Developing a Contactless Bankcard Fare Engine for Transport for London

by

Peter S. C. Lau

B.S. Electrical Engineering and Computer Sciences
University of California, Berkeley, 2007

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN TRANSPORTATION
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2009

Signature of Author: _____
Department of Civil and Environmental Engineering
May 21, 2009

Certified by: _____
George Kocur
Senior Lecturer of Civil and Environmental Engineering
Thesis Supervisor

Certified by: _____
Nigel H. M. Wilson
Professor of Civil and Environmental Engineering
Director, Master of Science in Transportation

Accepted by: _____
Daniele Veneziano
Professor of Civil and Environmental Engineering
Chairman, Departmental Committee for Graduate Students

Developing a Contactless Bankcard Fare Engine for Transport for London

by

Peter S. C. Lau

Submitted to the Department of Civil and Environmental Engineering on
May 21, 2009 in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Transportation

ABSTRACT

This thesis investigates the design of a fare engine which operates within the constraints of using contactless bankcards as a fare instrument, while satisfying the complex current and future fare requirements of Transport for London (TfL). A fare engine is a system which transforms user transactions at fare gates and validators into chargeable fares. Contactless bankcard fare payment differs from current fare smartcard systems by requiring a centralized fare engine.

The proposed fare engine utilizes a data structure which maintains each user's journey history in three successive tiers of linked objects. This structure enables transactions to be correctly sequenced without a guarantee of in-order arrival of gate and validator transactions. A cleanup routine prevents the data structure from growing without bound as journey history accumulates. A dynamic journey linking mechanism allows the effect of inserted transactions to be propagated throughout the data structure and reflected in the affected journeys with near-constant time complexity. This ensures scalability while providing real-time feedback for customer service and payment authorization needs.

A solution is devised for the coupling of arbitrary origin-destination fares with zonal period tickets. The paradigm of automatic ticket selection is introduced, overcoming the limitations of the existing capping algorithm used by TfL. Through the tracking of parallel fare scenarios, passengers are guaranteed a total fare no higher than if they had purchased the optimal period ticket for their usage profile.

With the solutions proposed in this thesis, a contactless bankcard fare engine for TfL appears feasible.

Thesis Supervisor: George Kocur
Title: Senior Lecturer of Civil and Environmental Engineering

Acknowledgements

I would like to express my thanks to my advisor, George Kocur, who never failed to provide gems of expertise and timely insight. My appreciation also goes to Professor Nigel Wilson and John Attanucci for their guidance and advice throughout the course of this degree.

This research is funded by Transport for London as part of its collaboration with MIT. I would like to acknowledge Will Judge and Shashi Verma of TfL for the support they have lent to this project, as well as the staff of Fares and Ticketing – Brian Dobson, Ian Thornley, Behdad Haddadien, Peter Svensson and Andrew Gaitskell for their assistance and feedback.

I want thank all of my colleagues in the MST program for their support and in particular, Michael Frumin and Candace Brakewood for their help in the area of fare payment. I am also grateful to Ginny Siggia, whose hard work has kept this program running smoothly.

Finally I owe the deepest appreciation to my parents and to Sara, who has given me endless encouragement and the strength to see this thesis through to completion from its uncertain beginnings.

Table of Contents

1 Introduction and Background

This thesis formulates the design of a contactless bankcard fare engine for Transport for London (TfL). The first chapter provides an introduction to the current state of the art in fare payment. We examine the drawbacks of current fare payment systems and why contactless bankcards are an attractive alternative to existing technologies.

In chapter 2, we frame the thesis question inside the context of a fare collection system. We define a fare engine, and the goals to be met along the way as we design one. In chapter 3, we examine TfL's current fare structure model and frame it in a systematic fare structure which we develop. In chapter 4, the requirements of the contactless bankcard fare engine are laid out, based on the fare structure constructed in chapter 3, as well as the expected future needs of TfL. We also review ongoing plans for the future ticketing fare collection system. This is the infrastructure that TfL is planning to support contactless bankcard fare payment. We define the role of the fare engine within this system.

Chapters 5 to 7 are devoted to the design of the fare engine. In chapter 5, we outline the two major modules that constitute the fare engine, the fare processor and the journey processor. We describe the data flows that connect the fare engine to other systems within the fare collection system and the data flow that connects the journey processor and the fare processor within the fare engine. Chapter 6 is an in depth discussion of the journey processor where we describe a solution to the problem of tap sequencing without guaranteed in-order arrival and a mechanism for dynamic journey linking. In chapter 7, we give a detailed treatment of the fare processor, including approaches for implementing National Rail support and true best value.

1.1 *Traditional Electronic Fare Payment Technologies*

Prevailing electronic fare payment technologies in use today fall into two categories – *magnetic farecards* and *smartcards*.

1.1.1 Magnetic Farecards

Magnetic farecards are produced from either a paper or polyester base material and contain a longitudinal magnetic stripe which is read-write capable. This technology dates from the 1960s, having first been introduced in London and on the Long Island Railroad of New York. Subsequent deployments on the Bay Area Rapid Transit (BART) system of San Francisco in 1972 and

Washington DC Metro (WMATA) in 1976 are more sophisticated and are capable of distance based fares [11].

The strengths of magnetic farecards include the low cost of the fare medium (as low as $0.02 per card as of 2003) and automation in vending (by means of Ticket Vending Machines or TVMs) and entry/exit control (by means of automatic fare gates). Finally, the discarding of farecards with small residual value by customers becomes an additional revenue stream for the transit agency in offsetting the cost of the system's operation. Both the Chicago Transit Authority (CTA) and WMATA experience unused fares of more than $3 million per year [12].

However, magnetic fare cards have limited data capacity, restricting the agencies' ability to implement multi-ride and fare pass options. Interoperability between transit systems is weak. The fare processing equipment (both TVMs and fare gates) for magnetic farecards is expensive and requires considerable maintenance due to the number of moving parts involved. Attempts to reduce these costs by using swipe readers, such as New York City Transit (NYCT) has done, have resulted in an unreliable user experience [12]. Finally, magnetic tickets generally have weak or no security features beyond the inaccessibility of card reading equipment; in other words, they rely on security from obscurity, which is unadvisable. Magnetic farecards are subject to exploitation and counterfeiting by individuals possessing the necessary equipment and technical knowledge.

1.1.2 Transit Smartcards

Unlike magnetic fare cards, smartcards are intended to be reusable over a long period. They are usually made of rigid plastic and conform to standardized credit card dimensions. ID-1 of the ISO/IEC 7810 standard defines this to be 85.60mm × 53.98 mm. Modern smartcards contain a microprocessor capable of basic data processing and substantial storage compared to magnetic farecards. The card communicates with the reader through either a contact interface, which consists of a set of small metallic contacts on the face of the card, or through a *contactless* interface. A contactless interface consists of an antenna coil embedded inside the plastic of the card that serves both to collect power to operate the microprocessor and to transmit and receive data from the reader. The technology used to implement contactless smartcards is known as Radio Frequency Identification (RFID). The radio frequency and transmission protocol used in RFID have been standardized into several standards, one of which is ISO14443, a popular standard for both transit and financial applications.

A pilot study in contactless transit smartcards took place in London as early as 1990. Since then, this technology has been introduced in over fifty transit systems worldwide [12]. The first widespread deployment was the multi-modal Octopus Card in Hong Kong in 1996. The first US deployment was launched by WMATA in 2000, while London Transport (now Transport for London) introduced the current form of its smartcard system, the Oyster Card, in 2003.

Smartcards have some notable advantages as a transit fare technology. Durable and reusable smartcards eliminate the waste associated with disposable magnetic farecards, and reduce ongoing costs. In particular, contactless reader devices have no openings or moving parts, enabling them to be completely sealed against environmental factors and vandalism. They have significantly lower acquisition and maintenance costs compared to traditional technologies. For this reason smartcard deployments in transit have largely gravitated toward contactless interfaces. The data capacity and processing capabilities of smartcards support advanced applications, including a combination of passes and stored value and business logic for implementing and tracking complex fare structures. Finally, contactless smartcards are extremely easy and intuitive to use. Reduced user interaction time has increased passenger throughput on both rail and bus systems.

One former disadvantage of transit smartcards is the high unit cost of each card. However this is becoming less of an issue today as the unit cost has dropped from over $10 in 1994 to less than $1 [12]. This cost is often passed directly onto the user in the form of a purchase price or deposit, impacting the take-up of the technology. Although many systems have been designed with regional participation, a multitude of competing transit smartcards standards still proliferate. Inter-regional interoperability is weak.

1.1.3 Common Challenges

The two electronic fare technologies each have their pros and cons. However, they share a common drawback. As currently implemented, most transit fare payment systems are solely-owned and custom-designed [7]. Each system is tailor designed for the transit property using it, often at great cost and with limited opportunities to leverage economies of scale. Transit agencies must set up and maintain the infrastructure necessary to support their fare payment system. This infrastructure must support:

- **Card lifecycle management** – This includes the procurement, distribution (issuing), tracking, replacement and disposal of farecards or smartcards. For example, a network of ticket/reload machines is required. This is in turn associated with high acquisition, maintenance, and cash handling costs. Another cost is the payment of commissions to distribution and reload vendors (such as convenience stores).

- **Revenue allocation** – If the fare payment technology is shared among multiple agencies, a settlement clearinghouse must be established for distributing funds among the participating agencies.

- **Customer service** – A complete customer service system, including sales, inquiries, dispute resolution and fraud protection must be implemented. Staff must be trained to use specialized fare processing and diagnostic equipment.

As the existing generation of fare payment systems mature and their replacement becomes a concern on the horizon, many agencies are seeking to reduce their role as an issuer of closed fare payment media. Contactless bankcards are gaining traction as a viable alternative to existing fare payment technologies.

1.2 *Overview of Contactless Bankcards*

1.2.1 Contactless Smartcard as a Financial Instrument

We have discussed the value of contactless smartcards as fare payment medium. The value of contactless technology has also been recognized by the financial industry. The term *contactless bankcard* (CLBC) covers the application of RFID technology to credit, debit and prepaid cards. In particular, we use the term to refer to credit, debit and prepaid cards that are compatible with one of the major payment networks. The three largest networks of interest, in no particular order, are Visa, MasterCard and American Express.

1.2.2 State of the Technology in the US

As of 2007, 35 million contactless bankcards are in circulation in the US, up from 19 million in 2006 and 13 million in 2005. In other words, up to nine percent of the US population now holds a contactless credit or debit card. Consumer research has shown widespread public acceptance of and

satisfaction with contactless bankcards [7, 16]. In 2005, contactless payment was accepted in over 32,000 merchant locations, a number likely to have since increased [7]. Contactless bankcards are now issued in the US by most major banks and all three payment networks. Well known national merchants, such as McDonald's, 7-Eleven, CVS Pharmacy and AMC Theater now accept contactless payments. Contactless bankcard readers are beginning to be seen on soft-drink vending machines and at smaller local merchants. The rapid introduction of contactless bankcards in the US is aided by the relative simplicity of the US bankcard requirements. Traditional US bankcards employ an unencrypted magnetic stripe which stores the bankcard number. Weak card security is backed up by real-time verification of the transaction against the bankcard issuer (online authorization) where available, and to a lesser extent, signature request. Duplication of this functionality over a contactless interface is relatively straightforward. From both a merchant's and a customer's perspective, contactless payment in the US is simply another way to present a bankcard to the point-of-sale system with little distinction between the functionality of the contactless payment card and the standard magnetic stripe card [7].

Payment Network	Contactless Bankcard Product
Visa	PayWave
MasterCard	PayPass
American Express	ExpressPay

Figure 1.1 – Major bankcard payment networks.

1.2.3 State of the Technology in UK and Europe

In the UK and Europe, however, contactless bankcards have yet to gain traction owing to the more complex bankcard technology currently in use. UK and Europe bankcards conform to the Europay-MasterCard-Visa (EMV) standard which utilizes a contact smartcard. EMV bankcards are capable of both online and offline transactions. Online transactions are authorized against the card issuer in a similar fashion to what happens with US cards. In offline transactions, the EMV bankcard authorizes the transaction by itself without any communication with a remote entity.

These features are enabled by the intrinsic high security of an EMV card. Unlike US bankcards, an EMV card is able to authenticate itself as being genuine (in other words, it is very difficult to clone a usable EMV card). Furthermore, EMV cards can securely store and verify the PIN number that the user must enter to complete the transaction. In the UK, EMV bankcards are known as Chip-and-Pin

cards, referring to the microchip embedded inside the smartcard and prominent role of PIN numbers in transactions.

The security features of EMV have significantly reduced bankcard fraud. On the other hand, it has also slowed the introduction of contactless bankcards in regions, such as the UK, where the standard has been adopted. Development of a contactless version of EMV has been complicated by the complexity of the handshaking needed to authenticate a card, which involves a much higher volume of data transfer than the simpler US-style transaction. Another consideration has been the complex business requirements associated with offline transactions. For example, a transaction floor mechanism has been implemented in which contactless transactions are allowed until a floor limit, say £20 is reached, at which point the card must be used in a PIN-verified or online transaction to reset the limit. However these difficulties have been resolved. EMV compliant PayPass and PayWave cards were introduced in London in 2007 and are being offered by major UK banks, such as HSBC.

1.2.4 Bankcard Payment Process

As we have seen, contactless technology has brought significant changes to how customers use their bankcards, as well as the bankcards themselves and the customer-facing infrastructure for supporting them. However, the merchant-side services have remained largely similar to what a traditional bankcard transaction requires. In either case, the following entities are involved [10]:

- **Card Holder** – The customer.

- **Merchant** – This is the company or organization that the card holder is purchasing from. E.g. McDonalds or TfL.

- **Merchant Acquirer** – The merchant acquirer is a financial institution responsible for a merchant's transactions with the network. It accepts bankcard charges on the merchant's behalf and deposits funds into a bank account held in the merchant's name (either with the merchant acquirer or a different bank). In the case of member-structure based payment networks such as Visa and MasterCard, the merchant acquirer is typically either a bank that is a member of a payment network, or a consortium involving such a bank. An example of a merchant acquirer is Paymentech, a joint venture between Chase and First Data Corporation.

- **Issuing Bank (Issuer)** – Issuers are financial institutions that give the bankcards to customers and are ultimately responsible for the purchases they make. When a bankcard charge arrives at the issuer, the amount of the transaction is either posted to the user's credit account, or debited from the user's bank balance. The funds are then sent to the merchant acquirer. An issuer must also belong to a payment network. Examples of issuers include Citibank and HSBC.

- **Payment Network** – The payment network is the connection between the merchant acquirer and the issuer. The payment network forwards a charge from the merchant acquirer to the correct issuer. For this to happen, the merchant acquirer and issuer must belong to the same network. In practice, most merchant acquirers and issuing banks belong to both Visa and MasterCard, providing most bankcard users with a seamless and transparent experience. Note that some payment networks do not follow this model exactly. For example American Express, a payment network, is also itself a bank which performs merchant acquiring functions.

Below we will describe the processes involved in a typical bankcard transaction, as they apply to both traditional and contactless bankcard payments [7].

1. The merchant equipment, also known as the point-of-sale (POS) sends an authorization request to the merchant acquirer with the transaction amount and the card number.

2. The merchant acquirer may forward the authorization request through the payment network to the issuing bank of the card; it may check a shared database of credit card status without going to the issuing bank; or it may apply other rules, such as floor limits to authorize the transaction.

3. The issuing bank, if contacted, verifies the validity of the card (e.g. whether it has been marked stolen) and performs other checks to determine whether to accept or deny the charge. Tests may include whether there is a suspicious pattern of transactions and size of the transaction as it relates to the user's current credit balance and his credit limit. Once a decision is made it is passed back to the merchant acquirer, again via the payment network.

4. The merchant acquirer transmits the result back to the merchant, where it is displayed on the POS terminal.

The above authorization sequence occurs within a short interval (typically in the order of seconds) after the user's card has been swiped at the terminal. Bear in mind that this sequence of events is only to authorize a purchase. No charges are actually made until the end of each day, when the day's transactions are bundled together (captured) by the merchant, and sent to the merchant acquirer, who distributes the charges to the appropriate issuing banks. In return, funds are transmitted from issuing banks back to the merchant acquirers owed them, again routed though the payment network. This process is called *settlement*.

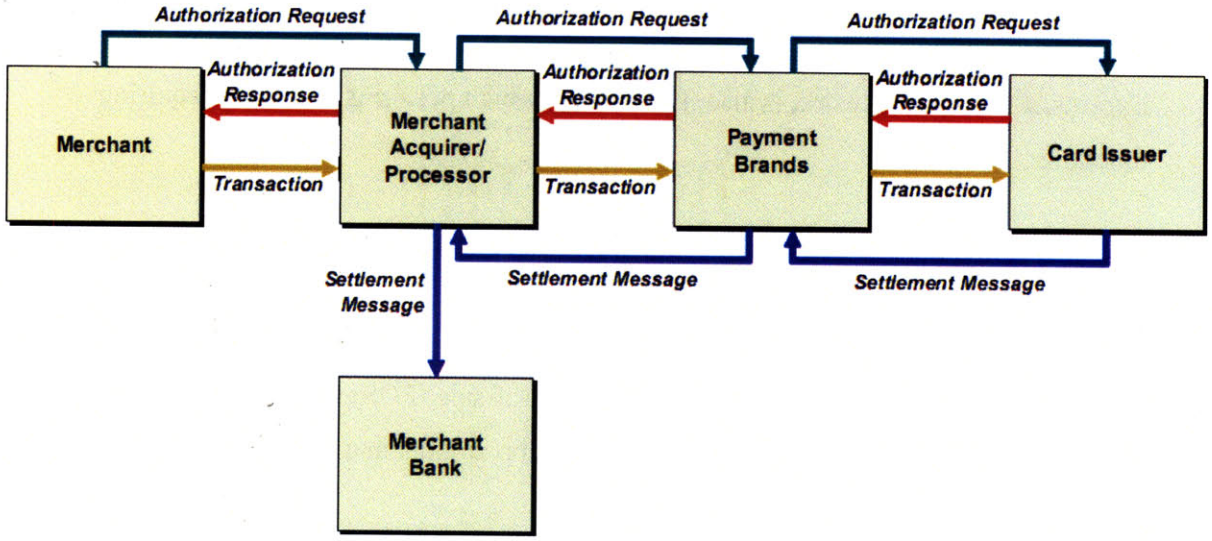


Figure 1.2 – Bankcard payment processes. Source: Smart Card Alliance/Booz Allen Hamilton [7]

1.3 *Direct Fare Payment with Contactless Bankcards*

1.3.1 Earlier Attempts at Transit-Contactless Bankcard Integration

Existing electronic fare payment systems are expensive to maintain. This has motivated the consideration of direct fare payment with contactless bankcards as an alternative to existing proprietary, closed-loop systems. *Direct* fare payment with contactless bankcards is not to be confused with the following similar, but *indirect* applications of contactless bankcards to fare payment.

- **Accepting contactless bankcards for fare product purchases** – This means enabling ticket vending machines and sales windows to accept contactless bankcards for fare purchases. This is not a true contactless bankcard fare payment system as the existing fare payment technology is still maintained. Under this scenario, a contactless bankcard is simply used to buy a fare instrument, which in turn is used by the passenger to access the transportation system. However, the increased speed of a contactless bankcard transaction compared to conventional bankcard or cash transactions could alleviate crowding and lines at ticket machines, a major source of delay for many customers.

- **Co-branded multi-application contactless cards** – These are specially designed contactless cards that are compatible with both bankcard and fare card standards. While this provides users with a comparable experience to true contactless bankcard fare payment, it does nothing for the transit agency. The transit agency must still maintain its current system and distribute contactless fare cards to users not equipped with a special multi-application card. In effect, a co-branded multi-application card is the same result as a contactless farecard and a contactless bankcard taped together back-to-back. An example of a co-branded multi-application contactless card is the Barclaycard Onepulse product, which is a combination of a standard Oyster card with a Visa PayWave card.

1.3.2 Benefits of Direct Fare Payment with Contactless Bankcards

A true contactless bankcard fare payment system offers users walk-up accessibility to transit services using a standard contactless bankcard belonging to one of the major bankcard networks. Users have the ability to enter a gated rail system by presenting their contactless bankcard at the fare gate, or board a bus by presenting the bankcard to a reader on the bus. They are able to do so without first purchasing a different fare medium and without obtaining a special 'transit enabled' bankcard. In subsequent discussion we will assume any discussion of contactless bankcard fare payment refers to *direct* fare payment with these features.

There are many benefits for both users and transit agencies if such a system could be effectively implemented [7]:

- Customers would be able to use an existing contactless bankcard issued by the financial institution which they already have an existing relationship. This means fewer pieces of plastic to carry and manage.

- If support for contactless bankcard fare payment became widespread among agencies, users would be able to use the same bankcard on different systems in different cities. De-facto interoperability would be achieved without agencies having to collaborate on a shared fare technology, an often cumbersome and rarely successful process.

- Transit agencies may no longer need to issue their own fare media to most or all of their riders, depending on the ultimate solution for unbanked riders. This has the potential for bringing significant cost savings to an agency, reducing cash handling, TVM maintenance, customer service and other farecard lifecycle costs.

- Customers would treat their transit fare as any other purchase made with a bankcard. Disputes, funds management, theft protection, initial issuance and reissuance of lost cards would be dealt with by the bankcard issuer, not the transit agency.

- Card issuers would be able to participate in co-branding and other promotional programs based on standard contactless bankcards, for example, in a similar fashion to airline loyalty credit cards.

1.3.3 Issues and Challenges

The premise of contactless bankcard fare payment seems attractive enough. However, many challenges, both institutional and technical stand in the way of seamless transit contactless bankcard integration. Identifying and solving these challenges is an active area of research at this current point in time. Some of these challenges are listed below:

Institutional Challenges

- **Fee structure for micropayments** - Merchants are charged a per-transaction fee, known as a discount rate. This discount rate includes an *interchange fee* charged by the payment network, as well as processing fees levied by the merchant acquirer and the issuer. The discount rate may contain a fixed component, making it uneconomical for transit agencies to

charge numerous small value transactions, or *micropayments* that correspond to single journeys. Negotiating a favorable fee structure is one way this issue can be solved. Another approach is a technological one. In a process called *aggregation*, the transit agency may opt to buffer and combine multiple fares into a single lumped amount before presenting it to the merchant acquirer. Negotiations between the bank card associations and transit agencies are ongoing to determine the rules under which aggregation may be done, including the resolution of the risks of nonpayment.

- **Double-ended fares** – Many transit agencies, such as TfL implement distance based or zonal based fares where fares are charged depending on the entry and exit locations. These are what we will call *double-ended* fares, as opposed to *single ended* fares which involve a fixed charge at the point of entry. With double ended fares the system has no way of knowing at the time of entry what the eventual fare will be. Whether it deals with this by not authorizing at all until the actual fare is known upon exit, authorizing a zero-pound fare at entry, authorizing a maximum fare at entry, or some other way is dependent entirely on the transit agency's choice of policy. Each alternative carries with it a different risk that needs to be assessed.

- **Unbanked users** – A significant portion of the population either does not have access to bankcards or chooses not to use one from personal preference. For reasons of equity, public transportation must be accessible to all. A transit agency implementing contactless bankcard fare payment must also cater to these users. It may do so by maintaining an existing farecard or cash payment system in parallel; however this would severely diminish the cost savings of moving to a contactless bankcard system to begin with. Pre-paid bankcards have been proposed as a means of tackling this problem; negotiations are ongoing in this arena also. If third-party issued prepaid cards are used, a major issue is how the fees of the prepaid cards are assessed, to the user or the agency. Alternatively, an agency could create its own prepaid card program, using bankcard standards; the costs could be lower than a transit-specific card because many services could be shared with the bank card payment stream or outsourced to the payment industry.

Technical Challenges

1. **Transaction speed** – Existing farecard technologies are gauged against a 300ms litmus test for performance. This has been found to be the threshold of allowable time for a farecard transaction that does not hamper customer throughput [7]. This transaction speed requirement means it would be difficult to authorize transactions online in real time. Transactions could be authorized online subsequent to boarding or entry, but this carries risk implications for the transit agency. Transactions could be undertaken in the offline mode of the bankcard, if it is supported, as in the case of contactless EMV bankcards. However offline contactless EMV processing carries additional complications with regard to a charge floor and Chip-and-Pin re-enablement, as described in section 1.2.3. Double ended fares must also be considered.

2. **Lack of on-card scratch pad** – Although this matter is still in negotiation, for a number of reasons it is unlikely that transit agencies will gain the ability to write to contactless bankcards presented to them. The lack of such writable space or *scratch pad* means a contactless bankcard fare payment system cannot implement the decentralized stored-value model used by most existing farecard and smartcard systems. Transactions must be transmitted to a server and processed in a centralized fashion. The next three challenges below are corollaries of this fact.

3. **Bus-based transactions** – A reliable communication link must be in place to allow bus based transactions to be transmitted to the centralized server, if near-real time authorization and fare processing is to be achieved. The need has to be met by existing radio and cellular technologies.

4. **Fare inspection** – Fare inspection becomes an issue as fare inspectors are not able to determine whether a passenger has validated their entry into the system by inspecting data on their bankcard. Inspectors must have some means of determining whether the read-only bankcard has been validated, for example, through access to a central server.

5. **Fare processing engine** – A system needs to be in place on a centralized server that takes bankcard transactions transmitted from gates and validators throughout the system and

assembles them into chargeable fares in a fashion consistent with the fare structure of the agency. This last point leads us to a discussion on the specific purpose of this thesis.

2 Research Question and Framework

2.1 *Definitions*

Before we can frame the research question, we will formalize two important definitions.

2.1.1 What is a Fare Collection System?

In the broadest sense, a *system* is an "assemblage or combination of elements or parts forming a complex or unitary whole" [17]. Therefore we define a fare collection system as a collection of components which allows revenue to be generated from passengers who pass through a public transportation system in the form of collected fares. Note that when we refer to a fare collection system, we are restricting ourselves to *automatic* fare collection systems which are self sufficient with human interaction limited only to mechanical tasks, such as maintenance, cash handling and replenishment of consumables.

A system for fare collection is composed of both tangible customer-facing hardware such as the fare medium, devices that process the fare medium (e.g. gates, reader and vending machines), as well as infrastructural elements such as communication links, distribution and sales networks, data servers and databases. A high level sketch of a fare collection system is presented in Figure 2.1. A lower level sketch of a very simple hypothetical magnetic stripe fare collection system will be discussed in this chapter.

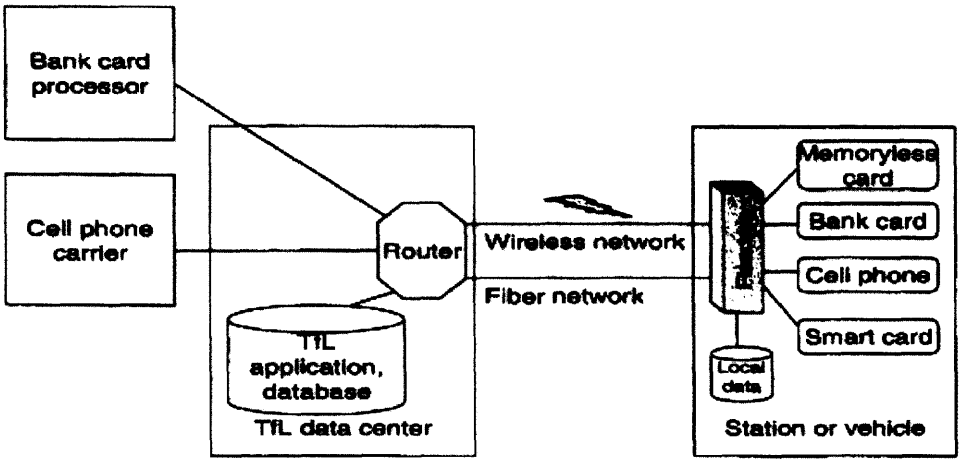


Figure 2.1 – Example of a fare collection system. [5]

2.1.2 *What is a Fare Engine?*

A fare engine is the part of an automatic fare collection system that physically implements the fare structure of a transit agency. As such, the input and outputs of a fare engine are consistent with the logical representation of a fare structure. The parallelism between a fare engine and a fare structure is shown in the diagram below. In chapter 3 we will examine what constitutes a fare structure and create a model for TfL's fare structure.

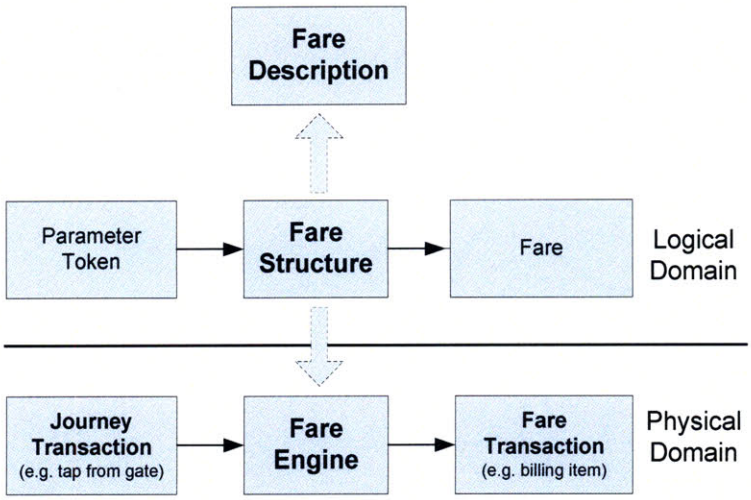


Figure 2.2 – Generic view of a fare engine as the physical manifestation of a fare structure within an automatic fare collection system.

2.1.3 *Example of a Simple Fare Collection System*

Below is an example of a very simple bus-only agency which uses a magnetic-stripe based fare collection system. The purpose of this example is to demonstrate the components of Figure 2.2 in a concrete context and show how they fit together.

Fare Description

The fare description is a simple statement explaining fares and rules.

"Every boarding costs $1.50. There are no transfers or concessions."

Fare Structure

Following the model for fare structures defined in chapter 3.2.1, the fare structure is a matrix with only one tier, labeled 'Fixed Fare'

Fixed Fare	$1.50

Figure 2.3 – Fare structure for our very simple fare collection system.

Fare Collection System

The fare collection system is summarized in the diagram below.

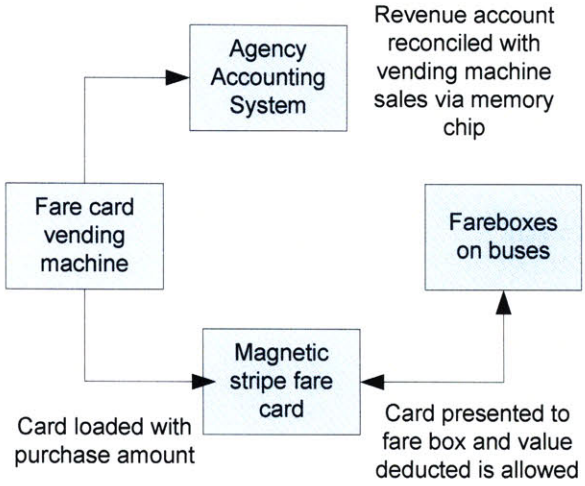


Figure 2.4 – Simple magnetic stripe fare collection system.

Fare Engine

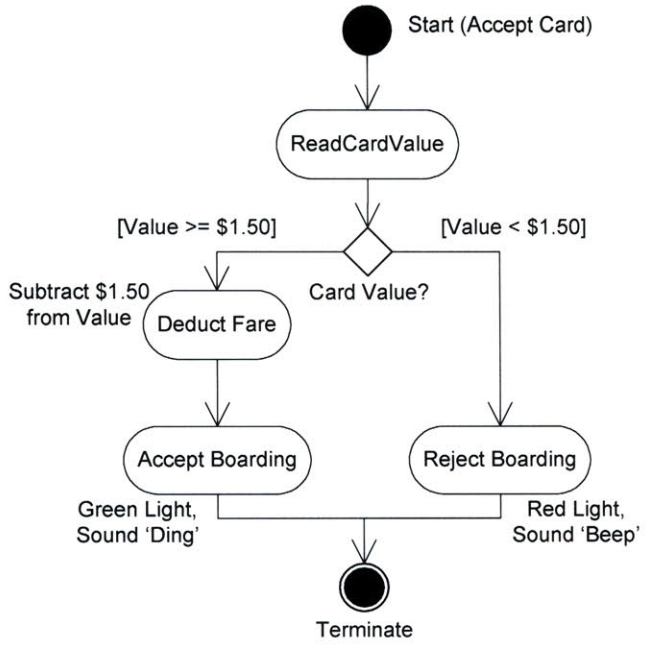


Figure 2.5 – UML activity diagram of a simple fare engine.

In this example, the fare engine resides in a fare box mounted inside each bus near the entrance door. These fare boxes are electro-mechanical devices with a mechanism to ingest, read, write and eject fare cards. The extreme simplicity of this example allows us to express the fare engine as a simple activity diagram. However, in a real-world fare collection system with a more complicated fare structure (no less TfL, which has an extremely complex fare structure), the fare engine is itself a system and will in turn be composed of multiple functional elements.

2.2 *Research Question*

The research question addressed by this thesis is:

> *Is it feasible to develop a fare engine capable of accepting and processing contactless bankcards as a fare instrument on the TfL network while satisfying current and future fare structure and performance requirements?*

Information infrastructure is a key component for TfL's Future Ticketing strategy of bringing bankcards as a fare instrument to the London public transportation network. While contactless bankcard technology is similar to the existing Oyster system in many respects, it also brings new data paradigms, such as a centralized data model, as well as new features and limitations. A fare engine capable of reconciling these differences with the business requirements inherent in TfL's complex fare structure is a critical component in any future contactless bankcard based fare collection system.

2.3 *Investigative Process*

In this thesis we seek to produce a functional design of a fare engine for contactless bankcards. In the course this design process, we will develop solutions for challenges arising from both the characteristics of contactless bankcard technology and the complexity of TfL's fare requirements.

The sequential steps of this investigative process as it relates to the structure of the thesis are summarized below.

1. Identify and formalize the existing TfL fare structure.

2. Formulate a set of system requirements for a contactless bankcards on the basis of the current structure and future needs of TfL.

3. Design a system architecture for a fare engine which satisfies the established requirements.

4. Develop system design details, including data structures and algorithms with a view of achieving performance requirements.

5. Consider issues surrounding the implementation of the system, or a subset as a demonstrative prototype.

2.4 *Development Considerations*

Throughout the process of developing a fare engine, and particularly in the system design steps (steps 3 and 4) we are constantly mindful of the following considerations.

- How well can the system cope with expected features and demands of a future fare structure?

- Can the system handle high transaction volumes, tight transaction time limits and complex journey, tap and fare rules?

- Can off-the-shelf software, such as a commercial rules engine, meet the requirements?

- If not, can custom software be designed and developed to meet the requirements at an acceptable level of complexity?

- What are likely the system and hardware requirements of this software?

- Can existing TfL technology and investments be leveraged in this new fare engine?

3 Current TfL Fare Structure

3.1 *Fare Description*

We define a fare description to be documentation listing fares charged and a corresponding set of rules that apply to these fares. A fare description may be published in any convenient form.

At TfL, fares are published periodically for public information in a series of pamphlets, such as the *TfL Guide to Fares and Tickets* [1 & 2]. For the use of its staff, TfL publishes the same information in greater detail as a manual, the *Staff Guide to Fares*. A simplified explanation of fares is also provided on the TfL website [3]. Together, this body of documentation constitutes the de-facto *fare description* for TfL. It describes the fare products which are available to the public in the form of fare tables, and elucidates the rules surrounding the application of these fares.

The fare description is augmented by an incomplete set of internal design documents for the Oyster system. These documents have proven useful for clarifying lesser known rules and fare services, however they are by no means definitive and can only be interpreted judiciously.

3.1.1 Lateral Consistency

TfL's fare description is thus operationally defined by pamphlets produced for passenger and staff information. As such, the fare description is stated in terms of use cases for these different target audiences. Although a use-case presentation is adequate for the purpose of user information, from an analytical perspective, a property we will call *lateral consistency* is desirable.

To clarify what we mean by 'lateral consistency', let us consider the organization of the 2009 edition of the Guide to Fares and Tickets, an outline of which is attached as Appendix 1. This document is differentiated at the root level into 'Adult', 'Discount', 'Visitor', 'River Rover' and '3 Day Travelcards'. It is not difficult to see that '3 Day Travelcard' does not belong in the root tier, and has presumably been attached there only as an afterthought; this is an example of lateral inconsistency.

As another example, consider the Discount root branch. It is first broken down by ticket type (single or period), and then by mode, and then by discount groups, and last by fare media. Contrast this with 'Adult' fares which are broken down by ticket type and mode, and then immediately by fare media.

The consequence of this discrepancy is that the same attribute, fare medium, is 4 levels deep in the Adult branch but 5 levels deep in the Discount branch.

Note that some single products are available in peak and off-peak formats (e.g. Adult single), while others come in only one variety (Child single). A similar observation is that some discount groups enjoy 1-Day products in both Travelcard and capping formats, while others (such as 16+) can only access 1-Day products in the capping format.

Finally, longer-range period products are available as 'Travelcards' only. Capping does not exist for periods longer than one day. Yet, longer duration Travelcards are frequently loaded onto an Oyster card, something that is not possible to do with a 1-Day Travelcard. The notion of a 'Travelcard' is a blurred one and the term itself is overloaded with multiple meanings.

The above is a partial and informal survey of TfL fare documentation; however it already reveals how the TfL fare description does not clearly indicate the factors that affect fare and how they interrelate.

3.2 *General Fare Structures*

3.2.1 What is a Fare Structure?

Based on the examples above, we recognize that the current articulation of TfL's fare description is difficult to apply systematically. In order to engineer a fare engine for contactless bankcards that assesses fare accurately for trips made by its users, we must first develop a more consistent understanding of how fares work at TfL, and synthesize this understanding into an applicable framework. We call this framework a *fare structure.*

Let a *fare structure* be defined as a representation of fares that can applied systematically and implemented programmatically. We want the fare structure to be a procedure into which we can input known facts about a user's travel, and in turn obtain the price to charge. Examples that would satisfy this requirement include trees, flow diagrams, lookup tables, and multi-dimensional matrices.

After a close reading of documents making up TfL's fare description, one could in fact construct a tailor made flow chart (or software logic) that accurately models the fare description, with all its peculiarities. By our definition so far, this would indeed constitute a legitimate fare structure in that

it can be applied systematically. However, such a piece of logic would contain all of the inconsistencies listed in section 3.1.1, and suffer corresponding drawbacks. For example, a decision tree with the same property represented at different levels would be hard to maintain against even a minor reorganization of fare-impacting properties.

3.2.2 Fare structure Evolution

To refine the concept of a fare structure, let us first pose two fundamental questions relating to what we want to achieve with a fare structure.

- What factors affect the fare?
- How do these factors affect the fare?

To answer these questions, let us lay out some essential nomenclature. This nomenclature is necessary as we find the term *fare structure* easily overloaded with conflicting meanings in multiple contexts.

3.2.2.1 Conceptual Fare Structure

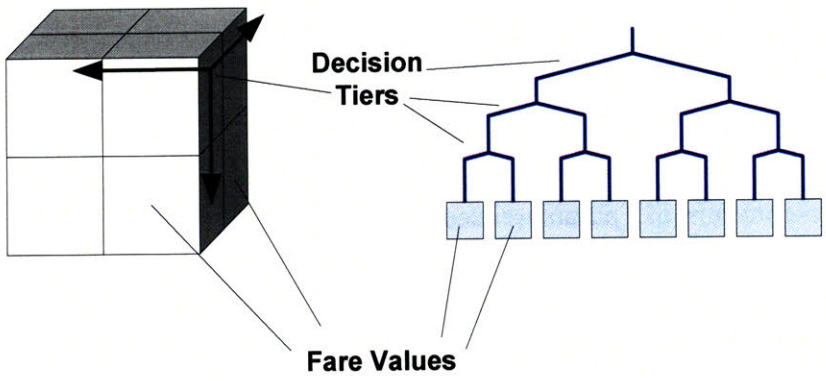


Figure 3.1 – Matrix and full tree representations of a conceptual fare structure.

Previously we stipulated a fare structure to be a procedure for fares that can be systematically applied and programmatically implemented. Now we add an additional stipulation, that this fare structure is a multi-dimensional matrix. We call this a *conceptual fare structure.* Note that *conceptual fare structure* is simply the notion of describing fares in a multi-dimensional matrix. It does not specify what this matrix should look like, how many dimensions there are in the matrix, or what these dimensions should stand for. A conceptual fare structure can be equivalently represented as a matrix

or a full[1] tree, as shown in the figure below. In the matrix representation, the *decision tiers* are the axes of the matrix while the fare values are the cells of the matrix. In the tree representation, the decision tiers are the non-leaf nodes of the tree, while the fare values are the bottom leaf nodes.

3.2.2.2 Parameterized Fare Structure

The conceptual fare structure is extended into a *parameterized fare structure* by specifying what the *decision tiers* are. In the matrix representation, this means defining the number of dimensions there are in the matrix and the names of these dimensions, or what they stand for. Under the full tree representation, this means specifying the number of levels in the tree and the names of each of these levels. We will call the set of names for the tiers of a parameterized fare structure the fare structure's *tier definition.* Contextually, a parameterized fare structure describes what properties affect the fare, but not the specific values these properties may take.

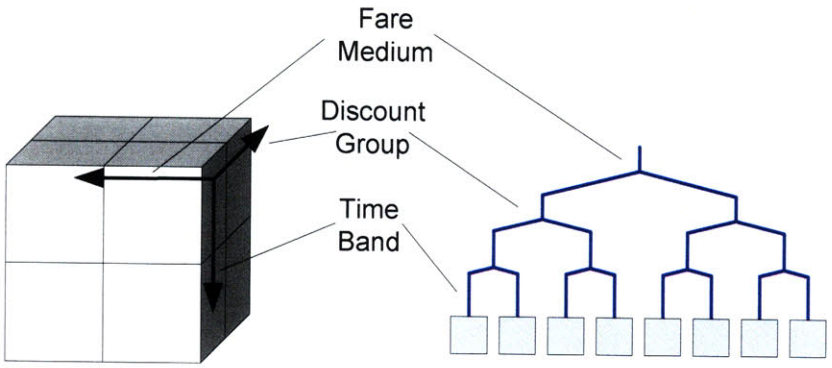


Figure 3.2 – Matrix and full tree representations of a parameterized fare structure.

From now on, the terms *tier* and *dimension* will be used interchangeably. Note that although the matrix structure implies no specific order in which dimensions are evaluated, we may nonetheless specify a preferred customary order based on practical considerations.

3.2.2.3 Domain-defined Fare Structure

We define the set of possible values for each tier of a parameterized fare structure to be the tier's *tier domain.* Furthermore the set of *tier domains* for the an entire parameterized fare structure is the

[1] A full tree is one in all leaf (terminal) nodes are at the same depth and same level. Furthermore, all possible leaf nodes are defined (the bottom level is complete and has no gaps).

structural domain of the parameterized fare structure. This is akin to defining the input domain of a mathematical function. In a tree representation, these are the labels of the branches at each level. The end result of assigning the structural domain of a parameterized fare structure is a *domain-defined fare structure.*

When *fare structure* is spoken of without a qualifying adjective in the remainder of this chapter, a domain-defined fare structure is assumed.

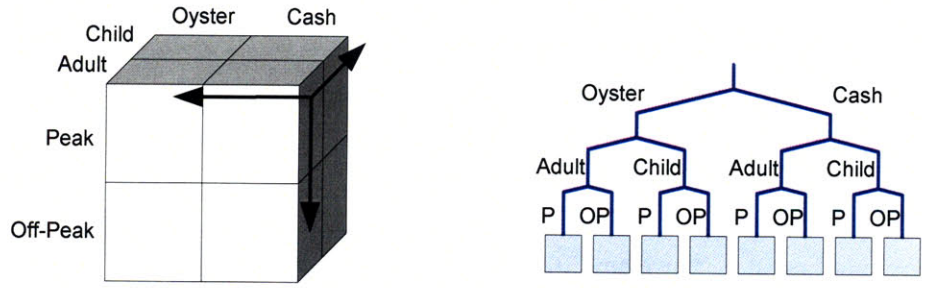


Figure 3.3 – Matrix and full tree representation of a domain-defined fare structure.

