Evolution of Prepaid Payment Processor's Software Architecture – An Empirical Study

Abdul Haleem Qureshi
Faculty of Information Technology
University of Central Punjab
Lahore, Pakistan
abdul.haleem@ucp.edu.pk

Ali Afzal Malik
Faculty of Information Technology
University of Central Punjab
Lahore, Pakistan
drali@ucp.edu.pk

***Abstract*—Prepaid cards are the payment option for consumers who want to use an electronic means of payment but do not want to tie up the payment with a credit or debit account. When a prepaid card transaction is initiated using a Point of Sale (POS) machine or an Automated Teller Machine (ATM), it travels through multiple entities for authorization. These entities include merchants, acquirers, branded networks, and payment processors. Each of the entities has its own software solution for processing its part of the transaction. In this paper we present an empirical study of the evolution of a payment processor's software architecture. We first describe a basic architecture which acts as a baseline for further evolution. Results of transaction processing on this baseline architecture are discussed to highlight different quality of service issues. This architecture is gradually evolved into subsequent architectures resolving the encountered issues.**

Keywords-architectural evolution; domain-specific software architecture; empirical study; prepaid payment processor; QoS requirements

I. INTRODUCTION

Nowadays a consumer has multiple methods or instruments available for payments such as cash, checks, plastic cards, etc. These payment instruments can be broadly categorized as paper instruments and electronic payment instruments [1]. Paper-based payment instruments include cash and checks while electronic payment instruments include Automated Clearing House (ACH), credit cards, debit cards, prepaid cards and virtual cards [1][4]. Prepaid cards are the latest innovation in the plastic card category.

Prepaid cards were introduced to replace small-value cash transactions. There is no bank account linked with a prepaid card. So, for using a prepaid card, the cardholder does not need to have a credit or debit account in any bank. All the cardholder needs to do is to load the amount on the card before using it. As indicated in [1], the credit card networks used their existing infrastructure and processing life cycle for prepaid cards. This support of existing widespread credit networks resulted in greater adoption of prepaid cards.

When a prepaid card transaction is initiated using a Point of Sale (POS) machine or an Automated Teller Machine (ATM), it travels through multiple entities for authorization [3]. At one end, there are merchants who accept a prepaid card for providing goods and services. As shown in Fig. 1, multiple merchants are connected with one acquirer. An acquirer is the entity which receives transaction data from merchants for processing and delivers it to the relevant network such as Visa, MasterCard, Discover, etc. In this way, an acquirer connects its merchants to global networks.

At the other end lie the payment processors which have the responsibility of issuing cards and maintaining their balances. The payment processors accept transactions initiated from merchants and decide whether to approve or decline these transactions based on available card funds and other pre-defined business rules such as those listed in [3]. The terms "issuer processor" and simply "issuer" are also used interchangeably for payment processors [2].

The network between acquirers and payment processors works as a bridge between them for communication [2]. It provides a common protocol of communication between a large number of acquirers and payment processors spread across the globe [6]. This network enables acquirers to accept transactions from a cardholder having a card issued by any payment processor. To facilitate the global acceptance of transactions, one network is also able to accept and route transactions of another network [5].

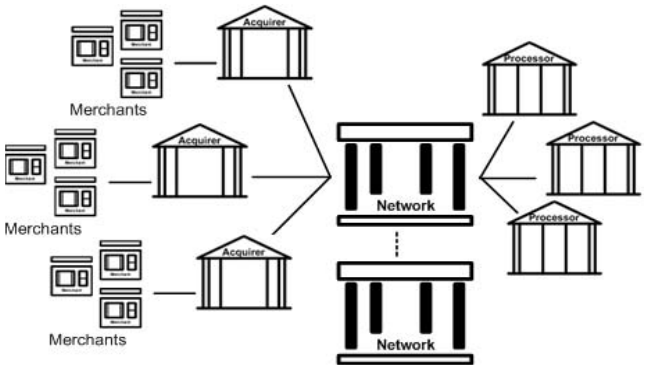

Figure 1. Prepaid payment processing network infrastructure

Transaction processing is an eight step process [7] as summarized in Fig. 2. In step one, a cardholder presents his card to the merchant for swapping on the POS machine or

978-0-7695-4927-9/12 $26.00 © 2012 IEEE
DOI 10.1109/FIT.2012.41

inserts it in an ATM machine for withdrawing money. In step two, the merchant passes the transaction to acquirer for further processing. In step three, the acquirer passes the transaction to the network it is connected with. In step four, network identifies the issuer using the card number. Transaction is then routed to that issuer. On receiving a transaction, issuer checks the card validity and the balance on the card. It may decide to accept or reject the transaction based on common or issuer-specific business rules. In the next step, the issuer sends the transaction back to the network which forwards it to the acquirer and the merchant. Finally, the cardholder gets the response of the transaction.

