Rory Mackintosh
Edinburgh Napier University
40492068

Software Architecture Coursework

Contents

1. Introduction

When deciding on which software architectures to recommend to DE-Store I took into consideration the key aspects required by DE-Store. The first key piece of information I considered was that DE-Store is a distributed system. A distributed system is an environment where multiple computers are working on a variety of tasks. In DE-Store's case they require a distributed management system for their retail branches for better coordination of their business. DE-Store's distributed management system aims to have five key functions.

A price control function will allow DE-Store managers to set the prices of their products and set a variety of sale offers on different products. For example, managers will be able to set offers such as buy one get one free on products of their choice.

An inventory control function will give DE-Store managers the ability to monitor stock by uploading data from a warehouse database. Items out of stock will be ordered from the central inventory system at the headquarters. This function will also automatically generate warning messages to store managers when stock levels are low by sending messages via mobile or email.

A loyalty card function will allow DE-Store managers to give further special offers on certain products to customers who hold a loyalty card.

A finance approval function will give DE-Store customers the opportunity to buy now and pay later using an online finance system known as Enable. This system will be linked to DE-Store via a portal which managers can access from the distributed system that will be created for DE-Store.

Finally, a reports and analysis system will give DE-Store managers a detailed report on multiple factors of the store such as tracking purchase activities of customers from the accounting database. These generated reports will tell managers how the store is performing.

Now that a clear understanding of what is required by DE-Store has been established. Two software architectures that can be recommended to implement into DE-Store's distributed management system are communicating processes architecture, specifically the client server-model and service-oriented architecture.

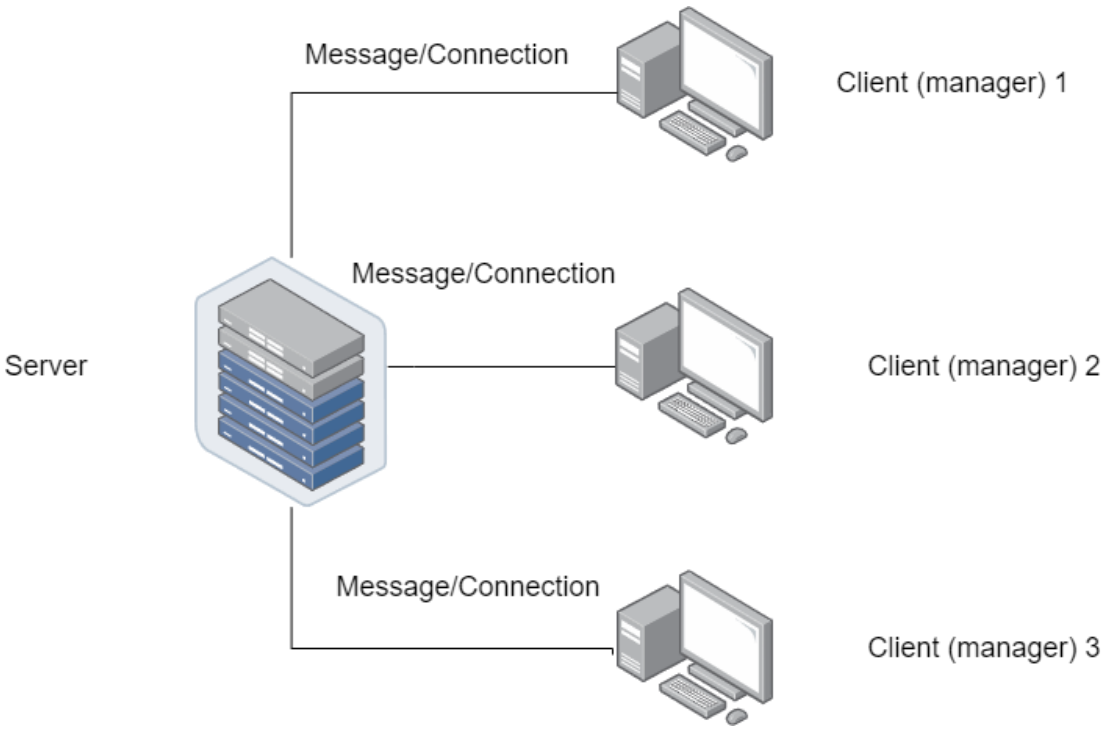
These two software architectures offer a number of benefits for distributed systems and will make for an ideal fit into DE-Store's distributed management system.

2. Client-Server Architecture

The client-server architecture style is a model of communicating processes architecture style that can be implemented within distributed systems. The client-server model is the most commonly used model amongst communicating processes architecture style. Before discussing why the client-server architecture style will be a good fit into DE-Store's distributed management system, a simple background explanation for the communicating processes architecture style will be discussed. Communicating processes architecture comes from a family of independent component architecture. As explained by Clements (2005), the communicating processes style is characterised by the interaction of concurrently executing components through various connector mechanisms. This explanation of communicating processes architecture style clearly defines the style as multiple independent components communicating via various connector types such as data exchange, message passing, synchronization, control and other types of communication across various hardware in concurrent units such as tasks, processes and threads. The communicating processes architecture style is most commonly used in large systems and are necessary in all distributed systems.

