Architect Enterprise Applications with Java EE

Activity Guide (Solutions)

D68136GC10
Edition 1.0
March 2013
D81157

Author

Tom McGinn, Joe Boulenouar, Joe Greenwald

Technical Contributors and Reviewers

Linda deMichiel, Ian Evans, Roberto Chinnici, William Markos, Shreechar Ganapaphy, Vishal Mehray

This book was published using: Oracle Tutor

Table of Contents

Preface

Profile

Before You Begin This Course

- Thorough knowledge of distributed computing and communication concepts, Java EE technologies, experience with notation languages such as UML
- Working experience with the analysis and design of object-oriented software systems

How This Course Is Organized

This is an instructor-led course featuring lectures and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills introduced.

Related Publications

Additional Publications

- System release bulletins
- Installation and user’s guides
- *Read-me* files
- International Oracle User’s Group (IOUG) articles
- *Oracle Magazine*

Practices for Lesson 1

Chapter 1

Practices for Lesson 1

Practices Overview

In these practices, you will test your understanding of Enterprise Architecture.

Practice 1-1: Exploring Enterprise Architecture

Fill in the blanks with the correct answers:

1. Using your site to answer this question, what are the three top tasks or responsibilities of the EA? EAA?

   Answer:

   Enterprise Architect (EA): Identify the functional requirements, the non-functional requirements, and the use cases.
   Enterprise Application Architect (EAA): Identify the refined use cases, domain model, and design model.

2. Be prepared to give an "elevator speech" defining what Enterprise Architecture is and the need for it.

   Answer:

   In recent years two crucial changes have occurred in software engineering. The first is a requirement for a system to scale. No longer can system count on a static of fixed number of users. Systems must be able to grow dynamically without an impact to the overall performance or availability of the system, even at peak load.

   The second change is that software applications and components are now distributed across multiple servers. The monolithic, single systems of the past have given way to smaller, hot-swappable, rack-mounted servers. These flexible systems allow Web applications, business applications and resources to be distributed across multiple systems, which increases the availability and performance of services and reduces single points of failure.

3. Which of the following are likely Enterprise Architect or Enterprise Application Architect functions or tasks? Why or Why not?
    a. Database migration and tuning
    b. Liaison between system designers and development team
    c. Lead facilitated workshops to derive functional requirements
    d. Model aspects of the application architecture
    e. Propose design and implementation recommendations for hardware and server software
    f. Determine interfaces for distributed services
    g. Choose the ORM solution provider
    h. Ensure IT goals and direction align with the business goals
    i. Participate in social functions with the CIO

   Answer:

   b, c, d, e, f

Practices for Lesson 2

Chapter 2

Practices for Lesson 2

Practices Overview

In these practices, you will model the initial architecture.

Practice 1-1: Exploring Enterprise Architecture

Business Background:

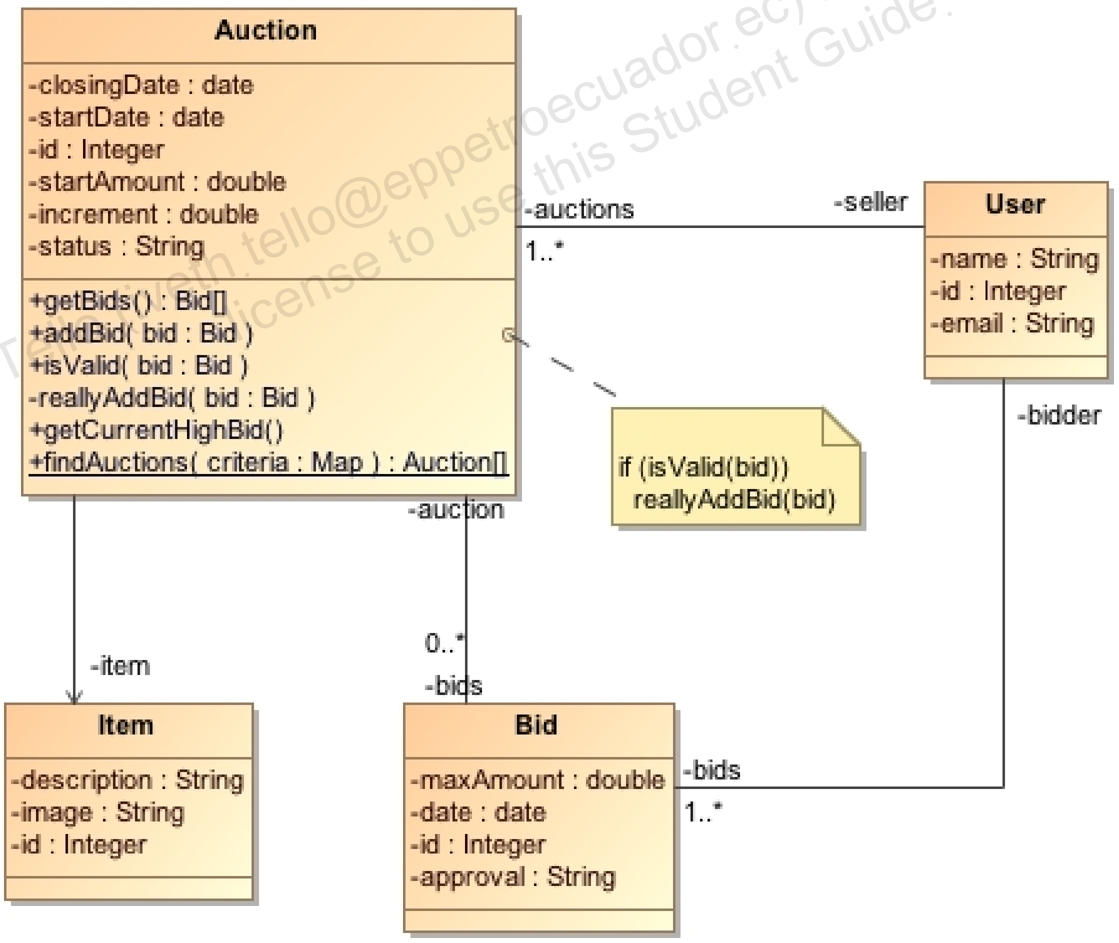
Your company is a leading on-line retailer that has decided to expand into the Auction business. The business analysts have delivered to you the requirements as a starter set of diagrams: a basic Domain model and set of Use Case descriptions (below).

For this exercise, you are to decompose the system into subsystems based on the provided requirements. As this is the first draft, these be done at a high level.

Per your site standards, as the Enterprise Application Architect, you are to draw up an initial set of UML diagrams for the architecture. You are to create a high level Component and Deployment diagram for a Java EE architecture.

For this first draft you should assume that you are deploying the Auction system as an enterprise application on a single Java EE server running on a single JVM implementation.

Domain Model



Use Cases

Login: All users of the Auction system must first log on to the system, before they can invoke other operations. To log on, users must authenticate themselves, for example, they could supply a user ID and a password. This use case is related to the addUser and findUser use cases.

Add User: Adds a new user to the system database. This use case is invoked by the login use case, when a new user accesses the Auction system for the first time. In addition to the user ID and password already supplied, the system asks the user for additional personal information: name, address (both location and email).

Find User: Finds an existing user in the system database. This use case is included by the login use case, when an existing user returns to the Auction system.

Create Auction: Sellers are allowed to create auctions, to auction items off. Each auction holds information, such as the item being auctioned, the initial bid price, the date the auction starts and the date it will end (or how long the auction will last), and a list of all bids placed on it.

Place Bid: Bidders are allowed to bid on auctions. Given an auction, the bidder is allowed to enter a bid on that auction. The Auction system should not allow a bidder to place a bid on an auction that is lower than the current bid price for that auction. The Auction system should also allow the user to find out what the current bid price for the auction is, or even notify the user automatically when the current bid price for the auction changes, or when the bidder is no longer the current high bidder for the auction. The system keeps a list of all auctions a bidder is currently bidding on. The bidder can retrieve and view this list.

Find Auction: Users must be able to provide criteria to be used to retrieve a collection of auctions in which the user might be interested.

You are also given a number of nonfunctional requirements to keep in mind as you consider how to build this Auction system:

Portability: It has been requested that the Auction system be implemented as a web-based application in which the client side is straight HTML. This maximizes the likelihood that clients can interact with the Auction system, because almost every platform supports rendering of straight HTML.

Scalability: The application's runtime characteristics must remain acceptable as the number of clients, or requests per client, increases.

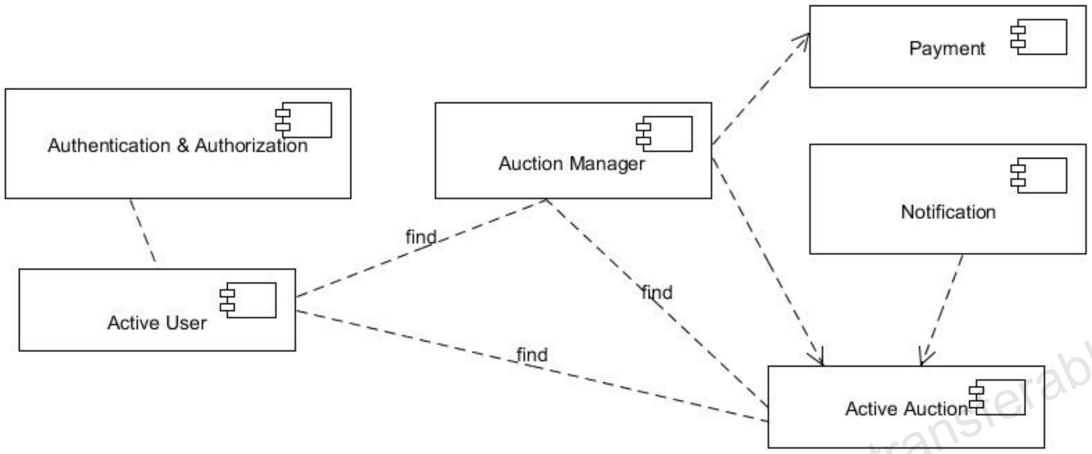
Performance: Response time when placing bids, or when selecting auctions to bid on, must remain short.

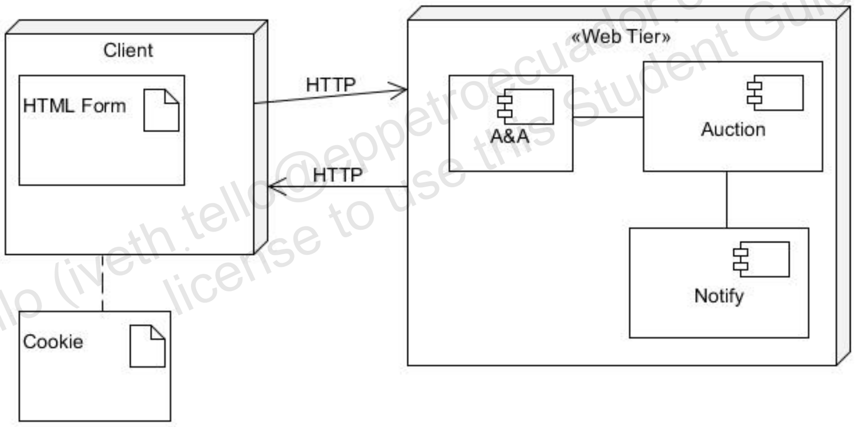
Tasks:

1. Identify the high-level subsystems (also referred to as services or components) that will be needed for a Java EE architecture and model these using a UML Component diagram. Using the Component diagram, draw the Deployment diagram showing where these components will be deployed.

Answer:

Architectural Diagram and Domain Model





2. Create the diagrams using one of the following approaches:
   a. Use the provided tool: UMLet to create the diagrams. Navigate to the Desktop folder and start the tool by double-clicking the **umlet** executable. Alternatively, open a terminal window and navigate to the Desktop and run the application using Java:

      ```
      cd D:\WINNT\Profiles\Administrator\Desktop\UMLet
      java -jar umlet.jar
      ```

   b. Use Flip Charts in the classroom or draw on the white board. This option is recommended when the classroom supports ample white board space.

Consider the UML models that you produced, and identify the risks that might be associated with that architecture.

Answer:

Risks include:

- Session management: the application must use either cookies or URL rewriting to maintain session state.
- Performance
- Scalability
- Reliability

Practices for Lesson 3

Chapter 3

Practices for Lesson 3

Practices Overview

In these practices, you will consider architectural choices for Security.

Practice 3-1: Describing Risks in the Auction system

All web-based, Internet-based applications must deal with security risks. Because the application is deployed over the Internet, the server side does not have control over the actual user interacting with the system, nor the application actually running on the client, nor the network that communications will travel on, between the client and server sides.

In this practice, you consider the following:

- The security risks with which an architect should be familiar
- The parts of the model of the Auction system that are vulnerable to these risks.
- The strategies you can employ to address these risks.

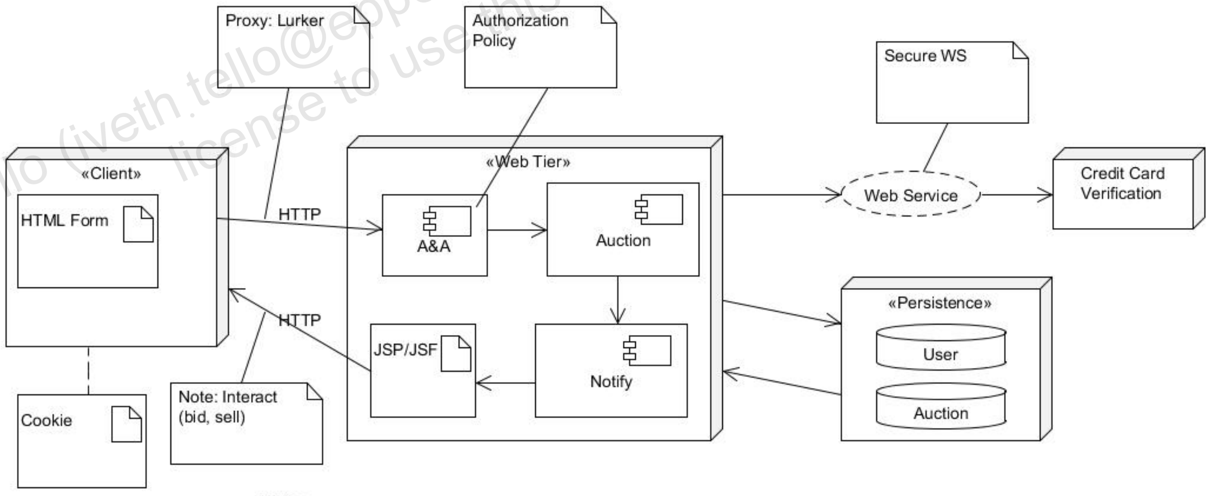
In Practice 2-1, you sketched a number of UML diagrams to describe a number of use cases for the Auction system.

Tasks:

1. Annotate your UML Component diagram with the security risks that are applicable to each use case, indicating where in the interaction the security risk could apply. Also, indicate which strategy you would introduce to address each risk, modifying the UML diagrams when needed.

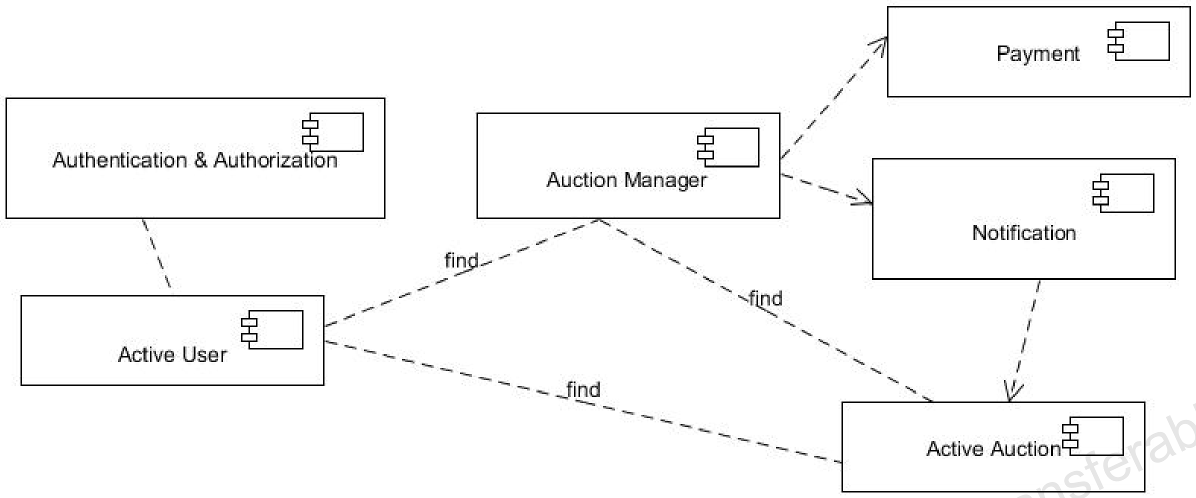
   Answer:

   Architectural Diagram:



Risks:
1. Must use cookies or URL rewritting.
2. Client communication unsecured.
3. A & A needs to be robust
4. Need policy for authorization
5. Secure WS w/ external credit systems

Domain Model:



Security risks include:

Potential security risks for Internet-based applications include:

- Code (SQL/Javascript) Injection

  Code injection is the exploitation of a computer bug that is caused by processing invalid data. Code injection can be used by an attacker to introduce (or ”inject”) code into a computer program to change the course of execution.