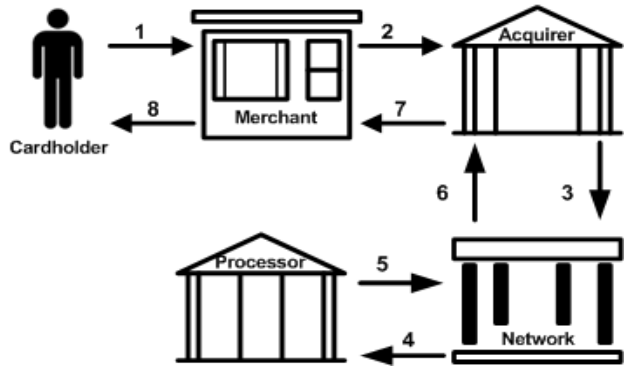

Figure 2. Transaction processing steps

Different architectures of payment processor software are presented and discussed in the literature. Different aspects of their quality of service attributes have been addressed such as performance, maintainability and reusability. In this paper, we have discussed the evolution of payment processor software architecture with respect to accuracy, security, performance and availability. We have presented empirical data to verify the architectural changes done for meeting the requirements.

Rest of the paper is organized as follows: section two gives a brief overview of existing literature in this area and discusses alternative architectures with respect to different quality of service attributes. Section three provides the details of the empirical setting of our study. Transaction execution results are presented and discussed in section four.

II. RELATED WORK

We have found very little work in the existing literature that focuses on the architecture of payment processor software. In this section we have chosen three studies that are relatively more pertinent to our research.

The architecture proposed in [8] primarily focuses on minimizing transaction execution time. It uses two thread pools and an in-memory cached data for maximizing the performance. Apart from performance, the data security is considered in the architecture by using a Hardware Security Module (HSM) for secure data validation. However, sensitive data is stored in the database as plain text – an obvious vulnerability. In order to achieve maximum performance, transaction response is sent to network before updating card balance and persisting transaction data in database. However, it does not cover the failover plan dealing with situations in which the database could not be updated due to any exception. Further, availability of the system is also not discussed.

Another empirical study presented in [9] uses an architecture similar to the one proposed in [8]. However, it focuses on the platform used for developing the payment processor software. It compares .Net and Java platforms for analyzing performance focusing particularly on thread management, thread synchronization, and thread pooling. Apart from performance, it also discusses software size and complexity with respect to .Net and Java platforms. Finally, transaction execution results are discussed for both platforms. After discussion it is concluded that performance of .Net version is slightly better. According to the presented data, the Java version of software is more complex than the .Net version and it contains more lines of code as well.

The study presented in [9] extends the work done in [8] to obtain further performance improvements. However, it has not discussed any other quality of service attribute such as availability and accuracy. It has similar drawbacks as discussed for the architecture presented in [8].

Another case study on payment processing system presented in [10] focuses on enhancing a chosen architecture with respect to maintainability and reusability of components. This architecture consists of multiple components such as transaction flow and business process components. For fulfilling the maintainability requirement, it has listed down a complete feature list of the payment processing system and then has identified frequently changing features. After the identification, it proposes to create new components consisting of these frequently changing features. After this activity, most of the future changes will be done only in newly created components and rest of the system will remain unchanged increasing the maintainability of the software. Next, for increasing the reusability of the components, it proposes to generalize the logic of common business process components. This activity will result in reducing code redundancy and will provide facility of more generic use of the components.

No empirical data is presented in [10] to support the proposed mechanism of increasing maintainability and reusability. Furthermore, complex deployment structure of the software including multiple hardware nodes and redundantly deployed software components for failover scenarios are not considered.

III. EMPIRICAL SETTINGS

In this section, we discuss transaction categories considered for data collection. This is followed by a discussion of the baseline architecture of payment processor software. Next, the hardware and software environment is described in which transaction execution data is collected.

A. Transaction Categories

The transactions we considered for data collection fall under five categories i.e. financial transactions, pre-authorization transactions, clearing transactions, status check

transactions, and reversal transactions. A financial transaction is a credit or debit transaction. It can be accepted or rejected based on the availability of the required amount in cardholder's account and other implemented rules. Upon accepting the transaction, the payment processor credits or debits respective transaction amount from the cardholder's account.

In case of pre-authorization transactions, a merchant provides an estimated transaction amount to the payment processor for reserving that much balance from cardholder's account for future payments. This amount is not available to cardholder until the merchant clears the transaction or a pre-defined time limit of keeping the amount reserved is expired. There are many reasons of doing a pre-authorization transaction instead of a financial transaction. For instance, in case of hotel bookings, final transaction amount is not known until the time of departure. However, while booking, merchant can perform a pre-authorization transaction of the estimated transaction amount.