3.2.2.4 Initialized Fare Structure

A domain-defined fare structure gives us a lot of information about how fares work; however it is not a complete specification of fares. The actual fare values need to be loaded into the structure. In the matrix representation, this means filling in the body of the matrix. In a full tree representation, this means populating the leaf or terminal nodes with fare values. We call the product of this step an *initialized fare structure*. Now the fare structure is ready for direct application.

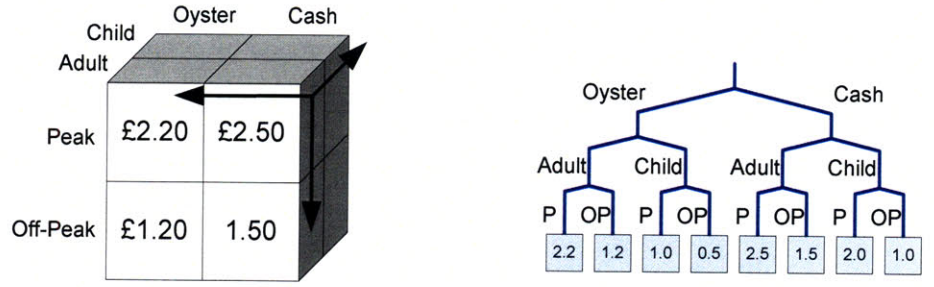


Figure 3.4 – Matrix and full tree representation of an initialized fare structure

3.2.3 *Applying a Fare Structure*

A fully initialized fare structure categorizes fares using a well defined set of parameters. These are parameters that pertain to the properties of the user and the geographic and temporal aspects of the user's journey. In the case of period products[2], the parameters pertain to the nature of the ticket and the geographic and temporal aspects of its validity. Parameters are presented to the fare structure as a unit called a *fare parameter token*. The number of parameters in a token should be consistent with the tier definition as described in section 3.2.2.2. The value of each parameter should be consistent with the structural domain as described in section 3.2.2.3.

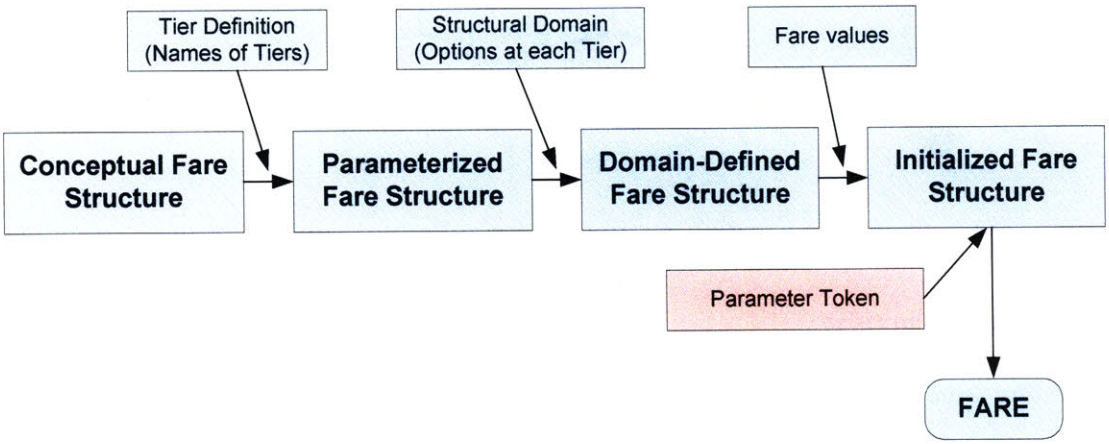


Figure 3.5 – A fare structure is built up through successive definition with agency-specific information.

3.2.4 *Normalization of Fare Variation*

Different fares are charged for different journeys. These fares are chosen from a set of fare values which are set arbitrarily. For example, the fare differential between a ticket for travel between stations A and B and one for travel between stations A and C may be determined by economic and operational factors of the agency. Similarly the amount of discount offered to certain subsections of the population, such as children, the elderly or the unemployed may stem from political considerations. This variation lies outside the scope of the fare structure to explain. From the perspective of the fare structure, the values are set *exogenously*.

[2] Examples of period products include daily, weekly and monthly tickets. The term period product is defined in section 3.5.

As a guiding principle, the goal of a good fare structure is to explain all possible fares using the smallest number of exogenously defined fare values. Consider the extreme case of a fare structure where all fare variation is exogenous - an unmetered gypsy cab where every fare could be different. The fare could vary based on the driver's perception of the passenger's willingness to pay or how much he feels a particular trip is worth making for him at that given instant. In either case, any apparent regularity would merely be coincidental.

Clearly, such a system is unsustainable for public transportation. We can inject order into this structure by stipulating that the fare is a function of the passenger's demographic. Let us call this 'passenger class'. This is a 1-tier definition, indicating that the fare structure can be represented by a 1-dimension matrix. Furthermore we stipulate which classes a passenger must belong to; for example, PassengerClass $\in$ {Adult, Child, Handicapped}. This is our structural domain. Finally, we instantiate the fare structure with fares for each passenger class. Note that we have now moved from one extreme of supplying an infinite number of fare values exogenously (the gypsy cab example), to the other extreme of supplying only 3 possible fares values. A single tier fare structure like this is in fact an accurate fare structure for many transit agencies. However in other cases greater sophistication is required.

3.2.5 Fare Structure Implementation

We note that the fare structure is a logical product which does not imply an implementation. The fare structure can be implemented in many physical forms, for example, as a printed ticket guide (a fare description). It can be implemented as a piece of software logic (a fare engine) inside a smartcard reader or a ticket vending machine. To support contactless bankcards, we will need to implement the fare structure as a fare engine on a central server. An implementation may only support a subset of the fare structure; for example, an Oyster reader only needs to support the part of the fare structure that pertains to Oyster based products.

Furthermore, an implementation of a fare structure need only provide functional equivalence, and does not have to bear any data model resemblance to the fare structure on which it is based. Our fare structure defined as a multi-dimensional matrix may in fact be evaluated in real-time from a smaller set of data. For example, a fare structure may define adult and child fares as two distinct branches, but in practice the child fare could simply be computed as one half of the adult fare, which is stored.

3.2.6 *Limitations of a Matrix Based Fare Structure*

Many intricacies of fare cannot be expressed fully using only the multidimensional structure proposed here. For example, TfL fares between origins and destinations are largely distilled into a system of fare zones. Yet there exist many exceptions to the zonal system that prevents the straightforward use of a zonal origin-destination (OD) matrix. Likewise, out-of-station interchange and capping are peculiarities that defy framing into a matrix. These features can only be described algorithmically.

In this chapter we will explore how fare matrices can be augmented with helper logic and reference tables in order to convey the full complexity of the TfL fare structure.

3.3 *Organization of a TfL Fare Structure*

Based on the above discussion of the properties of fare structures, we now construct a fare structure for TfL that is consistent with its fare description.

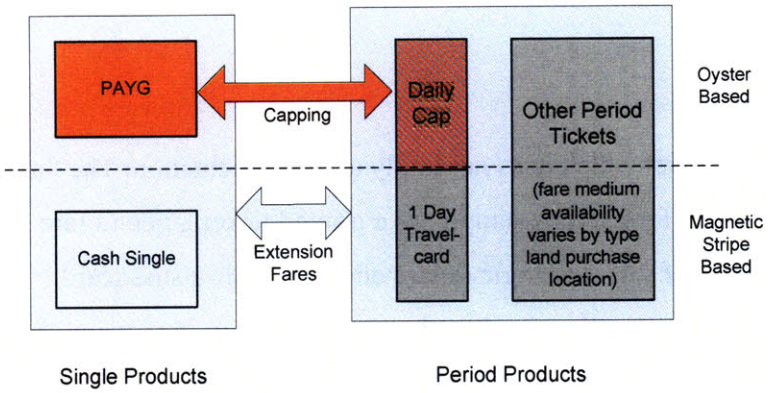


Figure 3.6 – TfL fare structure overview. The two product spheres are shown.

3.3.1 *Product Spheres*

We define a *product sphere* to be a set of products or services that can be organized into one multi-dimensional matrix. Using this definition, we can divide the TfL fare structure into two distinct product spheres. These are *single products* and *period products. Single products* encompass Oyster Pay-As-You-Go (PAYG), as well as cash single (one-way) fares. These are fares which are charged on a trip-by-trip basis. On the other hand, *period products* represent those products which give unlimited travel under a set of given restrictions. Separation of the fare structure into two spheres is

necessary because single products and period products are fundamentally different enough that they cannot be selected using the same set of parameters.

3.3.2 Hybrid and Bridging Features

The Oyster daily cap is a feature that does not fit easily into this dichotomous structure. On the one hand, the daily cap is an integral part of Oyster PAYG, which resides in the single products sphere. On the other hand, it is functionally similar to a period product and has a pricing structure analogous to that of a 1-Day Travelcard. For the latter reason we have classified the daily cap into the period sphere, recognizing however that it is really a hybrid. This is illustrated in Figure 3.6. In the remainder of this document the daily cap will be discussed as a period product.

The two main bridging mechanisms of Figure 3.6 are capping and extension fares. By a bridging mechanism it is meant that these features provide a linkage between the two product spheres. Capping, for example, is the mechanism that offers the period benefits of a daily cap to users of PAYG. This feature is available only for PAYG (not cash singles) and applies only on a daily basis; hence it is represented as an arrow connecting the internal boxes of PAYG and Daily Cap in the diagram. The rules of capping will be discussed in greater detail.

Extension fares, the second bridging mechanism, allow holders of period products to travel outside their zonal validity by purchasing a single product to extend their preexisting ticket. It is another mechanism that connects the two product spheres. In this case, because extension fares can be purchased both by holders of paper tickets, as well as by users of Oyster (in which case, it is applied automatically), the arrow representing the feature bridges the outer boxes encapsulating an entire sphere.

3.4 *Single Products*

The payment of single or one-way fares via a prepaid balance deposited on an Oyster card is known as Pay-As-You-Go (PAYG). Individual trips can also be paid for via magnetic stripe based "cash singles". As the fare structures for these two payment methods are very similar, they are considered under the same umbrella sphere of *single products*.

3.4.1 Tier Definition and Structural Domain

One possible tier definition for the *single products* sphere is provided in Figure 3.7. The tiers that we have assigned are fare medium, discount group, time band, origin station, and destination station. Note that the two bottom levels of this tree are an origin-destination (OD) matrix of the TfL network, consisting of approximately 307 x 307 = 94249 elements.

The expected size of the resulting matrix is 307 (origins) x 307 (destinations) x 2 (fare media) x 2 (time bands) x 6 (discounts), or 2.26 million rows

Although legitimate, this fare structure is unwieldy. Expressing origins and destinations as two independent dimensions does not explain at all how OD fares are assigned. Recall the guiding principle of explaining fare variation with the fewest exogenously defined fare values. We find that OD fares are not arbitrarily assigned, but are designated on the basis of certain properties of the origin and destination stations involved.

In the case of TfL, London has a system of concentric fare zones which is overlaid on the system map. One could further normalize the fare structure by collapsing the dimensions for origin and destination stations into origin and destination zones. Such a matrix would then be supplemented by a reference table that provides a mapping of stations onto one of the nine zones. Let us consider the size of the resulting matrix.

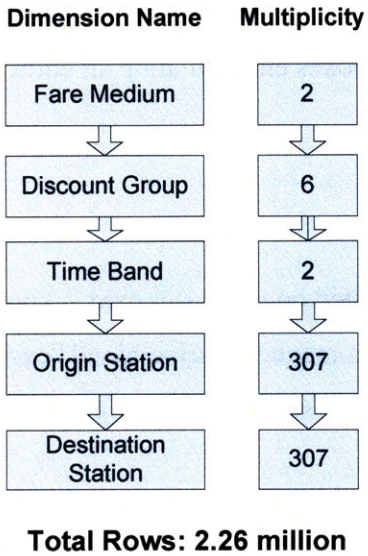


Figure 3.7 – Possible fare matrix for single products. This matrix contains an OD matrix of the TfL network.

2 (fare media) x 2 (time bands) x 6 (discounts) x 9 (origin zones) x 9 (destination zones) = 1944 rows

This approach indeed results in a more manageable matrix, yet it is inadequate for two reasons. First, according to the TfL fare description, fare is not simply a function of the origin and destination zones, but also of the zones travelled on the most likely route taken. Furthermore, there are special cases that cannot be resolved in terms of zones travelled alone. Examples of these exceptions include the special fare incurred by trips involving Watford Junction and the special day-ticket for DLR travel within zones 2-3 only.

In order to accommodate these problems, we will adopt a two part solution. This solution is based on a four-dimensional matrix, which we call the *master matrix*. Three of these dimensions represent the three *independent tiers* - fare medium, time band and discounts. These independent tiers are relatively straightforward. However, recognizing that the effect of the origin and destination on fare cannot be captured purely within a matrix framework, we separate it from the master matrix altogether, by introducing the *charge code*.

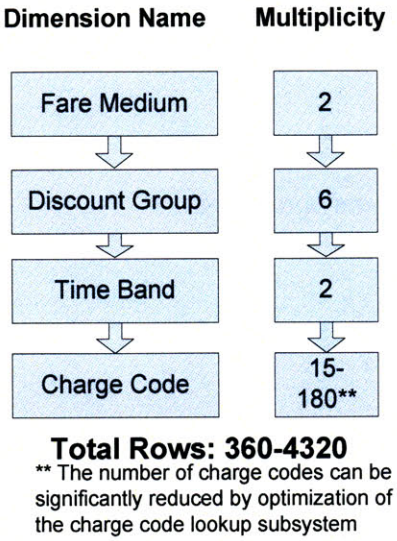


Figure 3.8 – Master fare matrix. Three independent dimensions with charge code removing effect of OD variation.

The charge code is a foreign key[3], or a value that captures the variation of fares as a function of OD

[3] In the context of relational databases, a foreign key links rows of one table to uniquely defined primary keys in a second reference table. For example, employees in a table of employees may each be assigned a department ID that references a table of all departments. The department ID is a foreign key in the table of employees.

without explicitly enumerating the full OD matrix. It forms the fourth tier of the master matrix.

Figure 3.8 shows the four dimensions of the master fare matrix as well as the preferred order of evaluation. We will call the first three tiers the *independent tiers*. Each combination of parameters from the three independent tiers is called an *independent combination*. An independent combination identifies a family of fares values that differ only by charge code. The charge code is not an independent tier as it has dependence on the journey route. The charge code will be addressed below in section 3.4.2.

3.4.1.1 Fare Medium

TfL currently supports two fare media, *cash* and *Oyster*. Under the TfL fare description, cash fares are significantly higher than Oyster fares in order to discourage the purchase of paper tickets and to lower cash handling costs. Fare medium is placed at the top of the hierarchy as the choice of fare medium fundamentally controls the scope of fare structure implementation. For example, an Oyster reader would only implement the Oyster branch of the fare structure, while, say, a magnetic stripe ticket machine would implement only the cash branch.

3.4.1.2 Discount Group

At this time, TfL supports six discount groups in the context of one-way products:

1. Adult (no discount)
2. Child (5-15)
3. 16+
4. New Deal
5. Adult Privilege Rate
6. Child (5-15) Privilege Rate

New Deal refers to a social welfare benefits program in the UK. 16+ is a concession fare for eligible secondary and tertiary students. Privilege rates apply to beneficiaries of employees, former employees and other union sanctioned persons.

3.4.1.3 Time Band

The start time of a journey determines whether that journey falls within the *Peak* or *Off-peak* time

band. These time bands are currently defined as follows

Peak: 0630 – 0930, 1600-1900 Weekdays, excluding public holidays

Off-peak: All other times

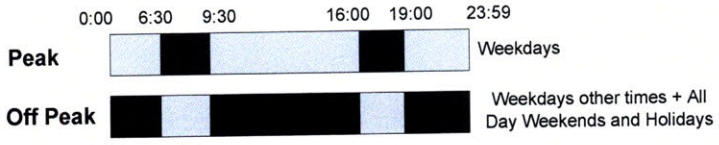


Figure 3.9 – PAYG time bands

According to the TfL fare description, these time bands for single fares are different from those defined in the context of period tickets. This is a confusing and easily missed distinction.

3.4.2 Charge Code

Previously we recognized that the relationship between the origin and destination stations and fare may be described by some mechanism beyond an exhaustive OD matrix. Furthermore we found that the relationship cannot be explained on the basis of the zones of the origin and destination stations alone. These complications have led us to separate this sub-problem from the master matrix through the use of charge codes.

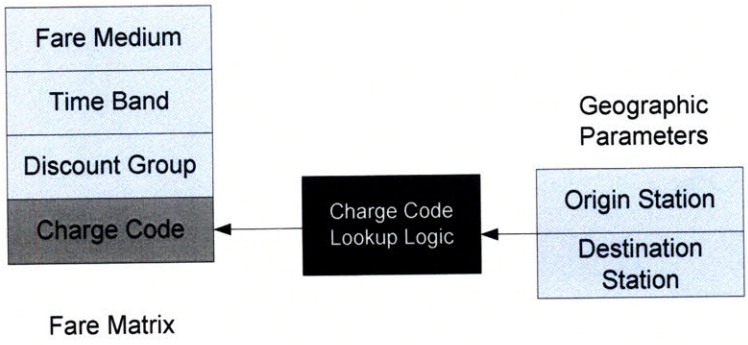


Figure 3.10 – The charge code is a strict function of the origin and destination stations.

The charge code is a foreign key that encapsulates the effect the OD stations have on the fare charged. It is connected to the origin and destination stations via a procedure that performs this mapping. Below we'll investigate the functionality of this procedure.

3.4.2.1 TfL Zones

TfL has 9 zones arranged concentrically around central London. The innermost zone is zone 1, being roughly coterminous with the common definition of 'Central London', while the outermost zone is zone 9, covering the extremities of the Metropolitan Line. The bulk of TfL's services lie within zones 1 to 6.

The inner and outer zones, which determine the far, of a given OD pair are determined by the *fare path* of that journey. If we go from an origin to a destination following an assumed *fare path*, enumerating each station along the path and recording the zone that it lies in, the lowest value recorded becomes the inner zone. The outer zone is the highest value thus recorded. One subtlety is that some stations lie on the border of two zones and can take the value of either zone based on which choice will minimize the zonal span for the journey concerned.

Ambiguity in this procedure arises from the definition of *fare paths*. TfL does not currently have systematic criteria for determining fare paths other than by manual decision. However, Maciejewski introduced a methodology for the automatic computation of fare paths by means of a shortest path algorithm [4].

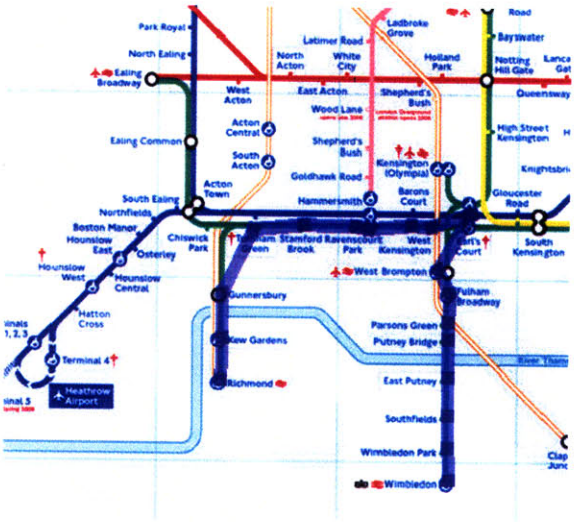


Figure 3.11 – Example of a fare path/zones travelled calculation. Richmond(zone 4) to Wimbledon(zone 3). Inner zone 2, outer zone 4.

3.4.2.2 Zonal Pairs

Let us assume that we are provided with the inner and outer zones used for a given journey. We will

call this unordered pair {inner zone, outer zone} the *zonal pair* for that trip (It bears repeating that these are not the origin and destination zones). Each such zonal pair corresponds to a charge code. If we enumerate all possible zonal pairs, in the combinatorial case there are $\sum_1^n n$, or $\frac{n(n+1)}{2}$ possible zonal pairs. Substituting in actual numbers, there are 21 zonal pairs if we consider only the core 6 zones where the bulk of TfL services are located. If the full 9 zones are considered, there are 45 zonal pairs, leading to the same number of unique charge codes.

Zonal pairs are undirected sets because fares are same in either direction (ignoring the Euston Overground case below). Ordered zonal pairs can be used as a basis for charge codes (doubling the number of charge codes) if support for asymmetrical fares is desired.

3.4.2.3 Special Cases

The ability to deal with special cases that do not fit neatly into a numeric zonal structure is the core reason behind the use of charge codes. At the time of writing, three special cases are documented in the TfL fare description. Special cases are inevitably subject to change and it is important that the mechanism used can accommodate future changes to these special cases.

- **DLR** – DLR only travel within zones 2 and 3 has a discounted cash fare: DLR zones 2-3 have a cash fare of £1.60, contrasting with a standard zones 2-3 cash fare of £3.20.
- **Watford Junction** - Journeys to or from Watford Junction (zone 9) are charged higher peak and off-peak fares compared the corresponding zone x-9 fares.
- **Euston Overground** – Overground journeys between Euston (zone 1) and Watford Junction (and all intermediate stops) are charged higher fares compared to corresponding zone 1-x fares. Furthermore the documentation notes that peak fares in this corridor are applied directionally. Peak fares are to be applied for Overground trips toward Euston in the morning peak and away from Euston in the evening peak only. However the documentation is contradictory in describing what fare would be charged in the reverse commute direction during the peak. Compounding the confusion, inspection of an internal fare table shows no directionality in these special peak Euston fares. These problems are symptomatic of the difficulties encountered in interpreting a fare description and reconciling parts of a fare description with each other. We will assume no directionality in Euston fares.

3.4.2.4 Charge Code Lookup

The system outlined in Figure 3.12 is a possible implementation of charge code lookup logic. This mechanism is suggested here to illustrate the operation of charge codes. This is not intended to be a reflection of the actual implementation of charge codes in Oyster or in a future fare engine.

Our charge code logic can be broken down into two components. The first component is the zonal pair resolver. This resolves a pair of OD stations into a zonal pair on the basis of the OD zones, using either the Maciejewski method or a lookup table.

The second component detects exceptions that lie outside of the ability of a pure zonal system to describe. This component triggers a set of *charge code tables* based on qualifying origin and/or destination station inputs. Each such triggering case is known as a *charge code case.*

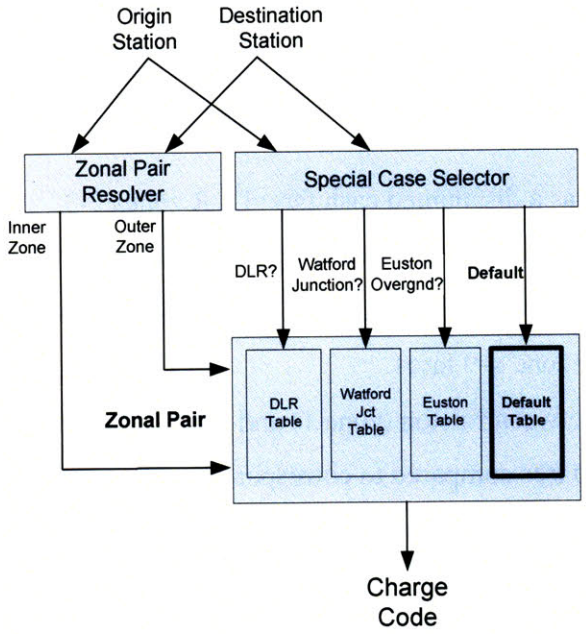


Figure 3.12 – Example charge code logic consisting of a zonal pair resolver and a special case selector.

For our discussion we will assume 4 charge code cases - 3 special charge code cases and one default charge code case. Each charge code case is associated with a charge code table that contains an exhaustive enumeration of zonal pair combinations, as described in section 3.4.2.2. For 9 zones there are 45 pairs. Multiplying 45 by 4 charge code cases results in 180 unique charge codes. There is one fare value initialized for each charge code for each of the 24 combinations of the 3 independent tiers. The total number of fares values initialized is therefore 24 x 180 = 4320. This is the upper bound

quoted in Figure 3.8. In enumerating charge codes as a matrix of zonal OD pairs and special cases we have assumed that all special cases follow the default zone structure. It is possible that one or more of the special cases does not follow the default zone structure.

3.4.2.5 Charge code Optimizations

Initializing 4320 fare values seems rather wasteful. We can look for strategies to reduce the number of charge codes used, and hence the number of fare values exogenously initialized.

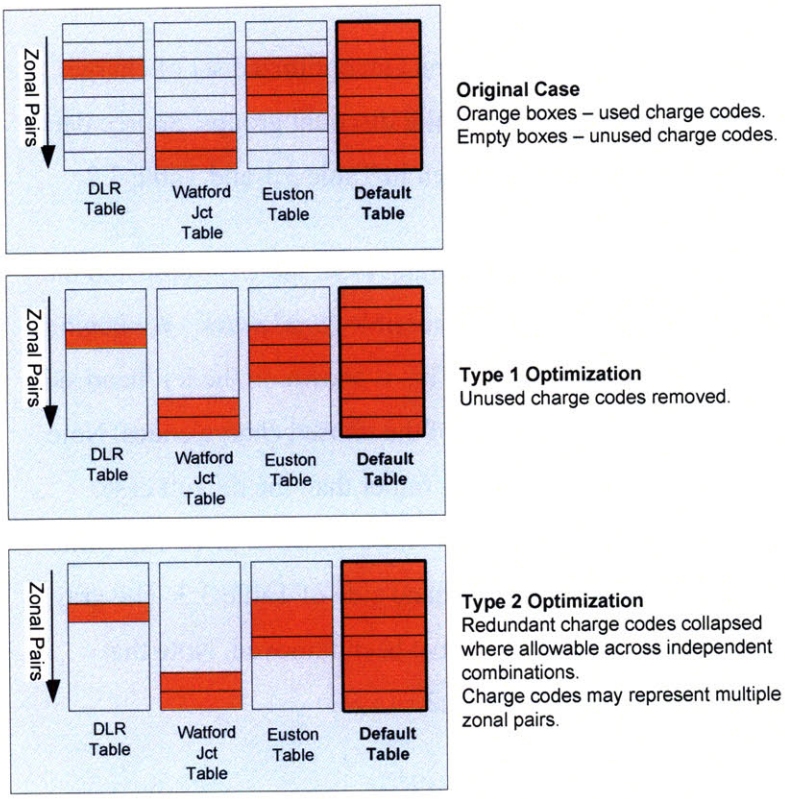


Figure 3.13 – Charge code reduction strategies

3.4.2.5.1 Type 1 Optimization

We can individualize the charge code tables for each *charge code case* by removing unused charge codes. Observe that some charge code cases are applicable only to certain zonal pairs. For example, a journey that triggers the Watford Junction case must originate from or end in zone 7. All zonal ranges that do not involve zone 7 can be removed from the Watford Junction charge code table. By assigning charge codes only to zonal pairs that are relevant for the given case, the total number of charge codes can be reduced. This will be demonstrated in an example below.

3.4.2.5.2 Type 2 Optimization

Up to now, we have assumed that charge codes correspond to one zonal pair each. However, by relaxing this requirement, charge codes can also be collapsed to represent multiple zonal pairs. The extent to which we can collapse charge codes is determined by our *basis zonal set*, or the set of sets of zonal pairs that represent the lowest common denominator for expressing fares across all 24 *independent combinations*[4]. This is also demonstrated below.

3.4.2.6 Charge Code Example

In order to explain the concepts described in this section, we construct an optimized set of charge codes as an example. For this exercise we only use the adult and child discount groups. Source data for this exercise is excerpted from the TfL fare description and given in Table 3.1 and Table 3.2.

In this example, we consider only three *charge code cases*[5]: the default case, the DLR case and the Euston case. Restricting ourselves to zones 1-6 only, there are 21 possible *zonal pairs*[6]. We begin by creating three charge code tables with 21 entries (21 zonal pairs). This is shown on the left hand side of Table 3.3. Now we apply Type 1 optimization by removing all of the unused charge codes. Note that unused charge codes are usually found within the special cases rather than the default case. Finally we apply Type 2 optimization by collapsing zonal pairs that share the same price value into the same shared charge code. The end result is shown on the right hand side of Table 3.3. The gray shaded cells in the table represent unneeded charge codes which have been removed. Note that charge codes 46 to 50 represent the special Euston fares in Table 3.2.

Recognizing that different *independent combinations* have to fit within the same common lookup structure, we introduce the concept of a *basis charge code set.* The *basis charge code set* is the set of charge codes that form the 'least common denominator' across all independent combinations, including all special cases for each independent combination.

---

[4] Recall an independent combination is a family of fares differentiated only by charge code. A single product independent combination is parameterized by fare medium, discount group and time band.

[5] For a definition of *charge code case,* see section 3.4.2.3.

[6] For a definition of *zonal pair*, see section 3.4.2.2.

Oyster single fare	Cash single fare
£1.00	£2.00

	Oyster single fare		Cash single fare
	Peak	Off-Peak	
Fares including travel in Zone 1			
Zone 1 only	£1.60	£1.60	£4.00
Zones 1-2	£2.20	£1.60	£4.00
Zones 1-3	£2.70	£2.20	£4.00
Zones 1-4	£2.80	£2.20	£4.00
Zones 1-5	£3.70	£2.20	£4.00
Zones 1-6	£3.80	£2.20	£4.00
Fares not including travel in Zone 1			
One or Two Zones: Zones 2, 3, 4, 5, 6 or 2-3, 3-4, 4-5 or 5-6	£1.10	£1.10	£3.20
Three, Four or Five Zones: Zones 2-4, 3-5, 4- 6, 2-5, 3-6 or 2-6	£2.00	£1.10	£3.20
DLR only in Zones 2-3	£1.10	£1.10	£1.60

Child (11-15 years)

	Oyster single fare	Cash single fare
Fares including travel in Zone 1		
Zones 1-6	£0.55	£2.00
Fares not including travel in Zone 1		
Zones 2-6	£0.55	£1.60
DLR only in Zones 2-3	£0.55	£0.80

Table 3.1 – Fare description excerpt for single fares. Left(a) : Adult. Right (b): Child. Zones 1-6 only. Watford Jct. exceptions not shown. Note that there are no bus-only products for children. Other discount groups (16+, New Deal, Privilege Rates) have been omitted. Children travel free on buses [1].

Origin-Destination	Adult PAYG Peak	Adult PAYG Off-Peak	Adult Cash All-Day	Child PAYG All-Day	Child Cash All-Day
Euston-Zone 1 (WJB)	-	-	-	-	-
Euston-Zone 2 (WJB)	£2.25	£1.65	£3.25	£1.65	£0.55
Euston-Zone 3 (WJB)	£2.75	£2.25	£4.05	£2.05	£0.55
Euston-Zone 4 (WJB)	£2.85	£2.25	£4.05		
Euston-Zone 5 (WJB)	£3.25	£2.25	£4.05		
Euston-Zone 6 (WJB)	£3.55	£2.25	£4.05		

Table 3.2 – Fares from Euston on the Watford Junction Branch. In the actual TfL fare description these Euston fares deviate from the standard zone 1-n fares in only one case (Euston-Zone 2, Adult Cash All-day). However for sake of illustration we will change the Euston fares so that they are all clearly different from the standard zones 1-n fares (by 5 pence).

Zonal Pair	Default case	DLR case	Euston case
1-1	1	23	45
1-2	2	24	46
1-3	3	25	47
1-4	4	26	48
1-5	5	27	49
1-6	6	28	50
2-2	7	29	51
3-3	8	30	52
4-4	9	31	53
5-5	10	32	54
6-6	11	33	55
2-3	12	34	56
3-4	13	35	57
4-5	14	36	58
5-6	15	37	59
2-4	16	38	60
3-5	17	39	61
4-6	18	40	62
2-5	19	41	63
3-6	20	42	64
2-6	21	43	65
0-0 (Bus)	22	44	66

Zonal Pair	Default case	DLR case	Euston case
1-1	1		
1-2	2		46
1-3	3		47
1-4	4		48
1-5	5		49
1-6	6		50
2-2	7		
3-3			
4-4			
5-5			
6-6			
2-3		34	
3-4			
4-5			
5-6			
2-4	16		
3-5			
4-6			
2-5			
3-6			
2-6			
0-0 (Bus)	22		

Table 3.3 – Charge code optimization process. Left (a): Type 1 Optimization; Unused charge codes are removed. Right (b): Type 2 Optimization; charge codes assigned to multiple zonal pairs.

For example, [Oyster, Adult, Off-Peak] and [Cash, Child, All-day] are two independent combinations. [Oyster, Adult, Off-Peak] can be expressed in a table that looks like Table 3.3b. [Cash, Child, All-day] can also be expressed in a table that looks like Table 3.3b. When we require a basis charge code set for these two independent combinations we are saying that we want these two tables to be identical and have the same charge codes.

When we apply charge code optimizations we must bear this requirement in mind. Doing so is trivial in this example because the following independent combinations [<any fare medium>, Adult, <any time band>] can be expressed using the same charge codes. Furthermore, simply by finding an optimized charge code set for [<any fare medium>, Adult, <any time band>], we would have found

the basis charge code set for [<any fare medium>, Child, <any time band>]. One may verify this by inspecting Table 3.1 and Table 3.2.

The first row of Table 3.1b (Child zones 1-6) is satisfied by charge codes 1-6, while the second row (Child zones 2-6) is satisfied by charge codes 7 and 16. But imagine if the Table 3.1b contained a row for Child zones 2-3 - we would then be prohibited from spreading charge code 7 as broadly as shown in Table 3.3b.

This example will be referenced below when we enumerate the master fare matrix[7] in section 3.4.5.

3.4.2.7 Charge Code Summary

To recap, the process of mapping of origin and destination stations to a charge code can be broken down into the following steps.

1. Resolving the zonal pair ({inner zone, outer zone}) from origin and destination stations.
2. Detecting special cases and identifying which special case to use.
3. Selecting the correct charge code based on the zonal pair and the independent combination. E.g., [Oyster, Adult, Peak] identifies one fare, [Cash, Adult, All-Day] identifies another.

The number of charge codes can be reduced through removing unused charge codes and collapsing multiple zonal pairs into the same code. Either type of optimization requires us to implement custom logic, in order to select the correct charge code for a given zonal pair. The cost of this additional complexity must be weighed against the benefits of a lighter weight master matrix. In the CLBC fare engine we will opt for a more general solution for fare resolution over the use of charge codes.

3.4.3 Bus Journeys

Under PAYG, the same flat fare is levied for any bus trip regardless of the distance of the journey, location of boarding or time of day; however, fares do vary by passenger type and fare media. Although we have not considered bus journeys in the above example, they can be implemented as just another charge code. We may represent bus journeys as an additional row in Table 3.5 and provide a charge code for it under the column for the default charge code case (charge code 22).

[7] The *master fare matrix* is where fare values are stored. See 3.4.1.

Bus fare variation due to different fare media, discount groups and time bands is naturally accounted for by the independent tiers of the master fare matrix.

3.4.4 Matrix Utilization and Treatment of Null Fields

Table 3.4 illustrates the utilization of the three independent tiers as documented by the current TfL fare description. At the time of writing, cash fares are not differentiated by time band. There is only one cash fare with respect to time and this fare applies all day. However, we note that this fact stems purely from the current fare description, and potentially the limitations of the magnetic stripe ticketing system in use. There is no fundamental reason why off-peak cash singles cannot be implemented. We therefore assume support for this hypothetical product in our TfL fare structure, which treats time band and fare medium as independent axes. To maintain consistency with time-band differentiated Oyster PAYG fares we have chosen to use the 'peak' time band to represent and store values of 'all day' cash fares. If cash fares were to become time-band differentiated, it would acquire the same time bands as defined for Oyster PAYG.

Discount Group	Oyster Peak	Oyster Off-Peak	Cash Peak*	Cash Off-Peak
Adult	Yes	Yes	Yes	
Child (5-15)	Yes		Yes	
16+	Yes	Yes	Yes	
New Deal	Yes	Yes	Yes	
Adult Privileged	Yes	Yes	Yes	
Child Privileged	Yes	Yes	Yes	

Table 3.4 – Population of the independent tiers of the fare matrix. Unavailable products expressed as empty cells (null values)
*** There is only one cash fare and this is effectively an all-day cash fare.**

In the table above we have represented unavailable products as null values. Representing undefined products as null fields in the fare structure is not our only option. Alternatively, null fields may be filled with the fare from the nearest corresponding product that we can defer to. This is shown in Table 3.5, which has the same format as Table 3.4 except that null fields are now replaced by deference indicators (shown in orange).

Bear in mind that each cell in the table represents an *independent combination*[8]. Each independent combination has associated with it a full set of fare values (there are as many fare values in each independent combination as there are charge codes). If we looking at a tree representation of the master fare matrix, we are not simply cross-referencing tiers, but rather copying over individual fare values at the leaf level. The structure remains a full tree. In other words, fares are cloned and not merely symbolically linked[9].

An automated fare engine is an example of where this treatment would be relevant. In this case, the fare engine is given a *fare parameter token*[10] that describes the properties of the user and the trip, and must automatically choose the appropriate fare where it is available, or defer to the correct next-best product when no fare product is defined for the given parameters.

Discount Group	Oyster Peak	Oyster Off-Peak	Cash Peak*	Cash Off-Peak
Adult	Yes	Yes	Yes	Adult Cash Peak
Child (5-15)	Yes	Child Oyster Peak	Yes	Child Cash Peak
16+	Yes	Yes	Yes	16+ Cash Peak
New Deal	Yes	Yes	Yes	New Deal Cash Peak
Adult Privileged	Yes	Yes	Yes	Adult Priv. Cash Peak
Child Privileged	Yes	Yes	Yes	Child Priv. Cash Peak

Table 3.5 – Fares for unavailable products replaced by fares deferred to.

3.4.5 Example Master Matrix

We will conclude this discussion of a fare structure for single products by presenting a master matrix based on the limited example given in section 3.4.2.6. Recall that we have 3 independent tiers. The structural domain is:

$$FareMedia = \{Cash, Oyster\}$$

$$TimeBands = \{Peak, OffPeak\}$$

$$DiscountGroups = \{Adult, Child, \sout{16+, NewDeal, AdultPriv, ChildPriv}\}$$

[8] The term *Independent combination* is defined at the end of section 3.4.1.

[9] Symbolic linking means data is not duplicated, but a pointer is used to forward the seeker to the original copy.

[10] See section 3.2.3.

Furthermore, we have defined a fourth tier, ChargeCodes, which represents a mapping defined as a function of the origin station and the destination station of a journey. Each charge code is an integer value which represents a unique pricing level within TfL's zonal price structure.

For the purpose of this example we will only consider two discount groups for brevity's sake - Adult and Child. Together we have a total of 2 x 2 x 2, or 8 independent combinations. In section 3.4.2.6 we arrived at a charge code scheme that has 15 different charge codes (including one for bus). We will therefore need to fill 8 x 15 = 120 cells exogenously in our master fare table.

Fare Media	**Cash**				**Oyster**			
Disc. Grp	**Adult**		**Child**		**Adult**		**Child**	
Charge Code	**Peak(AD)**	**Off-Peak**	**Peak(AD)**	**Off-Peak**	**Peak**	**Off-Peak**	**Peak(AD)**	**Off-Peak**
1	4.00		2.00		1.60	1.60	0.55	←0.55
2	4.00		2.00		2.20	1.60	0.55	←0.55
3	4.00		2.00		2.70	2.20	0.55	←0.55
4	4.00		2.00		2.80	2.20	0.55	←0.55
5	4.00		2.00		3.70	2.20	0.55	←0.55
6	4.00		2.00		3.80	2.20	0.55	←0.55
7	3.20		1.60		1.10	1.10	0.55	←0.55
16	3.20		1.60		2.00	1.10	0.55	←0.55
22(Bus)	2.00				1.00	←1.00	0.00	←0.00
34(DLR)	1.60		0.80		1.10	1.10	0.55	←0.55
46(Euston)	3.20		1.60		2.20	1.60	0.55	←0.55
47(Euston)	4.05		2.00		2.70	2.20	0.55	←0.55
48(Euston)	4.05		2.00		2.80	2.20	0.55	←0.55
49(Euston)	4.05		2.00		3.70	2.20	0.55	←0.55
50(Euston)	4.05		2.00		3.80	2.20	0.55	←0.55

Table 3.6 – Example master fare matrix following on from charge codes constructed in 3.4.2.6. Rows in this table represent charge codes, columns represent independent combinations. AD is short for All-Day.

One should verify that Table 2.6 indeed follows from the framework described thus far. For illustrative purposes we have used both approaches described for treating unavailable product options. On the cash side, we have left unavailable values blank, simply as null fields. On the Oyster side, we have populated the cells to enable automatic deferral to the next-best product.

3.5 *Period Products*

Period products are the second umbrella sphere in the TfL fare structure. Contained within this sphere are two classes of sub-products.

The first class is known as *period tickets*, or *tickets* in short. Period tickets give users unlimited use of a certain part of the system, subject to conditions attached to the ticket purchased. Period tickets are also known under the following names:

- **Travelcard** – Alternative name for any period ticket. Travelcards can be purchased as a magnetic stripe ticket or loaded on an Oyster card. We will consider this term synonymous with period ticket.
- **Day Ticket** – Refers to the 1 day or 3 day products.
- **Season Ticket** – A term usually used in conjunction with products with a validity period 7 days or longer.

Both day tickets and season tickets are a form of Travelcard.

The second class of products that make up the sphere of period products are *caps.* Caps will be explained in section 3.7. Caps are generally not considered a type of period ticket; however they are closely related. Capping prices have a corresponding, analogous day-ticket price. Due to this relationship, we will consider caps in the same context as period tickets.

3.5.1 Tier Definition and Structural Arguments

The tiers illustrated in Figure 3.14 and their domains are described below.

3.5.1.1 Application Context

Compared to the *single* tier definitions, which were discussed in section 3.4.1, we have substituted *application context* for *fare medium* as the initial decision tier. As it would be inaccurate to refer to a cap as a type of fare medium, we have created a new term to describe the way a period product is applied. This is the *application context*. The application context expresses whether a period product is applied as a ticket which has been pre-purchased prior to travel, or whether it is applied automatically as an Oyster PAYG cap. The application context can either take the value of *ticket* or *cap.*

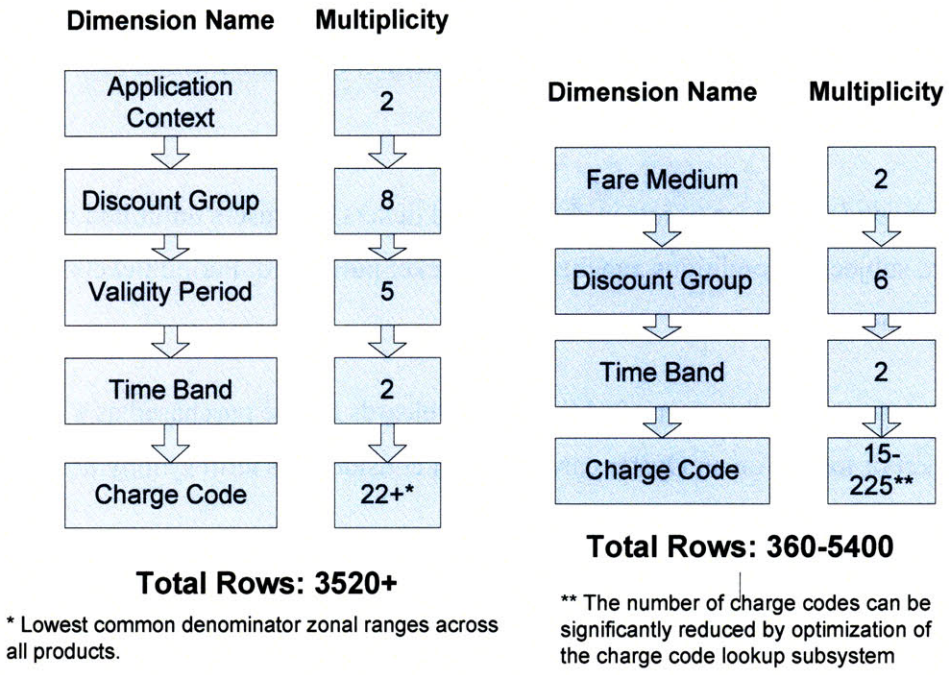


Figure 3.14 – Tier definition for period products is on the left. The tier definition for single products is provided on the right for reference. Note that *application context* is the period product analog of *fare medium* for single products.

