[1 Introduction 1](#_Toc502663579)

[2 Architecture 1](#_Toc502663580)

[2.1 Tiers 1](#_Toc502663581)

[2.2 Transactions, Status and History 1](#_Toc502663582)

[2.3 Interactive reports 2](#_Toc502663583)

[3 Product management 2](#_Toc502663584)

[3.1 Delivery method validation 2](#_Toc502663585)

[3.2 Delivery proposal 3](#_Toc502663586)

[3.3 Delivery validation 3](#_Toc502663587)

[3.4 Delivery posting 3](#_Toc502663588)

[3.5 Generate generic delivery instructions 4](#_Toc502663589)

[3.6 Generate delivery method specific documents 4](#_Toc502663590)

[3.7 Return processing 4](#_Toc502663591)

[4 Subscription management 5](#_Toc502663592)

[4.1 Subscription capture 5](#_Toc502663593)

[4.2 Package capture 6](#_Toc502663594)

[4.2.1 Cost 8](#_Toc502663595)

[4.2.2 Implementation issues 8](#_Toc502663596)

[4.3 Subscription Picker 8](#_Toc502663597)

[4.4 Renewal processing 8](#_Toc502663598)

[4.5 Cancellations and Credit notes 9](#_Toc502663599)

[4.5.1 No deliveries occurred yet 9](#_Toc502663600)

[4.5.2 Some deliveries have already occurred 9](#_Toc502663601)

[4.5.3 All deliveries have occurred 9](#_Toc502663602)

[4.6 Dormant subscriptions 10](#_Toc502663603)

[4.7 Skip logic 10](#_Toc502663604)

[5 Customer management 11](#_Toc502663605)

[5.1 Duplication detection 11](#_Toc502663606)

[5.2 Liability versus contractual obligations ( due ) 11](#_Toc502663607)

[5.3 Customer classification/profiling 12](#_Toc502663608)

[5.3.1 Classification by speciality 12](#_Toc502663609)

[5.3.2 Classification by medical aid and language 13](#_Toc502663610)

[5.3.3 Classification by location 13](#_Toc502663611)

[5.3.4 Data Warehouse 13](#_Toc502663612)

[6 Customer - Payment processing 13](#_Toc502663613)

[6.1 Current payment processing 13](#_Toc502663614)

[6.2 Overview 13](#_Toc502663615)

[6.2.1 MIMS specific 13](#_Toc502663616)

[6.2.2 Head Office 14](#_Toc502663617)

[6.2.3 Cashflow statement 14](#_Toc502663618)

[6.2.4 Algorithm 14](#_Toc502663619)

[6.3 DebitOrders 15](#_Toc502663620)

[7 Customer - Debit order processing 15](#_Toc502663621)

[7.1 Capture the static information 15](#_Toc502663622)

[7.2 Calculation of due values on a monthly basis 16](#_Toc502663623)

[7.2.1 History 16](#_Toc502663624)

[7.2.2 MDR 16](#_Toc502663625)

[7.2.3 MobiMims 16](#_Toc502663626)

[7.2.4 Continuous debit orders 17](#_Toc502663627)

[7.2.5 Equalised pro-actively 17](#_Toc502663628)

[7.2.6 Alternate approaches 18](#_Toc502663629)

[7.3 Submission to bank 18](#_Toc502663630)

[7.4 Import 18](#_Toc502663631)

[7.5 Bouncing of DebitOrders 19](#_Toc502663632)

[8 Operational reports reports 19](#_Toc502663633)

[8.1 XML 19](#_Toc502663634)

[8.2 Datasets 20](#_Toc502663635)

[8.3 Stored procedures 20](#_Toc502663636)

[8.4 MS Query 20](#_Toc502663637)

[8.5 Invoices and payments statement 20](#_Toc502663638)

[8.5.1 Filtering 21](#_Toc502663639)

[8.5.2 Past method. 21](#_Toc502663640)

[9 Business Intelligence Reports 21](#_Toc502663641)

[10 Diagnostic reports 21](#_Toc502663642)

[10.1 Stored procedures 22](#_Toc502663643)