A clearing transaction contains final transaction amount against an already done pre-authorization transaction. Upon receiving a clearing transaction, the payment processor finds the corresponding pre-authorization transaction to identify and clear the reserved balance according to the received final transaction amount. As per rules of prepaid payment processing, an issuer is bound to accept the clearing transactions regardless of its business rules and balance on card. However, issuer's liability is limited to the pre-authorization amount only. If the issuer has to approve greater amount in clearing transaction, it can be claimed and merchant has to pay the overdraft amount after going through the process of claim acceptance.

In status check transactions no transaction amount is involved. Such transactions are used to check a card's validity. Upon receiving such a transaction, the payment processor may check card number, expiry date, and status (e.g. active, inactive, lost/stolen, etc.). Balance inquiry transactions are also considered as status check transactions in which issuer returns cardholder's account balance to the merchant.

Reversal transactions are used to cancel the effect of an already done transaction. There can be many reasons to reverse a transaction such as merchandise return, occurrence of any network failure in the path back from issuer to merchant, etc. Just like clearing transactions, an issuer is bound to accept reversal transactions but is liable only up to the amount of actual transaction which is reversed. Issuer can claim the undue reversal amount and merchant has to pay back the amount after acceptance of claim process.

B. Baseline Architecture of Payment Processor Software

Fig. 3 shows the baseline architecture for our case study. The Communication Server node contains the Communication and Parsing component and the HSM Verification component. The Communication and Parsing component receives transactional data from network and parses it to extract transaction details for further processing. After parsing, confidential data such as cardholder's PIN and card's magnetic strip or chip data is verified using the HSM Verification component. The HSM Verification component works in coordination with the physical HSM node. The HSM node is required for maintaining digital keys used, for instance, in cryptographic and hashing functions. It also provides the functionality to perform authentication of encrypted or hashed data. It has a software interface indicated as HSM API in Fig. 3.

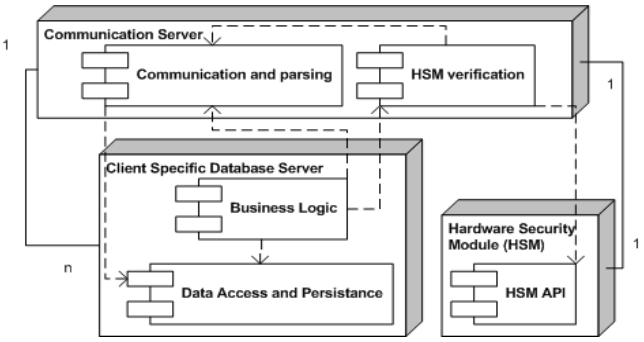

Figure 3. The baseline architecture

After the validation of confidential data, the transaction is handed over to the Business Logic component shown in the Client Specific Database Server node. In our baseline architecture, all business logic is implemented within a database using stored procedures. The Business Logic component uses the Data Access and Persistence component to access and store data within database tables. There is one database per client. So, for 'n' clients, we have 'n' Business Logic components and 'n' Data Access and Persistence components.

The Business Logic component contains issuer-specific and common business rules. We define common business rules as the mandatory logic which all payment processor architectures implement. In these common business rules, Account Validation is used to verify card number, its status and its expiry. Duplicate transmission detection logic verifies that the received transaction is not a duplicate one and it is not already processed. Merchant/Region-blocking logic verifies that the merchant and region from where transaction is initiated is not included in blocked merchants/regions list. Based on statistics of earlier transactions, suspicious activity detection logic verifies that the transaction is not a fraudulent one. In case of a clearing transaction, the pre-authorization transaction locator logic finds the corresponding pre-authorization transaction using card number and other transactional data elements. In case of reversal transactions, the original transaction locator logic finds the transaction whose reversal is received.

Table 1 shows the relationship between transaction categories and different components/subcomponents of payment processor architecture. Since Communication and Parsing component communicates with network, it is applicable for all transaction categories. HSM Verification component and account validation, and merchant/region-blocking subcomponents are not applicable to clearing and reversal transaction categories as an issuer is bound to accept transactions of these categories. Therefore, verification is unnecessary since such transactions cannot be rejected. Duplicate transmission detection and suspicious activity

detection subcomponents are not needed in status check transactions as no financial activity is performed in such transactions. Pre-authorization transaction locator subcomponent is required only for clearing transactions as in no other transaction category we need to locate pre-authorization transactions. Similarly, original transaction locator subcomponent is required only for reversal transactions.

