Analysis of Future Ticketing Scenarios for Transport for London

By

Saumil Mehta

Bachelor of Technology in Civil Engineering
Indian Institute of Technology, Bombay

Submitted to the Department of Civil and Environmental Engineering in partial Fulfillment of the Requirements for the Degree of

Masters of Engineering in Civil and Environmental Engineering

At the

Massachusetts Institute of Technology

June, 2006

Signature of Author ______________________

Saumil Jayant Mehta
Department of Civil and Environmental Engineering
May 24, 2006

Certified by ______________________

George Kocur
Senior Lecturer of Civil and Environmental Engineering
Thesis Advisor

Accepted By ______________________ [illegible]

Andrew Whittle
Chairman, Departmental Committee for Graduate Students

Analysis of Future Ticketing Scenarios for Transport for London

By

Saumil Mehta

"Predictions are difficult—especially about the future" – Neils Bohr (1865- 1962)

Abstract

Rapid advances in information and communications technology in the recent past have opened up new possibilities for ticketing in public transit systems. These systems offer several benefits like replacing cash, deployment of a richer fare structure and minimizing queuing times. Although smart card ticketing systems have existed for several years now, the full potential of the developments in information technology have yet to be leveraged in terms of cost effectiveness and diverse fare collection strategies.

In 2002, TfL (Transport for London) has introduced the Oyster card as a means of smart card ticketing. However TfL has the option to extend or possibly depart from the Oyster Card system in the years 2010 or 2015, when the existing contractual agreements for are open to change. With the aim of understanding the future requirements of ticketing systems, TfL wants to explore ticketing policy and technology options that would be effective and feasible over the next 5 to 10 years. The study results will then guide the development of hardware and software concepts to support such systems. We have proposed seven options for future ticketing systems, which can be broadly categorized into seven categories based on their principal technologies used.

The present work is aimed at guiding TfL to specify the system requirements of the future technology that will best suit the needs of TfL and its customers. A software model has been developed to analyze the effectiveness of the several possible alternatives by estimating their costs and the potential revenues that each of them can generate. The system costs and revenues that can be generated highly depend upon the fare structure used for pricing tickets which in turn depend on the ticketing media, organization that handles the transactions and type of technology used for fare collection. Another goal of the project is to estimate the transaction times required to validate tickets and possibly use real time pricing by performing simulations using the Java card environment. The analysis will help in understanding the hardware requirements and physical properties of the card.

Thesis Supervisor: George Kocur

Title: Senior Lecturer of Civil and Environmental Engineering

Acknowledgements

I heartily thank Dr. George Kocur for his constant guidance during the project. It has been a wonderful experience working with him.

I would also like to thank Matthew Dorfman for his feedback and support during the project. I am also grateful to Professor Connor for taking great interest in my work and encouraging me from time to time.

Finally I would like to thank my parents, without whom I never could have entered, let alone completed the Masters of Engineering program at MIT. Their love and support has always been with me, and I have them to thank for everything I have achieved.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The growing trend towards electronic cashless commerce coupled with extensive technology developments in payment media have lead to the worldwide acceptance of smart ticketing systems in public transport. There are four broad categories of benefits from smart cards – cost reductions, service improvements, fare policy flexibility and increased revenues.

Studies have shown that the benefits of smart cards and other advanced technologies are greatest when used on a system-wide level, across different operators and modes. Within a single mode or operator, older technologies such as magnetic stripe cards may offer similar levels of cost, data availability and passenger convenience. The city of London has been among the early adopters of the smart card technology for ticketing. This project focuses on analyzing upcoming ticketing technologies that could be implemented by London's public transport in the future.

1.1 Overview of the Ticketing System in London

In November 2002, TranSys, Transport for London, and London Underground began rolling out smart cards as part of a £1.2 billion world-class ticketing system designed to make travel in the capital faster, easier, and more convenient for London's commuters. The credit card-sized Oyster cards need simply to be touched on the card readers on buses or at gates. For some travelers, the Oyster card will carry a period ticket for unlimited travel, while others will use the card for a new PrePay ("pay-as-you-go") facility. The cards can currently be reloaded via on-line facilities and at ticket offices. Functionality coming online includes load capability at ticket vending machines and via merchant terminals in a network of over 2,300 merchants called PASS agents.

However, the system has not been without technical setbacks and criticisms. The most significant usability issue is that pay as you go customers who for whatever reason do not "touch out" at the end of their journeys will not be charged correctly. Also, the fact that Oyster card is not integrated with other transit operators like the National Rail fails to achieve an important purpose of implementing the smart card, i.e. to allow ticketing across various modes and operators.

1.2 Motivation for the study

TfL has the option to extend or possibly depart from the Oyster Card system after the year 2010, when its existing contractual agreements with Prestige are open to change. With several innovative fare collection systems like bank smart cards and cell phones being implemented worldwide, TfL would like to explore the effectiveness of such systems and understand how they can be implemented in the city of London. It may be possible to negotiate a contract more favorable to TfL, both in terms of price and service.

Several strategies for the next advances in smart card-based fare payment remain to be explored. Should TfL leverage the technology developed in closed transit systems to enter the retail micropayments market, or should it focus efforts on becoming an application on a card issued by others? Should it be just another retailer without transit specific pricing or applications? Regardless of what decision is made, the early adopters of smart card technology have found that customers readily accept a change in the transit payment paradigm. The key focus therefore is to understand how technology and collaboration strategies in ticketing will impact transit ridership and costs incurred by TfL. MIT and TfL have collaborated on a two year project to develop the requirements of a ticketing system that will be most appropriate for TfL and its customers.

1.3 Study Objectives

1. Evaluate the feasibility of implementing a fare collection system for a specific types of ticketing media (e.g. bank card, smart cards, cell phones) and processing organization.

2. Estimate the total costs (hardware, barriers, readers, etc) of feasible ticketing systems.
3. Estimate the amount of revenues that could be potentially generated from various systems based on certain assumptions about transit ridership.
4. Estimate the time required for JAVA cards to process transactions.

To understand the effect of various fare structures on transit ridership, the passenger entry and exit data for buses and trains was analyzed. Trip distribution by distance and zones was classified based on the analysis. A cost model was developed and implemented in a Java application. Last, a Java card simulator was useful to explore the performance of smart cards used for bank card or online phone transaction.

1.4 Summary

The study shows that the emerging technologies like contactless smart cards and mobile phones have a great potential to reduce ticketing costs to transit organizations like TfL. The study also indicates that introducing complicated fare pricing strategies do not necessarily increase revenues collected by these systems. The study shows that if fare processing costs could be outsourced to banks or mobile phone carriers, significant cost reductions are possible as demonstrated by public transit system in Seoul, Korea. Another important observation made from the study is that current smart cards need to be improved in terms of memory and communication speeds to process transactions quickly. This is especially important in heavily used public transport systems.

Chapter 2

State of the Art Ticketing Systems in Public Transits

In the face of tight funding controls and intense competition from other methods of transport, public transport operators have found ways of improving the customer travel experience, reducing costs and increasing revenues. Contactless smartcard-based ticketing potentially provide a smooth, seamless traveling experience by enabling tickets to be purchased or topped-up from one operator and used with any other operator throughout the region or country. They also provide opportunities for multi-purpose ticketing – for example, having station car park, rail travel and event entry included on one ticket – and more sophisticated loyalty/reward schemes.

Transit operators are moving away from multiple, non-integrated fare collection systems to systems that require only a single contactless smart fare-card and allow travelers to access multiple modes of transportation, regardless of whether the transportation is administered by one agency or by multiple agencies within a region. This common infrastructure can provide efficiencies across operators and improve overall customer service.

Retailers and financial institutions also have an opportunity to partner with transit operators to provide the consumer with a payment card that can be used to pay for goods and services such as snacks, bridge tolls, parking fees, or food in restaurants or grocery stores located near public transit stations. Pilot projects indicate that consumers welcome such multi-application payment cards, perceiving them as convenient and cost-effective.

In the last decade, a large number of transit operators across the world have implemented and experimented with various technologies for electronic fare collection. Smart cards issued by financial institutions, mobile phones, smart cards with long range readability and memory less cards have been one of the most recent technologies adopted for electronic fare collection. An overview of the operational and technical characteristics of the state of the art fare collection systems will provide important insight into the benefits and limitations of these technologies.

2.1. Bank Cards

Bankcards – financial smart cards issued by banks as credit or debit cards – can be used directly for transit fare payment. This gives passengers the advantage of having one less card to carry, and it gives the transit authority the advantage of having fewer or no cards to administer. This can have a significant impact on the transaction handling and staffing costs of the transit agency.

2.1.1 The TaiwanMoney card, Kaohsiung

Kaohsiung, Taiwan's second-largest city, and several neighboring counties in the same transit consortium, decided that when they began offering smart card fare payment, they would not issue the smart cards themselves, but allow local banks to issue the cards. The banks were already planning to transition from mag-stripe bankcards to contactless smart cards. The cards are initially available through 3 different banks, but have the same features regardless of the bank issuing the card. A passenger needs an account at one of these banks to be able to get a TaiwanMoney card.

Key Features

- There is a prior agreement between card issuer and transit agency: the banks provide all transit information on each cardholder to the transit authority.
- Passengers simply tap their card at the entrance and exit gates or on the bus, and their fare charges will appear on their bank statements. Prepayment is not necessary.
- Passengers are not queried for their PINs when they enter allowing quick passenger movement through fare barriers.

- Technical security measures are present on the card to deter electronic theft, such as by card ripper.
- With exit barriers on the subways, Fare pricing is likely to be based on zones. TaiwanMoney card users are rewarded with a loyalty program
- The minimum cost of a trip (i.e., 1 zone fare) is deducted at the start of the trip, and if there is an additional charge, it is deducted at the end of the trip. But if the user does not tap out, the "maximum price of the trip" is charged.
- Card reading is done with "EMV in offline mode." EMV is a standard used for authenticating debit and credit cards. EMV is an acronym of "Europay MasterCard VISA", the three companies which originally cooperated to develop the standard. Operating in offline mode allows the card to be validated in .6 to .7 seconds. However this is still not acceptable for subway trains.
- To increase demand for the contactless cards, Taiwan-based Acer Inc., the systems integrator for the TaiwanMoney card project, is working with card and chip vendors to speed the transactions to .4 seconds or faster.

Card Details

The hardware and software on TaiwanMoney cards and readers is based on the MULTOS platform, the smart card hardware and software platform favored by MasterCard. (Visa favors JavaCard).

TaiwanMoney cards have a security model. They are integrated according to the EMV standard. EMV (an abbreviation for Europay, MasterCard, Visa), is a secure contactless protocol developed by Europay, MasterCard, and Visa to reduce fraud.

The card also has an internal security model to allow different applications to run on the same card. To implement this, the MULTOS platform includes a memory firewall between different application memory locations:

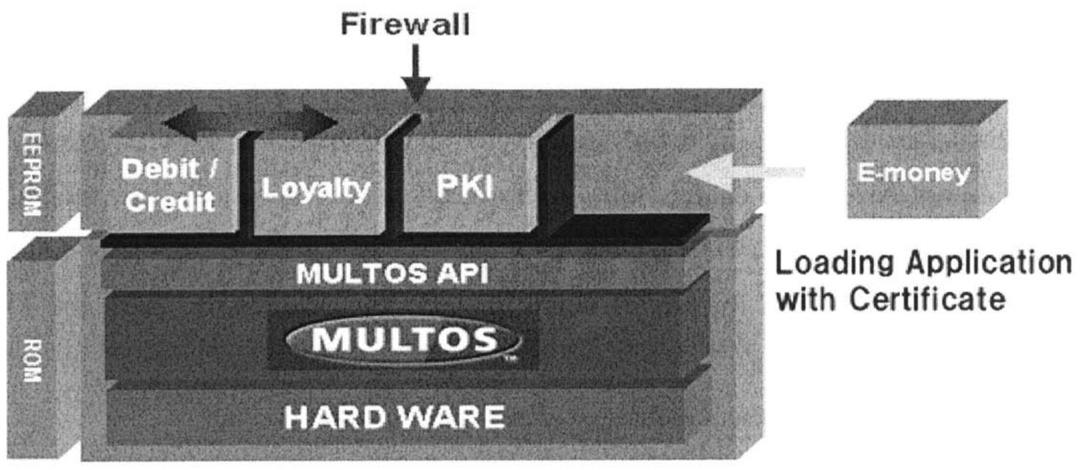


Figure 2.1 MULTOS Card platform

Source: http://www.hitachi.co.jp/Div/smartcard/english/multos.html

The TaiwanMoney card combines MasterCard, credit and debit, Mondex stored value , access to the Cirrus global ATM network and the MasterCard PayPass contactless functions.

The use of the card as credit, debit, pre-pay, or ATM, depends on the card reader. The readers that the transit authority uses will thus be different from the ones used in restaurants and banks.

The TaiwanMoney Cars is a microprocessor card. Data on TaiwanMoney cards is stored in a 32kB EEPROM, the largest memory size currently available on financial smart cards. 32kB provides a good deal of memory for each of the financial applications currently on the smart card (contactless credit, debit, e-wallet, and transit-specific applications), as well as room for other applications to be added in the future.

2.1.2 The T-Money bankcard, Seoul

When Seoul's transit authority launched its smart card program in July 2004, it simultaneously launched mobile phone payment and bankcard payment. By early 2006, bankcard payment had risen to 60% of the smart card fare paying population, with the remaining 40% divided between legacy magnetic stripe tickets and transit-authority-issued prepaid smart cards for users without smart bank cards.

Key Features

- Bankcard fare payments in Seoul are all postpaid (transaction) type payments, so the cards do not have to be charged up – users receive their charges in their monthly bank statements.
- Cards can also be used for buying small items at small stores that have the T-money mark.
- The cards can be bought and recharged at the subway ticket counter
- T-Money bankcards were, until recently, were not issued in conjunction with credit or debit cards – they were purely ATM cards with the T-Money function.
- Like the Kaohsiung TaiwanMoney card, the T-Money bankcard is an example of a transit bankcard with an agreement between the transit authority and the banks issuing the cards.
- Fare structure is distance based with transfer discounts between subway and bus
- Loyalty programs apply for T-Money bankcard users as well as T-Money Prepay card.
- Korea Smart Card, a card hardware and integration corporation founded to produce the T-Money card has distributes the T-money card. It negotiates commissions and fees with the banks for use of the card.
- Like the Kaohsiung TaiwanMoney card and all other financial smart cards, the T-Money card is an ISO 14443 smart card, though it uses a proprietary Moneta chip.

2.2. Mobile phone fare payment

Mobile phone fare payment is a convenience for passengers, who no longer need carry a transit or bank card at all – just their mobile phone, which they will in any case have. As with bankcards, the mobile phone fare payment gives the transit authority fewer cards to administer directly. Apparently most mobile phone payment means are linked directly to bank cards although it is possible for the phone carriers to bill riders directly.

2.2.1 Mobile SuiCa Payment in Tokyo

Transit fare payment has only begun quite recently in Japan – on January 28, 2006. Smart card payment in Tokyo urban transit – as well as on most of the JREast railway network, began in November 2001, when JREast introduced SuiCa, the "Super Urban Intelligent Card," an ISO 14443C smartcard with a FeliCa chip, featuring stored-value transit payment. The number of retail e-cash transactions was 15.8 million per month in 2005, more than double that of 2004. The large increase occurred after NTT DoCoMo added e-cash to its mobile phones in 2004.

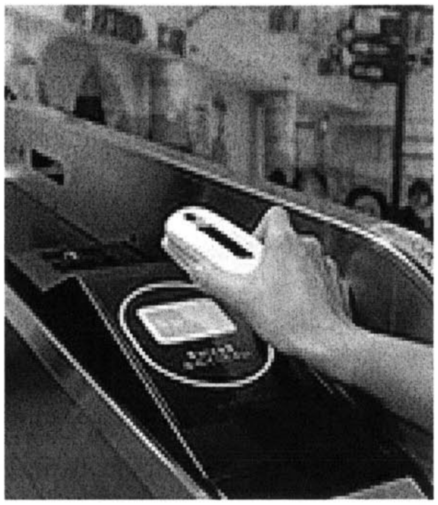


Figure 2.2 Mobile SuiCa payments at a JR East fare barrier
Source: www.sony.net

Key Features

- JR East subway and train gates accept mobile SuiCa payment from phones registered with all three Japanese mobile phone carriers.
- Readers at the barriers can read the SuiCa smartcards as well as mobile phones at tap-in and tap-out:
- Fare pricing is distance-based; SuiCa loyalty programs also apply to users of mobile SuiCa.
- In the initial stage, stored value and passes will have to be purchased over the air, via the mobile phone before it can be used as a transportation pass
- After mid-2007, the charge will be postpaid and appear on the passenger's mobile phone bill at the end of the month.

Technical Details

Many different FeliCa phones are available in the Japanese market. NFC or near-field communication is the incorporation of an ISO 14443C smart card into the mobile phone's electronics. NFC was developed by a consortium whose primary members were Philips and Sony. Philips now offers a generic NFC chip, while Sony offers the FeliCa chip, a chip that is compatible with NFC but contains Sony-proprietary functionality as well.

NFC phones have multiple memory areas that can be used for stored value, passes or transaction data. These memory areas can be read or written from the mobile phone to allow value to be added and fare plans to be updated. NFC payment does not drain the mobile phone's battery, and is even possible when the battery is dead, because the NFC electronics can act in "passive" mode – inductively powered by the RF signal from the reader. The NFC standard is still new and most mobile phones still use proprietary implementations, not the NFC standard.

2.3. Long Distance smart cards

Long distance smart cards are smart cards that differ from ISO 14443 smart cards, such as Oyster, in that they can be read by RFID readers at greater distances. These cards have the ability to provide the infrastructure to implement creative and flexible pricing policies by providing passenger route information to the transit authority (and thus real-time congestion information). Currently most transit agencies rely on flat fares and distance based fares. Theoretically they can also eliminate the need for tapping on a reader (and possibly the need for transit barriers, if an alternative method to eliminate fraud is used).

2.3.1 EasyRide Project, Switzerland

In the late 90's, Swiss National Rail and the Swiss public transportation consortium started a project called "EasyRide" aimed at making ticketing easier and more effective on national rail and public transportation throughout Switzerland. One of the primary strategies to emerge from this effort was a long-distance smart card ticket: passengers would simply enter the bus, tram, or train, and sit down. Their trip would be recorded by the card reader, and charged appropriately. Because Switzerland has no fare barriers and relies on the honor system (with spot checks) for anti-fraud enforcement, using the cards required no different passenger, driver, or ticket checker behavior.

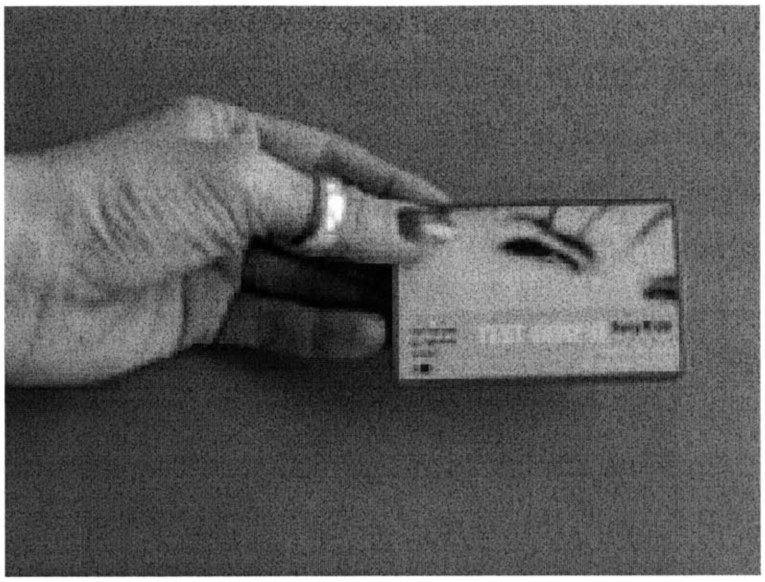


Figure 2.3 The EasyRide Ticket:

Key features

- This system worked at dual frequencies: 125 kHz to "wake up" the smart card, and 433 MHz to remain in contact with the smart card.
- They used a specially designed and patented 3D LF antenna for the 125 kHz signal, and a standard UHF antenna for the 433 MHz signal.
- The card was powered by a 3V lithium coin battery. It was the size of a credit card, about 3 times as thick.
- The 125 kHz signal was transmitted immediately inside the bus doors, and would awaken the card, which would then actively transmit a 433 MHz signal to (and read the 433 MHz signal from) the 433 MHz transceiver for the duration of the trip.

- The card went into sleep mode 30s after the 433 MHz signal was lost (passenger walked out of the vehicle).
- The 433 MHz signal was used to read and write data to the card continuously during the trip or at the entry and the exit of the trip only for a few seconds.

A similar study using long distance smart cards was performed in Dresden, Germany using tickets used that included a LCD display which showed the station where the vehicle was located – a feature to assure passengers that their trips were being recorded correctly. Vehicle location (to determine station of entry and exit for ticket charging) was done by GPS.

Figure 2.4 Allfa ticket with LCD used in Dresden, Germany
Source: http://rfid-informationen.de/info/news/archives/allfa-ticket.jpg

2.3.2 People Tracking at Microsoft Conference, South Africa

In August 2004, iPico Holdings, a South African based private company, introduced an Event Management System based on its proprietary iP-X_{TM} Dual Frequency RFID technology at the Sun City international convention centre in South Africa.

The Event Management Systems were used to track attendees at conferences staged by Microsoft, South Africa and several other global technology vendors. The system consisted of six Dual Frequency readers used at registration, and four Dual Frequency readers used to identify the traffic of attendees through four conference halls. The credit card sized tags were inserted into the swing labels attendees carried around the neck, and were used for hands-free tracking of attendees during the conference. iPico claimed that a tracking reliability of 100 % was obtained.

Technical Details

Long distance cards typically use Dual Frequency RFID technology which combines the ability of low frequency radiation to penetrate 'lossy' media (liquids, human and animal bodies), with the high speed data-carrying ability of higher frequency backscatter technologies, thereby achieving fast reading times and long reading distances associated with UHF RFID. The ability of Dual Frequency technology to be used in any free-flow people tracking application without any deliberate interaction between the individual and the tracking system, as is the case for conventional RFID based access control systems, makes it ideal for mass transit applications

Chapter 3

Future Ticketing Options for TfL

This chapter describes possible ticketing implementations for TfL based on emerging technologies. These options may differ from current practice of early adopters, and may require a staged implementation. The benefits and limitations of the technologies themselves as well as implementation issues are discussed.

3.1 Opportunities for Improvement

For TfL to decide whether or not to continue with the current Oyster card, it must assess the fundamental changes in operations and technologies and their impact on future systems. Some of the important changes that TfL may need to consider are:

1. Multi-application contactless smart cards

The current Oyster card is mainly used for public transportation. Recently TfL has studied whether certain retailers would accept Oyster cards for payment and TfL may want to push the case for contactless multi-application smart cards. No full-scale implementations exist yet, but pilot projects have been conducted by several e-purse providers such as Proton, Visa-cash. Financial institutions were reluctant to support transport usage because of security issues. Banks now realize that transport usage smart cards have millions of transactions per day and they want to capitalize on this opportunity.