[10.1.1 Rectification procedures 22](#_Toc502663644)

[10.2 SQL scripts 23](#_Toc502663645)

[11 Peculiarities 23](#_Toc502663646)

[11.1 GetUniqueNumber 23](#_Toc502663647)

[11.2 CustomerUpdate 23](#_Toc502663648)

[11.3 Mapping to Medical Aids 24](#_Toc502663649)

[11.4 ReportViewer 24](#_Toc502663650)

[11.5 MLedgerData003 25](#_Toc502663651)

[12 Batch and remote processing 26](#_Toc502663652)

[12.1 WCF service 26](#_Toc502663653)

[12.2 Connection to database 27](#_Toc502663654)

[12.3 NT service host 27](#_Toc502663655)

[12.4 NT ServiceHost Setup 27](#_Toc502663656)

[13 State machine in DeliveryAddress2 27](#_Toc502663657)

[14 Financial reporting 28](#_Toc502663658)

[14.1 Transaction table 29](#_Toc502663659)

[15 Adaptation points 29](#_Toc502663660)

[15.1 Settings 29](#_Toc502663661)

[15.2 Presentation tier 29](#_Toc502663662)

[15.3 Business tier 29](#_Toc502663663)

[15.4 Data tier 30](#_Toc502663664)

[15.5 AdministrationData as static member 30](#_Toc502663665)

[16 Suspended functionality - MAPIT 30](#_Toc502663666)

[17 Suspended functionality - Archiving 31](#_Toc502663667)

[17.1 Temporary bypass 31](#_Toc502663668)

[17.2 Transactions 32](#_Toc502663669)

[17.3 Subscriptions 32](#_Toc502663670)

[17.4 Customers 32](#_Toc502663671)

[17.5 Data warehouse 32](#_Toc502663672)

[18 Roadmap 32](#_Toc502663673)

[19 To do 33](#_Toc502663674)

# Introduction

This document is something between a requirement specification and an implementation specification. It could be seen as a bridge between the two.

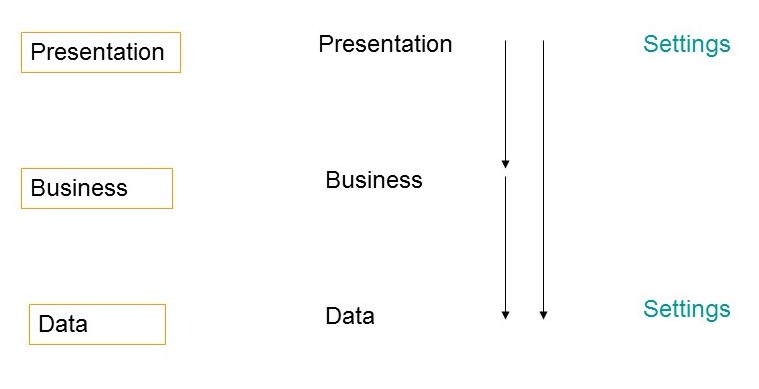
The system originated out of a failure of the Paradise system to provide the functionality required for MIMS.

An advantage of being a somewhat independent self-contained system is that it makes it possible to sell the MIMS system and the MIMS business as a unit.

This is a pure MS Dotnet application. It does not rely on third party or open source components. The idea is to take advantage of the economies of expertise achievable by reusing Dotnet components. Due to the resolution of the building blocks in Dotnet, the system is very adaptable – if you know DotNet.

# Architecture

## Tiers



This is a 3 tier system. As can be seen, the Presentation tier can see the other two tiers, but not the other way round.

A pointer to the database is set in the settings of the Presentation Tier, and is visible to all tiers.

The settings at the data tier pertains only to the design time environment. While not designing, it points to MIMSx (sic) deliberately, in order to prevent leakage to the run-time environment. There is no database called MIMSx.

All of the Business Tier is implemented as static classes. State is kept in the Data tier objects.

The idea is to make all of the data tier objects self-validating, so as to simplify it being used by different Presentation technologies. This has not been done in a consistent and exhaustive way yet.

## Transactions, Status and History

The status of a subscription is kept in the subscription record and in the SubscriptionIssue records.