Now that an understanding of the communicating processes architecture style has been established a thorough discussion and proposal of the client-server model can be reviewed. As explained by modassirali (2022), the client-server model is a distributed application structure that partitions tasks or workload between the provider of a resource or service, called servers and service requesters called clients. When the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packet requested by the client. The client-server model would be an excellent choice for DE-Store's distributed management system for many reasons. The client-server model offers DE-Stores managers the ability to request important data from their servers such as stock count and store performance. The client-server model also gives DE-Store's managers the ability to change data held on the server such as price control

functionality. The client-server model is an effective architecture type for a system such as DE-Store's distributed management system as multiple managers can access DE-Store's servers at the same time. The client-server model has the ability to offer access to a server from multiple clients across different locations.



(Figure 1: Simple Diagram Of DE-Store's Client Server model)

Client-server models are typically made up of three components. Client, server, and networking devices. The client component is the hardware that users use to send requests to the server. Clients can be computers, phones or any other device that allow users to connect and communicate with a server. The server component is a piece of powerful computing hardware that retain a fast-processing speed, more storage space and robust memory to deal with multiple requests approaching simultaneously from various clients. At the same time a server preforms a number of different functions such as database servers or file servers. Clients and servers are interconnected with each other by means of network device. Each networking device used in client-server architecture has its own operations and properties.

As established the client-server model allows servers to provide a service to an unknown number of clients. Clients know the location of the server and request information via a connector in which the server replies. Connectors are message-based connections using queues. Connectors can be connection oriented also

known as a TCP (transmission control protocol). Connection oriented protocols establish and maintains connections until the application programs at each end have finished exchanging messages. Client-server model connectors can also be connectionless also knows as a UDP (user datagram protocol). A UDP connectionless client-server model works by having the user send the server a datagram. This means that the server does not need to accept a connection and can just wait for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data back to the correct client. The main comparison that can be made about both these connector types, stated by Gorman (2023). While TCP is more reliable, it transfers data more slowly. UDP is less reliable but works faster. This shows that both connection types are a valid option depending on the task the user is trying to accomplish. Other connector types of the client-server model are RCP (remote procedure call) which is a procedure call that can be used to request a service from a program located on another computer or network without having to understand that networks details. Broadcast messages are allowed within the client-server model which allows a message to be sent to all clients currently connected to that server. Finally web based client-server models can send messages across the world wide web between client and server by using http protocols allowing web-based applications to function with thousands of users across the world.

Now that an understanding of the client-server model has been established, this raises the question of how it can be applied to DE-Stores's distributed management system. As mentioned previously DE-Store requests a number of features for their distributed system. Each feature will require the client to send a request to the server in order to retrieve and execute that specific function. For example, if a manager wanted to change the price of an item within their store, they would access the work computer where they would change the price of an item via the user interface. What this is really doing is turning that user's computer into a client which sends a message to a server. That message connects the client to the server giving the manager of that branch access to the DE-Store server. From there the manager can change the price of an item that is being held on DE-Store's server via a database which in turn will update the whole system for that store.

2.1 Client/Server Architecture Style Advantages

All software architecture styles come with a number of advantages and disadvantages. The client/server model is no different. As previously discussed, the client/server model works well within distributed systems which is what DE-Store are looking to implement. Distributed systems offer benefits such as economy, performance, scalability, reliability, and availability.

Some of the advantages of the client/server model are distribution for performance and scalability. Distribution in the client/server model allows for better performance as the workload is distributed between client and server. The server handles data processing and storage while the client manages user interface. This will allow DE-Store's system to handle larger amounts of users and data.

Highly cohesive components are another advantage of the client/server model. The components of the client/server model have distinct roles and responsibilities. The client focuses on presentation to the users whereas the server focuses on business logic and other processes. This separation makes it easier for developers to design, develop and maintain each component. This results in better code reusability and modifiability.

Modifiabilyt allows the client/server model to be easily changed for development purposes. As long as the communication protocols remain consistent, modularity makes it easier to update specific parts of the system without affecting the others. This improves flexibility and reduces the risk of errors.

Portability gives the client/server model the ability to work on different platforms such as windows, macOS and Linux. The client/server model relies on standardised communication protocols such as HTTP which further enhances its portability. This allows developers to develop a system that uses the client/server model that works on multiple systems regardless of operating system.

The client/server model promotes interoperability which allows this architecture style to have both client and server components communicate over networks using common protocols. This allows systems to be built with a mix of technologies and platforms, facilitating integration with existing system and services. This will allow

DE-Store's system to be developed upon when they wish to change their technologies.

