Oracle Financial Services
Revenue Management and Billing
Technology Architecture
Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Introduction

Oracle Financial Services Revenue Management and Billing is a modern billing application for banking, insurance, and health care verticals. The application is built on a 100% Service Oriented Architecture (SOA). It supports client business rules through configuration and customization, while still maintaining an upgrade path.

Architecture Overview

The product is a Java J2EE, N tier application built on an enterprise Oracle database with a web application server middle tier. All interaction between the client and the application is web based, through an Internet Explorer web browser. There is no fat client or backend “green screen” access required. The application supports a variety of hardware and Operating System choices, including Linux and Windows on commodity (x86) blade or rack mount servers, in addition to IBM AIX on POWER hardware.

Figure 1. Architecture Overview

The product separates the web application tier into business logic and user interface layers. Each layer is responsible for providing either user interface code or business logic. Some client side validation is performed, such as validating mandatory fields are not blank, reducing trips back to and processing on the server. All client side validation is re-validated on the server to combat injection and other attacks.
The business logic layer performs all business rules, validation, etc. Data access code is contained solely within the business logic layer, and is consolidated to manage all database access from a single code location. This allows effective database pool management and prevents open database connections from being abandoned in various places across the application. The application’s database is used strictly as a data store. No business logic, stored procedures, or PL/SQL are used.

Web Services and Integration

Oracle Financial Services Revenue Management and Billing supports integration with both modern and legacy systems, in order to meet both existing and future integration requirements. The application is built from the ground up around a completely Service Oriented Architecture (SOA). The web service layer is not bolted on after the fact, but is instead a core component of the application architecture.

Everything that you can see (data) and do (functionality) within the application is accessible through real time, standards based, XML web services. This means that everything accessible through the user interface is also accessible to back-end services and system integrations. The web service API is publically exposed, with all web services containing WSDL descriptors. Theoretically, it is possible to “throw out” the entire user interface and rewrite it using the web service API, since all needed functionality is exposed and available. The web service API can also support real time messaging protocols, such as MQ or Java JMS messaging.

Figure 2. Integration Architecture

The application also supports legacy integration through batch file transfer. Batch transfers can be either through XML files, for integrations with systems or middleware adapters that support XML but
do not support real time web services, or through standard flat files. The application includes a translation adapter for both XML and flat files, such as Comma Separated Value (CSV). Batch processing, both for file I/O and for internal processes such as bill generation and collections calculation, is scheduled in batch but is performed in real time. The system is still fully up and available for usage during batch processes, and batch runs can occur at any frequency or time of day, even running hourly or more often to provide near real time performance for certain processes. Batch processes can also be executed on separate hardware from online users to ensure performance during batch operations. Batch jobs are threaded based on the size of the system hardware to ensure efficient operation of the batch process. The application includes a batch scheduler for starting and monitoring batch jobs, or can be integrated with standard schedulers such as IBM Tivoli.

Files can be both imported and exported using the adapter and integration layer. XML files can be translated using an XSD schema. Flat files can be mapped on a per field basis within the application. An external ETL or middleware tool, such as Oracle Data Integrator, can also be used. After translation, the data is staged within dedicated staging tables of the application’s database. From here, the data is loaded into the application third normal form main transactional data tables.

Web services within the application use the XML Application Integration (XAI) capability. The application uses Hibernate for object relational mapping between the relational database and the XML object oriented XAI layer. This allows business objects to be created and mapped to the database using XML object description. This can then further be exposed by creating XAI inbound services within the application, which map XML web services to the XML business objects. The XAI inbound service also creates the WSDL descriptor automatically for each web service.

Figure 3. Functional Overview with Integrated Systems (Oracle Revenue Management and Billing in Red)

The application is typically integrated with a wide variety of other systems using the methods mentioned previously. The above diagram represents the Oracle Revenue Management and Billing application in red boxes, showing the major functional areas of the application. The boxes in grey represent common external systems that would be integrated with the application. Moving from left to right and top to bottom, we start with administrative systems which feed account, product, price, transaction, usage, and other relevant information into the application. This can include sales, transaction, underwriting, CRM, and other similar systems. For transaction feeds, the application has a
Transaction Feed Management function that performs validation, mapping, and aggregation of incoming transaction data. The application consolidates this information to consolidate customer hierarchy, account hierarchy, products, prices, and usage.