As Table 3.7 shows, some period tickets must be purchased in paper form, while others can only be loaded onto an Oyster Card. TfL sells 1-Day Travelcards in paper form only. However, Oyster PAYG users can also enjoy the benefits of daily period travel owing to the policy of Oyster capping. To complicate the picture, the daily capping prices differ from the corresponding 1-Day Travelcard prices. This is the reason we have introduced the *application context* decision tier.

Longer period products are available in either Oyster or in magnetic stripe form, but not both at the same time when purchased from TfL. Capping is currently not available for validity periods longer than one day. For one-day validity, capping is available for all discount groups and on both an all-day and off-peak basis.

We accept that dedicating one whole decision tier to *application context* would result in large swathes of unused cells in the master matrix where capping is not available. On the other hand, this structure gives us the room to introduce longer-interval caps.

We have restricted our definition of application context to only caps and tickets because current TfL fare description does not differentiate paper and Oyster tickets by price. However, the definition of

application context can be extended to encompass fare medium, if it becomes necessary. In this case, accepted values for application context could be {Oyster ticket, Mag stripe ticket, Oyster cap}.

Validity Period	Oyster	Magnetic Stripe
1-Day	No* (but capping applied automatically)	Yes
3-Day	No	Yes
Weekly	Yes	No** (Sold at NR stations only)
Monthly	Yes	No** (Sold at NR stations only)
Annual	Yes	No** (Sold at NR stations only)

Table 3.7 – Period product availability across fare media.

3.5.1.2 Discount Groups

In the context of period products there are 8 discount groups, compared to 6 for single products. Although we have elected to keep them separate, there is also no reason why the tier domains for these instances of *discount groups* cannot be reconciled and unified. Possible values for this tier are:

1. Adult (no discount)
2. Child (5-15)
3. 16+
4. 18+ **(not in PAYG)**
5. Railcard Holders **(not in PAYG)**
6. New Deal
7. Adult Privilege Rate
8. Child (5-15) Privilege Rate

3.5.1.3 Validity Periods

Each period product has a specific period of validity. This dimension segments products by validity period. The tier domain for this dimension is:

1. 1 Day
2. 3 Day
3. Weekly
4. Monthly*

5. Annual

* Monthly tickets can be issued for any number of months, from one up to a limit of 11. Common denominators are 3 and 6 months. However this peculiarity can be ignored as the same monthly rate applies regardless of the number of months purchased. There is no discount beyond the convenience of not having to renew a ticket. In effect multi-month tickets can be considered single monthly tickets purchased back-to-back.

3.5.1.4 Time Bands

The definitions for Oyster PAYG and Travelcard time bands are not the same. Note how PAYG observes a PM peak, but period products do not, and also that the Travelcard AM peak begins at 4:30AM while the PAYG peak begins at 6:30AM. Even more confusingly, Oyster capping, although it is applied on Oyster PAYG fares, observes the Travelcard time bands. It appears that because Oyster capping has a pricing structure analogous to that of a Travelcard's, Travelcard time bands are also observed. Time band definitions are not relevant with respect to cash singles as cash singles are not differentiated in price by time band.

	Oyster PAYG	**Period Products**
Peak/All-day	0630 – 0930, 1600-1900 Weekdays, excluding public holidays (Peak)	Valid at all times (All-day)
Off-Peak	All other times (Off-peak)	Valid outside of 0430-0930 on weekdays. (Off-peak)

Figure 3.15 - Comparing single product and period product time bands.

More subtly, the Oyster PAYG time bands have a fundamentally different interpretation compared to period product time bands. With Oyster PAYG time bands, what fare gets charged depends which time band a journey falls in. Oyster PAYG time bands are therefore mutually exclusive – one is either charged the peak fare or the off-peak fare.

With period products however, there is no notion of a 'peak' ticket or a 'peak' cap. Instead, there is an all-day ticket or cap that allows travel at all times, and an off-peak ticket or cap which only allows travel in the off-peak. The all-day and off-peak time bands for tickets and caps are *not* mutually

exclusive. Specifically, the off-peak time band is a subset of the all-day time band. A journey occurring in the off-peak is covered by both the off-peak and all-day tickets or caps.

3.5.2 Charge Code

There are some subtle but important differences between charge codes for single products and charge codes for period products. Single product charge codes are a function of the origin and destination stations of a given trip. Period product charge codes, in contrast, represent the *zonal validity* of a ticket or cap. The *zonal validity* of a ticket indicates the zones over which the user is entitled to travel at no charge if they hold that ticket. Although not stated explicitly in the fare description, we assume that the zonal validity of a period ticket pertains to the route of a journey and not just the zones of the origin and destination stations. In other words, a user holding a zones 1-2 Travelcard is only entitled to free journeys that do not exceed zones 1-2 at any point. From a fare inspection perspective a user holding a zones 1-2 Travelcard is allowed to be onboard a service outside the zones 1 and 2 as long as he has tapped in at the start of the journey. This is because travelling outside one's ticket's zonal validity is allowed, although an extension fare would be incurred. A user holding a zones 1-2 Travelcard and travelling on a journey which begins and ends within zones 1-2 but takes a route that ventures outsides of these two zones would likewise incur an extension fare. The amount of the extension fare would be based on the outermost reach of the most likely route of his journey.

Recall that single product charge codes represent one or more *zonal pairs*, which have a non-trivial relationship to the OD stations. As a result, special logic[11] is required to make the conversion. In the case of period products, however, there is no need for such conversion here because charge codes in a period context can be thought of less as criteria among which we must find a match for our journeys, and more as the 'name' of a product which is either selected or not.

To put more clarity on this philosophical distinction, let us take look at example drawn from the TfL fare description. Relevant excerpts of adult and child period prices for zones 1-6 only are given in Table 3.8 and Table 3.9

[11] ZonalPair={*innerzone, outerzone*}, see discussion in section 3.4.2.4.

Bus and tram	
Daily Price Cap	One Day Bus & Tram Pass
£3.30	£3.80

Tube, DLR and London Overground				
	Daily Price Cap		Day Travelcard	
	Peak	Off-Peak	Anytime*	Off-Peak
Journeys in:				
Zones 1-2	£6.70	£5.10	£7.20	£5.60
Zones 1-3	£8.10	£5.80	£8.60	-
Zones 1-4	£9.50	£5.80	£10.00	£6.30
Zones 1-5	£12.10	£7.00	£12.60	-
Zones 1-6	£14.30	£7.00	£14.80	£7.50
Zone 2	£6.70	£4.60	-	-
Zones 2-3	£8.10	£4.60	-	-
Zones 2-6	£8.50	£4.60	£9.00	£5.10

Bus and Tram Passes		
7 Day	Monthly	Annual
£13.80	£53.00	£552

Travelcards			
	7 Day	Monthly	Annual
Rates including travel in Zone 1			
Zones 1-2	£25.80	£99.10	£1032
Zones 1-3	£30.20	£116.00	£1208
Zones 1-4	£36.80	£141.40	£1472
Zones 1-5	£44.00	£169.00	£1760
Zones 1-6	£47.60	£182.80	£1904
Rates not including travel in Zone 1			
Zones 2-3, 3-4, 4-5 or 5-6	£16.60	£63.80	£664
Zones 2-4, 3-5 or 4-6	£21.40	£82.20	£856
Zones 2-5 or 3-6	£25.60	£98.40	£1024
Zones 2-6	£32.40	£124.50	£1296

Table 3.8 – Left(a): TfL fare description for adult 1-day Travelcard and Cap. Right(b): Adult 7-day, monthly and annual Travelcards [1]. Note that 'Peak' is a misnomer.

	Anytime*	Off-Peak
Journeys in:		
Zones 1-2	£18.40	-
Zones 1-6	£42.40	£21.20

Table 3.9 – Adult 3-day travelcard fare description.

DCA	**1-Day Cap All-day**
DCO	**1-Day Cap Off-peak**
DTA	**1-Day Travelcard All-day**
DTO	**1-Day Travelcard Off-peak**
3TA	**3-Day Travelcard All-Day**
3TO	**3-Day Travelcard Off-peak**
7TA	**7-Day Travelcard All-day**
MTA	**Monthly Travelcard All-day**
ATA	**Annual Travelcard All-Day**

Table 3.10 – Independent combination codes for Table 3.11.

The main difference between this example and the one from section 3.4.2.6 is that in the earlier example we begin with a combinatorially exhaustive list of zonal pairs and then build a basis set of charge codes by optimizing the list. In this example, charge codes do not come from an optimization procedure, but rather they come directly as a consequence of the definition of the ticket products.

Charge Code	Zonal Validity	1	2	3	4	5	6	Independent Combinations[12]
2	1-2							DCA,DCO,DTA,DTO,3TA,7TA
3	1-3							DCA,DCO,DTA, 7TA
4	1-4							DCA,DCO,DTA,DTO,7TA
5	1-5							DCA,DCO,DTA,7TA
6	1-6							DCA,DCO,DTA,DTO,3TA,3TO
7	2							DCA,DCO
8	2-3							DCA,DCO
9	2-6							DCA,DCO,DTA,7TA,MTA,ATA
10	2-3							7TA,MTA,ATA
	3-4							
	4-5							
	5-6							
14	2-4							7TA,MTA,ATA
	3-5							
	4-6							
	2-5							7TA,MTA,ATA
18	3-6							
19	Bus	N/A						DCA,DTA,7TA,MTA,ATA

Table 3.11 - Example charge codes for period products. 14 charge codes are needed to cover all adult products. Independent combination codes are listed in table Table 3.10.

For instance, take charge code 10 from Table 3.11. It has a zonal validity that encompasses the zonal pairs {2-3, 3-4, 4-5, 5-6}. It has this zonal validity because such a Travelcard product covering these zones is offered by TfL (see line 6 of Table 3.8b). If the Travelcard product offered according to the fare description were to change, the zonal validity of the charge code would follow suit.

Recall that the term *independent combination* describes the set of parameters from all but the last tier of a matrix fare structure. An independent combination for period products consists of the following parameters: [<application context>,<discount group>,<validity period>,<time band>]. Each independent combination identifies a family of ticket prices that differ only by charge code. In Table 3.11 we assume the discount group is Adult. For the other tiers we give independent combinations three letter codes which are explained in Table 3.10.

[12] Each independent combination identifies a family of ticket prices that differ only by charge code.

3.5.3 Matrix Utilization

Of the five dimensions in the period master matrix, three of these dimensions are featured in the tables below. Rows represent the discount group and columns represent combinations of the validity period and time band. *Application context* is separated by table. Table 3.12 represents tickets while Table 3.13 represents caps

Discount Group	DTA	DTO	3TA	3TO	7TA	7TO	MTA	MTO	ATA	ATO
Adult	Yes	Yes	Yes	Yes	Yes		Yes		Yes	
Child (5-15)	Yes	Yes	Yes	Yes	Yes		Yes			
16+					Child		Child		Child	
18+					Yes		Yes		Yes	
New Deal	Yes	Yes	Child	Child	Child		Child		Child	
Adult Privileged	Yes									
Child Privileged	Yes									

Table 3.12 – Matrix utilization – Application context = ticket. See above for independent combination codes.

Discount Group	DCA	DCO	3CA	3CO	7CA	7CO	MCA	MCO	ACA	ACO
Adult	Yes	Yes								
Child (5-15)	Yes	Yes								
16+										
18+										
New Deal	Yes	Yes								
Adult Privileged	Yes									
Child Privileged	Yes									

Table 3.13 - Matrix utilization – Application context = cap. See above for independent combination codes.

Off-peak tickets do not exist for the longer period products, hence the 7TO, MTO and ATO columns are empty. However there is no reason why TfL could not offer, for example, a monthly off-peak ticket. Such an offering would fill the gaps above. Currently TfL only supports a daily cap, therefore only the first two columns in Table 3.13 are filled. Longer period caps can be supported by filling the remaining columns as necessary.

3.6 *Interchanges*

An interchange, also known as a *transfer* describes moving from one vehicle to another within the same logical journey. An interchange occurs when a user changes lines on an LUL service, for example, from the District Line to the Victoria Line at Victoria Station. However these types of platform-to-platform interchanges occur without any interaction with the fare collection system. In the context of fare collection, we are particularly concerned with three types of interchanges which leave a traceable mark. These are out-of-station interchange, bus-rail interchange and National Rail interchange.

3.6.1 Out-of-Station Interchange

From	To	Interval
Bank Central/Northern/DLR	Bank Waterloo & City	30
Bank Waterloo & City	Bank Central/Northern/DLR	30
Canary Wharf (LUL)	Heron Quays (DLR)	10
Heron Quays (DLR)	Canary Wharf (LUL)	30
Euston	Euston Square	30
Euston Square	Euston	30
Hammersmith D&P	Hammersmith H&C	30
Hammersmith H&C	Hammersmith D&P	30
Hanger Lane	Park Royal	25
Park Royal	Hanger Lane	25

Table 3.14 – Example of OSI station pairs. Interval in minutes.

Separate OSI tables exist for Oyster and magnetic stripe tickets, with the magnetic stripe table being a subset of a more extensive Oyster table. Oyster specifications support only a 2-leg OSI journey. Additional interchanges will trigger a new fare, even if they would otherwise qualify for OSI. Furthermore, OSI is not granted if a journey breaking transaction, such as the recharging of the Oyster Card has occurred between the first exit and second entry. Many of these limitations of OSI arise out of technological limitations of the Oyster system, such as limited storage on an Oyster card.

3.6.2 Bus-Rail Interchange

Bus-Rail interchange refers to a mode change from bus/tram to a rail mode (LUL/DLR) vice versa. Oyster infrastructure explicitly supports this type of interchange and allows for a reduced/free fare

when an eligible bus/rail interchange occurs. However, this facility is not used by in the current TfL fare description. On the other hand, holders of any Oyster or paper based Travelcard are entitled to unconditional free access to all London buses.

3.6.3 National Rail Interchange

National Rail interchange describes the transition from a National Rail service to a TfL service (e.g. LUL or Underground). There are two types of National Rail-TfL interchange users. \

The first group of users hold a paper ticket for National Rail and access TfL services via either Oyster or their National Rail ticket. At stations where National Rail platforms are located outside of the TfL gated area, these users may simply access the TfL service as if they had just arrived at the station by any other means. At stations where National Rail services terminate within the gated area of a TfL station, users wishing to access TfL services (e.g. Underground) with an Oyster card may tap-in on a platform mounted validator. Those users who hold NR paper tickets with Underground permissions may simply proceed to the underground service with no intervention required. They will use their NR ticket to exit the system. The opposite interchange follows in similar fashion.

The second group of users use Oyster PAYG on both National Rail and their TfL (e.g. underground) connection. Currently, Oyster PAYG is only supported on a limited selection of National Rail routes, particularly those that parallel the London Overground. Users taking advantage of this capability need only to tap in at the beginning of their trip and tap out at the end, assuming they enter and exit through stations that support Oyster PAYG. No further action or intermediate validation is necessary.

3.7 *PAYG Capping*

TfL seeks to deliver on a concept of *daily best value* for its Oyster PAYG users, and implements it through a system of *PAYG capping*. The concept behind capping is that a user's PAYG charges will be capped at or slightly below the cost of a daily ticket that would cover the same zones that he has used that day. As it will be seen, PAYG capping does not always result in true best value for the user. A number of scenarios where best value does not result from the current capping implementation will be discussed.

3.7.1 Peak and Off-peak Caps

One source of confusion is whether there is a peak cap and an off-peak cap, or an all-day cap and an off-peak cap. The difference between these two can be seen in an example where a user travels many times within zone 1 during the peak (before 9:30am), and also many times during off-peak (after 9:30am). In the case of there being peak/off-peak caps, the user would incur a total cost which is the sum of the peak and off-peak caps. In the latter case, the all-day cap would override the off-peak cap, and the total cost would equal the amount of the all-day cap.

Let us consult the TfL fare description on this issue. The first quote is stated in the TfL Guide to Fares:

> "The Oyster daily price cap is the most you pay in one day when you pay as you go on bus, Tube, tram, DLR, London Overground and some National Rail services. The appropriate Off-Peak daily price cap will apply for all journeys on the same day:
>
> • Monday to Friday: from 0930 and any journey that starts before 0430 the following day,
>
> • Saturday, Sunday and public holidays: from 0430 and any journey that starts before 0430 the following day.
>
> The appropriate Peak daily price cap will apply if you travel from 0430 and before 0930 Monday to Friday (excluding public holidays). If you only use buses and trams, the bus and tram daily price cap will apply." [1]

The second quote is from the internal TfL staff manual:

> "The all modes off-peak cap will apply if they make several journeys on the same day during the (off-peak hours).
>
> The all modes peak cap will apply if they travel (during the peak hours), irrespective of mode.

A separate charge is made for journeys made (in the peak hours), plus the off-peak cap, if the total cost of the journeys made is less than the peak cap." (1/09 Staff Guide to Fares and Tickets)

These statements suggest that there is an all-day cap and an off-peak cap, and that the all-day cap supersedes the off-peak cap in an event of mixed trips. We will assume that TfL has an all day cap, not a peak cap. The use of the terms *peak cap* and *all-day cap*, if interchangeable, will always refer to the latter concept.

	Travel in off-peak	**No travel in off-peak**
Travel in peak	Min(all-day cap, off-peak cap + Σpeak singles, Σsingles)	Min(all-day cap, Σsingles)
No travel in peak	Min (off-peak cap, Σsingles)	£0

Table 3.15 – Capped fare under different daily travel patterns.

In reading the examples in this chapter one should note that PAYG fares are calculated using Oyster PAYG (peak/off-peak) time bands which observe a PM peak, but capping is applied using an all-day/off-peak system with no PM peak.

Oyster PAYG: Peak - 0630-0930, 1600-1900. Off-peak – 0930-1600, 1900-0630
Oyster capping: All-day – self explanatory. Off-peak – 0930-0430 (next day)

3.7.2 Oyster Implementation

In Oyster, capping is implemented using four *running totals* and a *stretching window* for each running total.

The four running totals are:

- LUL Peak
- LUL Off-peak
- LTB (bus) Peak
- LTB (bus) Off-peak

Bus fares have been drastically simplified since the inception of Oyster. Buses used to have zonal fares but this is no longer the case. Therefore we will consider only the two LUL running totals.

Running totals are reset at the beginning of each day. Each running total consists of three fields.

- Pound amount
- Inner zone
- Outer zone

Time Band	Running Total	Inner Zone	Outer Zone
LUL Peak	A	C	E
LUL Off Peak	B	D	F
LUL All Day (imputed)	A+B	MIN(C,D)	MAX(E,F)

Table 3.16 – Oyster implementation of capping showing imputed running total and stretching window

A third running LUL total, LUL All-Day is imputed from LUL Peak and LUL Off peak. This is necessary as there is no peak cap, but there is an all-day cap.

3.7.3 Running Total Example

Here we will use a sequence of trips to illustrate the relationship between the peak, off-peak and the imputed all-day running totals. In this particular example we will ignore for now the effect of capping. The initial state has all running totals set to zero and all stretching windows uninitialized. Note that for the purpose of examples in this chapter, 2008 fares are used. These fare amounts may not correspond with examples cited elsewhere in this thesis constructed using the 2009 fare schedule.

Trip 1: 8am from zone 3 to 1 - This is a peak trip under both the PAYG and capping time band definitions. A zones 1-3 peak fare of £2.50 is added to the LUL Peak running total. The peak window becomes 1-3.

Running Total	Pound Amount	Inner Zone	Outer Zone
LUL Peak	£2.50	1	3
LUL Off Peak	£0.00	-	-
LUL All Day (imputed)	£2.50	1	3
Total charged	**£2.50**		

Table 3.17 – Running total example: AM Peak trip.

Trip 2: 3pm from zone 1 to 5 - This is an off-peak trip under both the PAYG and capping time band definitions. A zones 1-5 off-peak fare of £2.00 is added to the LUL off-peak running total. The off-peak window is stretched to 1-5. The imputed all-day total increases to £4.50 and the all-day window is stretched to 1-5.

Running Total	Pound Amount	Inner Zone	Outer Zone
LUL Peak	£2.50	1	3
LUL Off Peak	£2.00	1	5
LUL All Day (imputed)	£4.50	1	5
Total charged	**£4.50**		

Table 3.18 - Running total example: Off-peak trip.

3) Trip beginning at 5pm from zone 6 to 3 - This is a peak trip under the PAYG time band definition but off-peak under the capping time band definition. A zones 3-6 *peak* fare of £1.80 is added to the LUL *off-peak* running total. The off-peak window is now stretched to 1-6. The imputed all-day total is now £6.30 and the all-day window is now 1-6.

Running Total	Pound Amount	Inner Zone	Outer Zone
LUL Peak	£2.50	1	3
LUL Off Peak	£3.80	1	6
LUL All Day (imputed)	£6.30	1	6
Total charged	**£6.30**		

Table 3.19 – Running total example: PM peak trip.

3.7.4 Worked Example

The off-peak and all-day running totals are capped by an amount which is a function of the inner and outer zones. There is no cap for the peak running total. However the peak running total is implicitly capped as it constitutes a component of the imputed all-day total, which *is* capped. Now we will walk through an example that illustrates the capping mechanism.

Running Total	Pound Amount	Inner Zone	Outer Zone
LUL Peak	£2.00	1	2
LUL Off Peak	£4.00	1	5
LUL All Day (imputed)	£6.00	1	5
Total charged	**£6.00**		

Table 3.20 – Capping example: Initial state.

This example begins with the running total and stretching window registers in the state shown in Table 3.20. A peak 1-2 trip (£2.00) and two off-peak 1-5 trips (£2.00 each) have put the registers in this state. The user now makes two additional off-peak trips within zones 1-5. This should have incurred two more £2.00 fares, raising the LUL off-peak running total to £8.00. However, the off-peak total does not reach £8.00 as it is subject to a £6.50 off-peak zones 1-5 cap. The status now of the running totals is shown in Table 3.21 below.

Running Total	Pound Amount	Inner Zone	Outer Zone	Cap
LUL Peak	£2.00	1	2	N/A
LUL Off-peak	£6.50	1	5	£6.50 (OP 1-5)
LUL All Day (imputed)	£8.50	1	5	£11.30 (AD 1-5)
Total Charged	**£8.50**			

Table 3.21 – Capping Example: Step 1. Off-peak cap reached after two additional off-peak zone 1-5 trips.

The TfL fare guide states:

> "If the total cost for your journeys less than the all-day daily price cap, you will be charged for each journey you make from 0430 to 0930, plus the cheaper of the Off-Peak Oyster single fares/off-peak daily price cap for the remainder of your journeys". [1]

We can verify that this promise is indeed met in our scenario above. The total cost for all journeys (£8.50) is less than the all-day price cap for zones 1-5 of £11.30. The total amount charged therefore consists of the off-peak cap (£6.50) and a one-way peak 1-2 journey (£2.00).

The off-peak cap has now been reached. Any further off-peak trips between zones 1-5 will result in no change to the running totals and no stored value deductions. However, additional peak trips

within zones 1-5 will increase the peak running total. This is not currently possible as there is no PM capping peak. But we can assume there is and test this case by giving the user two zones 1-5 PM peak trips (£3.50). The peak running total increases until the all-day cap of £11.30 is reached. As a result of the all-day cap, the peak running total hits a ceiling of £4.80 rather than the individual trip costs of £2 + £3.50 x 2 = £9.00.

At this point, no running total can be increased for any subsequent peak or off-peak travel between zones 1-5. Consequently the user's Oyster balance will not be deducted any further for any peak or off-peak travel between zones 1-5. (The user's stored value is deducted only when the running total increases)

Running Total	Pound Amount	Inner Zone	Outer Zone	Cap
LUL Peak	£4.80	1	5	N/A
LUL Off-peak	£6.50	1	5	£6.50 (Zones 1-5)
LUL All Day (imputed)	£11.30	1	5	£11.30 (Zones 1-5)
Total charged	**£11.30**			

Table 3.22 – Capping Example: Step 2. Two more £2.00 peak zones 1-5 trips are made. This is only possible with a hypothetical PM peak. The total charged is £11.30 rather than £12.50 without capping.

3.7.5 Capping Limitations

The capping scheme employed by Oyster fails to provide best value to users under the following circumstances:

1. Zonal overextension
2. Cross-band zonal overextension
3. Non-contiguous zonal overextension
4. Capping with existing tickets

We will illustrate each case below with an example.

3.7.5.1 Zonal Overextension

Consider the following scenario:

					Off-Peak				All-Day			
Jrny No.	**Time**	**From Zone**	**To Zone**	**Fare**	**Inner Zone**	**Outer Zone**	**Cap**	**Running Total**	**Inner Zone**	**Outer Zone**	**Cap**	**Running Total**
1	7:30	5	1	£3.50	-	-	-	£0.00	1	5	£11.30	£3.50
2	8:30	1	1	£1.50	-	-	-	£0.00	1	5	£11.30	£5.00
3	8:40	1	1	£1.50	-	-	-	£0.00	1	5	£11.30	£6.50
4	8:50	1	1	£1.50	-	-	-	£0.00	1	5	£11.30	£8.00
5	9:00	1	1	£1.50	-	-	-	£0.00	1	5	£11.30	£9.50
6	9:00	1	1	£1.50	-	-	-	£0.00	1	5	£11.30	£11.00

Table 3.23 – Zonal overextension example: Oyster capping.

The first trip of the day from zone 5 to 1 extends the all-day zonal window to 1-5. This in turn raises the all-day cap to £11.30 (the price of the zones 1-5 daily cap). The five subsequent trips within zone 1 are charged individually and the user is charged a total of £11.00 for the 5 journeys. However, this does not represent best-value for the user. Best value for the user in this scenario is represented by the application of a zone 1-2 daily price cap (£6.30) in conjunction with a peak extension fare for zones 3-5 (£1.80), resulting in a total charge of £8.10.

	Cost
All day cap - Zones 1-2	£6.30
Peak extension one-way - Zones 3-5	£1.80
Best Value	£8.10

Table 3.24 – Zonal overextension example: True best value.

To highlight a quirk of the Oyster capping mechanism, we can also consider the same trips as above but made in the opposite order (five zone 1 trips followed by a zone 1 to 5 trip). All trips still occur in the peak. This results in a charge £9.80, still non-optimal, but less than the £11.00 charged in the first example. We can see that the current Oyster capping rules, the same trips (occurring in the same time band) but taken in a different order can result in a different total charge.

					Off-Peak				All-Day			
Jrny No.	Time	From Zone	To Zone	Fare	Inner Zone	Outer Zone	Cap	Running Total	Inner Zone	Outer Zone	Cap	Running Total
1	7:30	1	1	£1.50	-	-	-	£0.00	1	1	£6.30	£1.50
2	8:30	1	1	£1.50	-	-	-	£0.00	1	1	£6.30	£3.00
3	8:40	1	1	£1.50	-	-	-	£0.00	1	1	£6.30	£4.50
4	8:50	1	1	£1.50	-	-	-	£0.00	1	1	£6.30	£6.00
5	9:00	1	1	£1.50	-	-	-	£0.00	1	1	£6.30	£6.30
6	9:00	1	5	£3.50	-	-	-	£0.00	1	5	£11.30	£9.80

Table 3.25 – Zonal overextension example: Oyster capping with trips in reverse order.

3.7.5.2 Cross-band Zonal Interference

Zonal window overextension can reach across pricing time-bands. Because the all-day window is imputed from the peak and off-peak windows, off-peak travel will stretch not only the off-peak window, but also the all-day window.

					Off-Peak				All-Day			
Jrny No.	Time	From Zone	To Zone	Fare	Inner Zone	Outer Zone	Cap	Running Total	Inner Zone	Outer Zone	Cap	Running Total
1	11:00	5	1	£2.00	1	5	£6.50	£2.00	1	5	£11.30	£2.00
2	16:30	1	1	£1.50	1	5	£6.50	£2.00	1	5	£11.30	£3.50
3	16:50	1	1	£1.50	1	5	£6.50	£2.00	1	5	£11.30	£5.00
4	17:10	1	1	£1.50	1	5	£6.50	£2.00	1	5	£11.30	£6.50
5	17:20	1	1	£1.50	1	5	£6.50	£2.00	1	5	£11.30	£8.00
6	17:30	1	1	£1.50	1	5	£6.50	£2.00	1	5	£11.30	£9.50

Table 3.26 – Cross-band zonal interference: Oyster capping.

	Cost
All-day Cap – Zones 1-2	£6.30
Peak extension one-way – Zones 3-5	£0.90
Total Charge	£7.20

Table 3.27 – Cross-band zonal interference: True best value.

The first trip is an off-peak trip from zone 5 to 1 (£2.00). It stretches not only the off-peak window, but also the all-day window. Subsequently a series of five PM-peak zone 1 trips (£1.50) are made. Again, under the current fare description capping does not use a PM peak, making it impossible to incur a peak trip once the morning peak is over. However, we can assume for the sake of this

example that a PM peak exists between 1600 and 1900 (same as the Oyster PAYG PM peak). The resulting charge of £9.00 is not best value. Best value is represented by the charges in Table 3.27.

A trend emerging from these examples is that extension fares are often utilized in true best-value scenarios. The inability of the existing mechanism to incorporate extension fares results in sub-optimal charges, belying the spirit of 'best value' promised to customers as stated below:

> "During a 24 hour period from 0430 to before 0430 the following day, you will pay 50p less than the equivalent Day Travelcard (or One Day Bus & Tram pass) price for all your Oyster single fare journeys or we will refund the difference." (pp 45, 1/09 Staff Guide to Fares and Tickets)

On the other hand, if we consider the TfL's fare description at face value (reprinted from 3.7.4 above), it appears the fare description is not violated.

> "If the total cost for your journeys less than the all-day daily price cap, you will be charged for each journey you make from 0430 to 0930, plus the cheaper of the Off-Peak Oyster single fares/off-peak daily price cap for the remainder of your journeys." [1]

In both examples above (3.7.5.2 and 3.7.5.3), the total cost of journeys is less than an all-day daily price cap for zones 1-5. Therefore all 5 peak journeys are charged individually, and for the remainder (journey 1) the single fare is also charged as it is cheaper than the corresponding off-peak cap.

3.7.5.3 Non-contiguous Zonal Overextension

The zonal window is defined only in terms of its inner and outer limits. Non-contiguous windows are not supported. Consequently, zonal window overextension occurs when the system is confronted by a usage pattern that involves discontinuous zones. Such a usage pattern can arise when a user first travels within certain outer zones, then gets transported via alternative means, and finally resumes travel within a non-adjacent set of inner zones.

In the example below, four peak zone 1-2 (£2.00) trips are interrupted by a single zone 6-6 (£1.00) trip. This is a highly unlikely itinerary in real-world usage, but nonetheless a possible one. The zone 6-6 trip (Journey 4) extends the zonal window to 1-6, causing subsequent to zones 1-2 trips to not be

capped. Best value in this example is given by an all-day zone 1-2 cap (£6.30), and a single zone 6-6 one way fare (£1.00), for a total of £7.30. Oyster capping gives a much higher charge of £13.00.

					Off-Peak				All-Day			
Jrny No.	Time	From Zone	To Zone	Fare	Inner Zone	Outer Zone	Cap	Running Total	Inner Zone	Outer Zone	Cap	Running Total
1	4:30	1	2	£2.00	-	-	-	£0.00	1	2	£6.30	£2.00
2	5:00	2	1	£2.00	-	-	-	£0.00	1	2	£6.30	£4.00
3	5:20	1	2	£2.00	-	-	-	£0.00	1	2	£6.30	£6.00
4	6:20	6	6	£1.00	-	-	-	£0.00	1	6	£13.30	£7.00
5	7:20	1	2	£2.00	-	-	-	£0.00	1	6	£13.30	£9.00
6	9:00	2	1	£2.00	-	-	-	£0.00	1	6	£13.30	£11.00
7	9:15	1	2	£2.00	-	-	-	£0.00	1	6	£13.30	£13.00

Table 3.28 – Non contiguous zonal overextension: Oyster capping.

We can also see that *cross-band* non-contiguous zonal window overextension is also possible (if we have an afternoon peak). Again we assume a 16:00-19:00 afternoon peak. If Journey 4 in the table above were to occur during the off-peak at 13:00, it would stretch out the all-day window to 1-6. Subsequent journeys in the PM peak within zones 1-2 would fail to be capped.

3.7.5.4 Capping with Pre-existing Tickets

If the user holds a pre-existing period ticket, extension fares may be charged instead of the full single fare. Any extension fare charged is subject to the Oyster PAYG cap. In other words, contributions to the running totals may have already been reduced by a pre-purchased ticket. In this case, each contribution to the running total is no longer simply the one-way fare, but rather, the lesser of the raw one-way fare and the extension fare given the user's ticket holding. Under these circumstances, best value may not result. We will illustrate this with an example below.

Here we assume that the user is in possession of a valid zones 1-2 monthly ticket. The first peak zone 1-1 trip incurs no charge as it is fully covered by the user's monthly ticket. However the peak (not shown) and all-day zonal windows are still affected. The subsequent sequence of zone 1-6 trips (however unlikely in the real world) each incur a peak zone 3-6 extension fare of £1.80, until the 1-6 peak cap of £13.30 is reached

However, again this does not represent best value. In fact we can easily see that best value in this scenario is attained by applying the zones 2-6 daily cap, which is £7.90.

					Off-Peak				All-Day			
Jrny No.	Time	From Zone	To Zone	Fare	Inner Zone	Outer Zone	Cap	Running Total	Inner Zone	Outer Zone	Cap	Running Total
1	4:30	1	1	£0.00	-	-	-	£0.00	1	1	£6.30	£0.00
2	4:40	1	6	£1.80	-	-	-	£0.00	1	6	£13.30	£1.80
3	5:10	6	1	£1.80	-	-	-	£0.00	1	6	£13.30	£3.60
4	5:40	1	6	£1.80	-	-	-	£0.00	1	6	£13.30	£5.40
5	6:10	6	1	£1.80	-	-	-	£0.00	1	6	£13.30	£7.20
6	6:40	1	6	£1.80	-	-	-	£0.00	1	6	£13.30	£9.00
7	7:10	6	1	£1.80	-	-	-	£0.00	1	6	£13.30	£10.80
8	7:40	1	6	£1.80	-	-	-	£0.00	1	6	£13.30	£12.60
9	8:10	6	1	£0.70	-	-	-	£0.00	1	6	£13.30	£13.30
10	8:40	1	6	£0.00	-	-	-	£0.00	1	6	£13.30	£13.30

Table 3.29 – Capping with a zone 1-2 ticket.

3.7.6 Capping and Buses

The PAYG Oyster caps are 'all-modes' caps and include journeys buses. At the same time, an all-day bus daily price cap of £3.00 is in effect concurrently. Bus capping is implemented via an LTB (bus) running total which is also accounted for by the all-day imputed total. The store value deduced is actually computed from an 'all-day all-modes' total which is itself a minimum of the 'all-day LUL' and 'all-day LTB' totals. We have glossed over this point previously to avoid confusion.

For example, an Oyster user travelling during the peak LUL within Zones 1-2 as well as on London buses would be charged the lesser of £6.70 for all her trips (the all-modes zones 1-2 peak cap) or £3.00. This is consistent with the fact that a user who purchases a Zone 1-2 all-day Travelcard for £7.20 enjoys the use of all London buses in addition to TfL rail services within Zones 1-2.

3.8 *Other Services*

We will briefly summarize a number of other services currently offered by TfL's fare collection system. These services primarily pertain to fraud prevention, error correction and contingency operations. This discussion focuses on Oyster only.

- **Unfinished and Unstarted Journeys** - TfL implements a series of checks for unexpected behavior. Consecutive entries and exits at gated stations are detected and flagged. Normally this means a previous journey had not been exited properly (unfinished journey), or the

current journey had not been entered into properly (unstarted journey). In these cases a penalty fare is charged.

- **Continuation Rail Exit** - An exception to the above rule is if the user has to pass through two sets of gates to enter or exit a station. TfL nomenclature describes this as a continuation rail exit.

- **Passback** - If two entry or exit transactions are detected within a short interval at the same gate line, a condition known as passback is assumed to have occurred. Passback describes a possible attempt to exploit the system by using the same card to let two people through a gate line. When a passback event is detected the second tap results in denied passage (rather than a penalty fare).

- **Maximum Trip Length** - The interval between successive taps in a PAYG journey is bounded by a maximum time. If a user remains in the system for longer than this amount of time he will be charged a penalty fare. In a qualifying OSI journey, the trip timer is activated separately for each of the two legs. If we assume the maximum trip length is 2 hr, the maximum trip interval for an OSI journey would be 2 hr + (Interchange interval for given OSI pair) + 2 hr.

- **U-Turn Rail Journey** - The user enters and exits at the same station without having traveled anywhere. If this occurs in a short succession the user is assumed to have changed her mind about travelling, in this case she is charged a single zone fare. If this occurs over a longer time span, a fraud attempt is suspected (e.g. using two cards to manipulate the system). In this case a penalty fare is charged.

- **Automatic Resolution of Incomplete Journeys** – It is important to resolve uncompleted journeys in situations where passengers may get their Oyster card into an invalid state through no fault of their own. Overcharging customers a penalty fare where responsibility lies with TfL is clearly unacceptable. The detection and resolution of these circumstances is helped by a number of automatic processes.

  - **Auto Continuation** – If a passenger is evacuated from the station due to an emergency circumstance, the auto-continuation flag is set on his card so that when he re-enters the

system at a later point the earlier journey allowed to be continued (rather than treated as an unfinished journey). In auto continuation the exit-entry is pair is assumed to never have happened.

- **Auto Completion** - Auto-completion is closely associated with auto continuation. The difference between the two is that with the auto-completion flag set, when the user re-enters the system after being let out (due to overcrowding or emergency), the re-entry location is extrapolated to become the destination of the earlier (still open) journey, causing a fare to be charged.

- **Aliasing** - Aliasing is a service aimed at reducing the inconvenience to users and TfL of closed stations. Stations near a closed station are treated as if they are the closed station providing this treatment benefits the passenger. The closed station is the 'aliasing' station. The alternative stations are the 'aliased' stations. When a user enters or exits through an aliased station that results in a longer trip (and therefore more expensive fare) than had the aliasing station been used, Oyster gives him the benefit of the doubt and charges him the original lower fare.

3.9 *Motivations for a Fare Structure*

3.9.1 Motivation vis a vis Oyster

A large part of TfL's fare description is already implemented in Oyster. In fact, many of the issues discussed above would no doubt have been considered at length during the development of Oyster. This fact appears to stand in awkward juxtaposition with our search for a fare structure. One would not be unreasonable to question why we are attempting to reinvent the wheel, and what motivates us to explain something that has already been implemented in Oyster.

We will preface our answer with a clarification of what we are *not* doing.

This is not an investigation of just Oyster. Oyster only implements a subset of TfL's fare description. We seek to define a fare structure which accounts holistically for all fares supported by TfL. Other than Oyster, there are also magnetic stripe cash singles and Travelcards.

This is not an attempt to faithfully document the implementation of Oyster. While we do try to understand and document how Oyster works, as well as its limitations, ours is primarily an effort to develop a foundation of assumptions for a contactless bankcard fare engine. Structures proposed may differ markedly from how Oyster operates. In many cases we have not been able to gather sufficient information about how certain aspects Oyster works to be able to claim a basis on it. In these cases, new interpretations, structures and methods have been created.

Finally, this is not black-box reverse engineering. We are not attempting to replicate the functionality of Oyster without reference. Despite some missing elements, there is access to a degree of internal technical documentation for Oyster. For example, the content and organization of data storage on the Oyster smartcard card is known. When it makes sense, what is known about Oyster is actively referenced and incorporated into the fare structure we propose.

This chapter draws on TfL's institutional knowledge about Oyster, as well as the Oyster Card Travel Analyzer (OCTA), the agency's own internal reverse-engineered version of Oyster. Ultimately, instead of being in conflict with our own development of a fare structure, Oyster's existence provides us with an additional motivation – to produce as a side product a concise summary of a system which is immensely complex and for which comprehensive documentation is not readily available.

3.9.2 The User Experience Motivation

A coherent fare structure can assist users in determining which cash single (should they want one), or period ticket to purchase, by helping them narrow down their choices in a systematic manner. One venue where this can be done is the design of selection screens on a ticket vending machine. Another potential application is the TfL website. A systematic and consistent framework can help guide users in purchasing the correct period ticket product based on their needs and the characteristics of their travel.

The large array of fare and ticket offerings can be confusing for new users of the TfL system (and even some frequent users). The current fare description is difficult to articulate at the point of sale, whether at a ticket vending machine or the ticket window. A more structured framework may assist in the future evolution of the fare system.

3.9.3 *The Next Generation Fare Engine Motivation*

A fare engine can, if desired by TfL, remove the entire burden of product (fare and ticket) choice from the user. With the computational power available on a central server, and with good algorithm design and implementation, it appears possible to offer the traveler the best value across all available ticket options. Understanding the fare structure is the first step in enabling this level of automation.

4 Fare Engine Requirements

In this section we describe the scope and requirements for our fare engine for TfL. Requirements are separated into two categories, *general requirements* and *prototype requirements*. General requirements represent requirements and features that would allow for maximum flexibility in a production fare engine.

On the other hand prototype requirements represent a limited scope which is suitable for a demonstration prototype. These are a subset of the general requirements and take into account the time and resource constraints in building a prototype.

4.1 *Travel Services*

The fare engine will provide users with the following general travel services

1. **One-card passage through system** – Users should be able to use the same contactless bankcard for passage through the entire TfL network, by means of tapping the bankcard on gates or validators encountered along the way. Upon the completion of a journey, a fare will be computed and posted to the account associated with the bankcard. Fare will be aggregated and the bankcard will be charged in due course.

2. **Direct interaction with gates and validators** – Users should be able to interact directly with station fixtures which they encounter in the course of their journey, such as gates and validators. Users should not be required to purchase any other fare instrument with their contactless bankcard, or be asked to initialize their travel at a kiosk or similar machine.

3. **Intuitive interaction at stations and aboard vehicles** – Users should be able to complete a journey legally by interacting with devices found along the quickest or any common sense path through a station or vehicle. Users should not be required to undertake any critical transactions which are non-intuitive, easy to forget or require a detour, the omission of which would result in the levying of a penalty fare.

4.2 *Product Spheres*

The fare engine should support *single fare products* as described in section 3.4. These are products which allow users to pay for their travel through the TfL system on a trip-by-trip basis.

The fare engine should support *period products* as described in section 3.5. These are products which allow users travel on an unlimited basis through the TfL system within the constraints of the period product.

4.3 *Fare Media*

The fare engine should accommodate only contactless bankcards on the TfL network. While Oyster and magnetic stripe fare media are expected to remain in concurrent use with contactless bankcards, they will be supported via parallel and independent fare collection systems.

4.4 *User Identification*

Users are identified by unique user account numbers. If a new, previously unseen contactless bankcard is presented to the system, a new user account linked to this card may be created. User accounts may be associated with one or more contactless bankcard number. Account numbers may correspond with bankcard numbers or be separately generated.

The contactless bankcards associated with each account may be changed by the user. The fare engine may safely assume that only one bankcard is presented unambiguously at a time. If multiple bankcards are presented to the reader simultaneously, RFID anti-collision standards will cause the transaction to be rejected.

4.5 *Modes Supported*

There are two general classes of modes. *Station Based* and *On-board*. Station-based modes involve travel between geographically defined stations. Station based journeys have a well defined origin and destination. On-board modes are modes where validators are installed aboard vehicles. An on-board journey is identified by service number.

General Requirements	Prototype Requirements
Station Based • Underground • Overground • DLR • National Rail • River Boats **On-Board** • Buses • Trams (Tramlink)	**Station Based** • Underground • Overground • DLR **On-Board** • Buses

4.6 *Single Products*

4.6.1 Fare Calculation

General Requirements	Prototype Requirements
Station Based The calculation of fare for a given journey may be based upon any fare calculation logic module which satisfies a standardized interface. Such a model will use the origin and destination (OD) as well as any intermediate validation signature available (see 4.8.5 below). Such a module should be deployable on a plug-and-play basis. Fares may be computed based on some arbitrary fare function (Non-zonal fares) of the OD and intermediate validation (IV) signature. Fares may be computed in terms of the zones travelled of the journey path (zonal fares) and a	**Station Based** Fares are computed in terms of the zonal reach of the journey path (Zonal fares) and a zonal fare matrix. The fare engine supports the TfL fare zone system as described in sections 3.4.2.1. Fares from a given origin station to a given destination station are calculated on the basis of the inner most and outer most zone traversed following a predetermined 'most likely case' path as described in section 3.4.2.2. Any IV signature given will be taken into account.

zonal fare matrix. The fare engine should support the TfL fare zone system as described in sections 3.4.2.1. Concentric zones are defined centered around central London. Stations may either reside fully within one zone or straddle two zones. Border stations may be assigned to whichever zone results in the lower fare for the journey. Zonal fare special cases as described in section 3.4.2.3 should be supported. Fares may be computed in terms of both zonal and arbitrary (non-zonal) contributions. **On-Board** Fares are generally calculated on a fixed rate basis depending on the service used. Fares may be calculated with an optional dependence on the boarding and alighting location data. (e.g. distance of travel, zones spanned) Where location data is used, an arbitrary fare matrix or calculation regime may be deployed on a plug-and-play basis.	**On-Board** Fares are calculated on a fixed rate basis. The same bus fare is charged for all bus journeys.