Finally, the client/server software architecture style offers reusability within development. For example, client libraries, server modules, or APIs can be used across various applications and services. This gives developers an advantage when creating new systems as it reduces development time.

2.2 Client/Server Architecture Style Disadvantages

The client/server model has a number of disadvantages that can be the deciding factor whether this architecture style is an appropriate fit into a system that is being developed. Two of the main disadvantages of the client/server architecture style are unreliable communications and the client/server model is non-deterministic which means the vary or request can vary based on factors such as network connection. Non-deterministic behaviour makes the client/server model hard to test.

The client/server model relies heavily on network communications to allow the client to communicate with the server and vies-versa. This can be disadvantage as there are various factors that can lead to unreliable communications. Some factors are network congestion, latency, packet loss, and network failures. This can cause the data that was being delivered to be delayed which can disrupt the systems normal functionality. This could cause DE-Store problems as their system will be used for functions such as stock control and price control. If the network happens to be down, then this could cause damage to their business and halt operations.

Due to the client/server model being a non-deterministic architecture type, it can be difficult to test. Some of the main challenges that developers face when trying to test a client/server system are writing deterministic test cases that consistently produce the same results. Testing specifically for concurrency issues as they become complex and may require specialised testing such as frameworks and techniques.

2.3 Client/Server Architecture Style Evaluation

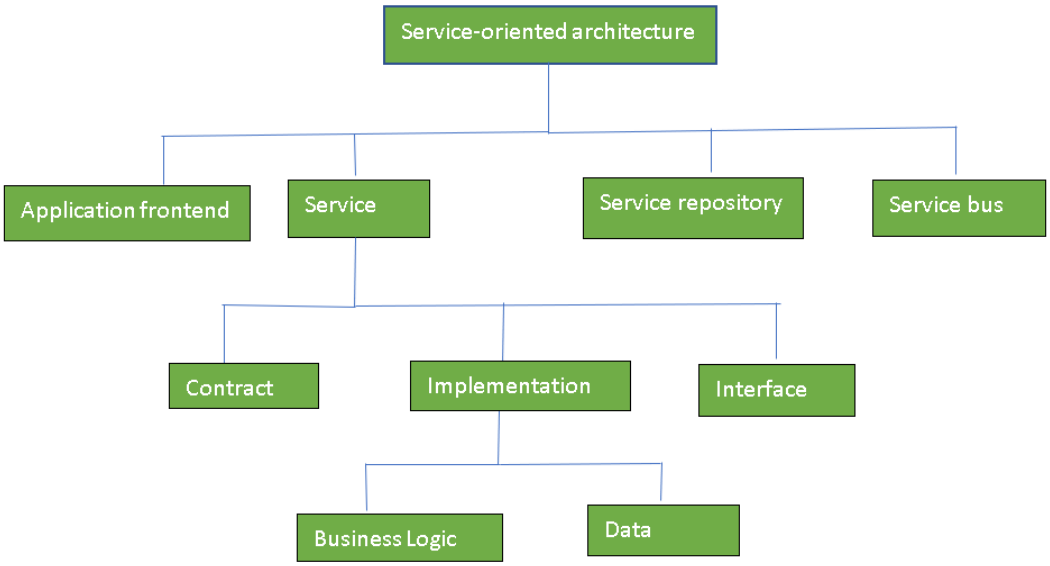
Overall, the client/server model is an ideal software architecture style for DE-Store's distributed management system. The client/server model works well within distributed systems due to the many factors previously discussed such as its modularity and individual component-based architecture. The client/server model will provide the benefits needed to implement DE-Store's distributed management system along with the requested functions. The client/server model will allow multiple managers to connect to the server at the onetime to allow an increase in productivity and allow a constant smooth operation within DE-Store's business.

3. Service-Oriented Architecture Style

The service-oriented architecture style is the second architecture style proposed for DE-Store's implemented distributed management system. An enterprise system such as DE-Store's distributed management system faces several challenges. Seamless integration of various systems, system agility, allowing access from anywhere anytime, providing services to customers and partners inside and outside the enterprise. In addition to system functionality, quality considerations like extensibility, flexibility, connectivity, and interoperability also demand enterprise functions to be easily accessed and composed via published interfaces. Challenges such as these faced by enterprise systems requires a flexible, standardised architecture to better support the connection of various applications and the sharing of data. The service-oriented architecture style offers solutions to such problems.

The service-oriented architecture style unifies business processes by structuring large applications as collection of smaller modules called services for specific purposes. These applications create a system that can be used by different groups of people both inside and outside the company. This benefits a large business such as DE-Store as it can allow them to tailor the system to individual stores based on store details such as specified functionality for different stores. Service-oriented architecture style can be built from a mix of services from the global pool this allows the architecture style to exhibit greater flexibility and uniformity.

