**Experiment No.: 10**

**Implementation of Software Architecture for identified system/application.**

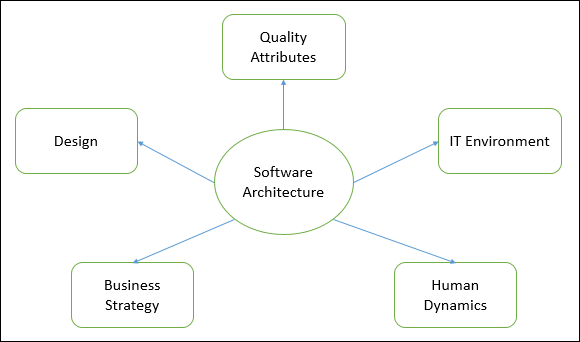
**Learning Objective:** Student should be able to understand wrapper the implementation of Software Architecture for a system/application.

**Theory:**

Software architectures describe how a system is decomposed into components, how these components are interconnected, and how they communicate and interact with each other. When poorly understood, these aspects of design are major sources of errors.

Software architecture of a system describes its major components, their relationships, and how they interact with each other.

It essentially serves as a blueprint. It provides an abstraction to manage the system complexity and establish communication and coordination among components.

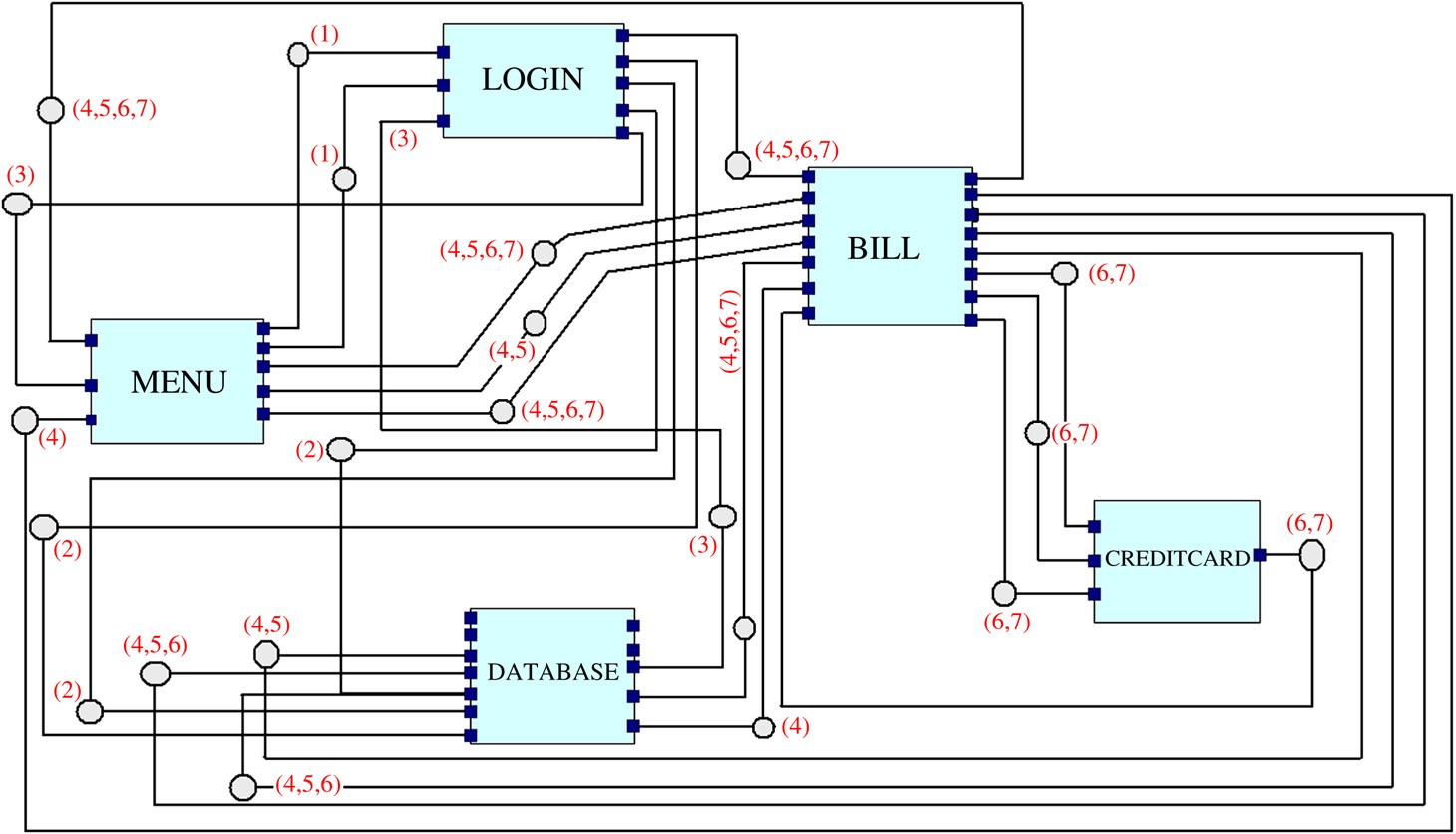


It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.