Next, the application produces bills from this consolidated data. From here, raw bill data is normally sent to one or more systems. An invoice print formatter, such as Oracle Documaker, translates raw bill data into printable PDF views of the bill which can be printed, mailed, or displayed electronically. Bill data can also be sent, in conjunction with the PDF viewable bill, to a self service system designed for end customer use.

The application includes a web self service module, which can be used for Electronic Bill Presentment and Payment (EBPP). It can be used to display bills, pay bills online through credit card or bank account, update account information such as customer address, and other similar, end-customer facing functions. The module is written using Java Server Pages (JSP) and the source code is included, so it can be customized and extended by customers. This can include adding new functionality, such as extending bill dispute capabilities, as well as integrating the source code into an existing web portal to match the look and feel of that existing portal.

The web self service module leverages the built in XML web service API of the application, making real time web service calls to execute all functionality. Additional web service calls of the API can be called during customization of the web self service module. It is also possible to use the web self service as a template, leveraging the used web service calls to build similar functionality into an existing web portal.

Following billing and collections, the application performs financial calculations, typically serving as the Accounts Receivable (A/R) and a sub-ledger to the General Ledger (G/L). Data comes in from financial payment processing systems, such as bank lockboxes, credit card processors, and others. The application processes this data and generates sub-ledger transactions, which are fed to a G/L. It can also calculate payments, such as taxes, commissions, and refunds, and feed this data to an Accounts Payable (A/P) system for payment. The G/L and A/P integrations for Oracle E-Business Suite and PeopleSoft financials typically utilize Oracle Data Integrator templates. These templates allow rapid integration between the application and Oracle Financials, with minimal setup such as entering accounting distribution codes which are unique to each installation.

Finally, the application can integrate with reporting tools, such as Oracle Business Intelligence. Reports can be generated in the application natively, and then fed to reporting tools to further analyze and display in different formats. The database can also be fed into a datamart, where it can be reported by these reporting tools without impacting or using production resources of the application.

Configuration and Customization

Oracle Financial Services Revenue Management and Billing supports modification to meet client needs through a wide combination of configuration and customization options. The application is generally built on a “configuration, not customization” approach, which allows most business logic to be configured through the web interface by a business user rather than requiring customization, or coding by an IT user. In the event that configuration is not sufficient to meet business needs, the application allows a variety of customization options, including writing custom Java code. Even with configuration and customizations, including custom Java code, the application is still upgradeable, due to the method in which modifications are made and managed, preventing clients from being ‘stuck’ on previous versions.
The application’s User Interface can be configured through the use of Zones, which are panels on the screen that can be configured to display specific data. Zones can be based on users, so each user or user role can have a different view of data, based on their job and needs. Zones allow drag and drop configuration, which allows end users to modify the fields shown in a Zone on the fly. Further configuration of Zones is possible by writing custom SQL queries. Zone data can be exported to Excel and other reporting tools for ad-hoc and real time reports.

Application configuration is achieved through the user interface and can be performed by business users to modify application functionality and business rules. For example, to change a payment plan from billing on the 1st of the month to the 15th of the month, a business user would simply change the bill date field from 1 to 15. The application configurations are stored as metadata within the application’s database. The application contains separate tables for configuration data and transactional (customer) data. During an upgrade, this configuration data is retained in the configuration database tables, so business rules are carried over to the next release.

In the event a business rule cannot be configured within the system, Technical Configuration can be utilized to script new, custom business rules. Technical Configuration leverages XPath, a standards based scripting language to create Plugins to execute new business logic. For example, if a client wanted to change a business rule from billing an account on the 1st of the month to billing on the 15th, unless that day was a Sunday, and then billing on the following Monday, the user can write an XPath script to perform the calculation in the application as a Plugin to execute when evaluating the bill date. The XPath script is stored in the application’s database configuration tables. Therefore, during an upgrade the script and associated functionality is preserved.
In the event that Configuration and Technical Configuration (XPath plugins) are not sufficient, the application allows Java code customization. This allows completely new and different functionality to be added to the application. Rather than modifying base Java code, which would negatively impact or preclude the upgrade path, the application supports class extensions. These extensions allow Java extended classes to be written on top of the existing application base code.