The status of a customer is kept in a Customer record.

Any change in status is recorded in the Transactions table.

It is possible to reconstruct the status by accumulating the transactions. See Diagnostic Reports.

Since the **transaction table** retains the history of every transaction, it can also be used in to generate business documents and to populate a data warehouse.

In order not to load the transaction record with too much redundant data, it *refers to* the static data in the subscription record. It thus assumes that certain fields there will never change and that the subscription record will never be deleted.

Since this table can grow very fast, it should be possible to archive parts of it.

## Interactive reports

? A passive report requires of the operator to make a decision and then to manually navigate to that part of the system from where the decision can be executed.

In this system, some reports are rendered in a DataGrid, rather than in Reporting Services. This makes it possible to provide immediate action points via context menus. The option for formatted reports are still available to support correspondence with customers.

In addition, many reports can be rendered as XML with corresponding XSDs, thus enabling end user computing via Excel.

# Product management

## Delivery method validation

When a subscription is captured, the delivery method is fixed there as well.

DeliveryMethod processing is implemented as a set of static classes, each inheriting the same signature from Subs.Data.DeliveryBase. The static classes are declared and defined in Subs.Data.DeliveryMethodsStatic.

When you validate a subscription, you select the relevant deliverymethod object and use it to do the job. Because each delivery method is applicable only to certain situations, the choice of delivery method has to be validated. Each deliveryMethod also has its own pricing structure.

## Delivery proposal

Based on current subscriptions and on the issue involved, the system can generate a proposal of what should ideally be delivered.

## Delivery validation

Before the delivery of an issue can be authorised, the system has to ensure that:

* Stock is available
* A delivery address is available
* Money has been allocated to the delivery, or the delivery has been marked as a credit issue.

The validation results are presented to the operator, so that corrective action can be taken.

One corrective action is to **skip** the delivery of a specific issue, and add another issue at the end of the subscription.

## Delivery posting

Once all the problems have been resolved, or removed, the survivors can be posted into the system. At this point delivery is recorded on each receiver, and the stock count is updated.

In order to be able to rerun the posting of the ALL XML, we put the whole validated proposal in XML. It is DeliveryDoc.DeliveryRecord. This is the source, both for the posting, and for the generation of the delivery method specific XML. IN the latter case, the XML is actually kinds of subsets of the original.

In the posting of the delivery, every delivery is treated as a transaction.

You should not be able to generate delivery XML unless all the posting went through cleanly. You did that by disabling the Generate XML unless you had a clean post run. That is still a good idea.

Even so, the job as a whole can fail. In that case, you will have to check the error log to see how far the posting got.

You will need a way of restarting the batch posting after the problem has been resolved. So, you are going to need an XML file to fall back on the redo a posting. Alternatively, the system has to be clever enough not to post the same entry more than once.

In the past, you kept only an in core version of the Delivery Record table. This prevented a user to create the delivery method specific XML later on, on a Post that might not have run or might not have run to completion.

What I do now is to make a copy of the Delivery Records just before posting it, so if something goes wrong, the batch can be rerun. The way it happens is that I load it from the saved file and from there the process carries on as per normal.

## Generate generic delivery instructions

Once the posting process has succeeded, one can create delivery instructions. One XML file is created for each delivery method.

## Generate delivery method specific documents

Each set of delivery instructions can now be processed and reprocessed in a variety of ways, in order to create a variety of documents.

## Return processing



# Subscription management

## Subscription capture

Generally speaking, the dataset resides in the data object. To connect dataset to reports and controls, we use a copy of the dataset inside the data objects. That is not the ideal.

On the subscription screen, we differentiate between the dataset that carries the result of a selection, called subscriptionDoc1, and subscriptionPayer, which carries the cluster of a payer and all related subscriptions.

## Package capture

A **package** is a situation where more than one product are sold *as* a unit.

A promotion is not necessarily a package.

A package is not necessarily a promotion?

A promotion should not be renewed, but a package can be renewed?

? How much and how long should the system remember and keep track of the association?

* Start together - they start together, but thereafter each subscription is on its own.