TABLE I. ARCHITECTURAL COMPONENTS AND APPLICABLE TRANSACTION CATEGORIES

Components/Subcomponents	Transaction Categories				
	Financial Transactions	*Pre-authorization Transactions*	*Clearing Transactions*	*Status Check Transactions*	*Reversal Transactions*
Communication and Parsing	Yes	Yes	Yes	Yes	Yes
HSM Verification	Yes	Yes	No	Yes	No
Business Logic – duplicate transmission detection	Yes	Yes	Yes	No	Yes
Business Logic – account validation	Yes	Yes	No	Yes	No
Business Logic – merchant/region-blocking	Yes	Yes	No	Yes	No
Business Logic – pre-authorization transaction locator	No	No	Yes	No	No
Business Logic – original transaction locator	No	No	No	No	Yes
Business Logic – suspicious activity detection	Yes	Yes	Yes	No	Yes
Data Access and Persistence	Yes	Yes	Yes	Yes	Yes

C. Basic Transaction Flow

The Communication and Parsing component discussed in the baseline architecture consists of two subcomponents i.e. Communication subcomponent and Parsing subcomponent. The Communication subcomponent receives transactional data from the network and puts it in a first-in-first-out queue. The Parsing subcomponent picks transactional data from the queue and parses it to extract the transaction details. Fig. 4 shows this flow of the transactions. In this figure, transaction queue represents the first-in-first-out queue. Rest of the processing is shown as a single "Transaction Processing" component which picks transactions from the queue for further processing.

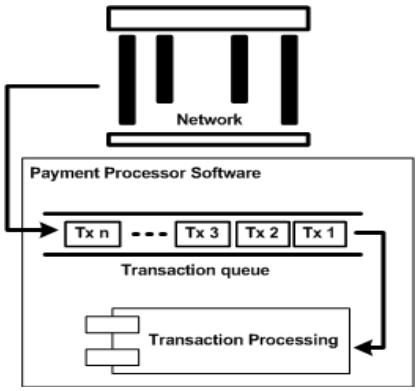

Figure 4. Basic transaction flow

D. Hardware and Software Environment

The empirical data used for this case study was collected on communication servers and database servers each having AMD Opteron 2356MHz Quad-Core processors, 16 GB RAM and SunOS 5.10 as operating system. We used SafeNet HSM version Mark II-M080400W as hardware security module.

The payment processor software was written in Java and executed on jre1.6.0_23. The DBMS used on database servers was IBM Informix Dynamic Server Version 11.50.FC8. All nodes were deployed on the same LAN.

E. Data Collection

In this paper, architectural evolution is depicted through five successive architectures. For each of these five architectures, a separate set of ten thousand transactions belonging to five different categories – financial, pre-authorization, clearing, reversal, and status check – is used for evaluation. The breakup of these transactions with respect to different categories is shown in Fig. 5.

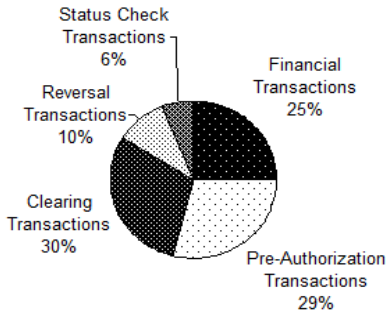

Figure 5. Percentage share of transaction categories

The payment processors received, on average, seven transactions per second and they were capable of processing ten transactions in parallel. However, while ten transactions

were in execution, the next coming transactions had to wait in the transaction queue.

We categorized the transaction processing results in seven groups. "Successful Transactions" group contained all transactions which encountered no system failure and were processed correctly. "Decline transactions - Data Accuracy" group contained transactions which had any data accuracy issue. "Decline Transactions – Business Logic" group contained transactions which encountered failure of Business Logic component for reasons such as processing exceptions, high transaction load, etc. Similarly, "Decline Transactions – Communication and Parsing", "Decline Transactions – Data Access" and "Decline Transactions – HSM Verification" groups contained transactions declined due to failure of respective components. "Timeout Transactions" group contained transactions which were processed successfully but reached the maximum allowed transaction processing time threshold. The value of this threshold was set to five seconds for this empirical study.

IV. Results and Discussion

In this section we have presented five successive payment processor software architectures each better from the previous with respect to one or more quality of service attributes. This architectural evolution begins at the baseline architecture discussed above and proceeds to address issues related to performance, availability, accuracy and security.

A. Baseline Architecture

When the baseline architecture was used, the average transaction processing time (considering both successful and non-successful transactions) was 1.7 seconds. The maximum transaction execution time was 6.5 seconds. There were 3.33% timeout transactions which took more than five seconds for processing. The average waiting time in the transaction queue was 0.75 seconds.

Fig. 6 shows the statistics of successful and declined transactions executed on the baseline architecture. 78.74% transactions were successful. The highest percentage for any type of decline transactions is 9.64%, which is due to failure of Communication and Parsing component. The second highest failure percentage is due to HSM Verification component (5.58%).

