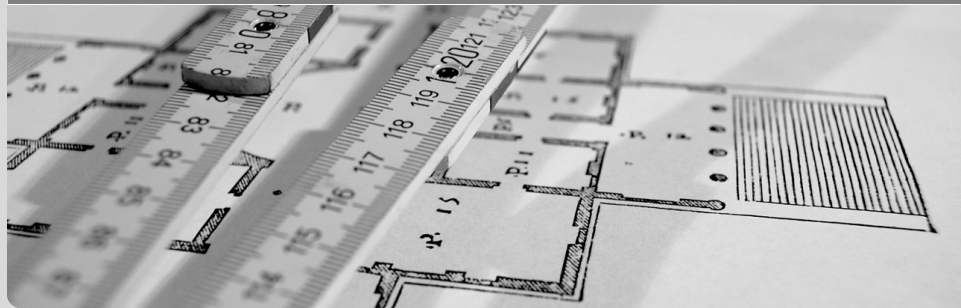


DetOnBan: a Novel Approach for Detailed Online Banking and Bookkeeping

Summer Term 2011: Report of the Project Group z10 (Advisor: Dr.-Ing. Michael Kuperberg)
Christoph Föhrdes, Fabian Gorsler | July 31, 2011

INFORMATICS INNOVATION CENTER (IIC), CHAIR FOR SOFTWARE DESIGN AND QUALITY (SDQ)



- Modern mainframes (e.g. IBM z10): foundation of mission-critical software in banking, trading etc.
- IBM Böblingen R&D GmbH provided the KIT with a z10 for teaching and research
- **Scenario:** extending online banking feature set with an integrated “online receipt” to provide one-stop access to purchase details
- ⇒ Full spectrum of tasks: domain research, technology learning, requirements engineering, software design and implementation, documentation, testing and deployment

- 1 Introduction
- 2 Prerequisites
- 3 Interfaces for Financial IT
- 4 Implementation Aspects
- 5 Current State
- 6 Future Work
- 7 Conclusion

- Card payments today: online banking entry only with amount/date/vendor
 - Details separated on paper or in email/PDF bill
- ⇒ Integrate and co-process information with card payment!
 - Implementation using real-world interfaces and formats
- ⇒ Potential for banks and HW/SW vendors
 - Banks can offer additional services to customers
 - Hardware vendors benefit through selling more computing resources
 - Software vendors can offer new software solutions and integration services

Introduction to the Project Group Organization

- The “Project Group z10” started in summer term 2011 with the intention to learn and discover the features of the IBM System z
- The team is currently consisting of two members, plans to grow :-)
- The work of the team consists of two parts:
 - research phase (seminar paper and presentation, 3 ECTS)
 - practical phase (lab course, 6 ECTS)
- The Project Group z10 is supervised by Dr.-Ing. Michael Kuperberg

Description of the Targeted Platform

- IBM z10 BC with an IBM DS8700 enterprise storage system
 - located at ATIS data center at KIT
 - dedicated access to the z10 for students
- Operating system partition with z/OS 1.12 and UNIX System Services enabled
- IBM WebSphere 7 as Java EE container
- IBM DB2 as enterprise-grade database management system

- Later phases: IBM CICS for advanced transaction processing, using COBOL code
- Later phases: usage of RMF, WLM and similar performance management tools

Technologies used for Implementation

- The main application should be built on top of Java technology using Java EE 5 features [1]
- Enterprise Java Beans technology was used for implementing business logic, persistence and transaction management
- All team members were familiar with the Java programming language, rapid project start with early prototypes

- Subversion for collaboration and version control
- JUnit 4 for unit testing and bug hunting
- JIRA for project management

- All components should be separated and accessed over interfaces
- All components should work as independent as possible
- The composition of components should be realized dynamically
- The amount of static configuration should be minimal

- The application must be designed with flexibility and extendability in mind
- Systematic reuse of existing libraries for the implementation
- A performance evaluation and optimization to the target platform should be possible

A Real World Scenario...

