Software Architecture Documentation for Shipping Tracking System

Table of Contents

Table of Figures

1. Introduction

The purpose of this document is to provide a clear view of the architecture of the Shipping Tracking System. This document allows stakeholders to see the inner workings of the system, and to understand exactly what processes are invoked where, and how they interact with each other. This document will provide users of varying expertise and needs with detailed information at both high and low levels of abstraction. Sections 1 – 5 are at the level of detail that most stakeholders can understand, while sections 6 – 7 provide more detailed information at the module level for expert users. Sections 8 – 10 provide background information such as acknowledgements, references, and appendices to further the reader's knowledge.

2. Background

Currently, the business that commissioned this project does not have a publicly available interface to their shipping and tracking system for their customers. The business needs to provide a simplified, unified, and reliable interface to their companies shipping manifestos to better inform the customer about the status of their package shipments. Additionally, one of the primary goals of the business is to capture more of the online shipping market. Therefore, it is necessary for the product to be developed as World Wide Web service. In accordance with the business' initiative to modernize their system, the business desires to replace the use of paper forms with web-based forms, capture customer signatures using a delivery person digital assistant device (DDA), and allows users to display the location of packages on a digital map using GPS coordinates in real-time.

The business currently has an internal shipping and tracking system distributed among multiple warehouses. The system is a mixture of paper and bar-coded forms as well as legacy inventory management software. The various warehouses are connected to a central server located at the corporate headquarters. The system being developed will replace the business' existing system. One of the reasons for this decision is that the business' existing system does not effectively translate into a service oriented architecture. The underlying legacy systems will simply not be able to handle the load associated with servicing potentially hundreds of package status requests and real-time updates per minute.

The primary stakeholders of the project are the members of the customer service team. Currently, the team is responsible for the operation of an automated phone system that informs customers of the status of their packages. Users must enter package tracking numbers by either using the keypad on their phone or by using voice recognition software. The customer service team would like the business's package tracking system to be fully automated and web-based in order to cut down on the costs associated with maintaining the customer service phone line. The primary expense is the payment of personnel to interact with users who cannot properly utilize the service. These personnel must be available to assist users during normal business hours. If the system was web-based, customer service personnel could interact with users through email, and could be attending to other duties during business hours instead of managing the phone line.

Also, the marketing personnel of the packing shipping business are the secondary stakeholders of the project. In order to market the business' shipping services to Internet-savvy customers and capture the online shipping market, the system being developed must have unique features that set it apart from the competition. The primary feature that accomplishes this goal is the ability for users to track packages in real-time on a digital map.

The third major stakeholders of the project are the existing business employees and Internet users of the system. These people must be able to quickly learn how to utilize the system in an efficient manner.

3. Functional Requirements

- User Authentication and Authorization: Registered users can log into the system by providing a username and password combination. Business employees are authorized to perform employee-specific behavior through the system website. Refer to use cases 2.8 and 2.9, and features 3.1 and 3.2, in TrackingSysReqs.doc.

- Display Package Status and Location on Map in Real-Time: The system can display the location of a package on a map using GPS coordinates in real-time. All delivery trucks will have a GPS device in them that allows for the tracking system to determine their location. If the package is currently moving across the country on a truck, the user can see the package move across the map as it is happening. Refer to use cases 2.1 and 2.2, and feature 3.7, in TrackingSysReqs.doc.

- Track Package History: The system can keep track of the distribution centers that the package passes through while it is in the possession of the package shipping business. Refer to use cases 2.3 and 2.4, and feature 3.4, in TrackingSysReqs.doc.

- Plan Package Travel Itinerary: The system can plan the travel of the package from the source address to the destination address across distribution centers. Also, the system can estimate the arrival time of the package at the destination address. Refer to use cases 2.1 and 2.2, and feature 3.5, in TrackingSysReqs.doc.

- Electronic Signatures Upon Receipt of a Package: When a package is received, the receiver signs a handheld device that records the signature and sends it to the server for storage. At any time, the shipper and receiver of the package can then view the signature online as proof that the package was signed for and delivered appropriately. Refer to feature 3.8 in TrackingSysReqs.doc.

- User Adds New Package to System: Employees and registered users can add packages to the system to be delivered. Non-employee users are charged a fee according to the details of the shipment. Refer to use cases 2.5 and 2.6 in TrackingSysReqs.doc.

4. Quality Attributes

Usability Scenario	
Source	End user
Stimulus	Wish to learn how to navigate the system (little to no training)
Artifact	System
Environment	Normal business usage
Response	Provide ability and help in easy to locate fashion
Measure	User satisfaction and task efficiency

Availability Scenario	
Source	Internal
Stimulus	System crash
Artifact	Data storage
Environment	Normal runtime
Response	Switch to backup
Measure	Minutes of system downtime

Security Scenario	
Source	User
Stimulus	Accessing unauthorized information
Artifact	System data
Environment	Online
Response	Block access and require authentication
Measure	Number of blocked intrusions out of total intrusion attempts

Testability Scenario	
Source	Tester
Stimulus	Running system tests
Artifact	Module level of code
Environment	Testing phase
Response	Output a detailed test results report
Measure	Number of reports generated out of total tests completed

Performance Scenario	
Source	Internal
Stimulus	10,000 hits per minute
Artifact	Online connectivity and system response
Environment	High traffic times
Response	Server feedback in less than five seconds
Measure	Average system response time

5. Patterns and Tactics

Architectural Drivers (High Level Business Goals)
1. Have a system that can accept and display real-time GPS information used to track packages and respond to user requests.
2. Have a system that remains consistently available to the users.
3. Have an intuitive system that is easy to use for both customers and employees.

Quality: Availability	
Tactic	Description
Passive Redundancy	The system backs up its data every two hours. In order to ensure that all of the backup systems are working correctly, the backup systems are periodically rotated into the primary position.
Transactions	Because the system uses concurrency to manage requests from both a number of customers and employees, transactions are used to ensure that no two threads collide accessing or manipulating the same package information.
Exceptions	Exceptions are used to recognize faults in the individual system services.

Quality: Security	
Tactic	Description
Authenticate Users	Both customers and shippers are required to ensure that they are who they say they are by having a password-protected account.
Authorize Users	Authorization is used both to ensure that a customer can only access their own account and also to ensure that employees are the only ones who can manage certain package information.

Quality: Performance	
Tactic	Description
Fixed-priority Scheduling	A semantic importance prioritization strategy is used to allow employee actions to be executed before customer actions. This ensures that the most critical tasks, such as updating package shipping information, occur in a timely fashion.
Concurrency	Concurrency is used in order to allow multiple customers and employees the ability to use the system simultaneously. In order words, concurrency provides the system with the ability to handle multiple user requests simultaneously. *Note: Concurrency is an inherit trait of this architecture because it uses a distributed architectural pattern and as such it must be able to handle multiple requests from a number of sources.*
Increasing Computational Efficiency	The system plans a package travel itinerary, and estimates the package arrival time, only once when the package is added into the system (such information can later be updated by certain

	events). The itinerary is then kept until the package is delivered, at which point it is removed from the system since it is no longer needed and will only waste resources.

Quality: Testability	
Tactic	Description
Internal Monitoring	Each of the components of the system maintains information about their state and events that occur in order to provide status reports on request.

Quality: Usability	
Tactic	Description
Support System Initiatives	When an employee enters a package's shipping information into the system, a task model is used in order to automatically find a customer's username, if it exists, based on the shipping or return address. This task model also automatically provides the shipper with a new, unique, tracking number for each package added to the system. Also, a system model is used to provide the user with estimated shipping and delay times.

The primary quality attributes that the stakeholders are concerned with are, as ranked in the architectural drivers, performance, availability, and usability. The performance of the system is paramount because it is required to provide real-time mapping of packages being shipped and also respond to user requests at the same time. Both concurrency and increasing computational efficiency are tactics that support this goal. Since the system must potentially respond to 10,000 requests every minute (as estimated by the stakeholders), concurrency is necessary process all of these requests in parallel. Note that because the system is designed to run as a series of services on a web server, concurrency is already built into the web server framework. This enforces the client-server architecture pattern. Also, because of the large number of user requests, increasing computational efficiency, as described above, will provide large reductions in latency because the overhead associated with calculating shipment information is reduced.

