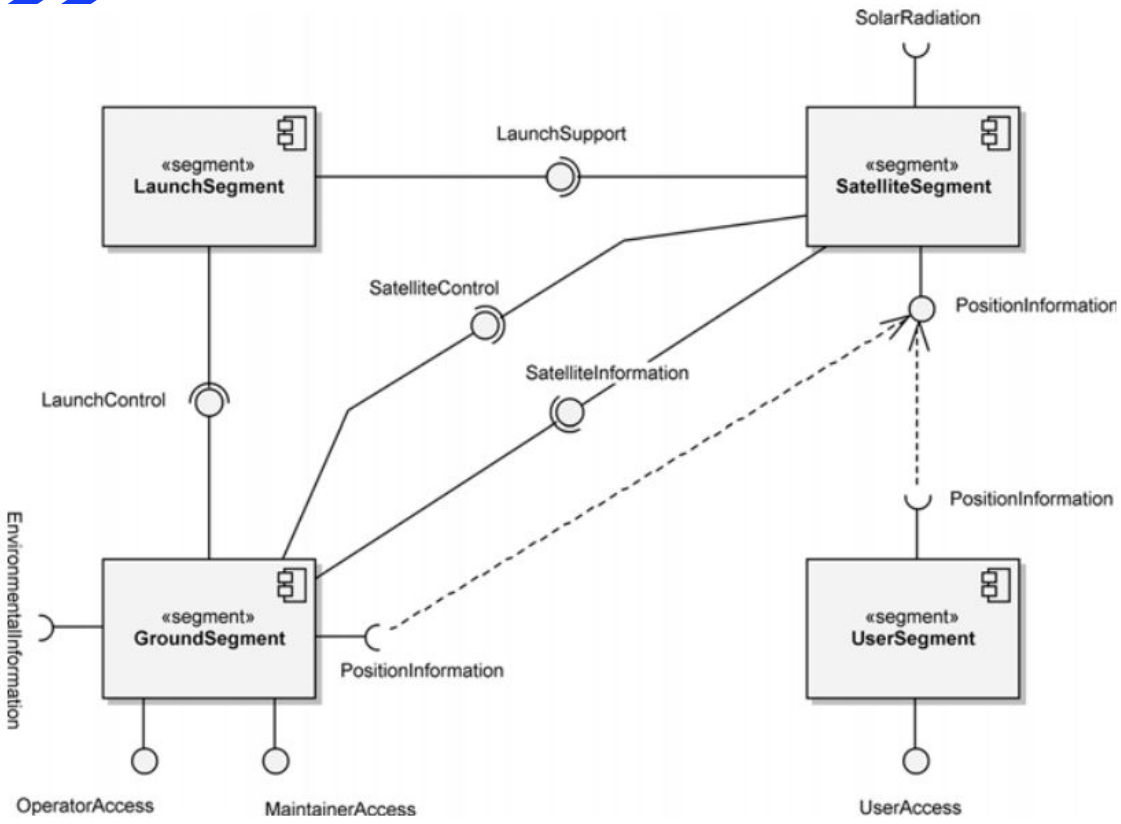
Allocating Nonfunctional Requirements and Specifying Interfaces

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 External Power and External Communications 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.