* The architecture helps define a solution to meet all the technical and operational requirements, with the common goal of optimizing for performance and security.
* Designing the architecture involves the intersection of the organization’s needs as well as the needs of the development team. Each decision can have a considerable impact on quality, maintainability, performance, etc.

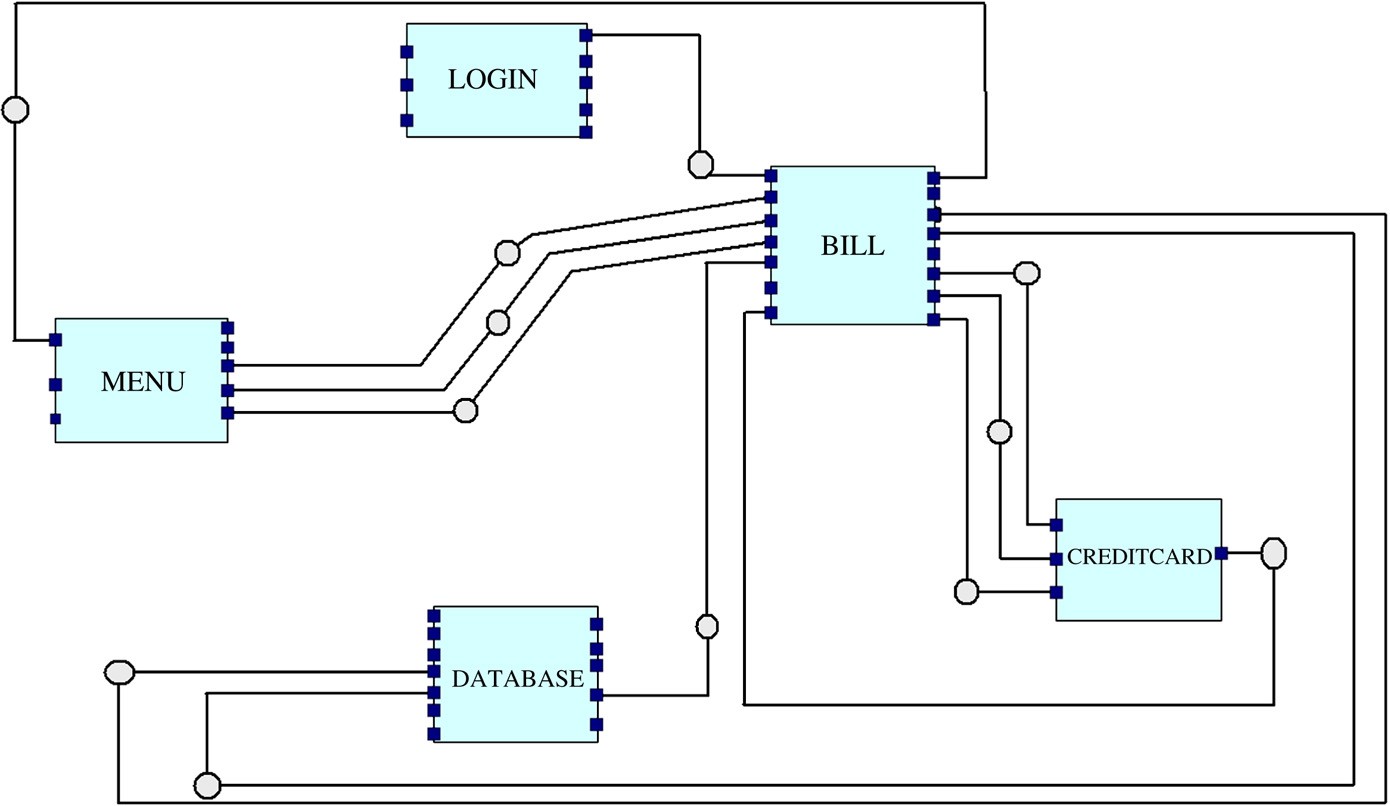
**Implementation of Application :**

The system aims at addressing the needs of a typical hospital accounting service, using a component-based approach. The system supports account keeping for each patient. Each patient has his private account, and each user has an authorization level. The accounts can be charged or paid only by users with high authorization, while users with normal authorization are only able to print statements for the patients’ accounts. For the payment of an account, a number of alternative payment methods such as credit card payment, cash payment and banking account transfers are provided.