2. Reduction in Life cycle cost:

With cheaper technologies and opportunities for collaboration with financial/mobile phone companies, investment in cards, equipment, clearing houses can be greatly reduced or replaced by an outsourcing fee.

3. Route and Time Specific Pricing

With development of smart cards having longer read range, TfL can track travelers and give them incentives to travel along non-congested routes and during non-peak periods. This is particularly important to London because of its highly interconnected network that allows multiple paths between many points.

4. Stored Value vs. Transaction

TfL currently uses "Stored Value" cards which require customers to have a certain amount of money on the card and to recharge cards when the value on the card drops below a certain value. Moving to a transaction based system would relieve travelers from recharging cards.

3.2 Description of Options

The technologies discussed in the previous chapter can be broadly categorized based on the ticketing media used for fare collection:

1. Smart cards (with memory and processor)
2. Memoryless smart cards (EPC tags)
3. Mobile Phone
4. Bank Cards
5. Long distance smart cards

Since most of these technologies are in their early stages and have the potential to add significant benefits to public transit in general, future ticketing systems are likely to be governed by one or more of these technologies.

3.3 Assessment of Options – Advantages and Limitations

3.3.1 Incremental Improvements to Existing Oyster Cards

Smart cards with read/write memory issued by the transit operator are the current standard. These can be improved principally by allowing interoperability among many transit operators. A common standard to read and write information to and from the cards is required. An open format known as the Extensible Markup Language (XML) has been used in trials and is a likely candidate for this purpose.

Operation

The operation of the current system would be very similar to that of the existing Oyster card. Although the current Oyster card implements zonal and flat fare structures with price caps, the extended system can possibly implement congestion pricing and transfer discounts and other policies, as could the current Oyster card.

Advantages

In addition to the networking simplicity they allow, writeable smartcards have a significant advantage that is currently not being used: the ability to permit cards to be used in different, distinct transit systems with inter-system payment agreements. With non-writeable smartcards, different transit authorities would have to work with either a common database or, more likely, use Web services to permit interoperability of smartcards. The extensive use of Web services in Service Oriented Architectures (SOA) in the next 10 years is possible, but with writeable smartcards, the information is all stored on the card, so that as long as the different transit authorities know how to read and write to the card, they can all honor one another's cards.

Such a feature would be very advantageous for a London traveler on a business trip to Manchester. He can simply get on the tram in Manchester and let his fare be deducted from his London Oyster card. Obviously, the more cities that participate in such a system, the more desirable this feature is.

Concerns and Issues

- The largest issues center on the governance and operation of a smart card system used by multiple organizations: resolution of billing issues, maintenance of data, interoperability testing, selection of vendors, and similar matters.

- As discussed elsewhere, stored-value smartcards have the disadvantage of being significantly more expensive than non-writeable smartcards – in a reasonable time horizon; it is unlikely that stored-value smartcards will cost less than $0.50, even when purchased in enormous quantities, while non-writeable smartcards may cost as little as $0.02 in the near future when purchased in bulk.

- Another disadvantage is that the writeable memory of a smartcard must be reprogrammed by a smartcard transceiver. So a user cannot simply get on the Internet at home and charge up the stored value on his or her card. However, there are cheap writers that can do this.

3.3.2 Memoryless smart cards

For smartcards to deliver their promise of seamless travel, reduced operating costs and improved information and planning, the transport industry needs smartcard solutions that can be rolled out economically and in a way that meets agreed international standards [4]. This can be done with the help of memoryless cards which are popular in the supply chain industry. Memoryless smart cards (MSCs) are similar to Oyster Cards, but have no data storage. They have only a serial number. They may possibly hold multiple serial numbers or have other security features.

Operation

- They cannot store prepaid value, fare plans, or other information on the card.

- All information associated with a card and account is stored on a database server; this data is retrieved and updated each time the MSC is read.

- To provide acceptable performance and reliability, copies of the ticketing database will be held on a server at each underground station and a subset will be held on units on each surface vehicle. Distributed database technology will be used to ensure transaction integrity across copies of the database; this technology is available now, though it is expensive. Its price is likely to drop significantly.

- Customers would all have smart cards, possibly of more than one type, on all services and for all trips. Registration is optional.

- Data would be available for essentially 100% of all passenger trips. Near real-time data would be available at underground stations for entries and exits, and for entries on each bus vehicle, correlated with location by using the GPS unit or AVL integration, and WiMax connectivity to transmit the information.

- Since all data for the card is stored on a central server and must be retrieved and updated with each cardholder's entry and exit from the system, fare barriers must have high quality communication with the central server.

- MSCs may or may not integrate with bankcard or banking payment systems; they could be a TfL-centric ticketing system.

Advantages

- The cards can be quite inexpensive. They currently cost US$0.50 in small quantities, and are expected to drop in price to US$0.05-0.10 or possibly lower over the 5 to 10 year horizon that we are examining. These cards are based on the technology being used in supply chain applications to track pallets, cases and individual items of consumer packaged goods, so there is very strong pressure to lower the costs substantially.

- An inexpensive card will allow it to be used economically for less expensive prepaid, single-trip, return, day tickets or similar relatively low-value tickets.
    - It is possible that a processor capable of acceptable security can be implemented at low enough cost on memoryless cards to resolve the security issues.

- These cards can be stocked and sold at kiosks at bus stops, or sold at retail outlets along bus lines, possibly without the retail outlet or kiosk needing a card reader/writer, and thus lowering the cost of providing the tickets.

- By storing all ticketing and fare policy data on a server, fares and policies can be changed flexibly. This option will enable possibilities such as:
    - Frequent traveler programs, in which free TfL trips or points are accumulated by frequent travelers and can be redeemed.
    - Marketing-driven pricing, such as 3-day sales on lines or portions of lines that have service improvements, or perhaps have had service problems that are now resolved. Such pricing can be used for customer acquisition.
    - By having virtually complete data on ticket issuance and use, marketing, operations and planning data quality is substantially enhanced.

- This will provide near real-time data on passenger entries and exits, by station. It will also provide near real-time information on origin-destination patterns, though this may not be necessary. Estimates of the passenger load of trains on segments of lines served by only one line can be provided if the passenger entry and exit data is combined with train arrival data at each station.

- If TfL wishes, over the 5 to 10 year period being considered, to have near real-time data on passenger entry, exit and origin-destination patterns regardless of ticketing options, then this option incurs little or no incremental cost above that needed to provide the near real-time data from stations and barriers.

Issues and Concerns

- The security and fraud risks of the MSC must be assessed. Acceptable risk levels must be achieved, and several options appear feasible:
    - If the MSC is used only for lower-value tickets (and the existing Oyster Card or similar card with memory and CPU is used for higher-value tickets), the incentive to counterfeit or otherwise commit fraud with the MSC is limited.
    - The MSC may transmit its serial number to the card reader at a fare barrier or reader on a bus without encryption, or MSC cards are now being developed with some security features.
    - In most cases, the read range of the reader is a few centimeters, so the opportunity to intercept the serial number of the MSC is limited. However, readers and antennas (operating at illegally high power levels with high-gain antennas) have been developed to demonstrate that it is possible to read a smart card at a range of 30 meters.
    - An attacker could read MSC serial numbers and clone them, as one form of fraud. The attacker would need to read valid serial numbers, burn a card, and then sell it at a substantial discount to a TfL user. Since this would require reading individual cards and then creating counterfeit cards with those serial numbers, substantial effort is required for relatively little gain.
    - MSC cards must use 128 bit or larger serial numbers, and these numbers must not be in a pattern, such as each card's number being one more than the previous one, that an attacker could clone a large number of cards efficiently. With 128 bit or larger numbers, the chance of an attacker guessing a valid number is very low.
    - Cards with a more capable processor could transmit one of several serial numbers in rotation, include delays and other measures to improve security, possibly including limited encryption capabilities. With no stored value on the card, the number of security vulnerabilities is lower than on a conventional smart card.

- Some standards, such as EPCGlobal, impose a structure on the 128 bit card that limits the ability to use pattern-less serial numbers. Transit MSCs might require a modification to some standards.
- Other security risks do not exist in MSCs that do exist in stored value cards:
  - There is no stored value or other data that must be protected.
  - Attacks based on having a card fail to decrement its stored value, even though it reports doing so to the reader, are not possible.
  - MSCs are much simpler than stored value cards, which allow the security analysis to be much simpler; there are fewer risks, even though the card is less capable of sophisticated actions to mitigate risks.

3.3.3 Bank Cards

Using bankcards for transit fare payments, instead of a TfL-issued smart card, is a substantially different approach to ticketing. Smart bankcards are another option for transit fare payment.

Variations in Configuration

There are several categories along which bank cards for transit can be configured:

1. Transaction vs. prepaid: In a transaction smart card, no e-wallet/e-cash is stored on the card – each travel purchase is a new item on the user's monthly bank statement, as with a credit or debit card. A means for quick card authorization is necessary. Prepaid cards, like Oyster today, use an on-card e-wallet. Passengers must manually charge up their cards, or arrange an automatic top up. Since the money is on the card, no authorization is needed. Transaction cards are more convenient for passengers, because they never have to charge their cards, but may incur higher transaction fees from banks, because the fraud risk is higher, and require quick card authorization. (However, stored value transaction fees at some banks are identical to credit transaction fees if the bank absorbs liability for lost or stolen cards.)

2. Arrangements with bank for tap-out: If zonal or distance-based fares are to be charged, TfL must know the tap-in station when a passenger taps out, which can be accomplished in one of four ways.
    a. *Post processing by bank or third party processor*: The bank records the entry gate, exit gate, and times, and computes the correct fare subsequent to the passenger tapping out, given the passenger's pass or payment type.
    b. *TfL can read ID on card*: TfL is allowed to read a unique ID number on the cards, which can be stored in a database after tap-in, so that when a passenger taps out, the database can be referenced to compute the correct fare, which is sent to the bank. Passes can be stored in the database
    c. *TfL can write to the card*: TfL can write to a section of memory of the smart card. It writes pass type and/or tap in gate, which can then be read at tap-out by the bank to compute payment.
    d. *Bank response at tap-out*: TfL cannot read or write to the card, but the bank stores the tap in gate (and pass type), and when a passenger taps out, the bank's network gives that information to TfL, which then responds with the appropriate charge.

3. Exclusivity of bank cards:
    a. If bank cards are the only fare payment method possible, then an alternative for passengers without bank accounts is necessary, such as bank-issued stored-value cards.
    b. If cell phones are a payment medium, TfL must make similar arrangements; TfL may process cell phone charges in a different way than bank card charges, although this may have customer service issues.
    c. If a generic smart bank card without transit-specific features can be used, all banks will be participants. This maximizes customer ease of use, but may require a simplified fare policy.
    d. The fewer the number of banks allowed to have transit-capable smart cards, the more valuable the contract is to the bank, and the more leeway TfL would have in

the initial negotiation, but the more dependent TfL would be on those banks after the system is in operation.

Operation

Using bank cards for transit fare payments instead of a TfL-issued smart card is a substantially different approach to ticketing: Bank cards would be used to tap in and tap out at underground stations, and to tap in on bus and other surface vehicles.

- The bank cards would use similar technology to the existing Oyster Card, but would be general-purpose smart bank cards issued by banking institutions. These would be MasterCards, Visa cards, and the like, used by consumers for their other credit and debit transactions.

- Existing Oyster Card readers and fare barriers are able to read bank cards with little modification. However, the authorization and processing of payments would be substantially different; it would be handled entirely by the banking system.

- TfL would not be involved in the transaction processing at all; it would be 'just another retailer'. Passengers would tap their cards and the transaction would be handled by the bank as any other bankcard purchase.

In the previous chapter, the operational characteristics of "transaction" payment mechanisms were described as in the case of TaiwanMoney and T-Money bank cards. A description of how bank cards with "stored value" could be used is described below.

Transit payments are typically low-value transactions that are based on a stored value payment paradigm. Transit customers prepay a certain amount, either by presenting cash at transit station or retailer terminals or by using web-based tools to add value from pre-selected bank accounts. The prepaid amount is then stored in an "electronic purse" (e-purse), either on the fare medium (e.g., a smart card) or in a central account on a host system that communicates with the fare medium. Both credit and debit cards are widely accepted in the transit industry for purchasing fare media and loading cards.

Transit e-purse transactions resemble typical cash transactions with two important differences:

The transmission between a local reader and a card completes the e-purse transaction. In many cases, a transit transaction occurs offline (e.g., on a bus) and a central system is later updated by a batch file. In other cases, a virtually real-time data exchange is possible. In most cases, however, the transit system extracts payment at the time of use rather than accumulating charges and billing later. A system such as the Washington, D.C. system supports card-to-reader exchange but maintains full detail of each card's history via a fully auditable transactional database.

During the transaction, data elements describing past usage history are transmitted to permit the terminal to apply fare rules in calculating the specific fare due. These transactions must be made in real time by the terminal with total transaction time between 70 and 300 milliseconds.

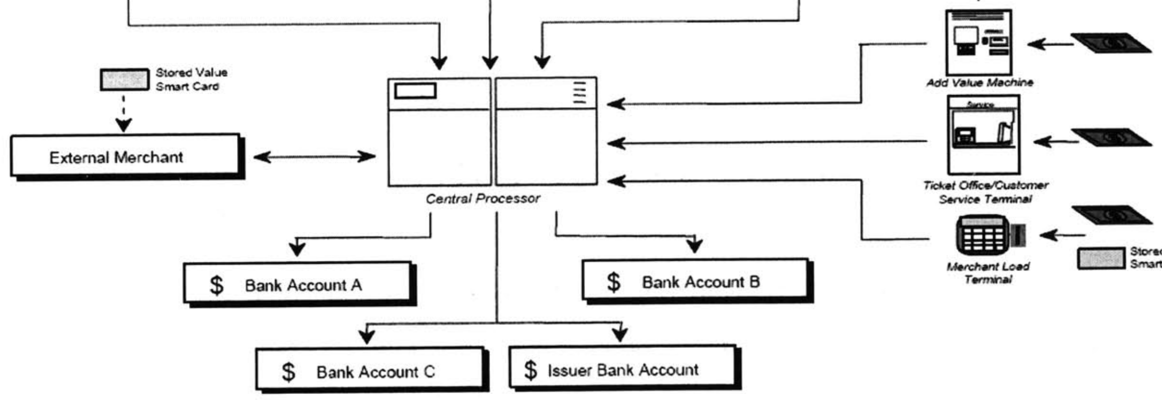


Figure 3.1 Multi-operator smart card architecture [1]

In a multi-operator system, clearing functions can be handled by one of the participating operators or contracted to a third party. If sufficient internal resources/capabilities are not available at one of the operators, a central clearinghouse can be established. Figure 1 illustrates a smart card-based transit payment system architecture incorporating multiple operators, multiple modes and a central clearinghouse. Transaction data is collected and reconciled at the central system, and the corresponding fare revenue is deposited in the appropriate operator's account. If external merchants participate in the program, the clearinghouse can also clear funds for the merchants as well.

Advantages

MasterCard International and Visa International announced that they have reached an agreement to share a common communications protocol or radio frequency-based contactless payments at the point of sale. The use of a common protocol for conducting contactless payments will enable vendors to streamline product development and testing, leading to reduced implementation costs and faster time to market for financial institutions and merchants. With a common protocol in place, merchants will also have the assurance that a single point of sale terminal may support multiple payment brands and will require less time for terminal programming and testing.

Since the cards are issued directly issued by the financial institution, the transit agency's staffing and card handling cost are greatly minimized. In some cases where transaction is done by bank, there is a significant reduction in TfL's data handling costs. Despite the lack of PIN entry, fraud levels are expected to be moderate. Technical security measures on the card make them immune to electronic theft, such as by card ripper. (see technical details below).

Limitations

1. Typically banks charge a minimum transaction sum in addition to a small percent of the transaction amount. This would mean a high percent of the value of transactions would be given to banks if the transaction amounts are small as in the case of transit payments.

However Asian public transport systems have negotiated acceptable commission levels with banks.

2. In addition, if banks do the processing, there would be a limitation on the fare structure. In fact with "stored value" bank cards, only flat fares can be supported.

Mobile Phone

Cell phones are being used in London for payment of congestion tolls via short messaging service (SMS), a data service widely available on cell phones. This is not a real-time system and requires manual input by the traveler; it is not feasible for transit use. However with emergence of Near Field Communication (NFC), mobile phones could become a major mode of ticketing in London. Although very few mobiles now have this technology, one major American market research firm says that by 2009, 50% of mobiles manufactured will include NFC. The main advantages of NFC are as follows

- The NFC communication has the potential to operate in passive mode i.e. it can continue to operate even when the cell phone battery is dead, a common occurrence.
- NFC establishes connections more quickly than Bluetooth or infrared. Since it uses essentially the smart card communications protocol, it will operate in the same manner as the Oyster Card today in connecting to a fare barrier reader.
- The NFC data transfer rate, 212 kbps, is comparable to existing smart cards, which is acceptable for transit fare payment. While Bluetooth and infrared have higher data transfer rates, these are not needed.
- Customers would need to obtain a new cell phone, equipped with NFC, to use it for transit fare payment. Over the next 5 to 10 years, if NFC is adopted by other retail segments and is supported by cell phone manufacturers, as seems likely today, most customers will have an appropriate cell phone.

Operation

- Users would tap in and, if needed, tap out with their mobile phones as they presently do with Oyster Cards. The processing time at a barrier or on a bus would be similar to existing Oyster Card times for stored value and passes; transactions would be treated as with bank smart cards, and would also be processed in the same time.
- The primary difference is the financial situation. Now, instead of loading the card at a TfL reader, the card is charged by the mobile phone provider. The price will appear as a line item on the user's phone bill. TfL may need to pay the phone service provider a fee for each charge of the card (but certainly not for each use of the card).
- Payment would be made either from stored value or a stored fare plan on the mobile phone, the same as current Oyster Card usage, or via debit or credit transaction, similar to a bank card transaction. If users had both stored value and transaction capabilities, TfL would use the best option for the customer. If multiple payment options are possible, the customer account would indicate their preference and TfL would honor it.
- If the fare system were simple enough, it would be possible for NFC mobile phone users to use TfL services without having a TfL account, at least for single tickets, daily or potentially weekly passes, or other fare types suited for ad hoc use by visitors or casual users.
- The customer experience with cell phones will be similar to that of smart cards, both on the underground and on surface vehicles. The processing time at a fare barrier will be similar.

Advantages

1. The possibility to work in passive mode, i.e., when the mobile phone battery is dead. With Bluetooth and Infrared, the mobile battery must power the handset to allow payment. NFC can operate just as Smartcards now operate – with power generated via induction from a transceiver's signal, so prepay or tickets charged on an NFC phone can be used when the battery is dead.
2. The convenience of a single device will be present, and potentially a choice on whether to prearrange payment for transit use or treat it as just a retail purchase.

Limitations

1. "Slow" data rate of 424 kbps. This limit is likely not to matter for transit applications, where we consider transmitting at most 4kB or 8kB, which still takes only a fraction of a second.
2. Necessity of specialized handset. This is actually the case for all 3 technologies, but Bluetooth-enabled handsets are relatively common (infrared handsets are a rarity outside Asia). Currently, Nokia, Motorola, and Samsung all offer NFC-ready phones. Other handset manufacturers will likely follow. But passengers who want the convenience of a payment by mobile phone instead of a smart card will have to purchase a new handset. Handsets turnover in 18 months on average so this may not be an issue.
3. TfL operational processes would probably be simplified. The management of fare plans or stored value on NFC cell phones is simpler for TfL than Oyster Card, and TfL may not need to be involved in the process at all. Data on passenger entries, exits and origin-destination flows would be provided by the payment vendor, as in the bankcard dossier. It is possible that raw counts from fare barriers could be sent in real time to TfL operations staff, as a separate data stream from the payments information, which would probably be processed as a batch, perhaps nightly, and sent to TfL with perhaps one day's delay.

Long distance smart cards

- The major difference is that they can be read at a range of approximately 3 meters instead of the existing 5 -10 centimeter range of Oyster Cards.
- They would operate in a similar manner to existing highway toll tags read at toll booths, although different implementation and technology options are required for public transportation.
- These cards could be used for both high value and low value tickets, though their cost is high for use in low value tickets.

- For rail use, readers for these cards could be placed at wayside, along the tracks between stations, and could read the smart cards of all passengers on a train, similar to how highway vehicle transponders are read while traveling through a toll plaza. Readers would be placed close to stations, to keep the train velocities within the range of operation for the readers.
  - The readers would be pointed to read through windows of rail stock. If LDSCs are in passenger's pockets or purses next to the side of the car, not in line of sight of the reader, they may not be read reliably. Microwave frequency cards and readers (2GHz and above) may overcome this issue, but may have greater difficulties with other factors, such as rain on the windows.
  - A less possibility is to mount LDSC readers inside the rail vehicles at several points in a car. The readings would be transmitted via wireless connections either while in motion or at a 'hot spot' at the next station.

Operation

- Passenger boards and exits the train with/without passing through any barriers. However it seems likely that there will be barriers only for entry to charge a minimum fare.
- Train passes across the reader which detects her smart card.
- When the passenger boards off the train, she will no longer be detected when the train next crosses the reader. This will indicate the end of journey by that transit.
- If the traveler, transfers to another transit within a fixed time interval (perhaps assume one hour), a discounted price may be applied for that trip.
- At the end of all transfers, the appropriate fare price will be charged.
- Random inspections are made to check availability and validity of cards.

Advantages

- By reading cards along track segments, the route or path taken by a passenger is known, and pricing by path can be instituted, primarily at peak periods as a congestion pricing measure.
- Alternative paths between many origin-destination pairs exist in the London underground. For example, some of these paths traverse central London to reach a station on the other side of the city center, and are capacitated during peak periods. An alternate route might exist that avoided the center to reach the same destination, and is not at capacity.
- Users who had a lower value of time, if given the option, might choose to take the longer path with available capacity.
- This would release capacity on the more direct path, and would allow an increase in ridership by users with a higher value of time.
- This option would allow users with a lower value of time potentially three options to make trips in peak periods:
    - Rail via the most direct path, at the highest fare
    - Rail via an indirect path, at a lower fare
    - Bus, usually the slowest journey option, at the lowest fare

- This option would allow either static fares for alternative paths, or it could allow dynamic fares, to be displayed, for example, at monitors at stations with alternate paths, showing the fares by time period for each service that day.
    - This option will have substantial customer information and acceptance issues, whether the fares are static or dynamic!
    - Dynamic fares would be need to be controlled by software that may need to examine train and line loads, line operating conditions (signal problems, police actions), and to make short-term forecasts (perhaps 30 minutes into the future) where and when congestion will occur. This would be a powerful but complex tool.

    - Another option to measure train loads would be to instrument the load/empty brake valve settings on train cards, which would indirectly measure the number of passengers in each car, and thus on each train and line. This would avoid the use of an LDSC for measuring train loads, but would not allow TfL to determine which passengers are on which train or line if line- or train-specific congestion fees are charged.

