E-Tendering System

1. Project overview

eTendering application is used to manage the Request For Proposal (RFP) life cycle including the management of the exchanged documents as well as the approvals required during the tender process. The process start when a department submits an RFP, then the latter goes through a series of reviews and approvals. If approved the RFP is published as a Tender to seek applications from external suppliers. Then the application manages the receipt of tender applications and associated documents. Then the system manages the selection process to decide the tender winner. The objective of eTendering application is to increase productivity during the tendering process by decreasing paper handling and speeding up communication and easing interactions. The ultimate goal is to shift from the current manual paper methods to a fully electronic and automated tendering system. One of the major strengths of the proposed eTendering system is the remote Web-enabled accessibility of the system. Thus making it possible for a tender manager, tender applicants or contractors to access the system functionality from anywhere without being impeded by geographical location constraints.

2. Architectural Patterns

2.1. Process Coordinator Pattern

Design Rationale

The Process Coordinator pattern is selected to manage the Request For Proposal (RFP) life cycle from RFP submission to tender closing. It manages the control flow and the data workflow between the processing stages of a RFP. Thus, It ensures that the sequence steps of RFP are followed.

The RFP workflow is depicted in Figure 1. Examples of the RFP process steps:

- When an RFP is submitted then notify the department manager and the Tender Committee members by email that an RFP from his/her department was submitted.
- If the RFP is approved then convert it to a Tender

The rationale for using this pattern is that the RFP workflow is subject to frequent changes. Hence externalizing the workflow definition will make it much easier to change and evolve. This will significantly enhance maintainability compared to embedding the workflow logic within the implementation of various components.

Table 1. Evaluation of the impact of the Process Coordinator pattern on the key software qualities

Quality Attribute	Impact (+ for positive and – for negative)
Availability	- The Process Coordinator (PC) introduces a single point of failure and it might become a bottleneck in case of high load. If the PC is down then the whole system is down. Hence a load balanced deployment or multiple process servers could be required. For our system this could acceptable since the current load is low and the users can accept up to one hour downtime to get the system back online. When the system usage grows this limitation could be addressed by replicating the PC to create a high availability solution. Alternatively a Message Queuing could be placed in from the PC to accept incoming message even if the PC is down. For example the RFP submissions could be placed in a queue and the PC can fetch them for processing. Incoming messages (such as RPF requests) can be stored on the MOM server until the PC is restored. This solution will also help protect the PC from sudden overload spikes. - Also this architecture introduce one extra component (i.e., Process Coordinator) that we need to purchase, deploy and maintain. However introducing this component is a worthwhile investment since it can be used to deploy other workflows belonging to other systems.
Failure handling	- Failure handling is complex, as it can occur at any stage in the workflow. Failure of a later step may require earlier steps to be undone. Hence handling failures needs careful design to ensure consistency. This is not an issue for our case since our workflow is simple. Also all our workflow operations can be retried without side effects. For example, in case setting the RFP status failed to update the database, this request can be retired without affecting the system's consistency.
Modifiability	+ Process modifiability is significantly enhanced because the process definition is externalized and can be easily maintained. The components used (such as the RFPRepository) can change their implementation without affecting the coordinator as long as the exposed interface remain unchanged. For example the RFPRepository can be change to use an Oracle database instead of MySQL database without affecting the PC. + This pattern also minimizes the dependences between the various components (e.g. RFP UI and RFP Repository) as they are not aware of each other. In other words, it enhances loose coupling. For example the UI components are unaffected by changes to the back end components as they only use the abstract interfaces exposed by the deployed workflow. + By using this pattern, we can externalize RFP process definition which in turns allow visualizing it as shown in Figure 1. From the graphical representation of the RPF workflow we can generate an executable process definition that can be deployed to a Workflow Management System. The workflow logic becomes no longer

	embedded in the implementation of backend components. Hence it gives us: (1) the flexibility in choosing the workflow management system to manage the execution of the RPF workflow (2) ease workflow changes and extensions (i.e., increase maintainability).
Performance	To achieve high performance, the coordinator must be able to handle multiple concurrent requests and manage the state of each as they progress through the process. We do not expect any performance issues because the current load is low. Also the current RFP workflow definition is simple and does not keep any state expect the current activity that a particular workflow instance has reached. As the load and the workflow complexity grows, the performance can be enhanced by adding more resources to the PC server and replicating its deployment.
Scalability	The coordinator can be replicated to scale the application both up and out. The Oracle BPEL Process Manager that we are planning to use has built in scaling capabilities without any changes to the workflow. The initial deployment will have one PC server. When the number of requests per hour exceeds 1000 requests per hour we can scaled out the PC server on a two machine cluster to handle the increased request load.