1.11% of the transactions were declined due to data accuracy issues. Upon investigation, it was found that the metadata used for processing transactions contained discrepancies. The metadata in our payment processor software included transactional data parsing rules, software configurations, lists of country and currency codes along with values of minor currency units, etc. For successful transaction processing, the metadata had to be the same on all databases but it was not the case and there were missing data and incorrect data issues.

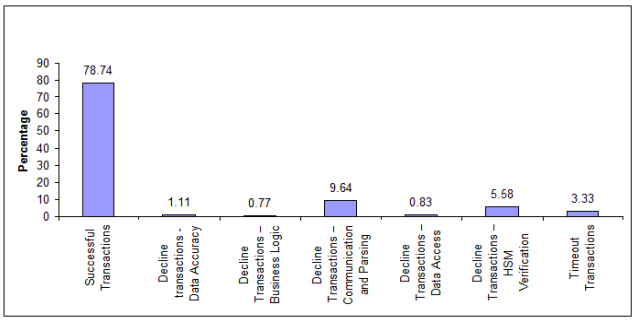

Figure 6. Baseline architecture statistics

After analyzing the security aspect of the baseline architecture, we found that HSM was used to verify PIN and card's magnetic strip or chip data. This data was not stored in the database. However, cardholder's sensitive data was stored as plain text in database. This sensitive data included card number, cardholder name, address, contact-numbers and social security number (SSN) or national identity card (NIC) number.

The next architecture tries to eliminate these data accuracy and data security issues.

B. Architecture-1

Fig. 7 shows the architecture obtained after the first evolutionary step. In this architecture, one new node – Common Database Server – is added. All metadata replicated on each client specific database was moved to this new node to maintain only one copy of the data. This eliminates the need to synchronize multiple copies. To resolve the data security problems, a Data Security component is added in the Client Specific Database Server node. The Business Logic component communicates with it to encrypt cardholder's sensitive data before persisting it in the database. The Data Security component is also used to decrypt the data at the time of retrieval.

Fig. 8 shows the distribution of transactions executed on Architecture-1. As is clear from this figure, there were no declines due to data accuracy issues.

The average and maximum transaction processing times in Architecture-1 were similar to the baseline architecture with values of 1.8 seconds and 6.8 seconds respectively. Similarly, with a negligible increase from the baseline architecture, average waiting time in the transaction queue was 0.9 seconds. However, the percentage of timeout transactions increased to 5.77%. This increase resulted in the reduction of the percentage of successful transactions by about 2%. The percentages of the rest of the declined categories were similar to the baseline architecture.

The next architecture focuses on the Communication and Parsing and the HSM Verification components which still have the highest percentage of declined transactions.

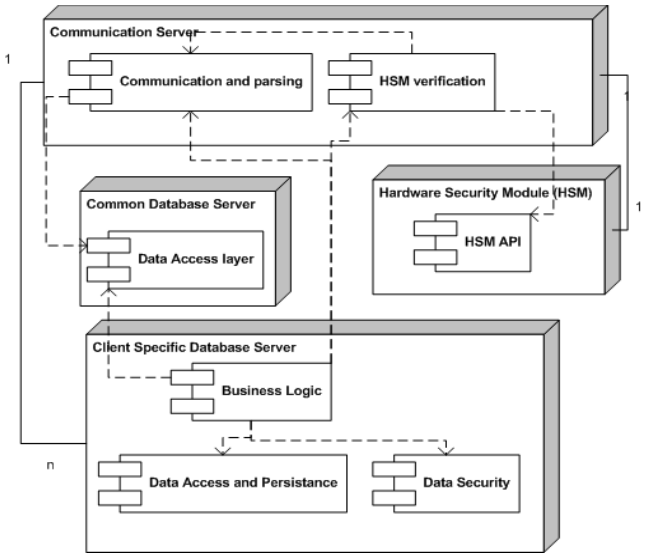

Figure 7. Architecture-1

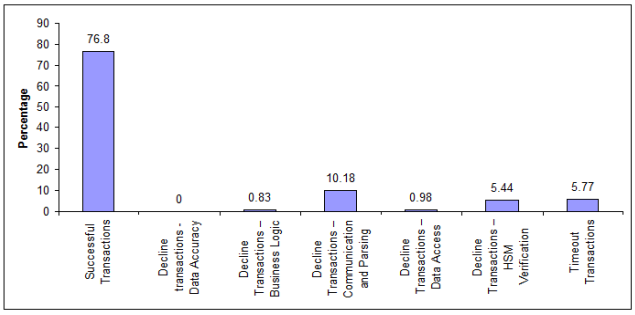

Figure 8. Architecture-1 statistics

C. *Architecture-2*

Fig. 9 shows the architecture obtained after the second evolutionary step. In this architecture, a secondary Communication and Parsing component is added to work as a backup of the primary Communication and Parsing component. At any given time, only one Communication and Parsing component receives transactional data from network. If the primary component encounters any failure, the secondary component takes its place and starts processing. For communication with Business Logic, another component Communication Router is added. It decides which Communication and Parsing component is active and gets data from it and passes the data to the Business Logic component.