- The LDSC allows much more precise congestion pricing than a smart card that is tapped at origin and destination. It allows pricing to be sensitive to:
    - Direction of travel during the peak, by segment of line, so that uncapacitated segments do not have a congestion charge
    - Smaller time intervals. For lower time value users, having a peak charge only at the actual peak time may be a substantial benefit, especially if there are 'shoulder' or other intermediate fares at intermediate levels of travel. A low income user shifting his or her trip by a half hour to gain a lower fare is more acceptable than being excluded from an entire peak period, for example, if the line in question has a short peak period.
    - Dynamic conditions. If lines that are usually congested are not on a given day, low value of time users can be allowed to use them without paying a congestion charge. The issue of a stable versus dynamic fare is complex from the rider's perspective, of course, but dynamic pricing could offer benefits to riders that would lead to its acceptance by them.

- Another speculative benefit of an LDSC is that cards would be read only when the rider is on board the train in the underground system. Fare barriers could possibly be eliminated, although there are substantial issues with doing so.
    - If fare barriers were eliminated, TfL would not need to control station platforms.
    - This would potentially open up the platform to more intensive retail uses, since customers could patronize the retail establishments without being transit users or paying a fare. Users would only be charged a fare if they boarded a train.

    - There are probably other ways in which to enhance the retail value of the real estate at underground stations, by moving the location of fare barriers, or possibly by changing the programming on the current Oyster Card readers to not charge a fare if the person exits the same station he or she entered within a defined period (and did not tap out at another station).

Issues and concerns

- The LDSC may possibly be used on bus routes to implement congestion pricing. The technical challenges are substantial with current technology; future technology may be able to address these, but there are substantial uncertainties:
    - LDSCs cannot be read through metal and, on the underground, readers must be positioned carefully to read through windows on both sides of a train. Buses are not as constrained as they travel on a roadway, of course, and so readers would read a lower fraction of cards on a bus.
    - LDSC readers would need to be placed at all bus stops to capture all short trips and to allow all trip origins and destinations to be captured. There are many more bus stops than underground stations, of course.
    - Alternatively, LDSC readers could be placed at multiple points on a bus, and the data read and transmitted via Internet continuously or at 'hot spots' at bus stops.
    - Alternatives to LDSCs may be feasible: route- and time-specific pricing with more conventional smart cards, and others.
        - Conventional smart cards are read on buses, not at station platforms as in the underground, so vehicle-specific charging is possible with conventional cards.
        - If most bus routes do not traverse the congested area, but terminate or originate in it, the direction of travel and time of day might be a sufficient proxy for whether the bus was congested. This would support static congestion pricing that did not vary from day to day.
        - Dynamic congestion pricing might be possible with conventional cards. If buses have continuous Internet connectivity, the load on each bus route

can be measured and prices adjusted. The price for each bus could be displayed on its electronic signboards.

- The need for GPS, WiMax and other supporting infrastructure on the bus would be essentially the same with LDSCs as for other smart cards.

- LDSCs could have stored value or be memoryless. If they have stored value, the value would be decremented the first time the card was read and then adjusted, as needed, when read at other points. For example:
  - A card could be decremented by the minimum peak fare from an origin station.
  - If the card is read on a capacitated route, an incremental charge is decremented from the card's stored value
  - The same issues exist as with other cards if a user enters the system with a card that has only the minimum possible value but does not cover incremental congestion or zone charges.

- An additional issue with LDSCs and congestion pricing is that unlimited pass ticket policies might need to be modified.
  - A user could purchase a pass good for all lines, including capacitated lines, which would remain a truly unlimited pass.
  - Other users might purchase a pass good for the incapacitated peak fare. If they use a capacitated line, they must have prepaid value on the card (or on the server for their account, if the card is memoryless) to cover the increment.
  - Pass users should be subject to the congestion charge, since they make up a substantial portion of peak period travelers.
  - If congestion charges are dynamic, essentially all pass users might see dynamic pricing. Substantial user education and acceptance issues exist with this option.

Chapter 4

Cost and Revenue Estimation

In the previous chapter, several options for TfL's future ticketing systems have been proposed. However even after analyzing their risks and benefits qualitatively, none of the option seems to dominate over the others. Each alternative has a unique set of advantages and disadvantages. Moreover, possibilities of variations in type of ticketing media used and fare processing agencies make the analysis even more complicated. Figure 4.1 shows the the impact of various factors on operating costs of ticketing and fare collection (TFC) systems.

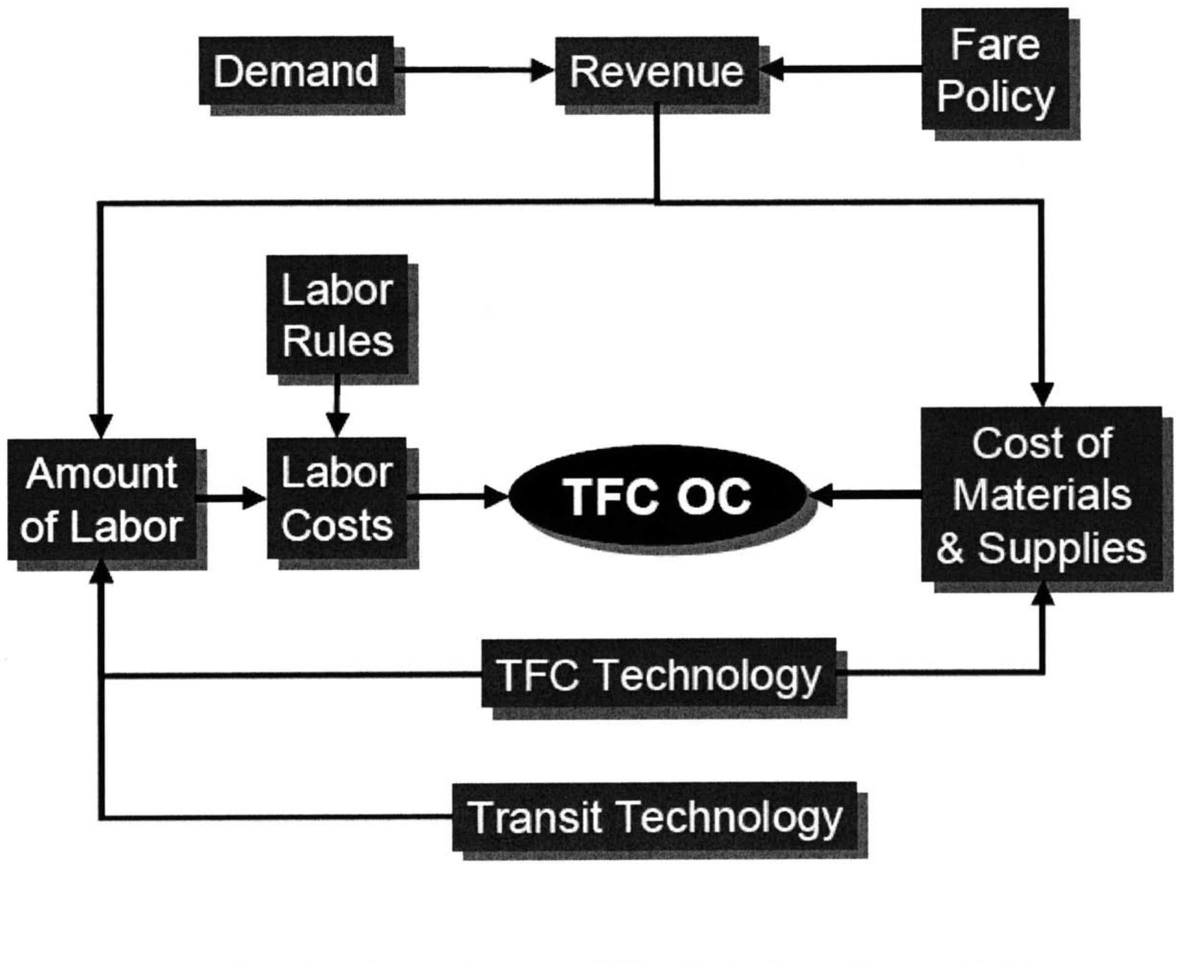


Figure 4.1 Calculation of operating costs (OC) of ticketing and fare collection systems [2]

4.1 Price Structure

Several price strategies can be implemented using electronic fare collection systems. Table 4.1 summarizes the transaction mechanism for the various pricing strategies. In addition to these strategies, other features like price caps, unlimited ridership, transfer discounts, special group discounts and loyalty benefits can be implemented depending on the fare processing mechanism.

Fare System	Description
Flat Fare	Payment takes place at the beginning of the journey. A fixed amount is deducted from the contactless smart card, regardless of the distance traveled.
Zonal/ Distance based Fare	At the beginning of the journey the entry point (check-in) is recorded on the contactless card. Upon disembarking at the final station (check-out), the fare for the distance traveled is automatically calculated and deducted from the card. In addition, the card can be checked at each change-over point for the existence of a valid 'check-in' entry. To foil attempts at manipulation, the lack of a 'check-out' record can be penalized by the deduction of the maximum fare at the beginning of the next journey.
Congestion Pricing – fixed	Fare is charged based on the route of the journey and time of day. Travelers are rewarded for traveling during off-peak hours or on less congested routes. The path of traveler is calculated based on minimum distance between stations
Congestion Pricing - real time	This is similar pricing strategy except that the path of the traveler is calculated by monitoring where the traveler is during different points on the journey.

Table 4.1 Transaction mechanisms of fare pricing systems

4.1.1 Revenue Estimation Based on Pricing Strategy

The revenue generated through various pricing strategies will primarily depend on the elasticity of transit demand i.e. the change in transit ridership per unit change in price. This is particularly important, because certain technologies cannot support rich pricing structures.

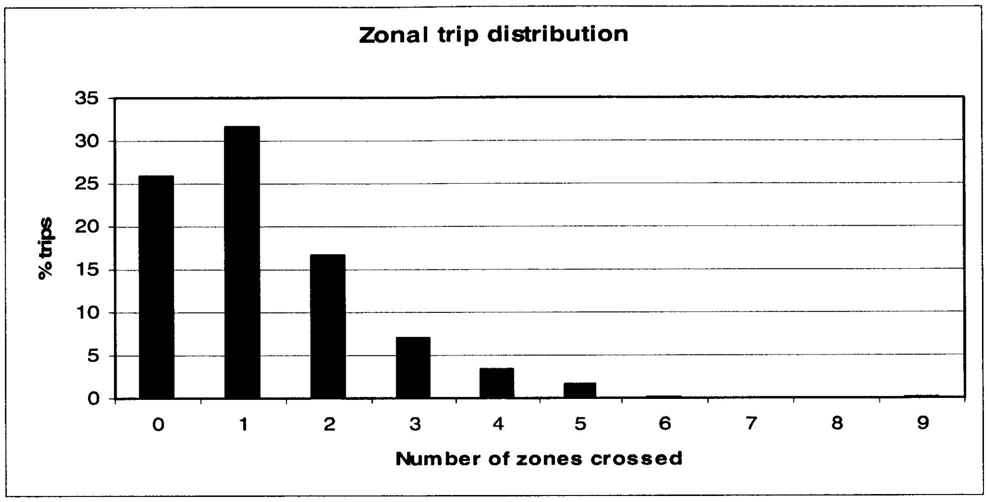


Figure 4.2a Distribution of number of weekly trips on tubes based on number of zones crossed

The graph in Figure 4.2a represents the trip distribution based on the starting and ending location of rail journeys. From another sampled data of 5% of the rail journeys, the distribution of train trips as a function of the distance traveled was observed. The variation in trips with trip lengths is shown below.

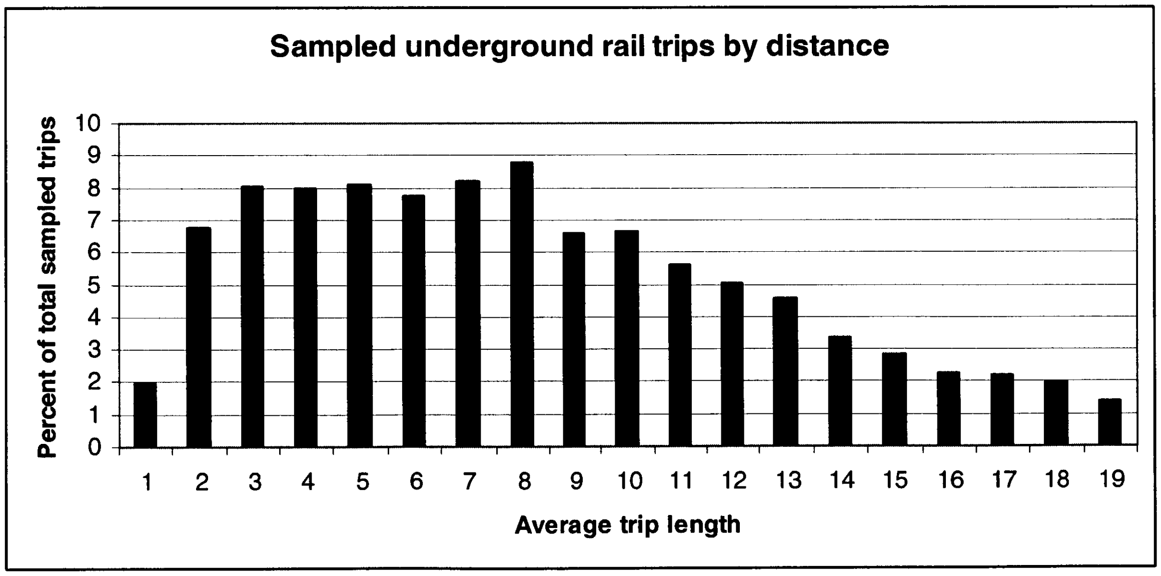


Figure 4.2b Distribution of number of weekly trips on tubes based on distance

From the above graph, we can say that the trips vary almost uniformly with distance and then gradually decrease. There is no obvious way to group stations and implement a zonal fare structure which will closely mimic a distance based fare structure. Hence it is not very clear why a zonal fare structure, which London currently has, will offer significant advantages over a flat fare structure.

4.2 Compatibility Issues

The ticketing options vary, not only in the technology used but also in terms of the ticketing media used and fare processing agency. The pricing structure, ticketing media and fare processing agencies are dependent on the ticketing technology deployed and hence may not support each other.

Fare Policy	**Ticketing media**				
	Oyster Cards (Extended)	Memoryless cards	Bank Cards	Cell Phones	Long Distance smart cards
Flat fare	√	√	√	√	√
Zonal fare	√	√			√
Congestion pricing –fixed		√			√
Congestion pricing -real time					√
Price Cap	√	√	√	√	√
Unlimited Pass	√	√			√
Special Groups (discount)	√	√			√
Transfer Discount	√	√			√
Loyalty incentives	√	√	√	√	√

Table 4.2 Fare structures supported by various ticketing media

Fare Policy	Fare Processing organization					
	TfL (stored)	TfL transaction	Bank (Stored Value)	Bank (transaction)	Cell Phones (Stored Value)	Cell Phone - transaction
Flat fare	√	√	√	√	√	√
Zonal fare	√	√				
Congestion pricing -fixed	√	√				
Congestion pricing -real time	√	√				
Price Cap	√	√		√		√
Unlimited Pass	√	√				
Special Groups discount	√	√		√		√
Transfer Discount	√	√				
Loyalty incentives		√		√		√

Table 4.3 Fare structures supported by various fare processing organizations

Ticketing Media	Fare Processing organization					
	TfL (stored)	TfL transaction	Bank (Stored Value)	Bank (transaction)	Cell Phones (Stored Value)	Cell Phone - transaction
Oyster Cards (Extended)	√	√		√		√
Memoryless cards		√				
Bank Cards		√	√	√		√
Cell Phones		√		√	√	√
Long Distance cards		√				

Table 4.4 Ticketing media supported by various fare processing organizations.

4.3. System Architecture

The costs of implementing the ticketing solution are a function of the size of the entire public transit system governed by TfL, the operating system deployed, the prices of equipment at the time of procurement. It is likely that TfL would want to change its existing equipment with new equipment when the new system is implemented. The choice of the operating system will largely depend on transaction processing mechanism and ticketing media that TfL chooses.

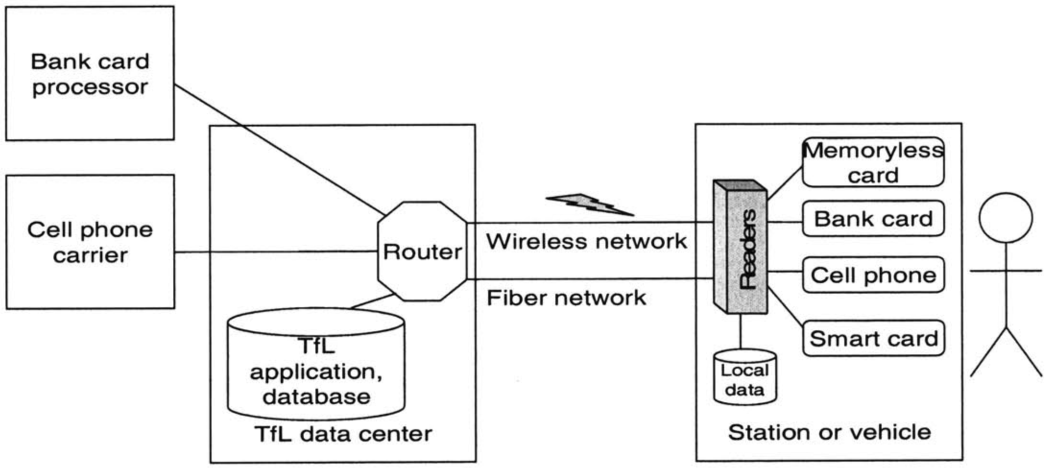


Figure 4.3 General System Architecture supporting all options

Figure 4.3 shows a very general framework capable of supporting most ticketing scenarios. A brief description of the architecture is as follows:

At each station or on each vehicle, a multimode reader will accept all valid media, with appropriate security. If the card has stored value or a fare plan, it can be processed locally at the station or vehicle. Data is sent via the network on the boarding or alighting, regardless of fare medium, to the TfL server, so basic operations and planning statistics can be kept.

If a transaction is required, the reader first verifies the fare medium is valid. For sites connected via fixed networks, this data can be close to real-time; for surface vehicles, it may be close to real-time if good wireless coverage is available; otherwise the data may be a few hours old. If the fare medium is valid, the reader attempts to contact the appropriate server to obtain a transaction authorization. If the transaction takes too long or the network is unavailable, the reader will accept the fare payment based on the serial number of the ticket and will process the transaction later, so boarding or exiting are not delayed. The local store is expected to be solid state memory for durability, and of sufficient capacity to hold most or possibly all TfL-issued cards, other frequent users' mobile phone or bank card IDs, and a 'black list' of invalid cards.

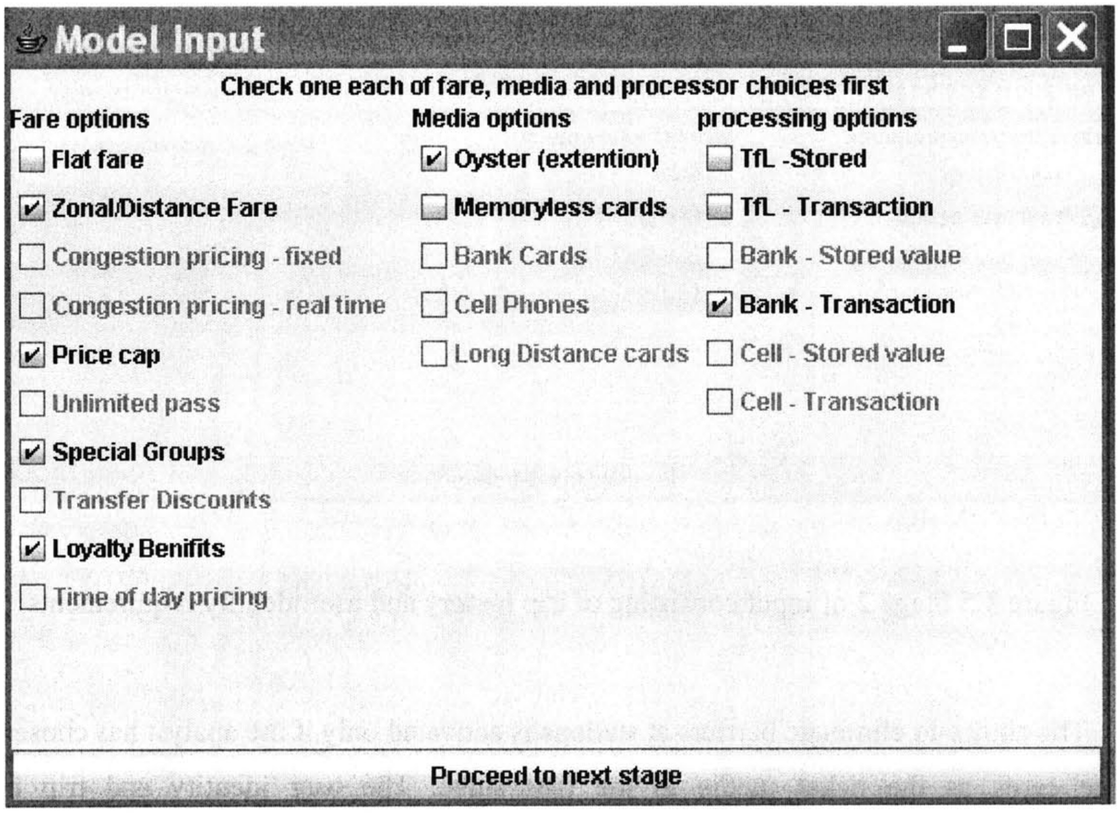
The greatest advantage of the proposed architecture is it offers great flexibility among all the ticketing options. The TfL application database may vary depending upon trip history requirements.

4.4 Feasibility and Cost Estimation Model

Choosing the best ticketing solution is challenging since the costs and revenues generated from various ticketing solutions are governed by the policies TfL adopts for its operations. These policies are with respect to the choice of fare structure, ticketing media, fare processing organizations, amount of trip history to be recorded, user identity issues and requirements for barriers. Hence a software model has been developed which can help decision makers at TfL to compute costs based on the policy choices they make. The model will also guide them as to what ticketing solutions are feasible in the first place. The model is built in the programming language Java.

4.4.1 Model Description

The model consists of a series of user interfaces, containing checkboxes and textboxes required to be filled by the TfL analyst. In the initial GUI the analyst is required to specify the fare policies, ticketing media to be supported and the organization responsible for processing the transactions. As the analyst chooses the policies, the interface modifies the available choices to disallow the user from choosing incompatible policies. For example, if a user chooses the transaction processing organization as cell phone (transaction), she cannot use any ticketing media except cell phones. The diagram below shows the first stage of the model in which some fare, media and processing options have been chosen.