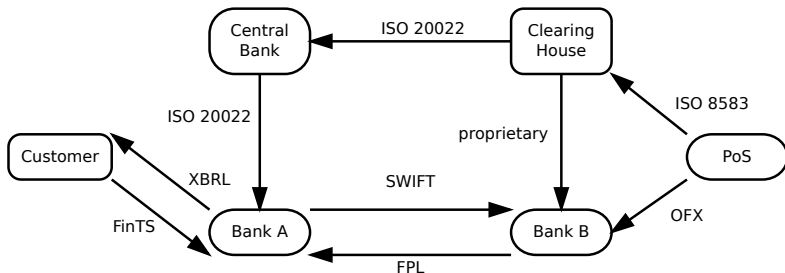


Figure: *Some* Standards and Protocols in the Financial World

A first simplified Scenario

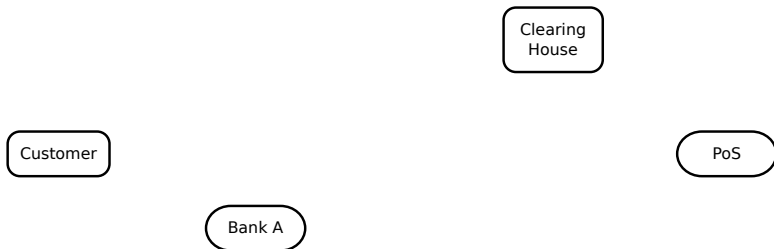


Figure: Only the most important Stakeholders for our Scenario

- For the selection of data formats and interfaces a literature research was done in order to identify adequate products:
 - SEPA approach based on ISO 20022 with the intention to provide a comprehensive approach for universal financial messaging [2, 3]
 - FinTS approach as a comprehensive suite of protocols, data formats and messages for home banking [4]
 - OFX as a data format for home banking built by home banking software vendors influenced by demands of the U.S. market based [5]
 - XBRL as data format for the exchange of any kind of business information through meta models (XBRL taxonomies) [6]
- Standard documents not freely available, e.g. for the most demanding standards (ISO 8583, ISO 20022)

- Use a clean separation of business logic and data persistence through the use of Enterprise Java Beans
- Split the business logic in components and implement all components independently
- Components should provide services to access entity beans belonging to a component
- Allow the distribution of components, therefore allowing horizontal and vertical scaling

- In the first iteration, **functionality** is the main goal, performance-related tasks will follow in the next term

- Services are mapped to methods of stateless session beans
- Persistence is implemented through the use of entity beans
- The web interface is connected through Java RMI with data transfer objects (DTO)
- JSF is used to implement the web interface
- JAXB is used for XML processing
- Payments can be received by SOAP web services

- Definition of components led to four distinct components:
 - ① Account Management: Create and manage customers, accounts and account transactions
 - ② Card Payment: Handling of ISO 8583 based messages and reception of detailed transaction information
 - ③ Data Processing: Processes detailed transaction information and provides access to bills and digital signed content
 - ④ GUI: The web frontend for accessing the application by customers
- Entities are organized by their components and should be only directly accessed by their “home” component

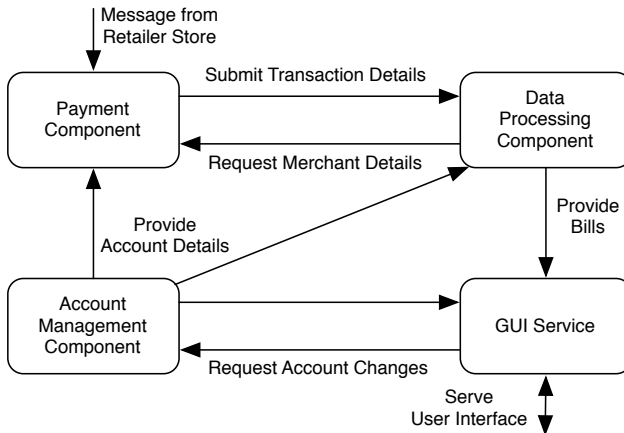


Figure: Architecture: Data Flow Overview

- Data model consists of about 25 entities modeled in 3NF without views and materialization
- Creation of the data model by reverse engineering of a manually created database layout
- Refinements in Java source code, regeneration of the model by the persistence layer, iterative improvements
- Database independent design (cross-platform testing), product-specific tuning will follow
- Manual creation of indexes by analyzing executed queries generated by the persistence layer