4.6.2 Time Bands

We define *time bands* to be labels that are associated with their own fare brackets. E.g. Peak, Off-peak, Late-night, Weekend, etc. Time bands determine what fare a journey should incur based on when it took place. An arbitrary number of time bands is supported by the fare engine.

Time band intervals are intervals of time for which a given time band applies for that OD pair. Time band intervals may be defined differently for each OD pair or in general for all journeys.

We define *timeline* as an unbounded, non repeating time axis.

We define *complete operational coverage* to mean that every time instance along the timeline during which the system is operating must be assigned to one and only one time band.

General Requirements	Prototype Requirements
For each OD pair, the timeline must be given complete operation coverage by mutually exclusive time band intervals. Time band intervals may have a pattern that recurs on a daily cycle basis (e.g. 7-9am every weekday), a weekly cycle (e.g. weekdays and weekends) or they may be non-repeating and arbitrarily defined (e.g. irregular bank holidays which occur at different times each year). Time band intervals for different OD pairs may be distilled down to a finite number of *time band patterns*. For example, 'inbound' (AM peak), 'outbound' (PM peak) or 'bidirectional' (both peaks)	Two time bands are supported by the fare engine, Peak and Off-Peak. These time bands apply to all journeys. In other words, the time band intervals are identically defined for ODs. These are: **Peak:** 4:30am – 9:30am weekdays **Off-peak:** After 9:30am weekdays and weekends No bank holidays implemented.

4.6.3 Discount Groups

We define *discount groups* to be labels that are associated with their own fare brackets. Every user must belong to discount group fare bracket. Which discount group a user belongs to depends on his eligibility to receive a discount. Discount group is a property of the user account and does not vary from trip to trip.

General Requirements	Prototype Requirements
The default discount group for an unrecognized contactless bankcard number is Adult. An arbitrary number of independent discount groups is supported by the fare engine.	The default discount group for an unrecognized contactless bankcard number is Adult. • Adult • Child • New Deal

4.7 *Journey Properties*

Journeys may consist of travel on station-based modes, on-board modes, or a combination of both. Stations may use fare gates to define *fare paid area,* or they may be *open access*, equipped with only validators. Stations which are gated at some entrances but ungated at other entrances (relying on validators at those entrances) are considered open access.

General Requirements	Prototype Requirements
Station Based Users must validate at both the start station and end station by touching the contactless smartcard on a fare gate or a validator at each station. Such an entry-exit pair constitutes a journey and incurs one fare[13]. A journey begun with a tap at a station must end with a tap at a station. Likewise a journey ending with a tap at a station must have begun with a tap at a station. All stations are equipped with functioning validators or fare gates. Malfunctioning or	**Station Based** Same as general requirements **On-Board** User validates only at time of boarding. Validations are associated with service identifying data (e.g. bus route and direction)

[13] Unless preempted by journey linking, see section 4.8.

disabled equipment will lead to a penalty fare being charged. There is no requirement for a journey begun at a gate to end at a gate. A journey may be started at a gate and end at a validator, or vice versa. **On-Board** User may validate at time of boarding, at the time of alighting, or both. Validations are associated with service identifying data (e.g. bus route and direction) Validations may be associated with location identifying data (e.g. fare zone, milepost or GPS coordinates).	

4.8 *Journey Linking*

4.8.1 Definition of a Linked Journey

A *linked journey* is defined as a movement through the TfL system from an origin to a destination represented by a single fare. A journey may involve travel on only one type of vehicle, multiple vehicles of the same mode, or multiple vehicles of different modes.

We will outline the requirements for a journey below. Note our definition of a journey is a *fare-centric* one. A journey is only considered such when the criteria below for a single fare are met. If a user's travels incur two fares by the requirements below, they are considered two journeys regardless of whether it is his intention or not. Conversely, consider a user who travels from A to C making a stop at station B to meet a friend briefly. Even though the user's intention here is to make two journeys, the fact that his travels incur only one fare by application of rules below means we consider them to be one single journey.

4.8.2 *Journey Segments*

A journey segment is defined as the smallest unit of measurable travel. A journey is composed of one or more journey segments. Note that the exact same journey may be composed of different journey segments. For example, if a user taps in at Station A and taps out at Station C, we can deduce from the available information that the user has travelled from A to C. However we do not know what route he may have taken. Here the smallest unit of measurable travel is from A to C, which incidentally also constitutes the linked journey itself. Now consider another user who travels from Station A to C, and performs an *intermediate validation* (see below) at Station B. From the given information we now know that the user has travelled from Station A to C via station B. The smallest units of measurable travel here are trips from A to B and B to C. The linked journey is still from A to C, however this linked journey is now composed of two *journey segments*, A->B and B->C.

Note also that journey segments have no immediate correlation with physical lines or vehicles. It is incorrect to equate a journey segment with a one ride on a bus or underground railcar. To see why this is, consider the straightforward example of a user who travelled from Putney Bridge to Vauxhall by way of the District Line and the Victoria Line. He used two distinct underground lines and up to three different trains for this journey. However, he only had two interactions with fare devices; he tapped in at Putney Bridge and tapped out at Vauxhall. There are no validators on underground platforms and no way to physically register interchanges within the underground. For this reason, the user's entire trip is the smallest measurable block, and it constitutes one journey segment. Because he is charged one fare for this trip, his linked journey is also coterminous with the journey segment.

4.8.3 *Journey Linking Criteria*

Under certain conditions the fare engine must link journey segments into linked journeys so that users are charged one fare for the journey rather than separately for the constituent segments. These requirements ensure that users whose journeys require multiple interchanges between modes that each consist of a measurable journey are not disadvantaged.

The fare engine will support three journey linking scenarios: *out-of-station interchanges*, *intermediate validation* and *cross-mode interchange*.

4.8.4 *Out-of-station interchange (OSI)*

Out-of-station interchanges allow users to change lines on the Underground (or more generally, interchange between any two gate-line controlled station-based services) at closely located but physically disjoint stations. As these stations are not connected, users must tap out of the first station and tap into the second station. Without OSI this would be considered two journeys and incur two fares. OSI allows these station pairs to serve as interchange points without a fare disadvantage compared to purpose-built interchange stations that allow behind-fare-gate interchanges. Existing TfL fare description regarding OSI is described in section 3.6.1.

General Requirements	**Prototype Requirements**
The fare engine should support an arbitrary number of OSI station pairs. OSI station pairs are ordered pairs and are defined uni-directionally such that an interchange from station A to B is represented separately from an interchange from station B to A (see section 3.6.1). Each OSI station pair is associated with an interchange interval within which an interchange must be completed. Interchange intervals may differ between OSI station pairs and between the two opposite OSI pairs of the same two stations. A user who taps out of the first station in an OSI pair and taps in at the 2nd OSI station within associated interchange interval will have made only one journey and incur one fare. Any number of OSI journey segments may be chained together to form one merged journey. An upper bound on the number of OSI journey segments that can be merged into one journey may be set.	Same as general requirements with no upper bound on the number of consecutive or non-consecutive OSI journey segments which may be accepted into one journey. In other words, 3 consecutive OSI's, or an OSI followed by an intermediate validation followed by another OSI are both acceptable scenarios.

4.8.5 *Intermediate Validation*

We saw in section 3.4.2.2 how path choice indirectly affects the fare by dictating the inner and outer zones. Currently, the actual path chosen by the user cannot be known. Path choice on TfL is assigned either manually or by algorithm on a 'most likely case' basis.

The fare engine will support National Rail services and stations. This requirement makes explicit path choice an important consideration for two reasons. First, it may be a business requirement to charge fares as a function of journey path rather than levy a standard 'most likely case' fare. Second, even where fares are levied on a 'most likely case' basis, knowledge of the path chosen by each user will allow more accurate allocation of revenue among the Train Operating Companies (TOCs).

Intermediate validation is a technique which allows users to indicate mid-journey waypoints by means of interaction with validator devices. This allows a journey which was previously one measurable unit of travel to be broken up into two or more measurable units of travel. Such a journey has gone from being composed of one single journey segment to being composed multiple journey segments.

Note that intermediate validation affects the waypoints of a journey but preserves the origin and destination.

General Requirements	**Prototype Requirements**
The fare engine should support intermediate validation at stations equipped with platform validators. Users should be able to perform intermediate validation by tapping on validators on the platform or within stations as they interchange between services. Intermediate validation may be performed at validators specifically installed for the purpose of intermediate validation (e.g. within the paid area	The fare engine should support intermediate validation at stations equipped with platform validators. Users should be able to perform intermediate validation by tapping on validators on the platform or within stations as they interchange between services. Intermediate validation is performed at standard entry-exit validators (such as at DLR stations) Intermediate validation is optional. Users may be

of a gated station) or it may be performed at standard entry-exit validators (such as at DLR stations) Intermediate validation should be optional. Users may be charged a higher default fare for failing to intermediate validate but should not be unduly penalized by the application of penalty fare. For each given OD, a list of accepted intermediate validation stations may be specified. If such a list is provided then only validations at legitimate IV stations are processed, otherwise the transaction is considered illegal. Alternatively, as a less rigorous constraint, a global list of intermediate stations may be provided such that only intermediate validations at these stations will be accepted. No consideration is given to the OD of the wider journey. An upper bound on the number of intermediate validation transactions may be imposed. This limit may be imposed on a global basis or on the basis of individual OD pairs.	charged a higher default fare for failing to intermediate validate at intermediate points but should not be unduly penalized by the application of a penalty fare. There is no restriction on which stations may serve as an IV station. There is no upper bound on the number of valid intermediate validation points. Intermediate validations may occur consecutively or non-consecutively (punctuated by OSIs).

4.8.6 Cross-mode Interchange

A cross-mode interchange is an interchange involving non-rail/underground modes, for example between station based and on-board validation modes. An example of a cross-mode interchange is a bus-rail interchange. Although it does not strictly involve a change of mode, we will also classify interchange between buses as a cross-mode interchange. When a cross-mode interchange is applied the user is considered to have made one continuous journey and is charged one fare for that journey composing of multiple bus or rail legs.

General Requirements	Prototype Requirements
Bus-rail interchanges may be allowed at all times, or restrictions may be in place such that each station is associated with a fixed set of services (e.g. bus lines) for which bus-rail interchange is allowed. This means, for example, that a user can only claim the interchange discount if he boards a bus that is known to stop at or near the station which he exited from. Similarly bus-bus interchanges may be allowed between buses at all times, or restrictions may be in place such that for each bus line, interchange is permitted only to a given set of intersecting bus lines. Bus-rail and bus-bus interchanges may be free or may result in an interchange surcharge. The interchange surcharge may be a fixed cost that depends on the type of interchange only (e.g. bus-bus or bus-rail). The interchange surcharge may optionally be dependent on the bus line used. The interchange surcharge may optionally be dependent on the boarding or alighting location on the bus leg of the combined journey (e.g. zone as determined by GPS coordinates, fare stage) An interchange interval is defined. This is time limit within which an interchange must occur in order to qualify. An upper bound on the number of consecutive	The fare engine defines bus-rail interchanges so that users changing from bus to rail and vice versa will be considered to have made one continuous journey and be charged one fare. No bus-bus interchange is defined or allowed. Bus-rail interchanges incur a fixed cost with no dependence on the bus line. An interchange interval is defined. This is time limit within which an interchange must occur in order to qualify. A limit of 1 bus-rail interchange is permitted per linked journey. In other words, a bus journey segment followed by an underground segment and a second bus segment would *not* be a valid linked journey. In this case the first bus journey segment would be admitted into a linked journey alongside the underground journey. The second bus journey would constitute its own linked journey and incur a single bus fare.

bus-bus and bus-rail interchanges in one journey may be imposed. This limit may be imposed globally.	

4.9 *Period Products*

Period products give users unlimited use of a certain part of the system, subject to conditions attached to the product purchased. Period products as currently available at TfL are described in section 3.5.

TfL currently offers two types of period products – tickets and caps. The contactless bankcard fare engine will only support tickets. The best value functionality currently fulfilled by caps will be met using standard tickets.

General Requirements	Prototype Requirements
Users may hold any number of period tickets. Any number of period products could be simultaneously applied to a journey (if applicable)	Users may hold only one period product at a time. Only one period ticket maybe applied to a journey at a time.

4.9.1 Best Value

Best value is defined as a guarantee to users that they will not be charged more in single fares than it would have cost them to purchase the most cost-effective period ticket for their usage scenario.

General Requirements	Prototype Requirements
Best value should be an option for all period products.	Best value is implemented only for daily tickets.

4.9.2 Modes

General Requirements	Prototype Requirements
The fare engine should support tickets for all station-based modes and on-board validation modes for which single fares are supported. The fare engine should support tickets that apply to any individual mode as well as any combination of modes. The fare engine should fall back to single fares when the product held is not valid for the mode the user is attempting to use. E.g. user holding a bus-only period ticket tapping in at an underground station. Where it is not possible to unambiguously identify the mode being used from information provided by contactless bankcard transactions, the fare engine should assume eligibility and leave enforcement to fare inspection. E.g. User with a (hypothetical) DLR only ticket tapping in at a shared underground/DLR station.	The prototype will focus on tickets for TfL station-based modes only. Specifically, LUL and DLR.

4.9.3 Time bands

General Requirements	Prototype Requirements
A ticket may exist for any number of arbitrarily defined time bands. These time bands do not have to be the same as single product bands.	Same as current TfL period product time bands. • **Peak:** 4:30am – 9:30am weekdays • **Off-peak:** After 9:30am weekdays and weekends

4.9.4 *Discount Groups*

General Requirements	Prototype Requirements
Tickets should be available for any arbitrary number of discount groups, including custom group fares.	Tickets support the same discount groups as supported by single fares. • Adult • Child • New Deal

4.9.5 *Zonal Validity*

The notion of zonal validity in period products is discussed in section 3.5.2. To review, the user is granted unlimited travel within the zonal validity of a ticket he holds.

General Requirements	Prototype Requirements
Any number of fare zones and the full combinatorial set of zonal validities.	The 8 zonal validities defined in the current TfL fare structure for zones 1-6. • Zone 1-2 • Zone 1-3 • Zone 1-4 • Zone 1-5 • Zone 1-6 • Zone 2 • Zone 2-3 • Zone 2-6

4.10 *Other Services*

As a detailed security analysis lies outside the scope of this thesis, we will stop at restating the more important measures currently found in Oyster. These services are described in section 3.8.

General Requirements	Prototype Requirements

• Incomplete Journeys
  o Unfinished Journeys
  o Unstarted Journeys
• Passback
• Continuation Rail Exit
• Maximum Trip Length
• U-Turn Rail Journey
• Automatic resolution of incomplete journeys
  o Auto Continuation
  o Auto Completion
• Aliasing
	• Incomplete Journeys
	  o Unfinished Journeys
	  o Unstarted Journeys
	• Maximum Trip Length

5 TfL Future Ticketing Architecture

Figure 5.1 highlights the high-level design of the fare engine proposed in this thesis, and its role within the context of TfL's contactless bankcard fare collection ("Future Ticketing") system. The architecture of the fare collection system as a whole is under ongoing development by TfL's Future Ticketing group and the representation shown below is a only a general, conceptual representation of TfL's plans.

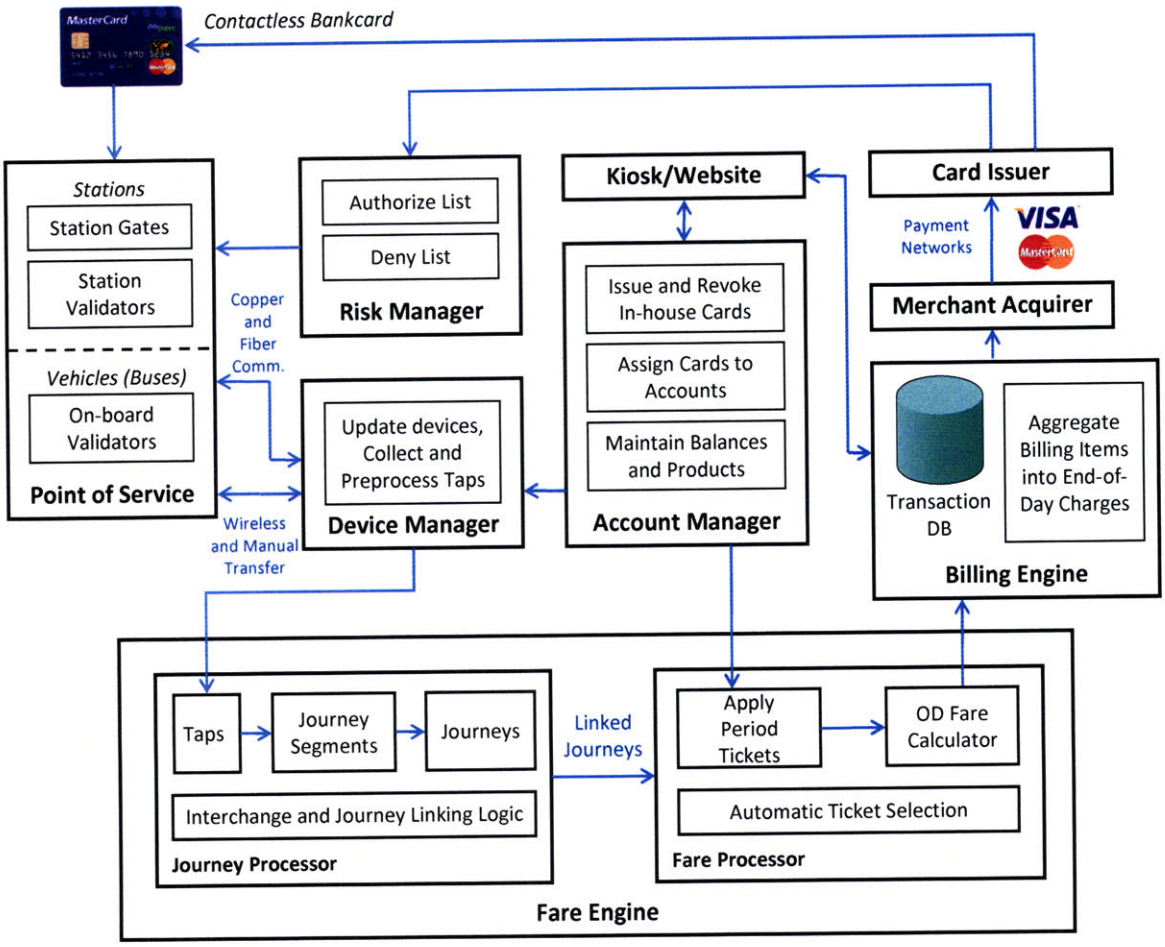


Figure 5.1 Fare Engine System Diagram

As a detailed description of the unified fare collection-bankcard payment processing system lies outside of the scope of this thesis, the components of Figure 5.1 outside of the *fare engine* box will only be cursorily examined.

This system can be divided into the financial entities, customer interfaces and internal subsystems.

- **Financial Entities** - These are organizations, networks or instruments external to TfL which are involved in the payment processing. One of the major motivations for the adoption of contactless bankcards is to insulate TfL from the cash handling and financial services it must currently provide to support the Oyster system.

- **Customer Interfaces** – These are customer facing aspects of the system that form a gateway between what the user (rider) sees and the internal complexity of the underlying system. If we apply the model-view-controller (MVC) framework to our fare collection system, these customer interfaces constitute the 'view' of the system. This view consists of both physical infrastructure (e.g. gates, validators and kiosks) as well as services such as a website or ticket booth agents. Customer interfaces are distinguished from internal aspects of the system in being a presentation layer, that is, they do not themselves implement any business logic or processes.

- **Business Subsystems** – These are the internal components of a fare collection system which are not visible to the user. Within the MVC framework, these constitute the 'models' and 'controllers' of the fare collection system. Specifically, the models consist of account information stored in the account manager and journey history stored in the transaction database of the billing engine, while the controllers consist of the business logic inside the engines and managers that describe fare processes. Note that this is a logical division only, and not necessarily a representation physical implementation. A business subsystem could take the form of multiple physical servers. It could be comprised of different pieces of software operating on the same machine, or multiple components of one software application. Parts of a subsystem may even be delegated to devices in-field.

5.1 *Financial Entities*

5.1.1 Contactless Bankcard

A contactless bankcard is a standardized bank card taking the form of an RFID smartcard. Contactless bankcards are now supported by international electronic payment networks such as Visa (PayWave), Mastercard (PayPass), and American Express (ExpressPay). Within North America, contactless bankcards provide a direct replacement for the de-facto standard of magnetic stripe credit and debit cards. Within continental Europe and the UK, where EMV (Chip and Pin) bank cards are

universally adopted, contactless bankcards will function within the security framework and restrictions of the EMV standards. A more detailed discussion is found in section 1.2.

5.1.2 Merchant Acquirer

The merchant acquirer is a vendor which acts as an agent to the bankcard network for the merchant (in this case, TfL) and takes care of converting bankcard transactions into monetary payment. From TfL's perspective, sales are transmitted to the merchant acquirer on a periodic basis, and funds are deposited for those transactions into TfL's bank account. The merchant acquirer also handles the verification of cards and authorization of purchases. Bankcard transactions are discussed in section 1.2.4.

5.1.3 Card Issuer

This is the bank or other financial institution underwriting the contactless bankcard. Issuers have no direct relationship with the merchant (TfL), except as a possible source of risk management data.

5.2 *Customer Interfaces*

5.2.1 Point of service

This encompasses all physical infrastructure which directly interacts with contactless bankcards, such as fare gates and validators. In retail terms, such devices are also referred to as point of sale (POS) devices. A fare gate is a mechanical barrier taking the form of a turnstile or pneumatic or electromechanical paddles which permits passage of individual passengers only when presented with acceptable authentication/authorization, such as a valid ticket, stored value smart card or contactless bankcard. A validator, in contrast, allows for the validation of tickets or smart cards without the ability to deny access. Note that a *reader* refers to the pad-shaped electronic sensor that communicates with the bank card, and is found on both fare gates and validators. In particular, a smartcard reader is distinguished from a fare validator in that the reader is only one component of the validator, which also contains other systems for communications, card validation and visual and aural feedback, etc.

5.2.2 *Website/Kiosk*

Under future ticketing, user information kiosks at stations will replace the classes of fare vending devices currently known as *ticket vending machines* (TVM) and *multi-fare machines* (MFM). Users will be able to purchase period products and inspect the past usage of their account as well as estimated and final charges.

This functionality will also be available on a website which users can access at their own leisure. In fact, station kiosks, the card management website and the internal customer relationship management system can be driven by variations of the same software.

5.3 *Internal Subsystems*

5.3.1 *Device Manager*

The role of the device manager is as follows.

1. **To collect and pre-process taps** - Gates and validators at fixed stations may communicate with the fare collection center using a combination of dedicated copper and fiber links as well as public IP networks, while vehicle mounted validators may use a combination of wireless communications such as 3G or GPRS and manual transfer at depots using data keys. Even over the same type of physical network, data from stations may be transmitted using a variety of data protocols and formats depending on the equipment used. Furthermore devices may communicate as a solitary unit or through a station-based intermediate server which bundles transactions from all gates and validators at the station.

2. **To provide a uniform interface to other services** - The *device manager* insulates the fare engine from having to deal with the different physical implementations of station equipment and station-center communications. To do so device manager provides a normalized feed of bankcard interactions (taps) through an application programming interface (API). By communicating with the account manager, taps can be defined in terms of user accounts rather than raw bankcard numbers.

3. **To configure and maintain gates and validators** – The device manager distributes firmware upgrades to in-field devices. Configuration and settings can be uploaded from a

centralized location. The implementation of certain business functions, such as hotlists or blacklist may be delegated to individual validators and gates, and the updates to these functions may also be dispensed through the device manager. The device manager monitors the health of readers, and possibly gates, fare boxes and validators, reporting alarms and failures to a central monitoring system.

5.3.2 Risk Manager

The risk manager is responsible for maintaining the integrity of the fare collection system by restricting the acceptance of high-risk bankcards. It implements a deny list through information collected from a wealth of sources. These may include the billing engine (representing cards which TfL has had difficulty obtaining authorization for in its own experience) as well as higher level 'known fraudulent' card lists obtained from the merchant acquirer, the bankcard networks, or from individual issuing banks themselves.

5.3.3 Account Manager

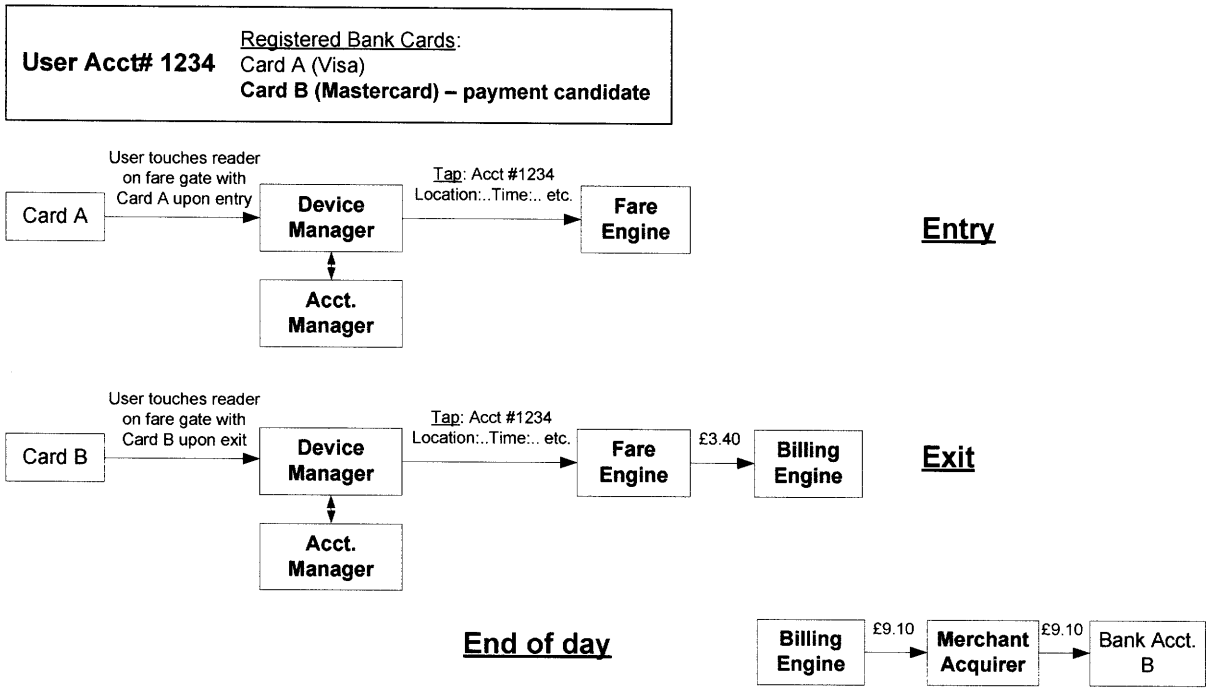


Figure 5.2 – Multiple bankcards associated with the same user account.

Conversely, the risk manager may hasten the processing of known reliable bankcards by compiling an 'accept list' of trusted cards based on user pre-registration and individual account histories. The

exact scope and capabilities of the risk manager remains under development and negotiation by TfL with its vendors and financial stakeholders.

The account manager is responsible for maintaining user accounts. This includes the creation, editing and deletion of accounts. An example of sophisticated capabilities that an account manager can provide, beyond basic account management, is shown in Figure 5.2.

By permitting the assignment of multiple bankcards to one user account, the transactions as shown in Figure 5.2 become possible. Here, the user enters the system by tapping on the gate with bank card A, and leaves the system by tapping out with bank card B. However because taps are pre-processed in consultation with the account manager, only the account number is presented to the fare engine. This enables an otherwise invalid transaction sequence involving two different bankcards to be accepted and processed. At the end of the day, billing engine charges the final amount incurred to the appointed payment candidate, in this case Card B.

5.3.4 Billing Engine

The contactless bankcard purchase process involves the following steps:

1. **Authentication** – This occurs between the contactless bank card and the reader at the time a bankcard is presented and does not involve either the Merchant Acquirer or the Billing Engine. This option is available only for European (EMV) contactless bankcards. In North America, in the absence of EMV, authentication is done online by querying an acquirer or bank card association server and is usually considered part of the authorization process.

2. **Authorization** – Contactless bank cards which cannot be authenticated (such as the North American variety) or for which authentication has been temporarily disabled (EMV cards which have seen sustained use in contactless mode) require authorization. This process is performed against the merchant acquirer. During authorization the validity of the bank card and the availability of funds are assured and a given amount may be set aside for a future transaction, without the amount being charged. Authorization is a contract for payment up to the amount authorized for, although the eventual sale does not have to utilize the full amount.

3. **Aggregation** – Billing items may be aggregated over a day, or even many days resulting in a larger amount which is charged as one single purchase. Depending on contractual terms this could result in lower processing fees for TfL.

4. **Capture and Settlement** - The capture and settlement process also occurs between the billing engine and the merchant acquirer. Final sale amounts are transmitted to the merchant acquirer in exchange for funds being deposited in TfL's bank accounts.

The role of the billing engine is to buffer and aggregate 'billing items' until settlement with the merchant acquirer can occur. The billing engine acts as the fare engine's exclusive interface to Merchant Acquirer, who, in turn, acts on TfL's behalf in transacting with the bankcard payment networks.

The fare engine will notify the billing engine of each billing item as soon as it becomes known. Immediate authorization for these items may be necessary depending on circumstances as dictated by transaction rules. At other times, billing items may be left to sit in the billing engine until they are authorized in a batch prior to settlement. Regardless of whether billing items have been authorized they should be available to users and staff for viewing immediately after being received by the billing history. Entries in the transaction database of the billing engine is what the user sees when he looks up his journey history on a website or at a kiosk.

5.3.5 Fare Engine

The role of the fare engine is to convert bank card-reader interactions in the form of 'taps' collected from throughout the system into 'billing items' which can be charged to a user account. The fare engine obtains its input taps from the device manager. The fare engine is informed of the ticket holding status of a user either through querying the account manager or through a publish-and-subscribe interface where the account manager posts or notifies the fare engine of account status, rather than requiring a query each time. Once a fare has been computed, a billing item is generated and dispatched to the billing manager in a 'charge-and-forget' operation. This means that as far as the fare engine is concerned, a journey has been 'paid for' as soon as it is sent to the billing engine. It does not matter to the fare engine how long it takes before the billing item is formally authorized, charged or settled.

6 Fare Engine Design

The design and development of the fare engine is the main thrust of this thesis and will be elaborated to greater detail in the sections below.

6.1 *Design Criteria*

This is a reiteration of the main design requirements and criteria for the contactless bankcard fare engine.

1. Input in the form of 'taps'. Each tap contains information such as the identification of the user (through his bank card and the account manager) and the circumstances of the transaction (e.g. timestamp, location, direction).

2. Output in the form of 'Billing Items' consisting of a pound amount to be charged to the user's account and a description of the charges.

3. Support for directional zonal fares and route-dependent zonal fare calculation, possibly varying by time, as well as flat fares, possibly varying by time. Support for period passes of varying lengths, with possible time restrictions. Support for best value calculations.

4. Support for trip linking, including intermediate validation, out-of-station interchanges and cross-mode interchanges.

5. Efficient, real-time conversion of taps to billing items while allowing for out-of-sequence arrival and reinterpretation of taps

6.2 *Fare Engine Organization*

With reference to the bottom of Figure 5.2, the fare engine can be broken down into two sub-components.

- **Journey Processor** – The journey processor converts taps into linked journeys in a manner consistent with zonal and journey linking rules, without consideration of any fare or ticketing aspects.

- **Fare Processor** – The fare processor assigns a fare for a given journey on the basis of applicable fare and ticketing rules.

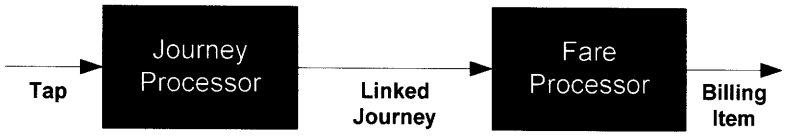


Figure 6.1 – Fare engine data flow.

Subcomponents are organized as black boxes such that the internal operation of the journey processor is invisible to fare processor, and vice versa. To complete this encapsulation, data is exchanged between subsystems in a self-contained data flow. These objects are defined below.

6.3 *Fare Engine Data Flows*

The fare engine has three main data flows. Each is represented by a Java object class.

- **Tap** – Transactions from the point of sale (stations and buses) are collected, pre-processed and introduced to the fare engine as Tap objects. The component of the fare engine that deals specifically with Taps is the journey processor.

- **LinkedJourney** – Linked journeys are the output of the journey processor and the input of the fare processor. The LinkedJourney object represents this flow between the two modules of the fare engine. A linked journey represents a complete and billable unit of travel. Linked journeys are described in requirements section 4.8.1.

- **BillingItem** – Billing items are the output of the fare processor and of the fare engine as a whole. This data flow conveys the final charge to be made to a user account in response to a journey made. The BillingItem object represents this data flow.

All three fare engine data flow classes described above are immutable. This means that once created, their contents cannot be changed. They can only be abandoned and recycled. Two layers of protection are used to enforce the objects' immutability. First, all fields are declared final so they can

only be initialized at creation by the construction. Second, all fields are declared private[14] and accessed through accessor methods (getters).

6.3.1 Tap

Tap objects are relatively short-lived. The journey processor readily clones any Tap object it receives into an internal representation.

Java offers multiple sizes of integer primitives, including short (16 bit), int (32 bit) and long (64 bits). Where a data type of 'Integer' is specified in the tables below, it may be a short, regular or a long integer, depending on need.

Field	Data Type	Description
UserID	Integer	Unique identifier of user account. Accounts may correspond to one or more contactless bankcards. If an unregistered bankcard is presented, the account manager will create an account ID before the Tap is passed to the fare engine.
TransactionTime	Timestamp	Timestamp of the transaction, precise to the nearest second with respect to the local clock. Gates and validators will have clocks which are synchronized on a regular basis. The standard Java library has no 'Timestamp' class, and although a Date class is provided, timestamps are most efficiently stored in Java simply as *long* primitive.
LocationID	Integer	Unique identifier of a transaction location. For station-based transactions this ID will identify the station, or part of a station where a station consists of disjoint gated areas. For on-board transactions this will identify the bus

[14] In Java, private fields are hidden from all external objects not of the same class, including subclasses.

		route. Note that the transmission from the device may pinpoint location with greater specificity (e.g. the specific gate or validator device the transaction occurred at). However this specific information is not visible to the fare engine. It is removed by the device manager and mapped to station level identification.
TransactionType	Integer	For station based transactions, this is a code identifying the direction of travel through a fare gate. For undirected validator transaction, this identifies whether the validator is located inside the secured paid area of a station (if any) or located in a freely accessible location. For on-board transactions, this field may be used to identify whether the transaction occurred at boarding or exit. In the prototype boarding transactions are always assumed. Current TfL operation practice provides no mechanism for distinguishing boardings from alightings. **Values:** ENTRY (entry through gate) EXIT (exit through gate) IPVAL (internal validator) EPVAL (external validator) BUS (on-board validation)
GPSLocation	*Location*	*This field identifies the transaction location by means of AVL. Location may either be GPS coordinates or a stop ID. It is not implemented in the prototype.*

Table 6.1 – Fare engine data flow – Tap.

6.3.2 *LinkedJourney*

The LinkedJourney object is a physical representation of a *linked journey* as defined in requirement section 4.8.1. Logically, linked journeys are defined as a sequence of journey segments that constitute a notional trip. LinkedJourney objects are merely a parent container for JourneySegment objects. A LinkedJourney cannot be viewed independently of its JourneySegment children.

Field	Data Type	Description
UserID	Integer	Unique identifier of user account. Copied directly from UserID of *Tap* objects used to create this LinkedJourney.
NodeList	ArrayList<JourneySegments>	This is a collection of JourneySegments that make up this linked journey. An ArrayList, which is a relatively efficient implementation of dynamic list, is used to maintain the sequence of journey segments. ArrayList is part of the Java collections library.

Table 6.2 – Fare engine data flow – LinkedJourney.

6.3.2.1 *JourneySegment*

A JourneySegment object represents one journey segment in a linked journey. JourneySegment objects are self-contained. In other words, they carry their own location and timestamp values as primitives and do not reference any tap objects.

JourneySegment objects must be visible to users of LinkedJourney. Therefore JourneySegment cannot be an inner class of LinkedJourney. However it is only intended to be instantiated from inside the LinkedJourney class.

JourneySegment objects are immutable. Note that JourneySegment objects are exclusively owned by their parent LinkedJourneys and not shared. Therefore there is no need to store UserID.

Field Name	Data Type	Description
StartTime	Timestamp	Start time of journey segment. This is the timestamp of the first (or only) tap used in building the journey segment.
EndTime	Timestamp	End time of journey segment. This is the timestamp of the last tap used in building the journey segment. Where a journey segment is an bus segment the EndTime is undefined.
StartLocation	Integer	Start location of journey segment. This is the location of the first tap used in building the journey segment. For a bus segment StartLocation is a route.
EndLocation	Integer	End location of journey segment. This is the location of the last tap used in building the journey segment. For a bus segment EndLocation is undefined.
Type	Integer	The JourneySegment type. Accepted values are the constants: • ONBOARD • STATIONBASED

Table 6.3 – Fare engine data flow – JourneySegment (only referenced by LinkedJourney, not standalone)

Note that given two back-to-back JourneySegments we can easily infer the type of interchange that links them. There is no need to store this information separately.

- If either one of the two journey segments is of type ONBOARD, the interchange linking them must be a *cross-mode interchange*, also known as a *bus-rail* interchange.

- If both billing segments are STATIONBASED and the two StationLocations at the junction (BillingSegment1.EndLocation and BillingSegment2.StartLocation) are the same station, then the interchange must be an *intermediate validation (IV).*

- If both billing segments are STATIONBASED and the two StationLocations at the junction are not the same, the interchange must be an *out-of-station interchange.*

6.3.3 BillingItem

The BillingItem object represents the cost associated with a linked journey. The originating linked journey is included with the BillingItem.

Field	**Data Type**	**Description**
UserID	Integer	Unique identifier of user account. Copied directly from UserID of the originating *LinkedJourney.*
NodeList	ArrayList<JourneySegments>	The NodeList and its children JourneySegments are not modified when a LinkedJourney is converted into a BillingItem. The NodeList of the input LinkedJourney is reused directly (passed by reference to the output BillingItem.) This eliminates the need to clone and recreate the NodeList (an ArrayList) and its JourneySegments.
Fare	Integer	Billable charge for this journey in pence. This charge accounts for any period tickets held by the user as well as

		automatically assigned by the fare processor.
RevenueAllocationCode	Integer	Identifies revenue allocation particulars for this linked journey. See section 8.4.2 for a discussion of revenue allocation.
DiscountGroup	Integer	Code for single product discount group that was used to determine the fare for this billing item. The discount group of a user account may change over time (a student may graduate, or a former beneficiary may lose his status). The discount group used to calculate the price of each specific billing item is recorded for later auditing. Note also that this is the *single product* discount group and it may be different from the discount group of any tickets used in conjunction with the fare. For example, the holder of an 18+ ticket would incur extension fares charged at the adult rate. The discount group of any ticket used is folded into the TicketID fields.
TimeBand	Integer	The single-product time band used for this particular billing item.
PrePurchaseTicketID	Integer	If a user pre-purchased has been applied to this linked journey to reduce or eliminate the fare, it is identified here. If

		no ticket has been applied for this trip, this field takes a null value. Recall that each period ticket is identified by the following properties (decision tiers). The multiplicity of each tier in the context of the fare engine prototype is enclosed in parentheses. • Discount Group (3) • Validity Period (5) • Zonal Validity (8)
AutoSelectTicketID	Integer	If an automatically selected ticket has been applied to this linked journey to reduce or eliminate the fare, it is identified here. If an automatically selected ticket is not used, this field takes a null value.

Table 6.4- Fare engine data flow – billing item.

6.4 *Fare Engine Models*

6.4.1 Oyster Card - Stateless Fare Engine

The existing Oyster smart card fare collection system can be seen as a *stateless fare engine* acting on a small set of *state bearing registers*. A stateless system does not remember past inputs and executes its actions depend strictly on the inputs presented to it at a given instant. In this framework, the stateless fare engine is manifested in the gate and validator devices, while the state bearing registers take the form of an Oyster smart card which is freely readable and writeable by gates and validators. Note that statelessness applies with respect to the fare engine only. The fare collection system as a whole is stateful; however state information is carried on Oyster cards which are an external input to the fare engine. The use of discrete state variables in this way is suited to the limited storage available on each Oyster card.

One consequence of the decentralized, stateless architecture of the Oyster fare engine is that transactions are guaranteed to be processed in the order they are made. The fact that fare is calculated and deducted directly from the smart card at the point of interaction, and that the smart card follows the user as he travels ensure sequential synchronization between the occurrence and processing of transactions.

This model of processing does not use a data link in real-time for the processing of normal transactions. However a data link is needed for maintenance operations. This includes auditing of transactions, transfer of hotlists and the distribution of firmware and data table updates.

6.4.1.1 Transaction Sequencing

In contrast, synchronization between event and processing is lost in a contactless bankcard fare collection system. Here, bank cards are little more than an identifying token, and the fare engine has been physically relocated from the gate/validator to a centralized site. Under these circumstances, the time of processing is dependent on the latency and reliability of the communication link between each site and the fare management center (FMC). Temporary outages may cause a transaction to be delayed. By the time it arrives at the FMC, a transaction that took place after the delayed transaction may already have arrived. This poses a challenge, as the core principle of fare computation in a fare engine is to assemble a sequential series of taps into billable journeys. That taps are presented in sequence is an assumption that must be met if the current Oyster journey forming methodology is to be adopted.

One approach to maintaining the correct sequencing of transactions is to use a tracking number physically written on the contactless bankcard which is incremented after each transaction. Gaps in sequence numbers accompanying each incoming tap allow taps delayed in transmission to be distinguished from those that were never made. With the knowledge of which taps have been delayed, appropriate gaps are left in the sequential table of taps and the processing of journeys containing gaps is delayed until the gaps are filled. This approach mirrors the mechanism currently used to maintain the integrity of the Oyster audit database, which is a database of transactions maintained by TfL for auditing and research purposes. However this solution is not practicable under our assumption of no writeable registers on the contactless bankcard.

6.4.2 *Contactless Bankcard - Batch processing*

In Table 6.5, two alternatives for addressing transaction sequencing without writeable bankcards are presented, alongside the status quo of Oyster. The simpler approach, called batch processing, involves buffering transactions (Taps) as they arrive throughout the day in an unsorted, stream based data store, with no further action until batched processing is initiated at the end of a day. During batched processing, all received taps for a given user are sorted in the order of their timestamps. This process restores the correct sequencing of any out-of-order and delayed taps. At this point, the sorted taps can be processed sequentially using user state variables and a stateless fare engine, in the same fashion as the Oyster fare engine. Advantages of this approach include simplicity, ability to reuse existing logic (such as parts of the Oyster Central System), and efficiency. The average case running time of sorting small, nearly-sorted lists items approaches linear time O(n). The chief disadvantage is the unavailability of real-time usage information. Under a strict batch processing system there is no way for users or TfL view charges as they accrue during the day. This limitation also precludes the option of requesting authorization for high-value trips to limit the financial risk stemming from fraudulent or over-limit cards.