Performance tactics relating to the number of events to process were not applied to the system. The tactic related to managing the event rate potentially provides a way to reduce latency in the system by monitoring the input of new packages less frequently; however, this tactic cannot be applied to the system. This is because changing the system monitoring rate delays can break the system's real-time constraints. For the same reason, the tactic associated with controlling the frequency of sampling cannot be applied. Additionally, applying such a tactic can result the loss of service requests.

The second-most important quality attribute of the system is availability. The system must provide users with the ability to check package shipping status at any time and must provide employees with the ability to modify package shipping information at any time. The transaction tactic is utilized in order to prevent faults caused by the use concurrency. The use of transactions will help prevent collisions between threads that simultaneously access the same data and services from the system. Also, passive redundancy is used to help prevent system data loss when data on the primary system is corrupted by a spurious process, malicious user, etc. Since the system relies on a SOA exceptions are used to ensure that if a service experiences a fault it is handled appropriately without causing the rest of the systems functionality to fail or be degraded.

The usability of the system is very important to the success business. Users who cannot effectively interface with the system will choose not to use it, and the business will lose their market share. Also, employees who cannot effectively interface with the system will be less efficient and more prone to make errors. Thus, the system must have an intuitive design and be easy to learn how to use, thereby increasing user productivity. In order to achieve these goals, task and system models are used as part of the supporting system initiatives tactic. In order to promote learn-ability, a task model for registering a package is used in order to find shipper or receiver information if they are registered. Also, a task model is used to create a tracking id number for the package. This promotes learn-ability because it decreases the number of steps required to perform certain tasks. High productivity is gained through the use of a system model that provides the end user with commonly required information like estimated shipping times and planned package itineraries. Time and effort is saved because users will no longer need to manually calculate or guess this information.

6. Views

6.A. Uses Module View

1. Primary Presentation

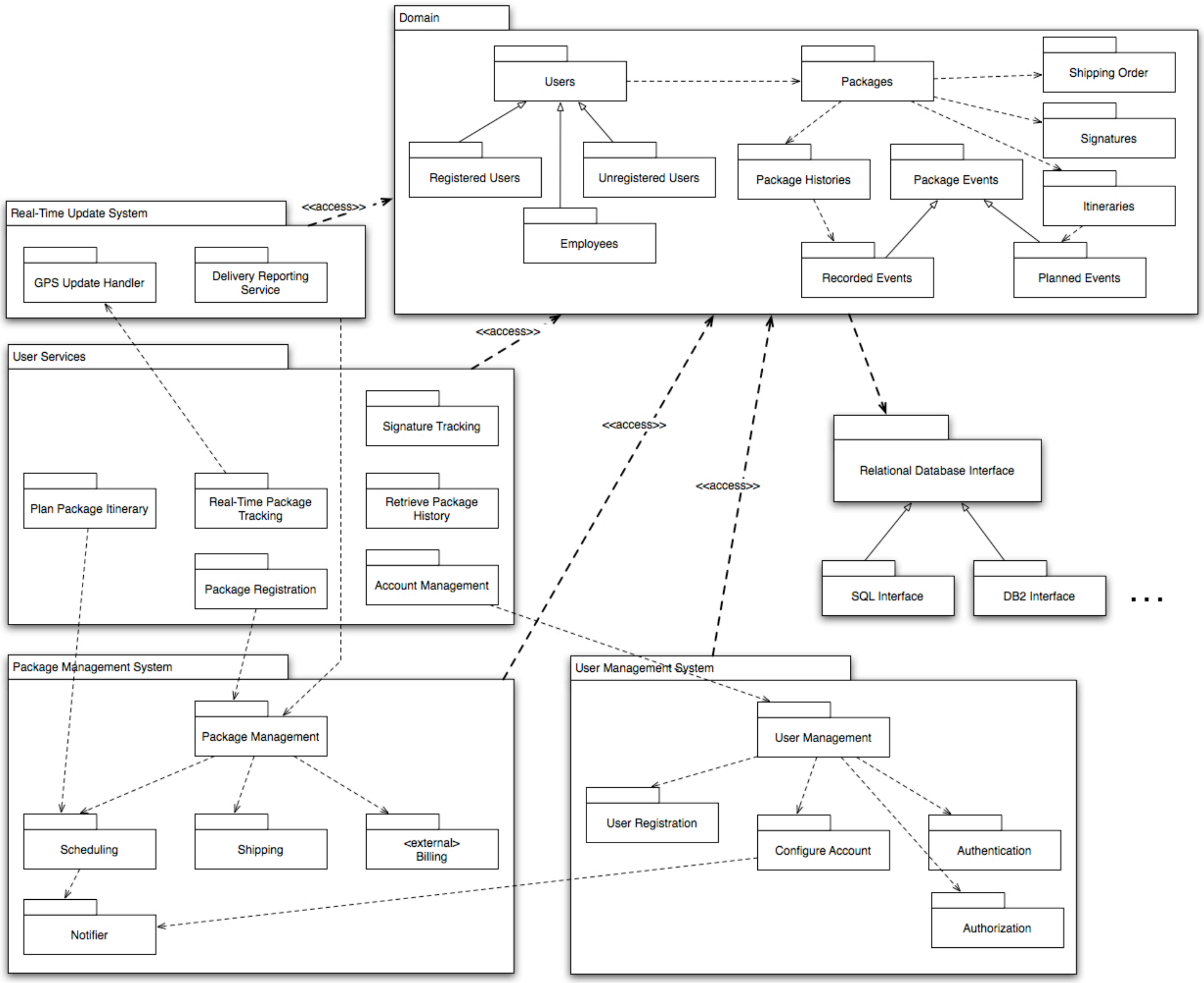


Figure 6.A.1: Uses Module View

2. Element Catalog

A. Elements

1. Domain Package
   - 1.1. Users
     - 1.1.1. Registered Users
       - Contains Information relating to registered users like a username.
     - 1.1.2. Unregistered Users
       - Holds temporary information related to an unregistered user of the system.
     - 1.1.3. Employees
       - Holds information about the employees who make use of the system.
   - 1.2. Packages
     - 1.2.1. Package Histories
       - Provides access to a history of all the events that occur during a package's shipping lifespan.
     - 1.2.2. Shipping Order
       - Contains packages related shipping information.
     - 1.2.3. Signatures
       - Holds information about the delivery signatory for packages.
     - 1.2.4. Itineraries
       - Holds information about packages' shipping routes.
   - 1.3. Package Events
     - 1.3.1. Recorded Events
       - Used to maintain a listing of events that have occurred during a package's shipping lifespan.
     - 1.3.2. Planned Events
       - Used to maintain a listing of events that should occur during a package's shipping lifespan.
2. Real Time Update System
   - 2.1. GPS Update Handler
     - Receives and interprets information about the location of where packages are using GPS reports.
   - 2.2. Delivery Reporting Service
     - Receives and interprets information about a packages delivery status/attempt from a DDA.
3. User Services
   - 3.1. Plan Package Itinerary
     - Used to create a list of locations where a shipment is expected to travel while en route to its final destination.
   - 3.2. Real-Time Package Tracking
     - Responsible for updating package information when the system receives GPS reports.
   - 3.3. Package Registration
     - Allows a user to enter a package into the system.
   - 3.4. Signature Tracking

- Maintains a listing of and provides access to signatures received from package deliveries.

3.5. Retrieve Package History
 - Provides access to a user's shipped and received package history.

3.6. Account Management
 - Allows customers to manage their own accounts and also provides employees with the ability to manage the user accounts.

4. User Management System
   4.1. User Management
    - Manages access to the classes within the User Management System.
   4.2. User Registration
    - Allows end users to create a registered account.
   4.3. Configure Account
    - Allows users to configure personal preferences.
   4.4. Authentication
    - Used to ensure that an employee or registered user is who they say they are.
   4.5. Authorization
    - Used to make sure that different users can only access and modify the data and services available for their user type.
5. Package Management System
   5.1. Package Management
    - Manages access to the classes within the Package Management System.
   5.2. Scheduling
    - Responsible for scheduling a packages course of travel and estimated delivery information.
   5.3. Shipping
    - Provides functionality that describes the shipping information of a package.
   5.4. Billing
    - External system that holds billing information for a package.
   5.5. Notifier
    - Creates notifications for scheduling events that occur, like shipping delays.
6. Relational Database Interface
 - Interface to a type of relational database, such as SQL.