As discussed above easier maintainability is the biggest positive impact of applying this pattern to our system. Other software qualities could be negatively impacted but the impact can be mitigated by appropriate deployment actions as discussed in Table 1.

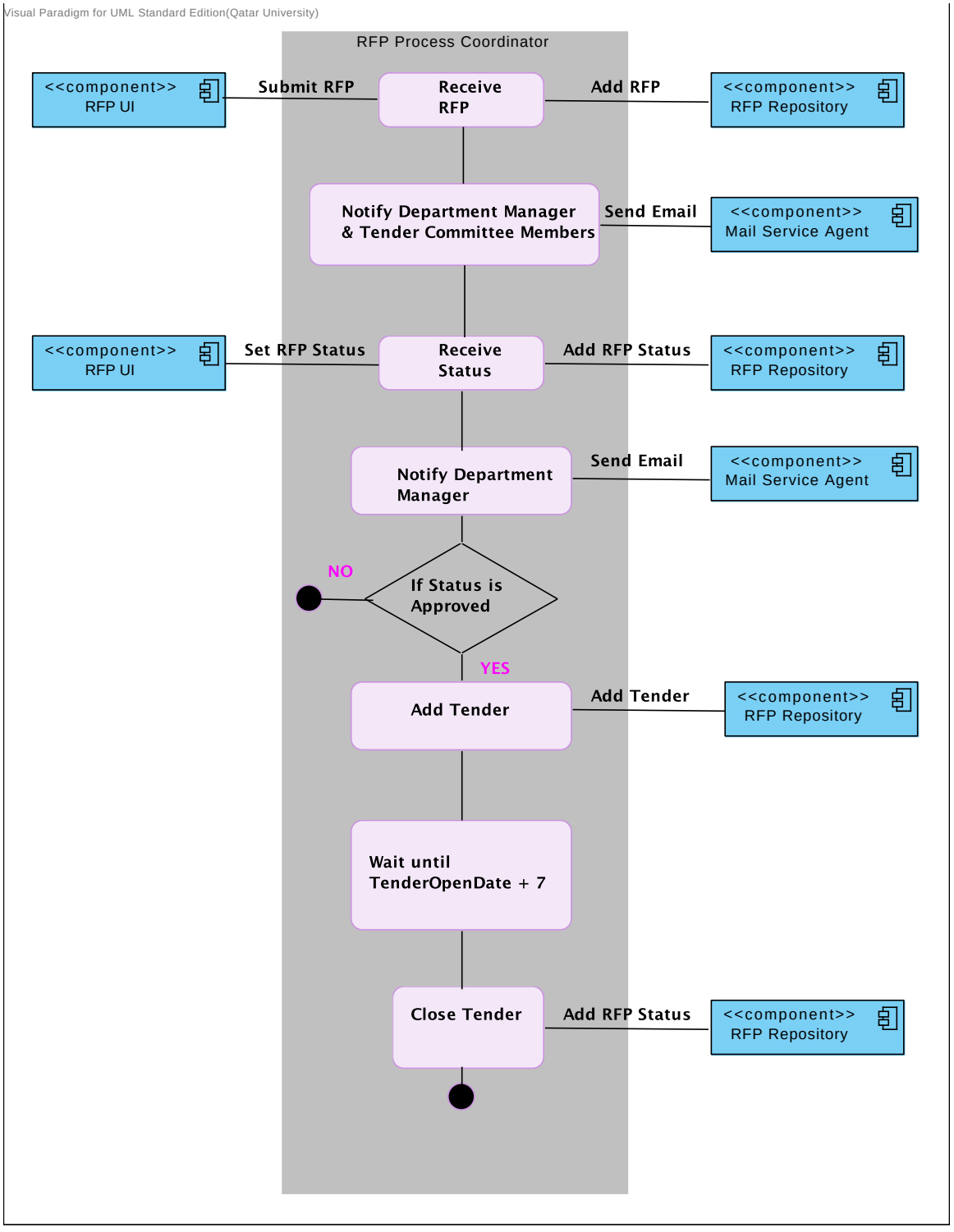


Figure 1. RFP workflow orchestrated by a Process Coordinator

2.2. Service Oriented Architecture Pattern (SOA)

We decided to use SOA for both consuming third party services (i.e., Payment Service, Budget Service and Mail Service) required to achieve our use cases. On the other hand SOA is also used to expose useful services from our implementation (i.e., RFP Statistics Service and RFP Process Service).