The service-oriented architecture style is composed of components and connectors. The components of service-oriented architecture are called services, and the connectors are called service calls and communication protocols. Services are essential un-associated units of functionality. Services typically implement functionalities recognised as business services; in DE-Store's case these business services would be the requested functionality that DE-Store would like implemented such as a price control function. Instead of services embedding calls to each other in their source code, protocols are defined which describes how services can talk to each other. The service-oriented architecture style then relies on a business process expert to link and sequence services, in a process known as orchestration to meet a new or existing business system requirement.



(Figure 2: Service-oriented architecture diagram)

The service-oriented architecture style follows a number of principles to provide a well-designed and robust architecture style. These principles are service loose coupling, service contract, service reusability, service composability, service autonomy, service optimisation, and service discoverability.

Service loose coupling maintains a relationship that minimizes dependencies and only requires that they maintain an awareness of each other. This allows service-oriented architecture to have scalability and flexibility within the system it has been

implemented in. minimizing dependencies between services reduces the risk of modifications or faults affecting the entire system.

A service contract is implemented to make services within service-oriented architecture adhere to a communication agreement defined collectively by one or more service description documents. These service contracts are agreements between service providers and service consumers. Service providers and service consumers must come to a legal agreement as well as a technical agreement so both parties can come up with a way to send data in a secure manner.

Service reusability is not only a principle of service-oriented architecture but a benefit as logic is divided into services with the intention of promoting reuse. Reusability allows developers to reuse code during development allowing for quicker results and more reliable code.

Service composability allows collections of services to be coordinated and assembled to form composite services. This contrast the service-oriented architectures promotion of independent services but allows services to be combined for specific functionality making it beneficial.

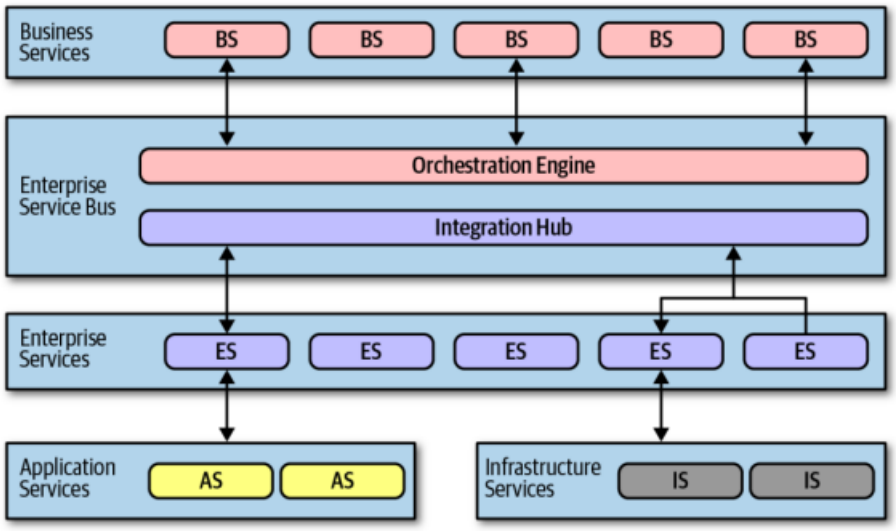
Service autonomy makes sure services have control over the logic they encapsulate. For services to carry out their capabilities consistently and reliably, their underlying solution logic needs to have a significant degree of control over its environment and resources. If achieved, then services within the service-oriented architecture will be autonomous allowing for greater benefits and a more robust system.

Service optimisation ensures high quality services are considered preferable to low-quality services. Optimised services will lead to more advantages and a better running system.

Service discoverability is an important aspect of service-oriented architecture. Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms. Services that are easily discoverable are also more likely to be reused in development.

Service-oriented architecture is based on service providers communicating with service consumers. Services are the most important part of service-oriented architecture. There are many different types of services based in a business system.

Each service is a different layer within a business system. Each layer communicates between one another based on different communication protocols. These layers of services include business services, enterprise services, application services and infrastructure services.



(Figure 3: Topology of Orchestration-Driven-Service-Oriented Architecture)

Business services are at the top of service-oriented architecture and provide an entry point for users. The business services layer is the most basic layer of the service-oriented architecture style, only containing inputs, outputs and sometimes schema information.

Enterprise services is the second layer of the service-oriented architecture style. Enterprise services contain fine-grained, shared implementations. Typically, a team of developers is tasked with building atomic behaviour around particular business domains. These services are the building blocks that make up the fine-grained business services, tied together via the orchestration engine.

Application services are one-off single implementation services. For example, if a business needs a geo-location function but the organisation does not want to take the time or effort to make that a reusable service, a single application team takes on the problem and solves it.

Infrastructure services supply the operational concerns such as monitoring, logging, authentication, and authorisation. These services tend to be concrete

implementations, owned by a shared infrastructure team that works closely with operations.