We have to create separate subscriptions, because they might differ in delivery frequency, and delivery method, and perhaps in other ways as well.

What happens if the person pays for only a part of the package?

* Intermediate

Some transactions have to apply to all the subscriptions in the package and some must be specific to some subscriptions in the package.

* End together

What happens if something happens to the common receiver, or payer?

Renewal notices, renewal prevention and actual renewals together?

Loyalty detection and loyalty reward?

A person might have received a package, but wants to continue with only a subset of the package?

Archive together?

‘-‘ = Not applicable

‘o’ = Obsolete feature

‘n’ = Future feature

‘\*’ = Applicable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction | Apply to payer | Apply to receiver | Apply to package | Apply to subscription |
| Pay = 1, | \* |  | - | - |
| Deliver = 2, | \* | \* | - | \* |
| Extend = 3, |  |  | o | o |
| Refund = 4, |  |  | - | - |
| Hold = 5, |  | \* |  |  |
| Suspend = 6, |  |  | \* | \* |
| Resume = 7, |  |  | \* | \* |
| UpdateCreditLimit = 8, | \* |  |  |  |
| CancelSubscription = 9, |  |  | \* | \* |
| Expire = 10, |  |  | \* | \* |
| Skip = 11, |  |  |  | \* |
| Return = 12, |  |  |  | \* |
| InitialiseCustomer = 13, |  |  |  |  |
| Credit = 14, | \* |  |  |  |
| CancelCustomer = 15, | \* | \* |  |  |
| InitialiseSubscription = 16, |  |  |  | \* |
| DeliverableUpTo = 17, Virtual |  |  |  |  |
| Balance = 18, Conversion from previous system. | \* |  |  |  |
| VATInvoice = 19, |  |  |  |  |
| UpdateCustomer = 20, | \* | \* |  |  |
| ChangeUnitsPerIssue = 21, |  |  |  | o |
| CreditNote = 22, |  |  |  |  |
| WriteOffMoney = 23, | \* |  |  |  |
| ReversePayment = 24, | \* |  |  |  |
| ReverseWriteOffMoney = 25, | \* |  |  |  |
| ArchiveBalance = 26, | \* |  |  |  |
| ReverseDelivery = 27 |  |  |  | \* |
|  |  |  |  |  |
|  |  |  |  |  |

The normal idea behind a package is to introduce customers to new products, in the expectation that they would afterwards subscribe to it. Therefore, it makes sense to us to attempt to renew all the products components of a package – independently. We also propose to provide a manual facility to prevent renewal attempts on any product should the customer so request.

The question now is whether there is any merit in wanting to renew the whole lot AS A PACKAGE. The only justification we can see in it is the advertisement fraud, i.e. attempting to up the number of circulation in an artificial way as is done in the MIMS case where the three products overlap in terms of content. We see this as an exceptional case, and not as the normal flow of things. We would not want to go to a lot of effort to hard code this practice, particularly if it is just a transient situation that is not likely to be used much longer.

As far as we can see, there is no justification to keep track of the grouping of subscriptions, once they have been captured ---- except preventing the abuse described next:

. What happens in the case where a person gets a package at a reduced price. He then pays for only a part of the price, receives the first product, and then refuses to pay any further. In this way, he takes advantage of us. How can we prevent this.

* Check to see if the invoice has been paid.

-

### Cost

It is assumed that the total cost for a package will be less than the sum of the individual costs. Even so, that reduced cost has to be distributed between the participating products. One way to do this is to do it proportionately to their corresponding base costs.

Differentiate between a **potential** package, where that potential is described ( blue data ) versus **actual** subscriptions of that kind.

At the moment, we create a separate promotion, one for each product. So, they are the same only in name, but they apply to different products. But the customer did not stick to this convention, so the situation becomes messy.

### Implementation issues

Some transactions have to apply to all the subscriptions in the package in the same way, and some must be specific to some subscriptions in the package.

The package capture screen invokes the subscription iteratively, after having initialised it with what is common data in the package.

## Subscription Picker

The Subscription Picker can select subscriptions on many criteria, and also facilitates direct execution of actions, based on the results of the selection.

