3. Architecture

The persistence framework developed in this report can be regarded as an intermediate layer between the user application and the persistence mechanism. User objects to be stored are passed to the framework which then perform the mappings to allow the objects to be stored in the persistence mechanism, which can be any possible data store such as relational databases, object databases, flat files, and so on. The framework shields the user from the internals of the persistence mechanism, thus allowing a different persistence mechanism to be used without having to make changes to the user application.

Using a layered architecture view as described in [24], the persistence framework can be viewed as a layer between the system layer and the domain/business layer of the user application, as shown in Figure 3.1. The arrows indicate dependencies between layers. The user interface layer consists of classes that interact directly with the user, such as dialog boxes, reports, and documents. This layer can send messages to the domain/business layer as well as the system layer. The domain/business layer implements classes that are specific to the domain of the application, such as *Patient* and *Hospital* in

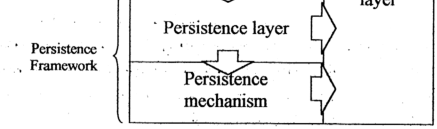

Figure 3.1 – Layered architecture (adapted from [6])

the healthcare domain, *Customer* and *Supplier* in the manufacturing domain, and so on. This layer communicates with the system layer and persistence layer. The persistence layer encapsulates access to the persistence mechanism, enabling the domain/business layer to store and retrieve its objects. This process is not visible to the user, and is effectively de-coupled from the user interface layer.

As shown in the diagram, the persistence framework consists of two layers, the persistence layer and the persistence mechanism layer. The persistence layer handles all requests through a main *PersistenceBroker* class which provides the persistence services for objects. Objects to be saved are passed to this layer by higher layers, and similarly will restore and return objects to the requesting layer. The persistence layer performs the necessary object-to-relational mappings and interacts with the persistence mechanism layer, which contains classes that facilitate handling of relational data from databases and also performs connections, transactions and querying.

The advantage of this class-type architecture is that the robustness and maintainability of source code increases due to the weak coupling between layers. Classes can be changed within a layer without affecting the external interface to other layers. Even if the interface is changed, it only affects the layers directly above it and not further. This allows the persistence layer much flexibility in using persistence mechanisms because the persistence mechanism layer does not depend on how the object-to-relational mapping is performed. It only stores and retrieves data in relational table form. Therefore objects can be stored remotely, transferred to other mechanisms or even distributed across several databases, all the time being transparent to the user. Also, support for new persistence mechanisms can be added in easily without having to modify any other code.

Using this architecture, the user application and persistence framework can be deployed in various hardware and network configurations. It is scalable from a small single-user system to a large enterprise system, as shown in the Table 3.1. For the stand-alone system, both the persistence layer and persistence mechanism are implemented on the client itself. Thin-client and Fat-client systems have the persistence services on a server whereas n-Tier systems can implement the persistence layer and persistence mechanism on separate servers. This allows the persistence mechanism layer to run on the database server itself, and is suitable for large relational database systems where one or more specialised servers are normally used to run the database system.

Class Type	**Stand-alone**	**Thin-client**	**Fat-client**	**n-Tier**	**Distributed**
User interface	Client	Client	Client	Client	Client
Domain/business	Client	Server	Client	Application server	Do not care
Persistence	Client	Server	Server	Persistence server	Do not care
Persistence mechanism	Client	Server	Server	Persistence or Database server	Do not care
System	Client	All machines	All machines	All machines	All machines

Table 3.1 – Deployment configurations for persistence (adapted from [6])