B. Relations

1. The connection between two packages represents that one package uses another package to complete its duties. So in order for one package, with a relation to another package, to work properly the second package must be present. This relationship is shown in Figure 6.A.1 by having a UML dependency between the individual packages. Top-level packages are shown to have this same type of relationship when the <<access>> stereotype is present on the dependency. Also packages can inherit the characteristics of their parent packages. This relationship is specified, and shown in Figure 6.A.1, by the UML generalization.

C. Interfaces

A. Relational Database Interface

1. Resources provided

1.1. Package GetPackageInformation(PackageID id)

- Returns the package identified by “id” if successful or NULL if no such package exists.

1.2. boolean AddPackage(PackageInfo info)

- Adds a package with shipping and receiving information contained in info to the DB.

- Returns true if the operation was successful, false otherwise.

1.3. boolean RemovePackage(PackageID id)

- Removes a package from the system

- Returns true if the operation was successful, false otherwise.

1.4. PackageHistory GetPackageHistory(PackageID id)

- Returns the history for the package identified by “id” if successful or NULL if no such package exists.

1.5. boolean UpdatePackageInformation(Package package)

- Replaces an old Package with new Package based on the “package” id.

- Returns true if successful, false otherwise.

1.6. Collection<Signatures> GetPackageSignatures(PackageID id)

- Returns a collection of signatures associated with the package identified by “id” throughout its travels or NULL if no such package exists.

1.7. String getUsernamePassword(String username)

- Returns the users encrypted password or null if the username doesn’t exist

1.8. AuthorityLevelID GetUserAuthorityLevel(String username)

- Returns the authority level identifier associated with the user identified by “username” or -1 if that user does not exist.

1.9. boolean setUsernamePassword(String username, String newPassword)

- Changes the password associated with a user to newPassword.

- Returns true if the operation succeeded false otherwise.

1.10. String findUsername(Address address, String FirstName, String LastName)

- Finds a user’s username based on their address and name.

- Returns false if a user was not found.

1.11. boolean updateUserInfo(UserInfo)

- Update’s a user’s name and address.

- Returns true if successful, false otherwise.

1.12. boolean addPackageEvent(PackageID id, Event event)

- Adds the contents of a package event to a packages history.

- Returns true if successful, false otherwise.

1.13. boolean addUser(String Username, String firstName, String lastName, Address addr, AuthorityLevelID accessLevel)

- Adds a new user to the system with the appropriate settings.
- Returns true if successful, false otherwise.

1.14. boolean removeUser(String Username)

- Removes a user from the system
- Returns true if successful, false otherwise.

1.15. UserInfo getUserInfo(String username)

- Returns the user's information if the user was found, null otherwise.

1.16. Itinerary GetPackageItinerary(PackageID id)

- Returns the package's itinerary if successful or NULL if no such package exists.

3. Locally defined data types

- Interface does not employ local data types.

4. Exception definitions

4.1. AccessException

- Thrown if the writer does not have the correct access level to modify or access data in the DB.

4.2. ConnectionException

- Thrown if there is no connection to the DB.

4.3. TimeoutException

- Thrown if the DB takes too long to respond to a request.

5. Variability provided for product lines

- Interface does not allow for special configuration.

6. Quality attribute characteristics

6.1. The quality attributes of this interface are heavily influenced by the underlying DB. Note that as the exact type of DB has not been selected yet, so the exact quality attributes cannot be specified precisely.

7. Element requirements

7.1. The underlying DB supports SQL.

8. Rationale and design issues

8.1. This interface provides a number of method calls to access data in the underlying database. It does this rather than acting as a direct conduit for SQL Queries in order to provide commonality between all database queries. This protects against malformed and incorrect queries. Also this provides security by allowing custom checks to be sure that the data sent to and from the database is correct.

9. Usage guide and protocols

9.1. SQL is used to execute commands on the underlying DB. However, the actual implementation of SQL might vary slightly from vendor to vendor.

Note that because a DB has not been selected for use yet the exact protocol used cannot be specified concretely.

B. Package Management Interface

1. Resources provided

   1.1. Package GetPackageInformation(PackageID id)

   - Returns the package identified by “id” if successful or NULL if no such package exists.

   1.2. boolean AddPackage(PackageInfo info)

   - Adds a package with shipping and receiving information contained in info to the DB.
   - Returns true if the operation was successful, false otherwise.

   1.3. boolean RemovePackage(PackageID d)

   - Removes a package from the system.
   - Returns true if the operation was successful, false otherwise.

   1.4. PackageHistory GetPackageHistory(PackageID id)

   - Returns the history for the package identified by “id” if successful or NULL if no such package exists.

   1.5. boolean UpdatePackageInformation(PackageID id)

   - Replaces an old Package with new Package based on the “package” id.
   - Returns true if successful, false otherwise.

   1.6. Itinerary GetPackageItinerary(PackageID id)

   - Returns the package’s itinerary if successful or NULL if no such package exists.

   1.7. boolean setPackageItinerary(PackageID id, Itinerary itin)

   - Associates a package itinerary with a package.
   - Returns true if successful, false otherwise.

   1.8. boolean BillUserAccount(BigDecimal amount, UserID id)

   - Executes a billing transaction for “amount” based on the user’s id number.
   - Returns true if successful, false otherwise.

   1.9. BillingInfo getBillingInfo(Package package)

   - Returns the billing information associated with this package.
   - Returns NULL if the package doesn’t exist.

   1.10. boolean notifyOfEvent(PackageID id, Event event)

   - Responsible for notifying users of shipping events related to their packages.
   - Returns true if user to be notified, or false otherwise.

   1.11. DeliveryRoute PlanDeliveryRoute(Package package)

   - Generates the package delivery route for “package” as it is exchanged between business package distribution centers.
   - Returns the delivery route if successful, NULL otherwise.

1.12. DeliveryRoute UpdatePlanDeliveryRoute(PackageID id, PackageHistory hist)

- Generates a new package delivery route for package “id” based on its shipping history.
- Returns the new delivery route if successful, NULL otherwise.

2. Locally defined data types

- Interface does not employ local data types.

3. Exception definitions

3.1. InvalidPackageException

- Thrown if a package being queried for does not exist.

3.2. NoItineraryException

- Thrown if no itinerary has been created for a package yet and someone has tried to retrieve it.

3.3. BillingException

- Thrown if there was a problem contacting the billing server.

4. Variability provided for product lines

- Interface does not allow for special configuration.

5. Quality attribute characteristics

5.1. Performance

- The relative execution time for any of the available resources is based on the priority of whether the request is coming from an employee or from a user and the current system load.

5.2. Availability

- The overall availability of this interface is partially based on the availability of the external billing service.

6. Element requirements

6.1. There must be an underlying connection to the external billing service.

7. Rationale and design issues

7.1. As the package management system is a core part of package tracking, this interface was created in order to limit the number of dependencies between the package management system and other subsystems.

8. Usage guide and protocols

8.1. Refer to Figure 6.A.1 to see how packages interact with this interface.

C. User Management Interface

1. Resources provided

1.1. UserInfo getUserInfo(String username)

- Returns the user’s information if the user was found, null otherwise.

1.2. boolean removeUser(String Username)

- Removes a user from the system.
- Returns true if successful, false otherwise.

1.3. boolean addUser(String Username, String firstName, String lastName, Address addr, AuthorityLevelID accessLevel)

- Adds a new user to the system with the appropriate settings.
- Returns true if successful, false otherwise.

1.4. boolean updateUserInfo(UserInfo info)

- Update's a user's name and address.
- Returns true if successful, false otherwise.

1.5. AuthorityLevelID GetUserAuthorityLevel(String username)

- Returns the authority level identifier associated with the user identified by "username" or -1 if that user does not exist.

1.6. boolean setUsernamePassword(String username, String newPassword)

- Changes the password associated with a user to "newPassword".
- Returns true if the operation succeeded, false otherwise.

1.7. String getUsernamePassword(String username)

- Returns the users encrypted password or NULL if the "username" doesn't exist

1.8. boolean setNotification(Notification condition)

- Adds a condition that can be triggered by a package event that is used to notify the user of some happening
- Returns true if the condition was set, false otherwise.

1.9. boolean removeNotification(Notification condition)

- Removes a notification condition.
- Returns true if the condition was removed, false otherwise.

2. Locally defined data types

- Interface does not employ local data types.

3. Exception definitions

3.1. InvalidUserException

- Thrown if an invalid user account is referenced.

3.2. InvalidAccess

- Thrown if a user does not have the proper access level for data access/modification.