The system caters for different kinds of result sets, so as to provide information relevant to potential actions. This is implemented by the use of two different user controls called SubscriptionDormantControl and SubscriptionDetailControl.

## Renewal notice processing

There is a facility by which one can generate renewal notices based on the issue on which a subscription expires.

Subscriptions can be marked such that renewal processing is blocked.

A subscription will not be selected if a successor already exists.

Whenever a subscription is renewed, it status is moved from expired to cancelled. What happens if it has not expired yet.

When a new subscription is created, it would be nice if the system swing corresponding expired subscriptions to cancelled state. As far as I know, that is not currently the case.

The converse of blocking renewal is to have continuous subscriptions with associated debit-orders. This is not implemented as yet.

## Renewal processing

If would be useful to be able to create a new subscription, using the old subscription as a template. This is not in place yet.

## Quotation

One can generate a quotation, and then one can convert a quotation into a real subscription.

## Cancellations and Credit notes

A credit note is a mechanism to modify an invoice.

In all cases, a reason has to be supplied.

From there onwards, we have three scenarios:

### No deliveries occurred yet

One can cancel the subscription, but you need to supply a credit note as well, whether the subscription has been paid or not.

### Some deliveries have already occurred

All changes here will apply only to the remainder of the deliveries, even if everything remains.

We cater for reductions in:

* UnitPerIssue
* NumberOfIssues

### All deliveries have occurred

This option allows one to cancel a subscription in order to prevent delivery notes to be generated in future. This option does not warrant a credit note.

## Dormant subscriptions

If a user does not pay, nothing will be delivered, and the subscription will become dormant. There is a report that can be used to identify and rectify such subscriptions.

## Skip logic

One way to resolve a dormant subscription, is to get payment going, and then to skip the outstanding issues to future issues.

Conceptually, we have to differentiate between data pertaining to:

* Past
* Present
* Future

As far as deliveries are concerned, we use the following tables to designate that:

* Past : Transactions
* Present : Subscription
* Future : SubscriptionIssue

In subscriptionIssue we use the following convention:

When Unitsleft == 0, that means that all the copies have been delivered.

When an issue is skipped, that whole row is deleted, as though it was never intended to be delivered.



## Delivery address processing

Note that there is a difference between a physical address of a customer, which is required on the Invoice, and the delivery address that is associated with a subscription.

Both physical and delivery addresses are stored in the same tables, however.

### State machine

I will describe here how the state machine works on MIMS.DeliveryAddress2. Refer to Subs.ea.

There is a state for each tab, as well as for Closing.

There is a class and a corresponding object for each state. There is also a pointer to the current state.

Each state object implements the same interface called ITabMovements. That interface specifies four methods:

* Bool ToSelect()
* Bool ToUpdate()
* Bool ToLog()
* Bool ToClose()

Each method returns a bool to indicate whether the state change should be allowed.

Each state object takes, as a parameter to the constructor, the DeliveryAddress2 form.

The system is driven from the event called TabControl\_Selecting.

I have subsequently realised that this is not a proper use of a state machine. A state machine should not be a façade for bad design. One form should do one thing. If you get one form to do too many things, through the use of many tabs, you are violating the principle of minimum coupling and maximum cohesion. One should not use the fancy name of a state machine to hide a multitude of if, but statements.

I have rectified this design error in the Sunrsise system, where I have split this functionality up into at least two forms – DeliveryAddressPicker and DeliveryAddressUpdate.

It might be a good idea to implement a state machine in the above way, but not in this particular case.

It might be a good idea to integrate the addresses we get from the Post Office and the DeliveryAddresses that we got from Mapit. This might be a future project.

# Customer management

## Duplication detection

We do not want two copies of the same customer. We try to prevent this by running a test before a customer is captured. Just to be safe, we plan to run the same test just before submission of a new customer.

There is a facility to consolidate duplicate customers, should they be detected.

## Liability versus contractual obligations ( due )

Liability is an indication of what will happen if the business closes it doors. How much do we owe the customer, or how much does the customer owe us.