For minimizing HSM Verification component failures, multiple numbers of HSM hardware nodes are added. Similar to the Communication Router component, there is a HSM Router component. The HSM Verification component communicates with the HSM Router component instead of directly communicating with HSM API within an HSM node. The HSM Router is responsible for identifying an available HSM node and communicating with its HSM API component for processing. The empirical data for this architecture was collected using four HSM nodes.

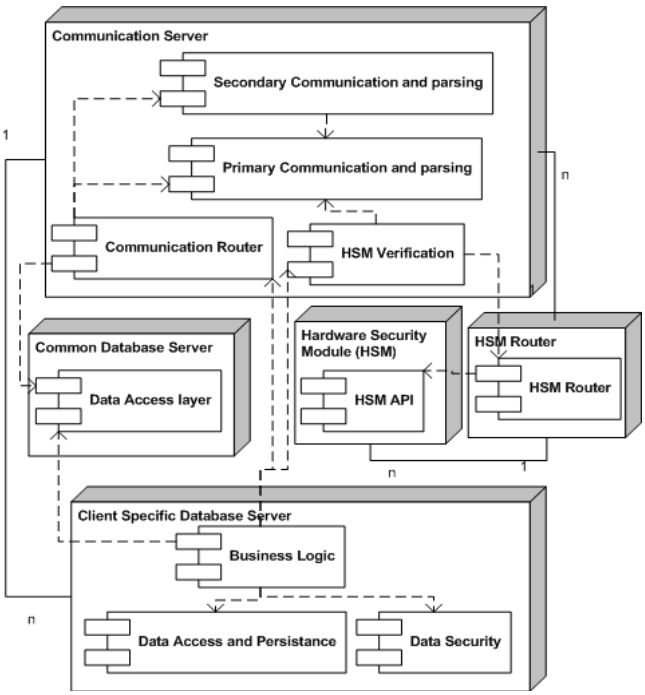

Figure 9. Architecture-2

Fig. 10 shows the distribution of transactions executed on Architecture-2. The percentage of declines due to Communication and Parsing component are less than half of the previous percentage. The declines due to HSM Verification component are almost negligible now (0.47% only). The percentage of timeout transactions, however, is now almost three times the previous value. This increase is because of processing overhead introduced by the Communication Router and HSM Router components.

The average transaction execution time measured on this architecture was 2.6 seconds (0.8 seconds more than Architecture-1) with maximum execution time of 9.4 seconds (2.6 seconds more than Architecture-1). The average waiting time in the transaction queue was also increased to 1.5 seconds (0.6 seconds more than Architecture-1).

The next architecture addresses the problem of increased percentage of timeout transactions and aims to further reduce the percentage of Communication and Parsing component declines.

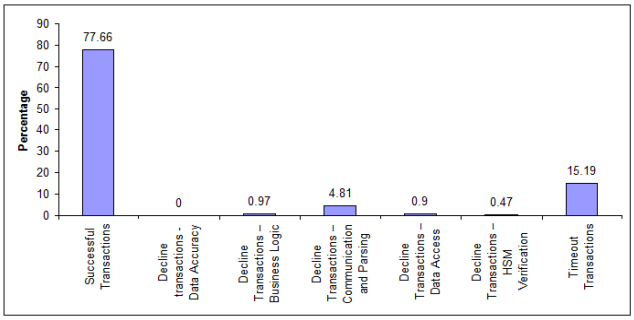

Figure 10. Architecture-2 statistics

D. *Architecture-3*

Fig. 11 shows the architecture obtained after the third evolutionary step. In this architecture, the Communication and Parsing component does not directly receive transactional data from network. A Load Balancer node is added which takes the responsibility of receiving the transactional data. Under the Load Balancer, the Communication Server nodes are increased just like HSM nodes were increased in Architecture-2. The job of the Load Balancer node is to route the received transactional data to the underlying Communication and Parsing components in round robin fashion. The empirical data for this architecture was collected using four Communication Server nodes. With these four nodes, we had four primary and four secondary Communication and Parsing components. Since with one Communication and Parsing component the payment processor software was able to process ten transactions in parallel, now it is able to process forty transactions in parallel.

Fig. 12 shows the transaction execution statistics of this architecture. The timeout transactions are totally eliminated and the percentage of Communication and Parsing component declines are significantly reduced to 1.32%. The percentage of successful transactions is now 96.2% - an almost 19% improvement over the previous architecture.