Model Input

Check one each of fare, media and processor choices first

Fare options	Media options	processing options
☐ Flat fare	✓ Oyster (extention)	☐ TfL -Stored
✓ Zonal/Distance Fare	☐ Memoryless cards	☐ TfL - Transaction
☐ Congestion pricing - fixed	☐ Bank Cards	☐ Bank - Stored value
☐ Congestion pricing - real time	☐ Cell Phones	✓ Bank - Transaction
✓ Price cap	☐ Long Distance cards	☐ Cell - Stored value
☐ Unlimited pass		☐ Cell - Transaction
✓ Special Groups		
☐ Transfer Discounts		
✓ Loyalty Benifits		
☐ Time of day pricing		

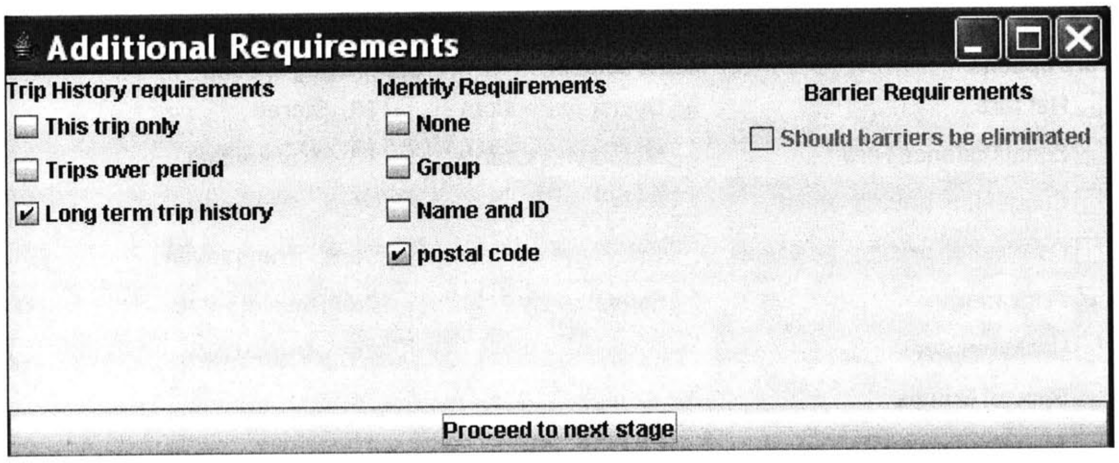
Figure 4.4 First Stage of the input checks for incompatibility

Every time the analyst chooses a particular option, other options which are incompatible are disabled i.e. they cannot be chosen. For example, in the above figure when Oyster card is chosen, options like bank-stored vale and cell-transaction are no longer available. In order to choose options which have been disabled, the analyst must reselect all options. Before proceeding to the next stage, the following conditions must be satisfied:

1. At least one pricing structure i.e. flat fare, zonal or congestion pricing must be chosen. Additional pricing features like price cap and transfer discounts can also be chosen.

2. Memoryless cards cannot be chosen as the only form of ticketing media. Because of the poor security, it is hard to have weekly/ monthly passes with these cards.
3. Exactly one processing organization can be chosen

Once the desired fare, media and processing options have been selected, the analyst proceeds to the second stage of input. Here she can specify trip history, user identity and barrier requirements.


Additional Requirements

Trip History requirements
- [ ] This trip only
- [ ] Trips over period
- [✓] Long term trip history

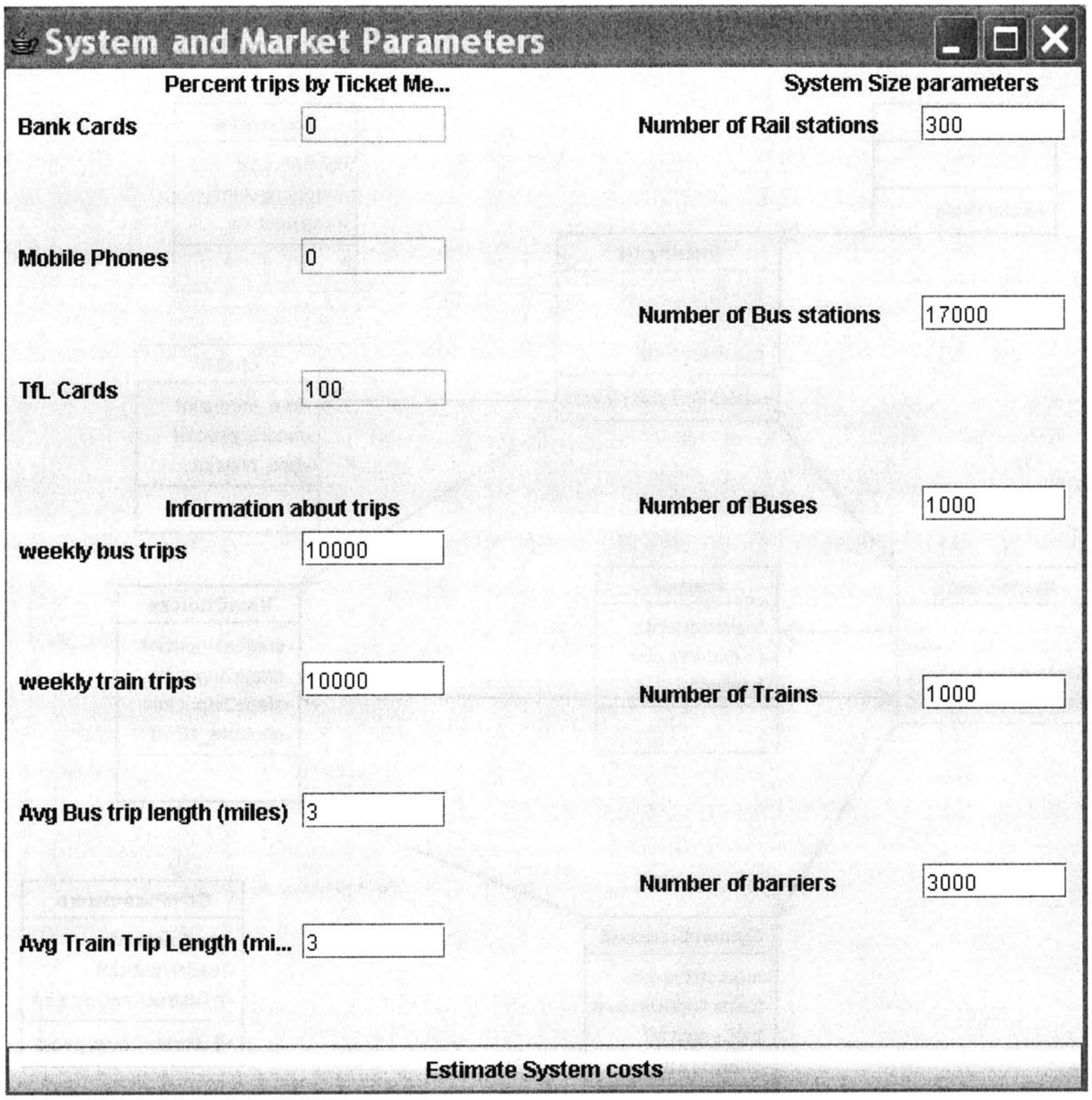
Identity Requirements
- [ ] None
- [ ] Group
- [ ] Name and ID
- [✓] postal code

Barrier Requirements
- [ ] Should barriers be eliminated

Figure 4.5 Stage 2 of input consisting of trip history and user identity requirements

The choice to eliminate barriers at stations is activated only if the analyst has chosen long distance cards as the ticket media in the first stage. The user identity and trip history requirements are primarily for the sake of completeness and do not necessarily affect the system costs. Storing longer trip history may have some effect on the overall system costs. Note that real time congestion pricing which involves tracking people does not necessarily require user identity. User identity is important only for data analysis for future planning and marketing purposes. Sometimes only recording a user's postal code on the database can give a fare estimate of transit demand by location. Finally, the analyst is taken to the last stage where she inputs estimates of transit demand and size of the TfL infrastructure.

The transit demand can be characterized by number of weekly bus and train trips and their average length. If multiple fare media are chosen, it is also important to estimate the fraction of the customers using different types of media. The entities characterizing the system size are number of buses, trains, bus stations, train stations and total number of barriers. All these parameters will determine the total hardware and infrastructure requirements of the system.


System and Market Parameters

Percent trips by Ticket Me...

Bank Cards 0

Mobile Phones 0

TfL Cards 100

Information about trips

weekly bus trips 10000

weekly train trips 10000

Avg Bus trip length (miles) 3

Avg Train Trip Length (mi... 3

System Size parameters

Number of Rail stations 300

Number of Bus stations 17000

Number of Buses 1000

Number of Trains 1000

Number of barriers 3000

Figure 4.6 Stage 3 requires analyst to estimate system size and trip statistics

4.4.2 Classes and Model Dynamics

The model is developed in Java. Several classes are used to implement the model.

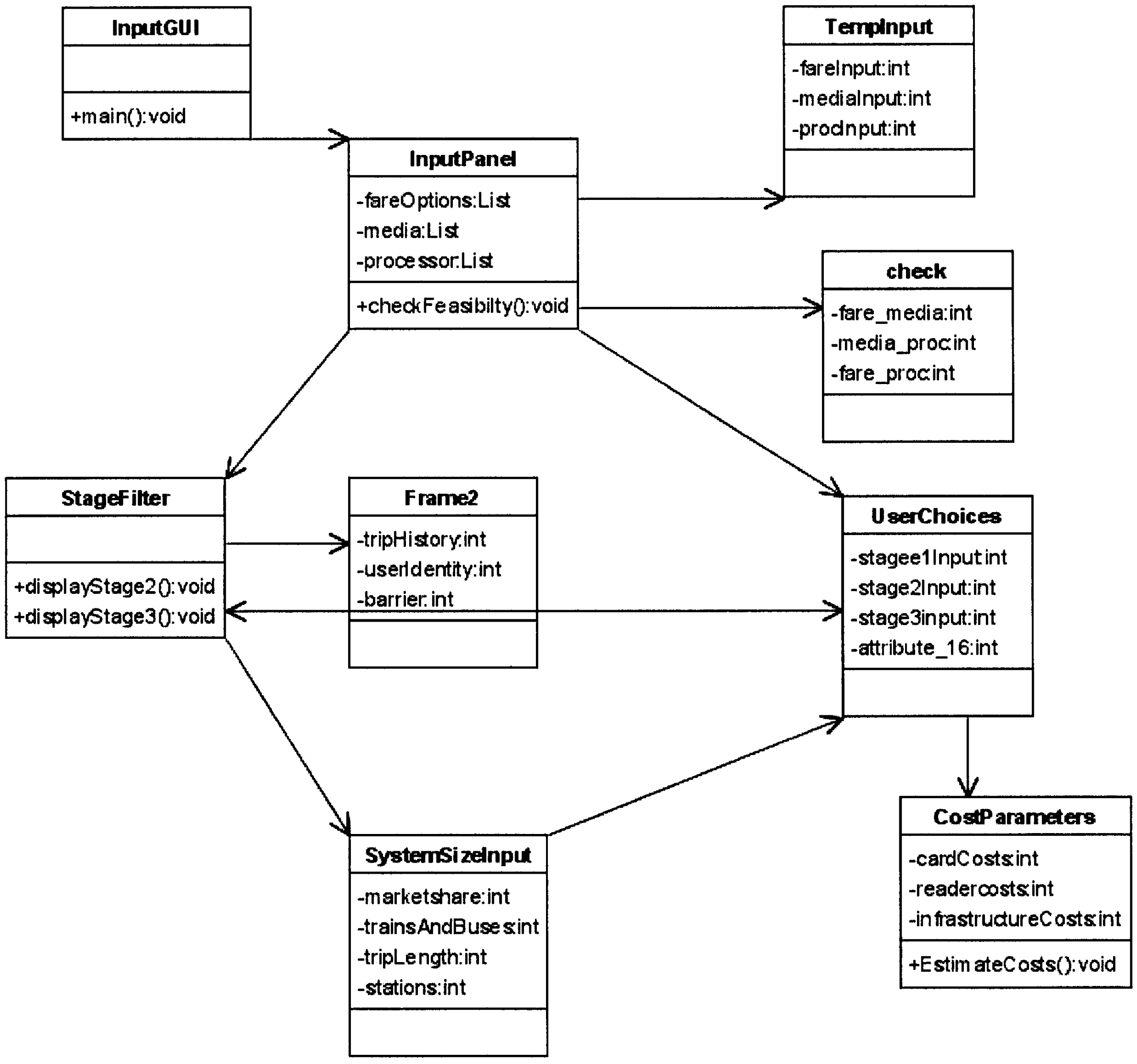
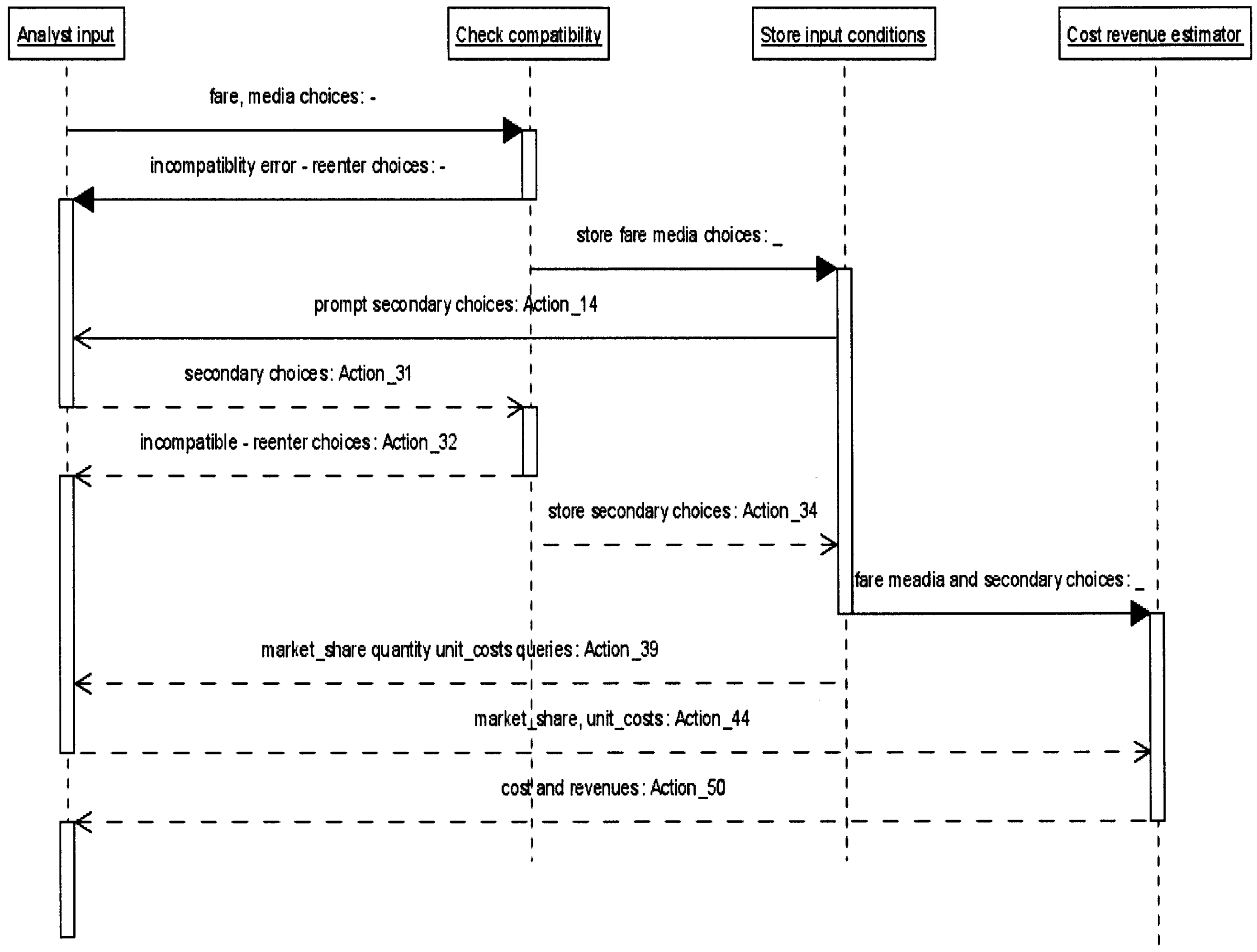


Figure 4.7 Classes used in the model

A summary of the classes used and their key attributes and methods are described in the diagram below. The code for all the classes used can be found in the appendix.

All user- inputs from stages 1, 2 and 3 are passed on to a single class named "User Choices". The costs of implementing the ticketing solution are calculated based on these values. The sequence diagram below summarizes the order in which the costs are generated.



Analyst input
Check compatibility
Store input conditions
Cost revenue estimator
fare, media choices: -
incompatiblity error - reenter choices: -
store fare media choices: _
prompt secondary choices: Action_14
secondary choices: Action_31
incompatible - reenter choices: Action_32
store secondary choices: Action_34
fare meadia and secondary choices: _
market_share quantity unit_costs queries: Action_39
market_share, unit_costs: Action_44
cost and revenues: Action_50

Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 4.8 Sequence in which the costs are processed

4.4.4 Parameters Affecting System Costs

We classify the variables affecting total ticketing costs into two components – parameters that represent the size of the public transit system operated by TfL (referred to as system specific parameters) and the parameters specific to the ticketing solution (referred to as the option-specific costs). For simplicity, the system specific costs and the option specific costs are annual costs. The capital expenditure has been converted to annual expenditure on the basis of average life of capital. The key parameters contributing to *system specific costs* are

Bus Costs

The costs of hardware and equipment required on buses will depend on:

- Number of buses: Each bus will have a ticket unit. A possible configuration is PDA with USB memory stick, RFID reader, Zebra printer or other small printer, GPS chip, WiMax or other broadband data card and a power supply

$$\text{Ticketing Unit Costs} = \frac{(\text{Number of buses} \times \text{Cost of a Ticketing unit} \times \text{Interest})}{\text{Life of a Ticketing unit}}$$

$$\text{Ticketing Unit} = \text{PDA} + \text{RFID Reader} + \text{Printer} + \text{GPS Chip} + \text{WiMax card} + \text{Power Supply}$$

- Number of bus stations: Provision of WiFi at each bus stop for continuous online/off-line data transmission. Alternatively WiFi can be provided just at the terminals.

$$\text{Total Wireless Costs} = \text{Number of bus stations} \times \text{Wireless costs/ year}$$

OR

$$\text{Total Wireless Costs} = \text{Number of bus routes} \times 2\ (\text{Terminals}) \times \text{Wireless costs/ year}$$

The total hardware costs on buses can be calculated as:

$$\text{Bus Costs} = \text{Ticketing Unit Costs} + \text{Operating Costs} + \text{Software Costs} + \text{Wireless Costs}$$

Rail Costs

The hardware and infrastructure required at rail stations will depend on

- Number of barriers at each station: Dual readers may be placed at each barrier, next to each other, for redundancy. Readers would be connected to dual servers at each station via USB hub.

Reader costs = (Number of readers/ barrier) × Number of barriers × Interest

- Number of train stations: Two servers would be located at each station for data storage and ticket validation. A room will be required at each station, with dual UPS power,

Station Unit = USB connection + climate control + UPS + network + servers at each station

climate control, security and dual network connections if possible.

Hence the total hardware and infrastructure costs at rail stations can be calculated as:

Rail Costs = Reader costs + Station Unit Costs + Operating Costs + Server costs

- Average daily ridership on buses and trains: This will affect the number of cards required annually for distribution. The number of cards is also dependent on the life of a card. The number of customer care representatives and station ticketing machines will also be determined by the level of ridership.

Based on the ticketing solution chosen, the option specific parameters may affect ticketing costs.

- Transaction fee charged by bank or mobile phone carrier: If TfL decides to outsource all the processing to either banks or mobile phone companies, it will save on capital costs like cards, ticket machines, etc and instead have to pay a certain transaction fee agreed by TfL and the bank or mobile phone.

- Fraud potential may vary with ticketing media. The highest risk in revenue collection may be with long distance cards and wireless on buses.
- If TfL issued its own cards, TfL will distribute cards, establish machines and hire customer representatives.

$$\boxed{\textit{Machine costs} = \textit{Number of machines/ station} \times \textit{Number of stations / Life of machine} \times \textit{Interest}}$$

- Depending on the trip history that TfL would like to store for planning and/or for loyalty programs the costs of data storage will vary.
- Memoryless cards would require a high performance fiber or wireless network.
- There is a possibility of completely eliminating barriers with the Long Distance Cards. However if readers were mounted inside trains, additional readers and a strong wireless network would be required.
- TfL will most probably outsource the card distribution and processing to cell phone carriers or banks or smart card vendors. The processing and customer service costs will vary by processing organization as follows

$$\boxed{\begin{array}{l}\textit{Bank Processing costs} = \textit{Bank Commission} \times \textit{Total Revenue} + \textit{Customer Rep wages}\\ \textit{Mobile Phones Processing costs} = \textit{Bank Commission} \times \textit{Total Revenue} + \textit{Customer Rep wages}\\ \textit{Card Vendor Processing costs} = \textit{Card vendor Commission} \times \textit{Total Revenue} + \textit{Customer Rep wages}\end{array}}$$

The final costs of the system is calculated as

$$\boxed{\textit{Final costs} = \textit{Processing cost} + \textit{Barrier cost} + \textit{Capital Costs}}$$

4.4.5 Key Assumptions for Cost Estimation

To understand the trade-offs between the various ticketing systems, assumptions about hardware costs have been made to calculate overall system costs (Table 4.1). It is unlikely that

TfL will manufacture, assimilate and process electronic payments in-house. It will rely on banks or mobile phone carriers or smart card distribution firms to process transactions.

Bus Hardware Costs	Personal digital Assistant (PDA)	$500.00
	RFID Reader	$250.00
	Printer	$200.00
	GPS Chip	$100.00
	WiMax Card	$100.00
	Bus Power supply	$200.00
	Unit life of equipment	2 years
	Wireless connection at stations(monthly)	$100.00
Hardware at Rail stations	Rail Station Unit	$600,000.00
	Operating costs as percent of capital costs	20%
	Central server costs	$500,000.00
Cards	Smart card with memory	$1.00
	Memoryless smart card	$0.10
	Long distance smart card	$5.00
	Average card life	1 year
System-specific parameters	Salary of Customer representative	$50,000.00
	Card Machines per station	2
	Life of Card Machine	2 years
	Number of sales outlets	3000
	Commission on smart card transaction	8%
	Commission on bank card transaction	3 – 5%
	Commission on cell phone transaction	3 – 5 %
	Commission on sales at outlets	1%

Table 4.5 Estimated costs of various components

4.4.6 Analysis of Results

Several scenarios were analyzed using the cost model to give an estimate of the operating and maintenance costs of the system. Table 4.2 summarizes costs of some of the important ticketing scenarios.

Ticket media	**Fare processing organization**	**Annual Cost (million $)**
Current Oyster	*Cubic*	*200*
Oyster -extended	TfL – stored value	227.5
Bank cards/ cell phones – 3% commission	Bank – transaction	115.7
Bank cards/ cell phones – 5% commission	Cell Phone - transaction	170
Long distance (barrier-less)	TfL – transaction	250
30% Bank cards + 30% cell phones + 40% Oyster cards	Bank – transaction	161.4

Table 4.6 Results of the Cost Model

In the model, it has been assumed that memoryless cards cannot account for more than 20% of trips (one way trips, two trips and daily passes) because of the security limitations of this card. It is interesting to note that when 30% of cards are issued by bank, 30% by cell phones and 40 % by TfL, and bank processes transactions for all cards, total costs are around three quarters of what TfL currently pays CUBIC. This scenario is very likely even from a conservative point of view (only 30% cell phones will have NFC and 30% users would use credit cards issued by banks).