The design rationale motivating our choice are summarized as follows:

1- It is better to consume services such as the Payment Service and the Budget Service rather than implementing them ourselves. This translate into considerable savings of time and effort. If we had to implement the functionality of these services ourselves, we would have to maintain them and implement the necessary updates. So reusing existing services we free us from maintaining them since this responsibility is shifted to the service providers who take care of any necessary updates. Additionally, we do need to invest on any infrastructure to host the services and ensure their smooth operation.

2- Isolation from the changes of the implementation of consumed services. Such changes will not affect our system as long as the agreed upon interfaces are not changed. For example if the Mail service provider decides to use Microsoft Exchange instead of Lotus Domino as the backbone of the email service then this change should not affect our system and it will require no changes at our side as long the interface of the MailService remains unchanged.

3- Our organization uses a Dashboard Reporting system running in IBM Unix platform and implemented using Java and Python. Hence it is important that our provided Tender Statistics Service be accessible regardless of the running platform and the implementation language of the consumer. The Dashboard Reporting system will be the first consumer of our service but other systems can access this service in the future such as the EndofYear Reporing App.

4- A TenderService is also provided as a Web service to allow users to submit an RPF from within other applications such as the Assets Management System (AMS). This is a very valuable and welcomed feature since it will enhance the user experience. The users do not need to switch and copy-paste data between systems to create a RFP. They can do so within other applications they use on day to day basis. For example the users of AMS can create a RFP related to partial asset within the AMS system. Also relevant asset details could be easily attached the request (instead of manually entering such details in the case when the RFP is created using the native eTendering interface). Current we envisage that AMS will be the first consumer of the TenderService but other applications will make use of it in the future. For example the approval process of RPFs could be implemented as an Add-on to Microsoft Outlook. This way, upon receiving a notification managers can approve/reject RFPs from within Outlook.

Table 2. Evaluation of the impact of SOA on the key software qualities

Quality Attribute	**Impact (+ for positive and – for negative)**
Availability	- The Process Coordinator (PC) introduces a single point of failure and it might become a bottleneck in case of high load. If the PC is down then the whole system is down. Hence a load balanced deployment or multiple process servers could be required. For our system this could acceptable since the current load is low and the users can accept up to one hour downtime to get the system back online. When the system usage grows this limitation could be addressed by replicating the PC to create a high availability solution. Alternatively a Message Queuing could be placed in from the PC to accept incoming message even if the PC is down. For example the RFP submissions could be placed in a queue and the PC can fetch them for processing. Incoming

	messages (such as RPF requests) can be stored on the MOM server until the PC is restored. This solution will also help protect the PC from sudden overload spikes. - Also this architecture introduce one extra component (i.e., Process Coordinator) that we need to purchase, deploy and maintain. However introducing this component is a worthwhile investment since it can be used to deploy other workflows belonging to other systems.
Failure handling	- Failure handling is complex, as it can occur at any stage in the workflow. Failure of a later step may require earlier steps to be undone. Hence handling failures needs careful design to ensure consistency. This is not an issue for our case since our workflow is simple. Also all our workflow operations can be retried without side effects. For example, in case setting the RFP status failed to update the database, this request can be retired without affecting the system's consistency.
Modifiability	+ Maximize loose coupling: services can be easily changed/replaced without affecting the service consumers. + Process modifiability is significantly enhanced because the process definition is externalized and can be easily maintained. The components used (such as the RFPRepository) can change their implementation without affecting the coordinator as long as the exposed interface remain unchanged. For example the RFPRepository can be change to use an Oracle database instead of MySQL database without affecting the PC. + This pattern also minimizes the dependences between the various components (e.g. RFP UI and RFP Repository) as they are not aware of each other. In other words, it enhances loose coupling. For example the UI components are unaffected by changes to the back end components as they only use the abstract interfaces exposed by the deployed workflow. + By using this pattern, we can externalize RFP process definition which in turns allow visualizing it as shown in Figure 1. From the graphical representation of the RPF workflow we can generate an executable process definition that can be deployed to a Workflow Management System. The workflow logic becomes no longer embedded in the implementation of backend components. Hence it gives us: (1) the flexibility in choosing the workflow management system to manage the execution of the RPF workflow (2) ease workflow changes and extensions (i.e., increase maintainability).