3.1 Service-Oriented Architecture Advantages

The service-oriented architecture style offers many advantages to businesses that implement the architecture into their systems. Some of these advantages are unique features such as open infrastructure, dynamic services, provider independence, public advertising, and service providing.

Service-oriented architecture provides open infrastructure to the systems this style has been implemented in to. Open infrastructure are pieces of functionality that are packaged and published as services. This allows functions to be reused as different services in different systems. This is an advantage of the service-oriented architecture style as it allows for more reusability and modularity within systems. Allowing services to be packaged and published also helps development of service-oriented systems as developers will be able to use the already developed functions within their own systems.

Service-oriented architecture allows services to appear dynamically, potentially at run-time. This is known as run-time service binding. They must be discovered via a communication protocol. The composition into a service-oriented system is loose and subject to frequent change. This gives the service-oriented architecture the advantage of being modular with each service being available at runtime rather than being made available at all times allowing for a more optimised system.

Another advantage of service-oriented architecture is provider independence. This allows different services to be provided by different providers. This advantage of service-oriented architecture allows for better modularity within systems as services can be switched or upgraded without affecting the overall system. This can help businesses save money on managing costs for businesses that need to adapt to new challenges.

Service-oriented architecture provides public advertising to developers who are publishing services and looking for services to implement into their systems. Services are normally registered in a service directory or repository. This makes

discovering services easier for developers. The advantage of publicly advertising services leads to enhance system scalability and interoperability since new services can be added and discovered dynamically.

Service providing is what makes service-oriented architecture a popular architecture style and is the main functionality behind it. As previously discussed, services can encapsulate specific functionality and can be reused across various applications. The advantages of service providing is reusability and modularity which saves on development time and improves software quality.

3.2 Service-oriented Architecture Disadvantages

As with every architecture style, service-oriented architecture comes with many challenges and disadvantages. Service-oriented architecture relies heavily on services communicating with each other via communication protocols this can be costly for both bandwidth and finances. Implementing service-oriented architecture systems can be costly in terms of human resources, development, and technologies. Large amounts of computational power may be required depending on the amount of services processing inputs at run time which may lead to extra overloading.

The service-oriented architecture style may require high bandwidth servers. Some web services send and receive messages and information frequently. This can build up to millions of requests per day. This can be a disadvantage to businesses who implement service-oriented architecture as high bandwidth servers can be costly.

The service-oriented architecture style is a costly architecture style to implement in terms of human resources, development, and technologies. Service-oriented architecture requires a large team of developers to develop and maintain service-oriented systems. New technologies are also required to keep systems up-to date and running smoothly. Up to date technologies will also allow bigger businesses to handle a larger number of users.

Rehman (2021) mentions that Service-oriented systems are at risk of extra overload leading to higher computational needs. Service-oriented systems can get extra overloaded due to all the inputs being validated before it is sent to the service. If

multiple services are required, then it will overload the system with extra computation.

3.3 Service-oriented Architecture Evaluation

The service-oriented architecture style would work within the DE-Store's distributed management system. However, the client/server architecture style will be more suitable for developing DE-Store's distributed management system. The service-oriented style offers a number of benefits that would give DE-Store the system they need to conduct a smooth operation within their business. The services that make up the component of this architecture style is a suitable way of implementing the requested functionality of the DE-Store distributed management system. Specific services can be called by communication protocols when a store manager accesses the system to do a certain function such as check stock.

Overall, the service-oriented architecture style would be a suitable architecture style to implement into DE-Store's distributed management system.

4. Proposing the Client/Server Architecture Style for DE-Store's Distributed Management system

A proposal of the client/server architecture style for the implementation of DE-Store's distributed management systems and plans to develop a prototype to display the benefits of implementing such system has been reviewd. The client/server offers a number of benefits that are well suited for the requested functionality of DE-Store's distributed management system.

The reasons to propose the client/server architecture style is due to the number of quality attributes the client/server model presents to systems. These quality attributes being applied to DE-Store's distributed management system prototype will display the value of the client/server model. The main quality attributes of the client/server model are modifiability, portability, interoperability, reusability, simplicity, centralised data management, and security.

The client/server model supports modifiability to the system. This allows developers to make changes to the client or the server components independently. This is a quality attribute of the client/server model as future or changing business requirements can be implemented without affecting the whole system. DE-Store's distributed management system will find this attribute useful as any further requirements requested by DE-Store that are client specific or server specific can be added without affecting the entire distributed management system.

Portability is another quality attribute of the client/server model that allows developers to port the system where needed. For example, moving the system to new servers can be made easier with a client/server model that has been made portable. Portability can vary based on the technologies chosen by the business the system is being built for. This will benefit DE-Store as portability will make setting up the distributed management system in multiple stores a lot easier for developers. This will also benefit DE-Store if they decide to change the technologies that the system is held on in the future.