Since the system allows for credit, is becomes possible for the customer to owe us money for services already rendered.

↓ Liability is connected to the payer, and not to the subscription.

This makes it possible to generate consolidated reports and consolidated invoices and debit orders.

In general, a contract is established with the customer, and it is expected of the customer top pay before delivery. In this way, a certain amount of money becomes due for payment, and that is indicated on an Invoice.

## Customer classification/profiling

### Classification by speciality

↓ This classification can be to any depth, and one customer can be classified under more than one heading.

You have to maintain the classification hierarchy, and that is done in the form of two tables called CustomerHierarchyClassification and Classification2.

You have to map the customers to the classification tree. This is done in the form called Profiling2, which calls a form called ClassificationPicker.

You populate the hierarchy proper via a form called AdministrationClassification.

The hierarchyId is used in MClassificationData001, i.e. in SQL, to build the full path from the root of to the leafnode of each classification branch a customer fits into.

The hierarchyId is however passed through the C# code so that it is used as nodename. In addition, it is passed down back to SQL procedures later on.

Bottomline – you need to carry hierarchyid in your C# code as well.

As far as customer classification is concerned, we do not need the hierarchyid on the C# side, so there is no need to keep it in the in-core datasets. In C# I use an integer in a linked list to manage the hierarchy. In addition, we found that the hierarchyid causes problems on Windows8 – another reason to discard of it.

The hierarchyId is used in the MIMS to MAPIT classification mapping, and in particular, it is copied to the clipboard such that it can be pasted in SQL. So, from that perspective it is required. But MAPIT is not going anywhere, so that functionality is now disabled.

### Classification by medical aid and language

The system caters for the capture of the medical aid and language preferences of the practitioners.

### Classification by location

The system caters for country groups as well as countries, and both postal and street addresses.

In addition the system caters for Geolocations and the translation between different formats of coordinates.

# Customer - Payment processing

## Overview

MIMS payments occur into two bank accounts:

* MIMS specific 012131830
* HeadOffice 9133

### MIMS specific

Debitorders appear here as internal transfers as a lumpsum.

The DebitOrder import captures the payments into the Subs system.

The allocation of those payments is done manually in a separate step, by reading the same XML as that which has been used to generate the debit orders. It is done in Subs4.Debitorder.XAML.

### Head Office

We also receive, on a daily basis, a spreadsheet for payments done into the HO account. The ones pertaining to MIMS are selected out of that manually and captured on the Payments/Transactions tab.

This caters for Direct deposits and for credit card payments that were done via the website.

### Cashflow statement

Riette integrates these two means of payment into a single statement on a monthly basis, and in this case it is also categorised by Product.

It turns out, when you look at delivery method implementations, that we will have to decide whether an issue is deliverable on the resolution of the issue itself.

The challenge is thus twofold:

* Allocate the payments to products for the cashflow analysis
* Allocate the payments to issues for delivery purposes.

It stands to reason that one cannot allocate money to issues that lie outside of the mapping to the subscription.

It is possible that a person can pay too much in general, and that should still be reflected in the liability to the payer.

In addition, it could be that a person pays too little, or such that it does not properly cover the next issue. That surplus will have to be captured in the liability as well.

I guess that the unit of allocation is an issue, irrespective of how many copies of that issue a receiver should get. You either get all the copies or you get none.

For every allocation, you should specify an amount and a subscription. The system will then allocate that to issues as far as it can go, and return the remainder of money that did not quite cover the next issue. That remainder is then added to the remainder of the payment and the system is then ready for another iteration of allocation.

Payments can be linked to Invoices, because the invoice numbers are used as a reference in the bank statements. You have a problem only if one payment covers more than one invoice.

If a payment fully covers all the subscriptions in the invoice, you can map revenue to subscriptions and indirectly to products. That should be automated. The invoices not paid in full should be flagged for exception reporting.

### Algorithm

**Due** refers to the amount of money that the payer has to pay if he wants all of his subscriptions delivered.

Due = Cost of all subscriptions – ( IssuesToBeDelivered + IssuesDelivered )

If Due > 0, then certain issues should not be delivered.