Performance	- Performance could be negatively affected because of the increased size of exchanged messages between interacting services (e.g., messages exchanged between our system and the Budget Service). The exchanges messages uses XML which is verbose and not efficient data format. Sometimes the XML tags required to construct a message are much higher than the actual data contained within the message. This limitation could be alleviated by using more efficient data format such as JASON. - Higher network latency due to the distributed nature of Web services and the increased size of exchanges messages. For example the Budget service is not co-hosted with the eTendering system and any interaction incurs costly network traffic and higher latency. - Higher computing requirements particularly due to the need to serialize and deserialize exchanged messages. For example every time our system needs to call the Budget service, the sent messages need to be converted from in-memory objects XML and the received messaged need to converted from XML to objects. To achieve high performance, the coordinator must be able to handle multiple concurrent requests and manage the state of each as they progress through the process. We do not expect any performance issues because the current load is low. Also the current RFP workflow definition is simple and does not keep any state expect the current activity that a particular workflow instance has reached. As the load and the workflow complexity grows, the performance can be enhanced by adding more resources to the PC server and replicating its deployment.
Scalability	The coordinator can be replicated to scale the application both up and out. The Oracle BPEL Process Manager that we are planning to use has built in scaling capabilities without any changes to the workflow. The initial deployment will have one PC server. When the number of requests per hour exceeds 1000 requests per hour we can scaled out the PC server on a two machine cluster to handle the increased request load.
Reuse	+ Maximize reuse and ease of interoperability with other systems: SOA helps us implementg loosely coupled services such as TenderStatisticsService and Tender Service that separates the actual implementation from its interface. This service can be integrated and programmatically reused by external systems (consumers) regardless of their implementation language or their platform. SOA is also useful when our system acts as a consumer. For example our system talks to the Payment Service without knowing the details its implementation, it only knows its interface.