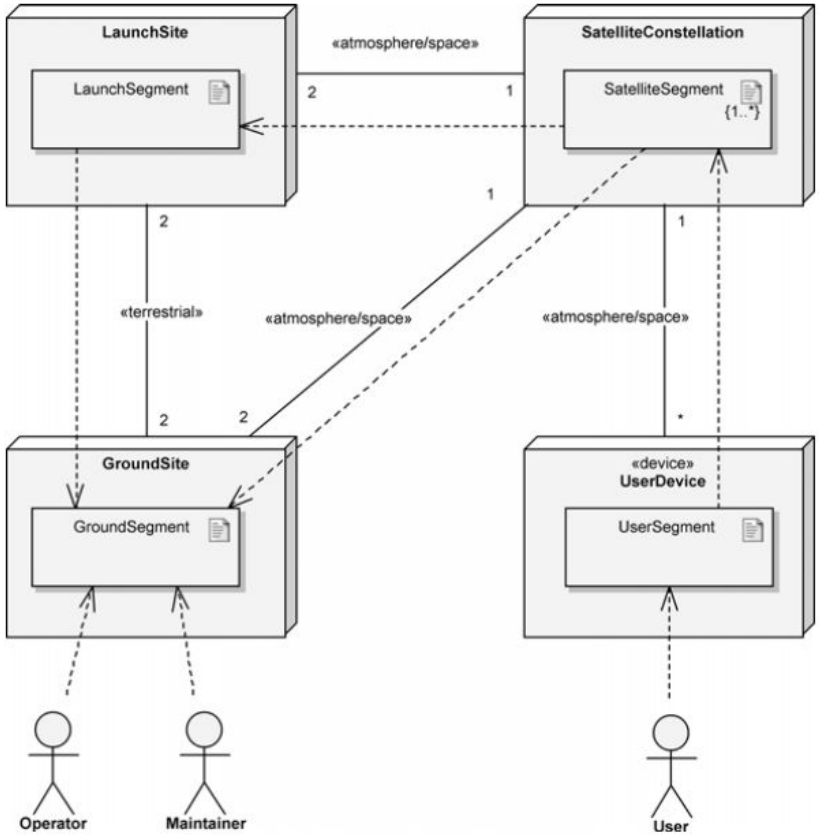
SNS Segment	Segment Use Case	Allocated Time (hours:minutes)
GroundSegment	Control Launch	11:22
	Control Flight	0:17
	Command Satellite Activation	0:01
	Command Satellite Separation	0:01
	Control Orbit Positioning	0:05
	Command Satellite Checkout	16:30
	Conduct Operation Preparations	4:30
	Command Operation	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]

Launch Time Allocations for Initialize Operationsa

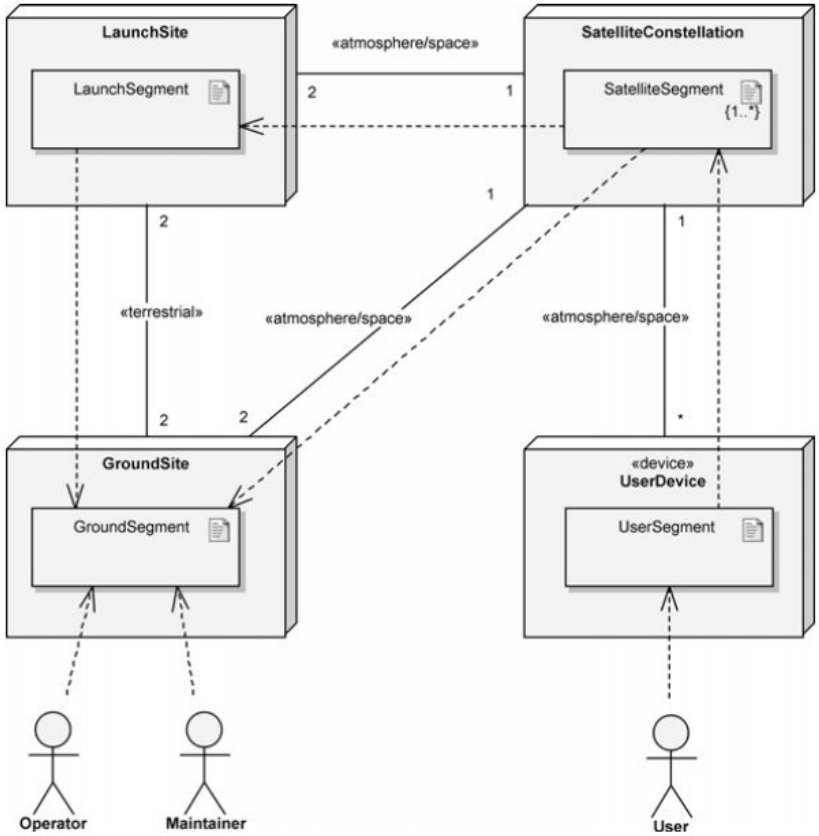
Stipulating the System Architecture and Its Deployment



The Component Diagram for the Satellite Navigation System



The Deployment of SNS Segments



The Deployment of SNS Segments

Decomposing the System Architecture

We must zoom inside each of the segments and further decompose them into their nested subsystems. This is accomplished by applying the same analysis techniques **but applied more completely** that we used to prototype the Satellite Navigations System's architecture of segments for the **Initialize Operations functionality**. These techniques are repeated through all the **levels of abstraction in the Satellite Navigation System** from the system to the **segments, to their subsystems,** and so forth to determine the use cases for each element at every level in the system's architecture. As we do this, the nonfunctional requirements are apportioned across the use cases, allocated to each element at every level in the system decomposition.