The average transaction execution time (1.1 seconds) and the maximum transaction execution time (3.8 seconds) on this architecture are better than the previous two architectures. The average waiting time in the transaction queue was reduced to 0.05 seconds – a significant improvement over the previous two architectures.

So far, we have achieved the goals of eliminating data accuracy, data security, transaction execution performance, and timeout issues. The next and final architecture focuses on improving the Business Logic component for further increasing the percentage of successful transactions.

E. *The Final Architecture*

In the last evolutionary step, processing load on database servers is considered and the Business Logic component is moved to a separate Application Server node as shown in Fig. 13. To further minimize the load on database servers, the Data Cache component is added. This component fetches all metadata from the Common Database Server node and all non-transactional data from the Client Specific Database Server nodes in the initialization phase of the payment processor software. The non-transactional data refers to unchanging data. The Data Security component is also moved to the Application Sever node. After these changes, the Client Specific Database Server node is used only for fetching transactional data and for data persistence.

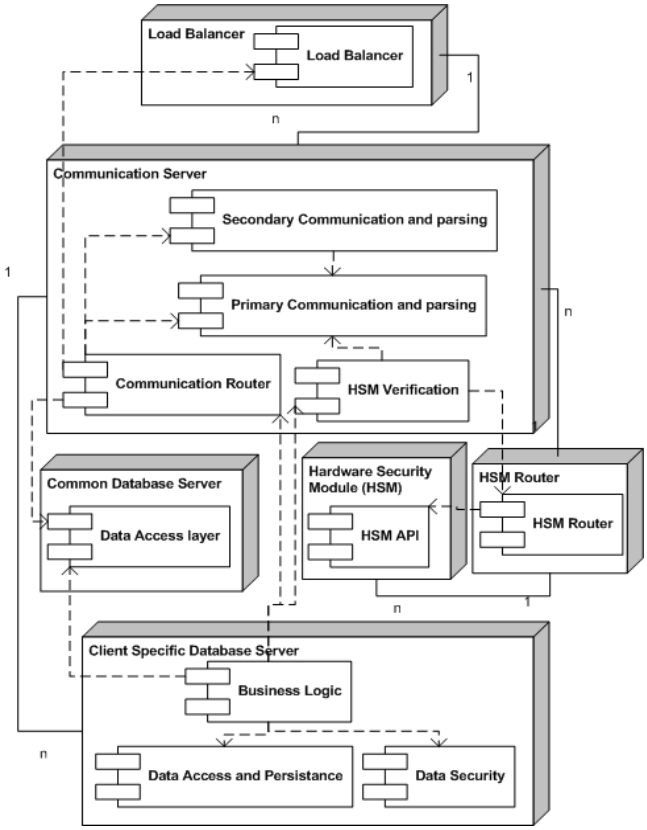

Figure 11. Architecture-3

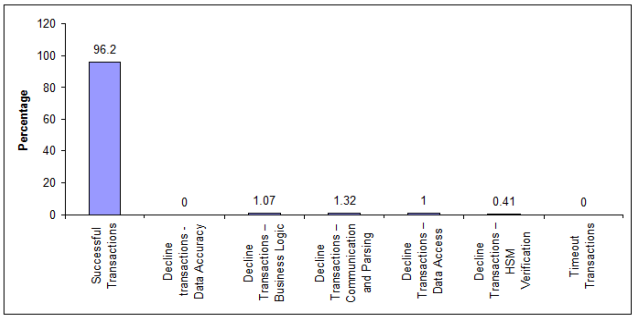

Figure 12. Architecture-3 statistics

Fig. 14 shows the transaction execution statistics for this final architecture. The percentage of successful transactions is further increased to 99.87%. Now, only a negligible percentage of Business Logic component and Data Access component failures remain.

The average transaction execution time on this architecture was further reduced to 0.9 seconds with maximum execution time of 3.1 seconds. Average waiting time in the transaction queue was also reduced to 0.03 seconds. These values are the best values across all architectures we have studied.