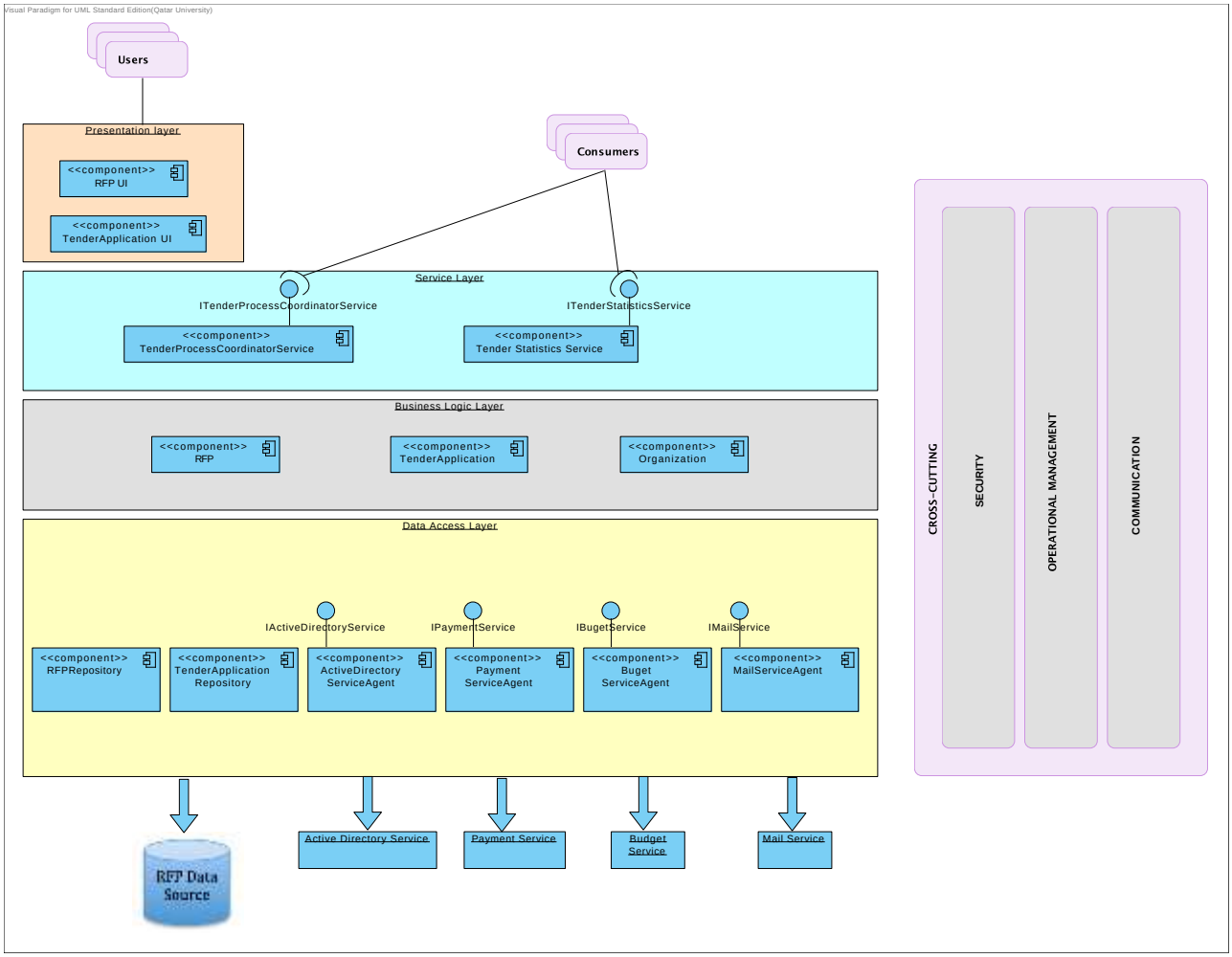
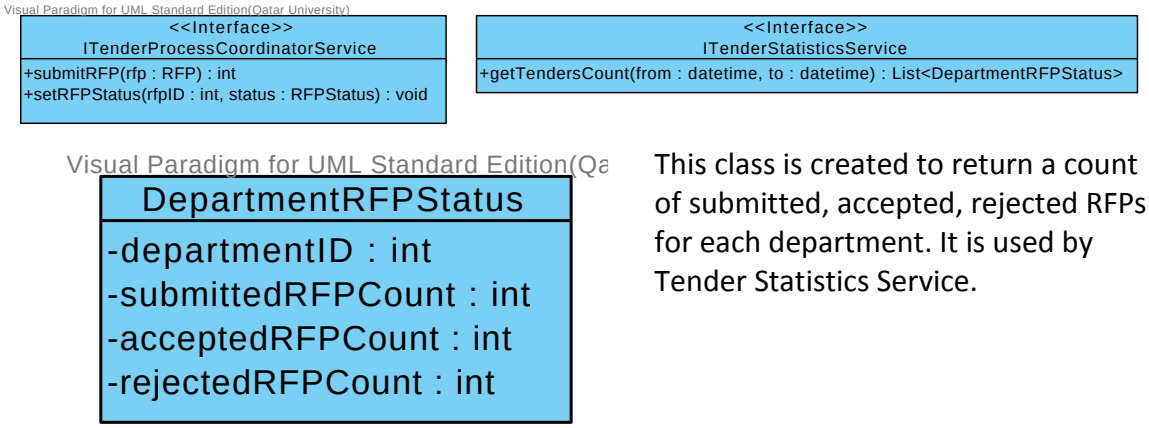


Figure 2. eTendering layered architecture augmented with the Service Layer and the Service Agents

The interfaces of the services offered by the eTendering are shown in Figure 3. While the interfaces of consumed services are shown in Figure 4.



This class is created to return a count of submitted, accepted, rejected RFPs for each department. It is used by Tender Statistics Service.