Data Model for Transactions

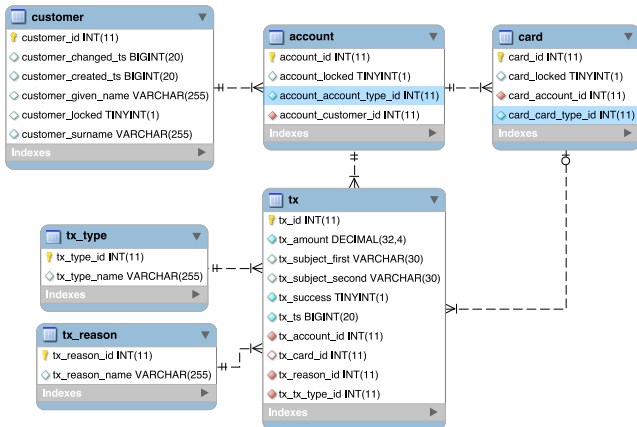


Figure: Data Model for Transactions

Data Model for Bills

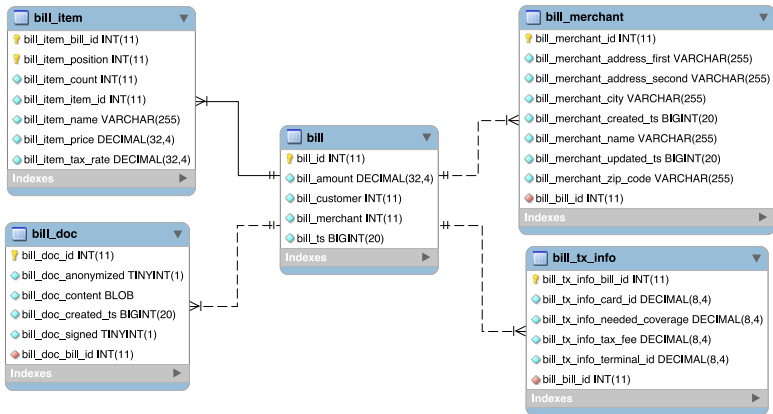


Figure: Data Model for Bills

- Currently transactions are implemented through the EJB container
- The transaction lifecycle therefore is as follows:
 - ① Remote connection by a client opened
 - ② Connection to an EJB established - Transaction created
 - ③ Invocation of methods of the EJB
 - ④ EJB dereferenced, connection closes - Transaction closed
- Very coarse grained transactions, long duration

- JAXB, a wrapper from XML to Java classes and vice versa
- JUnit, a software testing framework
- j8583, a library for the creation and parsing of ISO 8583 messages (no semantics)
- BCD4J, a library for creation and parsing of BCD (binary-coded decimal) numbers

- Random test data generation through an EJB client invoking remote services
- Unit tests for testing all remote capabilities (POS, GUI)
- EJB local service testing by helper remote bean

Implementation of Account Management Features

- The system is capable of managing customers, accounts and cards
- Each of these entities can be locked and for each transactions the whole chain needs to be unlocked
- Accounts and cards can be limited to maximum transaction amounts
- Most services provided by this component are rather generic and administrative
- Starting point for our banking approach and the foundation for other components

- A remote client (POS) can communicate with ISO 8583 messages over a SOAP web service with the payment component
- The payment component supports authorization (0x1*0) and payment (0x2*0) messages, other message types can easily be added
- The account management provides the payment process with the necessary data to verify the merchant, the customers card and check the account coverage
- The component is capable of receiving detailed transaction information in a custom XML format through a SOAP web service
- The raw detail information is delegated and processed by the data processing component