Our analysis techniques are presented here for completeness.

1. Perform **black-box analysis** for each system use case to determine its actions.
2. Perform **white-box analysis** of these system actions to allocate them across segments.

Decomposing the System Architecture

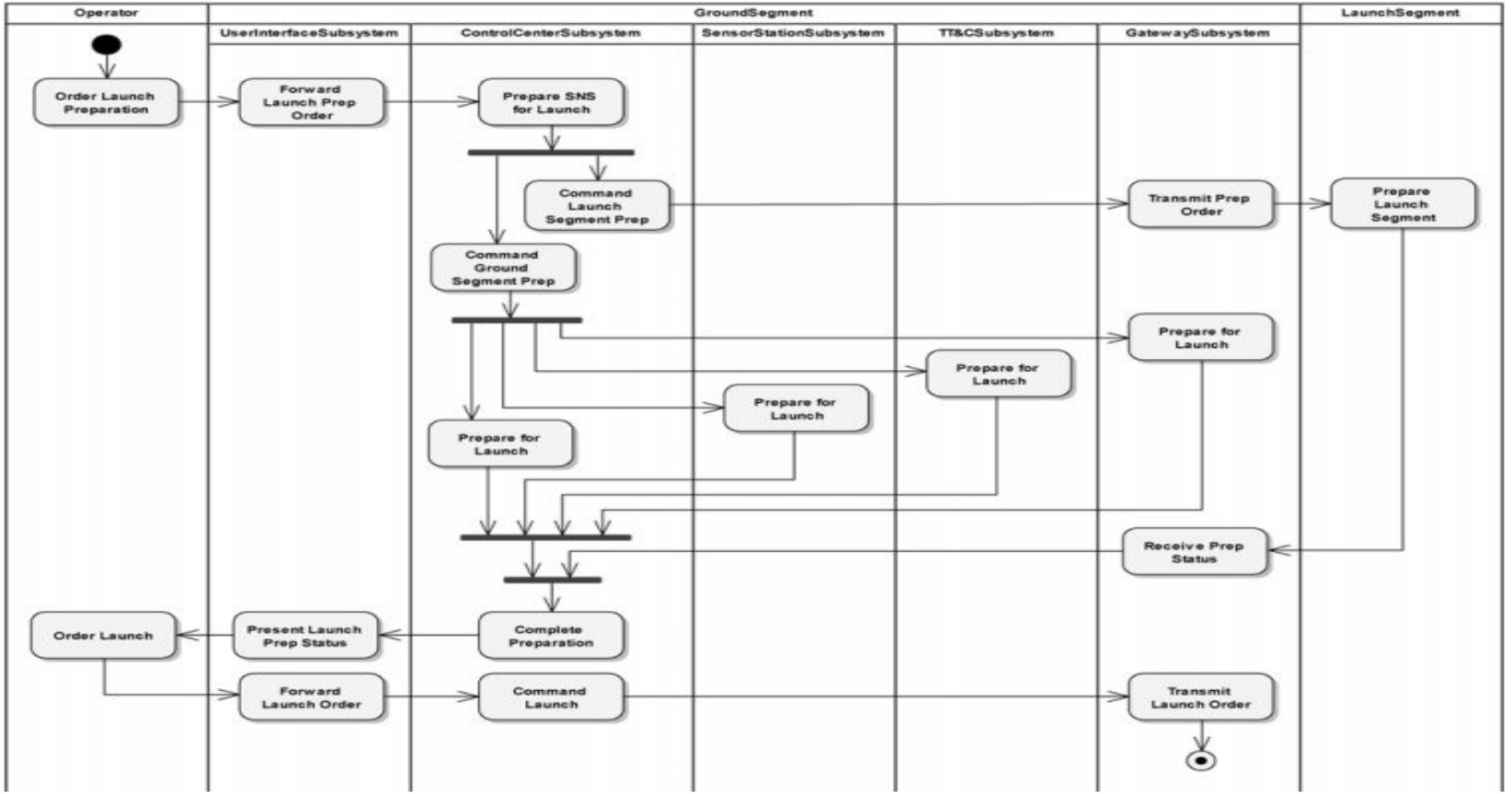
The architecture of the **Ground Segment** comprises five subsystems:

1. Control Center: 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 Satellite Segment.
3. SensorStation: Provides positional information from the Satellite Segment and environmental data.
4. Gateway: Facilitates communication between the Control Center, Launch Segment, and Satellite Segment for launch activities and satellite operations control.
5. User Interface: Grants access to Ground Segment functionality for the Operator and Maintainer actors.