Figure 3. Interfaces of provided services

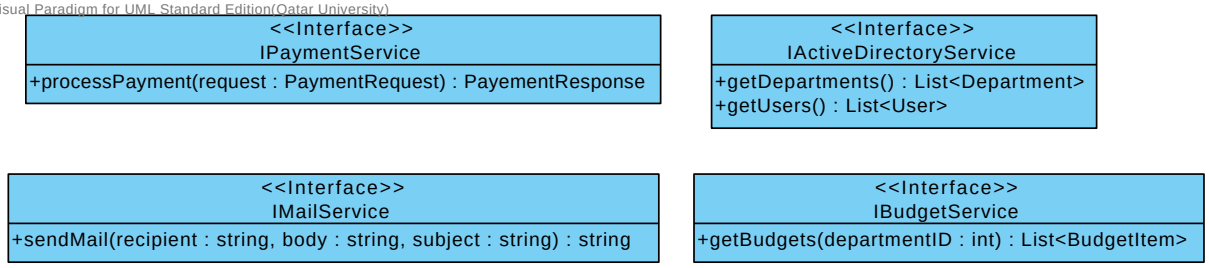


Figure 4. Interfaces of consumed services

3. Design Patterns

3.1. Proxy Pattern

We applied the proxy design pattern because it hides the communication details needed to talk to the real subject and it provides the flexibility to add any functionality before or after accessing an object in such a way that the client object does not know that it is talking to a different object. In eTendering system, proxy design pattern is applied to control the access to the budget service. The RFP controller (the client) instantiates an object of the proxy (BudgetServiceProxy) to get a list of budgets of a department (getBudgets).
Here, The RFP controller just does the invoking, and the proxy is responsible for:
1- Initiate the communication to the budget service
2- Do the required pre-processing. In our case, it authenticates the user, if the role of the user is not a budget viewer, the access will be denied.
3-Delegate the request (getBudets()) to the real subject (BudgetService)
4-And/or do post operations such as loggings.

Advantages

- BudgetServiceProxy hides complex protocol details of accessing the budget service (real subject) from the RFP controller (client). Thus, RFP controller accesses to the protected object only through proxy and the proxy keeps track of status and/or location of protected object. Therefore , we end up having an uncomplicated client (RFP controller).
- RFP Controller is not affected by migration of servers or changes in the networking infrastructure since location information & addressing functionality is managed by the proxy (BudgetServiceProxy).
- The proxy can be generated automatically by using the Add Service Reference

Disadvantages

- Since proxy pattern provides an interface for another system, there will be a possibility in some cases to deny the access for some reasons, so the proxy pattern is single point of failure.
- Less efficiency and overhead due to indirection