Implementation of Payment Transactions with ISO 8583

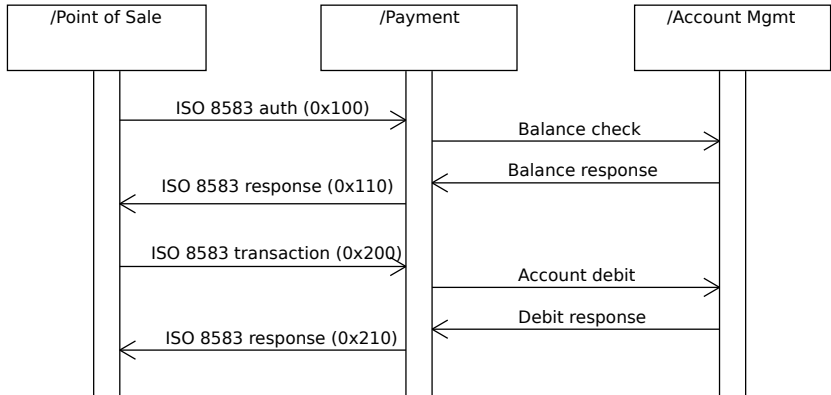


Figure: Implementation of Payment Transactions with ISO 8583

Implementation of ISO 8583 Message Processors

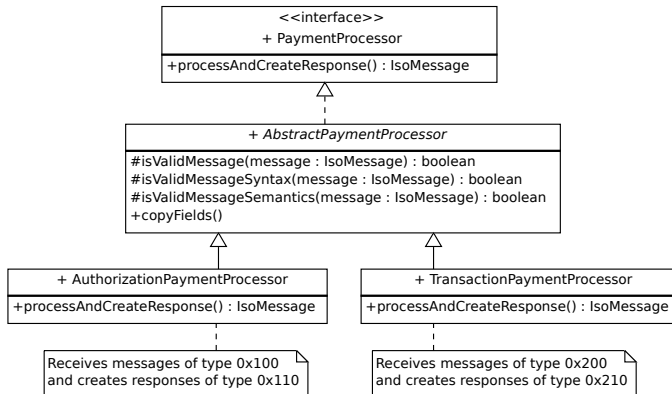


Figure: Implementation of ISO 8583 Message Processors

Implementation of the Submission of Details

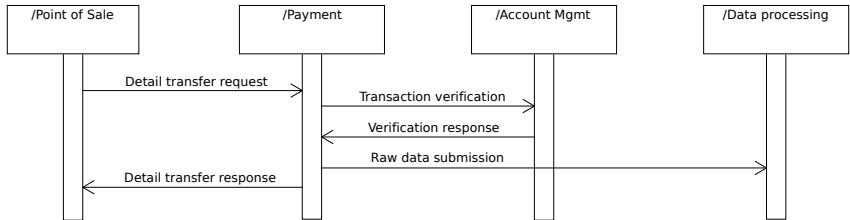


Figure: Implementation of the Submission of Details

Implementation of Data Processing Features

- The data processing component receives the raw detailed transaction information
- The data is mapped to entity beans and persisted by the component
- The component maintains own entities for bill data management
- Currently the component operates synchronously, should be converted to a WLM-controlled message queue
- The detailed bill information should be rendered as document, optionally signed digitally and archived

- The GUI service provides the web interface over a remote EJB interface with data and methods
- The data is transferred through DTOs filled with data from the other components
- The web interface was built on top of JSF technology for a clean MVC-based design
- The web application is developed and deployed as separate application
- No authorization in this prototype, just account selection

Interfaces for Interaction with the System

- The system provides remote connections through EJB remote interfaces and as SOAP web services
- SOAP web service interface by the payment component for
 - ISO 8583 communication
 - transaction detail transfer
- The GUI service component serves the web interface through a remote EJB interface







- Defining more fine grained transactions with explicitly committed actions
- Study explicit limitation of resource usage
- Transaction processing should be coupled with workload management features
- Implementation of asynchronous processing facilities
- Connection to a “real” system/prototype

Support of additional Data Formats and Protocols

- OFX for the representation of bills with detailed transaction processing
- FinTS and/or OFX for creating a home banking interface
- PDF for exporting bills with support for digital signing

- Banking-relevant and transaction-relevant technologies have been studied, compared and documented
- A functioning foundation has been designed and implemented
- Project goals have been adapted to fluctuating team size :-)
- The next term can be used to extend and complete the software, e.g. a more performance-related development and evaluation
- Existing code base will be reused and augmented
- Usage of z10 features for transaction management and performance control

References

-  E. Jendrock, J. Ball, D. Carson, I. Evans, S. Fordin, and K. Haase, “The Java EE 5 Tutorial,” 2010. [Online]. Available: <http://download.oracle.com/javasee/5/tutorial/doc/>
-  European Payments Council, “Homepage.” [Online]. Available: <http://www.europeanpaymentscouncil.eu/>
-  International Organization for Standardization, “ISO 20022 Universal financial industry message scheme.” [Online]. Available: <http://www.iso20022.org/>
-  Zentraler Kreditausschuss, “FinTS – Financial Transaction Services.” [Online]. Available: <http://www.hbci-zka.de/>
-  Open Financial Exchange, “OFX: Home Page.” [Online]. Available: <http://www.ofx.net/>
-  XBRL International, “XBRL.” [Online]. Available: <http://www.xbrl.org/>