The class extensions are not just logically separate from the base code, since they extend rather than replace base code. They are also physically separate, being stored in separate JAR files from the base code on the hard drive. During an upgrade, base code classes and JAR files may be overwritten, but the class extensions and associated JAR files are not. While it is possible to deliberately write a class extension that will break when the base class is updated, normally, updates to the base classes do not negatively affect extended classes. When an upgrade includes modifications to a class which has been extended, this gives an indication that testing of the extended functionality should be performed following the upgrade.

Characteristics are to the application’s database what Class Extensions are to the application. Characteristics allow an unlimited number of custom fields to be added to the application, extending the data model without changing the database schema. This allows the application to be upgraded without concern for custom data model changes.

Characteristics allow new, custom data fields to be added to customers, accounts, and a wide variety of other objects within the application. Characteristics are easily added within the User Interface by selecting the object to which the Characteristic is associated, naming the Characteristic, and choosing the data type (free form text, pick list, or validation). The Characteristic is stored as metadata in the application’s database, rather than as an actual change to the database schema. The application has separate tables for storing configuration and Characteristic data. New Characteristics are stored as rows within the Characteristics configuration table. This allows an unlimited number of Characteristics, no change of the database schema, and no impact on upgradability regardless of the number of custom fields added.

The application includes a tool called ConfigLab, which is used to migrate changes between environments. ConfigLab identifies metadata differences between environments and generates the necessary SQL statements to migrate the selected changes. ConfigLab also recognizes the differences between customer transactional data and configuration metadata.

Figure 5. ConfigLab Functionality
For example, ConfigLab can be used to select a sample of data from a production system, obfuscate it by replacing real sensitive information, such as credit card numbers, with fictitious data, and then load the data into a test environment. This would allow testing on realistic, but not real or sensitive, data. ConfigLab can also be used to identify configuration metadata changes made in a development environment, such as business rule configurations, data characteristics, XPath plugin algorithms, or other configuration changes, and promote them to a test environment. Following testing, ConfigLab can then be used to promote the changes to production. ConfigLab in this case only migrates configuration metadata, not customer data, as that should never be moved from a test to a production environment.

ConfigLab actually generates SQL statements to update the target environment. These can either be applied directly or they can be extracted, analyzed, and applied through a Configuration Management (CM) tool or process. This allows all changes to be CM managed in accordance with existing client requirements.

Security

Oracle Financial Services Revenue Management and Billing’s security architecture is designed from the ground up to meet the needs of the most demanding clients, such as banks and insurers, and legislation, such as SOX and HIPAA. The application has been successfully and securely deployed in these environments.
Securing the application starts with Authentication (verifying who is accessing) and Authorization (validating what they are permitted to access). The application integrates with an enterprise directory server, such as LDAP or Microsoft Active Directory, for Authentication. This allows central management and common logons for all users. The application can support standard username and password authentication, in addition to more advanced two factor methods such as tokens or cryptographic certificates.

The application uses a robust Role Based Access Control (RBAC) model for Authorization. The application maps users to roles and roles to privileges. Privileges are highly granular, and can be used to determine both permissions (read, write) and areas (activities, screens, views, fields, etc.). For example, a client might configure specific roles based on their business and security needs:

- Accounting: has read-only views of financial data, and cannot view or modify customers.
- Broker: can only view assigned customers and only modify certain fields.
- Collections: can only perform collections tasks and requires a supervisor approval to write-off more than a set level of debt.

The product complies with relevant legislation and standards, such as SOX, HIPAA, and PCI. Key database fields, such as credit card, bank, and social security numbers can be encrypted in the database using Oracle Advanced Security, for PCI and HIPAA compliance. The application supports partial or full masking of these sensitive fields in the user interface, as well. The application supports encryption in transit through standards such as SSL, VPN, and IPsec tunnel.

The application also supports configurable audit logging for compliance and accountability. Audit logs include the date/time of the change, the user making the change, the previous value, and the new value.