  Strategies to prevent code injection attacks include:

  – Input validation

  – Escaping dangerous characters

  – Coding practices which are not prone to Code Injection vulnerabilities, such as ”parameterized SQL queries” (also known as ”prepared statements”).

- Repudiation

  Repudiation consists of claiming that either the content of a request was corrupted, or the authentication of the sender was compromised and so the request should be invalidated.

  Strategies to prevent repudiation include:

  – Digital Signatures

  – Public/Private Cryptography with trusted certificate authorities

- Replay

  A “replay attack” is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it.

  Strategies to prevent replay attacks include:

  – Use of Session Tokens in each message

  – Use of nonces in each message

  – Use of timestamps in each message

  All of these strategies are variations on a scheme that allows the received to recognize a duplicate message.

- Man-In-The-Middle

  A “man-in-the-middle” (MITM) attack is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection when in fact the entire conversation is controlled by the attacker.

  Defenses against MITM attacks use authentication techniques that are based on:

  – Public key infrastructures

  – Stronger mutual authentication

  – Secret keys (high information entropy secrets)

  – Passwords (low information entropy secrets)

  – Other criteria, such as voice recognition or other biometrics

  – Off-the-Record Messaging for instant messaging

- Eavesdropping

  MITM attacks assume that the attacker wishes to affect the interaction between the two victim parties in the channel. However - the attacker may simply wish to “eavesdrop” on the channel, so as to “copy” (and presumably take advantage of) the information carried over the channel.

  Strategies to address eavesdroppers include:

  – channel-level encryption

  – message-level encryption

- Impersonation

  Impersonation attacks involve compromising the authentication scheme used between two parties, in order to allow the attacker to impersonate the first party, as far as an interaction with the second party is concerned.

  The same techniques used to address MITM attacks are relevant to stop impersonation attacks.

- Denial-Of-Service (DOS)

  A “denial-of-service attack” (DoS attack) or “distributed denial-of-service attack” (DDoS attack) is an attempt to make a computer resource unavailable to its intended user. (In the case of a DDoS attack, a (large) number of computers is used.)

  Strategies to address DoS attacks typically involve some kind of filtering of incoming traffic:

  – Firewalls can have rules such as to allow or deny protocols, ports or IP addresses, under given conditions

  – Switches can have some rate-limiting and ACL capabilities

  – Routers can have some rate-limiting and ACL capabilities, too - but they are typically simpler than what switches offer

  – Application front end hardware analyzes data packets as they enter the system, and then identifies them as priority, regular, or dangerous.
- The interaction between the client and the auction system requires some security mechanism - otherwise all data (credit card information, bid amounts, personal data) is sent in clear text and could be read by a lurking application or service. The client tier traffic can be secured using HTTPS (SSL, TLS).
- In the presentation tier, there is the risk of authentication and authorization. A clear policy must be established to determine what resources and services are available to each authenticated user. The strategy we use in this tier is JAAS (Java Authentication and Authorization Services) to interact with a robust authentication and authorization server (access manager server, identity manager.)
- Security between the business tier and the credit card service also needs to be addressed. The strategy used to solve this problem should include: the use of transport layer security, application layer security and message layer security.

Practices for Lesson 4

Chapter 4

Practices for Lesson 4

Practices Overview

In these practices, you will weigh capacity planning and performance options.

Practice 4-1: Architect for multiple tiers

The requirements for the system have increased:

- The current system, running inside a JVM on a single server, was sufficient for POC and initial startup, but the site has become popular and is no longer able to handle the capacity:
    - The number of concurrent users has risen sharply and is expected to increase at a rate of 10–25% per month.
    - Performance is noticeably degraded. For example: pages are taking a long time to display and many users are experiencing long wait times when submitting bids. Queries to search for auctions are taking many seconds to complete. Reports for managers are taking too long to run. Some bids have been lost or not recorded and some auctions have simply "disappeared' for no apparent reason. This has led to legal issues for your company.
    - Frequent downtimes and periods where the system is unavailable have been occurring. This needs to be resolved. Up times greater than 99.99% are needed. The business is experiencing significant revenue loss as a result.
- As popularity of the site increases, the business is realizing it will have to make changes to the application over time. The system should be architected in such a way as to make both application design and system performance and reliability easier to manage and expand over time.

Tasks:

1. Consider the topics discussed in the lesson.
    - What questions would you ask to further refine the business needs and requirements? What could you ask to help prioritize their QoS needs?
    - What options in terms of: hardware, software, design and architecture are available to you to meet and exceed these requirements and issues?
    - What are the constraints on your available choices?
    - How would you prioritize and what needs to be done in terms of architecture? What three things need to be addressed first? Plan the rollout of your choices and their impact on systems and people.
        - Hint: Use the matrix (Prioritization of QoS Requirements) presented in the lesson as a structure for organizing your decisions.

Answer:

In order to be able to establish a target performance, a baseline of the "acceptable" performance measured before the additional users were added must be created or agreed upon. Without a specific set of values, it will be impossible to reach an acceptable performance level.

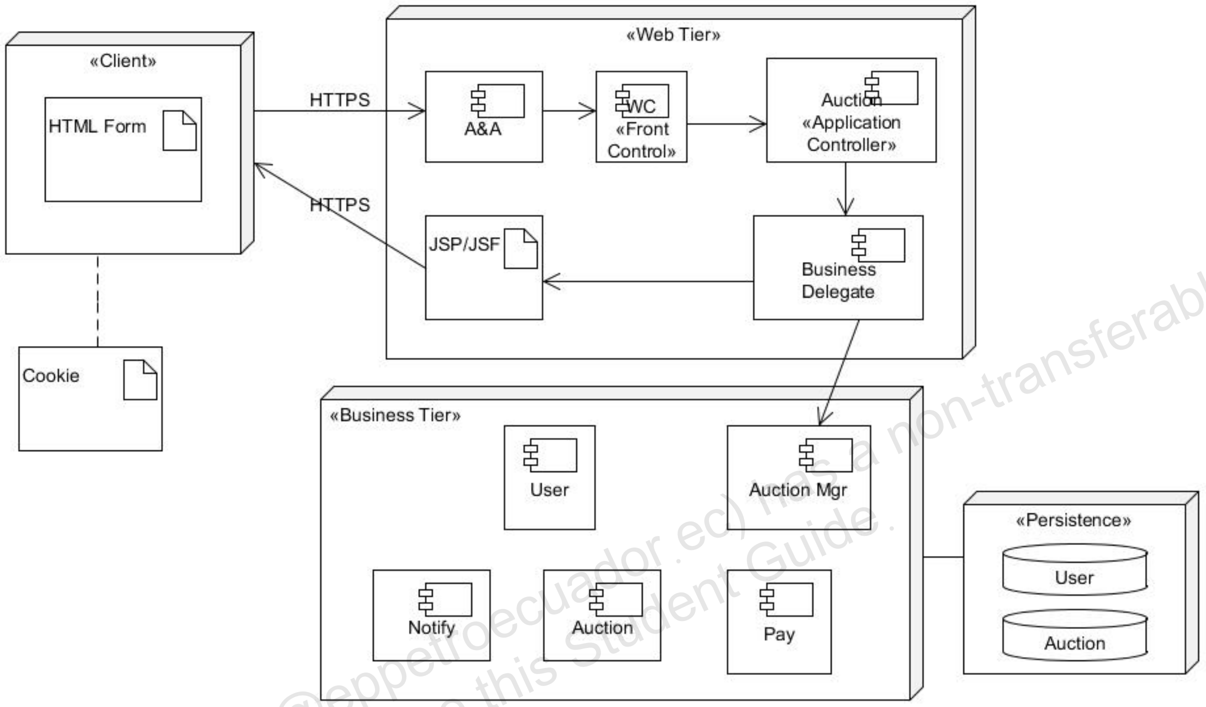
- Questions: What was the response time before when quality was acceptable? What is the specific maximum response time for placing a bid? What is the specific maximum response time for selecting an auction to bid on? What does a peak load of users look like?
- Tools like DTrace, profilers to measure the application performance, and load balancers, clustered application servers and clustered database o improve performance and reduce downtime. We need to provide a highly available database solution like Coherence.
- Constraints include cost (hardware cost, software cost, staff training cost, etc.)
- Three things that need to be addressed:
  - Improve scalability
  - Increase performance
  - Increase availability and reliability

2. Prepare a short (3 minutes or less) “sales pitch” to management to present your proposed architecture choices. Be sure to identify the benefits as well as potential risks and trade-offs. Also, describe the next steps you would take and why.

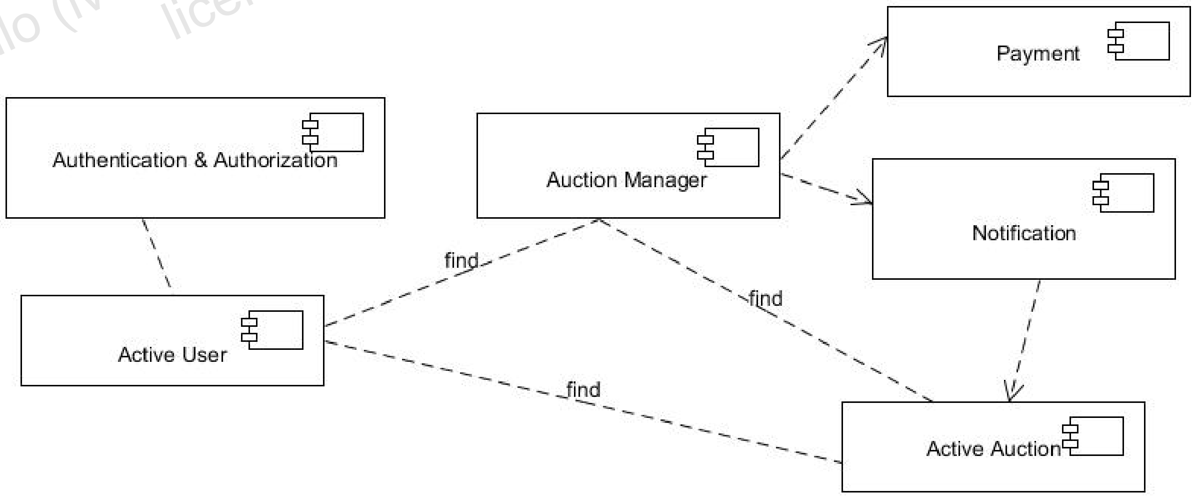
3. Update your logical UML diagrams to show the changes to your architecture as a result of your choices above.

   Answer:

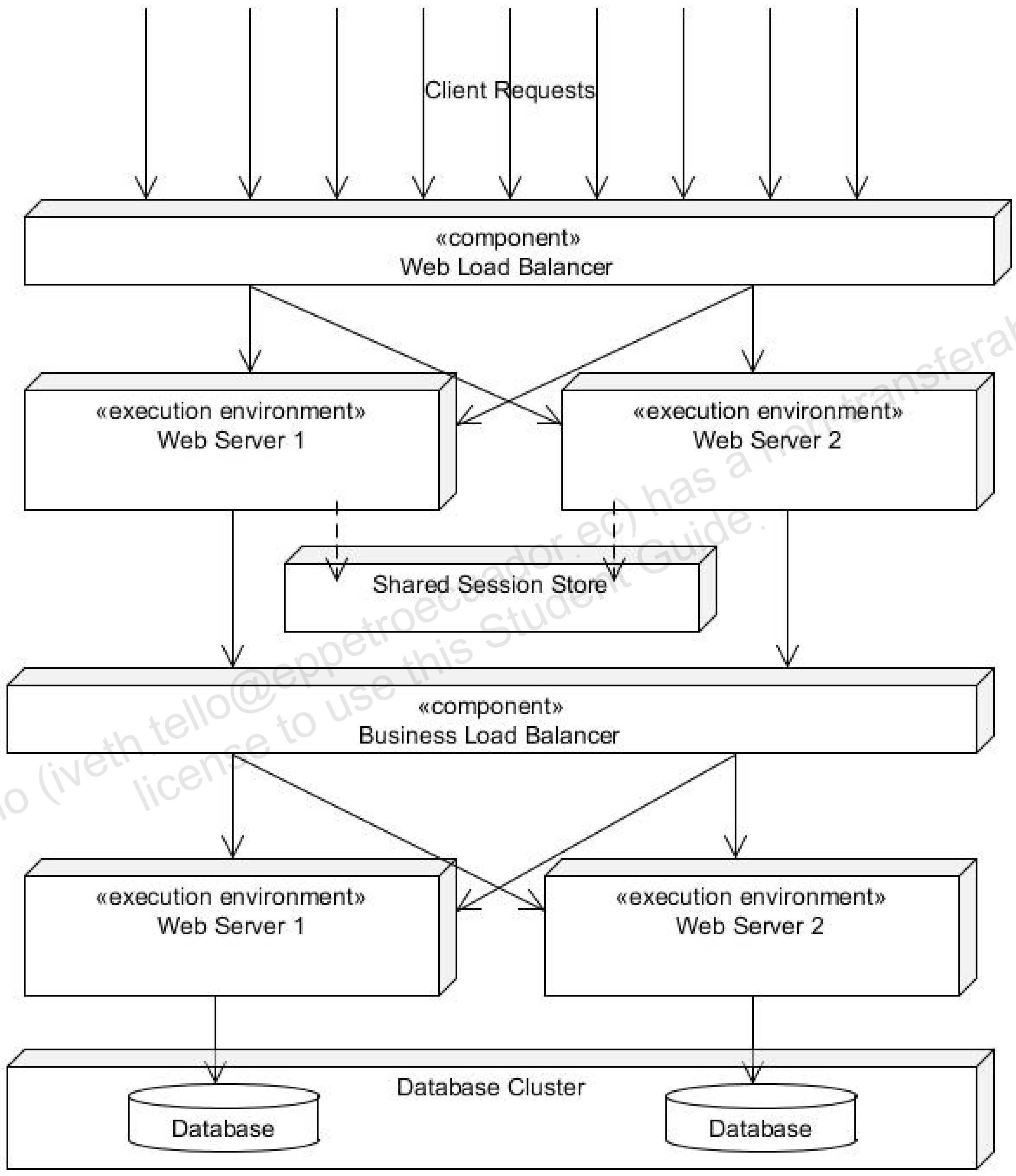
   Architectural Diagram:



   Domain Model:



Quality of Service Upgrades:



Practices for Lesson 5

Chapter 5

Practices for Lesson 5

Practices Overview

In these practices, you will modify your object model to increase flexibility and identify services.

Practice 5-1: Design a Flexible Object Model

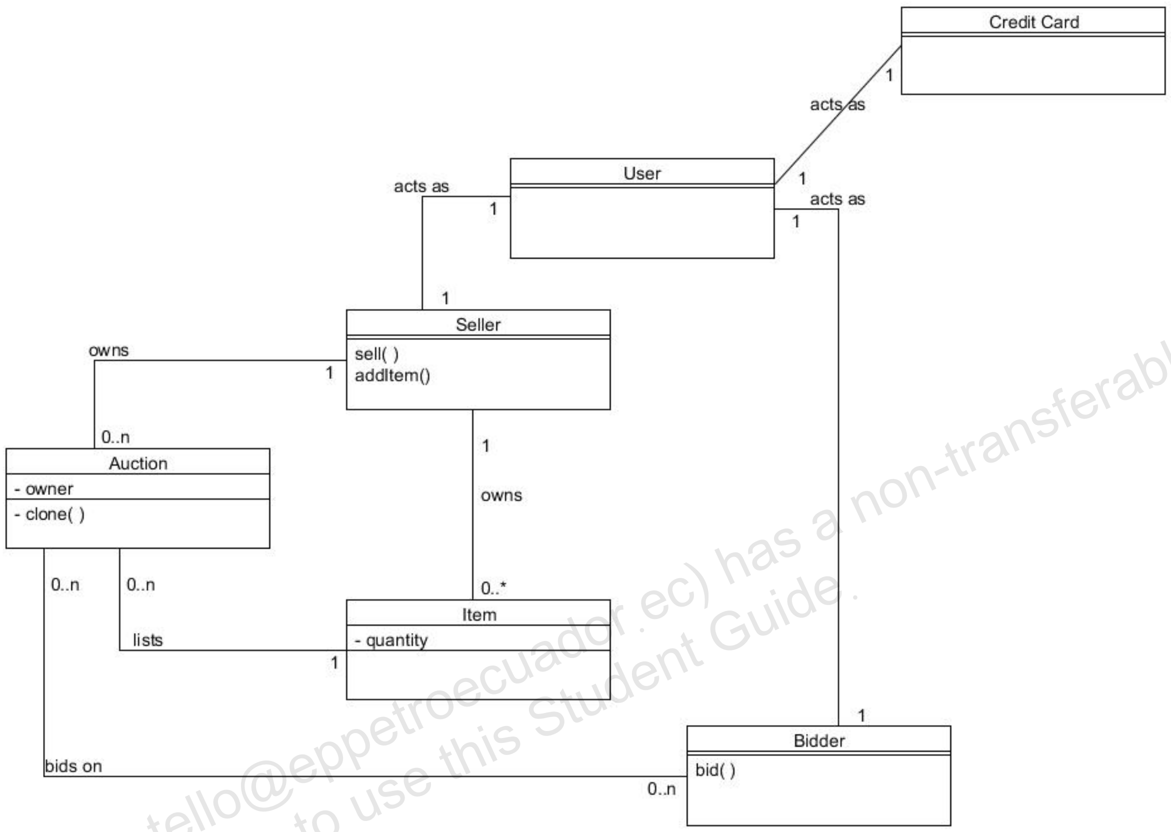
The business has given you additional requirements to add to your current architecture.