4.5 Summary

The cost model described in the previous section calculates essentially the capital costs (converted to an annual dollar value) and operating and maintenance cost of ticketing systems. To maximize profits, TfL must also consider the losses in revenue due to

a) Fare evasion: This will depend on security of the system as a whole, use of barriers, random inspection and frequency of system failure.
b) Inappropriate fare structure: The appropriate fare structure will depend on trip distribution by distance and the fare elasticity of transit ridership. An ideal fare structure is one where zonal trip distribution closely represents trip distribution by distance traveled. Inappropriate zone sizes or flat fare would lead to lesser revenue collection.

The losses in revenue can be considered as opportunity due to fare evasion and inappropriate fare structure can be considered as opportunity costs to TfL. Figure 4.9 gives a rough estimate of what the different cost components are likely to be.

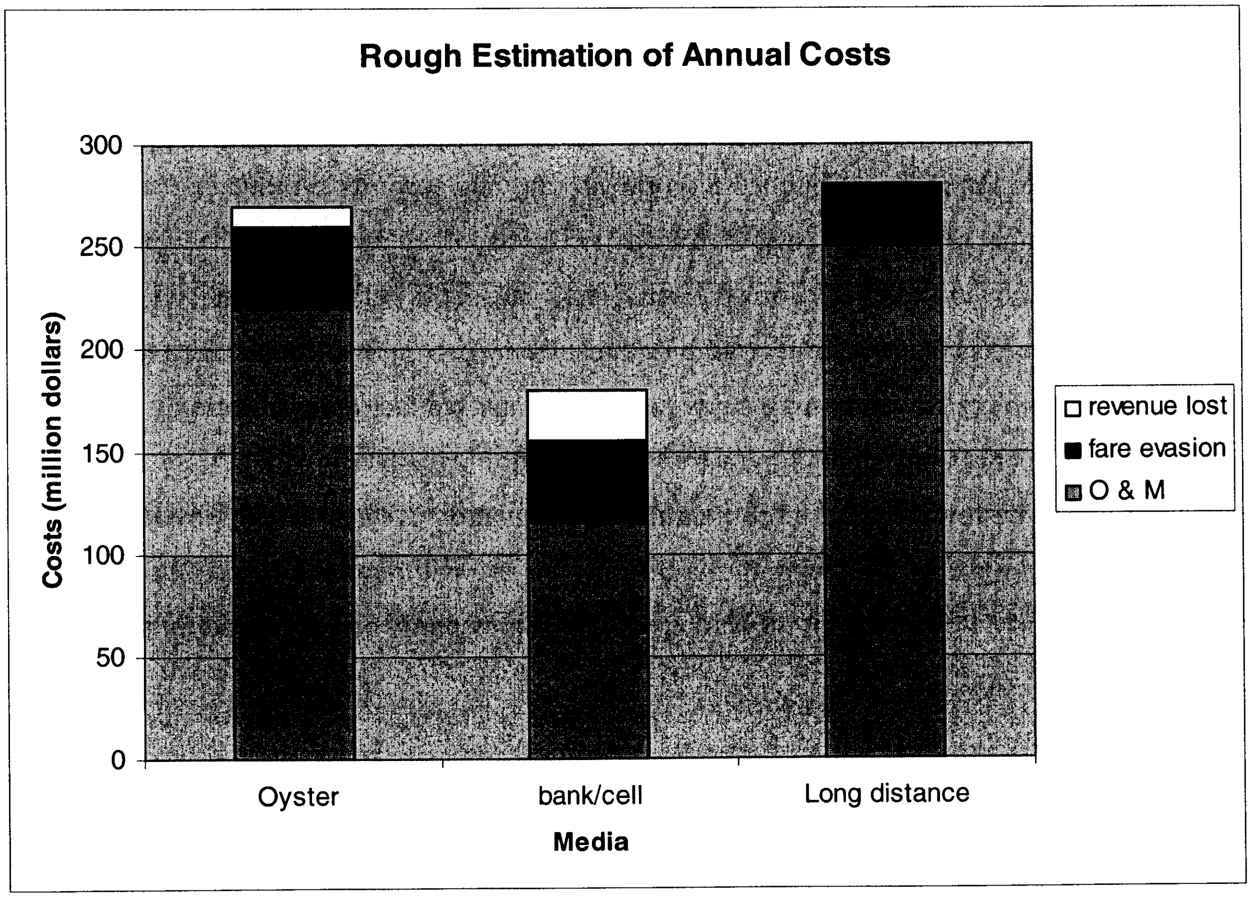


Figure 4.9 Estimation of Annual Ticketing Costs based on processing organization

The costs calculated below are based on several assumptions about technology and business rules which are likely to change to 2012. The model allows for changes in these assumptions. For example, it is very likely that banks and cell phone companies may offer lower commission rates for the huge business that TfL can offer. Secondly, even if TfL owns and distributes smart cards, it is likely that TfL would require the services of smart card companies to manage the system and carry out transactions smoothly. This fee is currently around 8 % under the PRESTIGE consortium as opposed to 3 to 5 % fees likely to be charged by banks or cell phone companies. As indicated by the estimates in Table 4.5, bank cards and cell phones can offer significant cost reductions to TfL. The reduction in costs is primarily caused by avoiding the need to manufacture and distribute cards. Also processing charges for card transactions will be cheaper. Clearly both smart bank cards and mobile phone payment appear attractive options for future ticketing system. The decision to implement such an advanced ticketing system will depend on the details of the contract that TfL can negotiate with financial or cell phone organizations.

Cell phone companies around the world are moving towards e-money on cell phones especially to capture retail payments. If transaction costs of micropayments become cheaper either by aggregation of payments or otherwise, TfL can get a very good deal on transaction commission. Since cell phones are carried by almost every person in London, the entire system will become a simpler. Travelers may enjoy several advantages like delay information, multiple ticket purchases using single media, best route information, etc. The revenue losses due to flat fare systems may be low because more than 80% of trips are bus trips, which already have flat fares. Moreover, around 70% of underground tube trips are within zones 1 and 2, the fares for which are £ 1.5 and £ 2 respectively. It is likely that TfL may partly or entirely adopt this option.

Chapter 5

Java Card Performance

The time taken for the purchase or verification of a travel pass is particularly critical in transport systems in which the pass can only be checked inside the vehicle. This is a particular problem in buses and trams. In the underground railway, passes can be checked at a turnstile, or by conductors. A comparison of different methods shows the clear superiority of RFID systems in terms of transaction times (Table 5.1).

Technology	**Passenger processing time (s)**
RFID I (remote coupling)	1.7
Visual verification by driver	2
RFID II (close coupling)	2.5
Smart card with contacts	3.5
Cash	>6

Table 5.1 Passenger processing times for different technologies.
Source: Transport companies in Helsinki, taken from Czako (1997)

To minimize queuing at barriers in underground rail systems, it is critical to cut down the transaction time per passenger. Smart cards have very slow processors (typically 13.56 MHz or less), as compared to computers (typically 3 GHz), and hence transaction times may take up to several hundred milliseconds. In public transits with very high levels of ridership, transaction times of even seven hundred milliseconds are not acceptable. Apart from processor speeds, the transaction time depends on the security protocol used. High security protocols in general require longer processing times. Hence the goal is to obtain low transaction times without significantly compromising security standards. In this chapter, the time required to complete transactions on a Java Card have been analyzed.

5.1 Java Card Technology

The term Java Card denotes a Java Card technology-enabled smart card. Java Card technology allows applets written in the Java language to be executed on a smart card. When it is inserted or tapped on a card acceptance device (CAD), the CAD selects an applet on the card and sends it a series of commands to execute. Each applet is identified and selected by its application identifier (AID). Commands such as the selection command are formatted and transmitted in the form of application protocol data units (APDUs). An applet can optionally reply to an APDU command with other data.

5.1.1 Authentication Mechanism

Digital signatures have become a vital component of many applications such as transportation, e-commerce, home-banking, intranet, identity, health care, physical access control services since they allow verification that a public key does, in fact, belong to a specific individual. The biggest challenge is to protect the private key of a digital signature to ensure authentication. Smart cards offer superior protection for private keys because they require not only a password/ PIN (which weaknesses were discussed above), but also “physical possession" of the card, in order to gain use of the private key.

.The digital certificate is a data file that contains an individual's public key along with other identifying information, including the owner's name, the certificate's serial number and expiration date and possibly other user-supplied information such as a postal address, email address or employer name and address. In addition, the digital certificate contains the name and digital signature of the certification authority (CA) that issued the certificate. The certification authority is a trusted third party, such as a bank, government agency or employer that verifies the identity of the certificate owner before issuing the certificate. Namely, digital signature is assured by another more familiar digital signature as a rebooting methodology. One of the widely used security protocols that use digital signatures is the Secure Security Layer (SSL) protocol (Figure 5.1).

Many modern smart card security protocols use slight variations in SSL to establish mutual authentication between smart card and terminal.

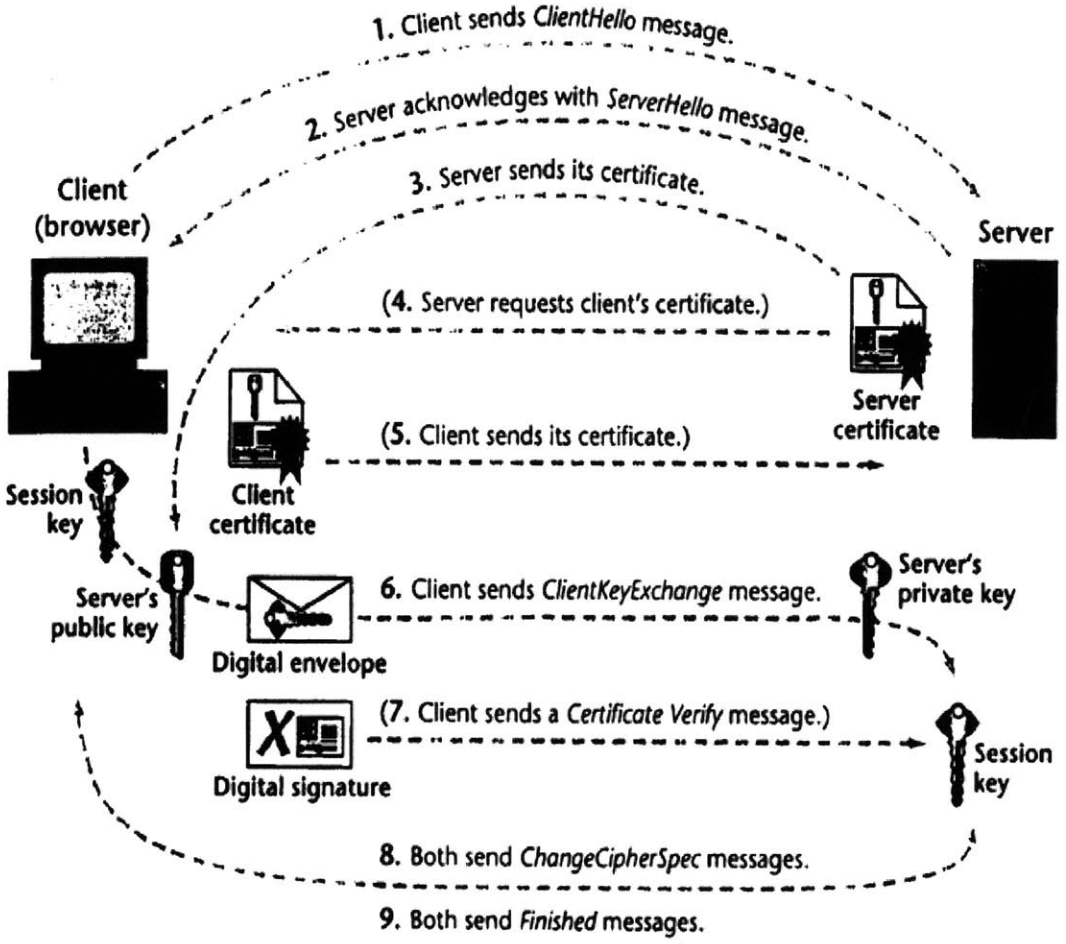

1. Client sends *ClientHello* message.
2. Server acknowledges with *ServerHello* message.
3. Server sends its certificate.
(4. Server requests client's certificate.)
(5. Client sends its certificate.)
6. Client sends *ClientKeyExchange* message.
(7. Client sends a *Certificate Verify* message.)
8. Both send *ChangeCipherSpec* messages.
9. Both send *Finished* messages.
Client (browser)
Server
Server certificate
Client certificate
Session key
Server's public key
Digital envelope
Server's private key
Digital signature
Session key

Figure 5.1 SSL protocol steps

A typical card- terminal authentication mechanism using SSL is described below.

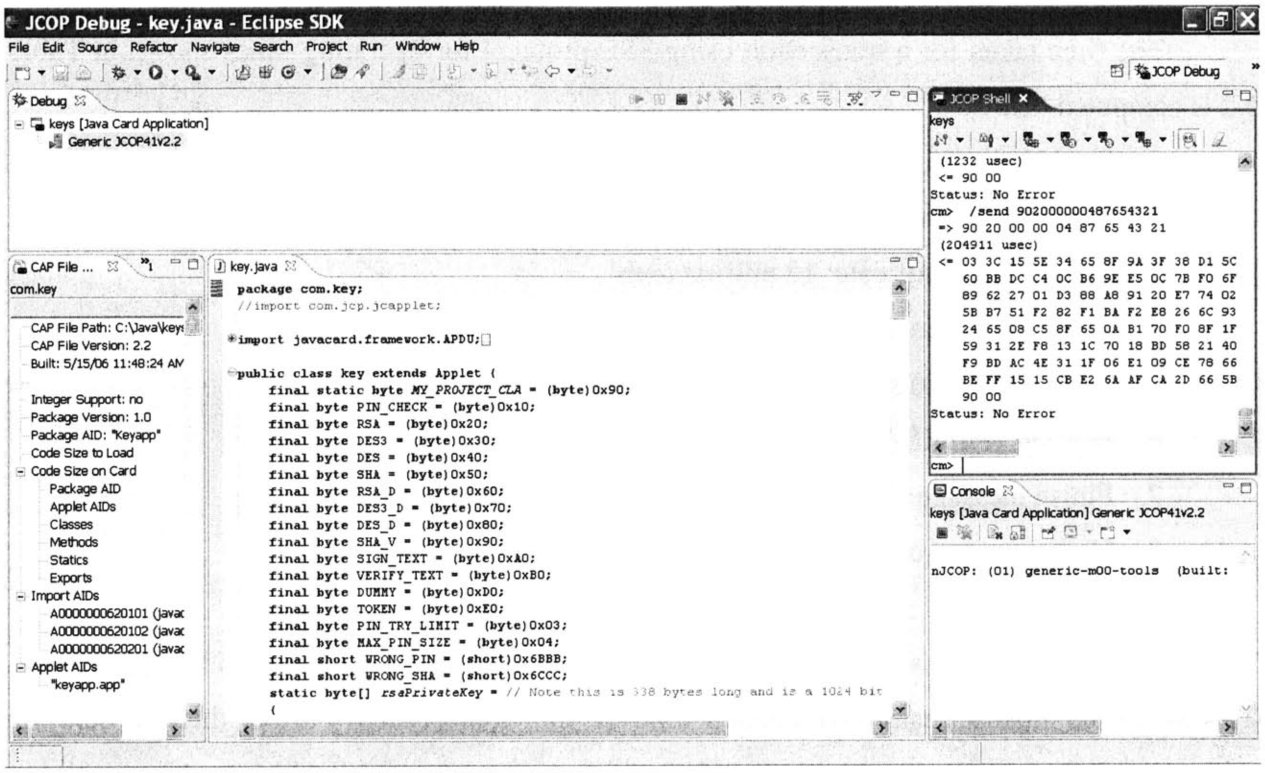
1. The first step in the process is for the terminal to send the card a Hello message. This hello message contains the SSL version and the cipher suites the terminal can talk. The terminal sends its maximum key length details at this time.

2. The card returns the hello message with one of its own in which it nominates the version of SSL and the ciphers and key lengths to be used in the conversation, chosen from the choice offered in the terminal hello.
3. The card sends its digital certificate to the terminal for inspection.
4. The card sends a Certificate request after sending its own certificate.
5. The terminal provides its Certificate.
6. The terminal generates a symmetric key and encrypts it using the card's public key (cert). It then sends this message to the card.
7. The terminal sends a Certificate verify message in which it encrypts a known piece of plaintext using its private key. The card uses the terminal certificate to decrypt, therefore ascertaining the terminal has the private key
8. The card sends its own finished message encrypted using the session key. The terminal verifies the session key then the negotiation is successfully completed. Note that the session key can be obtained by the card only if it owns the private key.

Recently, IBM has developed basic implementations of Java card and Global Open platform framework called JCOP. JCOP 30 is a dual interface (contact/contactless) edition, targeted at the intersection of banking and transportation markets. A Java card simulation is developed using IBM JCOP tools and Open Card Framework (OCF) to test performance of java cards and analyze processing times at various stages during a payment transaction.

5.3 Java Card Simulation

The purpose of the simulation is to analyze the transaction time i.e. the time required to first authenticate the card and then update the balance on the card. The SSL protocol is used a guideline for authentication. The simulation was used to calculate the time taken to process and respond to APDUs sent to the card for operations like connecting to card, verifying the PIN, RSA Encryption, RSA decryption, signing and verifying text encryption/decryption with Triple DES symmetric keys. The properties of the card assumed during the simulation are as follows.
Clock rate: 3.57 MHz
Communication speed: 106000 bits/ second
EEPROM: 16 kB


JCOP Debug - key.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
JCOP Debug
Debug
keys [Java Card Application]
Generic JCOP41v2.2
CAP File ...
com.key
key.java
JCOP Shell
keys
Console
keys [Java Card Application] Generic JCOP41v2.2

```
CAP File Path: C:\Java\keys
CAP File Version: 2.2
Built: 5/15/06 11:48:24 AM

Integer Support: no
Package Version: 1.0
Package AID: "Keyapp"
Code Size to Load
Code Size on Card
    Package AID
    Applet AIDs
    Classes
    Methods
    Statics
    Exports
Import AIDs
    A0000000620101 (javac
    A0000000620102 (javac
    A0000000620201 (javac
Applet AIDs
    "keyapp.app"
```

```java
package com.key;
//import com.jcp.jcapplet;

import javacard.framework.APDU;

public class key extends Applet {
    final static byte MY_PROJECT_CLA = (byte)0x90;
    final byte PIN_CHECK = (byte)0x10;
    final byte RSA = (byte)0x20;
    final byte DES3 = (byte)0x30;
    final byte DES = (byte)0x40;
    final byte SHA = (byte)0x50;
    final byte RSA_D = (byte)0x60;
    final byte DES3_D = (byte)0x70;
    final byte DES_D = (byte)0x80;
    final byte SHA_V = (byte)0x90;
    final byte SIGN_TEXT = (byte)0xA0;
    final byte VERIFY_TEXT = (byte)0xB0;
    final byte DUMMY = (byte)0xD0;
    final byte TOKEN = (byte)0xE0;
    final byte PIN_TRY_LIMIT = (byte)0x03;
    final byte MAX_PIN_SIZE = (byte)0x04;
    final short WRONG_PIN = (short)0x6BBB;
    final short WRONG_SHA = (short)0x6CCC;
    static byte[] rsaPrivateKey = // Note this is 338 bytes long and is a 1024 bit
    {
```

```
 (1232 usec)
 <= 90 00
Status: No Error
cm>  /send 902000000487654321
 => 90 20 00 00 04 87 65 43 21
 (204911 usec)
 <= 03 3C 15 5E 34 65 8F 9A 3F 38 D1 5C
    60 BB DC C4 0C B6 9E E5 0C 7B F0 6F
    89 62 27 01 D3 88 A8 91 20 E7 74 02
    5B B7 51 F2 82 F1 BA F2 E8 26 6C 93
    24 65 08 C5 8F 65 0A B1 70 F0 8F 1F
    59 31 2E F8 13 1C 70 18 BD 58 21 40
    F9 BD AC 4E 31 1F 06 E1 09 CE 78 66
    BE FF 15 15 CB E2 6A AF CA 2D 66 5B
    90 00
Status: No Error
cm>
```

```
nJCOP: (01) generic-m00-tools  (built:
```

Figure 5.2 Java card simulation using JCOP tools in Eclipse

5.4 Simulation Results

The average time taken for the operations were recorded as shown in Table 5.2

Operation	Time in mili-seconds
Connecting to card	45
Connecting to applet on card	20
Verifying the PIN	30
Communicating with the card without any processing	15
RSA Encryption with 1024 bit CRT formatted private key	230
RSA decryption	40
Signing text using RSA private key	250
Verifying text using RSA public key	50
Encryption/decryption with Triple DES symmetric keys	22
Hashing message using SHA	45
Updating Card balance	85

Table 5.2 Average operation times on card obtained from java card simulation

5.5 Inference

The time taken for a transaction using SSL protocol for operating times shown in Table 5.2 is computed below.

1. Card Hello: 45 + 20 = 60 milliseconds
2. Sending certificate: 15 milliseconds
3. Requesting certificate: 15 milliseconds
4. Pin Verification: 30 milliseconds
5. Decryption using Terminal's public key: 50 milliseconds
6. Signing with RSA private key: 250 milliseconds
7. Finish message using Session key: 22 milliseconds
8. Updating card balance: 85 milliseconds

The total time for the transaction = 60 + 15 + 15 + 30 + 50 + 250 + 22 + 85 = 527 milliseconds. This is greater than a desired transaction time of 400 milliseconds. Further there is delay in communicating data to a central server and checking against "hot-lists" which contain invalid or fraud card IDs. This increases the transaction times to .7 or .8 seconds. Further key encryption and decryption are random events, and hence the exact time of transaction may vary from .5 to 1 second.

Another group of researchers at NTT Information Platform laboratories, Japan developed a public-key based high speed payment processing system for contactless smart cards in 2001. Experimental results show that the use of elliptical curve digital signature algorithms can complete a payment cycle in less than 0.4 seconds. However elliptical digital signatures have weak security algorithms and are not suitable for high value transactions.

From the above findings, we see that it is difficult to achieve desirable transaction speeds with high security protocols like SSL. Hence the state of the art cards are incapable of high speed secure transactions for heavily used transportation systems.

Chapter 6

Conclusion and Future Work

Preliminary models developed show that these technologies offer significant cost reductions. It is likely that by deploying some of these technologies, TfL can reduce the fare collection expenditure from 8% of total revenues collected to approximately 5%. The overall performance of these fare collection systems will also depend on transaction times and card security. The amount of fare evasion can be significantly reduced if high security features are installed on the card. However having very high security protocols may negatively impact processing times at barriers, leading to queue lengths at barriers at rush hours. Current smart cards need to increase memory availability and communication speeds to meet the high performance requirements of public transits.