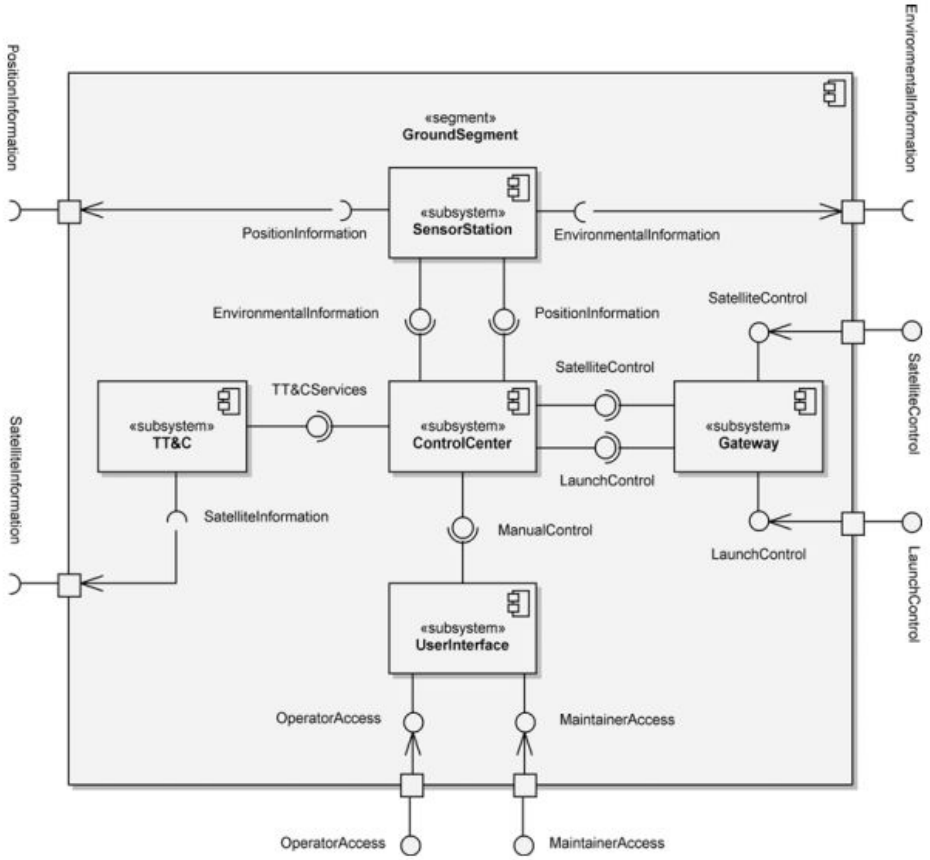
Decomposing the System Architecture

The logical architecture of the **Launch Segment** consists of three subsystems:

1. Launch Center: Offers command and control functionalities for the Launch Segment, to Control Center of the Ground Segment.
2. Launcher: Provides the necessary capabilities to deploy the Satellite Segment into its initial orbit.
3. Gateway: Facilitates communication between the Launch Center and the Ground Segment, enabling launch control support from the Ground Segment and providing launch assistance to the Launcher.