- It is important to distinguish between sellers and bidders because there are tasks in the Auction system that only one kind of user may request.
- New Use Case: **Pay Bid:** Bidders must be able to pay for their winning bid on an auction, once the action is over. The Auction system asks for credit card information at the time of payment. The system should also notify the seller of that auction, when the winning bidder actually pays for that auction. You will use an external clearinghouse payment processor.
- The user can provide credit card information to be used as a default method of payment. This should be saved so they do not have to reenter the information whenever they win an auction.
- New Use Case: **Add Item:** Sellers are allowed to create items to be auctioned off. The item to be auctioned and the auction for the item are considered to be independent from each other.
- The Auction system should also allow the seller to create multiple auctions for a number of copies of the same item conveniently. This can be achieved by allowing the seller to create a first auction, and then allowing the seller to create another just like it, or by allowing the seller to specify the number of copies of an auction to create, as part of the process of creating each auction.
- There is disagreement as to whether to use a database or LDAP server for user authentication. You should modify your architecture to easily account for either of these solutions later in the development.

Tasks:

1. Modify your Domain model to address these new requirements.

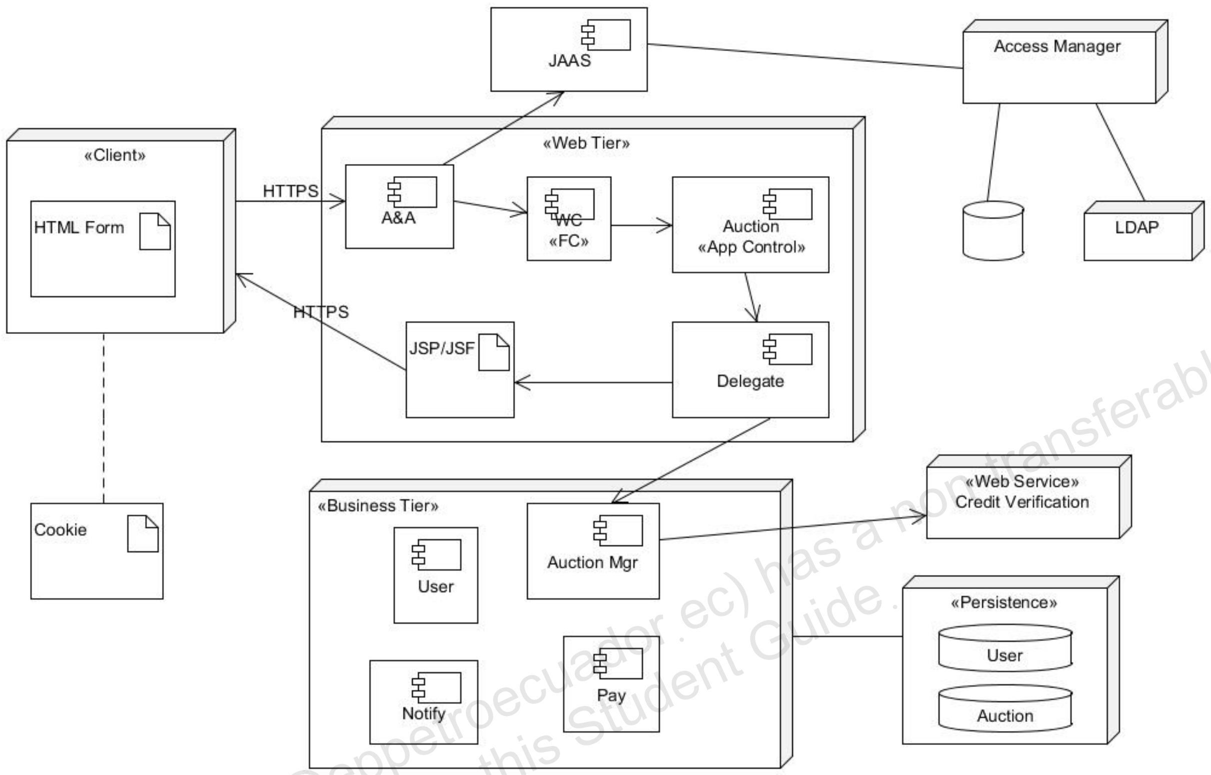
Answer:



Domain model: we modified the domain model to add the two structure classes Bidder and Seller which extends an abstract class User. The Bidder will be part of the payBid use case and Seller will be part of the addItem use case.

2. Modify your architecture to incorporate your changes. Identify and include new or changed subsystems or components to support this functionality.

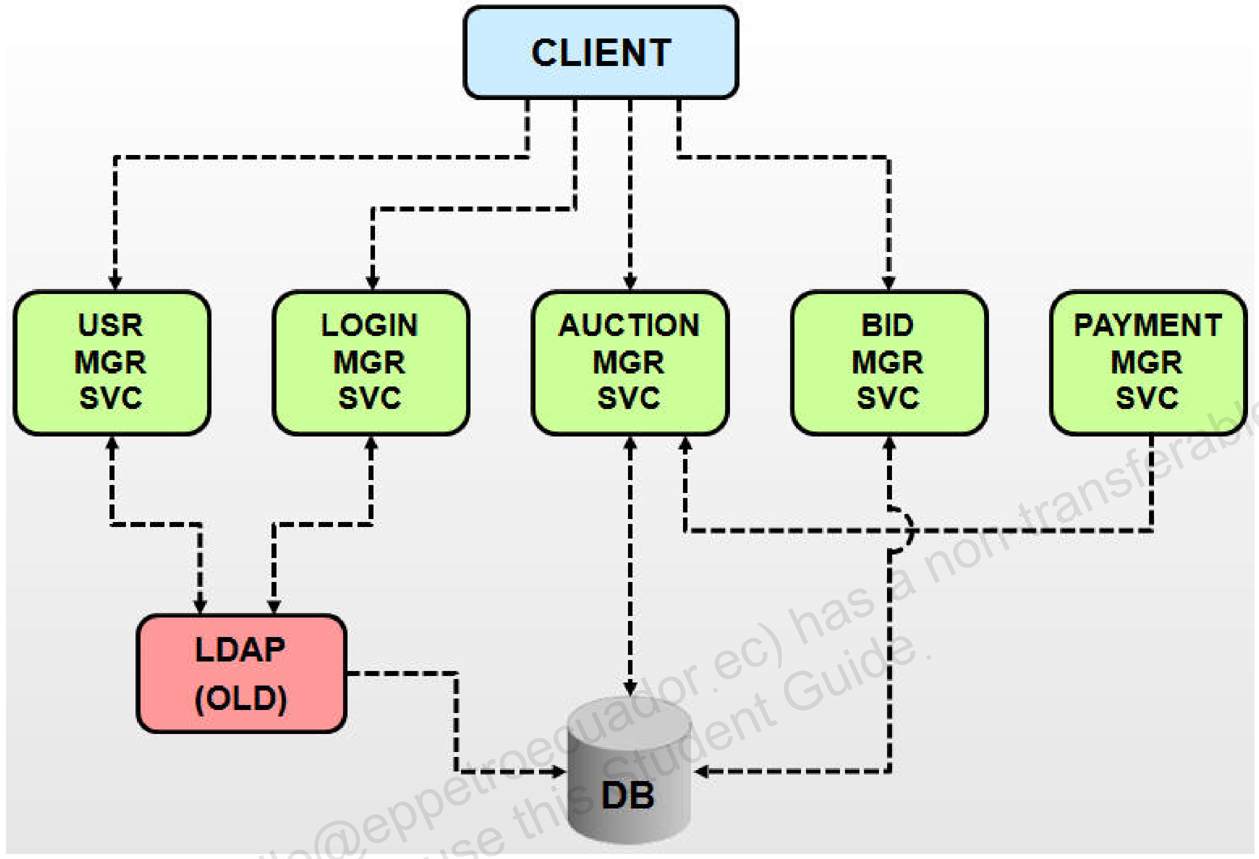
Answer:



In the architecture diagram, we added a JAAS interface from the A&A component to connect to an Access Manager service that can make use of a local database or and LDAP server for user A&E credentials. Note that we added this to satisfy the design as a flexible object model. We also added the user class in the business tier to represent the concrete classes Bidder or a Seller.

3. Make a first cut “guess” at coarse grained services that will be needed.

Answer:



We represent coarse grained services with the diagram. Note that fine grained services will be dealt with at the design level by using the robustness analysis methodology.

Practices for Lesson 6

Chapter 6

Practices for Lesson 6

Practices Overview

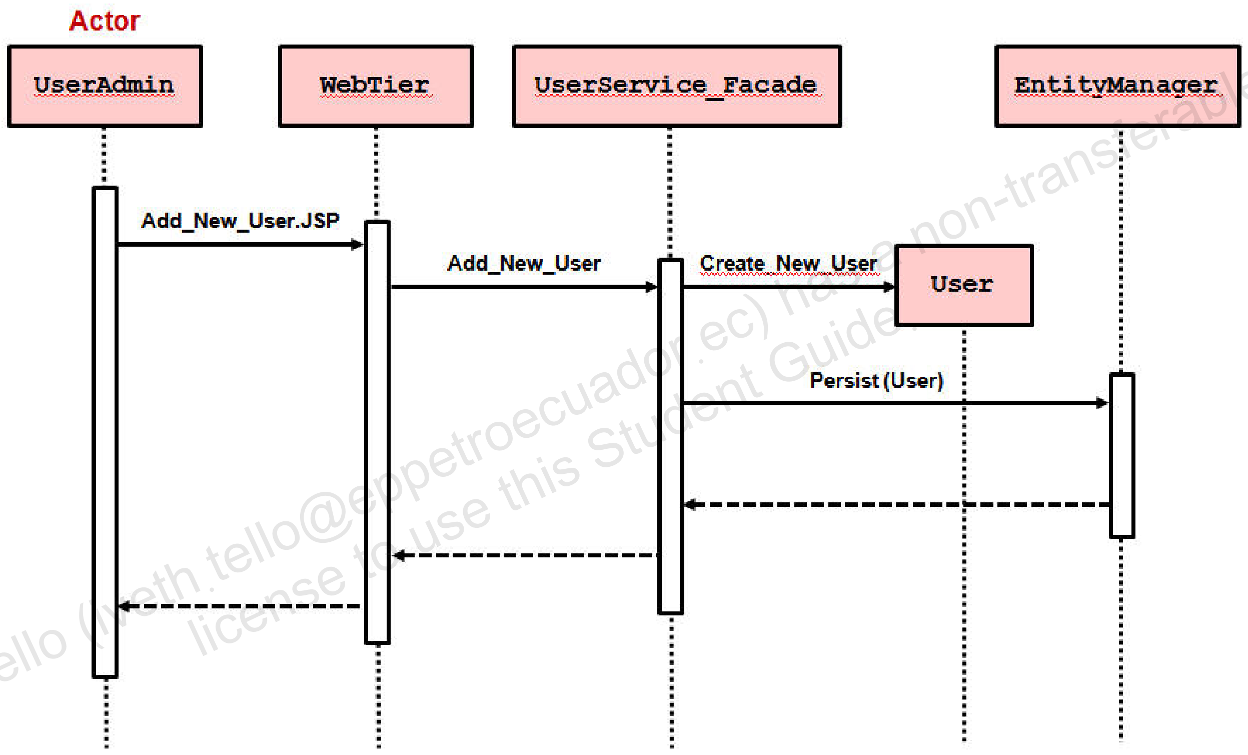
In these practices, you will plan for your transactions.

Practice 6-1: Transaction Planning

Tasks:

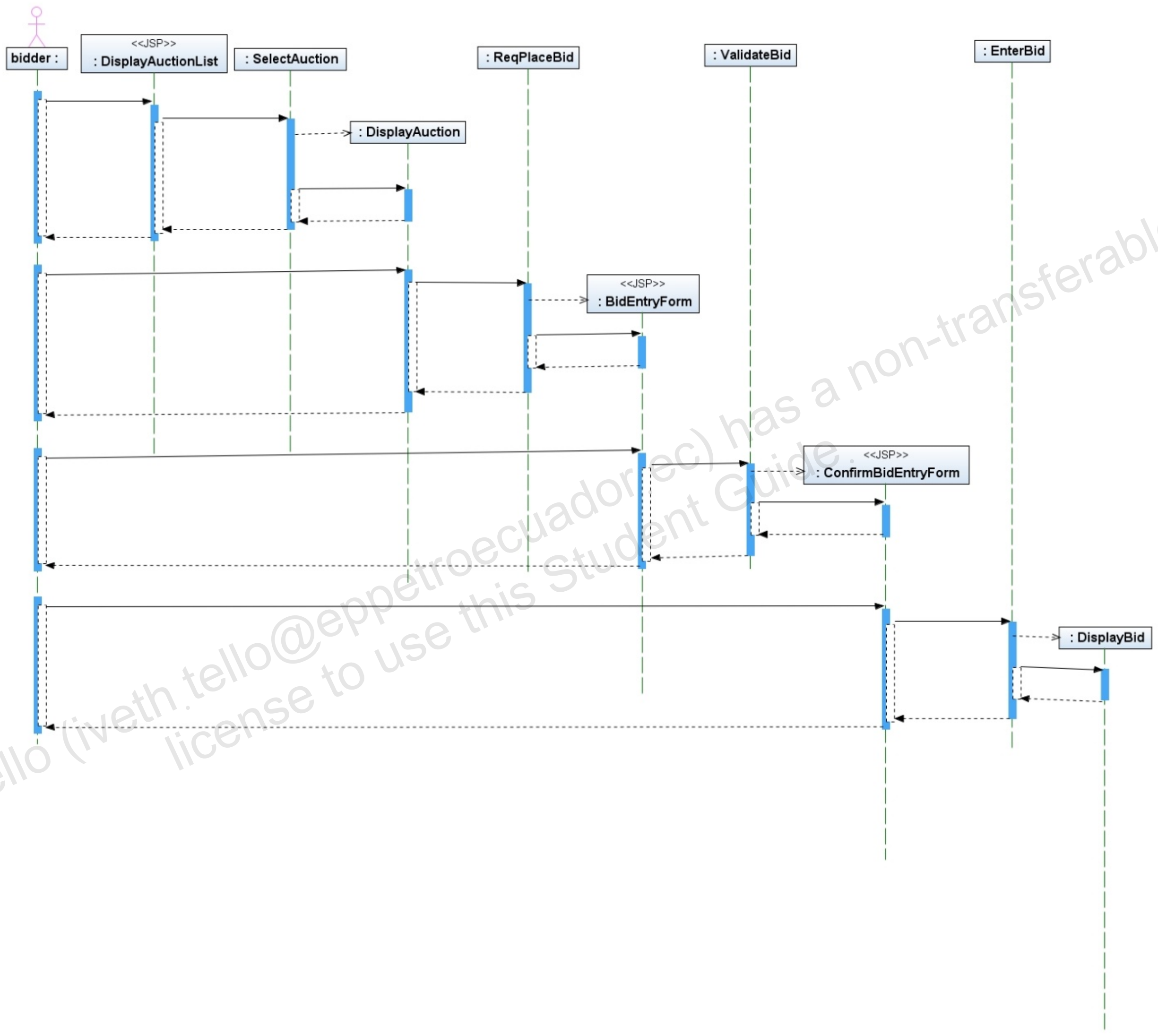
1. As a class, walk through documenting one use case's transaction with a Sequence diagram. Use the use case: Adding a New User.

Answer:



2. In pairs, document one transaction from the list provided below:
   a. Place Bid
   b. Create Auction
   c. Pay for Bid (**Note:** Consider the implication if you choose to offer the second highest bidder a chance to “win” the item if the highest bidder refuses to pay.)

Answer:



3. As a class, create a “prototype” bid system that does not use transactions. What impact does this have on your architecture? What are the risks associated with this approach? What are the benefits? Are there sufficient reasons to consider this as a viable approach? If so, what are they?

Answer:

Creating a new user without transactions or a transaction context will impact our architecture negatively. The risks associated with this approach include loss of integrity of the data because if the user was created but never persisted to the database, you could have a "phantom" user in the system until the application was reset.

The benefit of this approach is potentially higher performance; however the loss of data integrity could be a extremely detrimental and would cause business loss.

A bid system without transactions could disastrous for the business. Auctions are likely to have multiple simultaneous bids. Without a transaction system, there is great potential for lost updates - the last bidder wins (due to race conditions.)

4. As a class, analyze your transaction designs (1-3) with respect to possible service loss. What happens if there is a server crash and a bid is lost? Or an auction is lost? Or an auction cannot complete? Also, consider the effect of pessimistic versus optimistic database locking on your transaction.

Answer:

Without transactions, a loss of service in mid-flight of a bid transaction means that the data integrity of the system is suspect - there is a high probability that the last bid is lost and that the auction state is corrupt - either with the auction value changed without a bidder, or the loss of a bid altogether.

Pessimistic transaction locks require a serialization of process - therefore, depending upon the available granularity of the lock, in the worst-case scenario, only one bidder could place a bid at a time, negatively impacting system performance.

Optimistic transaction locks favor versioning on individual records, which does require the application to catch and recover from an optimistic lock exception - in general, this means that a bidder must repeat their bid, and at a higher value than the most current bid. This may lead to bidder frustration, but is ultimately the best way to preserve data integrity and maintain an acceptable level of performance overall.

Practices for Lesson 7

Chapter 7

Practices for Lesson 7

Practices Overview

In these practices, you will consider how you would modify your existing architecture to take advantage of the new features in Java EE 6.