The client/server model relies on standardised communication protocols to communicate between client and server. Standardised communication protocols involve http, TCP or UDP. This gives the client/server model more interoperability which is another quality attribute of this architecture style. Interoperability makes communication between client and server built on different technologies and platforms easier to achieve. This will allow DE-Store's managers have access to DE-Store's servers via standardised communication protocols even if the system is designed with different technologies.

Reusability is another quality attribute of the client/server model which can be supported by its modular design. Reusability simply means functions from both the client and server components can be reused in future or new projects. This will allow quicker development when modifying the DE-Store distributed management system when reusability can be utilised.

Centralised data management is one of the most important quality attributes the client/server model offers. Centralised data management contributes to data integrity, consistency, and easier access data control. This is an important aspect of DE-Store's distributed management system as a lot of the functions requested by DE-

Store requires the client sending request to the server to change data that is being held on the server. Having centralised data management embedded in their system will result in a well-functioning robust system that DE-Store can utilise.

The final quality attribute to discuss when proposing the client/server model for development is security. Security can be centralised on the server just like the data management. By adding security measures to the server such as controlling access to data security to DE-Store's distributed management system, security can be enhanced. Extra security features such as access controls, authentication, and encryption can all be added for more effective security. Adding this level of security to DE-Store's distributed management system will stop lower-level employees of DE-Store gaining access to the server and accidentally or purposefully changing the server data in any way.

The client/server model has a number of quality attributes that make it the perfect software architecture for DE-Store's distributed management system. To further expand upon these quality attributes, plans to design and develop a working prototype of DE-Store's distributed management system using the client/server software architecture have been made. The goal of my prototype is to display all the requested functionality DE-Store would like to see such as price control, stock control and customer buy now pay later function using the client/server architecture. Plans to display each of the quality attributed previously discussed within my prototype to reinforce my statements of the client/server architecture style have been made.

5. Design For DE-Store's Distributed Management System Prototype Using The Client/Server Architecture Style

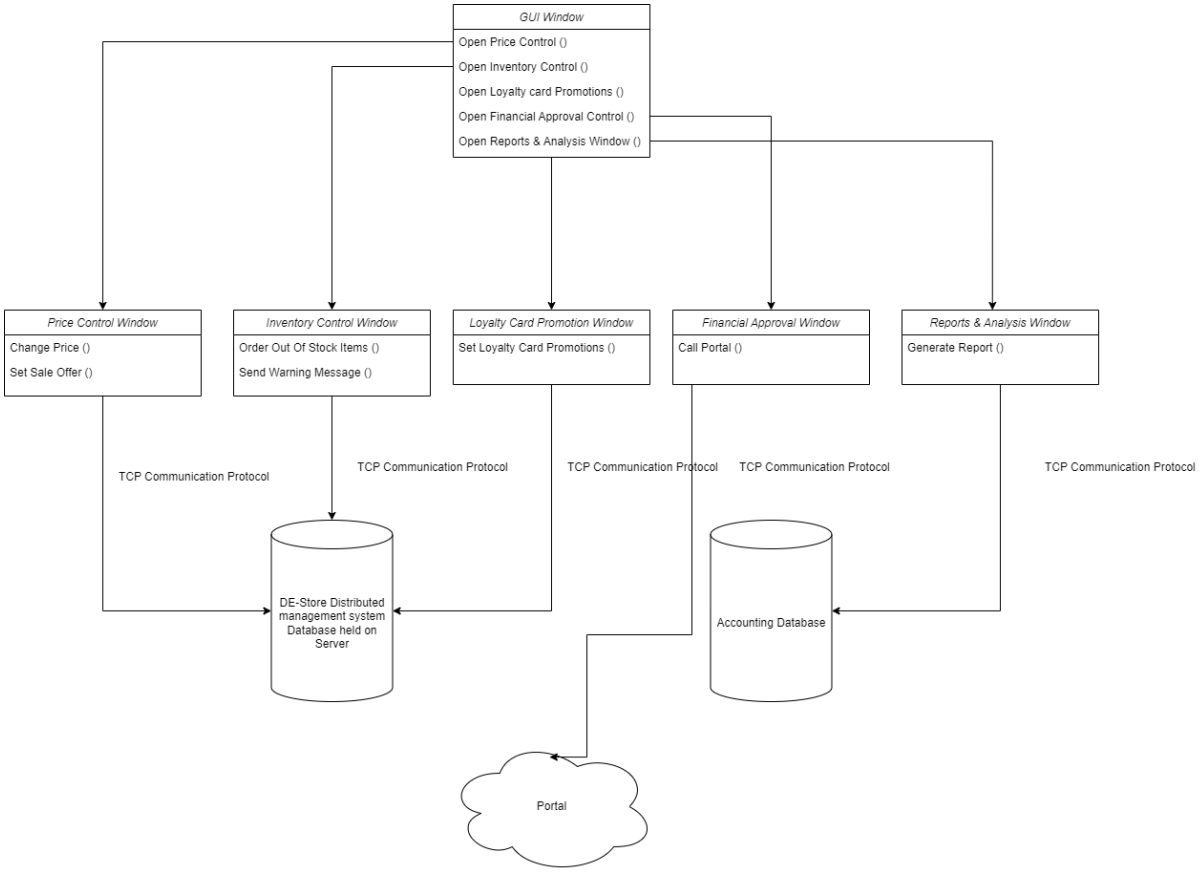
Plans to design a prototype that displays the key functionality requested by DE-Store along with displaying the quality attributes the client/server model offers have been designed. Implementation of a client/server model prototype of DE-Store's distributed management system using the python programming language have been developed. The prototype will be made up of a graphical user interface which will allow users to visualise what task they would like to complete. The graphical user