4. Variability provided for product lines

- Interface does not allow for special configuration.

5. Quality attribute characteristics

5.1. Performance

- The relative execution time for any of the available resources is based on the priority of whether the request is coming from an employee or from a user and the current system load.

6. Element requirements

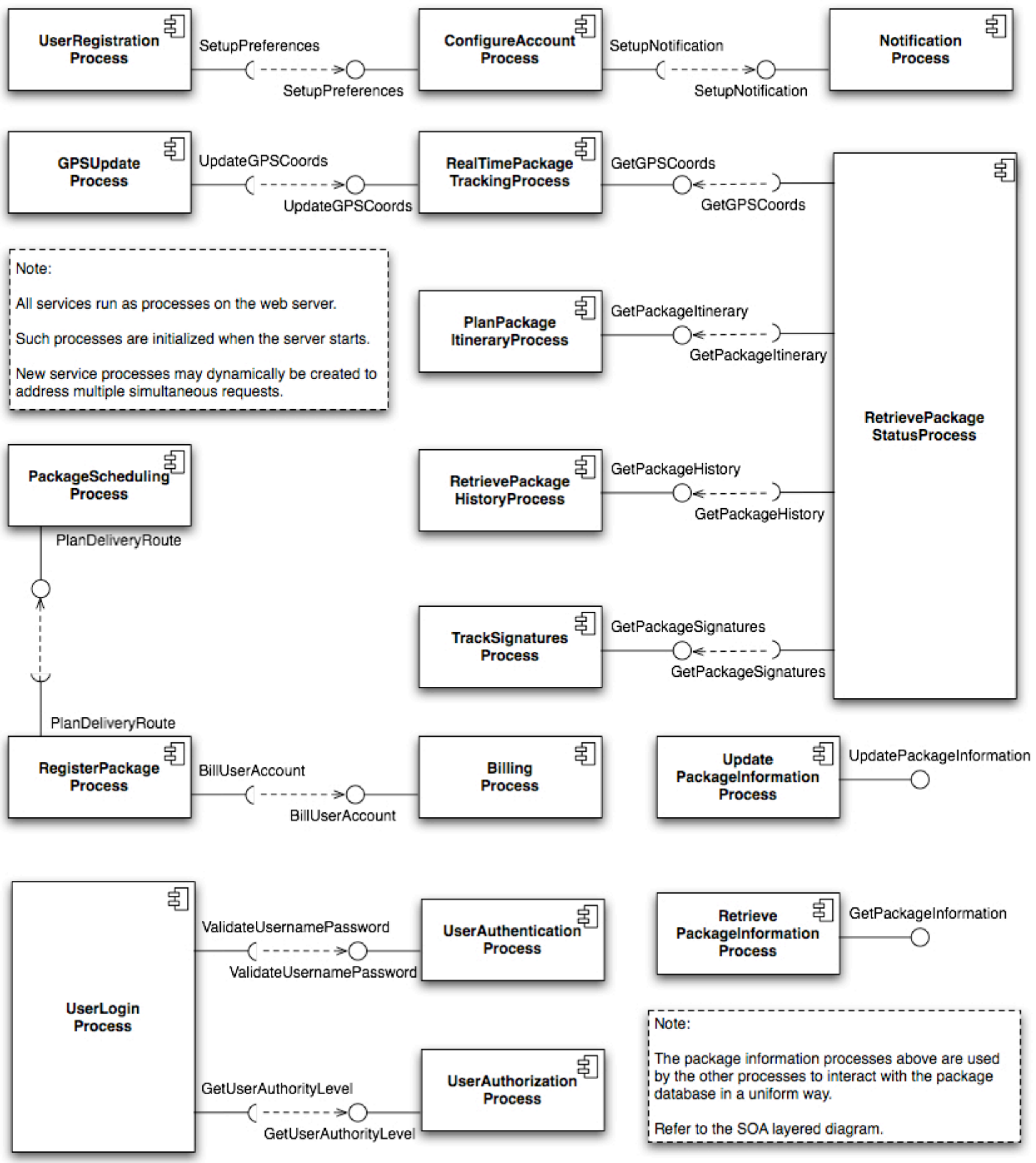
- There are no specific element requirements for this interface.

7. Rationale and design issues
    7.1. As the user management system is a core part of package tracking, this interface was created in order to limit the number of dependencies between the user management system and other subsystems.
8. Usage guide and protocols
    8.1. Refer to Figure 6.A.1 to see how users interact with this interface.

3. Context diagram
    3.1. Refer to the context diagram in section 10.2 of this document.
    3.2. Refer to the deployment diagram in section 6.C of this document. The real-time update system interacts with the GPS and DDA through a 3G networking protocol. The exact protocol has not been determined at this time and describing it is outside the scope of this document.
4. Variability guide
    4.1. The exact billing system middleware used in the package management system has not been determined yet. If an interface is designed to interact with this component, then the exact middleware used should not affect the rest of the system architecture.
    4.2. Note that the exact DB has not been determined yet. This decision should not influence other system components greatly as an interface has been provided for DB access.
    4.3. Note that the exact web technologies used to communicate through the HTTP protocol have not been determined at this time. This decision may influence system components.
    4.4. Note that the exact 3G networking protocol standard to utilize have not been determined at this time. This decision may influence system components used in the real-time update system package.
5. Architecture background
    5.1. This view stems from an overlaying service oriented architecture (see section 10.3). As such, the number of dependencies between the individual packages is limited and the package composition structure is decomposed into finer and finer grain components in order to avoid latency issues from components in one package having to spend too much time looking up other services from other packages.
    5.2. This view was chosen because it provides a way to see what system services require other services to be present for the first service to operate correctly in the system. This is very important in a SOA because if one service fails the remaining parts of the system may not operate correctly.

6.B. Process Component-Connector View

1. Primary Presentation



UserRegistration Process
SetupPreferences
SetupPreferences
ConfigureAccount Process
SetupNotification
SetupNotification
Notification Process
GPSUpdate Process
UpdateGPSCoords
UpdateGPSCoords
RealTimePackage TrackingProcess
GetGPSCoords
GetGPSCoords
Note:
All services run as processes on the web server.
Such processes are initialized when the server starts.
New service processes may dynamically be created to address multiple simultaneous requests.
PlanPackage ItineraryProcess
GetPackageItinerary
GetPackageItinerary
RetrievePackage StatusProcess
PackageScheduling Process
PlanDeliveryRoute
RetrievePackage HistoryProcess
GetPackageHistory
GetPackageHistory
TrackSignatures Process
GetPackageSignatures
GetPackageSignatures
PlanDeliveryRoute
RegisterPackage Process
BillUserAccount
BillUserAccount
Billing Process
Update PackageInformation Process
UpdatePackageInformation
UserLogin Process
ValidateUsernamePassword
ValidateUsernamePassword
UserAuthentication Process
Retrieve PackageInformation Process
GetPackageInformation
GetUserAuthorityLevel
GetUserAuthorityLevel
UserAuthorization Process
Note:
The package information processes above are used by the other processes to interact with the package database in a uniform way.
Refer to the SOA layered diagram.

Figure 6.B.1: Process View

2. Element Catalog

A. Elements

1. UserRegistrationProcess
   - Creates new user accounts and adds them to the system.
2. ConfigureAccountProcess
   - Modifies registered user account profile and preference information.
3. NotificationProcess
   - Generates notification messages to send to the user.
4. GPSUpdateProcess
   - Handles real-time GPS updates
5. RealTimePackageTrackingProcess
   - Uses GPS information to update package maps in real-time.
6. PackageSchedulingProcess
   - Uses location and existing package scheduling information to assign an employee, route, and timetable to a newly-registered package.
7. PlanPackageItineraryProcess
   - Uses the package scheduling information to generate easy-to-understand package itineraries.
8. RetrievePackageHistoryProcess
   - Generates package history reports based on the locations that packages have passed though and the times of certain events.
9. TrackSignaturesProcess
   - Obtains and presents information associated with electronic package signatures.
10. RegisterPackageProcess
    - Creates new package elements and adds them to the system.
11. BilingProcess
    - Creates invoices for business shipping services and bills users according to their billing preferences.
12. RetrievePackageStatusProcess
    - Obtains specific information about certain packages as requested by users.
13. UpdatePackageInformationProcess
    - Modifies specific information about certain packages in the package database.
14. RetrievePackageInformationProcess
    - Obtains specific information about certain packages from the package database.
15. UserLoginProcess
    - Handles the user login process, including encrypting the user password.
    - This process is destroyed when the logged-on user logs off of the system.
16. UserAuthenticationProcess
    - Verifies username and password combinations.
17. UserAuthorizationProcess
    - Determines a user's level of system access.