Practice 7-1: Consider impact of Java EE 6 new features on your architecture

Java EE 6 introduces some significant new architectural features and options. These include (but are not limited to):

- EJBs and servlets can be called asynchronously.
- CDI and the new definition of beans eliminate the need for supporting, helper classes.
- Services can be exposed as JAX-WS or JAX-RS and implemented by POJOS and EJBs.

Tasks:

1. Each group should consider their existing architecture in light of these (and other) new features and be prepared to answer the following questions:
   a) What major and minor changes would you make to your existing architecture in light of these new features?
   b) What impact do these new features and capabilities have on design, application architecture, and enterprise architecture?

Answer:

The impact of using Java EE 6 on the web tier:

- We can now use server-push strategies to update client information dynamically.
- Context and Dependency Injection greatly reduces the code required to configure XML code and resource lookups.

The impact of using Java EE 6 on the business tier:

- Using asynchronous session beans in our business tier reduces the complexity of the design by eliminating the need to communicate asynchronously with external EIS systems using Message-driven beans.
- The use of CDI in the business tier, like in the web tier reduces the complexity of resource lookups.

The impact of using web services with JAX-RS:

- Performance is improved when we have two Java EE web service partners interacting with each other.
- RESTful web services require some type of documentation requiring the web service API when accessing a non-Java web service partner.

2. Assume your management is reluctant to migrate from their existing Java EE 5 platform to Java EE 6. Craft a short (3 minutes) presentation to convince management why they should consider upgrading to Java EE 6. One group should take the position: yes we should upgrade. Another group should take the counter position.

Answer:

Pro:

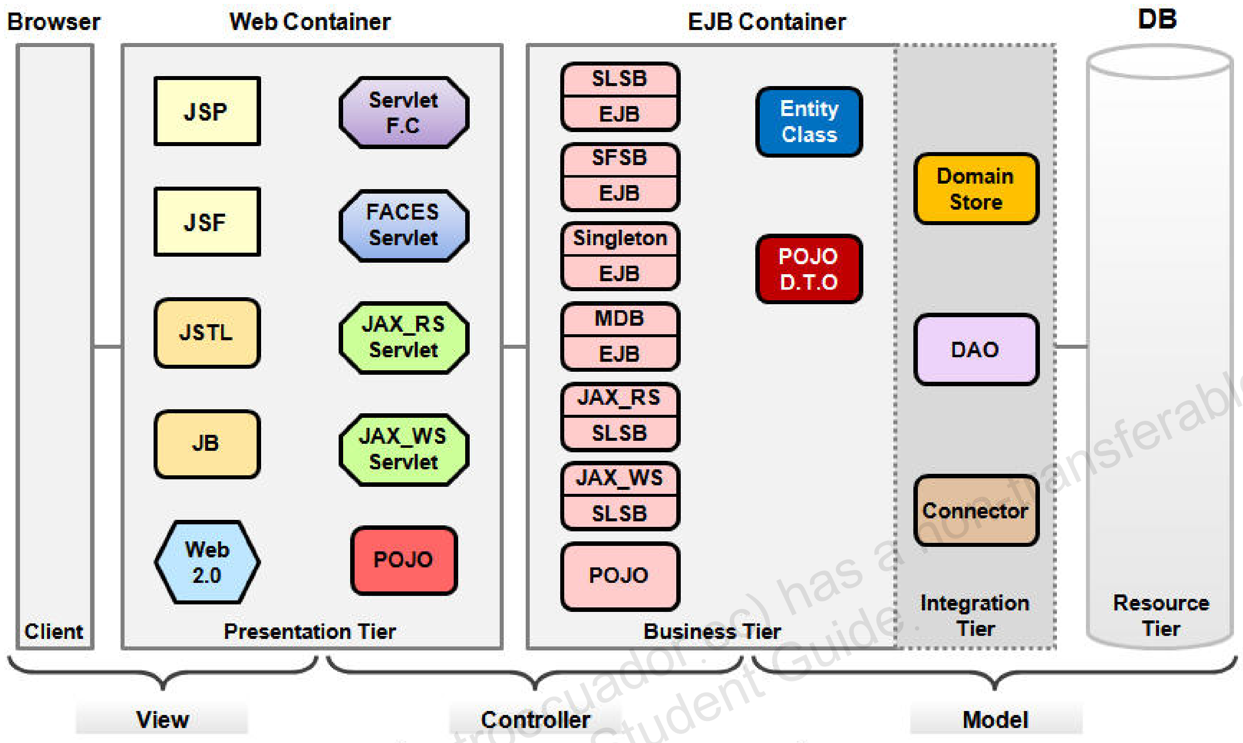
- Using Java EE 6 simplifies the design and architecture of Java enterprise applications due to the reduction of configuration code and XML.
- Java EE 6 offers additional features, such as asynchronous servlets and EJB, RESTful web services, CDI, and new JPA criteria API.
- Moving from Java EE 5 to EE 6 also provides a cleaner path for cloud-based enterprise application services, as EE 6 will set up the developers to work in Java EE 7 and beyond.

Con:

- Porting a running EE 5 application to EE 6 is only beneficial if there is a requirement to take advantage of the new features.

3. (Optional) If there is sufficient time and interest, a group may choose to alter their solution architecture to incorporate changes based on Java EE 6 new features.

Answer:



FC = Front Controller	SLSB = Stateless Session EJB	Domain Store ≈ Hibernate or TopLink
JB = JAVA Bean	SFSB = Stateful Session EJB	DAO = Data Access Object
POJO = Plain Old JAVA Object	DTO = Data Transfer Object	

Based on the Java EE 6 new features in the web tier, we can make use of Web 2.0 technologies, including Ajax, HTML5, web frameworks like Google Web Toolkit, Dojo, Wicket, and others.

Also in the web tier is JSF component frameworks that include PrimeFaces, IceFaces and Trinidad. JSF offers a clean separation between presentation and implements MVC architecture, supports VDL through facelets, supports multiple view renderers, has an excellent conversion and validation mechanisms, supports Ajax, supports custom components, and supports internationalization.

In the business tier we can use asynchronous session EJBs if needed, we can make use of Singleton session EJB's if needed, and we can consume web services, RESTful web services through a JAX-RS session bean. We can also use entity classes (JPA) through the database.

In the integration tier we can use a domain store pattern which represents an ORM software like Oracle TopLink, EclipseLink or Hibernate.

Practices for Lesson 8

Chapter 8

Practices for Lesson 8

Practices Overview

In these practices, you will consider Client tier architectural choices.

Practice 8-1: Architecture for the Client Tier

The client has requested that you enhance the user experience of the Find Auction use case in the Auction system. The suggestions that the client wants you to implement include:

- Improve the user interface. Make the user interface more fun to use (that is, use of the Aqua user interface: pulsing buttons, photo-realistic icons, and so on). Make it look “better.”
- Make the user interface more interactive. As currently implemented, the user can request that the system find auctions that match the criteria the user specified, but the list of auctions found is static. The client suggested that it would be better if this list were interactive:
  - When new auctions are added to the system, and those new auctions match the criteria specified by a user currently examining the corresponding list of auctions, the list should automatically update to include the new auctions.

  - When information for an auction that’s included in the results of a search (such as the current high bid price) is changed by some other user (by placing a greater bid on that auction, for example), the list of auctions should automatically update to present the new high bid price for the auction that changed. This can be achieved currently, but only by asking the system to repeat the current search, which is unacceptably slow and unfriendly to the user.

- The system seems to take too long, between the time the user requests that a search be performed or updated, and the time the screen is updated; or between the time the user requests to scroll to the next page in the resulting list, and the time the screen is updated.

- The user has requested the application be made available on mobile devices like smart phones and tablets.

Changes to address these concerns cannot compromise the application's ability to run on any platform currently supported.

Tasks:

1. Outline potential strategies, both application-level and in terms of technology choices that you could use to address these requirements. For each strategy listed, outline the way in which that choice could be used to address the concerns listed above.

Answer:

The requirements for the client could be reduced to just one: the ability to incrementally update the information presented to the client as circumstances require. A number of strategies could be used to implement this on the client side:

- Server-push

Server-push involves having the server notify clients when changes to the model on take place. This is the most efficient or scalable alternative to pursue, since the server would always know when (or even if) changes take place.

The key limitation associated with server-push strategies is the need for the server to be able to contact any client when appropriate to the server. Web-based technologies tend to be restricted to client-initiated communications, which makes server-push strategies cumbersome, if not impossible, to implement.

- Client-pull

Client-pull involves having the clients query the server for potential changes to the model. This is potentially more expensive than server-push: the client will not always know whether changes have taken place, and so must query the server anyway, just in case.

- Asynchronous ahead-of-time query

Asynchronous ahead-of-time query involves having the client predict the more likely request its user is likely to issue next, then issuing it in advance of the user actually requesting it. That way, when (if) the client actually makes that request, the processing required has already started - and the response may already be on its way back to the client, or even cached already on the client side.

- Client caching

Client caching involves having the client query the server for any numbers of things that the user may be interested in, in advance. The information retrieved is then cached on the client side, which can react to certain actions by its user immediately, since the information is already there.

Among the technologies that could be introduced:

- AJAX

Essentially HTML with embedded Javascript, this technology is attractive because it is a minimum combination of technologies available on most web browser, from platforms as small as cell-phones to desktops.

AJAX provides specific support for issuing asynchronous queries to the server side, which can be used to support efficient client-pull and async ahead-of-time querying.

- Rich clients using Java with JWS, VB/.NET, JavaFX, AIR/Flash/Flex

Rich client platforms provide richer tools to create client-side applications. Those rich-client platforms that are still web-oriented might still be limited to client-initiated interaction with the server; full-fledged client-side platforms on the other hand provide explicit support for server-initiated interaction with the client-side application.

2. For each strategy choice, consider the tradeoffs and risks associated with it. How can you find out and weigh the benefits and costs?

Answer:

Strategies		
Option	**Pros**	**Cons**
Client-pull	• maintainability: solution can be implemented anywhere	• scalability: client must request update, even when nothing has changed • performance: network delay, when triggered by user action that requires refresh
Server-push	• scalability: network interaction only when needed • performance: client updated as fast as possible • scalability: no redundant or superfluous work on the server side	• flexibility: limited platform support
Async Ahead-of-time Query	• performance: minimized response time when client correctly anticipates user's request and initiates processing ahead of time	• performance: requests that turn out to be useless incur overhead to compute answer that will then be ignored • scalability: server may be asked to perform more work that actually needed

Technologies		
Option	**Pros**	**Cons**
HTML with Ajax	• flexibility: widely available	• flexibility: does not support server-push
Java with JNLP	• flexibility, scalability: supports server-push • flexibility: rich framework	• performance: heavier-weight platform • flexibility: not always available
Java FX	• flexibility: robust, rich client platform, shortens development time • flexibility: easy deployment	• flexibility: does not support server-push • performance: requires a full JVM on the client
Flash and Flex	• performance: rich, lightweight framework • flexibility: rich framework	• flexibility: does not support server-push • manageability: licensing may be required for server side499

Practices for Lesson 9

Chapter 9

Practices for Lesson 9

Practices Overview

In these practices, you will consider Web tier architectural choices.

Practice 9-1: Architecture for the Web Tier

The preferred designs for the Auction system have revolved around a web-based front-end to the application, for portability. With such designs, all the client requires is a web-browser to render the HTML-based user interface (UI) to the application.

In Practice 8, you incorporated some HTML-related client- or server-side technologies into your design (such as AJAX) to improve its QoS characteristics.

A consequence of this approach is that most of the logic responsible for the front-end to the application actually resides on the server side, in the Web container that is responsible for generating the HTML-based UI on-the-fly, on a per-client, or per-session basis.

You previously sketched a high-level possible description for the Find Auction and Place Bid interactions. However, the actual application-level structure of the Web tier for the application is not really specified. You've only mentioned the Web components responsible for generating HTML at a rather high level. This lab asks you to architect a more detailed structure for the Web tier for the Auction system.

You have been given certain constraints, as far as the characteristics of the web-based application are concerned:

Portability: The application must be deployable to the widest variety of client platforms.

Maintainability: Changes to the flow through the user interface (in terms of the order in which pages must be displayed, or the ability to introduce additional pages into the flow) must require minimal effort.

Duplication of effort should be avoided. For example, when multiple pages incorporate the same fragments, you should not have to code this more than once.

Tasks:

1. Consider the patterns available for use in architecting the Web tier. Which ones would you use and why?

   Answer:

   - Dispatcher-View Pattern: combines the Front-controller and View Helper patterns. The view components request business processes and thus have to assume some of the controller responsibilities.
     - Disadvantages: View components can have a large amount of scriplet code to request business processing. Many view components can have identical code to request the same business processing.
   - Service-To-Worker Pattern: The controller requests business processes.
     - Disadvantages: The result can be poor cohesion.

2. Consider some of the changes to the Java EE architecture made possible with Java EE 6. How and where could you incorporate these changes into your architecture? For example, using Java EE 5 you might use a polling model. Using Java EE 6 you might use a push model for notification.

Answer:

- Using Java EE 6, we can use server-push strategy in our design by combining asynchronous 3.0 servlets and JSF Ajax-enabled components.

3. Choose a Web framework to use (ex: Struts, JSP-Servlet, JSF, ADF Faces, Spring Web Flow) in your system. Be prepared to give a short (3 minutes) presentation on what you chose and why. Make sure to address risks as well as tradeoffs. You should be considering; cost, ease of use, open standards, availability of developers, training issues, performance, and so on.

Answer:

There are other characteristics that frameworks could feature, and that would help improve our solution. For example:

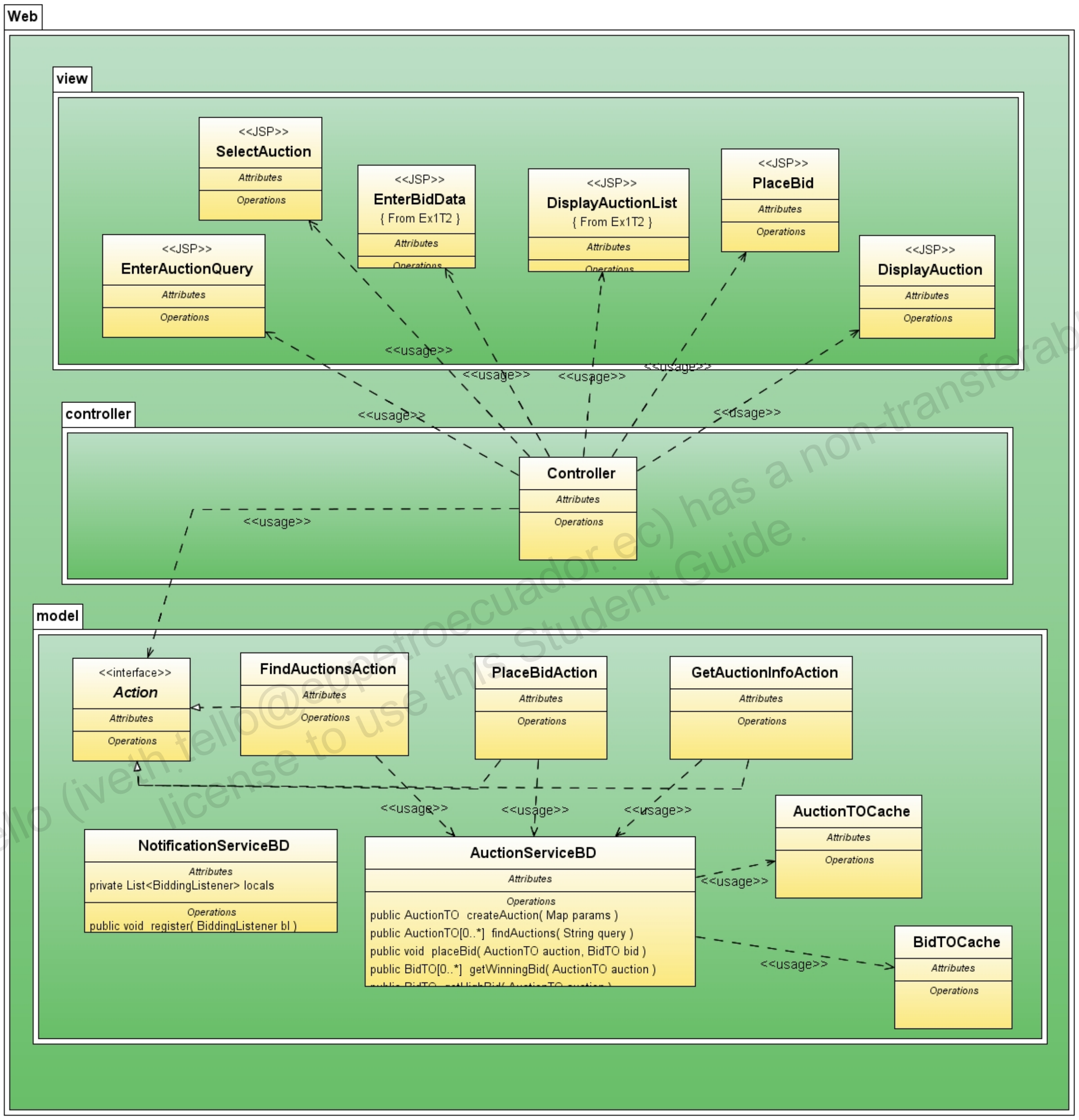
- There are similarities in fragments of content that are relevant to multiple views. This could be captured via shared helper classes, shared included view components, or custom tags.
- There are similarities in structure across multiple views, even if the specific content within that structure is actually different across views. Templating mechanisms that can represent these similarities, such as Struts Tiles, would reduce the effort required to create view components.
- A separation of concerns always leads to more maintainable code. Hence frameworks with support for these principle, such as frameworks that implement MVC, or the Java EE “Front Controller” or “Service to Worker” design pattern (Struts, JSF, ADF), are a win.
- In our model, we have view components responsible for fragments of a view that convey the same information always, but which might have to actually present it in different forms, depending on where those fragments are included. View components that are presentation independent (such as JSF or ADF components), would make this part of the model easier to implement.
- Technology-agnostic application-level components and helpers would make it easier to port our code to different frameworks, if we require a bit of experimenting identifying the best framework for our needs.
- Tool vendor support would help us incorporate new technologies into our design.
- Our application may need to manage session state throughout “conversations” between individual clients and our application. Frameworks that can manage such state automatically would be an improvement over explicit state management at application level.