If Due <= 0, then all issues should be delivered.

When money is received corresponding allocations should be made. When money is given back, corresponding allocations should be neutralised.

As soon as a subscription is captured, one value goes up, i.e. Cost of all subscriptions. Issued to be delivered does not go up, because no additional payment has been received. What is due should go up by the full price e.g. R1000. However, if the previous Due was –R600, the value of Due will change to R400.

That means that allocations of R600 should be indicated on the new subscription. The ideal is that I can just refer the matter to PaymentAllocation2 for arbitration, merely supplying the Payer after having captured a new subscription, i.e. if you want to do it manually.

In there, I will be able to determine the amount allocatable, which would be R1000. Due, at this point in time will be R400. From this I could infer that R600 should have been allocated to make the books balance. I might not know where it should be allocated, and that is why I will call up the screen to get the user to make that decision.

For the books to balance, lAllocatable == lDue under all circumstances.

When Allocatable > Due, you have missed some allocations.

When Allocatabe > Due, you have missed some deallocations.

# Customer - Debit order processing

## Capture the static information

A debitorder is created and maintained from the Customer Menu/Debitorder. That puts a row in the SBDebitOrder table. In effect, in invokes a form called

Subs.Presentation.Debitorder.

## Calculation of due values on a monthly basis

### History

Initially, the flow of money was tied to the payer alone – i.e. it was not broken down to subscription level. The transactions involved include: Pay, Refund, WriteOffMoney, ReversePayment, ReverseWriteOffMoney.

In line with that practice, deliveries were handled on a “pay up front” basis. I.e. the delivery run for a customer was blocked when a delivery would cause us to deliver more issues than we have been paid for.

One of the implications of this approach was that deliveries continued, in any sequence, as long as the customer had money left, irrespective of which product was involved.

### MDR

Some clients started demanding that they get the MDR up front and the pay it afterwards via debit order. To get this to work on the delivery system, we had to introduce the ability to provide products on credit.

### MobiMims

For MobiMims and EMims it is important to know whether a specific subscription has been paid – as opposed to merely know that the customer ows us money??

This immediately necessitated the requirement to map each and every payment to the subscriptions it is intended for. A possible problem is incurred when a person pays for more than one subscription, or when a person overpays or underpays relative to an invoice.

It is more complicated than that, there is also a requirement to cater for debit orders. In this case, the whole subscription does not have to be paid, as long as enough is paid!

In addition to the MOBIMIMS requirement, there is also the requirement to report cash flow, not only by payer, but also by subscription and by date.

This switch in requirements was partially addressed by the PaymentAllocation form, and by the way in which the MOBIMIMS site queries the database via a web-service.

A major problem with this approach is that it is very manual and labour-intensive. In addition, the allocation table does not record the date of payment – so although we know the date of overall payments, we do not have that information readily available at subscription level.

Much had to be done to automate this process.

In addition to this automation, the **rollout** of this new approach should be made consistent across:

* Subscriptions – not just MOBIMIMS
* Cashflow operations – not just payments
* Historical data

The problem with the historical data is that, in the past a customer’s money was used up in the sequence that the deliveries occurred. So, now it could happen, in the new approach, that we want to allocate a payment to a product that has already been delivered.

Part of this exercise might be the write-off of debt older than 3 years.

### Continuous debit orders

Continuous subscriptions pose a problem for accurate liability calculations. For that reason, vouchers and debit orders have to be revised during a fixed period. So, all subscriptions have a renewal date.

? With continuous debit orders, you cannot calculate what is DUE as you do in the fixed payment case. In this case you have to ask what is due for the next 12 months. With than in mind, you then have to calculate what is due in a particular month, i.e. what you are going to hit the bank with, such that, comes delivery time, the issue is paid.

Furthermore, the monthly subtraction should be allocated to the issues on a first deliverable, first paid basis.

In the case where the subtraction does not exactly cover an issue, that amount will stand over for the next month. I.e. every cent received increases our Liability, but it is countered by each delivery that we make. A positive Liability indicates that we have received more money than we have delivered. That money may or may not have been allocated to the relevant issues yet.