Several areas still need to be explored before finalizing the architecture of the ticketing system. Issues like overall system security, privacy, trip history requirements must be evaluated in a broader sense. Smart cards need be physically tested to understand their performance. It is also very important to understand the effect of different fare structures on revenues. The price sensitivities will determine the loss in revenues by moving to flat fare systems or increase in revenues by introducing congestion pricing. The choice of technology will greatly depend on the marginal cost of implementing more complex fare structures. Secondly, specific costs of required hardware and software should be investigated and the cost assumptions used in the model must be revised accordingly.

Forecasting market penetration of cell phones and bank cards is necessary to understand effectiveness of advanced ticketing systems. We also need to assess the issue of change management since systems are not expected to change overnight.

The current work develops a framework to understand the future requirements of ticketing systems for London public transportation. The primary goal has been to analyze how emerging technologies can improve fare collection in public transit systems. Emerging technologies like contactless smart cards, WiFi, NFC in mobile phones offer great potential to reduce ticketing system costs and provide creative pricing structures in public transportation. At this stage of the study, most new technologies appear competitive and it will be interesting if London implements an advanced far collection system before the 2012 Olympics.

Bibliography

1. Smart Card Alliance Report. 2003. Transit and Retail Payment: Opportunities for Collaboration and Convergence.

2. P.T. Blythe. 2004. Improving Public transportation through smart cards: Proceedings of the Institution of Civil Engineers Municipal Engineer 157 March 2004 Issue ME1 Pages 47–54

3. V. Plotnikov. 2001 An Analysis of Fare Collection Costs on Heavy Rail and Bus Systems in the U.S. Virginia Institute of Technology

4. Ioannis Papaefstathiou. 2004. Evaluation of Micropayment Transaction Costs, Journal of Electronic Commerce Research, VOL. 5, NO.2, 2004

5. Innovision Research & Technology : Smart ticketing for mass transit - The new global opportunity created by low-cost, contactless ticketing, May 2005

6. Jorge M. Rebelo. 1999 Automated Ticketing Systems: The State of the Art. LCSFP. The World Bank

7. IBM : JCOP30 Technical Brief., ww.zurich.ibm.com/jcop/download/tools/JCOP30Brief.pdf

8. Karl Scheibelhofer. 2000.Using OpenCard in Combination with the Java Cryptography Architecture for Digital Signing

9. Hideo Yamamoto1, Tetsutaro Kobayashi1, Masahiro Morita2 and Ryuji Yamada. 2001. Public-Key-Based High-Speed Payment (Electronic Money) System Using Contact-Less Smart Cards: Attali and T. Jensen (Eds.): E-smart 2001, LNCS 2140, pp. 242-254, 2001

10. David M'Raihi and Moti Yung. 2001. "E-Commerce Applications of Smart Cards" Computer Networks 36(4):453{472, 2001

11. Jonathon Collins. 2004. New Two Frequency RFID System. http://www.emmicroelectronic.com/webfiles/news/RJdualfreq010904.htm

12. Accenture: 2005. Ticket to The Future: Smart Card Technology in Public Transportation

Appendix 1 - Code for the Feasibility and Cost Model

Class Input_GUI:

```
import java.awt.BorderLayout;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;

public class Input_GUI {

	public static void main(String[] args) {


		JFrame top = new JFrame("Model Input");
		top.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		InputPanel input =  new InputPanel();
		top.getContentPane().add(input, BorderLayout.CENTER);
		top.pack();
		top.setVisible(true);
		//top.setSize(600,600);
	}
}
```

Class InputPanel:

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

public class InputPanel extends JPanel{

	static final int flatFare = 0;
	static final int zonal =1;
	static final int con_fixed =2;
	static final int con_real =3;
	static final int price_cap =4;
	static final int unlimited =5;
	static final int special =6;
	static final int transfer =7;
	static final int loyalty =8;
	static final int timeOfDay =9;

	static final int Oyster = 0;
	static final int memoryless = 1;
	static final int bank= 2;
	static final int cell=3;
	static final int LongDistance=4;

	static final int TfL_stored=0;
	static final int TfL_trans=1;
	static final int bank_stored=2;
	static final int bank_trans=3;
	static final int cell_stored=4;
	static final int cell_trans=5;



	JLabel welcome  =  new JLabel("Check one each of fare, media and processor choices
first", SwingConstants.CENTER);
	FareOptions fare = new FareOptions();
	Media_input media = new Media_input();
	Processor_Input proc = new Processor_Input();

	JCheckBox Fare[];
	JLabel FareOpts ;

	JCheckBox Media[];
	JLabel MediaOpts;

	JCheckBox Proc[];
	JLabel  processorInputs;

	JButton NextStage;

	public InputPanel() {

		NextStage =  new JButton("Proceed to next stage");

		NextStage.addActionListener( new ActionListener(){
		public void actionPerformed(ActionEvent e){

		boolean fareFilled =false;
		boolean mediaFilled=false;
		boolean procFilled=false;

		for (int i = 0; i < 10; i++){
		if (Fare[i].isSelected())
		{User_choices.FareChoices[i] = true;
		if (i<4)fareFilled = true;
				}}

		for (int i = 0; i < 5; i++){
		if (i != memoryless && Media[i].isSelected())
		{User_choices.MediaChoices[i] = true;
		mediaFilled = true;
}      }
for (int i = 0; i < 6; i++){
if (Proc[i].isSelected())
{User_choices.ProcessChoices[i] = true;
procFilled = true;
}}

if (fareFilled && mediaFilled && procFilled)
Stage2filter.displayStage2();
else if (!fareFilled) {
JFrame frame = new JFrame("Incompatible conditions");
JOptionPane.showMessageDialog(frame,"Choose at least one pricing
structure - flat/zonal/congestion");
      }
else if (!mediaFilled) {
      JFrame frame = new JFrame("Incompatible conditions");
      JOptionPane.showMessageDialog(frame,"media options incomplete");
                        }
else if (!procFilled) {
JFrame frame = new JFrame("Incompatible conditions");
JOptionPane.showMessageDialog(frame,"Choose one processing option");
      }

}
      });

//Assigning compatibility
for (int i= 0; i<10; i++){
      check.fare_media[i][LongDistance] = true;
      check.fare_media[i][Oyster] =true;
      check.fare_media[i][memoryless] = true;
}
check.fare_media[con_fixed][Oyster]= false;
check.fare_media[con_real][Oyster]= false;
check.fare_media[transfer][Oyster]= false;
check.fare_media[con_real][memoryless] = false;

check.fare_media[flatFare][bank] = true;
check.fare_media[price_cap][bank]= true;
check.fare_media[loyalty][bank] =true;
check.fare_media[flatFare][cell] = true;
check.fare_media[price_cap][cell]= true;
check.fare_media[loyalty][cell] =true;


for (int j = 0; j<10; j++){
      check.fare_processor[j][TfL_stored] = true;
      check.fare_processor[j][TfL_trans] =true;
      }

      check.fare_processor[con_real][TfL_stored]= false;

for (int k = 2; k<6; k++){
      check.fare_processor[flatFare][k]= true;
      check.fare_processor[special][k] = true;
}
```

```
        check.fare_processor[special][bank_stored] = false;
        check.fare_processor[special][cell_stored] = false;
        check.fare_processor[zonal][cell_trans]= true;
        check.fare_processor[price_cap][cell_trans]= true;
        check.fare_processor[price_cap][bank_trans]= true;
        check.fare_processor[loyalty][cell_trans]= true;
        check.fare_processor[loyalty][bank_trans]= true;

        check.media_processor[Oyster][TfL_stored]=
        check.media_processor[Oyster][TfL_trans]=
        check.media_processor[Oyster][bank_trans]=
        check.media_processor[Oyster][cell_trans]= true;
        check.media_processor[memoryless][TfL_trans]=
        check.media_processor[memoryless][bank_trans] = true;
        check.media_processor[LongDistance][TfL_trans]= true;

        check.media_processor[bank][TfL_trans]=
        check.media_processor[bank][bank_trans]=
        check.media_processor[bank][bank_stored] =
        check.media_processor[bank][cell_trans] = true;

        check.media_processor[cell][TfL_trans]=
        check.media_processor[cell][cell_stored]=
        check.media_processor[cell][cell_trans] =
        check.media_processor[cell][bank_trans] = true;

    setLayout(new BorderLayout());
    add(welcome, BorderLayout.NORTH);
    add(fare, BorderLayout.WEST);
    add(media,BorderLayout.CENTER);
    add(proc, BorderLayout.EAST);
    add(NextStage, BorderLayout.SOUTH);

    }

    public class FareOptions extends JPanel{


        public FareOptions(){

        FareOpts = new JLabel("Fare options");
        Fare = new JCheckBox[10];
        Fare[flatFare] = new JCheckBox("Flat fare");
        Fare[zonal] = new JCheckBox("Zonal/Distance Fare");
        Fare[con_fixed] = new JCheckBox("Congestion pricing - fixed");
        Fare[con_real] = new JCheckBox("Congestion pricing - real time");
        Fare[price_cap] = new JCheckBox("Price cap");
        Fare[unlimited] = new JCheckBox("Unlimited pass");
        Fare[special]  = new JCheckBox("Special Groups");
        Fare[transfer] = new JCheckBox("Transfer Discounts");
        Fare[loyalty] = new JCheckBox("Loyalty Benifits");
        Fare[timeOfDay] = new JCheckBox("Time of day pricing");


        add(FareOpts);
        for (int i=0; i<10; i++)
        {add(Fare[i]);}
```

```
setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
setPreferredSize(new Dimension(200,400));


//flatFare.setEnabled(false);

Fare[flatFare].addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent e){
        if (Fare[flatFare].isSelected()){
        InputMatrix.fare[0]= true;
            for (int i=0; i<5;i++){
                    if(!check.fare_media[0][i]) {
                        Media[i].setEnabled(false);
                    }
            }
            for (int i=0; i<6;i++){
                    if(!check.fare_processor[0][i]) {
                        Proc[i].setEnabled(false);

                }
            }
        }
        else if (!Fare[flatFare].isSelected()){
            InputMatrix.fare[0]= false;
            reset();
        }
        }
    });


Fare[zonal].addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent e){
        if (Fare[zonal].isSelected()){
        InputMatrix.fare[1]= true;
        for (int i=0; i<5;i++){
            if(!check.fare_media[1][i]) {
                Media[i].setEnabled(false);
            }
    }
    for (int i=0; i<6;i++){
            if(!check.fare_processor[1][i]) {
                Proc[i].setEnabled(false);

            }
        }
        }
        else if (!Fare[zonal].isSelected()){
            InputMatrix.fare[1]= false;
            reset();
        }
        }
    });


Fare[con_fixed].addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent e){
          if (Fare[con_fixed].isSelected()){
          InputMatrix.fare[2]= true;
          for (int i=0; i<5;i++){
                if(!check.fare_media[2][i]) {
                      Media[i].setEnabled(false);
                }
    }
    for (int i=0; i<6;i++){
                if(!check.fare_processor[2][i]) {
                      Proc[i].setEnabled(false);

          }
    }
}
          else if (!Fare[con_fixed].isSelected()){
                InputMatrix.fare[2]= false;
                reset();
          }
          }
    });


Fare[con_real].addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent e){
          if (Fare[con_real].isSelected()){
          InputMatrix.fare[3]= true;
          for (int i=0; i<5;i++){
                if(!check.fare_media[3][i]) {
                      Media[i].setEnabled(false);
                            }
                }
                for (int i=0; i<6;i++){
                            if(!check.fare_processor[3][i]) {
                                  Proc[i].setEnabled(false);

                      }
                }
          }
          else if (!Fare[con_real].isSelected()){
                InputMatrix.fare[3]= false;
                reset();
                }
          }
    });


Fare[price_cap].addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent e){
          if (Fare[price_cap].isSelected()){
          InputMatrix.fare[4]= true;
          for (int i=0; i<5;i++){
                if(!check.fare_media[4][i]) {
                      Media[i].setEnabled(false);
                }
    }
    for (int i=0; i<6;i++){
                if(!check.fare_processor[4][i]) {
```

```
                              Proc[i].setEnabled(false);

                  }
            }
      }
                  else if (!Fare[price_cap].isSelected()){
                        InputMatrix.fare[4]= false;
                        reset();
                  }
                  }
            });



      Fare[unlimited].addActionListener( new ActionListener(){
      public void actionPerformed(ActionEvent e){
            if (Fare[unlimited].isSelected()){
            InputMatrix.fare[5]= true;
            for (int i=0; i<5;i++){
                  if(!check.fare_media[5][i]) {
                        Media[i].setEnabled(false);
                  }
      }
      for (int i=0; i<6;i++){
                  if(!check.fare_processor[5][i]) {
                        Proc[i].setEnabled(false);

            }
      }
}                 else if (!Fare[unlimited].isSelected()){
                  InputMatrix.fare[5]= false;
                  reset();
            }
            }
      });


      Fare[special].addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent e){
                  if (Fare[special].isSelected()){
                  InputMatrix.fare[6]= true;
                  for (int i=0; i<5;i++){
                        if(!check.fare_media[6][i]) {
                              Media[i].setEnabled(false);
                        }
            }
            for (int i=0; i<6;i++){
                        if(!check.fare_processor[6][i]) {
                              Proc[i].setEnabled(false);

                  }
            }
      }                       else if (!Fare[special].isSelected()){
                        InputMatrix.fare[6]= false;
                        reset();
                  }
                  }
```

```
        });


    Fare[transfer].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (Fare[transfer].isSelected()){
            InputMatrix.fare[7]= true;
            for (int i=0; i<5;i++){
                if(!check.fare_media[7][i]) {
                    Media[i].setEnabled(false);
                }
        }
        for (int i=0; i<6;i++){
                if(!check.fare_processor[7][i]) {
                    Proc[i].setEnabled(false);

            }
        }
    }                   else if (!Fare[transfer].isSelected()){
                InputMatrix.fare[7]= false;
                reset();
            }
            }
        });



    Fare[loyalty].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (Fare[loyalty].isSelected()){
            InputMatrix.fare[8]= true;
            for (int i=0; i<5;i++){
                if(!check.fare_media[8][i]) {
                    Media[i].setEnabled(false);
                }
        }
        for (int i=0; i<6;i++){
                if(!check.fare_processor[8][i]) {
                    Proc[i].setEnabled(false);

            }
        }
    }
            else if (!Fare[loyalty].isSelected()){
                InputMatrix.fare[8]= false;
                reset();
            }
            }
        });

    Fare[timeOfDay].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (Fare[timeOfDay].isSelected()){
            InputMatrix.fare[9]= true;
            for (int i=0; i<5;i++){
                if(!check.fare_media[9][i]) {
                    Media[i].setEnabled(false);
                }
        }
        for (int i=0; i<6;i++){
                if(!check.fare_processor[9][i]) {
                    Proc[i].setEnabled(false);

            }
        }
    }       else if (!Fare[timeOfDay].isSelected()){
                InputMatrix.fare[9]= false;
                reset();
            }
            }
        });


    }
}


public class Media_input extends JPanel {

    public Media_input(){
        MediaOpts = new JLabel("Media options");
        Media = new JCheckBox[5];
        Media[Oyster] = new JCheckBox("Oyster (extention)");
        Media[memoryless] = new JCheckBox("Memoryless cards");
        Media[bank] = new JCheckBox("Bank Cards");
        Media[cell] = new JCheckBox("Cell Phones");
        Media[LongDistance] = new JCheckBox("Long Distance cards");


    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    add(MediaOpts);
    for (int i=0; i<5; i++)
        {add(Media[i]);}

    Media[Oyster].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (Media[Oyster].isSelected()){
            InputMatrix.media[0]= true;
            for (int i=0; i<10;i++){
                if(!check.fare_media[i][0]) {
                    Fare[i].setEnabled(false);
                    }
                }
            for (int i=0; i<6;i++){
                if(!check.media_processor[0][i]) {
                    Proc[i].setEnabled(false);
                    }
                }
            Media[LongDistance].setEnabled(false);
            }
            else if (!Media[Oyster].isSelected()){
            InputMatrix.media[0]= false;
            reset();
            }
```

```
                }
        });


Media[memoryless].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Media[memoryless].isSelected()){
                InputMatrix.media[1]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_media[i][1]) {
                                Fare[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<6;i++){
                        if(!check.media_processor[1][i]) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                else if (!Media[memoryless].isSelected()){
                        InputMatrix.media[1]= false;
                reset();
                }
                }
        });


Media[bank].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Media[bank].isSelected()){
                InputMatrix.media[2]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_media[i][2]) {
                                Fare[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<6;i++){
                        if(!check.media_processor[2][i]) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                else if (!Media[bank].isSelected()){
                        InputMatrix.media[2]= false;
                        reset();
                }
                }
        });


Media[cell].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Media[cell].isSelected()){
                InputMatrix.media[3]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_media[i][3]) {
                                Fare[i].setEnabled(false);
```

```
                    }
                }
            for (int i=0; i<6;i++){
                if(!check.media_processor[3][i]) {
                    Proc[i].setEnabled(false);
                    }
                }
            }
            else if (!Media[cell].isSelected()){
                InputMatrix.media[3]= false;
            }
            }
        });


    Media[LongDistance].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (Media[LongDistance].isSelected()){
            InputMatrix.media[4]= true;
            for (int i=0; i<10;i++){
                if(!check.fare_media[i][4]) {
                    Fare[i].setEnabled(false);
                    }
                }
            for (int i=0; i<6;i++){
                if(!check.media_processor[4][i]) {
                    Proc[i].setEnabled(false);
                    }
                }
            for (int i=0; i<5;i++){
                if(i != memoryless && i!= LongDistance) {
                    Media[i].setEnabled(false);
                    }
                }
            }
            else if (!Media[LongDistance].isSelected()){
                InputMatrix.media[4]= false;
                reset();
            }
            }
        });



    }


}


public class Processor_Input extends JPanel {


    public Processor_Input(){

        Proc = new JCheckBox[6];
        processorInputs = new JLabel("processing options");
```

```
            Proc[TfL_stored] = new JCheckBox("TfL -Stored ");
            Proc[TfL_trans] = new JCheckBox("TfL - Transaction");
            Proc[bank_stored] = new JCheckBox("Bank - Stored value");
            Proc[bank_trans] = new JCheckBox("Bank - Transaction");
            Proc[cell_stored] = new JCheckBox("Cell - Stored value");
            Proc[cell_trans] = new JCheckBox("Cell - Transaction");

      setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
      setPreferredSize(new Dimension(200,200));

      add(processorInputs);
      for (int i=0; i<6; i++)
            {add(Proc[i]);}

      Proc[TfL_stored].addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent e){
                  if (Proc[TfL_stored].isSelected()){
                  InputMatrix.proc[0]= true;
                  for (int i=0; i<10;i++){
                        if(!check.fare_processor[i][0]) {
                              Fare[i].setEnabled(false);
                              }
                        }
                  for (int i=0; i<5;i++){
                        if(!check.media_processor[i][0]) {
                              Media[i].setEnabled(false);
                              }
                        }
                  for (int i=0; i<6;i++){
                        if(i != TfL_stored) {
                              Proc[i].setEnabled(false);
                              }
                        }
                  }
                  else if (!Proc[TfL_stored].isSelected()){
                        InputMatrix.proc[0]= false;
                  reset();
                  }
                  }
            });



      Proc[TfL_trans].addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent e){
                  if (Proc[TfL_trans].isSelected()){
                  InputMatrix.proc[1]= true;
                  for (int i=0; i<10;i++){
                        if(!check.fare_processor[i][1]) {
                              Fare[i].setEnabled(false);
                              }
                        }
                  for (int i=0; i<5;i++){
                        if(!check.media_processor[i][1]) {
                              Media[i].setEnabled(false);
                              }
                        }
                for (int i=0; i<6;i++){
                        if(i != TfL_trans) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                else if (!Proc[TfL_trans].isSelected()){
                        InputMatrix.proc[1]= false;
                        reset();
                        }
                }
        });


Proc[bank_stored].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Proc[bank_stored].isSelected()){
                InputMatrix.proc[2]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_processor[i][2]) {
                                Fare[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<5;i++){
                        if(!check.media_processor[i][2]) {
                                Media[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<6;i++){
                        if(i != bank_stored) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                                else if (!Proc[bank_stored].isSelected()){
                        InputMatrix.proc[2]= false;
                        reset();
                        }
                }
        });



Proc[bank_trans].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Proc[bank_trans].isSelected()){
                InputMatrix.proc[3]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_processor[i][3]) {
                                Fare[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<5;i++){
                        if(!check.media_processor[i][3]) {
                                Media[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<6;i++){
                        if(i != bank_trans) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                else if (!Proc[bank_trans].isSelected()){
                        InputMatrix.proc[3]= false;
                        reset();

                }
                }
        });



Proc[cell_stored].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Proc[cell_stored].isSelected()){
                InputMatrix.proc[4]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_processor[i][4]) {
                                Fare[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<5;i++){
                        if(!check.media_processor[i][4]) {
                                Media[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<6;i++){
                        if(i != cell_stored) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                else if (!Proc[cell_stored].isSelected()){
                        InputMatrix.proc[4]= false;
                        reset();
                                }
                }
        });



Proc[cell_trans].addActionListener( new ActionListener(){
        public void actionPerformed(ActionEvent e){
                if (Proc[cell_trans].isSelected()){
                InputMatrix.proc[5]= true;
                for (int i=0; i<10;i++){
                        if(!check.fare_processor[i][5]) {
                                Fare[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<5;i++){
                        if(!check.media_processor[i][5]) {
                                Media[i].setEnabled(false);
                                }
                        }
                for (int i=0; i<6;i++){
                        if(i != cell_trans) {
                                Proc[i].setEnabled(false);
                                }
                        }
                }
                else if (!Proc[cell_trans].isSelected()){
                        InputMatrix.proc[5]= false;
                                reset();
                }
                }
        });



    }
}

public void reset(){
for (int i=0; i<10;i++){
    Fare[i].setEnabled(true);
}
for (int i=0; i<5;i++){
    Media[i].setEnabled(true);
}
for (int i=0; i<6;i++){
    Proc[i].setEnabled(true);
}


for (int i=0; i<10;i++){
    if (InputMatrix.fare[i]==true) {
            for (int j= 0;j<6;j++){
                    if(!check.fare_processor[i][j]) {
                            Proc[j].setEnabled(false);
                    }
            }
            for (int j= 0;j<5;j++){
                    if(!check.fare_media[i][j]) {
                            Media[j].setEnabled(false);
                    }
            }

    }
}


for (int i=0; i<5;i++){
    if (InputMatrix.media[i]==true) {
            for (int j= 0;j<10;j++){
                    if(!check.fare_media[j][i]) {
                            Fare[j].setEnabled(false);
                    }
            }
```

```
                        for (int j= 0;j<6;j++){
                                if(!check.media_processor[i][j]) {
                                     Proc[j].setEnabled(false);
                                }
                        }
                }
        }
        for (int i=0; i<6;i++){
                if (InputMatrix.proc[i]==true) {
                        for (int j= 0;j<5;j++){
                                if(!check.media_processor[j][i]) {
                                     Media[j].setEnabled(false);
                                }
                        }
                        for (int k= 0;k<10;k++){
                                if(!check.fare_processor[k][i]) {
                                     Fare[k].setEnabled(false);
                                }
                        }
                }
        }