B. Relations

1. The connection between any two processes represents an attachment, or connection between two components. Using standard UML notation, one component provides an interface and another component requires the interface. Refer to the subsection C below for more details.

C. Interfaces

1. Shipping Tracking Inter-Process Communication Interface
2. Resources provided

2.1. boolean SetupPreferences(Preferences prefs)

- Replaces current user account preferences with "prefs".
- Returns true if successful, false otherwise.

2.2. boolean SetupNotifications(Collection<NotificationType> notifications)

- Replaces current user account notification list with "notifications".
- Notifications go into effect after the user logs off.
- Returns true if successful, false otherwise.

2.3. boolean UpdateGPSCoords(PackageID id, GPSCoords coords)

- Sets the GPS coordinates associated with the package identified by "id" to "coords".
- Returns true if successful, false otherwise.

2.4. GPSCoords GetGPSCoords(PackageID id)

- Returns the GPS coordinates associated with the package identified by "id" or NULL if no such package exists.

2.5. DeliveryRoute PlanDeliveryRoute(Package package)

- Generates the package delivery route for "package" as it is exchanged between business package distribution centers.
- Returns the delivery route if successful, NULL otherwise.

2.6. Itinerary GetPackageItinerary(PackageID id)

- Generates a package itinerary for the package identified by "id". The itinerary includes information about the package's delivery route, timetable, etc.
- Returns the package's itinerary if successful or NULL if no such package exists.

2.7. PackageHistory GetPackageHistory(PackageID id)

- Returns the history for the package identified by "id" if successful or NULL if no such package exists.

2.8. Collection<Signatures> GetPackageSignatures(PackageID id)

- Returns a collection of signatures associated with the package identified by "id" throughout its travels or NULL if no such package exists.

2.9. boolean BillUserAccount(BigDecimal amount)

- Executes a billing transaction for "amount" based on the user's billing preferences.
- Returns true if successful, false otherwise.

2.10. boolean ValidateUsernamePassword(String username, String encryptedPassword)

- Returns true if the username and password combination is correct, false otherwise.

2.11. AuthorityLevelID GetUserAuthorityLevel(String username)

- Returns the authority level identifier associated with the user identified by "username" or -1 if that user does not exist.

2.12. boolean UpdatePackageInformation(Package package)

- Replaces an old Package with new Package based on the "package" id.
- Returns true if successful, false otherwise.

2.13. Package GetPackageInformation(PackageID id)

- Returns the package identified by "id" if successful or NULL if no such package exists.

3. Locally defined data types

- Interface does not employ local data types.

4. Exception definitions

- Interface does not explicitly utilize any noteworthy exceptions.

5. Variability provided for product lines

- Interface does not allow for special configuration.

6. Quality attribute characteristics

6.1. Performance

6.1.1. The relative execution time for any of the available resources cannot be guaranteed because it is dependant on the system load and prioritized scheduling.

6.2. Reliability

6.2.1. The success of performing operations, such as BillUserAccount, that rely on third-party middleware cannot be guaranteed.

7. Element requirements

7.1. There must be an underlying interface to the relational database where user and package information is stored.

8. Rationale and design issues

8.1. Each service in an SOA runs as an independent process on the server. Processes interact through defined interfaces in order to share information and provide simple behaviors that are utilized in conjunction to perform more complex behaviors.

8.2. In the future, if new services are added to the system, this interface may be modified to address the interactions between the new and existing service processes.

9. Usage guide and protocols

9.1. Refer to the SOA Layered Diagram in section 10.3 of this document to understand the interactions between service processes.

3. Context diagram
   3.1. Refer to the context diagram in section 10.2 of this document. The service processes interact with the client machine browser through the HTTP protocol. More specifically, a combination of Javascript, AJAX, XML, and other web technologies may be used. The exact technologies have not been determined at this time.
   3.2. Refer to the deployment diagram in section 6.C of this document. The service processes interact with the GPS and DDA through a 3G networking protocol. The exact protocol has not been determined at this time and describing it is outside the scope of this document.
4. Variability guide
   4.1. The creation of the system services should occur when the system is initialized. The number of each type of service to be instantiated should depend on the ability of the underlying hardware and the expected system load. New processes can be instantiated as necessary to dynamically address load increases.
   4.2. Note that the exact web technologies used to communicate through the HTTP protocol have not been determined at this time. This decision may influence system components.
   4.3. Note that the exact 3G networking protocol standard to utilize has not been determined at this time. This decision may influence system components.
5. Architecture background
   5.1. Each service in an SOA runs as an independent process on the server. Because each service runs as a separate process, each process can be instantiated multiple times on the server to service multiple requests concurrently. Of course, careful attention must be paid to synchronizing these processes correctly in order to avoid read/write issues and resource contention.
   5.2. Each service process provides one core resource (behavior). Processes could have been developed to provide a subset of related behaviors. This would reduce the number of processes running in the system at any one point in time, thereby reducing system complexity, however, it would also mean that such a process could only perform one behavior in its subset at a time, thereby reducing performance. This was decided against since performance is a primary architectural driver.
   5.3. This view was chosen because it provides a way to see what system service processes interact in real-time and the way that such processes interact trough the inter-process communication interface.