Technologies		
Option	**Pros**	**Cons**
JSF	**maintainability and flexibility** • JSF custom renderers can hide differences in rendering between HTML-based requests and Ajax-based requests, at app level.	**flexibility** • does not support server-push

Technologies		
Option	**Pros**	**Cons**
Java with JNLP	**flexibility, scalability** • supports server-push..	**performance** • heavier-weight platform **flexibility** • not always available
Java FX	• flexibility: robust, rich client platform, shortens development time • flexibility: easy deployment	• flexibility: does not support server-push • performance: requires a full JVM on the client
Flash and Flex	**performance** • rich, lightweight framework	**flexibility** • does not support server-push **manageability** • licensing may be required for server-side

4. Add to your existing Component diagram to show the additional detail in the Web tier needed to support your choice for Client tier.

Answer:


Web
view
<<JSP>> SelectAuction
<<JSP>> EnterBidData { From Ex1T2 }
<<JSP>> DisplayAuctionList { From Ex1T2 }
<<JSP>> PlaceBid
<<JSP>> EnterAuctionQuery
<<JSP>> DisplayAuction
controller
Controller
model
<<interface>> Action
FindAuctionsAction
PlaceBidAction
GetAuctionInfoAction
NotificationServiceBD
private List<BiddingListener> locals
public void register(BiddingListener bl)
AuctionServiceBD
public AuctionTO createAuction(Map params)
public AuctionTO[0..*] findAuctions(String query)
public void placeBid(AuctionTO auction, BidTO bid)
public BidTO[0..*] getWinningBid(AuctionTO auction)
AuctionTOCache
BidTOCache
Attributes
Operations
<<usage>>

Practices for Lesson 10

Chapter 10

Practices for Lesson 10

Practices Overview

In these practices, you will consider Business tier architectural choices.

Practice 10-1: Architecture for the Business Tier

For this practice you will focus on detailing the Business tier architecture and design needed to support the two use cases: Find Auction and Place Bid.

Tasks:

1. Enumerate Java EE technologies that you could leverage to build the business tier for the Auction system. Indicate how each of the technologies selected could help you achieve any of the functional or non-functional requirements (maintainability, correctness, scalability, performance.)

Answer:

The following Java EE technologies might be relevant to realize the design of the Auction application:

- Stateful Session Beans (SFSB)

  The service objects (session facades) that constitute the external interface to the business tier of our application could be implemented as stateful session beans. The benefits of this choice include:

  maintainability and correctness

  The EJB framework would be responsible for realizing the transaction and security constraints incorporated in the deployment descriptor for the SLSBs, which is one less thing for application level developers to implement.

  In addition, session management also becomes the responsibility of the EJB container - something else application-level developers can ignore.

- Stateless Session Beans (SLSB)

  Some of these same service objects could be implemented as stateless session beans, instead. The benefits of this choice include:

  maintainability and correctness

  Just as for SFSBs, transaction and security constraints become the EJB framework's responsibility. Session management would not be an issue in this case, since SLSBs do not have state to maintain across client invocations.

- Message-Driven Beans (MDB)

  The placeBid use case includes requirement that a new high bid appears on the display of every other user present who is bidding on that same auction. This could be implemented using MDBs, by placing notices on a topic that every bidder reads, every time a new higher bid is entered. The benefits of this choice include:

  flexibility

  JMS would manage the propagation of each notification to each interested listener, regardless of their location.

  scalability

  JMS could optimize the overhead associated with propagating each notification – by using network broadcast protocols, for instance, where appropriate.

- EJB2 Entity Beans or EJB3 Entity Classes

  Persistent domain classes (such as Auction, Bid, and User) could be implemented as EJB2 Entity Beans, or EJB3 Entity classes. The benefits of this choice include:

  maintainability

  Mapping between Java OO model and database (relational) model is automatic. Framework handles differences between database engines.

  flexibility

  Designers are not tied to the relational schema: framework offers to translate any OO model to/from relational.

  performance

  Framework provides built-in optimizations to minimize overhead of persistent operations:

  – caching within transaction boundaries (EJB2)

  – caching across transaction boundaries (EJB3.x)

  – optimistic locking (EJB3)

  EJB2 Entity Beans incur higher overhead, among other reasons, in that they incorporate far more than just the ability to map instances to rows in a database. EJB3 Entity classes, on the other hand, do nothing else. The choice between the two technologies would require closer examination of the infrastructure associated with each of the two.

- Web Services

  Some of the service objects could be published to the world as web services. The benefits of this choice include:

  flexibility

  Framework supports platform independence – web service services and their clients can be written portable in any programming language or platform.

  maintainability

  Framework manages issues related to distributed design, including security (using JAX-WS extensions for WSIT interoperability).

2. If you were to choose to make some subsystems available as Web services, how might this change your architecture? Consider the impact of SOAP-based versus Restful web services.

Answer:

SOAP-based web services are interoperable (i.e. we can make a Java EE server with a .Net server using SOAP-based services.) One downside of SOAP-based web services is performance. When the platforms are both platform Java EE platforms, RESTful web may be used and perform better than SOAP-based services. The downside of RESTful web services is that they are not interoperable without documentation describing the web services APIs.

3. For each of the choices made in Task 1, in terms of how you would address the requirements given in your OO model for the auction system, consider whether there is an alternative to address the same requirement. The alternative might be to build application-level functionality or to delegate the requirement to the underlying Java EE platform. The alternative could also be in the choice of Java EE technology provided by the underlying platform with which you choose to leverage your application-level structure. Discuss the trade-offs associated with each of the alternatives you identified.

Answer:

Strategies		
Option	**Pros**	**Cons**
EJB2	• simple object mode; one entity is one object, always in sync • built-in transaction management • built-in security management	• complex framework • pessimistic, exclusive-access, locking
EJB3.x	• simpler framework • optimistic locking	• complex model: entities can be represented by multiple objects, which may or may not be in sync with each other or the database • explicit transaction management

Strategies		
Option	**Pros**	**Cons**
Entity framework (JPA)	• developer can ignore SQL, relational model	• requires an persistence ORM provider
Direct JDBC	• fine-grained control over implementation can lead to optimization	• implementation up to app-level developer • developer exposed to SQL, relational model

Strategies		
Option	**Pros**	**Cons**
Stateless Session Bean	• scalability: load balancer can address requests to any instance on any server • scalability: no state to replicate across cluster, so no overhead • maintainability: transaction, simple security management built-in	• performance: client must resubmit conversational state as a part of each request • maintainability: complex security management must be implemented at app level
Stateful Session Bean	• maintainability: transaction, simple security management built-in	• maintainability: complex security management must be implemented at app level • may require additional configuration (e.g., session affinity) to avoid synchronization overhead in a cluster
Web Service	• same as Stateless SB • complex security built-in via WSIT extensions, XACML	• same as Stateless SB
POJOs		• maintainability: must implement own transaction and security management

Strategies		
Option	**Pros**	**Cons**
Stateful SB	• relationship to bean maintained automatically, even if bean not in memory	• Higher overhead for EJB container
HttpSession (servlet)	• less overhead	• relationship to bean must be managed by client retrieval of proper HttpSession context, then bean, given session key • may require action to propagate across cluster

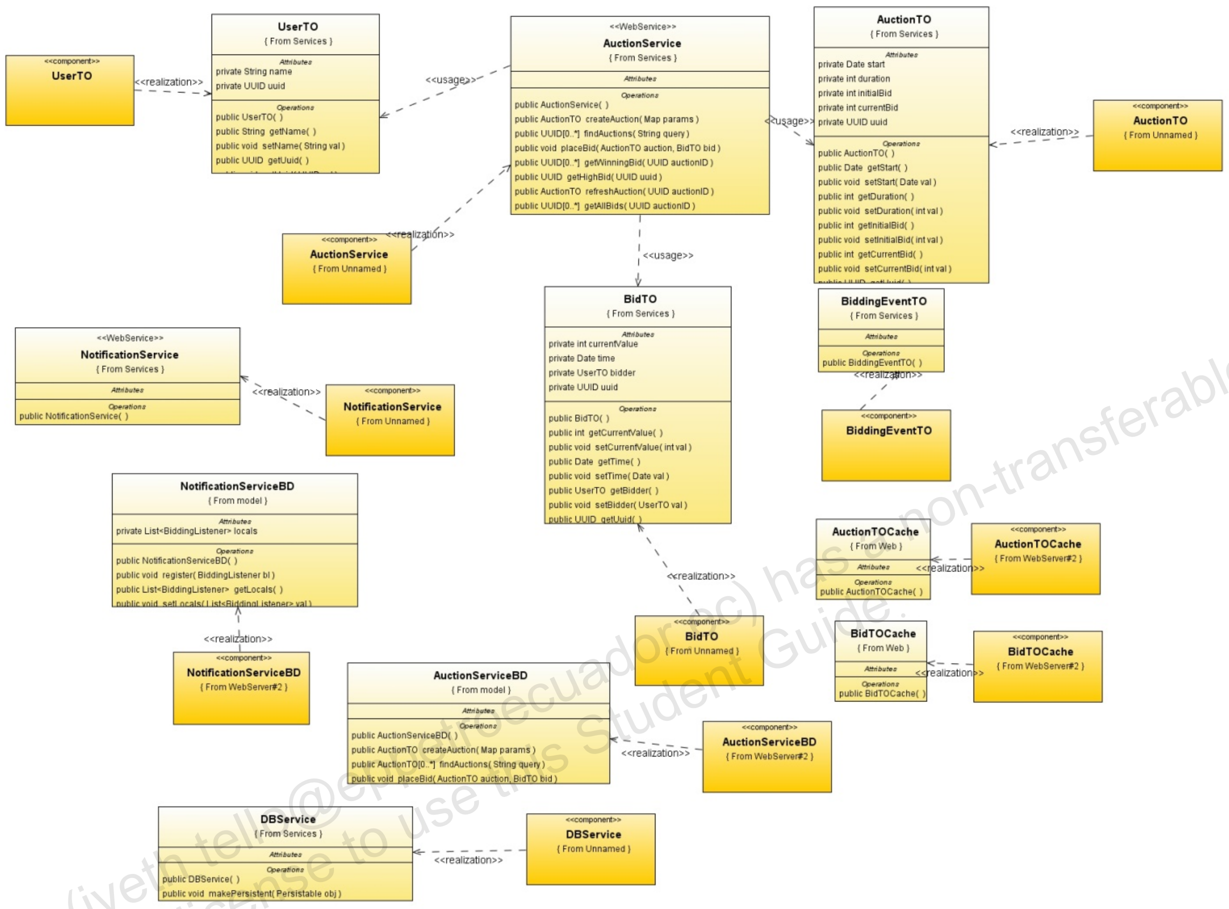
Strategies		
Option	**Pros**	**Cons**
Async notification (server push)	• incurs overhead only when notification is necessary	• not supported on every platform (e.g. J2ME or Ajax clients, clients behind firewalls)
Client polling	• portable to any platform	• incurs overhead every time client polls, even when nothing has changed

Strategies		
Option	**Pros**	**Cons**
OO Models	• familiar model	• verbose model, when capturing cross-cutting concerns
AOP, interceptors	• simple model when capturing cross-cutting concerns	• steep initial learning curve • limited platform support

Strategies: Alternative Technologies		
Option	**Pros**	**Cons**
Spring	• dependency injection for POJOs • non-invasive	• complex AOP
Hibernate	• lightweight • extensions to JPA	• vendor lock-in when extensions used

4. Update your UML models to reflect the changes you have made form the above questions.

Answer:



<<component>> UserTO
<<realization>>
UserTO { From Services }
Attributes
private String name
private UUID uuid
Operations
public UserTO( )
public String getName( )
public void setName(String val)
public UUID getUuid( )
<<usage>>
<<WebService>> AuctionService { From Services }
Attributes
Operations
public AuctionService( )
public AuctionTO createAuction(Map params)
public UUID[0..*] findAuctions(String query)
public void placeBid(AuctionTO auction, BidTO bid)
public UUID[0..*] getWinningBid(UUID auctionID)
public UUID getHighBid(UUID uuid)
public AuctionTO refreshAuction(UUID auctionID)
public UUID[0..*] getAllBids(UUID auctionID)
<<usage>>
AuctionTO { From Services }
Attributes
private Date start
private int duration
private int initialBid
private int currentBid
private UUID uuid
Operations
public AuctionTO( )
public Date getStart( )
public void setStart(Date val)
public int getDuration( )
public void setDuration(int val)
public int getInitialBid( )
public void setInitialBid(int val)
public int getCurrentBid( )
public void setCurrentBid(int val)
<<realization>>
<<component>> AuctionTO { From Unnamed }
<<component>> AuctionService { From Unnamed }
<<realization>>
<<usage>>
BidTO { From Services }
Attributes
private int currentValue
private Date time
private UserTO bidder
private UUID uuid
Operations
public BidTO( )
public int getCurrentValue( )
public void setCurrentValue(int val)
public Date getTime( )
public void setTime(Date val)
public UserTO getBidder( )
public void setBidder(UserTO val)
public UUID getUuid( )
BiddingEventTO { From Services }
Attributes
Operations
public BiddingEventTO( )
<<realization>>
<<component>> BiddingEventTO
<<WebService>> NotificationService { From Services }
Attributes
Operations
public NotificationService( )
<<realization>>
<<component>> NotificationService { From Unnamed }
NotificationServiceBD { From model }
Attributes
private List<BiddingListener> locals
Operations
public NotificationServiceBD( )
public void register(BiddingListener bl)
public List<BiddingListener> getLocals( )
<<realization>>
<<component>> NotificationServiceBD { From WebServer#2 }
<<realization>>
<<component>> BidTO { From Unnamed }
AuctionTOCache { From Web }
Attributes
Operations
public AuctionTOCache( )
<<realization>>
<<component>> AuctionTOCache { From WebServer#2 }
BidTOCache { From Web }
Attributes
Operations
public BidTOCache( )
<<realization>>
<<component>> BidTOCache { From WebServer#2 }
AuctionServiceBD { From model }
Attributes
Operations
public AuctionServiceBD( )
public AuctionTO createAuction(Map params)
public AuctionTO[0..*] findAuctions(String query)
public void placeBid(AuctionTO auction, BidTO bid)
<<realization>>
<<component>> AuctionServiceBD { From WebServer#2 }
DBService { From Services }
Attributes
Operations
public DBService( )
public void makePersistent(Persistable obj)
<<realization>>
<<component>> DBService { From Unnamed }

Practices for Lesson 11

Chapter 11

Practices for Lesson 11

Practices Overview

In these practices, you will consider Integration and Resource tier architectural choices.

When a bidder wins an auction, you do not want to be responsible for the actual processing of credit card payment requests, or the actual transfers of funds to the sellers' (credit card or bank) accounts. Traditionally, online e-commerce web applications rely on third-party credit card processing or bank clearinghouses to take care of financial operations. You need to interact with these third-party clearinghouses over the Internet and it might be that these clearinghouses are not Java EE-based services.

There are a number of nonfunctional requirements to keep in mind, as you consider how to build this part of our Auction system:

Portability: The system needs to be able to interact with third-party clearing houses, regardless of their protocol or implementation.

Maintainability: The changes that are required to switch the system from one third-party clearinghouse to another should be minimal.

Performance: The interaction with the third-party clearinghouse might be expensive or slow, but the Auction system must never block clients for significant lengths of time. Also, it might be that the clearinghouse only accepts financial transactions in batches. But you cannot ask the users to wait until other transactions are ready to issue them all in batch form to the third-party clearinghouse.

Correctness: The system should perform its own operations plus the clearinghouse-based financial processing as a single operation. You should not mark the auction as paid until and unless the clearinghouse processes the payment successfully.

Practice 11-1: Integrate with External Service Providers

Tasks:

1. List the integration technologies (Java EE or otherwise) that you could leverage to build this part of the Auction application, along with the way in which you would use each one to achieve the requirements listed above. For each one, include the risks associated with using it how you could mitigate these risks as well as the tradeoffs associated with choosing it. Risks and tradeoffs should include, but are not limited to: security, performance, interoperability, transactional capability (that is, ability to rollback or compensate should a part of the transaction fail or time out), impact on architecture, design and implementation factors, staffing and cultural issues.

Answer:

portability and maintainability

Implementation details of the remote service providers could be hidden behind their external service interface, which could be implemented on top of JMS (messaging-based), JAX-WS or JAX-RS (web services), or CORBA. All of these would initially require agreement on which technology to use, and what the external interface to the service would be.

ebXML could introduce an extra layer of abstraction on top of a basic web services interaction, that would allow it to act as an adapter between the assumption the client wants to make about its interaction with the service, and the assumptions captured in the public interface to that same service.