One possible compromise to strict batch processing is to perform preliminary journey calculations using a stateless engine based on taps as they arrive at the server, disregarding any Taps which fall out of sequence (as evidenced by incongruent timestamps). Batch processing is then performed to obtain final billable journeys and at this point out-of-sequence taps are reincorporated in place and accounted for. While this approach will bring some level of real-time information to the user and TfL, the fact that missing taps are not reincorporated until batched processing means that displaced taps will cause erroneous information to be displayed for the remainder of the day, or longer. The prolonged presentation of inaccurate and confusing information is undesirable. Another compromise is to re-sort transactions and re-run the sequential stateless fare computation each time an out of order tap is detected. Such an approach introduces additional complexity and more importantly, is clearly inefficient as each instance of an unsorted tap would require a complete recalculation of the current day's journeys and fares.

6.4.3 *Contactless Bankcard - Dynamic Object Oriented Data Structure*

In a second, significantly different approach, we dispense with the Oyster model of applying a stateless fare engine on a limited set of user state variables. Instead, a new fare engine is designed

from ground up with built-in support for out-of-order transactions. All recent transactions are retained in memory in a dynamic, object oriented data structure. As taps arrive they are dynamically incorporated into the data structure. No notion of tap sequence is maintained; therefore out-of-sequence taps are given the same treatment as those which arrive in sequence. Performance is optimized for insertion of new taps as well as the detection of linked journeys.

Taps are interpreted into linked journeys in real time. As additional taps arrive (in sequence or otherwise), this interpretation may change. This capability is accomplished by a data structure which reconfigures itself with incoming taps to maintain consistency with tap linking and journey linking rules defined in accordance with the fare description.

Under this regime, the fare engine maintains state in the form of a dynamic data structure of recent trips. This structure represents the best interpretation of journeys undertaking, as derived from information known at a particular point in time. While incorrect estimates may still be displayed to the user as a result of delayed transactions, the dynamic data structure ensures that corrections are posted without further delay as soon as the missing information comes to the server's knowledge. This real time capability also provides TfL with the option of obtaining near-real time authorization for the exact price of journeys as soon as they're completed. While this option may not be exercised in the event that authorization for some larger amount has already been obtained at the start of a journey (for example, authorizing the daily maximum on tap-in), it could prove useful in the case of high-value national rail journeys, as well as cases where the pre-authorization of a larger-than-necessary amount is objectionable on an argument of equity.

Operations on this dynamic data structure can be divided into two broad classes:

- **Bottom up** – Taps are inserted at the bottom of the stack reflecting real-time activity. These taps are sequenced, and their implications for journey identification and linking are propagated upward.
- **Top down** – Linked journeys are identified at the top of the stack and sent to the fare processor for fare calculation and conversion into billing items. At the end of the day when billing items go into settlement (and are therefore finalized), linked journeys are withdrawn from the data structure, again from the top down. When a linked journey is removed, corresponding journey

segments and taps must also be extracted. This mechanism prevents the memory from being consumed without bound.

The structure and operation of this dynamic object oriented fare engine will be explored in greater detail in the remaining sections of thesis.

6.4.4 Comparison of Fare Engine Models

	Oyster Card	**Contactless Bankcard (Batch processing)**	**Contactless Bankcard (Dynamic processing)**
Processing paradigm	Decentralized – transaction processed directly at the fare gate or validator against the presented smart card.	Centralized – transactions transmitted to a central server where processing occurs.	
Funding model	Stored-value on card. Funds topped up at user's convenience. Maximum fare deducted on entry and difference refunded upon exit.	Fares billed to bankcard. Funds may be authorized at the time of travel however final settlement occurs in bulk at some prescribed interval (e.g. at the end of each day).	
Card storage	Smart card constitutes fare instrument in and of itself. Monetary value is stored on the smartcard (stored value) and deducted during use.	Contactless bankcard used only as identifying token. No account or balance information is stored on card.	
Transaction records and trip history	Limited history stored on card. Transactions archived for auditing and research purposes only.	Server stored transaction record used directly for billing.	

	Transaction database not used immediately in direct operations.		
Processing expediency	Fares are charged and deducted immediately at the fare gate or validator. Users may inspect smart card balance and recent usage history at any time by presenting the smartcard at a MFM.	Fares are not calculated until time of processing. Charges and trip history cannot be viewed until processing has been completed (e.g. the next day).	Journeys and fares are calculated in real-time with available information. Tentative trip history and charges may be disclosed to the user at kiosks and over the internet, however these may be inaccurate and are subject to changes due to delayed arrival of transactions.
Sequential transaction ordering	Sequential ordering of transactions is required and guaranteed as a necessary consequence of system design. Decentralized fare calculation on each smartcard at the point of contact ensures that transactions are processed in the order of travel.	Sequential ordering of transactions is required and is provided through time-sorting of transaction records during batched processing. In other words, taps are numbered sequentially before batched processing.	Sequential ordering not required. Dynamic data structure accepts and incorporates out-of-sequence transactions seamlessly as they arrive. Journey interpretation and charges may change as a consequence.

Missing taps	Missing taps are closed out on the next transaction. Delayed but not missing transactions are not possible under the decentralized model.	Delayed taps can be collected up to time of batched processing. Those still not received at the time of processing (never made) are declared missing.	Delayed taps are incorporated into the dynamic data structure as long as its context remains in memory.
Data link requirements	Low integrity and latency requirements. Transactions transmitted for auditing and research purposes only.	Transaction data not required until time of batched processing. High integrity and low latency requirements.	Delays prevent the provision of accurate real-time usage information and fare estimates. High integrity and medium latency requirements.
Intermediate validation	No support	Full support	
Out of station interchange	Yes	Full flexible support	
Period tickets	Ticket products stored on smartcard	Ticket products stored on user accounts on server.	
Consumer best value	Best value implemented via capping with limitations	Automatic ticket selection provides a flexible mechanism for merging single fares with period products.	

Table 6.5 – Comparing fare engine design paradigms.

7 Journey Processor

As illustrated in Figure 5.1 and Figure 6.1, the *journey processor* forms the first stage of a fare engine. This journey processor is underpinned by a three-tiered data structure motivated by the three successive logical representations of a journey. Each tier is represented as an object class. The three classes are:

1. Tap
2. Journey Segment
3. Linked Journey

Journey Segment takes the role of an intermediate class which isolates taps from linked journeys. In other words, as far as taps are concerned, linked journeys do not exist, and as far as linked journeys are concerned, taps do not exist. This vertical abstraction reduces the complex problem of producing billable linked journeys from individual taps into two smaller, *independent* problems.

7.1 *Internal Journey Processor Objects*

Only data fields are included in the description of the three journey processor object classes below. Class methods are described in detail in subsequent discussions.

Internal journey processor classes are visible only in the journey processor package. Owing to the internal nature of these classes, we have elected to allow direct access of member values, rather than formal encapsulation with getters. Immutability will be enforced only through the use of the final keyword.

7.1.1 JPTap

The *internal journey processor tap* is structurally identical to the *external fare engine tap* as described in section 6.3.1, with the addition of some references fields. To differentiate this internal tap, which is not visible from outside the journey processor black box from the external tap, which is visible outside, we will call this tap the *JPTap*.

Field	Data Type
UserID	final Integer
TransactionTime	final Timestamp
LocationID	final Integer
Type	final Integer†
LeftSegment	*JPJourneySegment[15]
RightSegment	*JPJourneySegment
Next	*JPTap
Prev	*JPTap

Table 7.1 – JPTap data and reference fields. †For tap types constants see sec. 7.1.1.1.

JPTap objects are linked laterally it to each other in a doubly linked list to allow traversal and the inspection of adjacent elements. This is achieved by means of the Next and Prev references, and will be discussed in further in section 7.4.1.

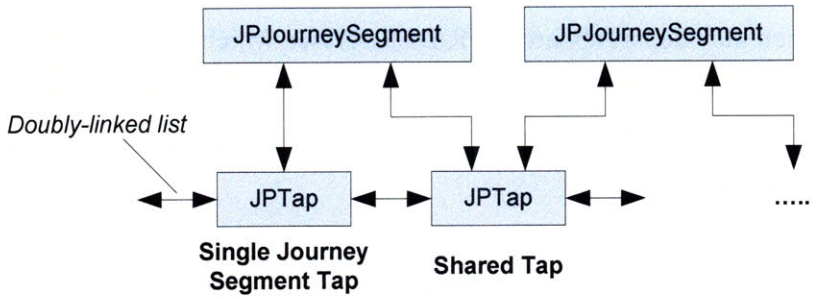


Figure 7.1 – A JPTap may be linked to one JPJourneySegment or shared amongst two JPJourneySegments. A JPJourneySegment must link to two station-based taps.

LeftSegment and RightSegment are reverse references to the JPJourneySegments that reference a given JPTap. LeftSegment refers to a JPJourneySegment that claims this tap as its EndTap, while RightSegment refers to a JPJourneySegment that claims this tap as its StartTap. Either or both LeftSegment and RightSegment may be assigned depending on whether this tap is claimed by one or two JPJourneySegments. If a JPTap is not claimed by any JPJourneySegment, then both fields would be null. If a BUS tap is claimed by a JPJourneySegment, both back-references will point to the same JPJourneySegment.

[15]An asterisk predicating a data type indicates a reference to an object of the specified class.

Invariant[16]*: If TransitionType==BUS, LeftSegment=RightSegment*

The data of JPTap objects are immutable, however references may be changed.

7.1.1.1 Station Areas

Before describing the available type constants we will first establish the nomenclature of *secured* and *unsecured* areas. Secured area refers stations and services that cannot be reached from the street without passing through a fare gate, with some caveats. Conversely, unsecured area refers to stations and services which users may enter freely from the street without passing through any access control device, again with some caveats.

One of these caveats involves National Rail or DLR (NR/DLR) services which are typically ungated but bring users directly inside the paid area of interchange stations, such as West Brompton or Stratford. Users may arrive inside the paid area without going through a fare gate on board one of these services. Such users are said to be *injected* into the secured area. Similarly, users are said to be *extracted* from the secured area when they depart aboard a NR/DLR service which leaves from within the secured area, and alight at an ungated station. For the purpose of the ongoing discussion, NR encompasses London Overground services as well as Train Operating Company (TOC) services for which contactless bankcard payment is accepted.

The other caveat involves stations which are partially gated. That is, some entrances to the station have gates installed while others are ungated and equipped with validators. In this case, whether the station (and corresponding service) is gated or not varies for each user and depends on his or her method of access.

7.1.1.2 Tap Types

ENTRY and EXIT

> These are taps at a fare gate. Tap type is either ENTRY or EXIT depending on the direction of passage through the gate. Fare gates are installed in a fixed physical configuration and therefore gate passages can always be assigned a direction.

[16] An invariant is a condition which must hold true as an inherent consequence of the system design. Invariants may be asserted in code as an automatic mechanism to detect programming errors.

ENTRY and EXIT taps are *terminal taps* because they must be at either end of a linked journey.

EPVAL

EPVAL stands for external validator. These are undirected validator devices located outside of the secured area. Differentiating between whether a validator is located within or outside of the secured area provides us with an additional means of filtering out invalid journey segments. For example, an IPVAL followed by an ENTRY tap cannot constitute a valid journey segment.

EPVAL is a *dual use tap* as it can start or end a linked journey, as well as be used in the middle of one (for intermediate validation).

IPVAL

IPVAL stands for internal validator. These are undirected validator devices located within the secured area. IPVAL is an *interior tap* as it must occur in the middle of a linked journey (not the beginning or end)

An important point to note regarding IPVALs is that they are provided primarily for users that enter the system from National Rail holding a paper ticket. Such users emerge inside the secured area without having had any interaction at all with the fare collection system. It is therefore mandatory for them indicate their presence on an IPVAL.

On the other hand, IPVAL usage is optional for users who arrive on NR services having previously validated at an EPVAL. This is in contrast to the Oyster indications for validator usage, which stipulates that all trips begun at a validator must be ended at an validator.

- **Oyster** – Westferry (EPVAL) → Putney Bridge (EXIT) is an invalid trip. A valid trip going from Westferry to Putney bridge would be: Westferry (EPVAL) → Bank(IPVAL) → Putney Bridge (EXIT)

- **Contactless Bankcard** - Westferry (EPVAL) → Putney Bridge (EXIT) is a valid journey segment. The interchange at Bank is implied and does not need to be indicated explicitly.

Although not required for pre-validated trips where the user is injected from an ungated station into a gated station by a NR/DLR service, the fare engine will not be confused by an extraneous

IPVAL validation. An IPVAL validation under these circumstances would be treated as an intermediate validation, resulting in two journey segments being produced.

BUS (On-board)

BUS taps take place on bus mounted validators. As a form of on-board validation, these taps have special properties:

1. BUS taps have a one-to-one correspondence with journey segments. They cannot be shared across journey segments and nor can a journey segment accommodate any other tap once it is associated with a BUS tap.

2. The location property refers to a route identifier and not a station.

7.1.2 JPJourneySegment

Journey segments, as defined in requirement section 4.8.2, represent the *smallest measurable unit of travel*. Journey segments do not duplicate any data, but rather contain references to JPTap objects. Maintaining consistency of notation we will call this class JPJourneySegment. For a JPJourneySegment to exist both StartTap and EndTap must be assigned.

Field	Data Type
StartTap	final *JPTap
EndTap	final *JPTap
ParentLinkedJourney	*JPLinkedJourney

Table 7.2 – JPJourneySegment data and reference fields.

Invariants: StartTap != NULL AND EndTap != NULL AND ParentLinkedJourney != NULL

The StartTap and EndTap references cannot be changed after initialization. Note that all three fields must take on non-null values.

ParentLinkedJourney is a reference to the JPLinkedJourney that has claimed this JPJourneySegment. ParentLinkedJourney must be non-null because every JPJourneySegment must be claimed by one and only one JPLinkedJourney.

7.1.3 *JPLinkedJourney*

While external fare engine linked journeys[17] always represent billable units of travel, the same is not true for the internal journey processor version. The internal linked journey is a top-level unit of travel which is subject to lateral merging. They are potentially, but not always billable entities. To distinguish internal linked journeys from external linked journeys we will call this class JPLinkedJourney. For a JPLinkedJourney to exist both StartSegment and EndSegment must be assigned.

Field	Data Type
StartSegment	*JPJourneySegment
EndSegment	*JPJourneySegment
Posted	Boolean
Fare	Integer

Table 7.3 – JPLinkedJourney data and reference fields

Invariant: StartSegment != NULL AND EndSegment != NULL

JPLinkedJourney can, and often will contain only one JPJourneySegment. In an JPLinkedJourney that contains multiple JPJourneySegments, references are maintained only to the first and last JPJourneySegments in the chain.

Within this prototype there is no upper bound on the number of station-based journey segments in each linked journey. In the general model our structure allows for multiple on-board validation segments. However, the logic to implement bus-bus interchange while preventing abuse could become complex. Therefore we have imposed a maximum of only one on-board validation journey segment in the prototype and it must either be the first or last journey segment within the linked journey.

The Posted flag indicates whether the linked journey has been sent to the fare processor for pricing and billing. The Fare field is the billed fare in pence.

[17] See section 6.3.2.

7.1.4 *Example configurations*

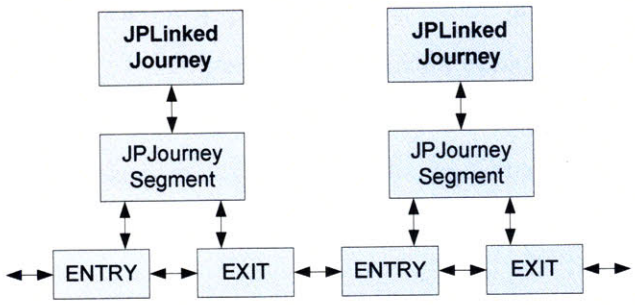


Figure 7.2 – Two 'standard' unlinked underground trips, e.g. trip to work in the morning and to return home in the evening. ENTRY and EXIT are JPTap objects.

We will illustrate the relationship between journey processor objects using a series of examples. In Figure 7.2, note that all of the JPTaps are claimed by only one JPJourneySegment. Which of a JPTap object's LeftSegment or RightSegment is filled depends on whether a tap is a StartTap or an EndTap. In this case, LeftSegment is unused in an ENTRY JPTap, while RightSegment is unused in an EXIT JPTap.

The following four examples show a JPLinkedJourney associated with multiple JPJourneySegments.

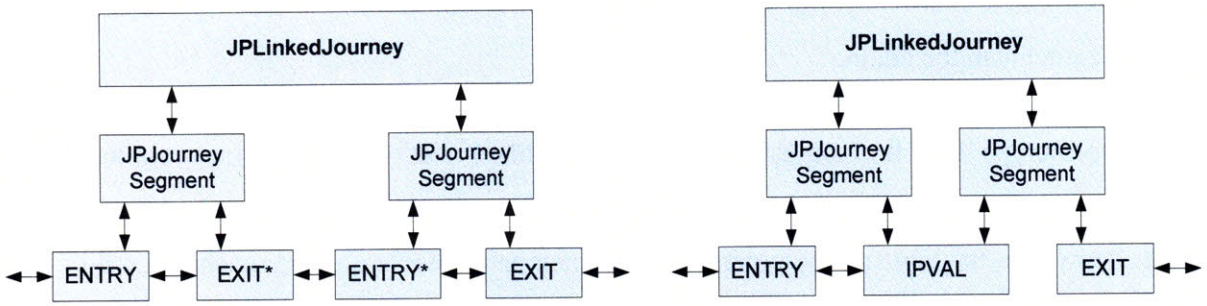


Figure 7.3 (left) – Configuration representing an instance of out-of-station interchange. * = Tap at OSI station. (right) – Configuration representing standard gate entry and exit with an intermediate validation at an internal validator.

Figure 7.3 (left) shows an intermediate validation. In Figure 7.3 (right) observe that validator taps (IPVAL and EPVAL) may have both LeftSegment and RightSegment occupied.

Figure 7.4 illustrates a composite journey representing a DLR journey followed by validation at Bank (by means of an internal validator), and an OSI before a final station exit. In particular, note the uni-directional pointer from the center JPJourneySegment to the JPLinkedJourney. Although each

JPJourneySegment has a pointer to the JPLinkedJourney to which it is claimed by, JPLinkedJourneys reference only the initial and final JPJourneySegments in the chain.

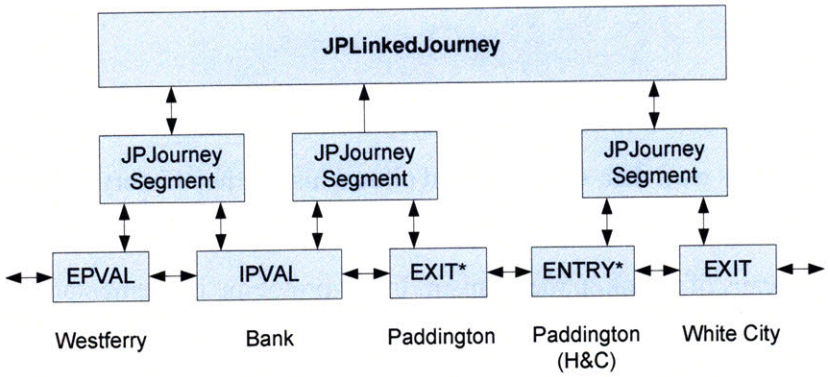


Figure 7.4 – A composite example. * = Tap at OSI station

The reason we only keep the first and last segment in each linked journey is to avoid the need for a dynamic list container, such an ArrayList in order to keep track of the journey segments. Remember that linked journeys may have any number of segments. There is no convincing case for such a list. The only time we would need to examine a linked journey systematically is to dispatch it for delivery (conversion into an external LinkedJourney object). When we do so we would simply iterate through the journey segments sequentially by means of an adjacent operator (see 7.5.1.1).

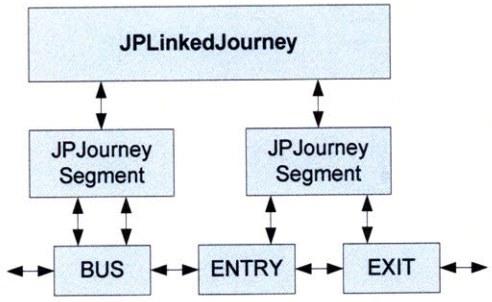


Figure 7.5 – A cross-mode (bus-rail) interchange.

Note that a bus JPJourneySegment consists of only one BUS JPTap and that both its StartTap and EndTap point to the same BUS JPTap. Conversely, both LeftSegment and RightSegment of a BUS JPTap point toward the JPJourneySegment that claims the Tap.

For sake of brevity, in subsequent discussion within this chapter we will assume the journey processor is our namespace. The JP prefix will be omitted and we will refer to JPTaps,

JPJourneySegments and JPLinkedJourneys as taps, journey segments and linked journeysrespectively.

7.2 *User Management*

An instance of the data structure described above in section 7.1 is created for each unique user. A JPUser object is the gateway to each user's own data structure, and allows his journey history to be manipulated independently of anyone else's. The JPUserManager object acts as a switchboard to direct Taps to the correct JPUser by means of a lookup mechanism. In the prototype implementation, a hashtable can be used to perform this lookup.

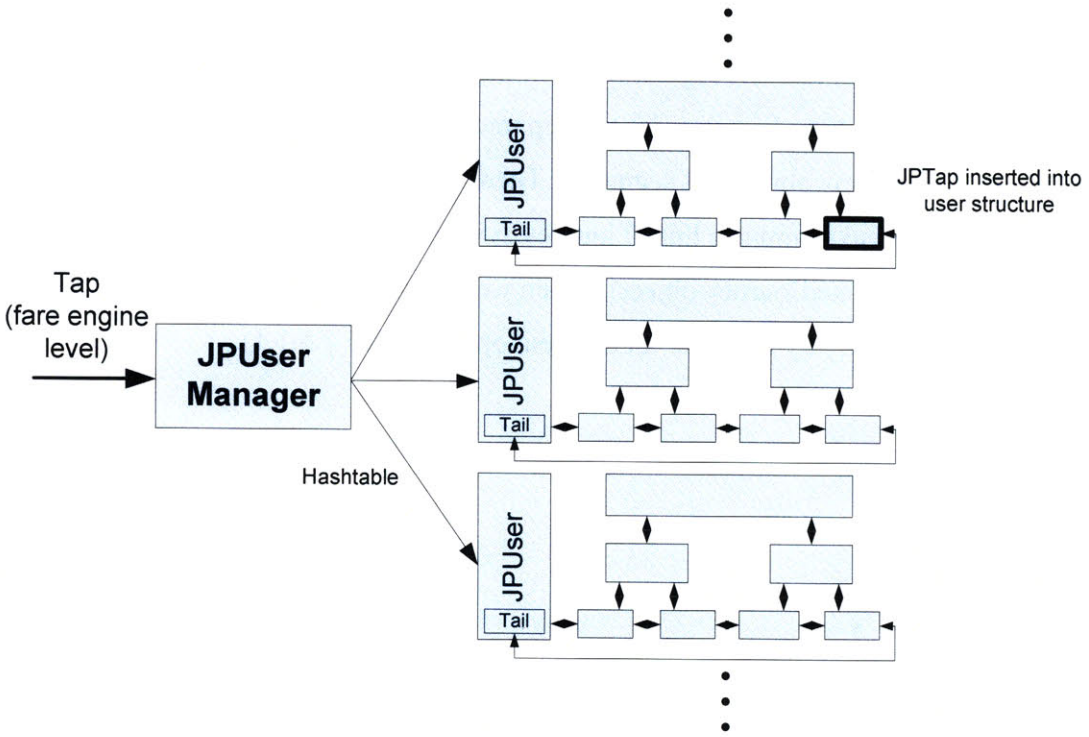


Figure 7.6 – User management in the journey processor.

7.2.1 Implications for Load Balancing

We noted previously that each user's data structure can be manipulated independently. This property provides a natural opening for load balancing. In load balancing we seek to equitably distribute the burden of fare processing (in terms of both memory requirements and CPU time) over two or more servers.

Within our architecture, this can be accomplished by maintaining one instance of the fare engine with its own journey processor on each processing server. A load balancing server assigns taps to the correct processing server based on the User ID of the tap and some arbitrary load balancing heuristic. The only requirement for this heuristic is that taps belonging to the same user must be load balanced to the same server consistently. Such a heuristic should account for differences in usage level between users. For example, a frequently used account with many trips in one day will be associated with a larger data structure, and therefore consume more memory than a seldom used account. Treating these two accounts equivalently would create an imbalanced load.

7.3 *Journey Processor Workflow*

The diagram below illustrates the workflow of the journey processor; this is the sequence of events that happen starting from the time a tap is inserted into the data structure. It shows the propagation of method calls up through the three layers. This process allows the insertion of a simple tap to cascade into a complete reconfiguration of linked journeys. Arrows represent method calls. Dashed arrows leading from object methods to static methods denote sequence of execution. For example, segmentCreated() is called after a new journey segment object is created, when the JourneySegment() constructor returns. Note that no destroyTap() methods exists as Taps cannot be destroyed in the context of journey linking. Taps are removed by the cleanup process, which is not shown here.

A notable feature is the use of static functions[18] to execute logic leading to the creation and destruction of objects *(linking logic)* at both the journey segment and linked journey levels. We clearly cannot express such logic inside methods of the objects affected, as those objects need to themselves be created first. Linking logic must be implemented in a context external to the objects being affected.

One possibility is to define linking logic within methods of the triggering objects. For example, the logic for creating linked journeys could be defined within JourneySegment() and destroySegment(), rather than in segmentCreated() and segmentDestroyed() as shown. However this approach puts

[18] A static function, or static method is a function which is not tied to any specific instance of the class it belongs to and can be executed without an object instance first being created.

business logic pertaining to the control of linked journeys within a journey segment. This breaks the principle of encapsulation and results in fragmented code which would be difficult to maintain.

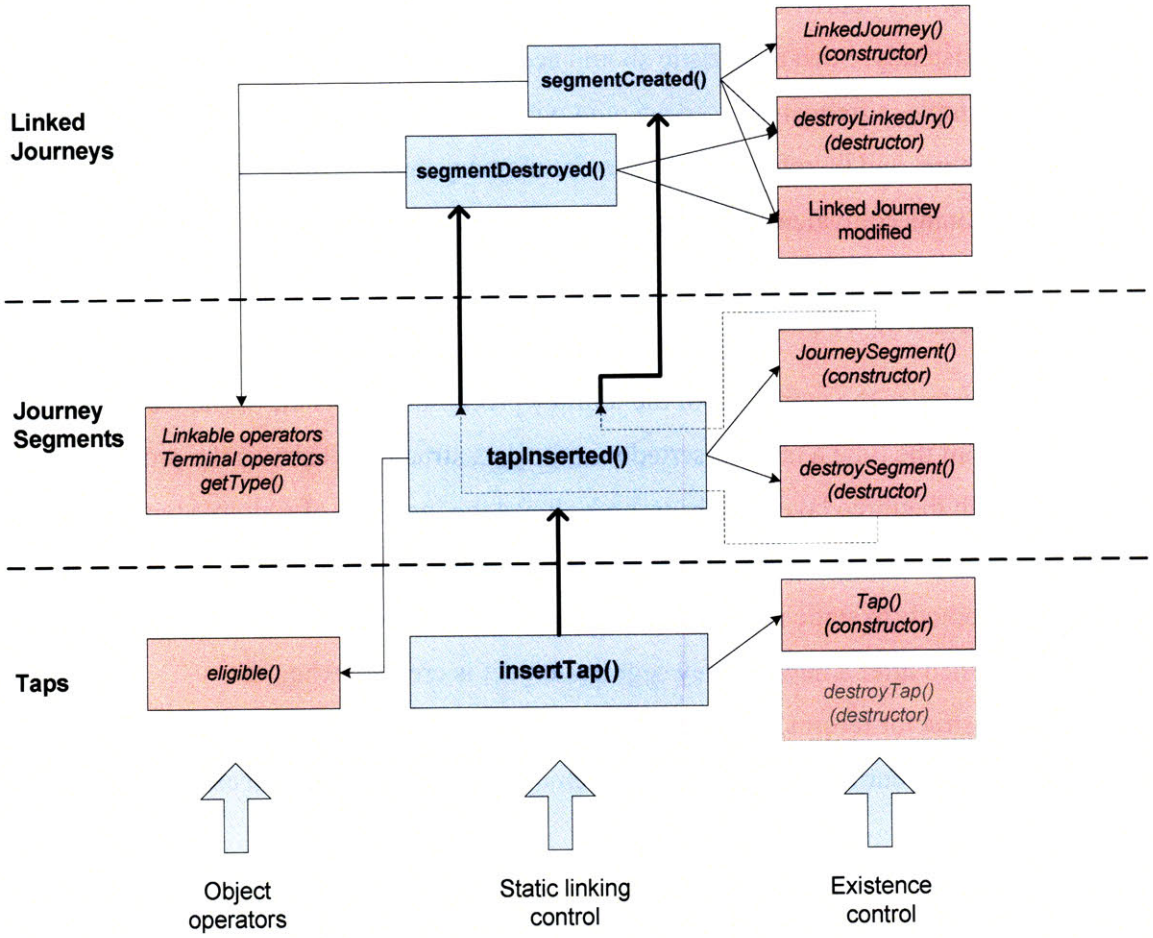


Figure 7.7 – Journey processor workflow.

Static functions provide the solution to this problem. Static functions are well suited for linking logic because they execute independently of data objects which they manipulate. In our particular implementation, these static functions are implemented as methods of singleton classes. The singleton pattern is an object creation design pattern described by Gamma et al. [18]. Use of singleton classes eases the plug-and-play replacement of critical linking logic and reserves flexibility for access control in future multi-threaded design compared to the use of static class methods.

7.4 *Tap Operations*

In Java and other common object oriented languages, objects cannot be intentionally destroyed on demand. We can only remove all handles to an object so that the garbage collector[19] will detect it as a redundant object and remove it. In light of this, taps, journey segments and linked journeys have destructor methods which perform the manipulations necessary to correctly detach the concerned object.

Below is a schematic view of the sequence of events that follows the insertion of a tap. This diagram is consistent with the lower part of Figure 7.7 and shows the internal operations of the *tapInserted()* method.

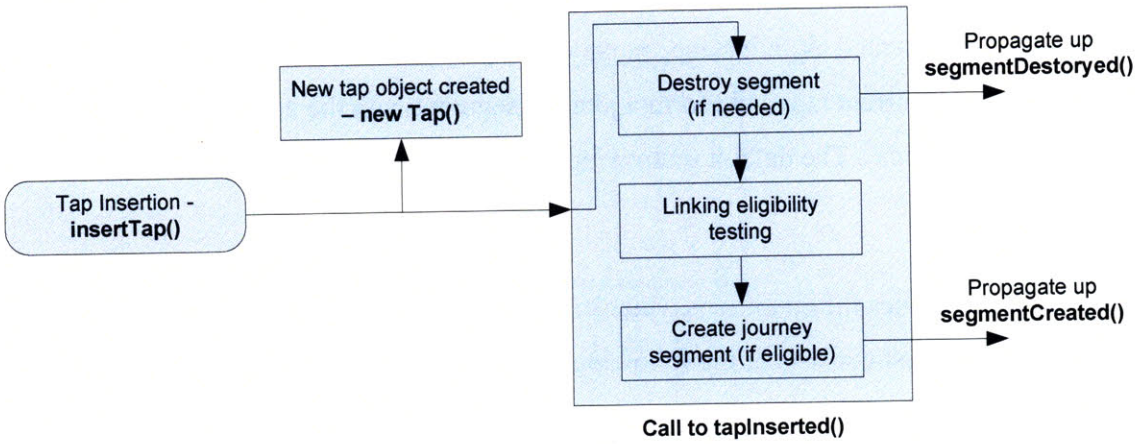


Figure 7.8 – Sequence of tap operations.

Tap processing begins with the calling of insertTap() by the user manager. An external tap is translated into the equivalent internal tap (JPTap in 7.1.1) and inserted into the correct position within the linked list. The newly created tap object is then passed to the tapInserted() method, which destroys and forms journey segments in response to the new tap.

7.4.1 Tap Insertion

Taps maintain constant references to their neighbors as a mechanism for enforcing sequential integrity and quickly determining adjacency. A doubly-linked list is used for this purpose. Such a

[19] In computer science, garbage collection refers to the background process which recycles the memory space occupied by abandoned data objects.

data structure permits the efficient insertion of out-of-sequence taps.

As shown in Figure 7.6, JPUser maintains a tail pointer (as well as the head pointer) to the doubly linked list of JPTap objects. Taps are inserted into linked list at the appropriate position (as determined by its timestamp) by walking backward from the rear of the linked list. The amortized time complexity of this operation in practice approximates to constant time O(1), as one would expect the vast majority of real-world taps to arrive in order or only out of order by a few positions. The insertion of a randomly timed out-of-order tap is of complexity O(n), where n is the number of taps in memory for the given user.

7.4.2 Eligibility Test

The eligibility test is implemented as an instance method of tap objects – eligible(). When invoked, eligible() tests whether the current tap could form a journey segment with the next adjacent tap in time (tap on the right hand side). The default return value where an adjacent tap does not exist to the right is false.

Tap linking eligibility describes the conditions which enable two station based taps to be connected by a journey segment. The test consists of the following components, all of which must be satisfied for the test to pass.

- **Maximum Journey Segment time** – The two journey segments must be separated by less than a set period. In the prototype this period may be set at 2 hours universally.

  Optional: An OD->Journey Time mapping table of journey segments customizing the maximum journey segment time may be employed. The table can be populated arbitrarily or using a schedule or performance based metric.

- **Location** – Prototype implementation has no specific limits on tap location. Any two station based taps anywhere in the system (whether gate or validator based) can qualify for linking given other necessary conditions are satisfied. Note that same-station entry-exit is considered legitimate and will result in a dummy journey segment.

  Optional: An OD->Eligibility mapping table of journey segment eligibility may be employed

to determine whether a given pair of taps can be linked at all (notwithstanding other constraints). This table may be merged with the OD->Journey Time mapping table described above.

- **Sequential Adjacency** – Taps must be immediately adjacent to each other in order to be considered for linking. The doubly-linked list structure allows adjacency to be detected rapidly. When an out-of-sequence tap is inserted into its correct place between two already-linked taps, the adjacency relationship between those taps is broken and that journey segment must be deconstructed.

- **Syntactic Pattern** - The type value of the leading and trailing taps of a journey segment must conform to a set of predefined *syntactic patterns*. For example, a journey segment can be formed from an entry tap followed by an exit tap, but not vice versa. These patterns correspond to legitimate travel behavior through the system, and are described in greater detail in the sections below.

7.4.2.1 Syntactic and Semantic Patterns

Journey segments can be viewed from two perspectives. A syntactic pattern is the view of a journey segment from the perspective of the fare engine. In contrast, a semantic pattern is the view of a journey segments from the user's perspective.

- **Syntactic Pattern** – A syntactic pattern is uniquely identifiable by a combination of entry and exit tap types. This is what the journey processor sees and has the ability to differentiate by inspecting taps.

- **Semantic Pattern** - The semantic pattern describes the physical travel behavior which would result in a certain syntactic pattern being produced. Each syntactic pattern may correspond to one or more semantic patterns. The fare engine is incapable of differentiating between different semantic patterns which produce the same syntactic pattern.

7.4.2.2 Semantic Pattern Variability

If intermediate validation is made compulsory, in other words, any time a user passes a platform validator he is required to perform intermediate validation, a journey segment with an ENTRY-

>EXIT syntactic pattern can only be produced by the following two semantic patterns (travel behaviors):

- A single underground journey without interchanges
- Two or more underground journeys with interchanges within the secured area which cannot be recorded.

However, in our contactless bankcard fare engine we have adopted an open policy with regard to intermediate validations. Every intermediate validation is considered optional. Under this relaxed condition, the same ENTRY->EXIT syntactic pattern above can be produced by numerous semantic pattern variations. One such variation is given in the below example:

1. User enters secured area at LUL station A through fare gate (ENTRY tap).
2. User travels via LUL services 1 and 2 to intermediate station C changing underground lines en route within the secured area of station B.
3. User proceeds to NR/DLR platform at station C and boards an extracting service 3 without validating on the platform.
4. User gets off service 3 at ungated station D and boards another NR/DLR service 4, without validating.
5. User injected into the secured area of interchange station E via service 4 and boards extracting NR/DLR service 5. He does not validate on platform.
6. User arrives via service 5 into the secured area of interchange station F. Although service 5 is an extracting service, because the user began in the secured area of station E and alighted in the secured area of station F, he has not left the secured area at all. This is not considered an extraction and reinsertion.
7. User leaves the NR/DLR platform without validating and proceeds directly the LUL platforms of station F (all within the secured area). He boards LUL service 6.

8. User arrives at destination station H and exits via fare gate (EXIT tap).

Despite its complexity, this journey is visible to the fare engine as a single journey segment. Without compulsory intermediate validation it is evidently impossible to enumerate every semantic pattern associated with a given syntactic pattern.

7.4.2.3 Typical Semantic Patterns

In the notation used in the table below, syntactic patterns are assigned numeric identifiers while semantic variations for each syntactic pattern are assigned an alphabetical postfix. As discussed previously it is impractical to enumerate every possible semantic pattern. Some typical patterns are shown.

Pattern	Syntactic Pattern	Semantic Pattern and Remarks	TfL Example
1	ENTRY → EXIT	Standard Underground entry-exit scenario. Travel fully within secured area.	Putney Bridge → St James's Park
2a	ENTRY → IPVAL (diff. stations)	User enters through a fare gate and validates at a validator within the paid area at a different station. This occurs when the user makes an interchange to an ungated mode (NR or LUL) at a station where the ungated service is brought directly inside the secured area.	Putney Bridge → Stratford (where user proceeds to NR platform behind fare gates) Putney Bridge → Bank (platform validator on DLR platform)
2b	ENTRY → IPVAL (same station)	User enters the secured area through a gate line at interchange station A. He proceeds to the NR/LUL platforms and taps on the platform internal validator before boarding an extracting NR/LUL service. This creates a dummy journey within	Bank (gate entry) → Bank (platform validator)

		interchange station A.	
3a	IPVAL → EXIT (diff. stations)	This is the converse case to 2a above. In this scenario the user is injected into a gated area aboard an ungated service which penetrates the security boundary by stopping behind fare gates.	West Brompton (Mag stripe ticket holder arriving on Overground platform) → Putney Bridge
3b	IPVAL → EXIT (same station)	User is injected into the secured area of an interchange station and taps on validator at the platform before proceeding to the gate exit at the same station. This effectively creates a dummy journey segment travelling from and to the same station.	Bank (platform validator) → Bank (gate exit)
4a	IPVAL → IPVAL	User injected into paid area by ungated service A, travels to a different NR/DLR interchange station within the secured area, and extracted through security boundary by ungated service B without passing through either entry or exit gates.	West Brompton (arrive by Overground) → (LUL) → Stratford (depart by NR) West Brompton (arrive by Overground) → (LUL) → Bank (depart by DLR)
4b	IPVAL → IPVAL	User is extracted from the secured area by an ungated service, and injected into the secured area at a second station. The bulk of this journey segment is completed on an ungated service.	West Brompton (arrive by LUL) → (Overground or NR) → Stratford (NR platforms)
5	EPVAL → EPVAL	User taps in at an external validator at an ungated station, travels via	Heron Quays → Westferry

		NR/DLR to a second ungated station, and taps out.	
6	EPVAL → IPVAL	User taps at an external validator at an ungated station, and injected inside secured area aboard a NR/DLR service. Taps on internal validator on platform upon arrival.	Westferry → Bank (platform validator)
7	IPVAL → EPVAL	User extracted from secured area aboard a NR/DLR service, in that process tapping at internal validator on the platform from within the secured area. User taps out at an external validator at the destination.	Bank (platform validator) → Westferry
8a	EPVAL → EXIT (exit at interchange station)	User taps at external validator at an ungated station and is injected inside paid area aboard a NR/DLR service. User bypasses the internal validator on platform and proceeds directly to the gate line through which he exits the gated area.	Westferry → Bank (gate exit)
8b	EPVAL → EXIT (exit not at interchange station)	User taps at an external validator at ungated station A and is injected inside the paid area of station B aboard a NR/DLR service. User then bypasses the internal validator on platform and proceeds to take an LUL service to LUL station C where he exits via the gate line.	Westferry → Bank (no interaction) → Putney Bridge (gate exit)
9a	ENTRY → EPVAL (entry at interchange	User enters the secured area through a gate line and, without tapping on a	Bank (gate entry) → Westferry

	station)	platform internal validator, proceeds directly onto a NR/DLR service which extracts him from the secured area. He taps out at an external validator at the destination station.	
9b	ENTRY → EPVAL (entry not at interchange station)	User enters the secured area through a gate line at station A, where he takes an LUL service to interchange station B. There he proceeds to a NR/DLR service located within the secured area without any interaction with the platform internal validator, and is extracted by this service to ungated destination station C, where he taps out at a platform validator.	Putney Bridge (gate entry) → Bank (no interaction) → Westferry

Table 7.4 – Common semantic patterns for acceptable journey segments.

The accepted syntactic patterns can be described in matrix form as shown in the table below. Cells marked with a dash indicate Tap permutations which cannot form a journey segment.

Leading\Trailing	ENTRY	EXIT	IPVAL	EPVAL
ENTRY	-	1	2	8
EXIT	-	-	-	-
IPVAL	-	3	4	7
EPVAL	-	9	6	5

Table 7.5 – Matrix form of acceptable syntactic patterns. Note that an EXIT tap could never be a leading tap and an ENTRY tap could never be a trailing tap.

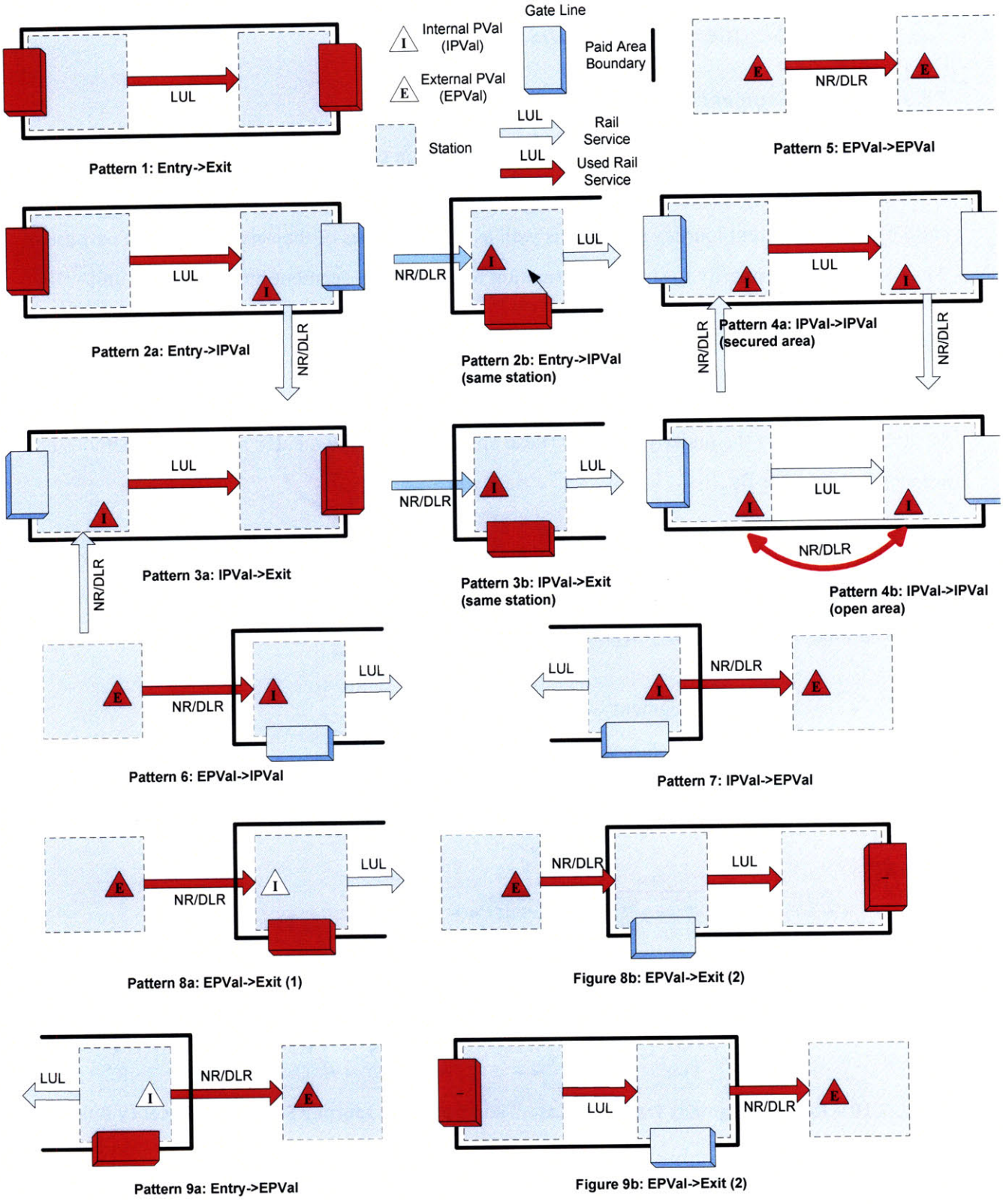


Figure 7.9 – Common semantic patterns. This diagram accompanies Table 7.4.