        }



}
```

Class Check:

```
public class check {

        static boolean[][] fare_media = new boolean[10][5];
        static boolean[][] fare_processor = new boolean[10][6];
        static boolean[][] media_processor = new boolean[5][6];


}
```

Class Stage2Filter

```
import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.JFrame;

public class Stage2filter {

        public static void displayStage2(){

                JFrame top2 = new JFrame("Additional Requirements");
                top2.setPreferredSize(new Dimension(600,400));
                top2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                Frame2 frame2 =  new Frame2();
```

```
		top2.getContentPane().add(frame2, BorderLayout.CENTER);
		top2.pack();
		top2.setVisible(true);



	}


	public static void displayStage3(){

		JFrame top2 = new JFrame("System and Market Parameters");
		top2.setPreferredSize(new Dimension(600,600));
		top2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		SystemSizeInfo frame3 =  new SystemSizeInfo();
		top2.getContentPane().add(frame3, BorderLayout.CENTER);
		top2.pack();
		top2.setVisible(true);

	}
}
```

Class Frame2:

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Frame2 extends JPanel {

	JLabel triphist;
	JLabel iden;
	JLabel barr;
	JCheckBox TripHistory[];
	JCheckBox Identity[];
	JCheckBox Barrier ;

	JLabel addOpts;
	JButton NextStage;

	TripHis triph = new TripHis();
	Id idReq = new Id ();
	Barriers  barrierReq = new Barriers();

	public Frame2() {
		addOpts = new JLabel("");

		NextStage =  new JButton("Proceed to next stage");
```

```
    NextStage.addActionListener( new ActionListener(){
          public void actionPerformed(ActionEvent e){
                for (int i = 0; i < 3; i++){
                if (TripHistory[i].isSelected())
                User_choices.TripHistoryReq[i] = true;
                }
                for (int i = 0; i < 3; i++){
                       if (Identity[i].isSelected())
                      User_choices.Identity[i] = true;
                      }
                if (Barrier.isSelected())
                      User_choices.BarrierReq = true;

                Stage2filter.displayStage3();
          }
          });


    setLayout(new BorderLayout());
    add(addOpts, BorderLayout.NORTH);
    add(triph, BorderLayout.WEST);
    add(idReq,BorderLayout.CENTER);
    add(barrierReq, BorderLayout.EAST);
    add(NextStage, BorderLayout.SOUTH);

}

public class TripHis extends JPanel{


    public TripHis(){

    triphist = new JLabel("Trip History requirements");
    TripHistory = new JCheckBox[3];
    TripHistory[0] = new JCheckBox("This trip only");
    TripHistory[1] = new JCheckBox("Trips over period");
    TripHistory[2] = new JCheckBox("Long term trip history");

    if (User_choices.FareChoices[8]){
          TripHistory[1].setSelected(true);
          TripHistory[1].setEnabled(false);

    }

    add(triphist);
    for (int i=0; i<3; i++)
    {add(TripHistory[i]);}
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    setPreferredSize(new Dimension(200,400));
    }

}


public class Id extends JPanel{
        public Id(){

        iden = new JLabel("Identity Requirements");
        Identity = new JCheckBox[4] ;
        Identity[0] = new JCheckBox("Group");
        Identity[1] = new JCheckBox("Name and ID");
        Identity[2] = new JCheckBox("postal code");

        if (User_choices.FareChoices[8]){
              Identity[1].setSelected(true);
              Identity[1].setEnabled(false);
        }
        if (User_choices.FareChoices[6]){
              Identity[0].setSelected(true);
              Identity[0].setEnabled(false);

        }
        add(iden);
        for (int i=0; i<3; i++)
        {add(Identity[i]);}

        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        setPreferredSize(new Dimension(200,400));
        }

    }


    public class Barriers extends JPanel{


        public Barriers(){

        setPreferredSize(new Dimension(200,400));

        barr = new JLabel("Barrier Requirements");
        Barrier = new JCheckBox("Should barriers be eliminated", false);
        if (!User_choices.MediaChoices[4]){
              User_choices.BarrierReq = false;
              Barrier.setEnabled(false);}
        add(barr);
        add(Barrier);}


    }
}
```

Class User_choice

```
import javax.swing.JTextField;

public class User_choices {

	public static boolean[] FareChoices = new boolean[10];
	public static boolean[] MediaChoices= new boolean[5];
	public static boolean[] ProcessChoices= new boolean[6];
	public static boolean[] TripHistoryReq= new boolean[3];
	public static boolean BarrierReq;
	public static boolean[] Identity= new boolean[3];
	public static int BankShare = 0;
	public static int MobileShare = 0;
	public static int TFLShare = 100;

	public static int BusTrips;
	public static int TrainTrips;
	public static int BusTripLen;
	public static int TrainTripLen;

	public static int Buses;
	public static int Trains;
	public static int TStations;
	public static int BusStations;
	public static int Barriers;

}
```

Class SystemSizeParameters

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class SystemSizeInfo extends JPanel{

		JLabel sys;
		JLabel tripInfo;

		JLabel shares;
		JLabel systemSize;
		JLabel bankShare;
		JLabel mobileShare;
		JLabel TfLShare;
JLabel busTrips;
JLabel trainTrips;
JLabel busTripLen;
JLabel trainTripLen;

JLabel nBuses;
JLabel nTrains ;
JLabel nTStations;
JLabel nBusStations;
JLabel nBarriers;

JTextField BankShare;
JTextField MobileShare;
JTextField TFLShare;

JTextField BusTrips;
JTextField TrainTrips;
JTextField BusTripLen;
JTextField TrainTripLen;

JTextField Buses;
JTextField Trains;
JTextField TStations;
JTextField BusStations;
JTextField Barriers;


JButton EstimateCosts;

JPanel marketShareValues;
JPanel SystemSizeValues;
JPanel tripValues;

public SystemSizeInfo() {

      sys = new JLabel ("");

      marketShareValues = new JPanel();
      marketShareValues.setPreferredSize(new Dimension(250,200));

      marketShareValues.setLayout(new BoxLayout(marketShareValues,
      BoxLayout.Y_AXIS));

      shares = new JLabel("Percent trips by Ticket Media");

      JPanel bank = new JPanel();
      bank.setLayout(new FlowLayout());
      bankShare = new JLabel("Bank Cards");
      bankShare.setPreferredSize(new Dimension(150,20));

      BankShare = new JTextField();
      BankShare.setPreferredSize(new Dimension(80,20));
      if (!User_choices.MediaChoices[2]){
            BankShare.setText("0");
            BankShare.setEditable(false);
      }
```

```
                bank.add(bankShare);
                bank.add(BankShare);

                JPanel cell = new JPanel();
                cell.setLayout(new FlowLayout());
                mobileShare = new JLabel("Mobile Phones");
                mobileShare.setPreferredSize(new Dimension(150,20));
                MobileShare = new JTextField();
                MobileShare.setPreferredSize(new Dimension(80,20));
                if (!User_choices.MediaChoices[3]){
                      MobileShare.setText("0");
                      MobileShare.setEditable(false);
                }

                cell.add(mobileShare);
                cell.add(MobileShare);

                JPanel tfl = new JPanel();
                tfl.setLayout(new FlowLayout());
                TfLShare = new JLabel("TfL Cards");
                TfLShare.setPreferredSize(new Dimension(150,20));
                TFLShare = new JTextField();
                TFLShare.setPreferredSize(new Dimension(80,20));
                if
(!User_choices.MediaChoices[0]&&!User_choices.MediaChoices[1]&&!User_choices.MediaChoice
s[4]){
                      TFLShare.setText("0");
                      TFLShare.setEditable(false);
                }
                else if
(!User_choices.MediaChoices[2]&&!User_choices.MediaChoices[3]){
                      //int val = 100 -
Integer.valueOf(BankShare.getText()).intValue() -
Integer.valueOf(MobileShare.getText()).intValue();
                      TFLShare.setText(""+100);
                      TFLShare.setEditable(false);
                      }

                tfl.add(TfLShare);
                tfl.add(TFLShare);

                tripInfo = new JLabel("Information about trips");

                JPanel bT = new JPanel();
                bT.setLayout(new FlowLayout());
                busTrips = new JLabel("daily bus trips");
                busTrips.setPreferredSize(new Dimension(150,20));
                BusTrips = new JTextField("5000000");
                BusTrips.setPreferredSize(new Dimension(80,20));
                bT.add(busTrips);
                bT.add(BusTrips);

                JPanel tT = new JPanel();
                tT.setLayout(new FlowLayout());
                trainTrips = new JLabel("daily train trips");
                trainTrips.setPreferredSize(new Dimension(150,20));
                TrainTrips = new JTextField("1000000");
```

```
TrainTrips.setPreferredSize(new Dimension(80,20));
tT.add(trainTrips);
tT.add(TrainTrips);

JPanel btL = new JPanel();
btL.setLayout(new FlowLayout());
busTripLen = new JLabel("Avg Bus trip length (km)");
busTripLen.setPreferredSize(new Dimension(150,20));
BusTripLen = new JTextField("3");
BusTripLen.setPreferredSize(new Dimension(80,20));
btL.add(busTripLen);
btL.add(BusTripLen);

JPanel ttl = new JPanel();
ttl.setLayout(new FlowLayout());
trainTripLen = new JLabel("Avg Train TripLength (km)");
trainTripLen.setPreferredSize(new Dimension(150,20));
TrainTripLen = new JTextField("3");
TrainTripLen.setPreferredSize(new Dimension(80,20));
ttl.add(trainTripLen);
ttl.add(TrainTripLen);

marketShareValues.add(shares);
marketShareValues.add(bank);
marketShareValues.add(cell);
marketShareValues.add(tfl);

marketShareValues.add(tripInfo);
marketShareValues.add(bT);
marketShareValues.add(tT);
marketShareValues.add(btL);
marketShareValues.add(ttl);


systemSize = new JLabel("System Size parameters");

SystemSizeValues = new JPanel();
SystemSizeValues.setPreferredSize(new Dimension(250,200));

SystemSizeValues.setLayout(new BoxLayout(SystemSizeValues,
BoxLayout.Y_AXIS));

JPanel TS = new JPanel();
TS.setLayout(new FlowLayout());
nTStations = new JLabel("Number of Rail stations");
nTStations.setPreferredSize(new Dimension(150,20));
TStations = new JTextField("300");
TStations.setPreferredSize(new Dimension(80,20));
TS.add(nTStations);
TS.add(TStations);

JPanel BS = new JPanel();
BS.setLayout(new FlowLayout());
nBusStations = new JLabel("Number of Bus stations");
nBusStations.setPreferredSize(new Dimension(150,20));
BusStations = new JTextField("17000");
BusStations.setPreferredSize(new Dimension(80,20));
```

```
                    BS.add(nBusStations);
                    BS.add(BusStations);

                    JPanel NB = new JPanel();
                    NB.setLayout(new FlowLayout());
                    nBuses = new JLabel("Number of Buses");
                    nBuses.setPreferredSize(new Dimension(150,20));
                    Buses = new JTextField("1000");
                    Buses.setPreferredSize(new Dimension(80,20));
                    NB.add(nBuses);
                    NB.add(Buses);

                    JPanel NT = new JPanel();
                    NT.setLayout(new FlowLayout());
                    nTrains = new JLabel("Number of Trains");
                    nTrains.setPreferredSize(new Dimension(150,20));
                    Trains = new JTextField("1000");
                    Trains.setPreferredSize(new Dimension(80,20));
                    NT.add(nTrains);
                    NT.add(Trains);

                    JPanel B = new JPanel();
                    B.setLayout(new FlowLayout());
                    nBarriers = new JLabel("Number of barriers");
                    nBarriers.setPreferredSize(new Dimension(150,20));
                    Barriers = new JTextField("3000");
                    Barriers.setPreferredSize(new Dimension(80,20));
                    B.add(nBarriers);
                    B.add(Barriers);

                    SystemSizeValues.add(systemSize);
                    SystemSizeValues.add(TS);
                    SystemSizeValues.add(BS);
                    SystemSizeValues.add(NB);
                    SystemSizeValues.add(NT);
                    SystemSizeValues.add(B);

EstimateCosts =  new JButton("Estimate System costs");

EstimateCosts.addActionListener( new ActionListener(){
public void actionPerformed(ActionEvent e){
if
(Integer.valueOf(BankShare.getText()).intValue()+Integer.valueOf(MobileShare.getText()).
intValue()+Integer.valueOf(TFLShare.getText()).intValue() !=100 )           JFrame
frame = new JFrame("Input Error")
JOptionPane.showMessageDialog(frame,"Sum of market shares by media should be 100");
                         }
try{
User_choices.TrainTrips = Integer.parseInt(TrainTrips.getText());
User_choices.BusTrips = Integer.parseInt(BusTrips.getText());
User_choices.TrainTripLen = Integer.parseInt(TrainTripLen.getText());
User_choices.BusTripLen = Integer.parseInt(BusTripLen.getText());
User_choices.TStations = Integer.parseInt(TStations.getText());
User_choices.BusStations = Integer.parseInt(BusStations.getText());
User_choices.Barriers = Integer.parseInt(Barriers.getText());
User_choices.Trains = Integer.parseInt(Trains.getText());
User_choices.Buses = Integer.parseInt(Buses.getText());
User_choices.BankShare = Integer.parseInt(BankShare.getText());
User_choices.MobileShare = Integer.parseInt(MobileShare.getText());
User_choices.TFLShare = Integer.parseInt(TFLShare.getText());

                    double c = CostParameters.EstimateCosts();
                    JFrame f = new JFrame("System costs");
                    JOptionPane.showMessageDialog(f,"The approximate cost is " +c/1e6 + "
                    million dollars");
                          }
                          catch (NumberFormatException f){
                                JOptionPane.showMessageDialog(null, "not an integer");




                          }
                          }
                          });


                    setLayout(new BorderLayout());
                    add(sys, BorderLayout.NORTH);
                    add(marketShareValues, BorderLayout.WEST);
                    add(SystemSizeValues, BorderLayout.EAST);
                    add(EstimateCosts, BorderLayout.SOUTH);

              }

}
```

Class CostParamaters

```
public class CostParameters {

      static double BusCost;
      static double TrainCost;
      static double TransactionCost;
      static double TfLTransaction;
      static double bankTransaction;
      static double cellTransaction;
      static double finalCosts;

      final static int PDA = 500;
      final static int Reader = 250;
      final static int printer = 200;
      final static int GPSChip = 100;
      final static int WiMaxCard = 100;
      final static int BusPowerSupply = 200;
      final static double TransitDuty = 1.5;
      final static int UnitLife = 2;
      final static double Operating_bus = 4e6;
      final static double Software = 3e6;
```

```
static int Wireless;
static double BusUnitCosts;
final static int WirelessPerYear = 30*12;

final static int  readersPerBarrier = 2;
static double RailStationUnit = .6e6;
static double BarrierCosts;
final static double UnitBarrierCost =  2000;
static double ReaderCosts;
static int LDReaderCost = 2000;
final static int USB =0;
final static int climateControl =0;
final static int UPS =0;
final static int network =0;
final static int ServersPerStation =2;
final static double Operating_train = .7e6;
final static double CentralServer = .5e6;

static int  cards = 10000000;

static double unitCardCosts;
final static double oystercost = 1;       //dollars
final static double memorylesscost = .05;       //dollars
final static double LDSCcost = 2;   //dollars
static double BusReaderCosts;

static int CardLife = 1;
static double CardCosts ;

static double salesCommission;

final static int customerReps = 50 * 30000;
final static int machinesPerStation = 2;
final static int LifeOfMachine = 2;
final static int machineMaintainance =1;
final static int SalesOutlets =3000;
final static int CostofOutlet =1000;
final static double bankCommissionPerCent = 2.7;
final static double ThirdPartyCommPerCent = 7.5;
final static double cellCommissionPerCent = 3.0;
final static double salesCommissionPerCent = 1.0;

final static double AvgTripCost = 1.5; // dollars



public static double EstimateCosts(){

BusUnitCosts =    User_choices.Buses * (PDA +printer + GPSChip +WiMaxCard +
BusPowerSupply) * TransitDuty/UnitLife;
Wireless = User_choices.BusStations * WirelessPerYear;
if (User_choices.MediaChoices[4])
      BusReaderCosts = User_choices.Buses * LDReaderCost;
else BusReaderCosts = User_choices.Buses * UnitBarrierCost;

BusCost =   BusUnitCosts + Operating_bus + Software + Wireless + BusReaderCosts;
```

```
      RailStationUnit = USB  + climateControl + UPS + network + ServersPerStation;
      if (User_choices.BarrierReq)
            BarrierCosts = 0;
      else BarrierCosts = User_choices.Barriers * UnitBarrierCost;

      if (User_choices.MediaChoices[4])
            ReaderCosts = User_choices.TStations * LDReaderCost;
      else ReaderCosts = User_choices.Barriers * UnitBarrierCost;

      ReaderCosts = readersPerBarrier * User_choices.Barriers ;

      TrainCost = BarrierCosts + ReaderCosts + RailStationUnit + Operating_train +
      CentralServer;


      if (User_choices.MediaChoices[0]&&User_choices.MediaChoices[1])
            unitCardCosts = (oystercost + memorylesscost)/2;
      else if (User_choices.MediaChoices[1]&&User_choices.MediaChoices[4])
            unitCardCosts = (memorylesscost + LDSCcost)/2;
      else if (User_choices.MediaChoices[0])
            unitCardCosts = oystercost ;
      else if (User_choices.MediaChoices[4])
            unitCardCosts = LDSCcost ;
      else  unitCardCosts = 0.0 ;

      CardCosts = User_choices.TFLShare/100 * cards * unitCardCosts/ CardLife;

      salesCommission = salesCommissionPerCent /100 * AvgTripCost *
(User_choices.TrainTrips + User_choices.BusTrips)*300;

      TfLTransaction = 1.5 * (CardCosts + customerReps + machinesPerStation*
User_choices.TStations/LifeOfMachine + machineMaintainance + salesCommission);

      cellTransaction = cellCommissionPerCent /100 * AvgTripCost *
(User_choices.TrainTrips + User_choices.BusTrips)*300;

      bankTransaction = bankCommissionPerCent /100 * AvgTripCost *
(User_choices.TrainTrips + User_choices.BusTrips)*300;

      TransactionCost = User_choices.TFLShare*TfLTransaction/100
+User_choices.MobileShare*cellTransaction/100 +
User_choices.BankShare*bankTransaction/100;

      finalCosts = 1.5 * (BusCost + TrainCost) + TransactionCost ;//

      return finalCosts;

      }

}
```

Appendix 2 - Code for Java Card Applet

```
package com.key;
//import com.jcp.jcapplet;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.OwnerPIN;
import javacard.security.DESKey;
import javacard.security.KeyBuilder;
import javacard.security.MessageDigest;
import javacard.security.RSAPrivateCrtKey;
import javacard.security.RSAPrivateKey;
import javacard.security.RSAPublicKey;
import javacard.security.RandomData;
import javacardx.crypto.Cipher;