The White-Box Activity Diagram for Control Launch



The Logical Architecture of the Ground Segment

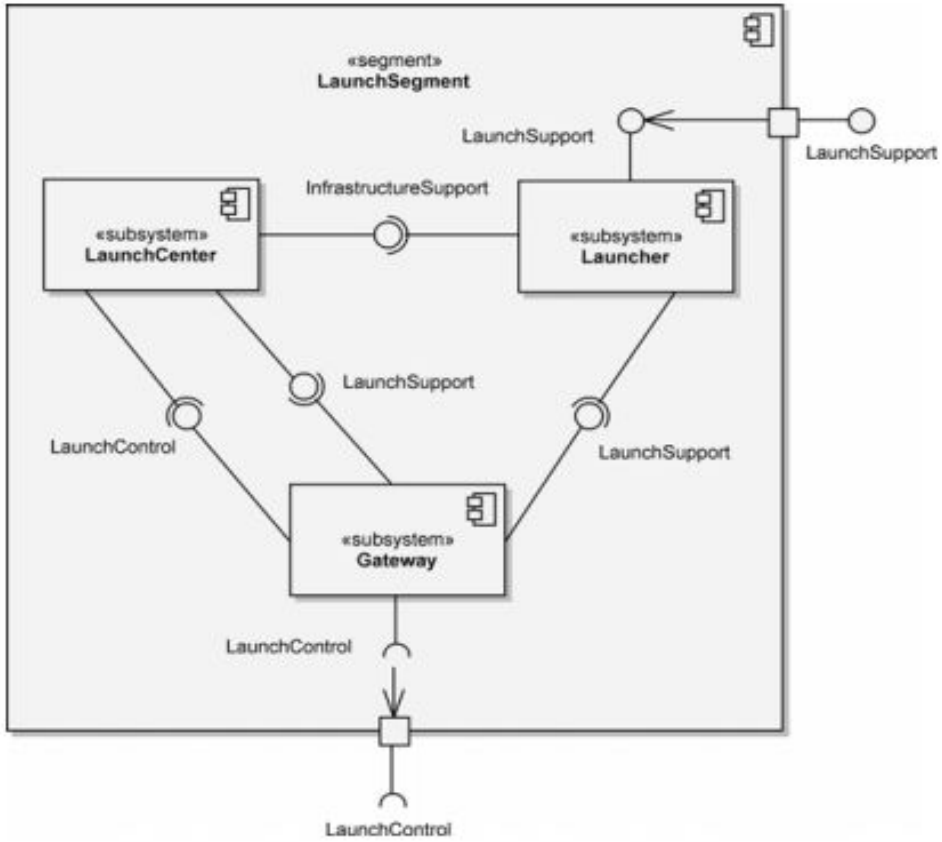
Decomposing the System Architecture

The Satellite Segment breaks down into two subsystems:

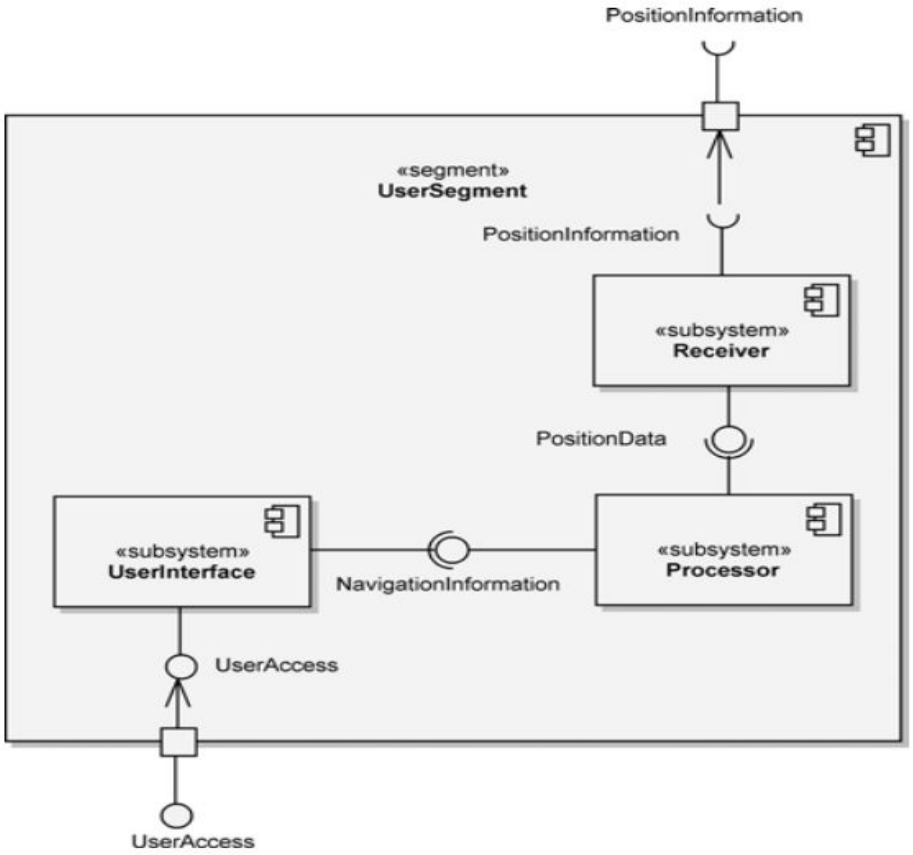
1. Satellite Bus: Offers infrastructure support for the Navigation Payload subsystem. It hosts equipment for power, attitude control, propulsion, and other services.