6.C. Deployment Allocation View

1. Primary Presentation

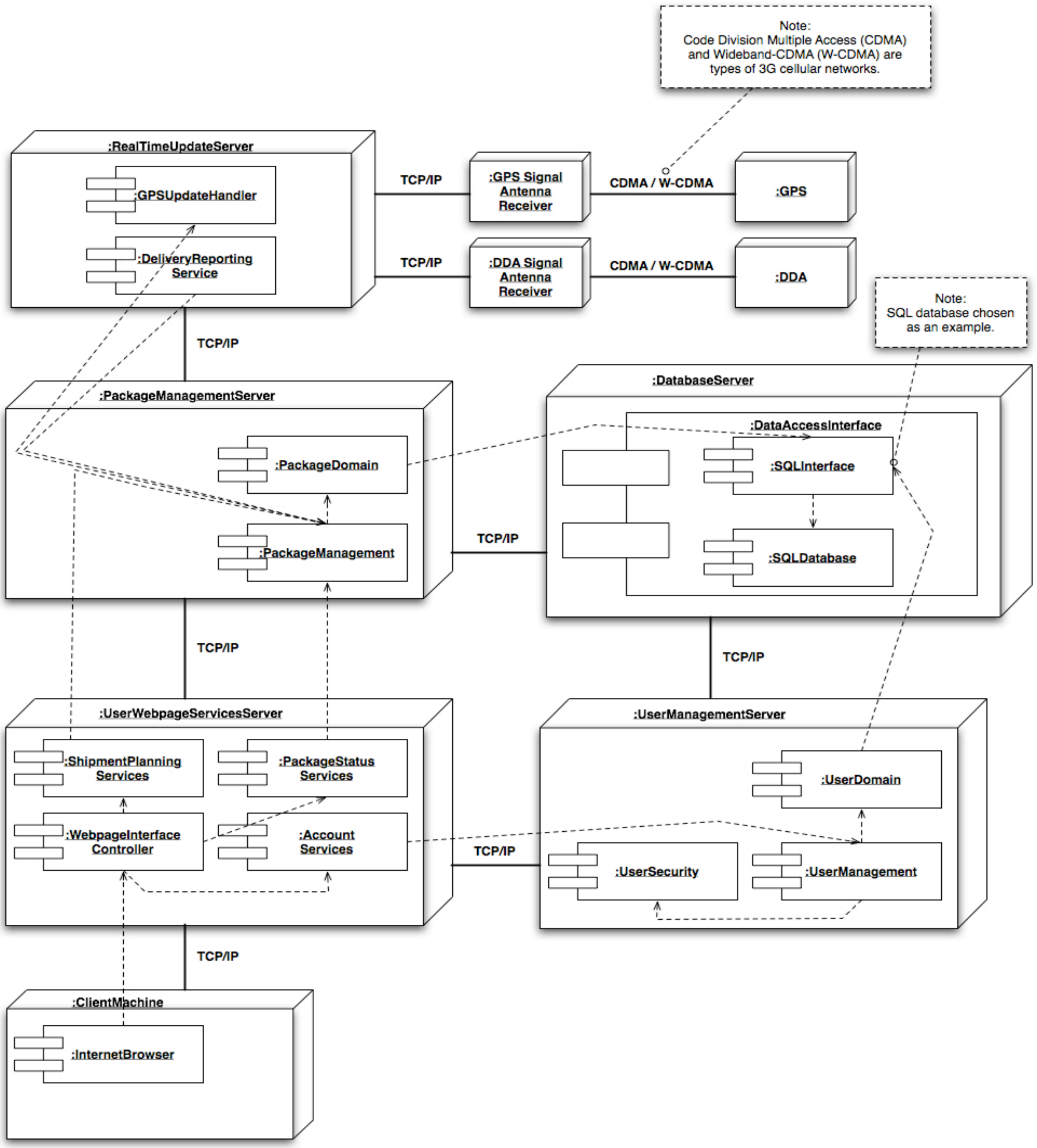


Figure 6.C.1: Deployment View

2. Element Catalog
   A. Elements
      1. RealTimeUpdateServer
         - Aggregates and receives regular updates from all of the GPS devices on the established delivery trucks, as well as delivery persons' DDAs.
      2. GPSSignalAntennaReceiver
         - Communicates with all of the GPS devices on the delivery trucks. The RealTimeUpdateServer receives event interrupts from the receiver to gather location information about all of the trucks.
      3. DDASignalAntennaReceiver
         - Communicates with all of the DDAs utilized by delivery personnel. The RealTimeUpdateServer receives event interrupts from the receiver to gather location information about package shipments.
      4. PackageManagementServer
         - Responsible for correlating location and delivery data received from the RealTimeUpdateServer and the DatabaseServer. Specifically, this server will update location and delivery information when it is received from the RealTimeUpdateServer. UserWebpageServicesServer obtains package information from this server.
      5. DatabaseServer
         - Keeps track of persistent data about each package and user.
      6. UserWebpageServicesServer
         - Acts as the aggregation point for all clients connecting to the system. The system is web-based so this server is responsible for providing the user interface to the system. This server also hosts the processes that initiate functional behavior on the PackageManagementServer and UserManagementServer.
      7. UserManagementServer
         - Responsible for user authentication and managing information about user accounts. This server also limits or grants permissions to perform certain tasks in the system based upon the authenticated users' credentials.
      8. ClientMachine
         - The end user's machine. Package information stored in the user's machine is very sparse; its primary purpose is to display the user interface to the system using a web browser. It does not perform any data modifications or calculations, rather, it requests services on the UserWebpageServicesServer.
   B. Relations
      - Note that software elements are allocated to each of the following hardware devices. These elements are shown within the hardware bounding boxes in the above figure. These software elements map to software modules (see section 6.A for more detailed information).
      1. RealTimeUpdateServer
         - This subsystem relies upon the receiver antennas to gather information about the package and delivery truck locations. Additionally, the PackageManagementServer relies upon this subsystem for location information.
      2. PackageManagementServer
         - Gathers information from the RealTimeUpdateServer and checks this information against the information contained in the DataBaseServer.

3. DataBaseServer
   - Used as a central storage facility for both the User Management Server and Package Management Server.
4. UserWebpageServicesServer
   - Relates to the ClientMachine, PackageManagementServer, and UserManagementServer. The UserManagementServer and PackageManagementServer are responsible for providing the functionality that the UserWebPageServicesServer implements. The ClientMachine invokes such behavior through this subsystem.
5. UserManagementServer
   - This subsystem verifies the information given by the web server and the retrieved credentials from the database. The UserWebPageServicesServer is then informed of the appropriate access credentials.
6. ClientMachine
   - The client machine is the end user's computer. End users access the web server through a web site.

3. Context diagram
   3.1. Refer to the context diagram in section 10.2 of this document. The service processes interact with the client machine browser through the HTTP protocol. More specifically, a combination of Javascript, AJAX, XML, and other web technologies may be used. The exact technologies have not been determined at this time.
   3.2. Refer to the deployment diagram in section 6.C of this document. The service processes interact with the GPS and DDA through a 3G networking protocol. The exact protocol has not been determined at this time and describing it is outside the scope of this document.

4. Variability Guide
   4.1. The creation of the system services should occur when the system is initialized. The number of each type of service to be instantiated should depend on the ability of the underlying hardware and the expected system load.
   4.2. The architecture allows for the addition of extra hardware to offset computational load that may be incurred by the system. For example, the UserWebpageServicesServer can employ server clustering to achieve better performance and the ability to service more users concurrently.

5. Architecture Background
   5.1. The deployed system is the result of a successful implementation of the SOA. Each different package does not make assumptions about the hardware that each package is running on. As a consequence special purpose hardware is not required and generic hardware can be added to the system to meet performance goals.
   5.2. Each component communicates through a TCP/IP medium (or some variant of CDMA for receiver antennas). As a consequence, the system can be split amongst different servers. This allows scalability and reliability to be easily added to the system during deployment. Scalability can be achieved because extra hardware can be added to meet increasing demand. Reliability can be achieved because each different service can be cloned on different sets of hardware, and switched in or out in case of a hardware or software fault.

5.3. This view was chosen because it provides a way to see which software elements are mapped to which hardware elements. Based on this, it is easy to see the communication pathways between hardware elements and they way in which such communication affects the interaction of software elements.

7. Framework

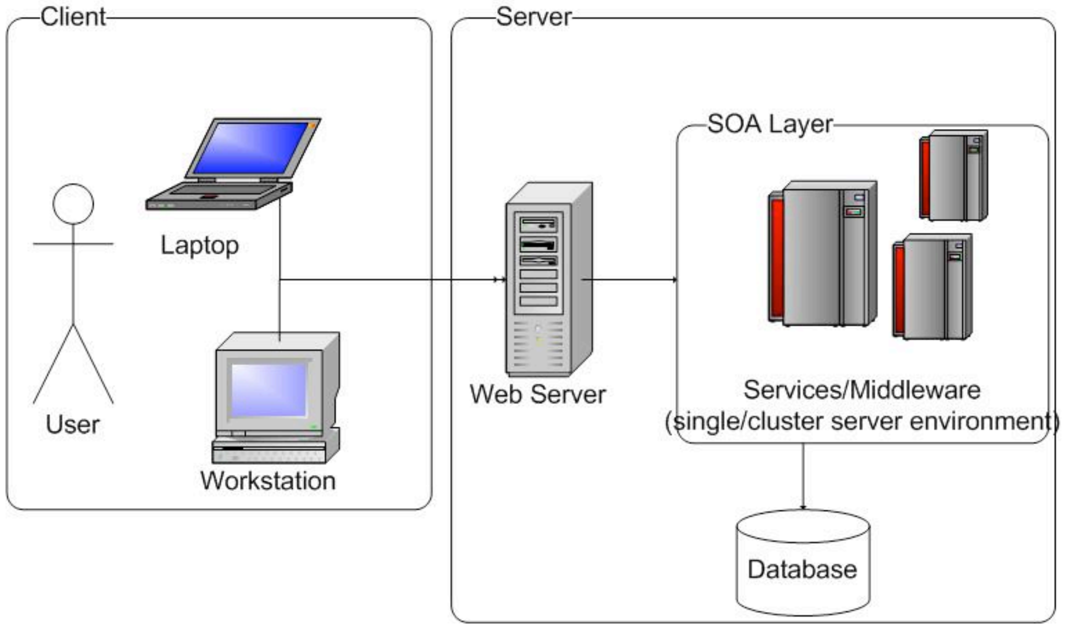


Figure 7.1: Framework Context Overview

In general, the shipping tracking system framework is organized as a combination of a client-server architecture and a service-oriented architecture. In accordance with the client-server architecture, as seen in the figure above, a centralized web server allows content to be served to multiple clients residing at remote locations at the same time. The client side is composed of a dynamic web interface viewed through a web browser. Content is generated on the server and delivered to the client through the TCP/IP protocol. The server side is composed of various software elements that reside on various hardware devices. These internal components communicate internally and externally using the TCP/IP protocol.