7.5 *Journey Segment Operations*

7.5.1 Journey Segment Operators

Journey segment *operators* are methods of journey segment objects which return a piece of information (e.g. a true/false value or another journey segment object) based on the properties of the child taps of the current journey segment as well as the properties of their neighbors. The purpose of these operators is primarily to supply information to the journey segment linking control unit[20].

7.5.1.1 Adjacency

Recall that journey segments are not linked and carry no data. Operators on journey segment objects function by querying the underlying taps. These operators are implemented as stateless methods and are evaluated on-the-fly.

The following operators allow journey segments to detect their own neighbors and enable traversal from one journey segment to the next. These operators utilize the linkage between the underlying taps to find the neighbor journey segment.

- Next Journey Segment - next()
- Previous Journey Segment – prev()

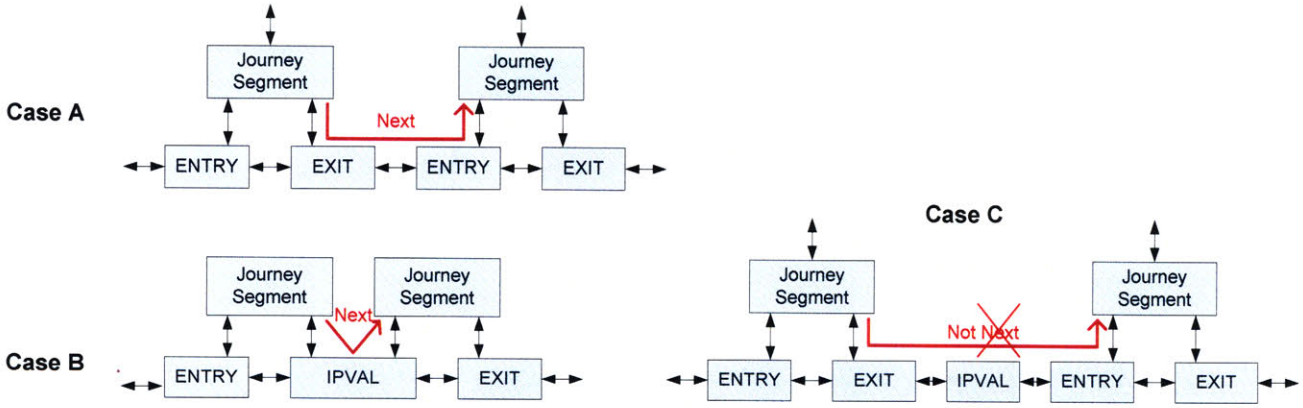


Figure 7.10 – Journey Segment Traversal. Case C illustrates the Journey Segment Adjacency Rule.

Figure 7.10 illustrates the operation of the next() operator. Two journey segments are considered adjacent if they a) end and start at taps that are adjacent, or b) a) share a tap. The next() operator will

[20]The Journey Segment Linking Control is described in section 7.5.2.

not return a subsequent journey segment if c) the current segment is separated from the subsequent journey segment by one or more taps. We will call this the *Journey Segment Adjacency Rule*. The journey segment adjacency rule is important as it allows orphaned taps to break apart an existing linked journey. This rule will be invoked again subsequent discussion.

7.5.1.2 Linking

There are three linking operators, representing the three ways that journeys linking can occur at TfL.

1. Intermediate validation at a passenger validator
2. Out-of-station interchange
3. Bus-Rail/Bus-Rail interchange

A linking operator detects whether a journey segment can be linked with its immediate subsequent neighbor, as determined by the next() adjacency operator described in 7.5.2. Each linking operator tests one linking condition and returns true if the journey is linkable by that reason.

If two journey segments are not adjacent (per the Journey Segment Adjacency Rule), the linking operator returns a default value of false. Journey segment linking operators can be implemented in the following equivalent contexts

- Parameterless object method – linkable():The operator is implemented as a method of a journey segment object and implicitly tests for linkability against its next segment, if found. If none is found then the default false value is returned.

- Single parameter object method – linkable(otherSegment): The operator is implemented as a method of a journey segment object and accepts a parameter which is the adjacent segment to test against. This operator is bidirectional will function correctly whether otherSegment precedes or comes subsequent to the current segment. If otherSegment is not adjacent to the current segment false will be returned.

- Two parameter static method – linkable(segmentA, segmentB): This operator is implemented as a static method taking two parameters, representing the two segments to be tested for linkability. This operator is bidirectional and will function regardless of

which segment precedes the other, assuming they are adjacent. If segmentA and segmentB are not adjacent to each other than false will be returned.

We will describe our three linking operators in the simplest context, which is the parameterless object method context.

Intermediate Validation – IVLinkable()

Return true if (EndTap.Type == IPVAL OR EPVAL) AND EndTap==next().StartTap

Intermediate validation is indicated if and only if the two journey segments share the same IPVAL or EPVAL. There is no test for interchange time. Intermediate validation interchanges are instantaneous by definition.

IV linking always returns false if one of the taps is a BUS tap.

Out of station interchange – OSILinkable()

Return true if

(EndTap.Type == EXIT) AND
(next().StartTap.Type == ENTRY) AND
(OSIInterval(EndTap.locationID, next().StartTap.locationID != null) AND
(next().StartTap.timestamp - EndTap.timestamp < OSIInterval(EndTap.locationID, next().StartTap.locationID))

In other words, the EndTap of the current journey segment must be of type EXIT and the StartTap of the next journey segment must be of the type ENTRY. Furthermore, we require that this EXIT-ENTRY pair be defined in the OSI table and that the time separation between the two is less than the required time interval.

OSI linking always returns false if one of the taps is a BUS tap.

Cross-mode interchange

BusRailLinkable() - Return true if

((EndTap.Type == BUS) AND
next().StartTap.Type is one of {ENTRY, EPVAL} AND
next.StartTap.timestamp – EndTap.timestamp < BusRailInterchangeInterval

RailBusLinkable() – Return true if

(EndTap.Type is one of {EXIT, EPVAL}) AND
(next().StartTap.Type == BUS) AND
next.StartTap.timestamp – EndTap.timestamp < RailBusInterchangeInterval)

Here we require an interface of a BUS tap with either an ENTRY or EPVAL tap. An operator is provided for each direction of interchange. These operators are always applied from the context of the current journey segment being the one on the left which is being tested for linkability against its immediately subsequent neighbor on the right.

The cross-mode interchange interval must be observed. This is the interval of time which the user has to perform the interchange if he wants to take advantage of the interchange credit. This value is asymmetrical because we can determine when a user has left the rail system based on his interaction with station-based gates and validators, but we do not know when a user has off-boarded a bus. The reference timepoint in this case is the boarding of the bus. To compensate we generally allow a longer period for bus-rail interchange, adding a maximum allowable travel time component to the raw interchange time.

The test will always return false if the interface of the two segments being tested consists of two BUS taps or two station-based taps.

7.5.1.3 Terminator

The end terminator operators exist to support the Billable operator on a linked journey (see 7.6.1.2). These operators are necessary as we do not allow taps to be directly visible to linked journeys. There are two end terminator operators:

StartTerminator()

Return false if *StartTap.Type==IPVAL*
Return false if *StartTap.Type==EPVAL AND StartTap.LeftSegment!=null*

else return true

StartTap being an IPVAL always disqualifies a journey segment from being a startTerminator. If StartTap is an EPVAL, then it must not be shared with another journey segment. (i.e. not intermediate validation).

EndTerminator()

Return false if End*Tap.Type==IPVAL*
Return false if *EndTap.Type==EPVAL AND StartTap.RightSegment!=null*
else return true

EndTap being an IPVAL always disqualifies a journey segment from being a endTerminator. If EndTap is an EPVAL, then it must not be shared with another journey segment. (i.e. not intermediate validation).

7.5.1.4 Type

Journey segments have a getType() operator, again motivated by the desire to isolate taps from linked journeys. Journey segments can be classified into two types.

- ONBOARD – detected by StartTap.Type=BUS. Note that BUS taps are always attached to both StartTap and EndTap or neither. We only need to test one.
- STATIONBASED – all other case.

7.5.2 Journey Segment Linking Control

The journey segment linking control unit effects the creation and destruction of journey segments in response to the creation of a new tap. This unit is embodied in the tapInserted() static function.

This process of linking adjacent taps to form journey segments should not be confused with the linking of taps to form a time-sequence using a doubly linked list as described in section 7.4.1. A sequential linkage exists between all adjacent taps regardless of the relationship between them. The formation of journey segments, however, is predicated on the properties of the tap(s) concerned.

One of the following scenarios will apply to a tap that has just been inserted into the sequence.

- The inserted tap is an on-board validation tap (BUS tap).

- Tap is inserted at the start or end of the sequence. *Inserted tap does not split existing journey segment.*
- Tap B is inserted into the middle of the sequence, between taps A and C. Taps A and C are not claimed by the same journey segment. *Inserted tap does not split existing journey segment.*
- Tap B is inserted into the middle of the sequence, between taps A and C. Taps A and C are claimed by the same journey segment. *Inserted tap splits existing journey segment.*

Inserted tap is an on-board validation (BUS) tap

An encapsulating journey segment linking claiming the BUS tap is automatically created (see example in Figure 7.5). StartTap and EndTap of the new journey segment point to the BUS tap. LeftSegment and RightSegment of the BUS tap reciprocate.

Inserted tap does not split existing journey segment

The insertion of a station-based tap which does not split an existing journey segment triggers an *eligibility test* between the inserted tap and its two adjacent taps (where available). The eligibility test is defined in section 7.4.2.

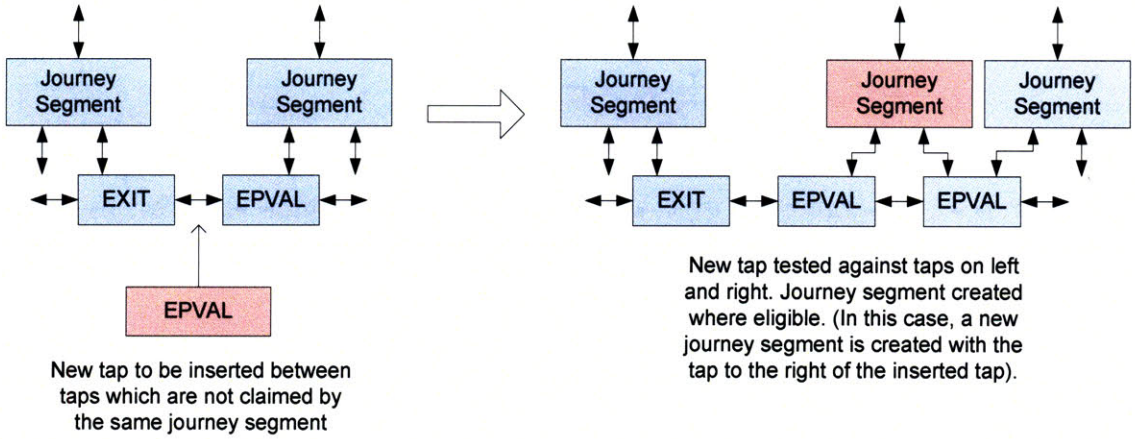


Figure 7.11 – Insertion of a tap which does not split an existing journey segment.

For each test passed, a new journey segment is created. The created journey segment claims the newly added tap and the tap against which the test has passed (by setting StartTap and EndTap). References are reciprocated by the claimed tap (via LeftSegment or RightSegment). If both tests

fail, no journey segment object is created. The inserted tap remains untouched and is not removed.

In summary, if an inserted tap does not split an existing journey segment, zero, one or two new journey segments will be created as a result. No journey segments will be destroyed. Note that the creation of each new journey segment in turn leads to changes at the linked journey level (propagating upward). This response is discussed in section 7.6.2 below.

Inserted tap splits existing journey segment

When a new tap is inserted between two taps claimed by same existing journey segment, the following sequence of events takes place:

1. Existing journey segment being split by incoming tap is destroyed.

2. The inserted tap is tested against adjacent taps to the left and right. (See section 7.4.2 for description of eligibility test)

3. Zero, one or two new journey segments may be created depending on which tests pass.

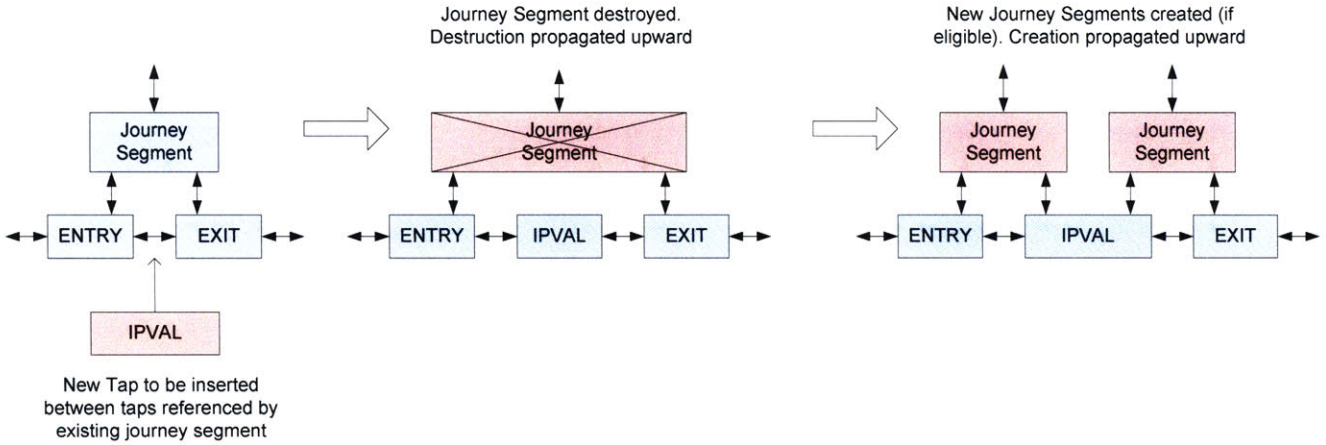


Figure 7.12 – Insertion of a tap that splits an existing journey segment.

It may seem paradoxical but additional information could disrupt and remove a previously legitimate journey segment. This occurs when an existing journey segment is removed but not replaced by anything. For example, if an EPVAL tap was inserted between previously claimed ENTRY and EXIT taps, the existing journey segment between the ENTRY and EXIT would be destroyed. None of the tests would pass in an ENTRY->EPVAL->EXIT sequence, so no new

journey segments would be created. This can occur if there were network outages; once all taps are received, valid segments are constructed.

7.6 *Linked Journey Operations*

7.6.1 Linked Journey Operators

First we will introduce some notation for describing linked journeys. A linked journey containing only one journey segment is called a *unitary linked journey.* A linked journey containing more than one journey segment is called a *chained linked journey.*

7.6.1.1 Adjacency

Linked journeys have the following traversal operators:

- Next Linked Journey: next() – Implemented as EndSegment.next().ParentLinkedJourney

- Previous Linked Journey: prev() – Implemented as StartSegment.prev().ParentLinkedJourney

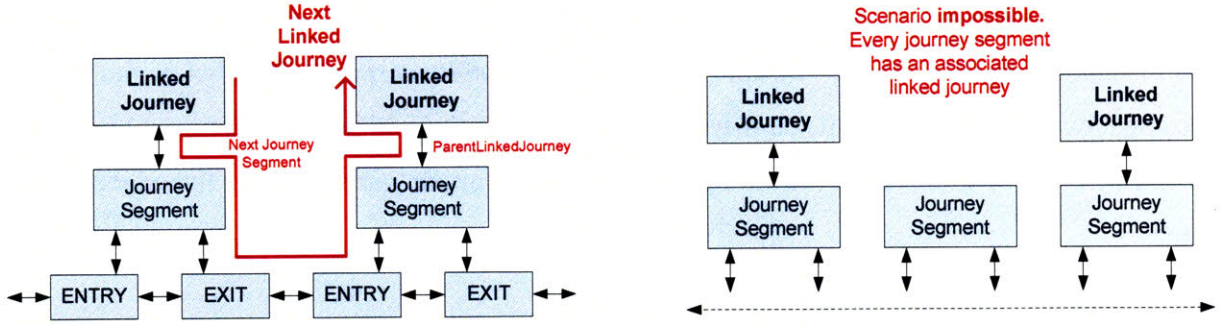


Figure 7.13 – Linked Journey Traversal.

Linked journey traversal operators are implemented using the next() and prev() operators for journey segments defined in section 7.5.1.1. The *Journey Segment Adjacency Rule* described above is observed. Note that journey segments can never be unclaimed. ParentLinkedJourney is always a non-null value.

7.6.1.2 Billability

A linked journey is only billable if it is properly terminated. A linked journey is properly terminated

if it does not begin or end in an IPVAL. The billability operator – Billable() is defined in terms of the journey segment terminator operators.

Billable() = StartSegment.StartTerminator() AND EndSegment.EndTerminator()

A linked journey is considered billable only if the first segment has a terminated front-end and the last segment has a terminated rear-end.

7.6.2 Linked Journey Linking Control

The linked journey linking control unit is the piece of logic responsible for the creation, destruction, reconfiguration or amalgamation of linked journey objects. For the same reasons discussed above in section 7.3, this unit is implemented as a static function (more accurately, a singleton method). This unit is embodied in the segmentCreated() and segmentDestroyed() functions.

When a journey segment is created or destroyed, one of the above functions is called. In either case, we first evaluate the *environment* of the concerned journey segment. The term *environment* is defined very specifically here to mean *whether the journey segment appears in a position where it 'splits' an existing linked journey or not.*

After evaluating the environment, we apply the *general linkability test* (described below) on the concerned journey segment and its partners. Finally, based on the evaluated environment and the outcome of the general linkability test we perform the necessary manipulations on linked journey objects.

The resulting manipulations vary. A new journey segment could either be wrapped in its own unitary linked journey, or it could be incorporated into an existing linked journey. In some cases, a new journey segment could even form the keystone that completes two previously separate linked journeys, causing them to be merged into one, bridged by the new journey segment. Similarly, a destroyed journey segment could either result in no action, or the splitting of an existing linked journey.

7.6.2.1 General Linkability Test

Figure 7.14 describes the *general linkability test*. In the discussion below a 'test for linkability' refers to the application of this general linkability test. The test is composed of the three decision

boxes in the flow chart below. We will elaborate on each of these boxes.

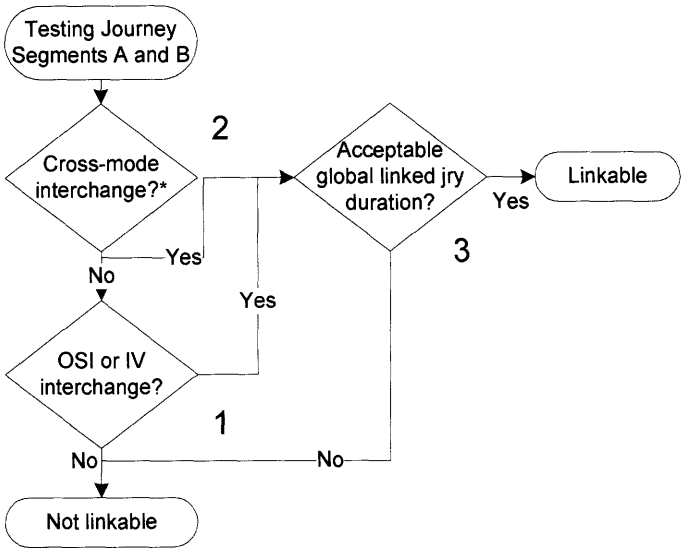


Figure 7.14 – The general linkability test. *The 'OSI or IV interchange' box is a direct application of those journey segment operators. However, cross mode interchange must also take not account the current constitution of the linked journey with relation to ONBOARD journeys.

1. Out-of-station interchange and intermediate validation

Station-based journey segments can be linked together either by out-of-station interchange or by intermediate validation. This test is a direct application of the OSI and IV linking operators as described in section 7.5.1.2. If we imagine that journey segments are linear jigsaw pieces, the linking operators describe whether their edges mesh together or not.

In pseudo code, the decision is given as.

IVLinkable() OR OSILinkable()

Remember that parameterless operators test for linkability against the next adjacent segment to the *right*. Thus to apply this test between adjacent journey segments A and B, we could simply call the linkable operators on object A.

2. Cross-mode interchange

Cross mode interchanges are governed by a condition that we must pay special attention to. In addition to a positive result in the cross-mode linkability test, we also require that a linked

journey contain only one bus segment (ONBOARD), and that this bus segment be positioned either at the front or the rear of the linked journey. The following logic enforces this condition. Again recall that linking operators are described in terms of the current segment being located on the left and a test is automatically implied against its next subsequent neighbor on the right. This is the parameterless method context. If the operator was implemented in a different context, the pseudo code below must be translated correspondingly.

segment.BusRailLinkable() AND segment.next().parentLinkedJourney.startSegment ==segment.next() AND segment.next().parentLinkedJourney.endSegment.getType() != ONBOARD

OR

segment.RailBusLinkable () AND segment.parentLinkedJourney.endSegment == segment AND segment.parentLinkedJourney.startSegment.getType() != ONBOARD

There is no need to scan a journey segment to determine whether it already contains a bus segment. The inclusion of a first bus segment into a linked journey will automatically 'cap' that linked journey such that future tests for cross-mode interchange will automatically fail.

3. Global linked journey duration limit

We have discussed various time intervals. The creation of journey segments is governed by the maximum journey segment duration. OSI and cross-mode interchanges have their respective interchange intervals. Neither of these is to be confused with the global linked journey duration.

The global linked journey duration describes the total length of a linked journey. This is the sum of the durations of all constituent journey segments and any interchange intervals separating them. The global linked journey duration sets the ultimate upper bound on the length of a journey. In effect, many short journey segments may be linked together, but the presence of one or few lengthy segments would use up the global quota and restrict further chaining.

The global linked journey duration limit is one of the stated requirements of the fare engine. It is motivated by variations in semantic patterns. Because intermediate validation is optional under our regime, a single journey segment could, in one case, represent small units of travel in a

journey made by a user who intermediate validates duly and frequently. However, when a user shuns intermediate validation, a single journey segment could end up representing a lengthy and circuitous journey with many unrecorded interchanges.

The first case above would be represented by one linked journey spanning the multiple intermediate segments. The second case above would be represented by one linked journey, encapsulating the lone journey segment. The enforcement of global linked journey duration levels the playing field between these two use cases. The two cases above would use up the exact same amount of the global linked journey duration cap.

7.6.2.2 Environment Scenarios

Having described the general linkability test, we can now describe the *environment scenarios* that make use of this test, and the actions taken upon each test outcome in these scenarios. A journey segment having just been created or being deleted is referred to as the *triggering journey segment.* Note that any time we manipulate linked journeys, reverse references on the affected journey segments must be adjusted to remain consistent with the new linked journey configuration. For brevity this step may not be explicitly mentioned.

Scenario A: A journey segment is created and its neighboring journey segments are claimed by the same linked journey.

This scenario results in the 'splitting' of an existing linked journey. The linked journey being split will be called '*the original linked journey*'.

First, we de-posted the original linked journey, if its posted flag is set. Then the triggering journey segment (B) is tested for linkability against its two neighbors (A & C), if A & B exist. Linkability test outcomes are as follows:

Neither AB nor BC linkable – The original linked journey is shrunk so that it claims only the journey segments lying to the left of the triggering journey segment. A new unitary linked journey is created for the triggering journey segment, and a second new linked journey is created for all the journey segments claimed by the original linked journey that lie to the right of the triggering journey segment.

AB linkable, BC not linkable – The original linked journey is shrunk so that it claims only the journey segments lying to the left of, and including the triggering journey segment. A new linked journey is created for all of the journey segments claimed by the original linked journey that lie to the right of the triggering journey segment.

AB not linkable, BC linkable – The original linked journey is shrunk so that it claims only the journey segments to the left of the triggering journey segment. A new linked journey is created for the triggering journey segment as well as any journey segment claimed by the original linked journey that lies to the right of the triggering journey segment.

AB and BC both linkable – The original linked journey is not modified. We only need to adjust the reverse reference on the triggering journey segment to point to the original linked journey.

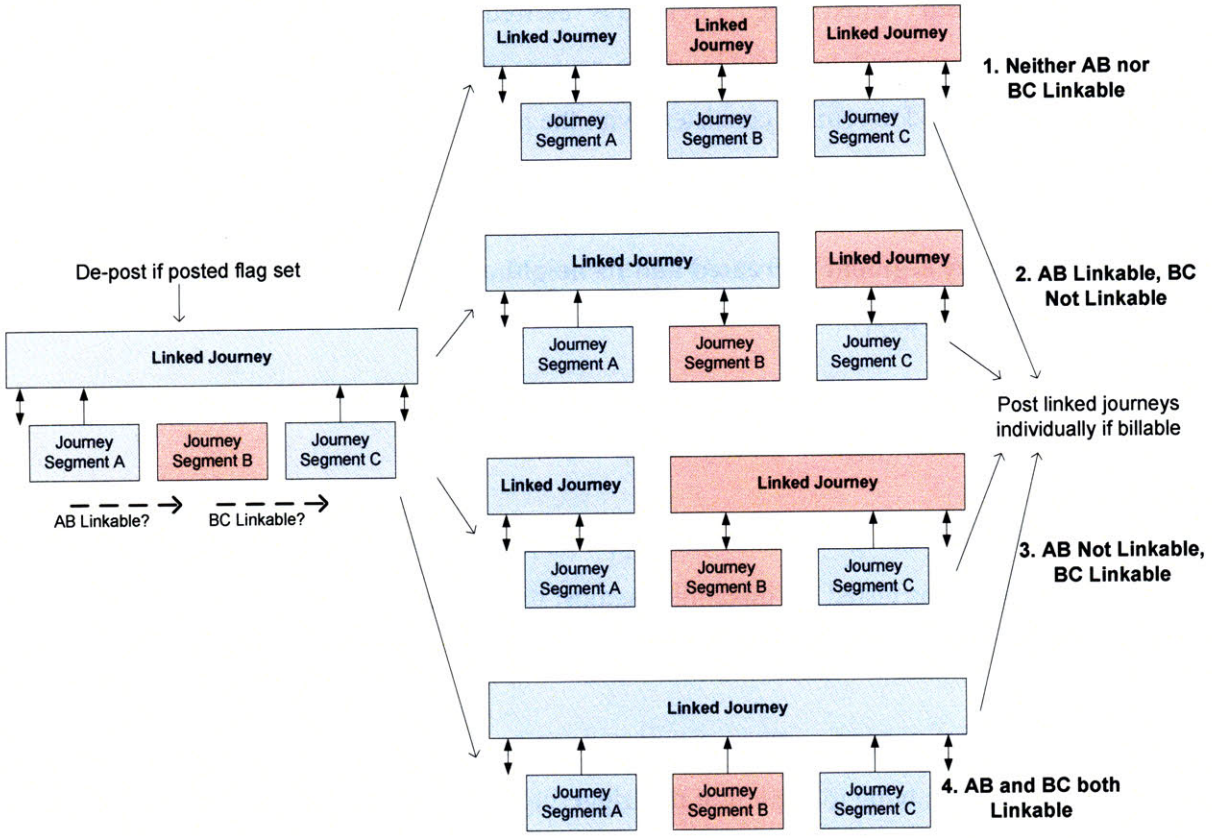


Figure 7.15 – Scenario A: New journey segment splitting existing linked journey. In this scenario, the triggering journey segment is a new journey segment and its neighboring journey segments are claimed by the same linked journey.

Whichever outcome is the end result, we test the resulting linked journeys for billability and post

them if billable, on an individual basis.

Scenario B: A journey segment is created whose neighboring journey segments are not claimed by the same linked journey

This scenario does not split an existing linked journey. As above, we test the triggering journey segment against its adjacent neighbors for linkability and four outcomes are possible. In this scenario, we do not de-post the original linked journeys by default, but only conditionally based on the outcome of the linkability test.

Cases where the triggering journey segment is missing one or both of its neighbors are a special subset of the general scenario described here. For example, the triggering journey segment could sit at either extreme of the data structure. Or the triggering journey segment could be insulated from successive or previous journey segments by an orphaned tap (the application of the Journey Segment Adjacency Rule ensures that segments are not considered adjacent where an intervening tap exists). In these cases, the linkability test would fail by default and the resulting outcome would still fall under either outcome 1, 2 or 3.

Neither AB nor BC linkable – A unitary linked journey is created, encapsulating the triggering journey segment. The new linked journey is then posted to the fare engine if billable. No changes are made to existing linked journeys.

AB linkable, BC not linkable – The existing linked journey claiming journey segments located to the left of the triggering journey segment is first de-posted if its posted flag is set. It is then extended to cover the newly created journey segment and re-posted, if billable.

AB not linkable, BC linkable – The existing linked journey claiming journey segments located to the right of the triggering journey segment is first de-posted if its posted flag is set. It is then extended to cover the newly created journey segment and re-posted, if billable.

AB and BC both linkable – The existing linked journeys claiming journey segments located on both sides of the triggering journey segment are first de-posted if their billable flags are set. The linked journey on the left (#1 above) is then modified to cover both the triggering journey segment and the journey segments located on its right, up to the rightmost extent of

the now obsolete linked journey 2. Linked journey 1 is re-posted if billable. Linked journey 2 is removed.

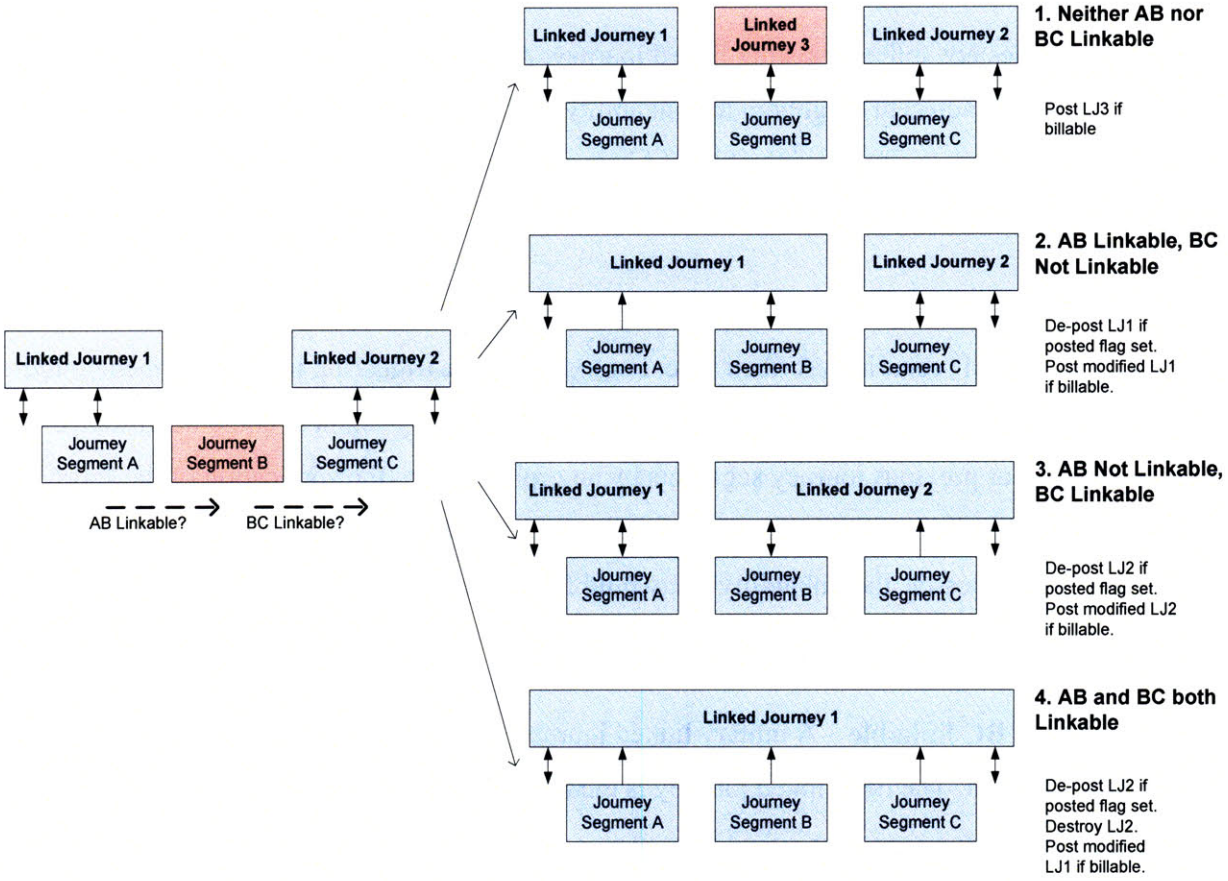


Figure 7.16 – Scenario B: Triggering journey segment is a new journey segment that does not split an existing linked journey.

Scenario C: A journey segment claimed by a unitary linked journey is destroyed

The deletion of an unencumbered journey segment is very straightforward scenario and leads to only one outcome. An unencumbered journey segment is one encapsulated by a unitary linked journey.

1. If unitary linked journey has the posted flag set, de-post with fare processor.

2. Destroy unitary linked journey.

One might wonder why we do not have to test the two remaining journey segments (A & C) for linkability, and merge their respective linked journeys if necessary. The answer is that this

operation, although logically necessary, is guaranteed to always return a negative result. We note that:

- Excluding cleanup, taps can only be added to the data structure, not deleted.
- The Journey Segment Adjacency Rule[21] states that journey segments can never be linked if separated by one or more taps.

The destruction of a journey segment must have been triggered by the *insertion* of a new tap, which would always prevent the remaining journey segments (i.e. A and C) from being conjoined.

Having said this, the insertion of a tap which leads to the destruction of a first journey segment and the creation of its replacement can still result in the union of existing linked journeys. A two-stage mechanism is in play here. The tap induces the deletion of the first journey segment, which is bubbled upward and triggers either scenario C or D. Then, the second journey segment is created. This in turn triggers either scenario A or B, creating a conjoined linked journey. At this point the structure finally stabilizes.

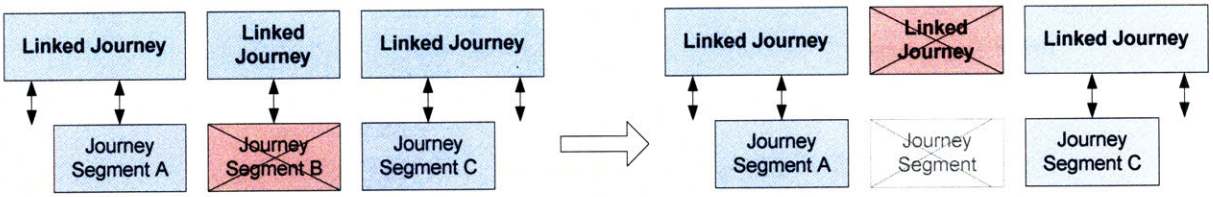


Figure 7.17 – Scenario C: Unencumbered journey segment deleted.

Scenario D - A journey segment claimed by a chained linked journey is destroyed.

This scenario describes the removal of a journey segment which 'splits' an existing linked journey. The triggering journey segment here refers to the journey segment being deleted.

1. If the linked journey being split has its posted flag set, de-post with fare processor.

[21] See section 7.5.1.1.

2. Shrink linked journey being split to cover only journey segments located to the left of the triggering journey segment, if there are any.

3. Create a new linked journey for remaining journey segments located to the right of the triggering journey segment. An exception to this rule occurs if the triggering journey segment is the leftmost element in a linked journey. In this case, the existing linked journey can be re-tasked for the journey segments on the right of the triggering segment.

4. Linked journeys are posted to the fare processor individually, if billable.

Again, the journey segments adjoining to the deleted segments do not have to be tested for linkability due to Journey Segment Adjacency Rule.

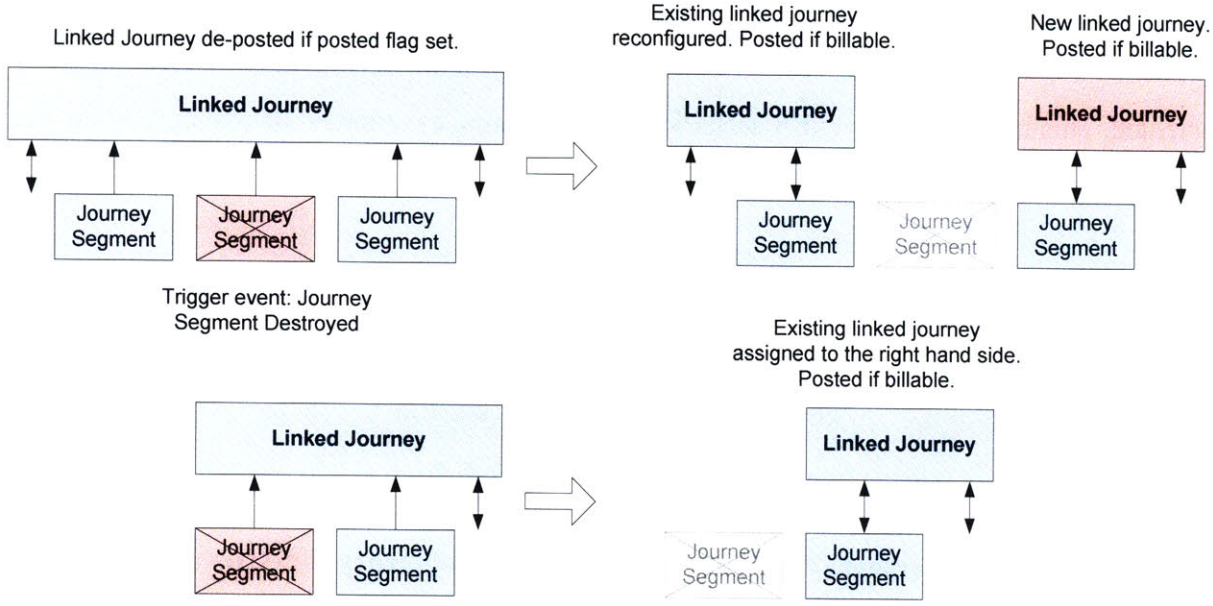


Figure 7.18 – Scenario D: Destruction of a journey segment claimed by a chained linked journey. The special case for deleting a left-most journey segment is shown.

7.6.3 Alternative Approach

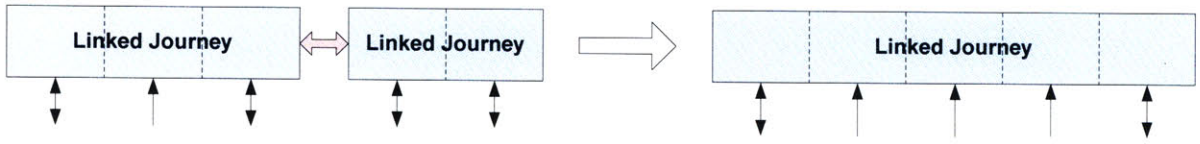


Figure 7.19 – Create and merge, an alternative journey linking paradigm.

In the examples above, we recognize that any action on linked journeys must be triggered by changes to journey segments. The reconfiguration of linked journeys takes place during a function call made upon creation or deletion of journey segments. We'll call this journey linking paradigm *trigger-and-reconfigure*.

An alternative approach to journey linking would be to wrap a unitary linked journey around any new journey segment. The unitary linked journey is then merged laterally with its neighbors in what we'll call a *create-and-merge* process. The advantage of this approach over trigger-and-reconfigure is that we would only need to implement one set of lateral merging logic, as opposed to the numerous tailor-made case handlers above. This results in a simpler and more general design. However, as a consequence, we trade performance for complexity.

In trigger-and-reconfigure we strive to reuse existing linked journey objects as much as possible and avoid the creation of new objects where they are not needed. On the other hand, create-and-merge is a wasteful approach. The lateral merging process leads to the creation and rapid abandonment of short-lived linked journey objects. In comparison to modifying an existing object, creating a new object in memory is a costly operation [19]. Ephemeral objects incur a penalty not only in the memory space that they use, and the cost of creation, but in increased garbage collection activity to remove them once they become redundant.

7.6.4 Posting and De-posting

Post() and De-post() are functions of the fare processor. Post() exists to furnish the fare processor with a description of the linked journey to be billed, including the location and timestamp of all its intermediate segments. The parameters of this call will be described in depth in our later discussion of the fare processor.

Conceptually, De-post() is an instruction to completely reverse the charges for a linked journey. This operation is necessary when new information (i.e. taps) arrive that results in a reinterpretation of existing, already-billed linked journeys. Rather than computing the difference in cost between the old and new interpretations, de-post() allows the old interpretation to simply be retracted, so that the new interpretation can be posted as if it had never been seen before.

The journey processor has internal versions post and de-post(). The purpose of these internal versions is twofold. Internal linked journey objects (JPLinkedJourneys) sit at the top of the now ubiquitous multi-tiered data structure and contain no actual data themselves. External linked journeys, on the other hand, are fully contained java beans with real data fields. The internal post and de-post functions exist to convert internal linked journeys into external linked journeys. In addition, they manipulate the *posted flag*.

The *posted flag* is a binary state variable which tracks whether a journey segment has been posted successfully or not. It is important to keep track of the posted state for two reasons. A linked journey that has been posted needs to be de-posted any time it is changed or removed in the future (except during cleanup). At cleanup time, linked journeys which do not have their posted flag set are assumed to be incomplete or problem journeys. These are sent to a bad-journey processor (a part of the fare processor) for additional analysis and processing.

- **Post()** – The linked journey is submitted to the fare processor for fare calculation and billing. If posting is successful, the posted flag is set to true, and the billed amount is recorded. The posted flag must not be set at the time of posting. Posting a linked journey which has a posted flag set causes an exception. Note that the journey processor is generally unconcerned with the billed amount of a journey. It is only recorded so that if a linked journey were to be de-posted, the amount to de-post could be found immediately without having to feed the linked journey through the fare processor a second time. Having said this, reprocessing a linked journey through the fare processor during de-posting in order to find out how much should be deducted from the bill is a legitimate operation. The fare processor is completely stateless with regard to each input linked journey. In other words, the buffering of the posted amount is purely a performance optimization.

- **De-Post()** – A linked journey can only be de-posted if its posted flag is set. De-posting a journey which has not been posted causes an exception. Otherwise a de-post call to the fare processor is made with the details of this linked journey, particularly the buffered posted amount. The fare processor fast-tracks de-post calls carrying a fare amount by completely bypassing the fare calculation routine and immediately sending the call to the billing engine. After a successful de-post operation the posted flag is reset.

7.7 *Cleanup Routine*

Previously we have mentioned that taps can only be inserted, and cannot ordinarily be removed except via cleanup. Without a cleanup process, taps would continuously be inserted and corollary journey segments and linked journeys would continuously be created. This would result in growth in memory usage without bound, a clearly unsustainable outcome.

The cleanup process follows a few simple rules:

- Cleanup begins with the tap at the left-most end of the data structure (*the head tap*). This is the oldest tap in the system.

- If the head tap is an orphaned tap (no journey segment), it is dispatched to the *orphaned-tap department* of the fare processor. The tap is removed and the next successive tap now becomes the head tap. The cleanup process is restarted with the new head tap.

- If the head tap is not an orphaned tap, then it must have one (and only one) associated journey segment, and that, in turn must have an associated linked journey. It is an invariant that the head tap cannot be claimed by two journey segments. Also recall that a journey segment must be claimed by one and only one linked journey. The identified linked journey is our *target linked journey.*

- If the target linked journey has been posted, no action is taken on this step. If the target linked journey has not been posted, it is dispatched to the *incomplete linked journey department* of the fare processor.

- The target linked journey is systematically dismantled, and its constituent journey segments are deconstructed from left to right. Taps are removed only when they are not claimed by any journey segment.

- After the last tap associated with the target linked journey has been removed, the next successive tap becomes the head tap. The target linked journey is itself removed.

- The cleanup process begins again with the new head tap.

- One complication arises when a linked journey finishes on an EPVAL. This is a very rare condition that can only occur if the subsequent taps have been disqualified from the linked journey on some ground (e.g. global linked journey duration exceeded). In this case, the last tap is not removed during the iteration. When the iteration finishes, this residual tap rather than the next tap in sequence is assigned as the head tap.

