**PART A**

**EXPERIMENT NO. 7**

**A.1 AIM: -** To draw Component and Deployment Diagram

**A.2 Outcome**

After successful completion of this experiment students will be able to -

1. Practice drawing the component and deployment diagram using StarUML

**A.3 Theroy**

This exercise focuses on component diagram, which depict the implementation of a system.

Component modeling is a specialized type of structural modeling concerned with modeling the implementation of a system. Using the UML, you can communicate the implementation of a system using component diagrams. You usually apply component modeling during design activities to determine how implementation activities will build the system; that is, to determine the elements of the system on which implementation activities will focus. Component modeling typically starts after the design of the system is fairly complete, as determined by your system development process.

1. **Component**

A component is a part of the system that exists when the system is executing. For example, the project management system may be decomposed into the following components:

*A user interface component*

Responsible for providing a user interface through which users may interact with the system

*A business-processing component*

Responsible for implementing business functionality, including all the project management functionality provided by the project management system

*A data component*

For implementing data storage functionality

*A security component*

Provides various forms of security functionality to the business-processing and data components, including user authentication and verifying user privileges when accessing data

You can use the UML to talk about classes of components as well as specific components of a class. When speaking of a class of components, it's customary to use the terms component or *component class*. Thus, while you might think of a component as a specific thing, in the UML, a component really represents a class of things. When speaking of a specific component of a class, use the term *component instance*.

A component exists during execution time and requires a resource on which to execute,. In the UML, a component is shown as a rectangle with two small rectangles protruding from its side. The rectangle is labelled with the name of the component class.

Figure 1 shows various components associated with the project management system, including user interface, business-processing, data, and security components.

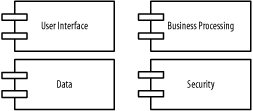


Figure 1- Components of the project management system

A component instance is a specific component. For example, specific components of the project management system include:

*A web user interface component instance*

Allows users to access the project management system via the Web

*A client/server user interface component instance*

Allows users to access the project management system in a client/server environment

*A local data component instance*

Stores project management data for a specific user or group of users

*An enterprise data component instance*

Stores project management data for a complete organization

A component instance is shown similar to a component class, but is labelled with the component instance name followed by a colon followed by the component class name, with all parts of the name fully underlined. Both names are optional, and the colon is present only if the component class name is specified.

Figure 2 shows various component instances of the component classes in Figure 1, including two user interface component instances, named Web and Client Server, two data component instances, named Local Data and Enterprise Data, a nameless business processing component instance, and a nameless security component instance.

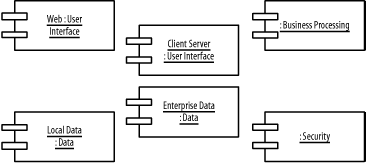


Figure 2- Component instances in the project management system

1. **Dependencies**

Figure 1 shows components associated with the project management system, and Figure 3 shows nodes associated with the project management system, but how are components related to undifferentiated and differentiated classes, packages, subsystems, and to other components and nodes? Specialized types of dependencies called reside, use, and deploy dependencies address these questions. The next few sections discuss these specialized types of dependencies.

**3.1 Reside Dependencies**

A *reside* dependency from a component to any UML element indicates that the component is a client of the element, which is itself considered a supplier, and that the element resides in the component. The element may be an undifferentiated or differentiated class, package, or subsystem. An element may reside in any number of components, and a component may have any number of elements that reside in it.

A reside dependency is shown as a dashed arrow from a client component to a supplier element marked with the *reside* keyword.

Figure 5 shows that the User Interface and Utility packages reside in the User Interface component. Because the User Interface package depends on the Utility package, the User Interface and Utility packages must reside in the same component; otherwise, the User Interface package would not be able to use the Utility package.

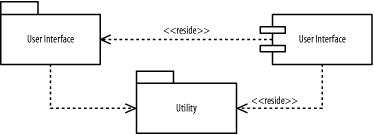


Figure 5. Reside dependencies for packages

Figure 6 shows that the Business Processing subsystem and Utility package reside in the Business Processing component. Because the Business Processing subsystem provides the IBusiness Processing interface, the Business Processing component also provides the interface. Again, because the Business Processing subsystem depends on the Utility package, the Business Processing subsystem and Utility package must reside in the same component; otherwise, the Business Processing subsystem would not be able to use the Utility package. Remember, it's perfectly fine for an element to reside in more than one component. For example, the Utility package resides in both the User Interface and Business Processing components, and, as you will soon see, in the Data component.

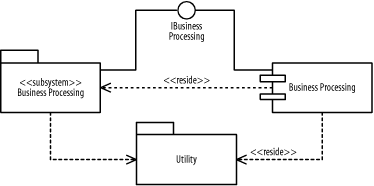


Figure 6. Reside dependencies for subsystems

Alternatively, an element that resides inside a component may be shown nested inside the component. Figure 7 shows that the Data subsystem and Utility package reside in the Data component. The Data subsystem is drawn inside the Data component, while the reside dependency to Utility is still drawn in the same manner as in Figures Figure 5 and Figure 6.