The system described here can be viewed as part of a more complex system, connected to other components and services. The latter more complex system is effectively an amalgamation of a couple of other systems, designed for tele-medicine applications within the context of research projects. These systems have been developed over a period of a year by 2–3 programmers. However, it is interesting to note that the approach developed in this paper allowed for the high-level modeling of the software architecture much faster and in a more natural way. Furthermore, the production of executable code was also a faster process, as much of the code was automatically generated from the tool. For implementation, we use Application as the coordination language and the components are written. ACME is used for the system’s architectural design. For the transition from the ACME code to the Application code, . Deriving the high-level architecture is a stepwise approach:

1. First, the components and their ports are identified. The only interaction of a component with its environment is through the ports. A port in the IWIM model as well as in most ADLs can be realized as a buffer, temporarily storing information, until it is read by another component (to be accurate, the connectors in ACME — the entities that connect two ports The architecture is designed in AcmeStudio (Eclipse plug-in). For clarification reasons, we enhanced the diagram with state transition information. When a component needs to output a specific value-result to another component, it just pushes this value into the appropriate output port.
2. Second, the connectors are detected, that is, the actual interactions between two components through their ports. More specifically, we detect which components interact using which ports. We also detect which of these connectors are executed concurrently and annotate them with the same Active\_on number. The idea is to detect all the expected states in a given component configuration. Such a state definition could include the following: e.g. in state 6, component Bill.Out\_port\_6 sends to component CreditCard.In\_port\_1 some data.
3. Third, the ADL design is enhanced with the properties defined earlier. This allows the generation of the better part of the coordination code, and leaves very few tasks for the programmer to complete manually.
4. Finally, the ACME code is parsed and the equivalent Application code is generated.



1. The application consists of the following five components:
2. Login: Verifies username and password and keeps the authorization level of the user.
3. Bill: Implements the basic accounting functions: insert/delete customer, pay/charge bill, print statement.
4. CreditCard: Communicates with appropriate bank systems to charge the account of a customer that pays using a credit card. This component is used by the Bill component in case the payment method selected by a customer is “credit card payment”.
5. Database: Receives requests for data handling from the other components and executes them.
6. Menu: Implements the interaction between the user and the system.

Each of the above components was implemented as a manager process coordinating an atomic process, which in turn implements the functionality of the component. The five manager processes are coordinated by the Main Application. It is interesting to note here that although the system could be implemented using five atomic processes coordinated by the Main Application, we have chosen to wrap every atomic process in a manager process in order to keep computational units unchanged in case of a system change or evolution. In other words, we have separated the computational from the coordination concerns and in that respect we have generated code that can potentially have a higher degree of reusability than it would have had otherwise, had we generated only an equivalent fragment of executable code.

In the current state of the system, each of the manager processes simply passes data from its input ports to the input ports of the atomic processes that it coordinates and reversely from the output ports of the atomic processes to its own output ports. However, more sophisticated coordination scenarios can be supported. The connectors represent Application streams. Each connector can be active in one or more states.

Our example includes the following states:

1. Menu\_Requests\_Login: The MENU component sends the username and password to the LOGIN component, so that the user’s authorization is checked.
2. Login\_Requests\_Verification: The LOGIN component asks from the DATABASE component to perform a query for the user’s authorization.
3. Login\_Gets\_Verification: The LOGIN component gets the authorization level from the DATABASE component, and returns the answer to the MENU component.
4. Menu\_Requests\_Bill\_Info: The MENU component requests information for a given customer from the BILL component, which in turn asks for some information from the DATABASE component and returns the answer to the MENU component.
5. Menu\_Requests\_Bill\_Create: The MENU component requests the creation of a bill from the BILL component for a new customer. The BILL component prepares and sends the related data to the DATABASE component. The DATABASE component sends back a confirmation to the BILL component, which returns the confirmation to the MENU component.
6. Menu\_Requests\_Bill\_Insert/Update: The MENU component requests the insertion or updating of a bill from the BILL component. The BILL component prepares and sends the related data to the DATABASE component. Also, if credit card payment is included, BILL sends the relevant data to the CREDITCARD component, which makes the transaction and sends the result back.
7. Menu\_Requests\_Bill\_Delete: The MENU component requests the deletion of a bill from the BILL component. The BILL component sends the relevant data to the DATABASE component, and waits for deletion confirmation. It then notifies the MENU component that the deletion is successful.

the methodology is able to implement a complete Application, i.e. the state and all the streams.

**Result and Discussion:**

**Learning Outcomes:** Students should have be able to understand

LO1: Define software Architecture.

LO2: Identify different phases in software Architecture.

LO3: Explain Implementation of software Architecture .

.

**Course Outcomes:** Upon completion of the course students will be able to understand Implementation of software Architecture.

**Conclusion:**………………………………………………………………………………………

**Viva Questions:**

1. Define Software Architecture.
2. Explain difference between software Architecture and software design.
3. Explain Implementation of software Architecture with a example.

For Faculty Use

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** |  |
|  |  |  |