2. Navigation Payload: 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 Satellite Segment efficiently delivers navigation services by separating infrastructure support from navigation payload functionalities.



The Logical Architecture of the LaunchSegment



The Logical Architecture of the UserSegment

Construction Phase

Construction Phase

At the end of the Elaboration phase, as we pointed 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. 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.

Transition Phase

Transition Phase

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.

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.

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.

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 Control Center 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 Control Center subsystem. Thus, this subsystem acts as an abstraction firewall, which shields all other clients from the intricacies of our particular computing technology.

Supporting Information

Basic UML Terms

- **Abstract Class** - A class that will never be instantiated. An instance of this class will never exist.
- **Actor** - An object or person that initiates events the system is involved with.
- **Activity**: A step or action within an Activity Diagram. Represents an action taken by the system or by an Actor.
- **Activity Diagram**: A glorified flowchart that shows the steps and decisions and parallel operations within a process, such as an algorithm or a business process.
- **Aggregation** - Is a part of another class. Shown with a hollow diamond next to the containing class in diagrams.
- **Artifacts** - Documents describing the output of a step in the design process. The description is graphic, textual, or some combination.
- **Association** - A connection between two elements of a Model.
- **Association Class**: A Class that represents and adds information to the Association between two other Classes.
- **Attributes** - Characteristics of an object which may be used to reference other objects or save object state information.
- **Base Class**: A Class which defines Attributes and Operations that are inherited by a Subclass via a Generalization relationship.
- **Class**: A category of similar Objects, all described by the same Attributes and Operations and all assignment-compatible.
- **Class Diagram** - Shows the system classes and relationships between them..
- **Collaboration**: A relation between two Objects in a Communication Diagram, indicating that Messages can pass back and forth between the Objects.
- **Communication Diagram** - A diagram that shows how operations are done while emphasizing the roles of objects

Basic UML Terms

- **Component Diagram**: A diagram that shows relations between various Components and Interfaces.
- **Dependence**: A relationship that indicates one Classifier knows the Attributes and Operations of another Classifier, but isn't directly connected to any instance of the second Classifier.
- **Deployment Diagram**: A diagram that shows relations between various Processor.
- **Encapsulation** - Data in objects is private.
- **Generalization** - Indicates that one class is a subclass on another class (superclass). A hollow arrow points to the superclass.
- **Event**: In a State Diagram, this represents a signal or event or input that causes the system to take an action or switch States.
- **Final State**: In a State Diagram or an Activity Diagram, this indicates a point at which the diagram completes.
- **Fork**: A point in an Activity Diagram where multiple parallel control threads begin.
- **Generalization**: An inheritance relationship, in which a Subclass inherits and adds to the Attributes and Operations of a Base Class.
- **High Cohesion** - A GRASP evaluative pattern which makes sure the class is not too complex, doing unrelated functions.
- **Low Coupling** - A GRASP evaluative pattern which measures how much one class relies on another class or is connected to another class.
- **Inheritance** - Subclasses inherit the attributes or characteristic of their parent (superclass) class. These attributes can be overridden in the subclass.

Basic UML Terms

- **Instance** - A class is used like a template to create an object. This object is called an instance of the class. Any number of instances of the class may be created.
- **Initial State**: In a State Diagram or an Activity Diagram, this indicates the point at which the diagram begins.
- **Interface**: A Classifier that defines Attributes and Operations that form a contract for behavior. A provider Class or Component may elect to Realize an Interface (i.e., implement its Attributes and Operations). A client Class or Component may then Depend upon the Interface and thus use the provider without any details of the true Class of the provider.
- **Iteration** - A mini project section during which some small piece of functionality is added to the project. Includes the development loop of analysis, design and coding.
- **Join**: A point in an Activity Diagram where multiple parallel control threads synchronize and rejoin.
- **Member**: An Attribute or an Operation within a Classifier.
- **Merge**: A point in an Activity Diagram where different control paths come together.
- **Message** - A request from one object to another asking the object receiving the message to do something. This is basically a call to a method in the receiving object.

Queries?

Thank you