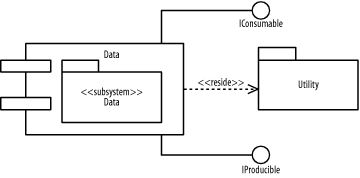


Figure 7. Reside dependencies using nesting

Notice that the Utility package resides in all the components in Figures Figure 5, Figure 6, and Figure 7, because each component described in those figures has a package that uses the Utility package.

**3.2 Use Dependencies**

A *use*dependency from a client component to a supplier component indicates that the client component uses or depends on the supplier component. A use dependency from a client component to a supplier component's interface indicates that the client component uses or depends on the interface provided by the supplier component. A use dependency is shown as a dashed arrow from a client component to a supplier component or a supplier component's interface. The dependency may be marked with the use keyword; however, the keyword is often omitted because this is the default, and the meaning is evident from how the dependency is used.

Figure 8 shows how the various components of the project management system are related:

The User Interface component-

Uses the Security component and the IBusiness Processing interface provided by the Business Processing component

The Business Processing component-

Uses the Security component and the IProducible and IConsumable interfaces provided by the Data component

The Data component-

Uses the Security component

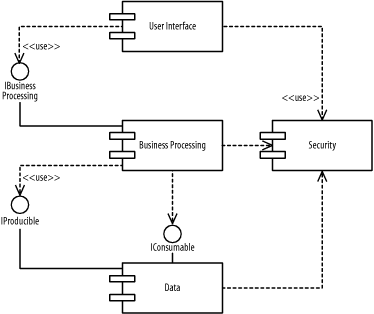


Figure 8- Use dependencies

**3.3 Deploy Dependencies**

A *deploy*dependency from a client component to a supplier node indicates that the client component is deployed on the supplier node.

A deploy dependency is shown as a dashed arrow from a client component to a supplier node marked with the *deploy* keyword.

Figure 9 shows that the User Interface component is deployed on the Desktop Client node.

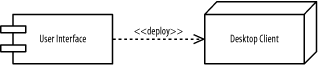


Figure 9- Deploy dependencies

Figure 10 shows that the *Business Processing* component is deployed on the *Business-Processing Server*node.

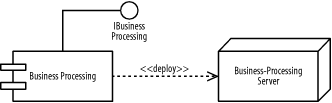


Figure 10- Deploy dependencies for a subsystem

Alternatively, a component that is deployed on a node may be shown nested inside the node. Figure 11 shows that the Data component is deployed on the *Database Server* node.

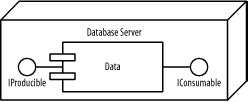


Figure 11- Deploy dependencies using nesting

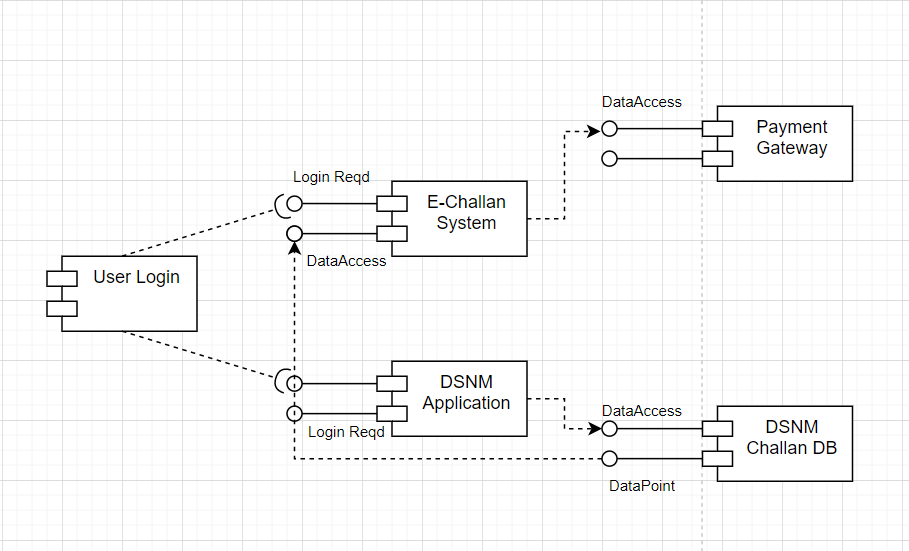
**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practicals. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

|  |  |
| --- | --- |
| Roll No: C101 | Name: Samkeet Shah |
| Class: C | Batch: B3 |
| Date of Experiment: | Date of Submission: |
| Grade: |  |

**B.1 Component Diagram**

****

**B.2 Conclusions**

The basic concepts of a component diagram were understood. With regards to the current case, the various states involving a system were implemented.

**B.3 Question of Curiosity**

1. What are the standard notations used for component diagrams?

Sol. A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development. In the first version of UML, components included in these diagrams were physical: documents, database table, files, and executables, all physical elements with a location.

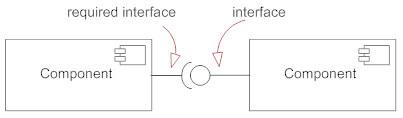
#### Component

A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word written above the name of the component to help distinguish it from a class.



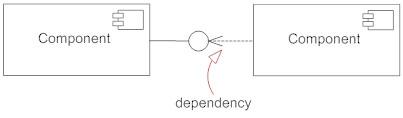
#### Interface

An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.



#### Dependencies

Draw dependencies among components using dashed arrows.



#### Port

Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.