JCA (the Java Connector Architecture) could be used within a JavaEE application to provide a means for client to contact a remote service, but in such a way that the details of the interaction would be hidden within the JCA connector. It also introduces an extra layer of abstraction that would allow the choice of technology used, and any mismatch between the client and server assumptions, to be handled by the JCA connector itself.

One can also apply the Business Delegate pattern, which would introduce an application-level component to hide any server-side details behind it.

security

CORBA (built-in, limited support) and JAX-WS (via WS-Security extensions) provide machinery built into their respective infrastructures to handle security concerns. JMS does not provide any security infrastructure- a JMS solution would require extra work at application level to handle these concerns.

correctness

The application could introduce distributed transactions that encompass both operations within the application itself and remote processing on the clearinghouse on behalf of these operations. This would guarantee all-or-nothing, correct, behavior for each of the requests processed by our application. This would require that the remote clearing house support transactions, and that it's possible to propagate transactions initiated by our application to that remote clearinghouse.

CORBA (built-in) and JAX-WS (via the WS-AT extension) can propagate transactions to the remote clearinghouse. JCA provides the machinery that allows a JCA connector to obtain the current transaction context so as to propagate it to the remote service. JMS does not contemplate transaction propagation.

These technologies assume that the request that will propagate a transaction context from client to server will be synchronous. When propagating transactions is not possible, one can apply patterns such as Web Service Broker to ensure that rollback always occurs when appropriate, even if it is implemented through a compensating transaction.

performance

Asynchronous interactions would achieve our goal to never block the client while time-consuming operations execute. JMS (built-in) and JAX-WS or JAX-RS (via Dispatch API) provide such mechanisms, while CORBA provides only limited support (through the oneway modifier).

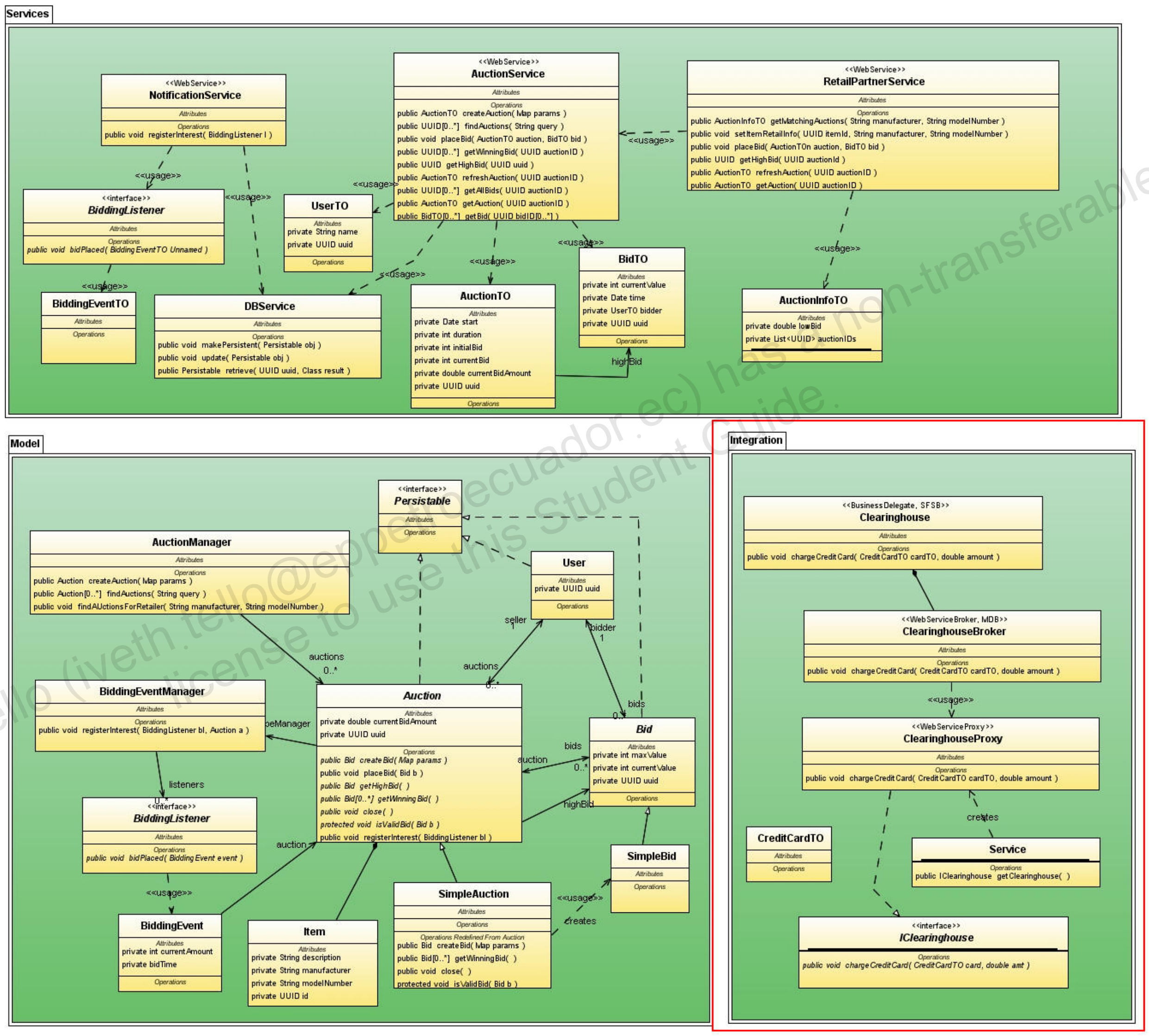
Neither JMS nor JAX-WS propagate transactions across async interactions, however. Support for both transactions and async interactions might need to be implemented purely at application level.

The trade-offs associated with these technologies are presented in the table below.

Technologies		
Option	**Pros**	**Cons**
JMS	• open-ended async interaction	• complex async interactions must be coded at app level • no transaction propagation or security • implementations not interoperable
JAX-WS	• interoperable, ubiquitous • async operation transparent to server	• no transaction propagation or security
JAX-WS + WSIT ext	• transaction propagation • rich security: WSIT, SAML, XAMCL • interoperable, ubiquitous	• transactions do not propagate across async calls
JAX-RS	• simple implementation, interoperable from Java-to-Java • async operations transparent to the server • transaction propagation • some security	• not fully interoperable (without documentation) • transactions do not propagate across async calls
CORBA	• transaction propagation • authentication • interoperable	• transactions do no propagate across async calls • limited security
JCA	• authentication information available • transaction context available	• security must be implemented at app level • transaction propagation must be implemented at app-level

2. Describe how you could use Java EE 6 technology to capture an interaction between the Auction system and the third-party clearinghouse that allows the Auction system to initiate a request to the clearinghouse to process a payment without requiring that the Auction system block while waiting for a response from the clearinghouse, but makes the response available to the Auction system as soon as it is available. Look for the most maintainable means to achieve this interaction.
3. Update your UML diagrams to reflect your architecture choices.

Answer:



Practice 11-2: Using the Auction System as a Service

Tasks:

So far, you have assumed that the Auction system is an interactive web application to be used by individual buyers and sellers over the Internet to create one item or one auction at a time. However, it might be attractive for you to enter into partnership with other retail sales outlets and enable them to make batch loads of items and auctions.

1. Defining Service APIs.
   a. Define a service API that allows you to offer support for batch loads of items and batch creation of auctions.
   b. Consider how you will allow outside vendors to query and obtain report data on their items and auctions. What types of data would you provide? How would you do so? What impact would this reporting requirement have on your current architecture?
   c. List Java EE technologies that could be leveraged to build the service API to the Auction system, and sketch the way in which the API is implemented.
   d. List trade-offs associated with your approach.

Answer:

There is one major difference between the API that our Auction would want to offer its retail partners, as opposed to the one it offers its regular customers: retail partners would only be interested in auctions that match the item they have for sale that their customer has currently selected. This way, the retail site could offer them a list of auctions for the same item, in addition to the choice to just buy the item from them outright.

The new APIs for the retail partner might include:

- `AuctionInfo getMatchingAuctions(String manufacturer, String modelNumber )`

This method retrieves information about auctions that match the given manufacturer and model number. AuctionInfo would contain the low and high bid values for the set of auctions selected, plus a list of auction IDs for all matching Auctions.

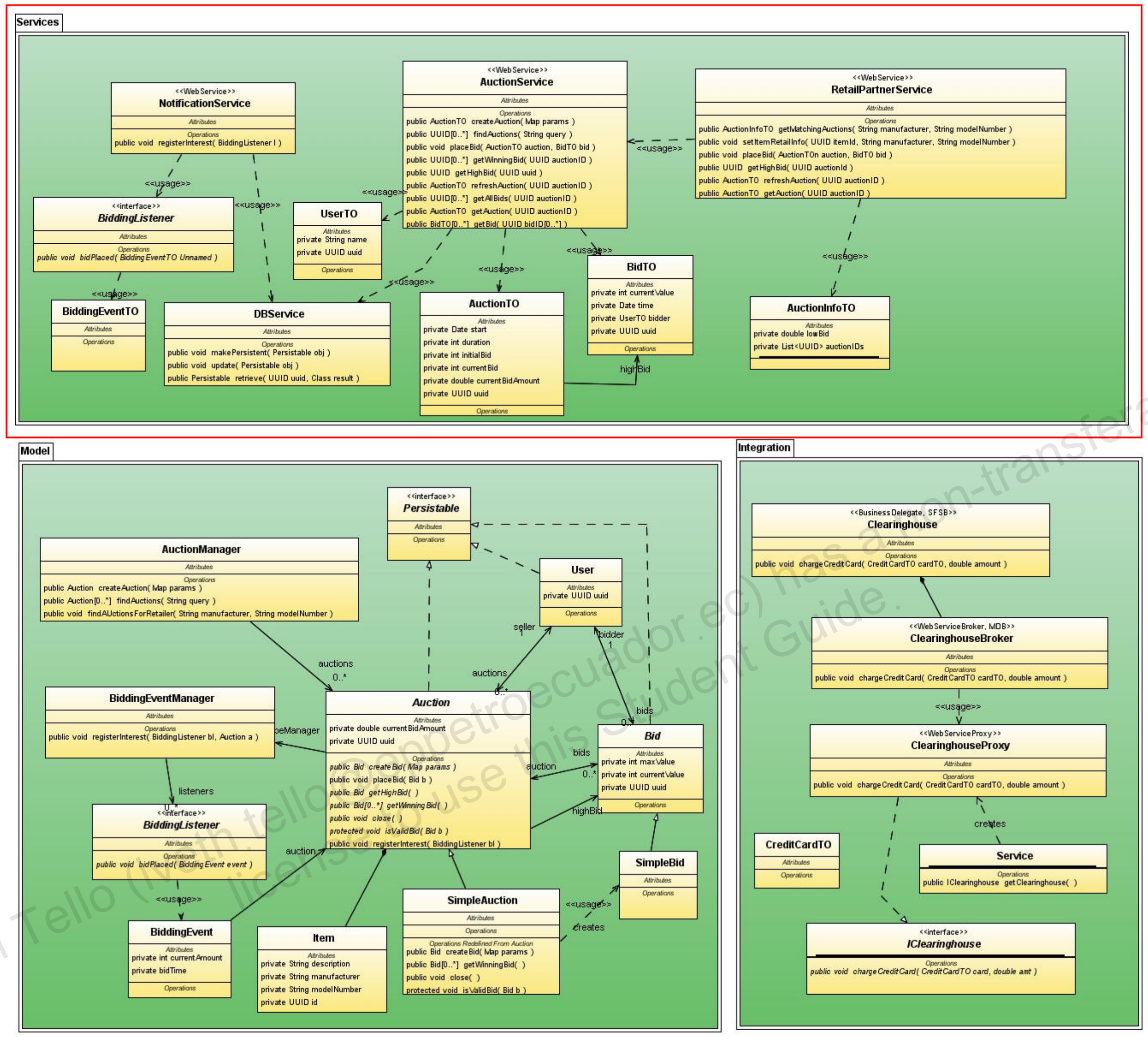
- `void setItemRetailInfo(String itemId, String manufacturer, String modelNumber)`

In order to set the manufacturer info needed for each item. The auction seller may not specify this, so the retail partner might have to.

The list of JavaEE technologies that could be used to implement this service, and their associated trade-offs, are similar to those listed in the table above.

2. Update your UML diagrams to reflect the changes in your architecture for these tasks.

Answer:



Services
<<WebService>>
NotificationService
Attributes
Operations
public void registerInterest(BiddingListener l)
<<WebService>>
AuctionService
Attributes
Operations
public AuctionTO createAuction(Map params)
public UUID[0..*] findAuctions(String query)
public void placeBid(AuctionTO auction, BidTO bid)
public UUID[0..*] getWinningBid(UUID auctionID)
public UUID getHighBid(UUID uuid)
public AuctionTO refreshAuction(UUID auctionID)
public UUID[0..*] getAllBids(UUID auctionID)
public AuctionTO getAuction(UUID auctionID)
public BidTO[0..*] getBid(UUID bidID[0..*])
<<WebService>>
RetailPartnerService
Attributes
Operations
public AuctionInfoTO getMatchingAuctions(String manufacturer, String modelNumber)
public void setItemRetailInfo(UUID itemId, String manufacturer, String modelNumber)
public void placeBid(AuctionTOn auction, BidTO bid)
public UUID getHighBid(UUID auctionId)
public AuctionTO refreshAuction(UUID auctionID)
public AuctionTO getAuction(UUID auctionID)
<<interface>>
BiddingListener
Attributes
Operations
public void bidPlaced(BiddingEventTO Unnamed)
UserTO
Attributes
private String name
private UUID uuid
Operations
BidTO
Attributes
private int currentValue
private Date time
private UserTO bidder
private UUID uuid
Operations
BiddingEventTO
Attributes
Operations
DBService
Attributes
Operations
public void makePersistent(Persistable obj)
public void update(Persistable obj)
public Persistable retrieve(UUID uuid, Class result)
AuctionTO
Attributes
private Date start
private int duration
private int initialBid
private int currentBid
private double currentBidAmount
private UUID uuid
Operations
AuctionInfoTO
Attributes
private double lowBid
private List<UUID> auctionIDs
highBid
<<usage>>
Model
<<interface>>
Persistable
Attributes
Operations
AuctionManager
Attributes
Operations
public Auction createAuction(Map params)
public Auction[0..*] findAuctions(String query)
public void findAUctionsForRetailer(String manufacturer, String modelNumber)
User
Attributes
private UUID uuid
Operations
BiddingEventManager
Attributes
Operations
public void registerInterest(BiddingListener bl, Auction a)
Auction
Attributes
private double currentBidAmount
private UUID uuid
Operations
public Bid createBid(Map params)
public void placeBid(Bid b)
public Bid getHighBid( )
public Bid[0..*] getWinningBid( )
public void close( )
protected void isValidBid(Bid b)
public void registerInterest(BiddingListener bl)
Bid
Attributes
private int maxValue
private int currentValue
private UUID uuid
Operations
<<interface>>
BiddingListener
Attributes
Operations
public void bidPlaced(BiddingEvent event)
SimpleBid
Attributes
Operations
BiddingEvent
Attributes
private int currentAmount
private bidTime
Operations
Item
Attributes
private String description
private String manufacturer
private String modelNumber
private UUID id
SimpleAuction
Attributes
Operations
Operations Redefined From Auction
public Bid createBid(Map params)
public Bid[0..*] getWinningBid( )
public void close( )
protected void isValidBid(Bid b)
auctions
0..*
seller
1
bidder
1
bids
0..*
beManager
listeners
0..*
auction
highBid
<<usage>>
creates
Integration
<<BusinessDelegate, SFSB>>
Clearinghouse
Attributes
Operations
public void chargeCreditCard(CreditCardTO cardTO, double amount)
<<WebServiceBroker, MDB>>
ClearinghouseBroker
Attributes
Operations
public void chargeCreditCard(CreditCardTO cardTO, double amount)
<<usage>>
<<WebServiceProxy>>
ClearinghouseProxy
Attributes
Operations
public void chargeCreditCard(CreditCardTO cardTO, double amount)
creates
CreditCardTO
Attributes
Operations
Service
Operations
public IClearinghouse getClearinghouse( )
<<interface>>
IClearinghouse
Operations
public void chargeCreditCard(CreditCardTO card, double amt)