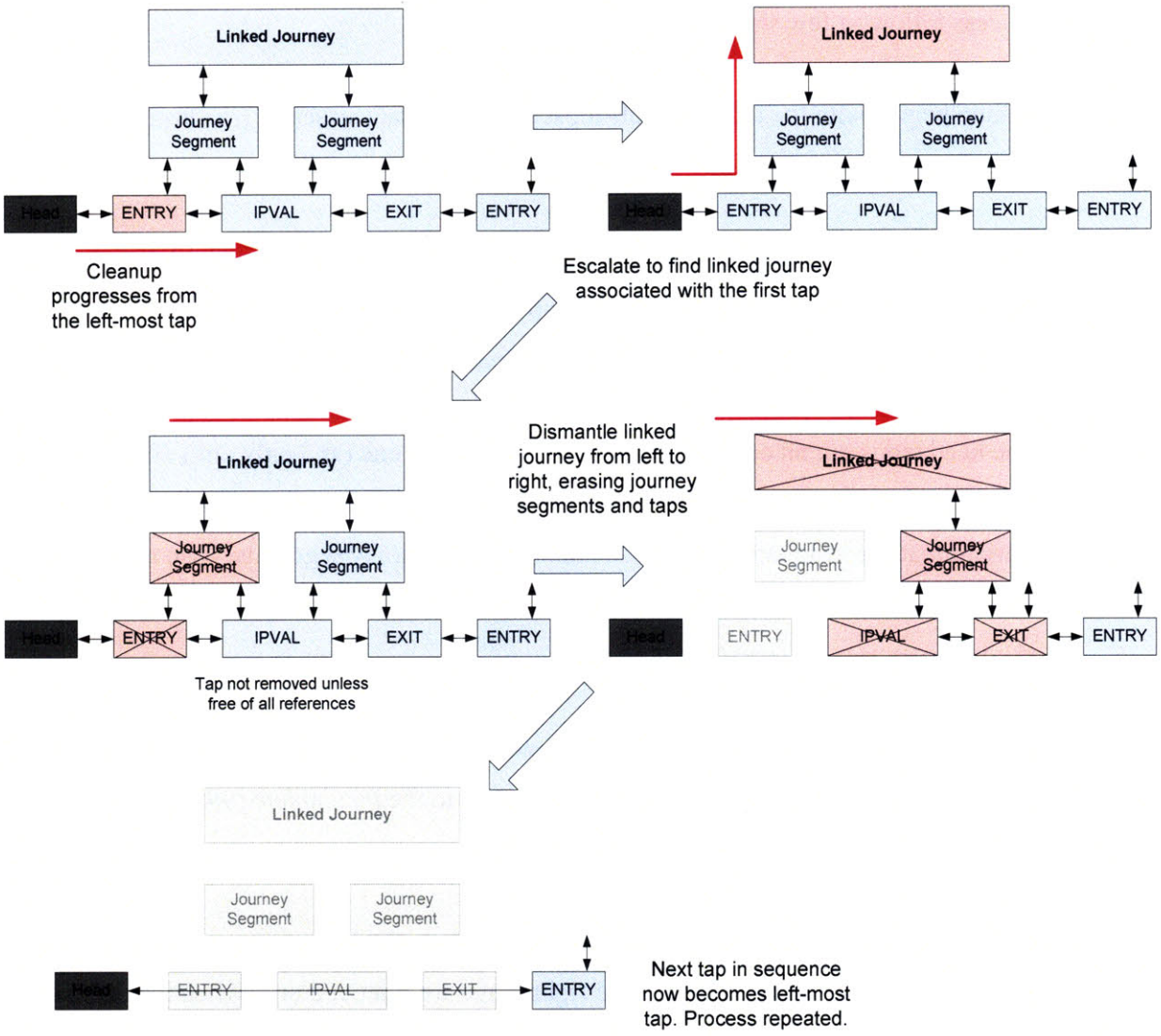


Figure 7.20 – Cleanup process.

As discussed previously, objects cannot be actively removed in modern object oriented languages such as Java. When we deconstruct or remove an object (whether it is a tap, journey segment or linked journey), we are simply eliminating references to the object so that it will eventually be recycled by the garbage collector.

Cleanup runs until the timestamp of the head tap encountered has a timestamp which is equal to or later than the specified cleanup threshold, or until the data structure is completely empty. Cleanup for this particular user is then complete.

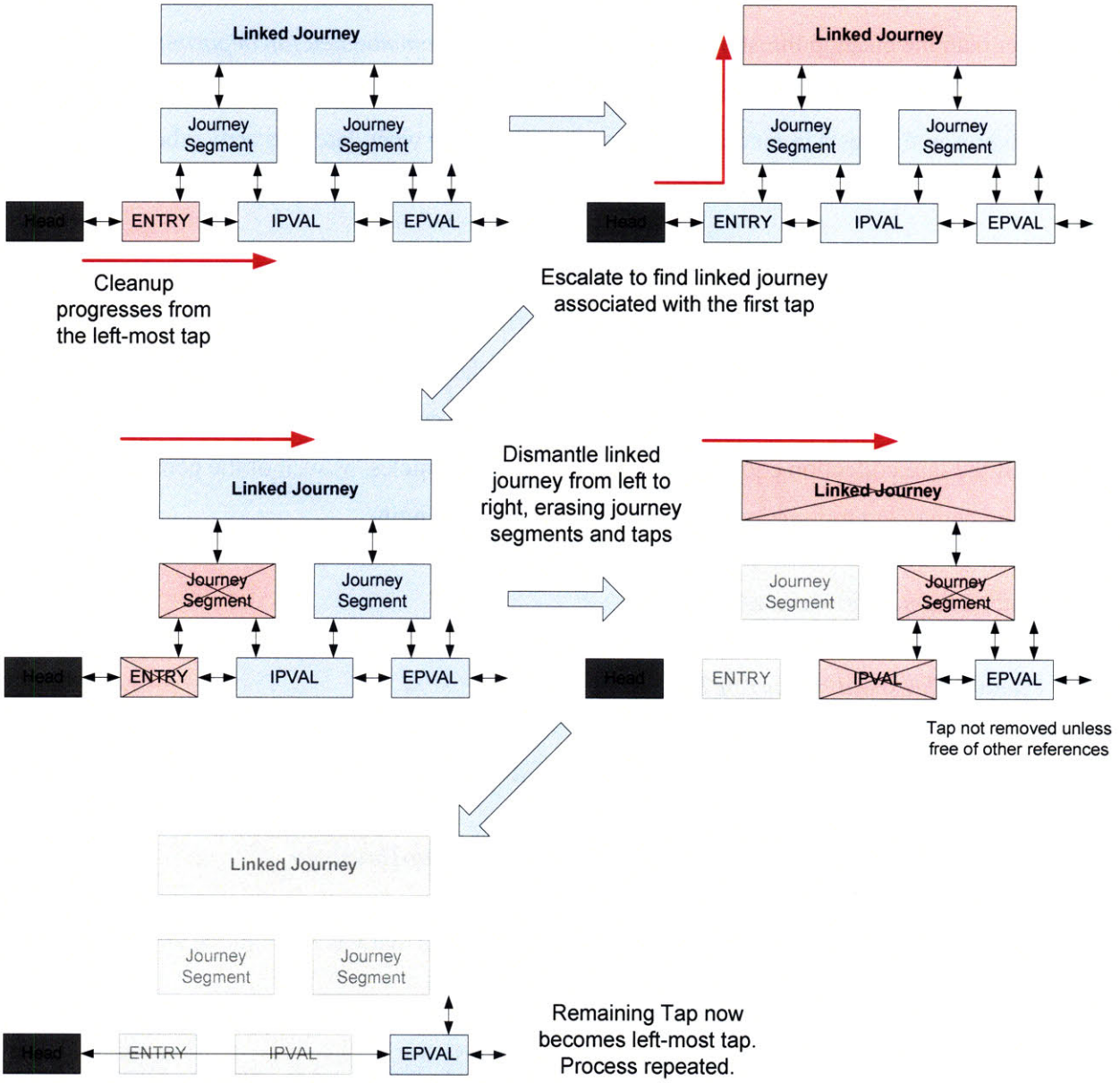


Figure 7.21 – Cleanup case where the last tap cannot be removed. This is a rare complication. The process is largely similar to the previously described standard case.

7.7.1 Cleanup Cycle

We will first lay out some groundwork in our discussion of cleanup cycles. If an out of order (late arriving) tap cannot be accommodated into the data structure because it is too old (it has a timestamp dated earlier than the cleanup threshold), it is sent to the *un-accommodated tap department* of the fare processor. In effect such taps are discarded. The discarding of over-late taps is undesirable and so we try to preserve objects in memory for long enough so that when late taps arrive they can still be placed into sequence

We define the *Time-to-live* (TTL) to be the minimum amount of time for which a tap must have existed in memory before it is erased. Set this time too short, and many late taps will have to be discarded. Set this value too long, and we will overrun available memory and/or impact the system performance. The lower bound for the TTL value is subject to calibration against the operational reliability of the fare collection equipment at stations and on vehicles, as well of the communication infrastructure conveying transactions from the field to the fare engine.

We define the *cleanup threshold* as follows:

$$CleanupThreshold = CurrentTime - TTL$$

With this, we can implement a *cleanup rule*. Note that a larger timestamp has a later time.

Cleanup Rule: *Remove tap if Tap.Timestamp < CleanupThreshold*

The cleanup rule binds the reach of the cleanup process. It states that we will remove any tap we see which is older than CurrentTime – TTL.

Finally, we define the *max-age* to be the age of the oldest data object (tap) in memory. Max-age is always greater than TTL.

7.7.1.1 Rolling and Fixed Cleanup

Two broad classes of cleanup cycles are possible. In a rolling cleanup schedule, cleanup is an ongoing process that is repeated at short intervals and not timed to an absolute schedule. Older taps are stripped away as time progresses, ensuring that the max-age of the data structure approximates the TTL which we have set.

In a fixed cleanup cycle, the cleanup operation is scheduled to occur at recurring time points which are defined in an absolute timeline. The time between scheduled cleanups is the *cleanup interval.* An example of a fixed cleanup schedule is a daily schedule where cleanup occurs at 3AM every day – this has a cleanup interval of 1 day. Under a fixed cleanup schedule, the age of the database is always older than the TTL. Specifically, the maximum age reached by the database is given by

$$MaxAge = TTL + CleanupInterval$$

In the example above, a TTL of one day and a CleanupInterval of one day means that at the time of cleanup, taps a in the data structure go back two days. This is significant as memory usage is determined by the MaxAge, not TTL. A finite amount of physical memory space is available. For performance reasons we wish to keep the heap size below size of our physical memory in order to avoid any part of the data structure being swapped into virtual memory[22]. Aspects of the virtual machine executing the journey processor software (such as the performance of the garbage collector) may also restrict the cleanup interval.

A journey processor that uses a fixed cleanup cycle must be planned on the basis of MaxAge, not just the TTL.

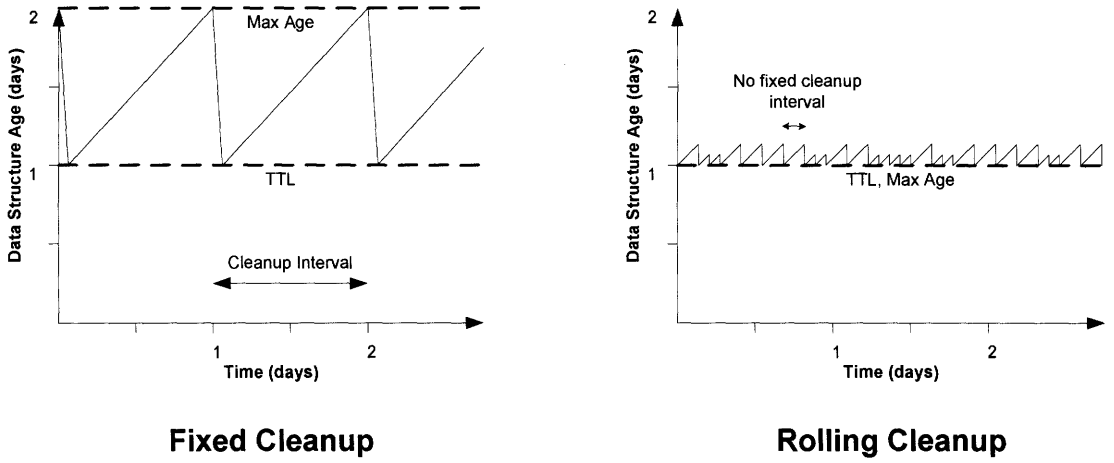


Figure 7.22 – Fixed vs. rolling cleanup.

[22] Virtual memory is used when the amount of memory used exceeds the system's physical memory. Blocks of data in memory (pages) are swapped out to temporary storage on disk to make room. This results in a heavy performance penalty as that data must be swapped to memory before it can be accessed.

7.7.2 Multi-threading Considerations

Up to now, multi-threading has not been a significant issue. For a given user, object manipulations take place in a well defined chain of execution where method calls are made and returned sequentially. At the same time, different users have completely non-intersecting object spaces, allowing us to divide the work of journey processing for different users into multiple threads with impunity, if we wish to do so.

7.7.2.1 Batch Cleanup

Multithreading issues in cleanup can be avoided completely if we dictate that the cleanup of objects in memory will occur only during the journey processor's down-time.

This approach is used with a fixed cleanup cycle of a relatively long cycle length. As we have discussed, fixed cleanup cycle incurs additional memory usage compared to a rolling cycle. However this is not a fatal impediment, as memory is a low cost and abundant resource. Furthermore, the long cycle length does not affect business capabilities or operations in any way. A long cycle length does not negatively impact the real-time posting and processing of linked journeys.

One drawback of phased cleanup is that we must halt fare engine processing during the cleanup process. All incoming taps need to be buffered in a queue so that they can be processed when cleanup completes. System throughput is expected to be extremely low during the early hours of the morning, providing a window of opportunity for executing once-daily cleanup. If the cleanup time can be reduced to an order of seconds, it should be acceptable to suspend the journey processor momentarily for cleanup even during daytime operations.

We have a feasible compromise. Nonetheless we shall consider more advanced techniques that will not result in service disruptions.

7.7.2.2 User Level Locking

Let us consider how concurrency problems can arise in journey processing. On the one hand, the insertion of taps and the subsequent object manipulations that follow occur in real time, at arbitrary moments as determined by the user's actual travel behavior. On the other hand, we have a cleanup process which operates either on a fixed schedule based, or on a rolling basis, but in any case, on a basis which bears no obvious link in causality to the insertion of taps.

Without further accommodation, the system would enter an inconsistent state and fail if we tried to insert a tap into a linked journey in the midst of that journey being deconstructed. Likewise, if the cleanup process attempts to destroy a linked journey whose construction is still in progress, the system would also enter an undefined state and the data structure would become corrupted. This is the simplest description of why we have a concurrency problem.

User level locking is one solution to this problem. We require both insertTap() and cleanup() to obtain a lock on a user's JPUser object before they can perform any action on that user. Object locking (or synchronization) is directly supported in Java, using the *synchronized* block.

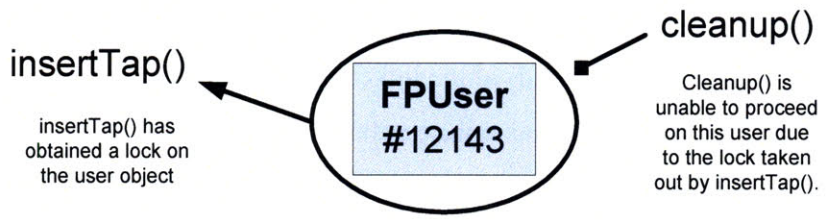


Figure 7.23 – Locking on an JPUser object.

If insertTap() has locked a given user and is operating on it, and cleanup() attempts to operate on the same user, it would be unable to do so because the lock that insertTap() has taken out would prevent cleanup() from obtaining its own lock (recall that we require each function to obtain a lock on an object before touching it). In the default Java implementation of locking, cleanup() would be asked to *block*, that is, to wait until insertTap() is finished, at which point it would remove its lock and free up the object. While it is technically preferable in the context of a cleanup process to skip over a blocked user and move onto the next user rather than sit idle waiting for it to free up, given that cleanup is strictly an optional process, to do so in an implementation would involve exchanging the standard java synchronization mechanism for a custom implemented solution, a daunting task that cannot be justified given the virtually imperceptible benefits. Insertion-cleanup collisions on the same user are extremely rare events, and the delay in incurred when such a rare collision does occur is also negligibly small.

User level locking is useful not only in preventing insertion-cleanup collisions, but as a side effect it also opens up the opportunity for multiple insertion threads to run concurrently. Up to now we have assumed that taps arriving from the device manager are processed sequentially in a single thread (the

device manager maintains a queue which buffers taps as they arrive physically from all points in the TfL network). However, on a dual or multiple core system, we have the option of dispatching multiple threads (a number commensurate with the number of CPU cores) to clear out this queue. Most of the time, successive taps will concern different users. User lever locking will very cleanly and effectively cater to the few rare occasions where we have both threads (assuming a dual core CPU) attempting to insert taps into the same user.

7.7.2.3 Journey Level Locking

Another optimization that we will mention briefly, if only for academic purposes, is the locking of specific journey objects. One can identify a theoretical weakness in the user level locking approach. Recall that cleanup proceeds from the left-most (head end) of the linked list of taps. On the other hand, the vast majority of tap insertions are expected to take place at the right-most (tail end) of the list, given that usually arrive in order. There is no reason why a tap cannot be inserted at one end of the structure while the other end of the structure is being dismantled. In fact, operations can be allowed simultaneously anywhere on the data structure as long as they do not spatially interact.

Observe that the insertion of a tap directly modifies its two adjacent taps. Also note that the cleanup process leaves the entire structure beneath a linked journey in an inconsistent state while it is in progress. The linked journey is the natural object to lock on. We can require cleanup() to lock on a linked journey before dismantling it, at the same time, we can ask insertTap() to obtain locks on the linked journeys associated with *both* neighboring taps, before it inserts a new tap between them. Special cases with orphaned taps and edge cases can be catered for accordingly.

However because insert-cleanup collisions are so rare, and the consequence of a collision is so imperceptible, not only would we not gain anything from implementing this elaborate fine-grain protection mechanism, we would in fact lose substantial performance due to the added overheads in locking and unlocking, and the traversals needed to find the linked journeys that we want to lock.

7.7.2.4 Recommended Solution

The recommended solution is user level locking using standard java synchronization. TTL can be set based on system reliability and physical limitations of the server configuration.

8 Fare Processor

The second sub-component of the fare engine is the fare processor. This unit is responsible for calculating the fare incurred for a given linked journey, taking into account any period ticket that the user may be holding.

8.1 *System Overview*

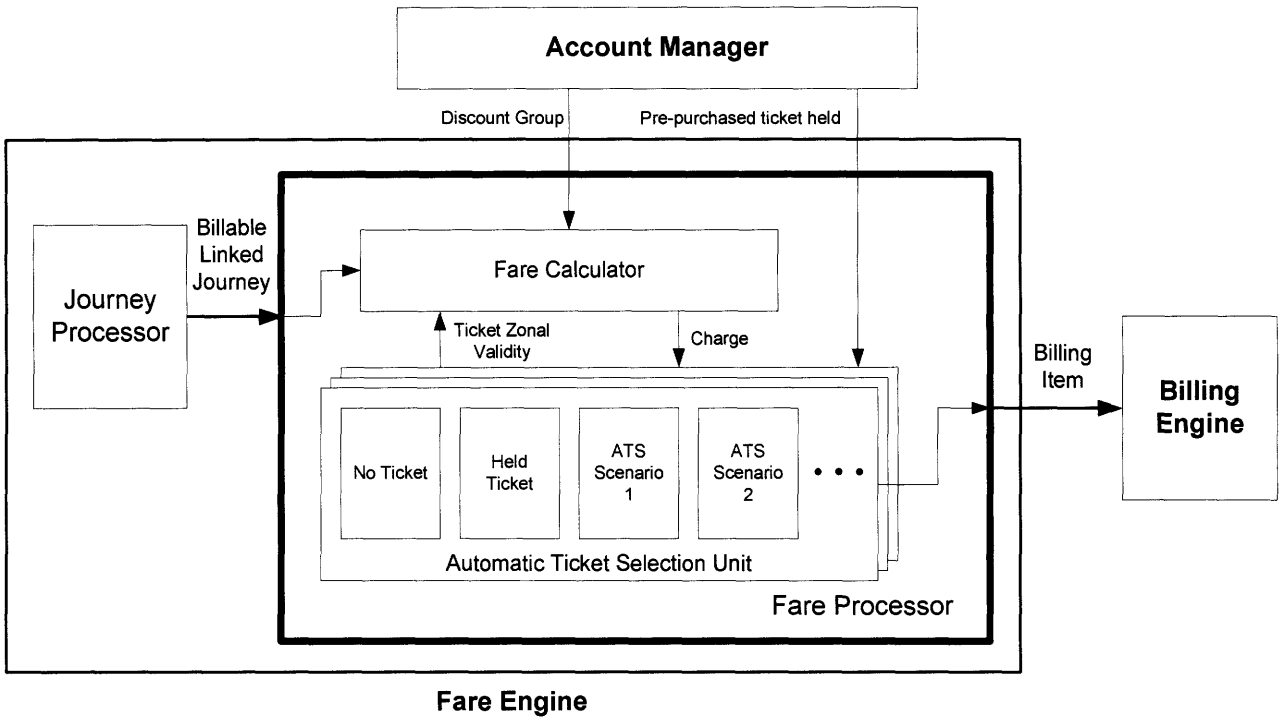


Figure 8.1 – Fare processor overview.

The fare processor is composed of two units, the Fare Calculator and the Automatic Ticket Selection Unit (ATSU). External systems that interact with the Fare Processor include the Account Manager and the Billing Engine. The fare processor receives its input directly from the Journey Processor.

8.2 *Inputs and Outputs*

The fare processor accepts LinkedJourney objects as described in 6.3.2, and outputs BillingItem objects, as described in 6.3.3.

A notable feature of the fare processor from a data flow perspective is that it is practically pass-

through system. In other words, the LinkedJourney objects which the fare processor accepts as an input are augmented with missing information and then passed onto the billing engine as BillingItems. In particular, we dispose of the extremely lightweight LinkedJourney object (essentially a placeholder) and replace it with a BillingItem, while recycling the list of journey segments (NodeList (an ArrayList) and its children JourneySegments). The alternative would have been to abandon and recreate the ArrayList and its children from start, a process that is wasteful both in memory space and processing power.

On the other hand, the JourneySegment objects referenced by LinkedJourney are cloned. Within the journey processor, recall that JPLinkedJourneys are merely proxy objects that reference the underlying JPJourneySegment objects, which in turn reference JPTaps. We want to detach the fare processor from this arrangement for two reasons. First, as a rule of encapsulation, the internals of the journey processor should only be accessed by elements of the journey processor, and not by classes outside of the journey processor package.

Secondly and more importantly, we want the fare processor to be able to run asynchronously from the fare processor. For asynchronous processing to work, we cannot allow linked journeys to be changed (or even destroyed) in between the time they are submitted for fare processing and the time they are actually processed. Therefore the interface between the journey processor and the fare processor must be of a 'fire-and-forget' nature, that is, proxy linked journeys must be cloned into stable linked journeys. Note that within a prototype, fare processing may occur sequentially (in the same thread) as journey processing. However, we maintain the option to break apart journey and fare processing into separate threads. Billing is clearly an asynchronous process and the need for billing items to be stable is not disputed. As mentioned prior, the billing engine can reuse the bulk of the immutable linked journey objects accepted by the fare processor.

8.3 *Fare Structure Assumptions*

The nuances of the current TfL fare structure were described in chapter 3. There, we found that TfL currently operates under a dichotomous fare structure, with a single product branch and a period product branch. Furthermore, we found that the each branch can be broken down systematically using a series of decision tiers, culminating in the concept of a charge code, which encapsulates the effect of a zonal structure.

TfL's fare structure is under constant flux and there is no reason to assume that future ticketing will operate under same structure as today's Oyster system. In the process of describing our fare processor system design, we will gradually point out assumptions that we have made about the fare structure under future ticketing. The goal of these assumptions is to satisfy our design requirements without necessarily mirroring the way Oyster works. Many technical limitations and legacy paradigms stemming from historical precedence have been folded into Oyster. For example, the "capping" algorithm used to implement best value in Oyster was motivated largely by the conservation of storage on Oyster Cards. Given the opportunity to design a system from ground up where these technical and historical constraints no longer apply, we will explore some new approaches that may depart significantly from the existing paradigm. However, we attempt to preserve backward compatibility where possible. Simplifications are made where it is possible to illustrate a concept without undue complexity (for example, the number of discount groups is limited to 3).

The first and foremost assumption we make in this fare structure is that we will deal with only one fare medium, that being contactless bankcards. Whether existing fare media, such as Oyster and cash/magnetic stripe will continue to operate in their existing form, or be adapted to become consistent with the contactless bankcard fare structure, or be eliminated altogether is a question which lies outside the scope of our discussion.

8.3.1 Bus Journeys

We adopt a very simple policy with regard to bus journeys. Stand-alone bus journeys are charged a fixed price based on the user's discount group. Bus journey segments which appear in a compound linked journey (as a result of the cross-mode interchange rule) are free of charge. They are simply stripped from the linked journey and ignored. Finally, if the user has any ticket holding whatsoever, bus journeys are made free.

8.4 *Fare Calculator*

The fare calculator accept three inputs, a *linked journey* from the fare processor, the *discount group* which the user belongs to from the account manager, and the *zonal validity* of the ticket held, if the user has a pre-purchased ticket. The zonal validity is also provided by the account manager. The fare calculator's outputs are the *fare* and the *revenue allocation code* for the journey. The fare is the

amount, in pence, which the given journey will cost the user, given his discount group status and his ticket holding. Note that the fare produced by the fare calculator does not account for any best-value capping. The revenue allocation code (ROC) references the primary key of a revenue allocation table. This value will be explained below.

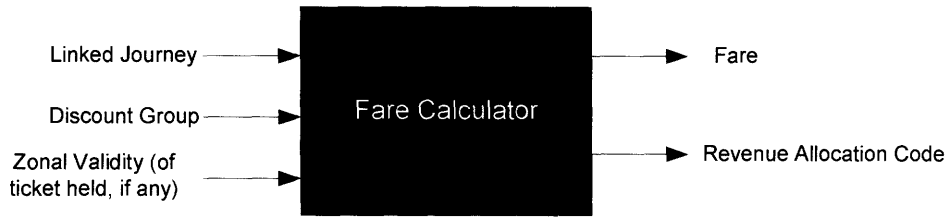


Figure 8.2 – Fare calculator in a nutshell. A black box view.

What we have described is a black-box view of the fare calculator, or a contract between the fare calculator and the fare processor. An important point to note is that the fare processor is not concerned with how the fare calculator computes its outputs. The fare calculator could be generating random numbers. As long as the fare calculator generates the same output repeatably on multiple trials given the same input, it is legitimately fulfilling its contract to the fare processor. In other words, the fare calculator must be a stateless function. Of course, in practice the fare calculator would not be a random function. The fare calculator is where much of the TfL fare structure is implemented. As TfL's fare structure for future ticketing is still under development, we will proceed with an example design that provides flexibility for National Rail integration while incorporating the significant investment that TfL has already made toward this goal.

8.4.1 Handling of Fare Zones

Our contactless bankcard (CLBC) fare calculator uses a fare zone system modeled geographically on existing TfL fare zones, which take the form of concentric rings centered on Central London (see 3.4.2.1). However, unlike the current TfL network, CLBC stations are not required to belong to a fare zone. Fare zones in the fare calculator serve two purposes:

1. The fare zones spanned by a journey *may* be used to determine the journey's single fare.

2. Period tickets are differentiated and priced in terms of the fare zones covered. Following established nomenclature, we call this coverage the ticket's *zonal validity*. Fare zones *may* be

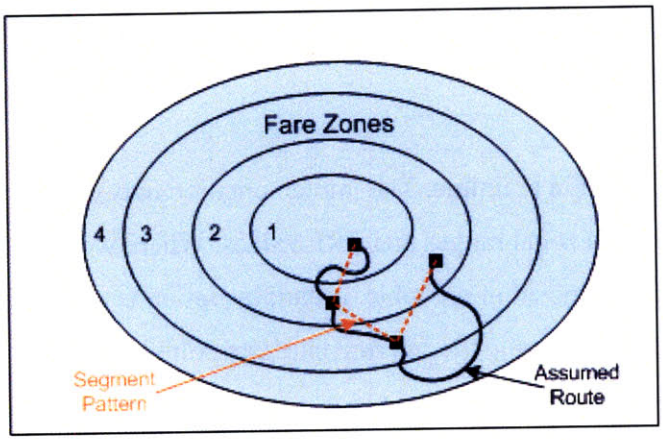
used to determine the applicability of a period ticket to a journey. The fare *may* be waived for the portion of a journey that is covered by the zonal validity of a ticket held by the user.

In the above description we have emphasized that both of these uses of fare zones are optional. Not all single fares have to be computed in terms of fare zones. Similarly, not all journeys have to be eligible to be covered by a period ticket. Below we will explore the consequence these statements.

The single fare of any journey can be broken down into two components. A *zone reducible fare* component and a *non reducible fare* component. The total fare is the sum of these two components. We will define zone reducible and non reducible below.

8.4.1.1 ***Zonal Reducible Fare***

A *zone-reducible fare* (ZRF) is a fare which can be reduced through the use of a ticket. It is the fare contribution assignable to a journey occurring inside of the defined fare zones. Note that the part of journey over which a ZRF is computed need not be coterminous with the empirically recorded journey segments. The application of a ticket could trigger the computation of 'residual' ZRF journeys which are not coterminous with recorded journey segments.



Zonal Range: JourneyInnerzone=1, JourneyOuterzone=4

Figure 8.3 – A Journey made within the fare zones has a characteristic Zonal Range and ZRF.

Each ZRF has *zonal range* associated with it.

ZonalRange is defined as [JourneyInnerzone, JourneyOuterzone]

ZonalRange = zonalBounds(segment pattern)

The user's ticket has a *zonal validity.*

ZonalValidity is defined as [TicketInnerzone, TicketOuterzone]

It is very important to realize that although every ZRF has a zonal range, a ZRF **does not have to be computed in terms its zonal range**.

An example of this is National Rail journeys within London. Nationwide, National Rail single fares are set on the basis of the origin and destination station of the journey, and are published in the *National Fares Manual,* while permitted routes for each OD pair are published in the *National Routing Guide*. The fare for a given journey is determined by a complex formula for which there is no compact expression, although it is generally commensurate with the distance travelled. However, despite the fact that NR single fares are decidedly non-zonal, zonal Travelcards allow users to travel on National Rail services throughout London. Here we have a situation where a fare can be waived if the user holds a ticket with the right zonal validity, but the fare itself is not based on fare zones.

To implement ZRFs, we can begin with the linked journey. Each linked journey has a characteristic profile of journey segments. A segment profile is essentially a refined form of the 'syntactic pattern' described in section 7.4.2.1. For each segment profile, we compute a 'most likely path'. The zonal range is assigned based on the zones traversed by the assumed route. Note that each segment profile is associated with its own ZRF.

Note that ZRFs and zonal ranges need not be computed in real time. The implementation may consist of a large table of segment profiles, with pre-computed zonal ranges and ZRF values. Where we encounter a compound segment profile that is not recognized in our table, a standby option is to merge the journey segments in that profile, effectively ignoring the intermediate waypoints and reprocessing the journey with only its origin and destination.

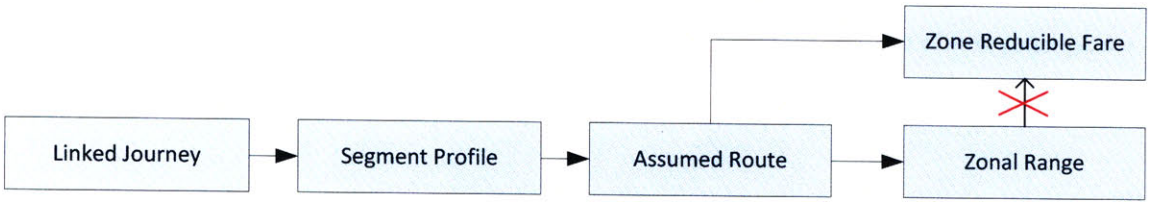


Figure 8.4 – ZRF Implementation. Note that the ZRF is not computed in terms of the Zonal Range.

National Rail single fares do not translate well into TfL fare zones. Not only do zonal fares offer significantly less granularity than non-zonal fares, they create some obvious discrepancies. For

example, a circumferential journey would incur a lower fare than a radial journey of the same length. Unless the Train Operating Companies (TOC) can agree to adopt single fares that are calculated in terms of TfL's fare zones, a fare collection system seeking to support National Rail must be prepared to implement ZRFs that are arbitrarily defined. This is the reason we cannot require ZRFs to be computed in terms of their zonal ranges.

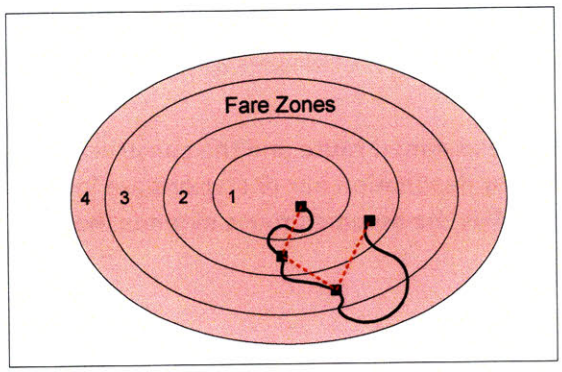
We will now take a look at how ZRFs behave in the presence of a ticket.

If the zonal range is contained fully inside of the zonal validity, the zonal reducible fare is zero.

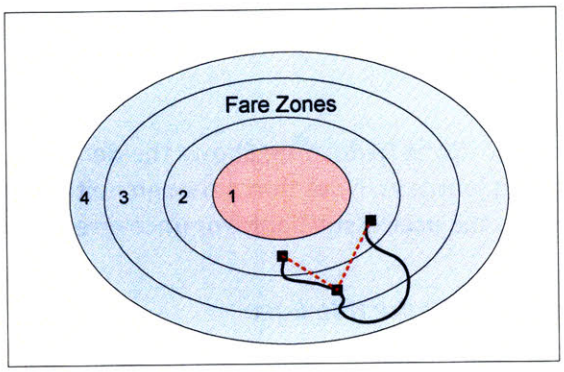
IF JourneyInnerzone >= TicketInnerzone AND JourneyOuterzone <= TicketOuterzone
ZRF=£0

If the zonal range does not intersect the zonal validity, the zone reducible fare is determined as a function of the origin, destination and waypoints.

IF JourneyInnerzone > TicketOuterzone OR JourneyOuterzone < TicketInnerzoneFare
ZRF = ZRFfarefunc(origin, destination, waypoints)



Zonal Range: JourneyInnerzone=1, JourneyOuterzone=4
Zonal Validity: TicketInnerzone=1, TicketOuterzone=4
ZRF: £0

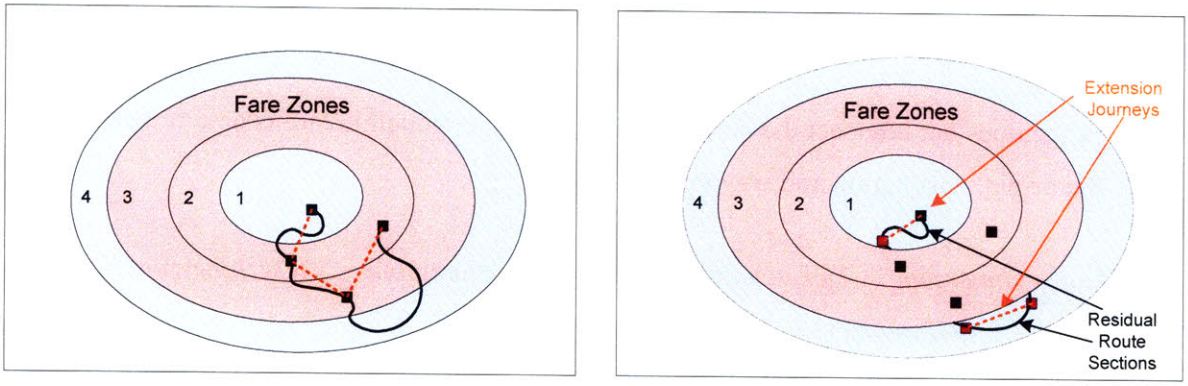


Zonal Range: JourneyInnerzone=2, JourneyOuterzone=4
Zonal Validity: TicketInnerzone=1, TicketOuterzone=1
ZRF: ZRFfunc(segment pattern)

Figure 8.5 – Left: Zonal validity fully contains zonal range of journey. No charge for journey. Right: Zonal validity does not intersect zonal range of journey. Full ZRF of journey is charged.

If the zonal range intersects the zonal validity, the assumed route of the journey is decomposed. The sections of the route which reside in the zonal validity of the ticket are removed (reduced away). The remaining sections of the original assumed route are the *residual route sections*. Next, we take

residual routes and construct *extension journeys* out of them. One extension journey is created for each contiguous residual route. These extension journeys are then queried for their corresponding *extension fare*. For sake of simplicity, extension journeys can be processed as a regular journey, and the ZRF returned is used as the extension fare. Note that extension journeys are *not* recursively zone-reduced. In other words, if by some unlikely occurrence the assumed route of an extension journey were to venture back into a zone which we have reduced away as part of a ticket holding, we would simply ignore it and use the extension fare as-is. The ZRF for the original journey as whole is computed as the sum of the extension fares of the two residual ranges.



Zonal Range: JourneyInnerzone=1, JourneyOuterzone=4
Zonal Validity: TicketInnerzone=2, TicketOuterzone=3
ZRF: Zone Reduction ...

Zonal Range: JourneyInnerzone=1, JourneyOuterzone=4
Zonal Validity: TicketInnerzone=2, TicketOuterzone=3
ZRF: ZRFfunc(ExtJry1) + ZRFfunc(ExtJry2)

Figure 8.6 – Zone Reduction: Shows the deconstruction of the assumed route and the construction of extension journeys. Note that although neither the origin, nor destination nor IV stations are located in zone 4, the user is still liable for uncovered travel in zone 4 as the assumed route extends there.

8.4.1.2 Non-Reducible Fare

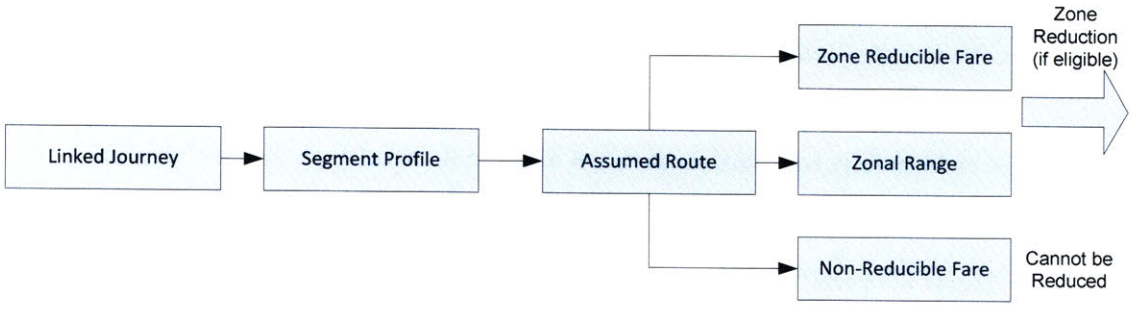
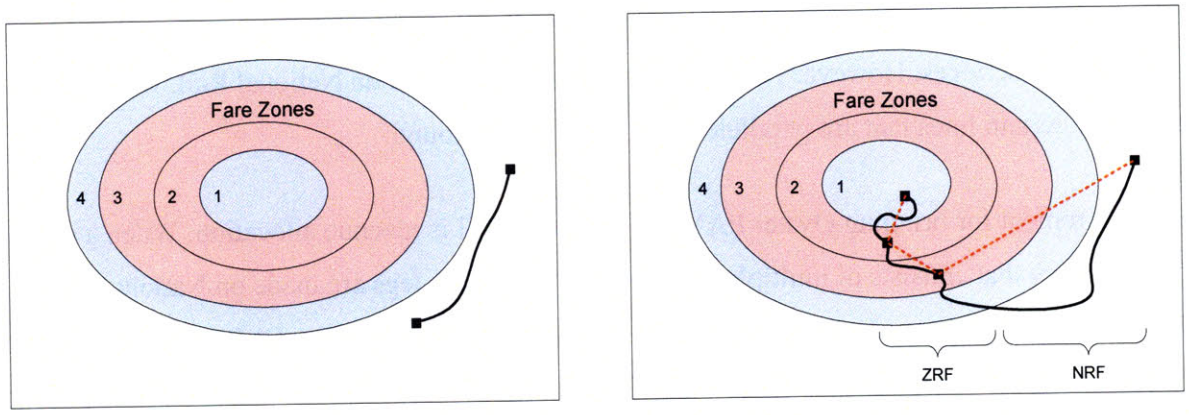


Figure 8.7 – Figure 8.4 updated with NRF.

A *Non-reducible Fare* (NRF) is the portion of a journey's fare that cannot be reduced by a zonal period ticket. The NRF represents the fare incurred when travelling outside of the defined fare zones.

The non-reducible fare is introduced in order to accommodate journeys to NR stations that are located outside of central London.

A NRF is defined for each segment profile whose assumed route extends beyond the bounds of the fare zones. The assumed route can be decomposed into reducible and non-reducible legs. Since each segment profile has a fixed assumed route, the NRF is fixed for a given segment profile and can be pre-calculated and stored in a table.



Zonal Range: JourneyInnerzone=N/A, JourneyOuterzone=N/A
Zonal Validity: TicketInnerzone=2, TicketOuterzone=3
ZRF: N/A
NRF: Yes
Total Fare: ZRF + NRF = NRF

Zonal Range: JourneyInnerzone=1, JourneyOuterzone=4
Zonal Validity: TicketInnerzone=2, TicketOuterzone=3
ZRF: ZRFfunc(ExtJry1) + ZRFfunc(ExtJry2) - see prev. ex.
NRF:Yes
Total Fare: ZRF + NRF

Figure 8.8: Left; A journey whose assumed route is located entirely outside the fare zones. Right: A journey with both zone-reducible and non-reducible components.

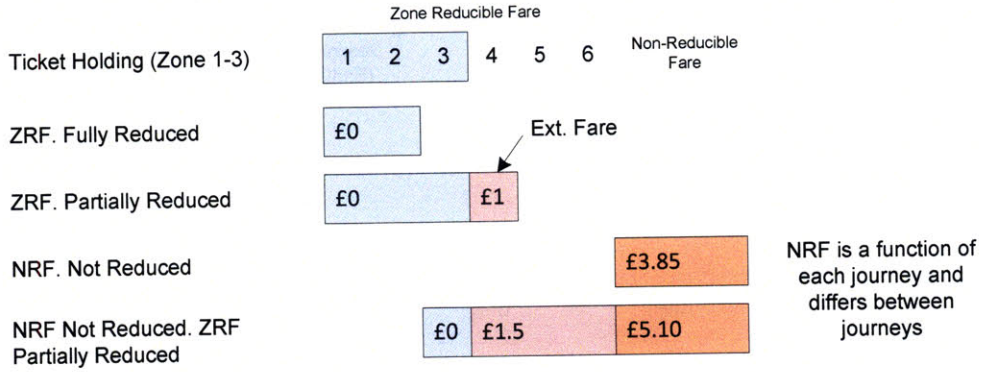


Figure 8.9 – Examples of journeys with zone reducible and non-reducible fares.

Note that the non-reducible legs of a journey do not necessarily have to be at the journey's beginning or end. For example, the non-reducible leg could bisect a journey that both begins and ends in a fare zone. In this case, the ZRF and zonal range of the journey reflects the discontinuous reducible legs.

The zonal range represents the inner and outer most reaches of the discontinuous legs, while the ZRF represents the combined fare contribution of those legs. A discontinuous ZRF can be reduced by a ticket in the same way as would a normal continuous ZRF.

