**Mahavir Education Trust's**

**SHAH & ANCHOR KUTCHHI ENGINEERING COLLEGE**

**Chembur, Mumbai - 400 088**

**UG Program in Information Technology**

| **Experiment No: 7** | | | | |
| --- | --- | --- | --- | --- |
| **Date of Performance:** | 9/9/24 | | | |
| **Date of Submission:** | 16/10/24 | | | |
| **Program Formation/ Execution/ Correction**  **(06)** | **Timely Submission (01)** | **Viva (03)** | **Experiment Marks (10)** | **Teacher Signature with date** |
|  |  |  |  |  |

**EXPERIMENT - 07**

**Aim :** Architectural Modeling : Component & Deployment Diagram

**COMPONENT DIAGRAM**

**1. Purpose:**

○ Represents the physical structure of the system, focusing on software components and their relationships.

○ Shows how components are deployed and interact to create a functioning system. **2. Components:**

○ Components represent modular parts of the system, such as software libraries, modules, executables, or databases.

○ Components are typically shown as rectangles with a smaller rectangle (a "lollipop") indicating provided/required interfaces.

**3. Interfaces:**

○ Provided Interfaces: Services offered by the component, shown as a line with a circle.

○ Required Interfaces: Services the component needs from others, shown as a line with a half-circle (or socket).

**4. Dependency:**

○ Dependency arrows show how components depend on one another for functionality.

○ Components may depend on other components for services or functionality. **5. Relationships:**

○ Association: Represents the communication between components.

○ Dependency: Represents one component relying on another for its functionality. ○ Delegation: Shows how a component delegates some responsibility to another component.

**6. Key Elements:**

○ Components: Software units/modules.

○ Interfaces: Points of interaction between components.

○ Dependencies: Indicate relationships and interaction.

○ Ports: Interaction points where interfaces connect components.

**7. Use Cases:**

○ Visualizing component interactions in complex systems.

○ Modeling systems that are distributed or modular.

○ Representing third-party libraries and external dependencies.

**8. Example Components:**

○ User Interface: Frontend component like a web interface.

○ Business Logic: Middle layer managing business rules.

○ Database: Backend component handling data storage.

○ External APIs: Interfaces to external systems like payment services.

**9. Focus on Reusability:**

○ Helps in identifying reusable software components across different systems. ○ Useful for modular system design where components can be swapped or updated independently.

**10. UML Notation:**

○ Component diagrams are a part of the Unified Modeling Language (UML) and follow its notation standards for consistency.

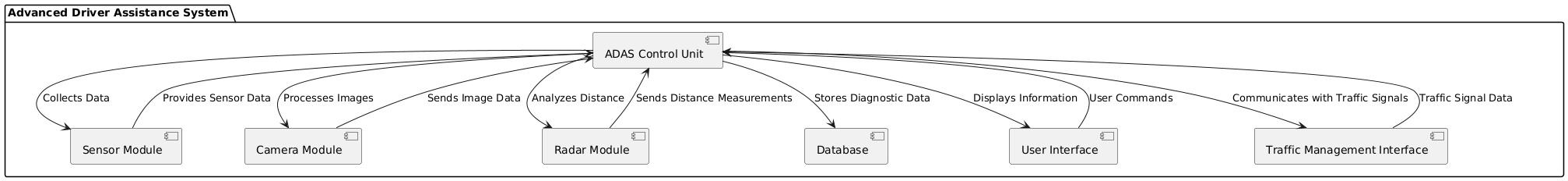
**11. Example Scenario:**

○ In an online shopping app, components could include a shopping cart, payment processing module, user interface, and product catalog. The diagram would show how these components interact and depend on each other.

**12. Design Perspective:**

○ Provides a high-level view of system architecture, making it easier to understand how the system is structured and how parts of the system communicate.

By using component diagrams, developers and architects can clearly visualize the structure of their system and ensure that components are properly integrated and reusable.



**DEPLOYMENT DIAGRAM**

**Purpose**:

● Represents the **physical deployment** of software components on hardware nodes. ● Focuses on how software systems are **distributed** across different **nodes** (hardware, servers, devices).

● Helps in visualizing the **execution environment** and **system infrastructure**. **Nodes**:

● **Nodes** are the physical or virtual computing environments where the system components are deployed.

● They are depicted as **3D boxes** in the diagram and represent servers, machines, or devices.

● Nodes can be **hardware nodes** (e.g., a server, router, mobile device) or **software environments** (e.g., web server, application server, database).