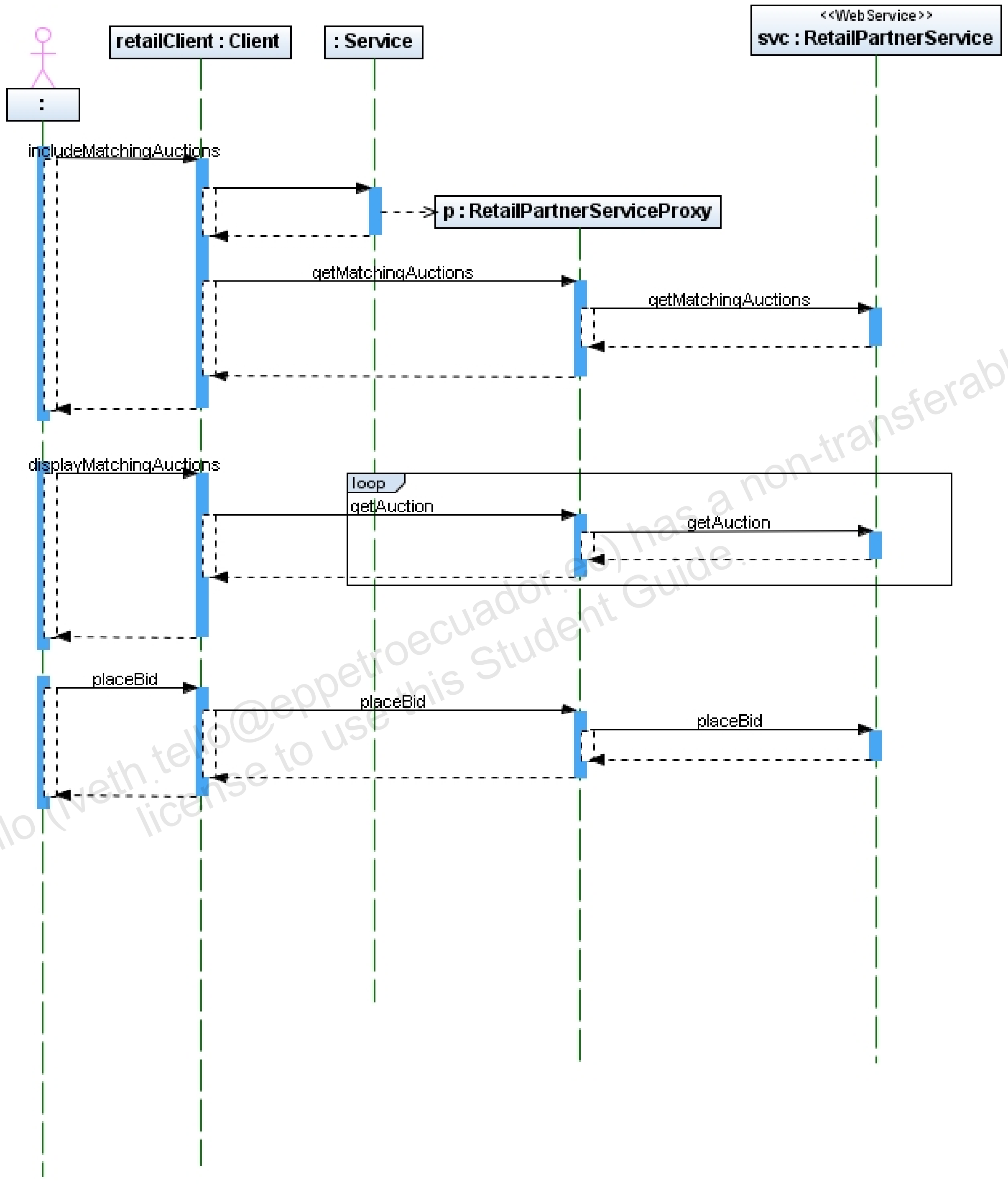
Practice 11-3: Software as a Service Discussion

You've been approached by a large retailer (such as "Buy More, Inc") to allow them to offer auction services on their website by relying on your Auction system to provide them with back-end support. The Auction system supplies their system with the business knowledge necessary to run auctions; they provide a front-end to handle auctions in such a way that the user interface integrates with the rest of their retail website.

Tasks:

1. Consider the requirement as stated. Is it feasible? Is it clear, concise and complete? Is it reasonable? If not, why not and what questions would you ask to clarify and refine the requirement?
2. Be prepared to present and defend your answers. What effect, if any, would supporting this requirement have on your architecture? What are the risks associated with supporting this requirement?

Answer:



retailClient : Client
: Service
<<WebService>>
svc : RetailPartnerService
:
includeMatchingAuctions
p : RetailPartnerServiceProxy
getMatchingAuctions
getMatchingAuctions
displayMatchingAuctions
loop
getAuction
getAuction
placeBid
placeBid
placeBid

Practices for Lesson 12

Chapter 12

Practices for Lesson 12

Practices Overview: MegaBank Case Study

In these practices, you will evaluate system requirements and make architectural choices, considering trade-offs.

Problem Summary

MegaBank (MB) is a traditional, brick and mortar, international bank. MB has nearly 100000 account holders worldwide. Analysts working for MB have determined that to successfully compete with other banks, MB must provide online banking services to its clients.

In pursuit of this goal, MB recently acquired a smaller, regional bank called Thirteenth National Bank (TNB). TNB has just over 35000 account holders and is well known for its customer loyalty. TNB has been providing online banking services to its customers for a number of years, which is of great interest to MB.

MB has decided to implement the TNB banking services throughout the international corporation. MB currently has a completely internal system with various applications used by its staff. TNB has both internal applications, as well as the online, customer-oriented applications.

MB management wants you to create an initial architecture in the form of a proposal to solve its current problems. MB would like to leverage the existing applications that are owned by MB and TNB and combine them into a single, unified architecture. MB recognizes the need to provide sophisticated online services, as well as the potential for business-to-business (B2B) integration that online services might bring in the future.

Functional Requirements

MB is concerned about the user interface for this online application. Because MB does not expect its users to be highly technical, every effort should be made to eliminate unnecessary or confusing steps. In short, MB wants the interface to be simple, consistent, and easy to use.

Though not a requirement, MB would someday like to have its internal staff use portions of this application. For example, customers should be able to view its past transactions online, and they should also be able to walk into the bank and have a teller give them a list of their past transactions. The same process could be used in both cases.

MB has already identified the following requirements for its online application:

- View account balances
- View pending transactions
- View transaction history
- Transfer funds between user accounts
- Buy stocks
- Sell stocks
- View stock portfolio
- View list of unpaid bills
- Pay a specific vendor
- Configure automatic bill payment rules:
  - Date of payment
  - Amount of payment
  - Account to debit

- Email and telephone support on a 24/7 schedule
- Update user information
- Change password

MB might have another process currently in place to fulfill some of these requirements, such as opening and closing accounts, and the existing TNB solution might fulfill other requirements.

It is expected that MB will use its existing databases to store all account and customer data.

Along with the user-oriented requirements in the previous list, the tellers and managers in the banking staff have identified the following requirements:

- Open new accounts
- Close existing accounts
- Set up online banking for users
- Correct errors with account balances
- Use OLAP tools to analyze customer data

MB also requires that the application provide some form of authentication. All users of this system must log in before accessing any of the services. Online users will only receive access by means of the Web. All interaction with bank data will be mediated by the application itself. Finally, all users should log off when they are finished with the system.

MB also has discussed the possibility of future enhancements to this application. MB would like to ensure that the architecture is capable of handling any of the following possible additions:

- A premium service for some customers, which would provide streaming stock information with the latest prices and market conditions
- Brokerage services for enhanced stock management, which might include automated buying and selling of stock based on user supplied criteria or interaction with actual brokers
- An online loan application service

Note – This is not an all-inclusive list of requirements, and you might discover some additional functional requirements as you analyze the MegaBank scenario.

Non-Functional Requirements

Flexibility

MB has identified flexibility as a paramount concern for this application. MB is willing to use its existing infrastructure (or portions of it) for now, but MB wants to avoid "vendor lock" in the future.

Scalability

With the acquisition of TNB, MB has about 135000 customers. MB would like to grow this user base by at least 20 percent in the next quarter. TNB reports that about half (or about 17500) of its customers are using the online services that TNB currently provides. However, there is no information about the concurrency of these users, or about how often the customers are using the application.

Availability

Because this application will often be the only interface that many users have with MB in the future, MB requires no less than 99 percent availability. MB is willing to invest in infrastructure, if needed, to achieve this goal because MB will save money in the long run by reducing paperwork and staff.

Reliability

MB will not accept more than one failure per every 100000 transactions, a reliability rate of 99.999 percent. MB places a high importance on reliability because of its industry. MB knows that users will not accept any glitches or errors in online monetary transactions. Ultimately, if the customers do not believe that the online application is safe and reliable, they will not use the application and might leave MB altogether.

Performance

MB recognizes that part of the benefit of online banking is the savings in user time. Therefore, MB wants a system that is responsive to user requests. While MB understands that there will be various bandwidths and connections from its users, MB feels that it is reasonable to expect a performance of no more than three seconds per request, as measured at the MB firewall.

Extensibility

One problem with the existing TNB online application is a lack of extensibility. Consequently, MB wants to ensure that this new architecture is extensible. MB already has plans for future enhancements, and MB wants a system that will allow for easy and cost efficient additions.

Security

MB is very concerned about security, and MB knows that its customers will share this concern. MB is mandating the use of the SSL protocol for all access to the online application. MB also wants the system to track and log all failed login attempts, so that after three failed attempts, the user account is “locked”. If an account is locked, the user will have to visit a branch of the bank to correct the problem. MB wants firewalls to protect its sensitive, internal data.

Manageability

MB has a small staff of system administrators, who will manage the online application after it is deployed. To avoid major manageability problems, MB wants to ensure that the architecture is only as complex as it needs to be to meet the functional and non-functional requirements

Existing Infrastructure

MB has the following software and hardware in place:

- An IBM AS/400 server that hosts MB's existing applications. These applications are written mostly in COBOL, with some C and C++ that has been added along the way to support enhancements.
- An IBM DB2 database that stores MB's customer and account information.
- An external clearinghouse that serves as an intermediary between banks, businesses, and other entities that need to transfer funds. The clearinghouse mainly handles checking account activity, but also handles any external fund transfers. All external transactions are processed nightly.
- An existing DB2 OLAP application that permits bank staff to analyze data to determine the usefulness of MB's services. This application is provided by IBM and runs on the existing AS/400 server.

TNB has the following software, services, and hardware in place:

- A recently purchased Sun E10000 server (Starfire™ hardware) that hosts TNB's existing application with the Solaris™ 8 Operating Environment (Solaris OE) as the operating system.
- A third party company that handles static web content.
- A centralized server that runs on the Sun E10000 server and is housed and maintained by TNB. This server manages all of the dynamic web content, which is CGI-based and written in Perl.
- A component in the application server that provides the rule-based logic for account processing. The business logic is written in COBOL and C, as well as a proprietary language provided by TNB's application server.
- An ORACLE® RDBMS that stores all data.

Assumptions and Constraints

MB has identified the following assumptions and constraints for the online application:

- Java EE technologies should be used. While other technologies could be used, the MB analysts prefer the standard of Java EE technology.
- Payments must not exceed the customer balance. No payment can be made for more than the amount available in the current account balance. Overdraft protection will not be offered in the initial release of the online application.
- The user must have at least one account with MB to be considered a customer.
- The interface should take bandwidth into consideration. Users will be accessing the application with varying bandwidths, and users with higher bandwidth connections could have a more robust interface than those with lesser bandwidth connections.
- The application that MB uses to interface with the external clearinghouse is proprietary. MB would like to continue to use this application. The application supports a CORBA interface.
- Stock market data is currently accessed by way of a proprietary messaging application owned by TNB. This application receives information from a third-party service. MB is interested in having a more "open" application that would enable them to choose different stock monitoring service providers in the future.

Risks

MB has identified the following risks as critical considerations for the online application:

- Though there is a small development group inside TNB that has familiarity with Java EE technology, MB has only COBOL and C/C++ developer.
- To provide Java EE technology services, a compliant application server will be required.
- All transactions made at the bank should be correlated with any online transactions. For example, if a customer goes to the bank to withdraw funds from a checking account, the balance that the customer sees online should reflect the pending withdrawal.
- Some customers will not be comfortable with using an online interface.

Note – While these risks are the most critical for MB stakeholders, the list is incomplete. Expand the list with any additional risks that you identify for this architecture.

Practice 12-1: System Requirements, risks and assumptions.

Tasks:

1. Read the problem summary for MegaBank.
2. List the functional requirements of the system. This might be completed as a class or small-group activity. Consider the following concepts:
- The existing functionality of both banks
- The new functionality that is required for this application
- The integration with existing applications
3. Organize the list of functional requirements into groups of distinct services. Label each service with a clearly defined name, such as Stock Service.
4. Discuss the non-functional requirements with the class or your group, and spend some time discussing their presumed priority. Consider the trade-offs that will be required to realize the non-functional requirements.
5. List any other risks and assumptions that you might have discovered beyond what is listed in the problem summary itself.

Answer:

The following list identifies the functional requirements for MegaBank, which are grouped into distinct services. This grouping is arbitrary, and you might have grouped the functional requirements differently.

Account services:

- View account balances
- View pending transactions
- View transaction history
- Transfer funds between user accounts

Stock services:

- Buy stocks
- Sell stocks
- View stock portfolio

Bill payment services:

- View list of unpaid bills
- Pay a specific vendor:
- Configure automatic bill payment rules
- Date of payment
- Amount of payment
- Account to debit

Customer support services:

- Support customers during normal business hours
- Email
- Telephone
- Support customers during non-business hours – Provided by third party

Profile management services:

- Update user information
- Change password

Administrative services:

- Open new accounts
- Close existing accounts
- Set up online banking for users
- Correct errors with account balances – Use OLAP tools to analyze customer data

Practice 12-2: Create use-case diagrams for one group of services

Tasks:

1. Identify and list the actors involved in the MegaBank scenario. Note that a single actor might be involved with several services.
2. Select one service that you wish to work with for the remainder of this lab. From this point forward, you will work with only the service you select.
3. For each functional requirement in your chosen service, document the use-case scenarios. Be sure to include any alternate flows of events.
4. Create a use-case diagram that models all of the use cases and their interactions for your chosen service.

Answer:

Identified Actors

The following list identifies an initial group of actors for the MB online application:

- Customer
- Internal bank staff:
- Teller
- Manager
- Administrator
- Vendor
- Customer service representative
- Broker
- Clearinghouse
- The OLAP application
- External stock monitor

Note – You might have discovered other actors in your analysis.

Use-Case Scenario Descriptions

In this exercise, you were asked to develop use-case scenarios for one service of the MB online application. This solution provides suggested use-case scenarios for the account service. The account service was selected because it represents the core of the bank's business model.

An analysis of the account service resulted in six initial use-case scenario descriptions.

Log in Use Case

- Precondition – The customer has loaded the main public bank web page.
- Main Flow – The customer is prompted to enter a user ID and password. The system verifies the login data. The system displays the main screen for the online banking page, which contains a set of icons for performing online tasks.
- Alternate Flow 1 – If the customer enters an invalid user ID or password, the application tracks concurrent login attempts and displays an error message for the first two failed attempts, which advises the customer to try again.
- Alternate Flow 2 – If the customer has made three consecutive unsuccessful login attempts, the application records the attempts and locks the account from online access until the customer contacts MB directly. The application returns an error message to the customer, which explains that the account is locked.

- Notes – The login use case must return successfully before any other use case can be accessed. All other use cases must ensure that the user is successfully logged in before they can proceed. Additionally, a timeout is associated with online customers that forces the login use case to run again.

View Account Balance Use Case

- Precondition – The customer must successfully log in to the banking system.
- Main flow – The customer selects the icon to view account balances. The system will look up the account information from a data store for the specific user. The system will return all of the balances for all of the accounts owned by the customer.
- Alternate Flow – The customer has not logged in successfully or the login has timed out. The login use case begins.
- Notes – Customers will not have the option to view the balance of just one account. The application returns all customer-owned account balances when the user selects the view account balance icon.

View Transactions Use Case

- Precondition – The customer must successfully log in to the banking system.
- Main Flow – The customer selects the icon to view transactions. The application presents the customer with a list of transaction types. The customer then selects the icon for the desired transaction type.
- Alternate Flow – If the customer has not logged in successfully or the login has timed out, the login use case begins.
- Notes – This is a generic use case for all types of transaction viewing. There are currently two defined subtypes including, view pending transactions and view past transactions. However, other types of transaction viewing might be added in the future, such as download transaction list. The difference in the types of transactions available for viewing is based on the location of the transaction information and the level of detail that is provided for the transaction.

View Pending Transactions Use Case

- Precondition – The customer must successfully log in to the banking system and select the icon to view transactions.
- Main Flow – The customer selects the icon to view pending transactions. The application presents the customer with a screen, which allows the customer to choose a start and end date. The application looks up the list of uncommitted transactions from the pending data store for the dates requested (inclusive) and returns a list of zero or more transactions.
- Alternate Flow 1 – If the start or end date is not valid, the system returns an error message, which is displayed to the customer. The customer should be able to try again with a modified start or end date.
- Alternate Flow 2 – If the customer has not logged in successfully or the login has timed out, the login use case begins.
- Notes – The information for the view pending transactions use case is pulled from a data store that holds all transactions that are not yet committed. These are transactions that have not yet been sent to the clearinghouse or that have not yet been committed to the persistent data store, such as checks deposited that have not yet cleared the issuer's bank.