In accordance with the service-oriented architecture, the server side uses services to handle user requests. Each service runs as a process. In turn, multiple services can be spawned as needed, allowing for a high amount of concurrency. Thus, system performance becomes a matter of hardware constraints. Considering that the system is expected to be under a high volume of use, it is convenient to simply upgrade hardware when required. Refer to Figure 6.B.1, the process component-connector view, for a diagram that outlines the interaction and process flow of the services in the system. Also, please refer to Figure 10.3.1, the SOA layered diagram, for an illustration of how the services are layered and their interaction within the system.

The server side of the shipping tracking framework can be divided into a number of basic components. Each of these components can be directly mapped to packages in the module view (see Figure 6.A.1) based on the specific type of shipping tracking system being developed. This is shown in the following diagram and component descriptions:

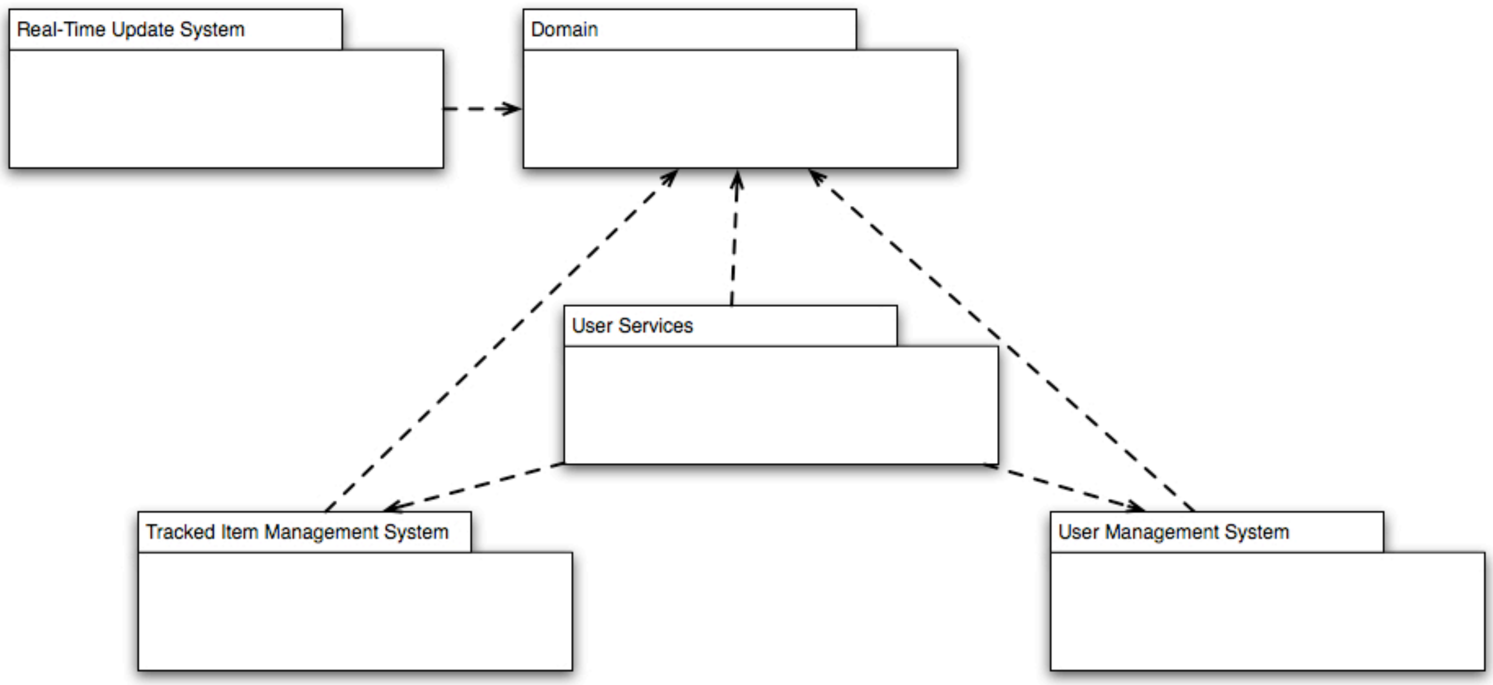


Figure 7.2: Framework Server Components

1. Real-Time Update System
   - Handles real-time update events and updates domain information based on these events.
   - May interface with external systems to receive notification of update events, such as those from a GPS locator.
   - May be used directly by the User Services component to provide real-time services, such as tracked item mapping.
2. Domain
   - Represents the data associated with the users of the system, the tracked item, and other necessary information.
   - May interface with a database to manage data.
3. User Services
   - Provides all of the services requested by users of the system.
   - Higher-level processes manage multiple lower-level processes to perform complex tasks.
   - Utilizes the Tracked Item Management System and User Management System to perform specific functional behavior associated with tracked items and users, respectively.
4. Tracked Item Management System
   - Manages the functional behavior of the system in regards to the tracked item information.
   - May notify users of changes to tracked item state.
   - May interface with external systems that need to know about the tracked item state, such as billing systems.
5. User Management System
   - Manages the functional behavior of the system in regards to user information.
   - May authorize and authenticate users of the system if necessary.

This division of server components helps facilitate the future expansion of the system via the client-server and service-oriented architectures. Note that the server component used to interact with the client user interface is separate from the functional system modules shown in the above figure. Such a component interfaces with the User Services component in order to provide the data needed to perform service requests and retrieve the data needed to respond to service requests. Clearly, as new technologies are developed for web interfaces and introduced into the system, the interactions between the server components in the figure above should remain the same. If a change needs to be made, it can be made to the User Services component by providing a specific service.

As mentioned, following the service-oriented architecture, new functionality can be introduced into the system as a new service. This requires that a new service module be developed for the User Services component. In turn, this module utilizes the Tracked Item Management System and User Management System components to perform the scheduled services. In this way, a service process simply becomes a series of tasks to call on the two aforementioned system components. Also, note that a separate Tracked Item Management System component can be developed for each type of tracked item in the system. This allows the system to handle each type of tracked item in a different way, if necessary. Additionally, many shipping tracking systems can use the same User Management System component if they share the same user base.

With service-oriented architectures, scalability and performance issues are also addressed by using services. Services run as processes, which can be instantiated as many times as needed to handle client demands. The services can also be distributed among a cluster of servers, which addresses load balancing and stability issues. Note that there is the possibility of a chain reaction occurring when one service fails to respond. Exception handling and fault prevention mechanisms will be in place to ensure that if a service does fail to respond it does not cause the rest of the system to fail. The implementation of the system using the framework needs to address synchronization among the services. Because each service can be spawned as a new process dynamically, special care must be taken to ensure that simultaneous read/write issues do not occur and that resources are managed appropriately. The User Services component is responsible for ensuring that service processes are created and scheduled appropriately without collision.

Another benefit of this framework is that the Real-Time Update System component can modify the Domain component information without interrupting the User Services component. This way, if one of those components fails, it will not affect the behavior of the other component. The Domain component is responsible for ensuring that read/write issues do not occur in the way that these two components access and modify the tracked item and user data.

8. Acknowledgements

We would like to thank the stakeholders involved with the shipping tracking software for their time and for providing our development group with the opportunity to serve their needs.

We would also like to thank Mark Secrist from Hewlett Packard for authoring his paper on service oriented architecture concepts. Secrist's work was a major influence on the design of the shipping tracking system architecture.

9. References

The following websites were used for product and feature comparison purposes and for service-oriented architecture guidelines:

Google. (2007). Google Maps. Retrieved April 12, 2007 from http://maps.google.com/maps

Secrist, Mark. (January 2006). SOA Concepts. Hewlett Packard Development Company, L.P, White Paper.