### Availability and Scalability

The application supports both availability and scalability, without any architectural limit. It can be scaled both vertically, by adapting to any size of hardware, as well as horizontally, through Oracle Real Application Clustering (RAC) of the database layer. The web application does not require clustering, as the application is generally stateless, and hence additional nodes can simply be added without clustering. There are no architectural limits on the scalability of the application.

An Oracle Exadata benchmark test was conducted which billed 100 million accounts in 12 hours and 1 billion transactions in 5 hours. Performance was found to scale linearly to the hardware on which the application is deployed. A questionnaire is typically filled out to collect client metrics, and from there a hardware calculator is used to determine the required hardware for given metrics and performance requirements for each client. Clients on the application framework have used the application in production to bill tens of millions of accounts and billions of transactions. Processing performance and response time are dependent on hardware, and hence can be targeted when selecting hardware based on metrics that need to be met.

The Oracle RAC mechanism can also be used for availability, by creating hot-cold, hot-hot, or other failover, redundant, clustered environments. The application can also support Oracle Data Guard for replication, to allow offsite replication for hot-cold systems. These technologies can be combined to
use both clustering and offsite replication, or only one mechanism can be used, depending on requirements. With a clustered environment, failover is immediate with no data loss, meaning zero Recovery Time Objective (RTO) and Recovery Point Objective (RPO). With a replicated environment, RTO is near zero if the failover instance is active, and the RPO is based on the latency of the data replication between environments.

Fusion

The application incorporates Oracle Fusion concepts, which center specifically around two key areas: integration and training. First, the application has open, standards based integration methods, described previously, including XML web services and XML file integration. These open integration methods are central to Oracle Fusion applications, as they allow integration between both Oracle products and those from other vendors.

The second component of Fusion is around User Interface, specifically with relation to training. The application follows a similar User Interface paradigm and model to other Oracle Fusion applications. This facilitates cross training on Oracle products, such that a user familiar with Oracle Financials or other Fusion products will be able to more easily train on and learn the application because of the similar User Interface and usage concepts. The application also includes context sensitive online help, user manuals, installation manuals, Database Administrator manuals, a data dictionary, Javadoc documentation, and other reference and help material to assist in training.

Global

The application was designed from the ground up to fully support global, multinational companies and operations. This is core functionality integrated into the billing and accounting engines, rather than merely a language pack bolted on after the fact.

The application supports multiple languages and character sets. Language translations are loaded into the application through language packs which can also contain different dialects, such as South American versus European Spanish. The application supports Double Byte Character Sets (DBCS), allowing non-Latin character languages such as Chinese (hanzi), Japanese (kanji), Korean (hanja), Arabic, Russian (cyrillic), and others. While many systems allow language packs, few support the necessary double byte Unicode encoding for different character sets.

The application supports operations across multiple regions, countries, time zones, and jurisdictions. Business rules can be configured based on the location (city, state, province, or country) to account for local processes, regulations, and legislations. The application inherently accounts for time zones during calculations. For example, if a transaction occurs at 24:00 server time, and a price change (such as a promotional rate expiring) occurs at 24:00 local time, the application will calculate the time zone where the transaction took place to determine which price list was in effect at the time.

The application also fully supports Multi-currency accounting as part of the core system. The application supports directional currency conversion calculations, meaning conversion from one currency to another can be at different (non-inverse) rates. This can be used for currency hedging or other reasons. Currency conversion rates can be loaded in real time from other source systems.

The application supports separation of pricing and issuing currency. For example, a banking customer may be charged 1 Euro (the pricing currency) for every 100 USD (the issuing currency) worth of transactions on their credit card. Alternatively, conversion could be used. For example, a banking
customer may be charged 1 USD, to be billed in Euros, for every 100 USD worth of transactions on their credit card. In this case, the bill would be based on the USD Euro conversion rate, e.g. 0.75 Euro.

Conclusion

Oracle Financial Services Revenue Management and Billing provides a fully modernized billing application, with the necessary SOA and legacy interfaces for existing and future client systems, client business rule support through configuration and customization, and global business capabilities. It is a proven, scalable, and future-proof platform for deploying billing, collections, receivables, and subledgering.