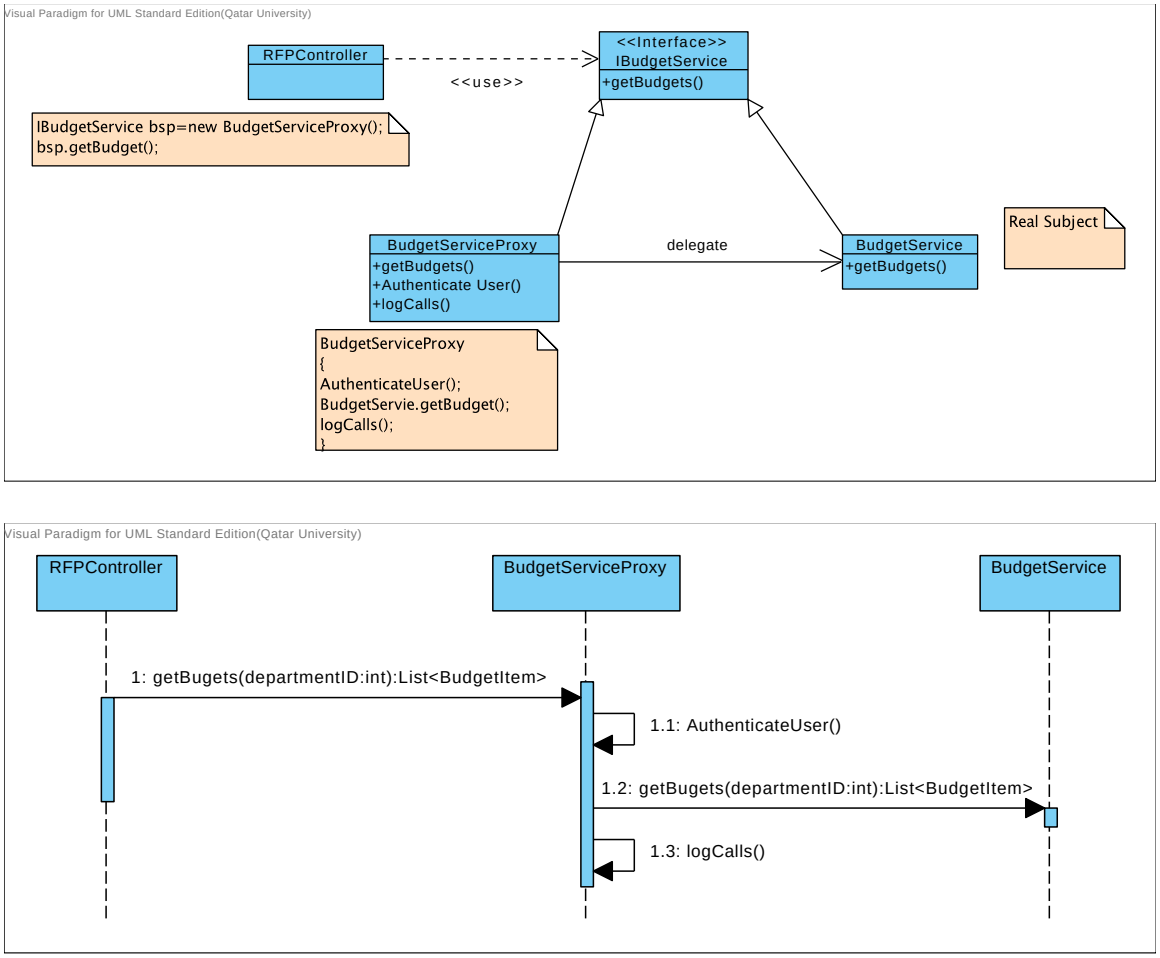


Figure 5. Application of the Proxy Pattern

3.2. Façade Pattern

In our system, active directory service is used to query for the department and user objects. However it's complex to work with active directory service interfaces directly from the client (RFP controller). Therefore, it is better to apply a design pattern that simplifies the complexity of active directory service interfaces. Façade pattern is the solution. By implementing a façade class, RFP Controller will need only to talk to the façade and the façade will have the complex implementations needed to access the active directory.

Advantages

First, it provides us with a high level interface to the active directory, leading to more uniform code and makes the active directory easy to use. Second, it isolates our system and provides a layer of protection from complexities in our active directory as they evolve. Third, this protection makes it easier to replace one subsystem with another because the dependencies are isolated. Fourth, building a façade component promotes reusability. In addition, façade increases maintainability and portability.

Disadvantages

- Since the façade is the only access point to the active directory , it will limit the flexibility and features the may be needed by power users.
- Facade adds an extra layer that does not do much apart from delegate to the actual service.

- Whenever an abstraction is layered on top of an existing abstraction it is possible to lose functionality