8.4.2 *OXNR Matrix Control System (MCS)*

Oyster eXtension to National Rail (OXNR) is an effort by TfL to extend its existing Oyster Pay-as-you-go (PAYG) system to National Rail services. Currently, Oyster based zonal period tickets are available on NR services. However PAYG is generally not accepted on National Rail, with the exception of certain lines that are paralleled by London Overground.

A key requirement for bringing Oyster PAYG to National Rail is revenue allocation. When a user makes a journey that consists of multiple legs, and some of those legs are made on National Rail services while others are made on TfL services (such as the Underground or DLR), a system must be in place to ensure that the fare collected for that journey is divided equitably between TfL and the TOCs. Note that, for the purpose of revenue allocation, London Overground is considered a TOC and treated in the same way as other TOCs. The Matrix Control System (MCS) is OXNR's solution to fare allocation problem. In addition, it enables support for Intermediate Validation.

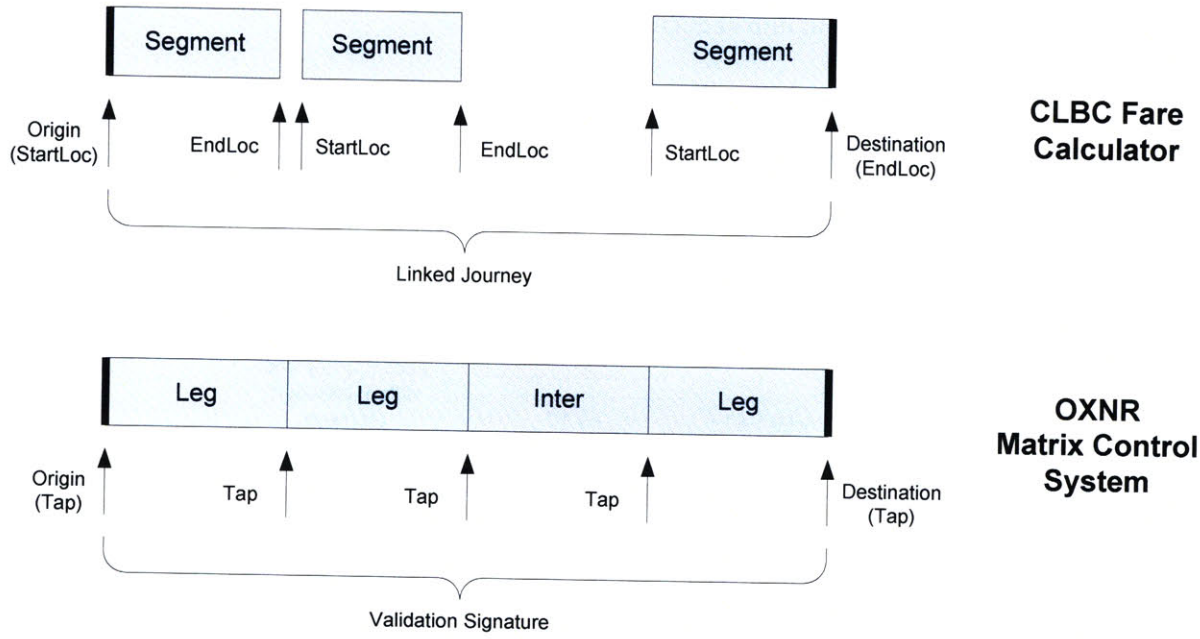


Figure 8.10 –Analogous views of a journey: MCS and fare processor. An excerpt of MCS output is provided in appendix B. For each given OD, the MCS calculates and stores all the plausible interchange possibilities, known as *validation signatures*. And for each validation signature, a path

choice model is used to enumerate the likely journey routes and the market share of each route as percentage of the total flow between the given OD with the given validation signature.

We can see strong parallels between MCS and our fare calculator design. Validation signatures in MCS are equivalent to the *segment profiles* we have described in section 8.4.1, which are themselves derived from linked journeys, the output of the journey processor. The path choices (percentage flows) generated by the MCS for each validation pattern is analogous to the *semantic patterns* we have described in 7.4.2.1. The percentage flow for each path choice can be interpreted as the probability likelihood that that path choice is in fact the true semantic pattern for a given linked journey, with the given characteristic syntactic pattern. It is from this percentage split that a revenue split between TfL and the TOCs is determined. Note that the further splitting of revenue assigned to NR between the individual TOCs is undertaken by the Association of Train Operating Companies (ATOC) using a proprietary model and is not a concern of TfL's.

8.4.2.1 Adapting MCS for the Contactless Bankcard Fare Processing

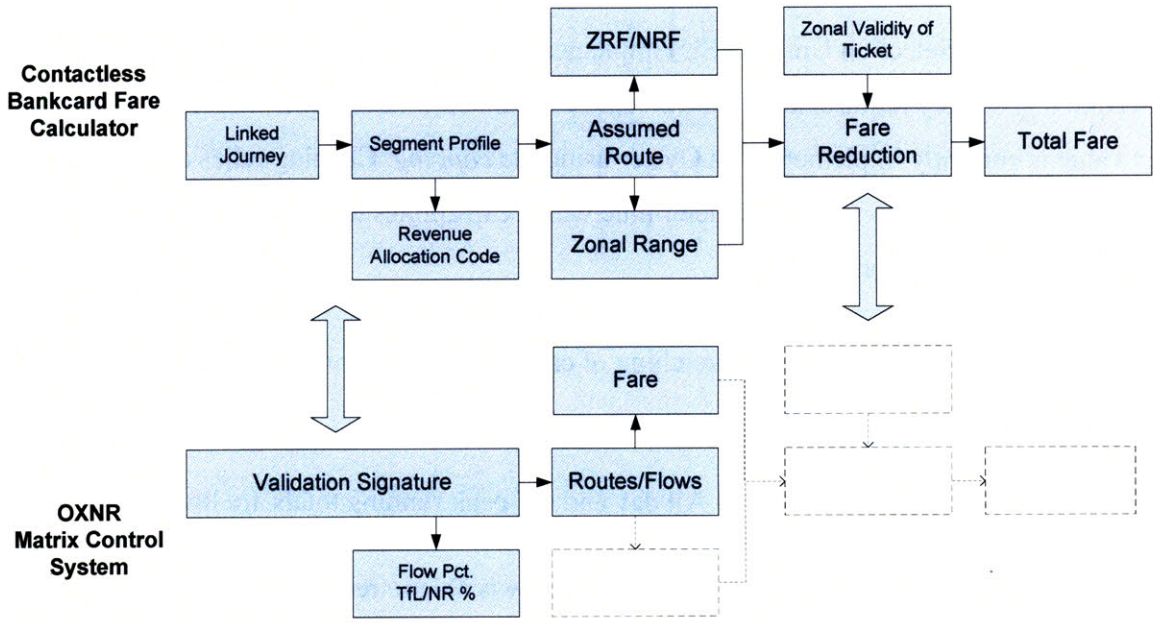


Figure 8.11 – Comparing services: CLBC fare processor and OXNR MCS.

Figure 8.11 shows a side by side comparison of the fare calculator and OXNR MCS services. As the diagram shows MCS is missing some services which the CLBC fare calculator requires. For example, MCS computes a range of feasible routes, but does not select an assumed route and identify the zonal range for that assumed route. The MCS does not differentiate the fare it reports for a given

validation signature into ZRF and NRF components. Finally, zone reduction logic needs to be implemented. However, despite these differences, MCS is structurally similar and suited for direct adaptation into a fare calculator for CLBC fare calculator. This means TfL's ongoing investment in MCS can be reused.

8.4.2.2 Prototype Implementation

For a prototype implementation, the Maciejewski fare calculator [4] can be used as a plug in replacement for the fare calculator module proposed above. The Maciejewski fare calculator calculates single fares on the basis of the zonal range of the selected route. Arbitrary OD-based single fares are not supported. Non-zonal single fares are also not supported. However, intermediate validation is supported and zone reduction can be implemented in the Maciejewski fare calculator with minimum modification.

8.4.3 Automatic Ticket Selection Unit

The Automatic Ticket Selection Unit (ATSU) implements *best value*, a feature of the existing TfL fare description and a key requirement of the contactless bankcard fare engine. Previously we saw that best value is currently implemented in Oyster using the *capping*. Capping relies on a system of stretching windows and running totals to determine which cap applies to the user. This approach suffers from the following drawbacks.

1. **Window overextension** – Over stretching of capping windows result in sub-optimal best-value decisions.

2. **Interference across time bands** - All day and off-peak running totals are linearly dependent.

3. **Non-contiguous zonal usage** – Stretching window is able to record only Contiguous zonal usage. 'Skipping' zones by non-Oyster means results in sub-optimal best value.

4. **No support for best value over a week or longer periods** – The capping architecture is not easily adopted for capping over longer period e.g. 'Weekly capping'.

The design of Oyster capping, with its limitations, was largely motivated by the decentralized architecture of Oyster and the capacity constraints of the Oyster card. Freed from these constraints, we are able to explore alternative best-value implementations.

In this implementation of the ATSU we assume that the user may only exercise one ticket at a time. This ticket could be one which the user has pre-purchased, or it could a ticket automatically selected for him. No combination of two tickets could be used at the same time.

In Oyster, the same period product is priced differently depending on whether it is presented as a 'Ticket' or a 'Cap'. We called this the application context. In our fare structure we will not make this distinction. There will only be one type of period product.

Note that there is no need to offer any ticket which has been enabled for automatic selection by the ATSU for pre-purchase. The ATSU ensures that any pre-purchase of an automatically selectable ticket will be redundant at best, and a bad economic decision in all other cases.

8.4.3.1 Daily Best Value

In our initial example we will explore the application of the ATSU to the daily best value problem. The premise of daily best value is that under no circumstance would a user be able to spend less money on fares by pre-purchasing a daily ticket than by simply using his contactless bankcard to travel without preparation. In other words, we remove from the user the need to decide which daily ticket, if any to buy out.

To implement daily best value in the ATSU, We will move away from the notion of 'capping' and introduce a new paradigm. The ATSU maintains a series of *Scenario Counters*. Each *scenario* represents a hypothetical 'alternate universe' in which the user is holding a particular ticket. Each scenario counter tracks the charges accumulated throughout the day under its corresponding ticket holding scenario. We will demonstrate this concept in an example below.

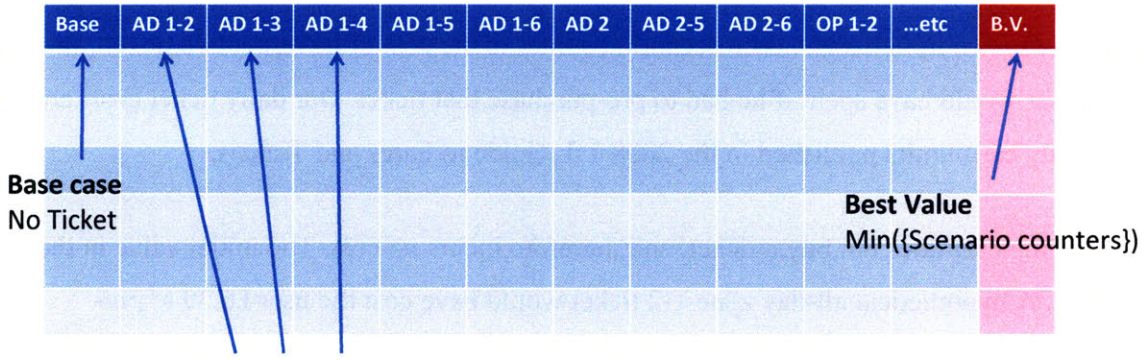


Figure 8.12 – Scenario Counters.

In the figure above, each column except the last represents a scenario counter. The following scenario counters are maintained:

- Base Case – This is the scenario where the user holds no ticket. Full single fares are charged for all journeys.

- AD n-n – These are a set of All-Day (AD) tickets. Our AD tickets are modeled after existing TfL all-day products as described in section 3.7.1. Once purchased an AD ticket allows the user to travel within its validity zones at any time at no incremental cost. Following the current Oyster offerings we provide AD tickets with the following zonal validity: 1-2, 1-3, 1-4, 1-5, 1-6, 2, 2-5, 2-6. For the sake of simplicity we will not demonstrate tickets with zonal validity greater than zone 6.

- OP n-n – These are a set of Off-Peak (OP) tickets. Our OP tickets are modeled after existing TfL off-peak products as described in section 3.7.1. Once purchased an OP ticket allows the user to travel within its validity zones during the off-peak time band at no incremental cost. The current Oyster off-peak time bands are described in section 3.5.1.4. We provide OP tickets with the same zonal validities as available for AD tickets.

The last column in the table labeled 'Best value' is not an actual scenario counter. Rather, it takes the minimum value chosen from all scenario counters. This value is given as min({scenario counters}). The *best value scenario* is given as argmin({scenario counters}).

8.4.3.2 Worked Example

Other than the base case, each hypothetical ticket is associated with an 'initial investment'. At the beginning of the day, we initialize the each scenario counter to the cost of its respective ticket. This is how much the user would have spent if he had to pre-purchase that ticket. Our daily ticket costs are based on the 'daily cap' limits published in the 2008 TfL Guide to Fares and Tickets.

In the base case, the user does not buy a ticket, and therefore incurs no cost. The initial value in the base case is zero. A hypothetical all-day zone 1-2 ticket would have cost the user £6.30 to pre-purchase. Therefore we initialize the scenario counter of the zone 1-2 ticket scenario to £6.30. We set the other scenario counters similarly. Finally, the best value column is computed as the minimum of

the scenario counters. In this case, argmin({scenario counters}), or the 'winning' scenario is the base case. The corresponding best value is therefore £0.00.

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	0.00

Winning Scenario

Figure 8.13 – Initializing Scenario Counters.

Now let us consider the user's first journey:

Journey 1: 7am Edgware-> Bank

Inner Zone – 1, Outer Zone – 5

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	0.00
Full1-5	Ext3-5	Ext4-5	Ext5	Free	Free	Ext1,3-5	Ext1	Ext1	Full1-5	Full1-5	

Figure 8.14 – Fare calculator queried for single fares. Journey 1: 7am Edgware-> Bank. Inner Zone – 1, Outer Zone – 5.

For each scenario, we use the fare calculator to generate the single fare for the current journey. The zonal validity of the scenario (if any) is supplied to the fare calculator. In this example, we assume that we have a simplified fare calculator which is based on the Maciejewski fare calculator. Recall that the Maciejewski fare calculator computes single fares based on fare zones. The fully featured fare calculator capable of arbitrary fares as proposed in section 8.4 could also be used; however we have chosen to use the simpler alternative for sake of more intuitive illustration.

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
+3.50	+1.80	+1.00	+1.00	+0.00	+0.00	+2.80	+1.50	+1.50	+3.50	+3.50	

Figure 8.15 – Fares returned by the fare calculator are added to the scenario counter.

In the base case, where the user holds no ticket, our journey incurs the full zone 1-5 fare. In the AD 1-2 scenario, because the user has free travel in zones 1-2, our journey incurs only a zone 3-5 extension fare. In the AD2 scenario, the user has free travel only within zone 2; therefore our journey incurs two extension fares for the zone 1 section, and the zone 3-5 section of the journey. Note that journey incurs the full zone 1-5 fare in all of the OP cases because an OP ticket does not give the user any free travel during the peak time band (into which our journey falls). Because we need to query the fare calculator for a fare for each scenario and for each journey, a high-performance fare calculator implementation is essential.

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
3.50	8.10	8.50	9.90	11.30	13.30	9.10	9.00	9.40	8.30	8.90	**3.50**

Figure 8.16 – Scenario counters after first journey.

Fares returned by the fare calculator are added to the scenario counters. The scenario counters now hold the user's total outlay under each scenario. For example, under the base case, the user would have spent £3.50. In the AD 1-2 case, the user, having had to pay for both the cost of the ticket and the extension fare for the first journey, would have spent a total of £8.10. The best value after the first journey is still the base case (no ticket).

Now, we consider a second journey.

Journey 2: 8am Bank -> St James's Park

Inner Zone – 1, Outer Zone – 1

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
3.50	8.10	8.50	9.90	11.30	13.30	9.10	9.00	9.40	8.30	8.90	**3.50**
Full 1	Free	Free	Free	Free	Free	Full 1	Full 1	Full 1	Full 1	Full 1	

Figure 8.17 - Fare calculator queried for single fares. Journey 2: 8am Bank -> St James's Park. Inner Zone – 1, Outer Zone – 1.

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
3.50	8.10	8.50	9.90	11.30	13.30	9.10	9.00	9.40	8.30	8.90	**3.50**
+1.50	+0.00	+0.00	+0.00	+0.00	+0.00	+1.50	+1.50	+1.50	+1.50	+1.50	

Figure 8.18 - Fares returned by the fare calculator are added to the scenario counter.

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
3.50	8.10	8.50	9.90	11.30	13.30	9.10	9.00	9.40	8.30	8.90	**3.50**
5.00	8.10	8.50	9.90	11.30	13.30	10.60	10.50	10.90	9.80	10.40	**5.00**

Figure 8.19 - Scenario counters after journey 2.

Again, the fare calculator is queried for the single fare for each scenario. The resulting fares are then added to the scenario counters. We repeat this process for three more zone 1 peak journeys. The

resulting state of the scenario counters is shown in the figure below. The base case still remains the winning scenario after journey 5. However will this still be the case when we add one more journey?

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
3.50	8.10	8.50	9.90	11.30	13.30	9.10	9.00	9.40	8.30	8.90	**3.50**
5.00	8.10	8.50	9.90	11.30	13.30	10.60	10.50	10.90	9.80	10.40	**5.00**
6.50	8.10	8.50	9.90	11.30	13.30	12.10	12.00	12.40	11.30	11.90	**6.50**
8.00	8.10	8.50	9.90	11.30	13.30	13.60	13.50	13.90	12.80	13.40	**8.00**
9.50	8.10	8.50	9.90	11.30	13.30	15.10	15.00	15.40	14.30	14.90	

Figure 8.20 – Scenario counters after three more zone-1 peak journeys. The base scenario still remains the winning scenario after journey 5. Now we add another zone 1 journey (the 5th such journey in a row).

Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	AD 2	AD 2-5	AD 2-6	OP 1-2	OP 1-3	B.V.
0.00	6.30	7.50	8.90	11.30	13.30	6.30	7.50	7.90	4.80	5.40	**0.00**
3.50	8.10	8.50	9.90	11.30	13.30	9.10	9.00	9.40	8.30	8.90	**3.50**
5.00	8.10	8.50	9.90	11.30	13.30	10.60	10.50	10.90	9.80	10.40	**5.00**
6.50	8.10	8.50	9.90	11.30	13.30	12.10	12.00	12.40	11.30	11.90	**6.50**
8.00	8.10	8.50	9.90	11.30	13.30	13.60	13.50	13.90	12.80	13.40	**8.00**
9.50	8.10	8.50	9.90	11.30	13.30	15.10	15.00	15.40	14.30	14.90	**8.10**

Figure 8.21 – The best-value scenario switches from the base case to AD 1-2.

After a fifth zone-1 peak journey, we can see that the base case is no longer the best value scenario. The winning scenario has now switched from the base case to the AD 1-2 scenario. In other words, had the user pre-purchased an AD 1-2 ticket, his total outlay at this point would be £8.10, including the initial cost of the ticket. In contrast, if he had not held a ticket and had to pay for every single fare his total outlay would now be £9.50. The ATSU has determined that an all-day zones 1-2 ticket would give the best value under these circumstances and automatically selected it for the user.

This example has been chosen to demonstrate that this ATSU algorithm is immune from zonal overextension. Under the traditional Oyster capping algorithm, the sequence of journeys shown above would have resulted in zonal over-extension. Specifically, the first journey, a zone 1-5 journey, would have extended the stretching window to 1-5. This would have activated the zone 1-5

cap for the remainder of the day, even though the subsequent barrage of zone 1 journeys mean that a zone 1 cap with a single zone 2-5 extension fare for the initial journey would have been more effective.

By working through the examples from section 3.7.5 using the ATSU algorithm presented in this chapter one can ascertain that this ATSU design resolves all of the identified drawbacks of Oyster capping.

8.4.3.3 Weekly Best Value

Our automatic ticket selection algorithm can be extended to support weekly best value with relative ease. Before we begin, we stipulate that weekly best value is supported on a fixed-window basis. Either the user or TfL appoints a day of the week that automatically selected weekly tickets would take effect from, for example, every Monday (beginning from 4:30 am when the system opens). Any weekly ticket selected by the system would take effect for the entire week from Monday to Sunday. Subsequent windows would adjoin the previous one (beginning the Monday after, and the Monday after that, etc.). The alternative to fixed-window is a rolling window, where weekly tickets may be applied beginning on any arbitrary day of the week as chosen by the algorithm. Rolling windows introduce additional complexity which will not be discussed in this thesis.

The most intuitive first step to support weekly best value is to add scenario counters corresponding the weekly ticket products. We will illustrate weekly best value in the figures below, which map out the same set of journeys as used in the example above in section 8.4.3.2.

Jrny	Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6	....	W 1-2	W 1-3	W 1-4	W 1-5...	B.V.
Init.	0.00	6.30	7.50	8.90	11.30	13.30	...	24.20	28.40	34.60	41.40	**0.00**
1	3.50	8.10	8.50	9.90	11.30	13.30	...	26.00	29.40	35.60	41.40	**3.50**
2	5.00	8.10	8.50	9.90	11.30	13.30	...	26.00	29.40	35.60	41.40	**5.00**
3	6.50	8.10	8.50	9.90	11.30	13.30	...	26.00	29.40	35.60	41.40	**6.50**
4	8.00	8.10	8.50	9.90	11.30	13.30	...	26.00	29.40	35.60	41.40	**8.00**
5	9.50	8.10	8.50	9.90	11.30	13.30	...	26.00	29.40	35.60	41.40	**8.10**

Figure 8.22 – Weekly best value. Weekly scenario counters added. Weekly ticket scenarios highlighted.

Weekly ticket scenarios are evaluated in the same fashion as their daily counterparts. Any time a journey is not fully covered by the zonal validity of the given weekly ticket, an extension fare is charged, adding to the total. For example, the first journey is not fully covered by the W1-2 ticket, incurring a £1.80 extension fare and therefore raising the W1-2 counter from £24.20 (the initial cost of an adult zones 1-2 weekly ticket) to £26.00.

However, adding weekly ticket scenarios alone is not sufficient to give us weekly best value capability. The problem is that the daily ticket scenarios require the best value counter to be reset at the beginning of each day. However, when we do so the cumulative cost of daily charges would be lost and the weekly scenarios would never be selected. In order for the weekly scenarios to be considered fairly, we must a maintain a total of charges accumulated from the use of daily tickets throughout the week.

To accomplish this, we two add counters. The first is the cumulatively daily best value (CDBV) counter. This counter is reset at the start of the weekly ticket bracket (we have chosen Monday for this example). Changes to the CBDV counter tracks changes to the daily best value (DBV) counter. As it can be seen below in Figure 8.23, on the first day the CBDV increases in synchronization with the DBV counter. When the daily best value scenario switches from the base case to AD 1-2, the daily best value ceases to increase; this is reflected in the CBDV.

At the end of Monday, the DBV counter is reset, as before. The CDBV counter, however, continues to accumulate. In essence, the CDBV counter tracks the user's cumulative spending for the week if only single fares and automatically selected daily tickets had been available. The CDBV counter accumulates each day's total in the same fashion for the remainder of the week.

The functioning of the weekly best value algorithm should now be clear. The user is charged the amount in the second of the counters we added, the weekly best value (WBV) counter. The WBV counter always carries the lesser of the CDBV counter and all of the weekly counters. For the first part of the week, the CDBV is the winning scenario. This means that the user has not 'broken even' with any weekly ticket yet and would be best served by some combination of no ticket and daily tickets. At some point later in the week, such as journey 28 in the example above, the CDBV finally loses its position as the winning scenario. In this case, it is overtaken by a W1-3 ticket. In other words, the user is now best served if she had purchased a W1-3 ticket at the beginning of the week.

We give her this benefit retroactively by letting the WBV assume the cost of the best value scenario, W1-3.

Jrny	Base	AD 1-2	AD 1-3	AD 1-4	AD 1-5	AD 1-6...	Daily B.V.	Cum. DBV	W 1-2	W 1-3	W 1-4	W 1-5...	Wk. B.V.
Init.	0.00	6.30	7.50	8.90	11.30	13.30	**0.00**	**0.00**	24.20	28.40	34.60	41.40	**0.00**
1	3.50	8.10	8.50	9.90	11.30	13.30	**3.50**	**3.50**	26.00	29.40	35.60	41.40	**3.50**
2	5.00	8.10	8.50	9.90	11.30	13.30	**5.00**	**5.00**	26.00	29.40	35.60	41.40	**5.00**
3	6.50	8.10	8.50	9.90	11.30	13.30	**6.50**	**6.50**	26.00	29.40	35.60	41.40	**6.50**
4	8.00	8.10	8.50	9.90	11.30	13.30	**8.00**	**8.00**	26.00	29.40	35.60	41.40	**8.00**
5	9.50	8.10	8.50	9.90	11.30	13.30	**8.10**	**8.10**	26.00	29.40	35.60	41.40	**8.10**
Init.	0.00	6.30	7.50	8.90	11.30	13.30	**0.00**	**8.10**	26.00	29.40	35.60	41.40	**8.10**
6	3.50	8.10	8.50	9.90	11.30	13.30	**3.50**	**11.60**	27.80	30.40	36.60	41.40	**11.60**
...	...	...	...	...	...	...	**...**	...	...	...	...	...	...
28	9.90	11.20	8.50	9.50	11.70	13.30	**8.50**	**36.30**	37.00	32.50	38.10	45.30	**32.50**

Daily B.V. resets every day. However Cumulative Daily B.V. carries over day-to-day

Weekly Best Value is Min of these columns

Figure 8.23 – Weekly best value. Weekly best value and cumulative daily best value counters introduced.

Automatic ticket selection can be extended to monthly and annual tickets extending this algorithm, by introducing monthly and annual best value counters. The limitation of a fixed window still stands.

8.4.3.4 Best Value with Pre-purchased Ticket

Offering tickets which are automatically selectable for manual purchase by users may appear to be completely redundant, since making a conscious purchase would always result in a zero or negative benefit for the user. However, TfL may opt to exclude some of its ticket products from automatic selection. Furthermore, if the fixed ticket windows are prescribed by TfL (and not selectable by the user), the user could benefit by purchasing a weekly, monthly or annual ticket which begins on a different date than the prescribed default (e.g. a monthly ticket beginning on the 8th rather than the 1st).

A user pre-purchased a ticket can be represented in the ATSU as a scenario counter with a £0.00 initial investment, accounting for the fact that the pre-purchased ticket is a 'sunk cost'. This counter is evaluated in accordance with the terms of the purchased ticket, including its zonal validity and reset behavior. The ultimate best value scenario is then represented by the lesser of the pre-purchase counter and the winning scenario of the existing automatic selection scenarios.

Note that the limitation that a user can hold only one ticket product at a time still stands. In other words, the pre-purchased product cannot be applied at the same time (in conjunction with) any automatically selected product, just as two products cannot be automatically selected together. A multi-ticket algorithm which supports more than one ticket simultaneously warrants further development.

Conclusions

We will restate here the thesis question that was posed in section 2.2.

> *Is it feasible to develop a fare engine capable of accepting and processing contactless bankcards as a fare instrument on the TfL network while satisfying current and future fare structure and performance requirements?*

8.5 *Summary of Research Process*

The process of answering this question was decomposed into successive steps represented by the seven chapters of this thesis.

Step 1 – Background research

First, we framed this discussion on contactless bankcard fare payment by examining the drawbacks of current fare payment systems in use by transit agencies around the world. We discussed that high operating costs were a prevailing challenge, and that direct fare payment using contactless bankcard is an attractive alternative owing to the ability to export much of the operational expense associated with a proprietary fare payment system to third party financial entities.

Step 2 – Thesis question

We defined the context of the question by laying out what fare engine is and where it fits in a fare collection system. We defined a fare engine to be the business logic which converts the passenger's use of a fare instrument into the charging of a fare. The thesis question was stated and decomposed into a three-part problem:

1. What are the current and future fare requirements for TfL?
2. What are the capabilities and limitations using contactless bankcards as a fare instrument?
3. Is it feasible to design a fare engine which satisfies the requirements of 1) and successfully solves the challenges posed by 2)?

Step 3 – Analysis of current TfL fare structure

We answered the first part of the thesis question by identifying and formalizing the existing TfL fare structure. This was accomplished by analyzing the de-facto fare documentation available to us, in the

form of published user pamphlets and staff guides. We found that TfL fares can be segmented into two product spheres – single products and period products. Each of these spheres can in turn be represented by a fare matrix which can be parameterized into multiple independent tiers. The only exception to this is that the geographic sensitivity of single fares can neither be explained by a fare table of origin and destination stations, nor by a fare table of origin and destination zones. A system of charge codes can be used to capture the intricate system of exceptions. We also documented the operation of the current capping algorithm in Oyster, and its inability to produce true best value under various input conditions. The operation of out-of-station interchange (OSI), automatic journey completion and aliasing were described.

Step 4 – System Requirements

Based on our analysis of TfL's current fare structure, we laid out the system requirements for a fare engine that meets current and future fare needs. In this process, we defined two sets of requirements, one set for an ambitious and scalable general design, and another set for a more limited proof-of-concept prototype. We also identified the three key journey linking requirements, which are OSI, bus-rail interchange, and intermediate validation. The former two are existing Oyster services, while the latter is not currently available in Oyster.

Step 5 – Fare Engine Design

The bulk of this thesis is devoted to the design of the fare engine. In chapter 4, we outlined the future ticketing fare collection system being developed by TfL. We positioned the fare engine as the subsystem that provides the linkage between field devices and the billing engine. In chapter 5, we established the internal structure of the fare engine. The fare engine was divided into two modules. The journey processor is responsible for converting raw bankcard interactions (taps) into linked journeys, while the fare processor is responsible for converting linked journeys into billing items which can be charged to the user's account, where they can be aggregated in any way the billing engine sees fit. We also defined the data flows that connect the fare engine to other systems within the fare collection system, and the data flow that connects the two fare engine modules.

8.6 *Identified Challenges and Solutions*

During the design process of the contactless bankcard fare engine, we identified some major challenges and were successful in proposing solutions to them. The more significant challenges were:

Tap sequencing without guaranteed in-order arrival

The processing of taps in Oyster is guaranteed to be sequential due to the decentralized nature of the system. However, contactless bankcard fare payment is a centralized system which relies on the transmission of taps from stations and vehicles to a central processing system. Communication delays could cause taps to arrive at the server in a different order than that in which the original transactions were made.

The consequence of this is that the fare engine can no longer be a stateless system (as in Oyster). Our solution is an object oriented data structure which resides in memory and keeps a full state of the user's journey history. An individual structure is maintained for each user to ease user management and multi-tasking. As taps arrive they can be inserted into the correct position based on their timestamp. A cleanup mechanism prevents the data structure from growing without bound as journey history accumulates.

Dynamic journey linking

A consequence of out-of-sequence tap arrival is that adjacency cannot be guaranteed. In other words, two taps formerly thought to be adjacent (follow one after the other) may turn out not to be so after the arrival of a delayed intervening tap. This means arriving taps could alter the interpretation of existing taps.

Our solution is a dynamic journey linking mechanism. Journeys are represented in a three level structure. The levels are Tap, Journey Segment and Linked Journey. As taps are inserted at the lowest level, their effect on existing journey segments and linked journeys is bubbled up through the structure using a set of triggering business rules. Linked journeys may have been created, deconstructed or reformed as a consequence. The mechanism is guaranteed to stabilize with near-constant time complexity, providing predicable and scalable performance. Inserted taps are reflected immediately in the structure, providing real-time feedback for customer service and payment authorization needs.

Arbitrary OD fares on National Rail

Unlike TfL fares, which are tightly coupled to the zonal structure, National Rail (NR) single fares are set on an arbitrary basis between origin and destination stations. At the same time, it is a requirement that zonal period tickets be usable for NR journeys. It is a challenge to integrate

these two conflicting requirements in a contactless bankcard fare engine so that tickets can be applied where eligible and extension fares are calculated correctly where needed.

Our solution proposes a fare calculator module that divides the fare for each journey into zonal reducible and non-reducible components. The former is affected by period tickets, while the latter is not. This solution cleanly unifies period tickets with NR journeys, providing seamless, automatic NR support from the contactless bankcard user's perspective. Moreover the existing TfL investment in the Matrix Control System (MCS) for Oyster Extension on National Rail (OXNR) can be leveraged, giving ready support for revenue allocation.

True Best Value

Oyster currently implements best value through a capping algorithm that suffers from a number of inefficiencies primarily concerned with its use of running totals and stretching windows to determine which cap to apply. Furthermore, extending the capping algorithm to support weekly best value is difficult and unwieldy.

Our solution is to abandon the notion of capping and instead introduce the concept of automatic ticket selection. By maintaining a battery of scenario counters that correspond to the possible ticket holdings and keeping these scenarios updated with user actions, we are able to select the counter with the lowest cumulative fare, to ensure true best value at all times. With minimum modification, this solution can be adapted for weekly best value.

As we have seen, taps can be sequenced without a guarantee of in-order arrival. Journey linking can be performed dynamically, in real time on these unsequenced taps. Arbitrary National Rail fares can be integrated into the existing system of zonal tickets, and finally, best value can be accomplished in a future ticketing system without the restrictions of today's Oyster capping.

Ultimately, we conclude that a fare engine which works with contactless bankcard payments and implements TfL's current and future fare needs appears feasible. This conclusion is based on our ability to find solutions to the key obstacles arising from both the limitations of using contactless bankcards for direct fare payment and TfL's complex fare requirements.

8.7 *Further Work*

In this thesis we have demonstrated evidence for the feasibility of a contactless bankcard fare engine. It has been shown from a systems engineering perspective that the difficulties posed by the properties of contactless bankcards and TfL's complex fare structure are not insurmountable. A fare engine design has been proposed. However the practicability of this design can be confirmed by enacting it in a proof-of-concept and subjecting the prototype to inputs that simulate the stresses of real-world usage.

Therefore, the next logical step in the development process of a contactless bankcard fare engine is to prove that such a fare engine is not only feasible, but practical with respect to hardware and software requirements by implementing the prototype implementation and making quantitative benchmarks of performance under stress testing.

In addition, we have so far glossed over the details of the fare calculator module. Integration with the Matrix Control System was proposed and a general mapping of services was provided. However the specific aspects of this integration, as well as performance and data considerations need further investigation.

References

[1] Transport for London, *Your guide to fare and tickets within zones 1-6*. London:TfL, 2009.

[2] Transport for London, *Your guide to fares and tickets Zones 7-9*. London: TfL, 2009.

[3] Transport for London, "Tickets", London: TfL, 2009. [online]. Available: http://www.tfl.gov.uk/tickets/ [accessed: May 2, 2009]

[4] J. Maciejewski, "Automating Journey Fare Calculation for Transport for London", MST Thesis, Massachusetts Institute of Technology. Cambridge, MA, 2008.

[5] S. Mehta, "Analysis of Future Ticketing Scenarios for Transport for London", MST Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2006.

[6] M. Dorfman, "Future Contactless Payment Options for Transport for London: Demand, Cost, Equity, and Fare Policy Implications", MST Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[7] Smart Card Alliance. "Transit and Contactless Financial Payments: New Opportunities for Collaboration and Convergence," Smart Card Alliance, Princeton Junction, NJ, 2006.

[8] Smart Card Alliance. "Transit and Retail Payment: Opportunities for Collaboration and Convergence," Smart Card Alliance, Princeton Junction, NJ, 2003.

[9] Smart Card Alliance. "Co-Branded Multi-Application Contactless Cards for Transit and Financial Payment," Smart Card Alliance, Princeton Junction, NJ, 2008.

[10] A. Kjos, "The Merchant-Acquiring Side of the Payment Card Industry: Structure, Operations, and Challenges," Federal Reserve Bank of Philadelphia, Philadelphia, PA, 2007.

[11] D. Fleishman, N. Shaw, A. Joshi, R. Freeze, "TCRP Report 10: Fare Policies, Structures and Technologies," TRB, Washington, DC, 1996.

[12] MultiSystems Inc, "TCRP Report 94: Fare Policies, Structures and Technologies: Update," TRB, Washington, DC, 2003.

[13] D. Gauthier, "Watch out, EMV is coming in contactless too," ContactlessNews, Dec 6, 2005. [online]. Available: http://www.contactlessnews.com/2005/12/06/watch-out-emv-is-coming-in-contactless-too [accessed: May 20, 2009].

[14] Visa Europe, "Visa Contactless," Visa Europe, London, UK, 2009. [online]. Available: http://www.visaeurope.com/pressandmedia/factsheets/visacontactless.jsp [accessed: May 20, 2009].

[15] HSBC plc, "MasterCard PayPass," HSBC plc, London, UK, 2009. [online]. Available: http://www.hsbc.co.uk/1/2/personal/credit-cards/paypass [accessed: May 20, 2009].

[16] Smart Card Alliance, "Contactless Payment: Try It, You'll Like It," Smart Card Alliance, Princeton Junction, NJ, 2008. [online]. Available: http://www.smartcardalliance.org/articles/2008/09/17/contactless-payment-try-it-youll-like-it [accessed May 20, 2009].

[17] B. Blanchard & W. Fabrycky, *Systems Engineering and Analysis*. Upper Saddle River: Pearson Upper, 2006.

[18] E. Gamma , R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[19] J. Shirazi, *Java Performance Tuning 2ed.* O"Reilly, 2003.

Appendices

A. Current Oyster Fare Structure Hierarchy

This hierarchy was compiled by noting the heading organization and formatting.

- **Adult**
  - **Single Fare**
    - Bus and Tram
      - Oyster single
      - Cash single
    - Tube DLR and Overground
      - Oyster
        - Peak
        - Off-Peak
      - Cash single
  - **Day Tickets and Oyster Daily Price Capping**
    - Bus and Tram
      - Daily Price Cap
      - One day bus and tram pass
    - Tube DLR and Overground
      - Daily price cap
        - Peak
        - Off-Peak
      - Day Travelcard
        - Peak
        - Off-Peak
  - **Season Tickets**
    - Bus and Tram Passes
      - 7 Day
      - Monthly
      - Annual
    - Travelcards
      - 7 Day
      - Monthly
      - Annual
- **Discount**

- **Single Fare**
  - Bus and Tram
    - 16+ and New Deal
      - Oyster Single
      - Cash Single
  - Tube DLR and Overground
    - Child
      - Oyster single
      - Cash single
    - 16+ and New Deal
      - Oyster single
        - Peak
        - Off-peak
- **Daily Price Caps and One Day Tickets**
  - Bus and Tram
    - 16+ and New Deal
      - Daily Price Cap
  - Tube DLR and Overground
    - Child
      - Daily Price Cap
        - Peak
        - Off-peak
      - Day Travelcard
        - Anytime
        - Off-peak
    - 16+
      - Daily Price Cap
        - Peak
        - Off-peak
    - New Deal
      - Daily Price Cap
        - Peak
        - Off-peak
- **Season Tickets**
  - Bus and Tram
    - Student
      - 7-Day
      - Monthly

        - Annual
    - 16+ and New Deal
        - 7-Day
        - Monthly
- Travelcards
    - Child, 16+ and New Deal
        - 7 Day
        - Monthly
    - 18+ Student
        - 7 Day
        - Monthly
        - Annual

- **Visitor and Group Tickets**
    - Adult
    - Under -18
- **DLR and River Rover Tickets**
    - Adult
    - Child
    - Family
- **3 Day Travelcards**
    - Adult
        - Anytime
        - Off-peak
    - Child/New Deal
        - Anytime
        - Off-peak

B. Matrix Control System Sample Output

The following excerpt illustrates the output of TfL's Matrix Control System. Disclaimer: Information below is provided for illustrative purposes only and is not to be interpreted as a representation of actual TfL policy or processes.

Matrix Engine Results run date **11/27/2008**
Matrix Version **1**

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 1	719	686	07475598	00000000	00000000	00000000	62.72	37.28	0.10	76.02

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	76.02	2489	1561	928	100.00

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	719	747	TfL	928	37.02	Stratford	Waterloo
Inter	747	5598		0	10.2	Waterloo	London Waterloo
Leg	5598	5570	NR	1561	28.8	London Waterloo	Richmond NR

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 2	719	686	00000000	00000000	00000000	00000000	62.20	37.80	0.10	87.43

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	76.20	2794	2794	0	50.28

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	5570	NR	2794	76.2	Stratford NR	Richmond NR

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
2	79.37	2504	0	2504	36.62

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	719	686	TfL	2504	79.37	Stratford	Richmond

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
3	93.94	2793	2408	385	8.53

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	5588	NR	2408	67.75	Stratford NR	Gunnersbury NR
Inter	5588	591		0	6.3	Gunnersbury NR	Gunnersbury
Leg	591	686	TfL	385	19.89	Gunnersbury	Richmond

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
4	100.20	2794	2794	0	4.56

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	6009	NR	724	28.25	Stratford NR	Highbury&Islington NR
Inter	6009	6009		0	12	Highbury&Islington NR	Highbury&Islington NR
Leg	6009	5570	NR	2070	59.95	Highbury&Islington NR	Richmond NR

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 3	719	686	69650513	00000000	00000000	00000000	27.79	72.21	0.10	96.80

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	85.17	2463	650	1813	91.09

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	6965	NR	650	16.85	Stratford NR	London Livrpl St
Inter	6965	513		0	15.4	London Livrpl St	Bank
Leg	513	686	TfL	1813	52.92	Bank	Richmond

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
2	108.42	2464	1036	1428	8.91

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	6965	NR	650	16.85	Stratford NR	London Livrpl St
Inter	6965	513		0	15.4	London Livrpl St	Bank
Leg	513	591	TfL	1428	47.22	Bank	Gunnersbury
Inter	591	5588		0	6.3	Gunnersbury	Gunnersbury NR
Leg	5588	5570	NR	386	22.65	Gunnersbury NR	Richmond NR

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 4	719	686	05425598	00000000	00000000	00000000	62.80	37.20	0.10	101.10

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	86.70	2525	1561	964	94.68

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	719	542	TfL	964	41.2	Stratford	Embankment
Inter	542	5598		0	16.7	Embankment	London Waterloo
Leg	5598	5570	NR	1561	28.8	London Waterloo	Richmond NR

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
2	115.50	2850	2285	565	5.32

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	6009	NR	724	28.25	Stratford NR	Highbury&Islington NR
Inter	6009	603		0	7.6	Highbury&Islington NR	Highbury
Leg	603	542	TfL	565	34.15	Highbury	Embankment
Inter	542	5598		0	16.7	Embankment	London Waterloo
Leg	5598	5570	NR	1561	28.8	London Waterloo	Richmond NR

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 5	719	686	69650513	07475598	00000000	00000000	90.32	9.68	0.10	88.95

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	88.95	2448	2211	237	100.00

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	6965	NR	650	16.85	Stratford NR	London Livrpl St
Inter	6965	513		0	15.4	London Livrpl St	Bank
Leg	513	747	TfL	237	17.7	Bank	Waterloo
Inter	747	5598		0	10.2	Waterloo	London Waterloo
Leg	5598	5570	NR	1561	28.8	London Waterloo	Richmond NR

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 6	719	686	69650513	05425598	00000000	00000000	89.01	10.99	0.10	90.25

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	90.25	2484	2211	273	100.00

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	6969	6965	NR	650	16.85	Stratford NR	London Livrpl St

Inter	6965	513		0	15.4	London Livrpl St	Bank
Leg	513	542	TfL	273	12.5	Bank	Embankment
Inter	542	5598		0	16.7	Embankment	London Waterloo
Leg	5598	5570	NR	1561	28.8	London Waterloo	Richmond NR

Stratford - Richmond	origin	dest	IV1	IV2	IV3	IV4	NR%	TfL%	Lambda	Av GJT
Validation Signature 7	719	686	07475598	55955595	00000000	00000000	62.72	37.28	0.10	92.32

Route	Total GJT	Total distance	NR distance	TfL distance	Flow %
1	92.32	2489	1561	928	100.00

Type	Start nlc	End nlc	Mode	Distance	GJT	Start name	End name
Leg	719	747	TfL	928	37.02	Stratford	Waterloo
Inter	747	5598		0	10.2	Waterloo	London Waterloo
Leg	5598	5595	NR	624	11.9	London Waterloo	Clapham Junction
Inter	5595	5595		0	13	Clapham Junction	Clapham Junction
Leg	5595	5570	NR	937	20.2	Clapham Junction	Richmond NR