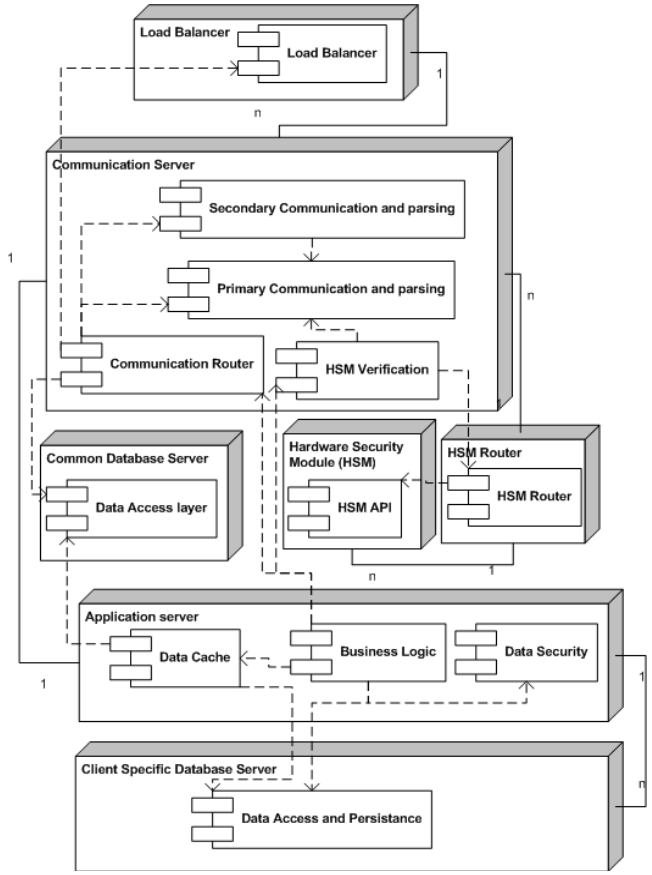

Figure 13. Final architecture

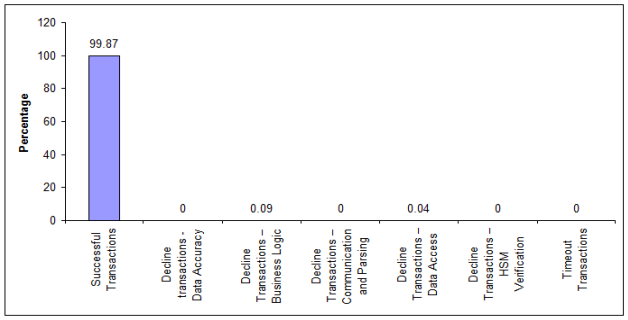

Figure 14. Final architecture statistics

F. *Conclusion and future work*

In this empirical study, we started from the baseline architecture of payment processor software. We presented transaction execution results using this baseline architecture and highlighted accuracy, security, availability and performance issues. After that, we selected the issues one by one and presented improved architectures to resolve them. As shown in Fig. 15, all highlighted issues were resolved in the final architecture and we achieved successful transaction ratio of more than 99% - an over 25% improvement from the baseline architecture.

This work can be further extended to analyze the scalability and maintainability of the architecture. Time consumed in software initialization can also be monitored for minimizing unnecessary delays during initialization. Another area which needs further examination is the memory consumption of the software.

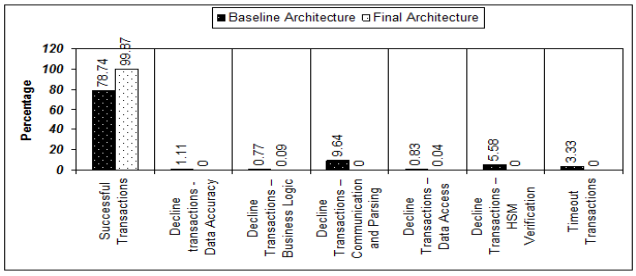

Figure 15. Baseline vs final architecture

REFERENCES

[1] S. Chakravorti and V. Lubasi, "Payment instrument choice: The case of prepaid cards", Federal Reserve Bank of Chicago, 2006.

[2] National Community Investment Fund, "Demystifying prepaid cards: An opportunity for the community development banking institution sector", 2009.

[3] MasterCard Worldwide, "Maestro global rules", 2011, pp 7.

[4] D. Stitilis and M. Laurinaitis, "Alternative payment systems: Lithuanian outlook" , Intellectual Economics, 2008, No. 2(4), p. 43–51.

[5] MasterCard Worldwide, "Benefits of open payment systems and the role of interchange", 2008.

[6] MasterCard Worldwide, "Bringing more value to every transaction - A look inside MasterCard technologies", 2011.

[7] MasterCard Worldwide, "The anatomy of a transaction", 2011.

[8] S. H. A. Hamid, M. H. N. M. Nasir, W.Y. Ming and H. Hassan, "Improving the performance of the authorization process of a credit card system using thread-level parallelism and Singleton pattern", Research Journal of Information Technology, 2008.

[9] S. H. A. Hamid, M. H. N. M. Nasir, and H. Hassan, "Java versus .Net: A comparative analysis of performance, size and complexity of credit card authorization system", Journal of Applied Sciences, 2009.

[10] K.C. Kang, J. J. Lee, B. Kim, M. Kim and C. Seo, "Re-engineering a credit card authorization system for maintainability and reusability of components – a case study", Lecture notes in computer science, Springer-verlag, 2006, pp 156-169.