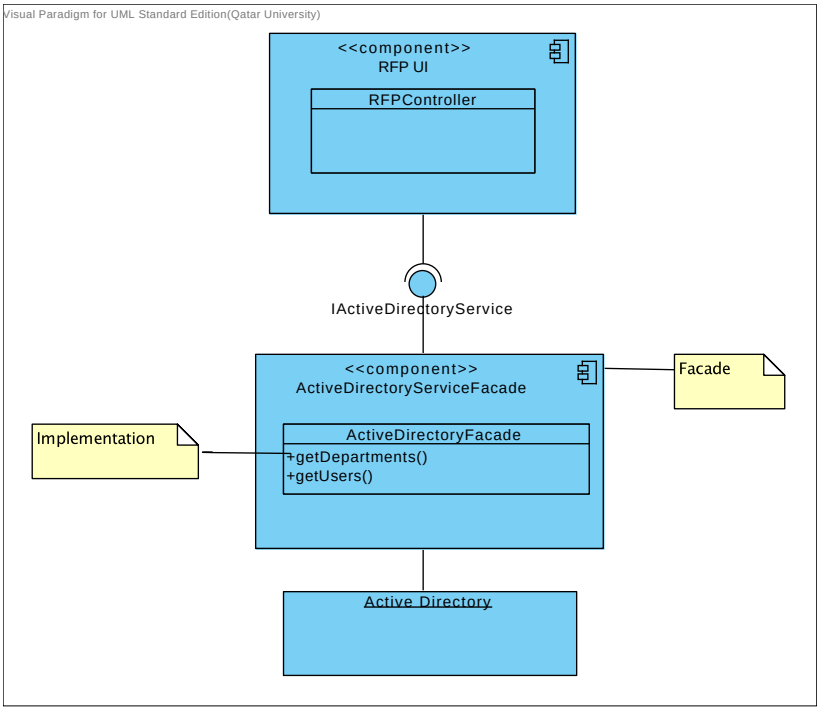


Figure 6. Application of the Facade Pattern

3.3. Observer Pattern

Following is the scenario of our problem. When a winner is selected, notify all applicants if they are rejected or being the one who is selected as the winner.

To implement this scenario, we need to use the observer design pattern. This pattern solves the problem of the difficulty to have all the applicants in the memory (no enough memory).

The steps we did to implement the observer design pattern:

1-Creating an interface named IObserver, this interface has the unimplemented update method (update (Tender tender)).

2-The subject is the RFPRepository class; it maintains a list of observers. It has two methods called subscribe and unsubscribe. Observers call these methods to register or unregister for updates. As a tender status is set to “Winner Selected”, it will iterate over the observers and invoke update () method for each. In our case we have only one observer explained in step 3.

3-Creating the observer class “Tender Results Notification”, this class implements IObserver. So it implements the update method. It subscribes itself to the RFPRepository to be notified when a winner is selected.

The update method does that:

First, it will retrieve all the tender applications for the tender. Second, for each tender application it will find the applicant's email. Third, it will send email to that applicant with his/her result (result= the tender application status)

Advantages

- Loose coupling: RFPRepository and TenderResultsNotification classes can be modified independently
- Simple and straightforward implementation.
- RFPRepository class is not bothered to implement the update code; it delegates the implementation to the observer (TenderResultsNotification).

Disadvantages

RFPRepository class can be reused only with the IObserver. This reduces the reusability of the RFPRepository in a totally different context. Another disadvantage is that the implementation is lengthy. There are at least 2 classes involved.

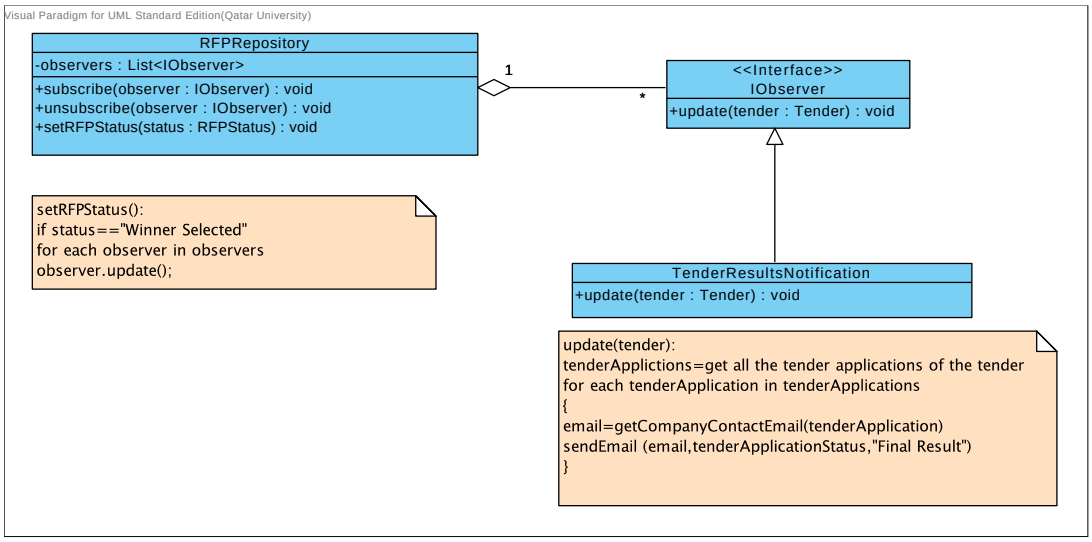


Figure 7. Application of the Observer Pattern