public class key extends Applet {
      final static byte MY_PROJECT_CLA = (byte)0x90;
      final byte PIN_CHECK = (byte)0x10;
      final byte RSA = (byte)0x20;
      final byte DES3 = (byte)0x30;
      final byte DES = (byte)0x40;
      final byte SHA = (byte)0x50;
      final byte RSA_D = (byte)0x60;
      final byte DES3_D = (byte)0x70;
      final byte DES_D = (byte)0x80;
      final byte SHA_V = (byte)0x90;
      final byte SIGN_TEXT = (byte)0xA0;
      final byte VERIFY_TEXT = (byte)0xB0;
      final byte DUMMY = (byte)0xD0;
      final byte TOKEN = (byte)0xE0;
      final byte PIN_TRY_LIMIT = (byte)0x03;
      final byte MAX_PIN_SIZE = (byte)0x04;
      final short WRONG_PIN = (short)0x6BBB;
      final short WRONG_SHA = (short)0x6CCC;
      static byte[] rsaPrivateKey = // Note this is 338 bytes long and is a 1024 bit CRT
      formatted Private key
      {
      (byte)0xC2, (byte)0x01, (byte)0x05,(byte)0xC2, (byte)0x41, (byte)0x00,
      (byte)0xC5, (byte)0xD7, (byte)0x20, (byte)0x28, (byte)0xFA, (byte)0xA7,
      (byte)0x91, (byte)0x55, (byte)0x23, (byte)0xE2, (byte)0x0D, (byte)0xE4,
      (byte)0x28, (byte)0x7C, (byte)0x65, (byte)0xB7,
      (byte)0x18, (byte)0x59, (byte)0xD9, (byte)0x0D, (byte)0xBA, (byte)0xE7,
      (byte)0xCF, (byte)0x6A,
      (byte)0xF1, (byte)0xE3, (byte)0x10, (byte)0xC3, (byte)0x7E, (byte)0x48,
      (byte)0x0D, (byte)0xBC,
      (byte)0x76, (byte)0x7B, (byte)0x04, (byte)0x86, (byte)0xB6, (byte)0x7F,
      (byte)0xCD, (byte)0x6C,
      (byte)0x84, (byte)0x1A, (byte)0x0B, (byte)0x86, (byte)0xB9, (byte)0x96,
      (byte)0xAA, (byte)0x83,
      (byte)0x68, (byte)0x63, (byte)0x3C, (byte)0x4D, (byte)0x43, (byte)0x84,
      (byte)0xB8, (byte)0x6D,
      (byte)0x48, (byte)0xAA, (byte)0xC4, (byte)0xC9, (byte)0x1B, (byte)0x50,
      (byte)0x47, (byte)0x49,
      (byte)0xC2, (byte)0x41, (byte)0x00,
      (byte)0xC9, (byte)0x5E, (byte)0x4A, (byte)0x08, (byte)0x0E, (byte)0xDC,
      (byte)0xA5, (byte)0x45,
      (byte)0x90, (byte)0x1C, (byte)0x52, (byte)0xF8, (byte)0x3E, (byte)0xB0,
      (byte)0x6B, (byte)0x8F,
(byte)0xCF, (byte)0xEA, (byte)0xA8, (byte)0xF5, (byte)0x1E, (byte)0xAD,
(byte)0xD3, (byte)0x82,
(byte)0x52, (byte)0x30, (byte)0x43, (byte)0x04, (byte)0x9C, (byte)0x25,
(byte)0x63, (byte)0x87,
(byte)0x37, (byte)0x20, (byte)0x64, (byte)0x3F, (byte)0x16, (byte)0xB0,
(byte)0x50, (byte)0xCC,
(byte)0x38, (byte)0x06, (byte)0x1B, (byte)0xDF, (byte)0xE6, (byte)0x78,
(byte)0xC3, (byte)0x99,
(byte)0xF7, (byte)0xC4, (byte)0x3F, (byte)0x95, (byte)0x81, (byte)0xE0,
(byte)0x77, (byte)0x5C,
(byte)0xA3, (byte)0x78, (byte)0xB8, (byte)0x9C, (byte)0x8D, (byte)0x9B,
(byte)0x46, (byte)0x5D,
(byte)0xC2, (byte)0x41, (byte)0x00, (byte)0x7F, (byte)0xDE, (byte)0x17,
(byte)0x36, (byte)0xDA, (byte)0x18, (byte)0x1E, (byte)0xEF,
(byte)0xD0, (byte)0x50, (byte)0xBD, (byte)0x2C, (byte)0x57, (byte)0xBE,
(byte)0x10, (byte)0x7B, (byte)0x08, (byte)0x7D, (byte)0xC6, (byte)0xC7,
(byte)0xF5, (byte)0x76, (byte)0xA2, (byte)0x59,
(byte)0x35, (byte)0x7D, (byte)0xEA, (byte)0xB0, (byte)0x73, (byte)0xE6,
(byte)0x51, (byte)0xEB,
(byte)0xD3, (byte)0x07, (byte)0xDD, (byte)0x73, (byte)0x56, (byte)0x8B,
(byte)0x76, (byte)0x46, (byte)0x3F, (byte)0xAE, (byte)0x0A, (byte)0xB9,
(byte)0xA3, (byte)0xE3, (byte)0x0D, (byte)0x71,
(byte)0xFB, (byte)0xFE, (byte)0x55, (byte)0x02, (byte)0xF6, (byte)0xB6,
(byte)0x4D, (byte)0xC2, (byte)0x79, (byte)0x64, (byte)0xFD, (byte)0x28,
(byte)0xBB, (byte)0x13, (byte)0x23, (byte)0xB2,
(byte)0xC2, (byte)0x41, (byte)0x00,
(byte)0xA3, (byte)0x8D, (byte)0xA3, (byte)0xED, (byte)0x9C, (byte)0xC2,
(byte)0x18, (byte)0xB8,
(byte)0x9D, (byte)0x10, (byte)0x8D, (byte)0x51, (byte)0x58, (byte)0x52,
(byte)0xF6, (byte)0xB7,
(byte)0xB5, (byte)0xEE, (byte)0xD9, (byte)0x2C, (byte)0xAB, (byte)0x9E,
(byte)0x65, (byte)0xEF,
(byte)0xD0, (byte)0x86, (byte)0x59, (byte)0xDE, (byte)0x73, (byte)0xB0,
(byte)0x57, (byte)0x82,
(byte)0xBD, (byte)0x24, (byte)0x17, (byte)0xEA, (byte)0xD2, (byte)0x46,
(byte)0xB7, (byte)0x69,
(byte)0x85, (byte)0x90, (byte)0x0E, (byte)0x85, (byte)0x53, (byte)0x3A,
(byte)0x06, (byte)0x3E,(byte)0xDA, (byte)0x76, (byte)0x67, (byte)0x6C, (byte)0xAC,
(byte)0x6B, (byte)0xB5, (byte)0x17,
(byte)0xCB, (byte)0x62, (byte)0x39, (byte)0x8A, (byte)0xD4, (byte)0x04,
(byte)0xBA, (byte)0xD9,
(byte)0xC2, (byte)0x41, (byte)0x00,
(byte)0x44, (byte)0x03, (byte)0x03, (byte)0xB0, (byte)0x1B, (byte)0x0C,
(byte)0xED, (byte)0x09,
(byte)0x44, (byte)0xB6, (byte)0x3C, (byte)0x53, (byte)0xBA, (byte)0x20,
(byte)0xAE, (byte)0x03,
(byte)0xA1, (byte)0xAE, (byte)0xD9, (byte)0x28, (byte)0x09, (byte)0x17,
(byte)0x9E, (byte)0xC3, (byte)0x7A, (byte)0x6C, (byte)0xF0, (byte)0x85,
(byte)0xC3, (byte)0x13, (byte)0x61, (byte)0xBD,
(byte)0x4E, (byte)0xA2, (byte)0x33, (byte)0x19, (byte)0x97, (byte)0xD9,
(byte)0x2F, (byte)0x40,
(byte)0xFA, (byte)0x7F, (byte)0x1D, (byte)0xB5, (byte)0x0E, (byte)0xCB,
(byte)0xA5, (byte)0x0D, (byte)0x00, (byte)0xC1, (byte)0x18, (byte)0xD4,
(byte)0xAF, (byte)0x4C, (byte)0x18, (byte)0x24,
(byte)0x82, (byte)0xD6, (byte)0x08, (byte)0x4C, (byte)0x60, (byte)0x0B,
(byte)0x9C, (byte)0xC5
};
static byte[] rsaPublicKey = // Note this is 140 bytes long
{
(byte)0xC1, (byte)0x01, (byte)0x05,
(byte)0xC0, (byte)0x81, (byte)0x00,
(byte)0x9B, (byte)0x9E, (byte)0xC6, (byte)0x74, (byte)0x65, (byte )0x5A,
(byte)0xBC, (byte)0xBA,
        (byte)0xC6, (byte)0xB0, (byte)0x5D, (byte)0x3C, (byte)0x24, (byte )0x3C,
        (byte)0x90, (byte)0x59,
        (byte)0x7D, (byte)0x5B, (byte)0x6D, (byte)0x03, (byte)0x7B, (byte )0xAD,
        (byte)0x9A, (byte)0xB4,
        (byte)0x58, (byte)0xFE, (byte)0x80, (byte)0x22, (byte)0xEC, (byte)0xF0,
        (byte)0xAB, (byte)0x84,
        (byte)0xF9, (byte)0x07, (byte)0x84, (byte)0x91, (byte)0xE2, (byte )0x08,
        (byte)0xA6, (byte)0x9B,(byte)0xED, (byte)0xD7, (byte)0x45, (byte)0xC9, (byte)0xDD,
        (byte
        )0xBF, (byte)0xF8, (byte)0x7A,(byte)0x8B, (byte)0xE1, (byte)0x17, (byte)0xCD,
        (byte)0x3D, (byte
        )0xD3, (byte)0x6F, (byte)0xB2,(byte)0x59, (byte)0x0C, (byte)0x0E, (byte)0x47,
        (byte)0x0B, (byte
        )0x8E, (byte)0x83, (byte)0xC5,(byte)0x0F, (byte)0xAF, (byte)0xD8, (byte)0x21,
        (byte)0x0C, (byte)
        0xD1, (byte)0x0B, (byte)0xB4,(byte)0x24, (byte)0x8D, (byte)0x66, (byte)0xAC,
        (byte)0x93, (byte)0xA3, (byte)0xE4, (byte)0x61,
        (byte)0xEF, (byte)0x26, (byte)0x50, (byte)0x1C, (byte)0x30, (byte)0xED,
        (byte)0x73, (byte)0xF1,
        (byte)0x92, (byte)0xD8, (byte)0x2C, (byte)0xC6, (byte)0x38, (byte)0xD4,
        (byte)0x6D, (byte)0x81,
        (byte)0x48, (byte)0x2B, (byte)0xCC, (byte)0x42, (byte)0xF8, (byte)0x60,
        (byte)0x61, (byte)0xAD, (byte)0x8D, (byte)0x7F, (byte)0x8D, (byte)0x6D,
        (byte)0x87, (byte)0xBF, (byte)0x7D, (byte)0x1C,
        (byte)0x61, (byte)0x2B, (byte)0xC0, (byte)0x42, (byte)0x47, (byte)0xDB,
        (byte)0xDD, (byte)0xC9,
        (byte)0x3F, (byte)0x07, (byte)0x23, (byte)0xE1, (byte)0x3D, (byte)0xDA,
        (byte)0xDB, (byte)0x85,
        (byte)0xC0, (byte)0x04, (byte)0x00,(byte)0x01, (byte)0x00, (byte)0x01,
        };

        static byte[] PIN_STRING =
        {
        (byte)0x87,(byte)0x65,(byte)0x43,(byte)0x21};
        static byte[] singleDESKey =
        {
        (byte)0x83,(byte)0x25,(byte)0xAF,(byte)0x49,(byte)0xE4,(
        byte)0xAB,(byte)0x6F,(byte)0x17
        };
        static byte[] tripleDESKey =
        {
        (byte)0x83,(byte)0x25,(byte)0xAF,(byte)0x49,(byte)0xE4,(
        byte)0xAB,(byte)0x6F,(byte)0x17,
        (byte)0x38, (byte)0x12, (byte)0xA4, (byte)0x19, (byte)0xC6,
        (byte)0x3B, (byte)0xE7, (byte)0x71
        };
        OwnerPIN pin;
        Cipher cipherDES;
        Cipher cipherRSA;
        DESKey deskey;
        DESKey des3key;
        RSAPrivateCrtKey rsa_PrivateCrtKey ;
        //RSAPrivateKey rsa_PrivateKey ;
        RSAPublicKey rsa_PublicKey ;
        MessageDigest messageDigest;

//       constructor
        private key(byte buffer [], short offset ,byte length ){
        pin = new OwnerPIN(PIN_TRY_LIMIT,MAX_PIN_SIZE);
        pin .update(PIN_STRING,(short)0,(byte)4);
        deskey = `
        DESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_DES,KeyBuilder.LENGTH_DES,false);
        des3key = (DESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
        KeyBuilder.LENGTH_DES3_2KEY,false);
        rsa_PrivateCrtKey = (RSAPrivateCrtKey)KeyBuilder.buildKey(
        KeyBuilder.TYPE_RSA_CRT_PRIVATE,KeyBuilder.LENGTH_RSA_1024,false);
        rsa_PublicKey =
        (RSAPublicKey)KeyBuilder.buildKey(KeyBuilder.TYPE_RSA_PUBLIC,KeyBuilder.LENGTH_RSA
        _1024,false);
        cipherDES = Cipher.getInstance (Cipher.ALG_DES_ECB_NOPAD, false);
        cipherRSA = Cipher.getInstance (Cipher.ALG_RSA_PKCS1, false);
        messageDigest = MessageDigest. getInstance (MessageDigest.ALG_SHA, false);


                //rsa_PrivateKey = (RSAPrivateKey)KeyBuilder.buildKey(
        KeyBuilder.TYPE_RSA_PRIVATE,KeyBuilder.LENGTH_RSA_1024,false);

        if ( buffer [ offset ] == 0) {
                register () ;
        }
                else {
                register ( buffer , (short) ( offset +1) ,( byte) (
                buffer [ offset ]) ) ;
                        }
                }
//               install method
                public static void install (byte buffer [], short offset ,byte length ){
                new key(buffer , offset , length ) ;
                }
//               select method
                public boolean select (){
                pin . reset () ;
                return true ;
                }

                public void process(APDU apdu) {
                        // Good practice: Return 9000 on SELECT
                byte apduBuffer [] = apdu. getBuffer () ;
                if ( selectingApplet () ){
                ISOException.throwIt (ISO7816.SW_NO_ERROR);
                }
                if (apduBuffer[ISO7816.OFFSET_CLA] != MY_PROJECT_CLA){
                ISOException.throwIt (ISO7816.SW_CLA_NOT_SUPPORTED);
                }
                byte ins = apduBuffer[ISO7816.OFFSET_INS];
                switch ( ins ){
                case PIN_CHECK: pinCheck(apdu); return;
                case RSA: RSAencode(apdu); return;
                case DES3: DES3encode(apdu); return;
                case DES: DESencode(apdu); return;
                case SHA: SHAdigest(apdu); return;
                case RSA_D: RSAdecode(apdu); return;
                case DES3_D: DES3decode(apdu); return;
                case DES_D: DESdecode(apdu); return;
                case SHA_V: SHAverify(apdu); return;
                case SIGN_TEXT: signText (apdu); return;
                case VERIFY_TEXT: verifyText (apdu);return;
                case DUMMY: dummyMethod(apdu);return;
                case TOKEN: tokenize(apdu);return;
                default : ISOException.throwIt (ISO7816.
                SW_INS_NOT_SUPPORTED);
                }
                }

                private void pinCheck(APDU apdu){
                byte apduBuffer [] = apdu. getBuffer () ;
                short byteRead = (short) (apdu.setIncomingAndReceive());
                if (byteRead != 4) ISOException.throwIt (ISO7816.SW_WRONG_LENGTH);
```

```
            if (pin .check(apduBuffer ,( short)ISO7816.OFFSET_CDATA,(byte)4) ==
            false) ISOException.throwIt(WRONG_PIN);
            }

            private void RSAencode(APDU apdu){
            byte apduBuffer [] = apdu. getBuffer () ;
            if ( ! pin . isValidated () )ISOException.throwIt
            (ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
            short byteRead = (short) (apdu.setIncomingAndReceive());
//           Create Cipher
            rsa_PrivateCrtKey.setQ( rsaPrivateKey , (short) 6, (short)64);
            rsa_PrivateCrtKey. setP ( rsaPrivateKey , (short) 73, (short)64);
            rsa_PrivateCrtKey.setPQ( rsaPrivateKey , (short)140, (short)64);
            rsa_PrivateCrtKey.setDQ1(rsaPrivateKey, (short)207, (short)64);
            rsa_PrivateCrtKey.setDP1(rsaPrivateKey , (short)274, (short)64);
            cipherRSA.init( rsa_PrivateCrtKey ,Cipher.MODE_ENCRYPT);
            short outbytes = cipherRSA.doFinal(apduBuffer ,( short)ISO7816.
            OFFSET_CDATA, byteRead, apduBuffer, (short)ISO7816.
            OFFSET_CDATA);
//           Send results
            apdu.setOutgoing () ;
            apdu.setOutgoingLength((short) outbytes ) ;
            apdu.sendBytesLong(apduBuffer, (short)ISO7816.OFFSET_CDATA, (
            short)outbytes);
            }
            private void DESencode(APDU apdu){
            byte apduBuffer [] = apdu. getBuffer () ;
            if (! pin . isValidated () ) ISOException.throwIt (ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
            short byteRead = (short) (apdu.setIncomingAndReceive());
//          if (byteRead != 8) ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
//           create cipher
            deskey.setKey(singleDESKey,(short)0);
            cipherDES.init(deskey,Cipher.MODE_ENCRYPT);
            short outbytes = cipherDES.doFinal(apduBuffer ,( short)ISO7816.
                        OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
                        OFFSET_CDATA);
//                       send results
                        apdu.setOutgoing () ;
                        apdu.setOutgoingLength( outbytes ) ;
                        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                        )outbytes);
            }
                        private void DES3encode(APDU apdu){
                        byte apduBuffer [] = apdu. getBuffer () ;
                        if (! pin . isValidated ()
                        ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                        short byteRead = (short) (apdu.setIncomingAndReceive());
                  if (byteRead != 8)ISOException.throwIt (ISO7816.SW_WRONG_LENGTH);
//                       create cipher
                        des3key.setKey(tripleDESKey,(short)0) ;
                        cipherDES. init (des3key,Cipher.MODE_ENCRYPT);
                        short outbytes = cipherDES.doFinal(apduBuffer ,( short)ISO7816.
                        OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
                        OFFSET_CDATA);
//                       send results
                        apdu.setOutgoing () ;
                        apdu.setOutgoingLength((short) outbytes ) ;
                        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                        )outbytes);
            }
                        private void SHAdigest(APDU apdu){
                        byte apduBuffer [] = apdu. getBuffer () ;
                        if (! pin . isValidated () )ISOException.throwIt
(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                        short byteRead = (short) (apdu.setIncomingAndReceive());
//                       create signature
                        messageDigest. reset () ;
                        short outbytes = messageDigest.doFinal(apduBuffer ,(
short)ISO7816.

      OFFSET_CDATA,(short)byteRead,apduBuffer,(short)(ISO7816.OFFSET_CDATA+byteRead));
//                       send results
                        apdu.setOutgoing () ;
                        apdu.setOutgoingLength((short) (byteRead+outbytes)) ;
                        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                        )(outbytes+byteRead));
                        }
                        private void RSAdecode(APDU apdu){
                        byte apduBuffer [] = apdu. getBuffer () ;
                        if ( ! pin . isValidated () )ISOException.throwIt
(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                        short byteRead = (short) (apdu.setIncomingAndReceive());
//                       create cipher
                        rsa_PublicKey.setModulus(rsaPublicKey, (short) 6, (short)128);
                        rsa_PublicKey.setExponent(rsaPublicKey, (short)137, (short)3) ;
                        cipherRSA. init ( rsa_PublicKey , Cipher.MODE_DECRYPT);
                        short outbytes = cipherRSA.doFinal(apduBuffer ,( short)ISO7816.
                        OFFSET_CDATA, byteRead, apduBuffer, (short)ISO7816.
                        OFFSET_CDATA);
//                       Send results
                        apdu.setOutgoing () ;
                        apdu.setOutgoingLength((short) outbytes ) ;
                        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA, (
                        short)outbytes);
                        }
                        private void DES3decode(APDU apdu){
                        byte apduBuffer [] = apdu. getBuffer () ;
                        if (! pin . isValidated () )ISOException.throwIt
(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                        short byteRead = apdu.setIncomingAndReceive();
                        if (byteRead != 8)ISOException.throwIt
(ISO7816.SW_WRONG_LENGTH);
//                       create cipher
                        des3key.setKey(tripleDESKey,(short)0) ;
                        cipherDES. init (des3key,Cipher.MODE_DECRYPT);
                        short outbytes = cipherDES.doFinal(apduBuffer ,( short)ISO7816.
                        OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
                        OFFSET_CDATA);
//                       send results
                        apdu.setOutgoing () ;
                        apdu.setOutgoingLength((short) outbytes ) ;

      apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short)outbytes);
                        }
                        private void DESdecode(APDU apdu){
                              byte apduBuffer [] = apdu. getBuffer () ;
                              if (! pin . isValidated ()
)ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                              short byteRead = apdu.setIncomingAndReceive();
                              if (byteRead != 8)ISOException.throwIt
(ISO7816.SW_WRONG_LENGTH);
//                             create cipher
                              deskey.setKey(singleDESKey,(short)0);
                              cipherDES. init (deskey,Cipher.MODE_DECRYPT);
                              short outbytes = cipherDES.doFinal(apduBuffer ,(
short)ISO7816.
```

```
                              OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
                              OFFSET_CDATA);
//                             send results
                              apdu.setOutgoing () ;
                              apdu.setOutgoingLength((short) outbytes ) ;

     apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                              ) outbytes);
                        }
                              private void SHAverify(APDU apdu){
                              byte apduBuffer [] = apdu. getBuffer () ;
                              if (! pin . isValidated ()
)ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                              short byteRead = apdu.setIncomingAndReceive();
                              byte in_sha [] = new byte[(byte) 20];
                              byte text [] = new byte[(byte) (byteRead-20)];
                              byte out_sha [] = new byte[(byte) 20];
                              for (short i=0;i<(short)(byteRead-20); i++){
                              text [ i]=apduBuffer [( short) (ISO7816.OFFSET_CDATA+i)];
                              }
                              short j=0;
                              for (short i=(short) (byteRead-20);i<byteRead;i++){
                              in_sha [ j++] = apduBuffer [( short)
(ISO7816.OFFSET_CDATA+i
                              )];
                              }
                              messageDigest.reset() ;
                              messageDigest.doFinal( text ,( short) 0,( short)
(byteRead-20),out_sha,(short)0) ;
                              boolean same_sha=true;
                              for (short i=0;i<(short)20;i++){
                              if ( in_sha [ i ]!= out_sha [ i ])same_sha=false ;
                              }
                              if (!same_sha)ISOException.throwIt(WRONG_SHA);
                        }
                              private void signText (APDU apdu){
                              byte apduBuffer [] = apdu. getBuffer () ;
                              if (! pin . isValidated () )ISOException.throwIt
(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                              short byteRead = (short) (apdu.setIncomingAndReceive());
//                             create signature
                              messageDigest. reset () ;
                              short digest = messageDigest.doFinal(apduBuffer ,(
short)ISO7816.

                              OFFSET_CDATA,(short)byteRead,apduBuffer,(short)(ISO7816.
                              OFFSET_CDATA+byteRead));
//                             Create Cipher
                              rsa_PrivateCrtKey.setQ( rsaPrivateKey , (short) 6,
(short)64);
                              rsa_PrivateCrtKey. setP ( rsaPrivateKey , (short) 73,
(short)64);
                              rsa_PrivateCrtKey.setPQ( rsaPrivateKey , (short)140,
(short)64);
                              rsa_PrivateCrtKey.setDQ1(rsaPrivateKey, (short)207,
(short)64);
                              rsa_PrivateCrtKey.setDP1(rsaPrivateKey , (short)274,
(short)64);
                              cipherRSA.init ( rsa_PrivateCrtKey , Cipher.MODE_ENCRYPT);
                              short outbytes = cipherRSA.doFinal(apduBuffer ,(
short)ISO7816.

                              OFFSET_CDATA, (short)(byteRead+digest), apduBuffer,
(short)

                              ISO7816.OFFSET_CDATA);
//                             Send results
```

```
                              apdu.setOutgoing () ;
                              apdu.setOutgoingLength((short) outbytes ) ;
                              apdu.sendBytesLong(apduBuffer,
(short)ISO7816.OFFSET_CDATA, (
                              short)outbytes);
                              }
                              private void verifyText (APDU apdu){
                              byte apduBuffer [] = apdu. getBuffer () ;
                              if ( ! pin . isValidated () )ISOException.throwIt
(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                              short byteRead = (short) (apdu.setIncomingAndReceive());
//                             create cipher
                              rsa_PublicKey .setModulus(rsaPublicKey, (short) 6,
(short)128);
                              rsa_PublicKey .setExponent(rsaPublicKey, (short)137,
(short)3) ;
                              cipherRSA.init( rsa_PublicKey , Cipher.MODE_DECRYPT);
                              short outbytes = cipherRSA.doFinal(apduBuffer ,(
short)ISO7816.
                              OFFSET_CDATA, byteRead, apduBuffer, (short)ISO7816.
                              OFFSET_CDATA);
//                             create signature
                              byte encipheredBuffer []=new byte[(byte) outbytes ];
                              for (short i=0;i<outbytes; i++){
                                    encipheredBuffer [ i]=apduBuffer [( short) (ISO7816.
                                               OFFSET_CDATA+i)];
                              }
                                          byteRead = (short) encipheredBuffer . length ;
                                          byte in_sha [] = new byte[(byte) 20];
                                          byte text [] = new byte[(byte) (byteRead-20)];
                                          byte out_sha [] = new byte[(byte) 20];
                                          for (short i=0;i<(short)(byteRead-20);i++){
                                          text [ i]= encipheredBuffer [ i ];
                              }
                                    short j=0;
                                    for (short i=(short) (byteRead-20);i<byteRead;i++){
                                               in_sha [ j++] = encipheredBuffer [ i ];
                              }
          messageDigest.reset () ;
          messageDigest.doFinal( text ,( short) 0,( short) (byteRead-20),out_sha,(
                                               short)0) ;
                                               boolean same_sha=true;
                                               for (short i=0;i<(short)20;i++){
          if ( in_sha [ i ]!= out_sha [ i ])same_sha=false ;
                              }

     if(!same_sha)ISOException.throwIt(WRONG_SHA);
                              }
                                               private void dummyMethod(APDU apdu){
                                          byte apduBuffer [] = apdu. getBuffer () ;
                                    if (! pin . isValidated () )ISOException.throwIt
(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                                               short byteRead =
apdu.setIncomingAndReceive();
                                               apdu.setOutgoing () ;
                                               apdu.setOutgoingLength(byteRead);

     apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short)byteRead);

                              }



     }
```

6943-112