interface will be developed within the client component of the distributed management system as the client component handles all user interface functionality. The main functionality of the distributed management system will be held within the server component of the client/server model. The main functionality that will be held within the server component will be a price control functionality, inventory control functionality, Loyalty card functionality, financial approval functionality, and Reports and analysis functionality.

The client/server model communicates via network protocols. For this prototype the client and server communicate by IP addresses from local machines. This will display the client/server models communication protocols functionality. DE-Store's distributed management system will use a communication-oriented process which will use the TCP protocol which allows systems to communicate even if they use different types of network hardware.

The DE-Store distributed management system will hold a database on the server component as a centralised data management unit. Managers of DE-Store will be able to access the database via the graphical user interface that is held on the client component of the client/server model. Managers will be able to make changes to the database based of the requested functionality DE-Store would like implemented.

A class diagram that creates a clear preview of how DE-Store's distributed management system will be laid out.



(Figure 4: Class Diagram of DE-Store's Distributed Management System Using the Client/Server Architecture Style)

As seen from the class diagram in figure 4 plans have been designed to implement multiple GUI windows for users to complete each function. For example, the main GUI window will give users the option to select which function they want to use within the distributed management system. Each function will have its own GUI window which will allow users to make changes to the database based on the function they have selected. The price control, inventory control, and loyalty card promotion functions are all able to make changes to the store database via the TCP communication protocol over a network connection. For this prototype the database will hold test data such as example inventory levels with example prices. These test stock levels will show functionality such as sending warning messages for low stock and sending order requests for inventory that is out of stock. The price control function will allow users to make changes to the prices of inventory items held within the database. Along with changing prices users will also be able to add special promotions to certain items. The Loyalty card promotion function will have a similar affect by applying special promotions to items for customers who own a loyalty card.

These three functions will affect the DE-Store database held within the server component of the client/server model. The financial approval function allows users to check the ‘Enabling’ financial system to see customers who are buying now and paying later. For this prototype I will assume that DE-Store has already implemented the ‘Enabling’ financial system and will only display users accessing the financial system via a portal. The portal will be a web-based platform that displays customer test data for users to access. Finally, Users will be able to access automatically generated reports and analysis of DE-Store’s performance as a store. DE-Store will receive these reports from a different database held within the DE-Store server. This second database will be the accounting database which will be used to track the purchase activities of customers and generate reports for users. For this prototype example customer purchase data will be added to the accounting database to display a generate report.

Overall, the proposal for DE-Store’s distributed management system to utilise the client/server architecture style shows a clear understanding of the client/server architecture style as well as the requested functionality of DE-Store’s distributed management system. Now that a clear proposal has been presented a thorough prototype of DE-Store’s distributed management system can now be implemented.

6. Evaluation

After the implementation of the client/server architecture style into DE-Store’s distributed management system, various results have been concluded. At the current stage the distributed management system is capable of handling all requested functionality while utilising the client/server architecture. Price control, inventory control, loyalty card control, financial approval, and reports and analysis functionality have all been implemented within the prototype. Users can launch the server program and connect via the client program to launch the distributed management system. The client program is displayed via a GUI, in which the user can perform several different requests to modify and retrieve data from the server and databases. The clients request are communicated to the server via TCP. The next steps of this prototype will be to analyse how the program and this architecture style handles different types of scenarios for future development.

One such area of discussion will be to analyse how the program and architecture style handles an increase in scalability. As the number of DE-Store business are developed, the distributed management system will need to handle more request as more users join the system. The client/server architecture will be able to handle an increase in users and request. However, a number of changes may be required to allow for more scalability. For example, upgraded or additional servers may be required to handle an increase in scalability. Multiple servers may be required to run multiple stores.