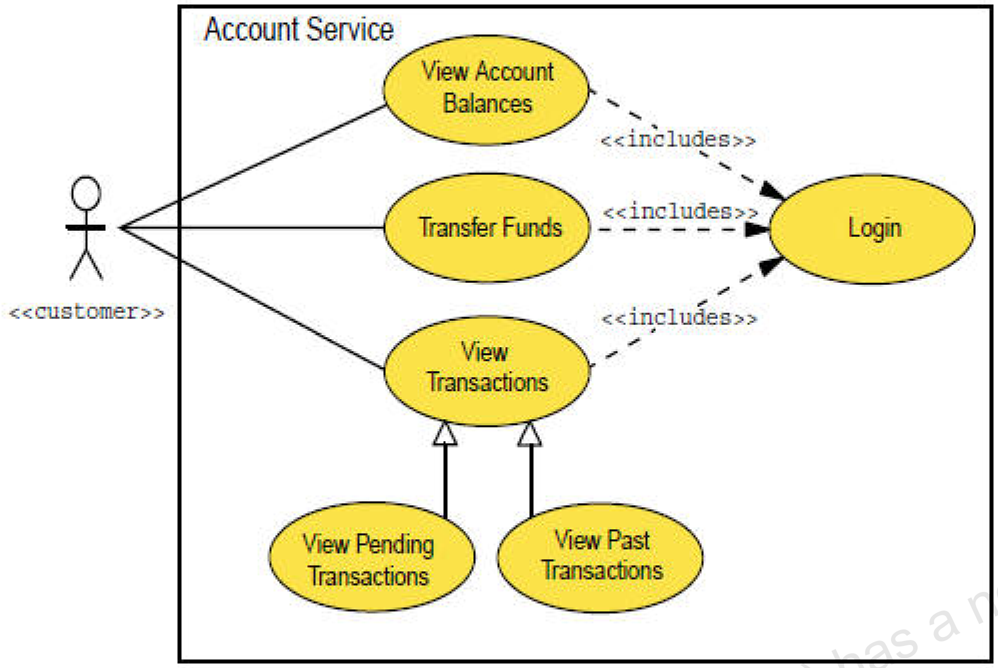
View Past Transactions Use Case

- Precondition – The customer must successfully log in to the banking system and select the icon to view transactions.
- Main Flow – The customer selects the icon to view past transactions. The application presents the customer with a screen, which allows the customer to choose a start date or to accept the default of 30 days. The application looks up the list of committed transactions (called bank transactions) from the persistent data store. The application uses the requested start date (inclusive) and returns a list of zero or more transactions.
- Alternate Flow 1 – If the start date is invalid, the application returns an error message, which is displayed to the customer. The customer should be able to try again with a modified start date.
- Alternate Flow 2 – If the customer has not logged in successfully or the login has timed out, the login use case begins.
- Notes – The information for the view past transactions use case is only pulled from the persistent data store. In other words, the data only represent the committed transactions, and the information reflects cleared deposits, transfers, and other payments.

Transfer Funds Use Case

- Precondition – The customer must successfully log in to the banking system.
- Main Flow – The customer selects the icon to transfer funds. The application returns a list of available accounts. The customer chooses the account from which to transfer the funds, the account to which the funds should be transferred, and then enters the transfer amount. The application verifies that sufficient funds are available. The transfer amount is debited from the first account and credited to the second account. The entire transaction is logged.
- Alternate Flow 1 – If there are not sufficient funds to complete the transfer, the system returns an error message, which is displayed to the customer. The customer should be able to try the transaction again with either a different source account or a different amount.
- Alternate Flow 2 – If the customer has not logged in successfully or the login has timed out, the login use case begins.
- Notes – The application only allows the transfer of funds between customer-owned accounts at this time. There is no provision for transferring funds between accounts that are owned by different customers.

Account Service Use-Case Diagram



The diagram shows only the customer as an actor. Although not mandated in the problem summary, members of the internal bank staff might also use the online application. If bank staff were using these use cases as well, then they would be defined as actors and be placed on the right side of the diagram. Actors on the right side of the diagram represent internal actors on the system.

In addition, notice that the customer does not directly interact with the login use case. This is because a use case is supposed to provide added value to the system user. Although customers are forced to log in to the system before being able to interact with their accounts, the account service must be sure that the customer is authenticated. For example, an authenticated customer could have a token that is validated before carrying out account-related operations.

The view transactions use case is generic in this solution, which allows various types of transaction "viewing." Although this model is not the only possible solution for the view transaction service, it provides for extensibility.

Practice 12-3: Create a class diagram

Tasks:

1. Identify and list the actors involved in the MegaBank scenario. Note that a single actor might be involved with several services.
2. Select one service that you wish to work with for the remainder of this lab. From this point forward, you will work with only the service you select.
3. For each functional requirement in your chosen service, document the use-case scenarios. Be sure to include any alternate flows of events.
4. Create a use-case diagram that models all of the use cases and their interactions for your chosen service.

Answer:

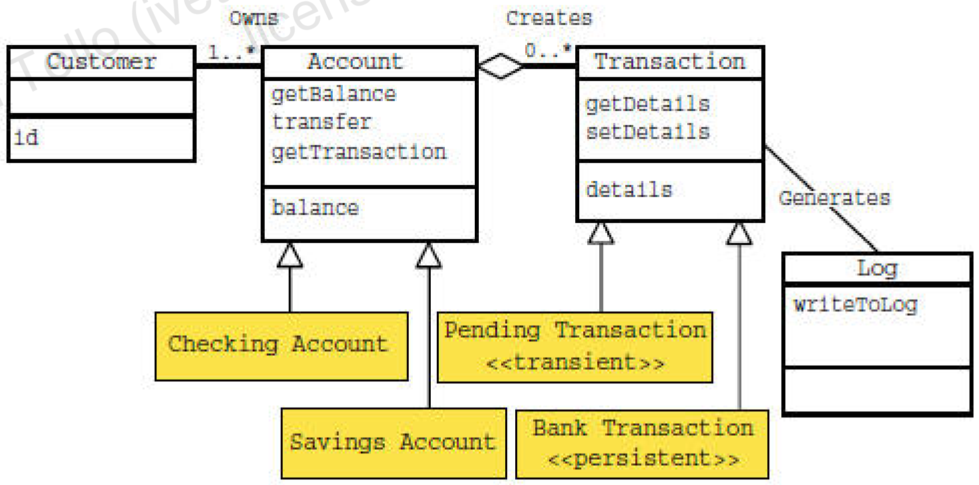
Derived Data Types for the Account Service

The following list shows the persistent and transient data types for the account service:

- Persistent data:
  - Bank transaction
  - Account:
    - Savings account
    - Checking account
  - Customer
  - Log
- Transient data – Pending transaction

Account Service Domain Model

The figure below shows an example of a class diagram domain model for the account service.



Both Account and Transaction are generalizations in this diagram. The more specific Checking Account, Savings Account, Bank Transaction, and Pending Transaction are the key subtypes. The model shows the extensibility that is built into the domain model.

The two transaction subtypes have stereotypes attached to them that describe their nature. The Bank Transaction is persistent because it is a committed transaction, and any displayed funds have already been appropriately and correctly credited or debited to the account. A Pending Transaction is transient only because it is not yet an “official” transaction. An example of a transient transaction is a stop payment order on a check. You cannot stop payment on a check that has already cleared. You can only stop payment on a check that has yet to clear.

The Account is considered an aggregation of Transaction types. This is an aggregation relationship and not a composition relationship because transactions could live beyond the lifetime of an account. For example, if an account is closed, the transactions are still data that the bank will need for its own internal reports. However, the only way a customer can view transactions is in relation to an active, valid account.

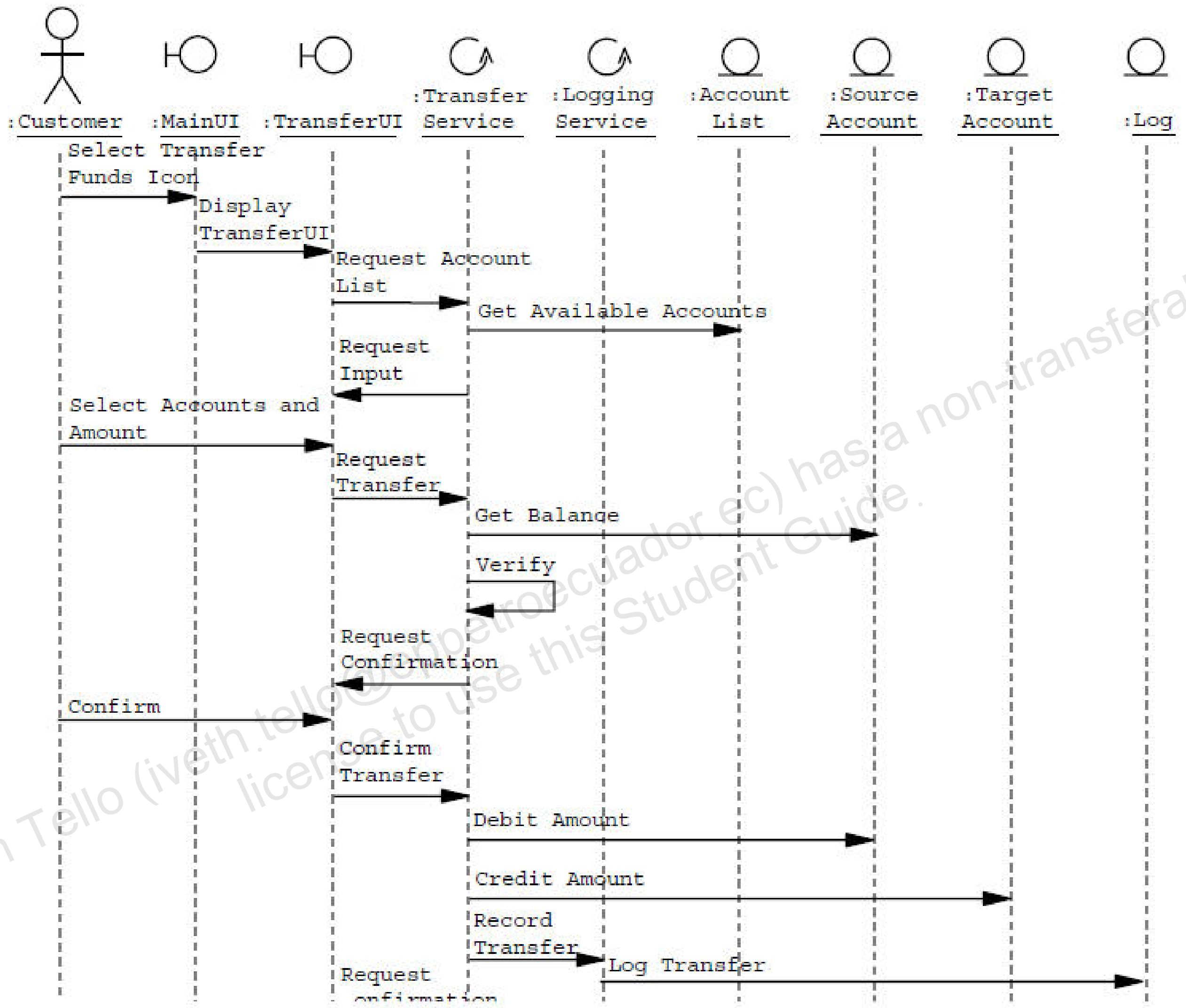
Also, the Log is identified as a persistent data type. This is an example of an emerging type. The use-case scenario descriptions specify that transactions are logged, so this means that the Log should be captured in the domain model. This Log will also be part of future diagrams.

Practice 12-4: Create a sequence diagram

Tasks:

1. Select one of the use cases from your chosen services and create a sequence diagram.

Answer:



The sequence diagram shows the `TransferService` performing a step that is labeled "Verify" on itself. The point is that the verification step is accomplished by the `TransferService` and does not directly involve other components, other than initially getting the balance from the `SourceAccount`.

Practice 12-5: Create a high-level deployment diagram

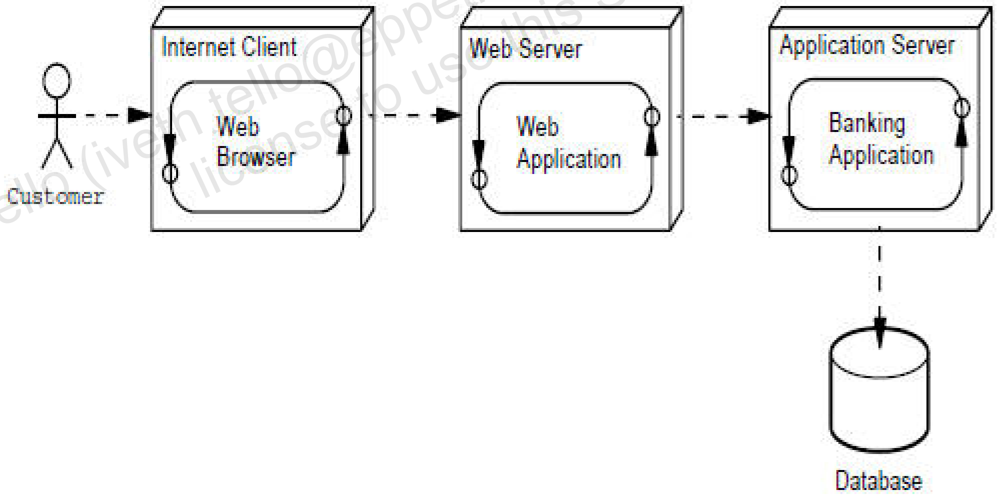
Tasks:

1. Identify the components that you will need in the system to fulfill your selected use case from Exercise 4. Determine which components you will build, which you will buy, and which are already owned.
2. Analyze the relationships between these components with regard to:
   - Extensibility
   - Containment
   - Dependencies
3. Create a component diagram to capture the key components and their interactions.
4. Determine if there are any components that could benefit from reification. If so, alter your component diagram accordingly.
5. Create a high-level deployment diagram to capture the fundamental applications that are involved, along with their physical location, and the interaction between them.
6. Refine your high-level deployment diagram into a detailed deployment diagram that includes the components

Answer:

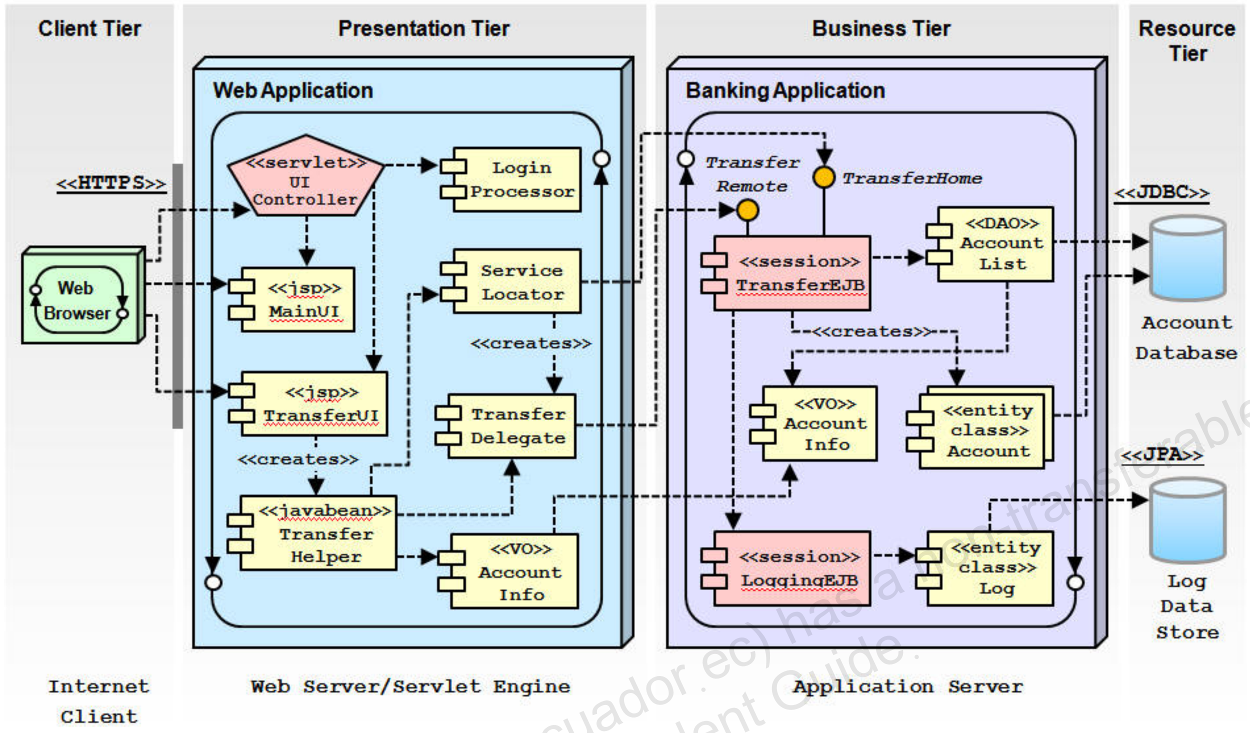
High-level Deployment Diagram

The figure below shows a high-level deployment diagram of the transfer funds use case. This diagram captures the distinct, physical nodes that are expected to be involved in the transfer of funds and the applications that will live on each of the nodes.



Detailed Deployment Diagram

The figure below shows a detailed deployment diagram for the transfer funds use case.



In the presentation tier, the `UIController` servlet is the "front door" for all access. This servlet will handle authentication and the initial validation of the input. This servlet will also dispatch requests to the two JavaServer Pages™ (JSP™) that provide the view of the application. The initial validation that this servlet performs is only a basic check of the input, such as ensuring that the customer enters a number for an amount and not a word. The actual `TransferService` handles any actual validation of the account balances and of the accounts themselves.

The `AccountInfo` component is a VO that is created in the business tier and sent back to the presentation tier. Copying live objects back and forth between components is a feature of Enterprise JavaBeans™ (EJB™) technology. This feature allows fewer calls to the business tier from the presentation tier and allows the account information to be cached in the presentation tier. In the next iteration, you could identify the `AccountInfo` component as a transient data type, and add it to the list created in Practice 3. The thick black bar between the presentation tier and the business tier represents a fire wall.

In the business tier, the `AccountList` is a DAO, which encapsulates the details of finding the accounts for a customer and returning the `AccountInfo` object.

The `Log` entity stores its data in the log data store. At this point, the solution does not clearly define the log data store. It might be a database, a file system, or an XML repository. More information is required in the next iteration to better define the log data store. Also, the stereotype of Java EE CA means the Java EE Connector Architecture because we presume different connectors could be used to interact with various logging mechanisms.