**Components/Artifacts**:

● **Artifacts** (software pieces like executables, libraries, databases) are deployed on **nodes**. ● Artifacts represent the **software code** or **files** that are executed within a specific node.

**Communication Paths**:

● **Communication paths** are lines connecting nodes, representing how data is transmitted between different nodes.

● These lines often indicate protocols such as **HTTP**, **TCP/IP**, or **REST API**. **Key Elements**:

● **Nodes**: The physical or virtual computing infrastructure.

● **Artifacts**: Software elements deployed on the nodes.

● **Communication Paths**: How nodes communicate, showing network connections. ● **Deployment Environment**: Different configurations like development, testing, production.

**Relationships**:

● **Deployment relationships** show which software artifacts are deployed on which nodes. ● They also show the dependencies between different nodes and the interaction between different parts of the system.

**Use Cases**:

● Modeling **distributed systems**, where different parts of the software run on different machines.

● Visualizing the **cloud infrastructure** for a system hosted on platforms like **AWS**, **Azure**, or **Google Cloud**.

● Representing how components interact across **local** and **remote servers**. **Example Components**:

● **Client Devices**: Such as user computers, mobile phones, or web browsers. ● **Web Server**: Handles client requests and forwards them to the application server. ● **Application Server**: Runs the core business logic and connects to the database server. ● **Database Server**: Stores the application's data.

**System Deployment**:

● Shows the system’s **deployment environment**, e.g., development, staging, or production environment.

● Can represent the **hardware** and **software setup** of the system infrastructure. **Cloud or On-premise**:

● **Cloud-based systems**: Deployment diagram might show nodes like **virtual machines**, **cloud storage**, and **load balancers** in platforms like AWS or Azure.

● **On-premise systems**: Focuses on physical servers, databases, and network devices in a local data center.

**UML Notation**:

● Deployment diagrams are a part of **UML**, following standard notation. Nodes are depicted as boxes, and relationships between nodes and artifacts are shown through connectors.

**Example Scenario**:

● For an **online shopping app**, the deployment diagram may show:

○ **Client Nodes**: Web browsers and mobile apps.

○ **Web Server Node**: Handles incoming requests.

○ **Application Server Node**: Manages business logic.

○ **Database Server Node**: Manages data storage.

○ **External Services**: Such as a payment gateway or notification service.

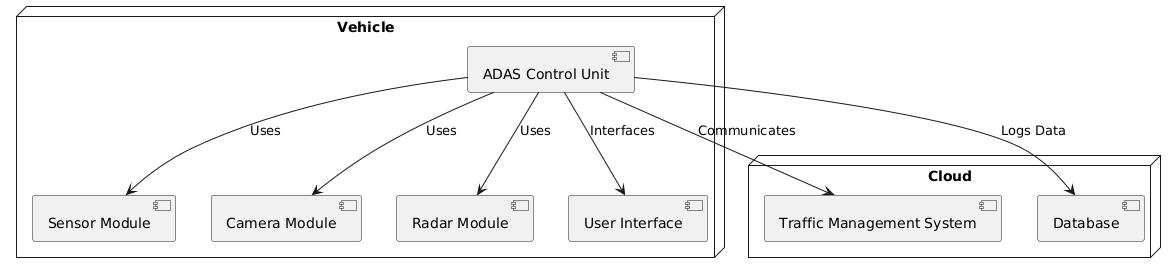
**Communication Protocols**:

● Lines between nodes typically show the **communication protocols** being used, such as **HTTPS**, **REST**, **SOAP**, or **MQTT**.

**Design Perspective**:

● Provides a **high-level view** of the **system’s infrastructure** and how different parts of the system are distributed across hardware and software environments.

● It’s useful in understanding the physical setup of a system, particularly in terms of scalability and distribution.



**Conclusion :**

Component diagrams provide a high-level view of the system architecture, illustrating the various components of an Advanced Driver Assistance System (ADAS) and their interactions. They help identify dependencies and facilitate modular design, enhancing maintainability.

**Key Points:**

* Visualizes system components and their relationships.
* Aids in identifying responsibilities and dependencies.

Deployment diagrams focus on the physical architecture, showing how software components are deployed on hardware nodes. For an ADAS, these diagrams clarify the distribution of components like the ADAS Control Unit and sensor modules, highlighting communication with external systems.

**Key Points:**

* Illustrates the physical deployment of components.
* Analyzes system performance and scalability.

Together, component and deployment diagrams provide a comprehensive understanding of both the design and implementation of a system, ensuring effective communication among stakeholders and guiding development for robust solutions like ADAS.