Another key area to discuss when developing the DE-Store distributed management system will be flexibility. Services may require modification in the future development of the system. New services may be needed all together. One suggestion to handle this problem in the client/server architecture style would be the implementation of the broker architecture pattern. The broker architecture patterns help in a number of key areas such as flexibility and maintainability. The broker architecture pattern works by forming a ‘deal’ between client and server. This allows clients to pass request to the broker which locates the server and forwards the request which then transmits the results and exceptions back to the client. This uses the same communication protocols that a solid connection between the client and server use, such as TCP. This allows further flexibility in the client/server architecture style as the client does not always need to know where the server is located. This allows new services to be implemented without affecting the overall design of the program and architecture style.

Interoperability is another key factor to discuss in the future development of DE-Store’s distributed management system. Interoperability discusses how well the system handles working with different technologies and platforms by integrating various services. The client/server architecture style offers systems the ability to utilise new services by separating client and server. New technologies can be implemented into the server without affecting the client and vice versa due to them being individual components. This helps with further development of the DE-Store distributed management system when new features are implemented within the system.

Finally, performance of the DE-Store distributed management system which uses a client/server architecture style must be focussed upon. The client/server architecture style is able to handle a larger number of requests from multiple clients at the same time. However, there are drawbacks to this benefit such as unreliable communications between client and server. Unreliable communications could host a number of errors for the systems integrity and performance. If the server becomes congested with too many requests or too many unreliable communications access to the server by the client may become inaccessible or very slow as the result would be a bottle neck situation as the server tries to handle multiple requests and errors at the one time. Stable network connections and properly handled request will lead to a high performing system but when these factors fail the system is vulnerable to a large loss in performance and a complete crash in all operations.

A SAAM developed approach has been illustrated to show the drawbacks and benefits between the client/server architecture style and service-oriented architecture style in the implementation of the DE-Store distributed management system with several scenarios in mind.

Architecture	Scenario 1: Adding a new feature	Scenario 2: Increase in request and operations	Scenario 3: Integration of new finance system	Overall
Client/Server	Implementation: Modify server to include new required feature. Update client to display data from server relevant to new feature. Impact: Modularity requires both	Implementation: Scale up server to be able to handle more request and operations via upgraded hardware. Optimise communications between	Implementation: Modify server to implement the finance system. Update client to utilise new finance function. Impact: Changes impact both	Overall the client/server architecture style offers a number of benefits such as simplicity allowing for better modifiability and centralised

	components to be updated separately Allowing changes to stay separate	client and server. Impact: Scalability is handled correctly but may require expensive upgrades.	client and server components would need to be modified	control. The client/server architecture style also faces some drawbacks such as Modifiability and scalability.
Service-oriented	Implementation : Develop a new service to handle new feature. Update the client to interact with new service. Impact: The new service is independently deployable. No changes to already deployed services required.	Implementation : Deploy additional instances of existing services or introduce new services to handle increased load. Impact: Better scalability by distributing load across multiple services and servers.	Implementation : Develop a new finance service and integrate it with existing service-oriented architecture. Update client to interact with new service.	The service-oriented architecture style offers a number of benefits such as scalability and interoperability . The service-oriented architecture style also comes with drawbacks such as complexity in service orchestration making it difficult to implement new features.

From this SAAM approach we can conclude that the client/service architecture style is the better suited architecture style for DE-Store's distributed management system. It is presumed that further development of the management system will focus on further functionality for different features to be implemented within the system. The client/server architecture style make it easier for developers to implement new features into the overall system compared to the service-oriented architecture style. The service-oriented architecture style is more complex when it comes to implementing new features in the overall system due to the service orchestration component built to the architecture.

In conclusion, the client/server architecture style has proven beneficial for the implementation of DE-Store's distributed management system. Many benefits have been evaluated with minimum drawbacks. The client/server architecture style leaves DE-Store the ability to further develop upon the given prototype. With consideration of the service-oriented architecture style has been evaluated the client/server architecture style offers more benefits for the type of distributed management system DE-Store has request to be developed.

References

Clements, P., et al. (2005). Documenting Software Architectures: Views and Beyond. Robert Glass, Editor-in-Chief, Journal of Systems and Software and Editor/Publisher, The Software Practitioner.

modassirali, syed, & singh, shubham. (2022, December 2). Client-server model. GeeksforGeeks. https://www.geeksforgeeks.org/client-server-model/

Gorman, B. (2023, February 23). TCP vs UDP: Differences between the protocols. TCP VS UDP: What's the difference and which protocol is better? https://www.avast.com/c-tcp-vs-udp-difference#:~:text=TCP%20vs%20UDP%3A%20Differences%20between%20the%20protocols,reliable%20but%20works%20more%20quickly.

Rehman, J. (2021, November 4). Advantages and disadvantages of Service Oriented Architecture (SOA). IT Release. https://www.itrelease.com/2018/10/advantages-and-disadvantages-of-service-oriented-architecture-soa/

Garg, S. (2023, January 10). Service-oriented architecture. GeeksforGeeks. https://www.geeksforgeeks.org/service-oriented-architecture/

Figures