United Parcel Service of America, Inc. (1994-2007). UPS United States Tracking. Retrieved April 12, 2007 from http://www.ups.com/tracking/tracking.html

10. Appendices

10.1. Glossary

3G – 3rd Generation standard of mobile phone and wireless technologies
AJAX – Asynchronous Javascript and XML
CDMA – Code Division Multiple Access form of 3G signal modulation
DB - Database
DDA – Delivery-person Digital Assistant
GPS - Global Positioning System
HTTP – Hyper Text Transfer Protocol
Real-Time - Processing on a system which responds immediately to user instructions
SOA – Service Oriented Architecture
UML – Unified Modeling Language
UPS - United Parcel Service
W-CDMA – Wideband Code Division Multiple Access form of 3G signal modulation
XML – eXtensible Markup Language

10.2. Context Diagram

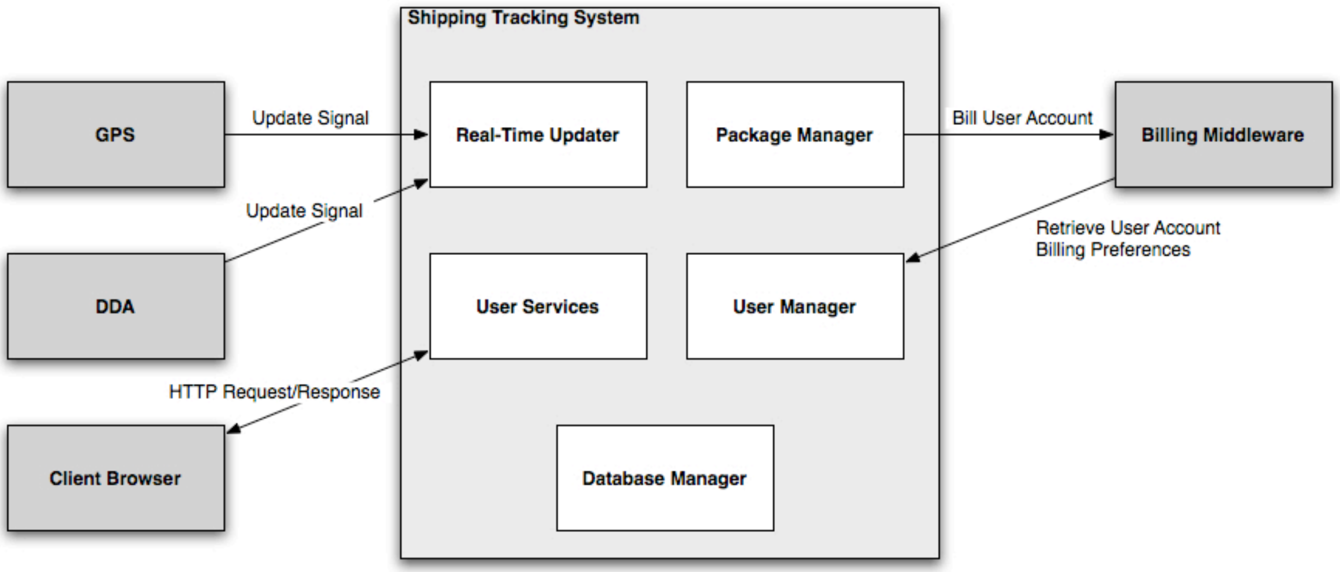
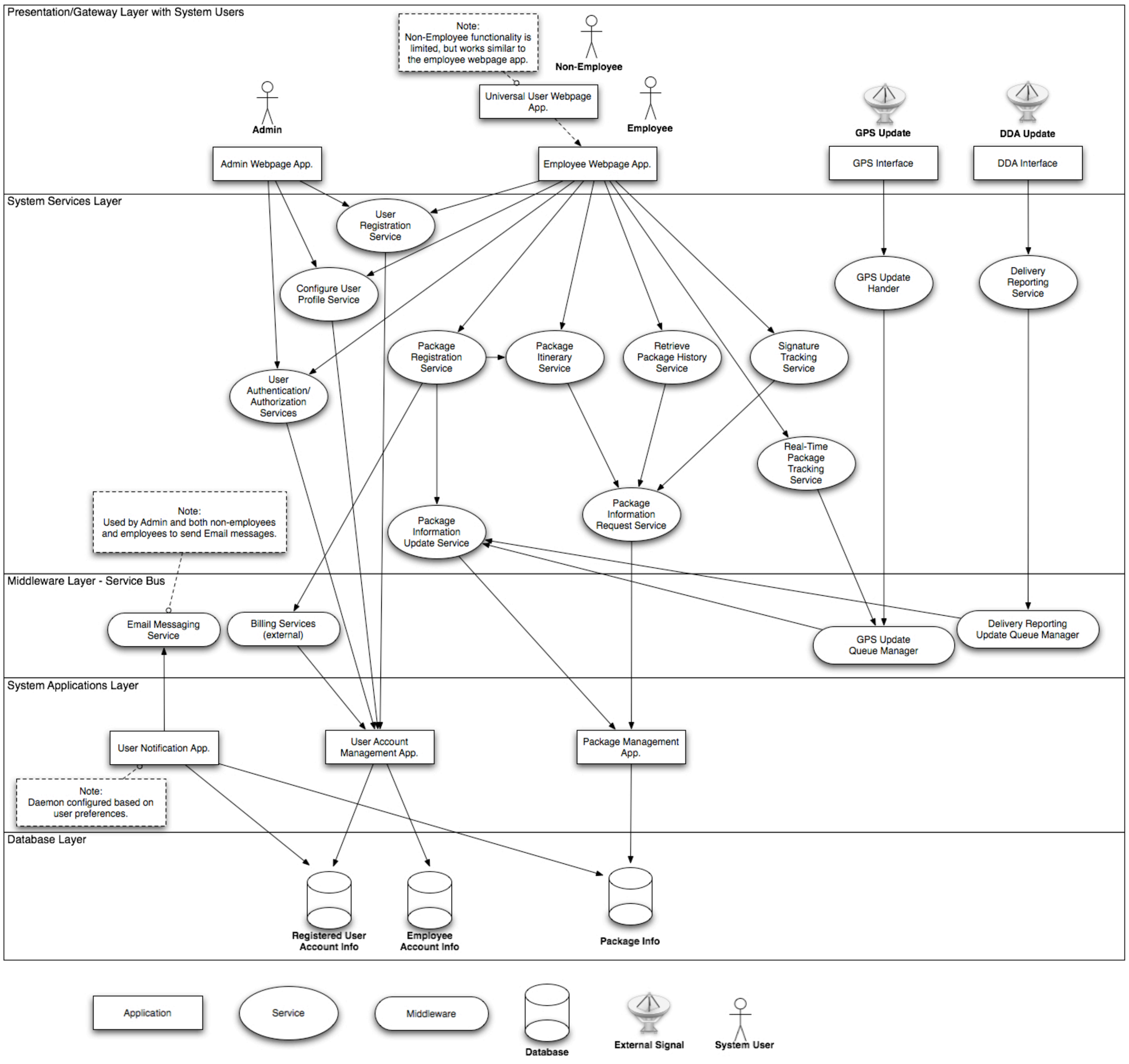


Figure 10.2.1: Context View

The above diagram shows how the core web-based shipping tracking system relates to external systems. The boxes inside the shipping tracking system bounding box represent subsystems and system modules and the dark gray boxes to the left represent external entities. Essentially, the system must interact with client browsers through the HTTP protocol, interact with external billing middleware, and interact with GPS and DDA systems in order to receive real-time updates.

10.3. SOA Layered Diagram



Presentation/Gateway Layer with System Users
Note: Non-Employee functionality is limited, but works similar to the employee webpage app.
Non-Employee
Universal User Webpage App.
Admin
Employee
GPS Update
DDA Update
Admin Webpage App.
Employee Webpage App.
GPS Interface
DDA Interface
System Services Layer
User Registration Service
Configure User Profile Service
GPS Update Hander
Delivery Reporting Service
Package Registration Service
Package Itinerary Service
Retrieve Package History Service
Signature Tracking Service
User Authentication/ Authorization Services
Real-Time Package Tracking Service
Note: Used by Admin and both non-employees and employees to send Email messages.
Package Information Update Service
Package Information Request Service
Middleware Layer - Service Bus
Email Messaging Service
Billing Services (external)
GPS Update Queue Manager
Delivery Reporting Update Queue Manager
System Applications Layer
User Notification App.
User Account Management App.
Package Management App.
Note: Daemon configured based on user preferences.
Database Layer
Registered User Account Info
Employee Account Info
Package Info
Application
Service
Middleware
Database
External Signal
System User

Figure 10.3.1: SOA Layered Diagram

The above diagram shows how shipping tracking system elements relate to the various layers of service oriented architecture. Note that nearly all of the services listed in the systems services layer map to system modules and processes (see Figure 6.A.1 and Figure 6.B.1, respectively). Also, note that many of the variabilities associated with the system are associated with the choice of middleware applications used for billing, email messaging, and queue managing. System variabilities also include the way in which the system interfaces with external entities in the gateway layer. The protocols used to communicate with external entities are shown in the deployment view (see Figure 6.C.1). In general, this diagram shows how various users interact with the system and the flow of system element interactions. The same information found in this diagram is represented by the various views in section 6. Please refer to that section for details.