We now attempt to recover debt via debit order in such a way that all the debt will be recovered by June every year.

### Calculation of proposal

It is assumed that:

* each customer has no more than one debit order
* when a customer uses debit orders, that is his only method of payment.

We want to keep the subtraction value as constant as possible. At the same time, we do not want the delivery of any issue to be blocked due to lack of funds.

It turned out later on that the calculation should be such that users pay the full amount of the issue the month before delivery, except for MDR, which is delivered up front on credit, and then that credit is recovered during the period up to the next June.

This approach is implemented on a spreadsheet that is populated by MIMS.DebitorderDoc.DeliveryProfile.FillBy4.

T. Include a flow diagram.

## Submission to bank

The system can create a proposal for a debit-order run and returns the result in the form of XML.

Riette then edits it to suit her needs.

She then goes onto a Standard Bank Website and manually enters the debitorders for each and every person.

This data is then inspected and OKed by the Accounting department, who releases it in the Website, probably with a password. This gives the bank the go ahead to take the money from the customers.

## Import

Once the bank did that, they send Riette a text file of the result, which includes amongst other things whether the customer did actually have money in the bank.

This text file is imported via Subs.Presentation.DebitorderImport. In this form, the result is validated, and, if OK, the payments are captured into the MIMS system as a payment.

It could even happen that, at this point in time, the customer paid in another way, in which case she has to be refunded, i.e. the debitorder has to be reversed. This is done manually with consent from the Accounting Department .

She accept the payment into MIMS and then execute a refund transaction against it. This is good accounting practice.

## Bouncing of DebitOrders

A debitorder might be accepted in say April, and imported as a payment in the Final version of the April bank statement. But, after that, a debitorder might bounce. In that case, in the May Final import, an amount will be indicated as a negative number, and that should result in a reverse payment.

That should happen automatically. Sometimes that causes a person to go into the red, into credit, and if there is not enough credit assigned to him, the system rejects the transaction. The rest pass the post process, and the outstanding one has to be again, after Riette has catered for deliveries on credit.

Note that when you import a negative value, it does not do a ReversePayment transaction, it merely posts a negative payment.

# Operational reports

The boundary of the system is between the business tier and the presentation tier.

In this context, the presentation tier is implemented in many technologies:

* Dotnet 4 datagrid
* Reporting Services
* Query analyser
* Excel with XSD and XML
* Excel with Sql Server
* Excel with Analysis Services

In order to empower the end users, provide them with an ER diagram and a listing of the enumerators used in the database.

Complicated queries are provided for by stored procedures with the name UQueries001 etc.

Caution is applied such that the security is such that the end user tools do not modify the database. A special read only user is employed for this purpose.

## Reporting Services

For human readable data, tables have been defined with names that end with Detail, e.g. SubscriptionDetail for repeatable data items and ReportInfo for non-repeatable items.

In reporting Services, I use embedded reports, rather than external reports. The reason is that too many redundant .rdlc files were hanging around, causing confusion and deployment problems. Also, there is no need to replace reports on the fly.

## Stored procedures

The following stored procedures have been created already:

UQuery001 – find duplicate customer based on surname.

UQuery002 – find duplicate customers based on company.

UQuery003 – profile of current subscribers

## Invoices and payments statement

Mledgerdoc.Statement.FillBy2

There is a difference between Liability that relates the movement of money and goods, as opposed to Invoices and payments, that is focussed on contractual agreements between the parties.

A subscription is smallest building block of a contract. Subscriptions can be grouped together in what is called a package and that is associated with a promotion.

More than one subscription can be grouped together under a single invoice.

Movement of money is recorded, both against deliveries, and against invoices. Such movements also include: refunds, write-off of money, reverse payments, and reverse write-off.

### Filtering

After that a split is made by summarising the data prior to FromDate into operation 29 called “brought forward”. Note that the value showed there pertains only to balance of payment, and has nothing to do with the deliveries or the allocation of payments.

NB We do not show credit notes on the statement, but it is factored into the balance.

### Past method.

Rather than just look it up from transactions of operation = 16, it is recalculated so as to take into consideration any modifications that might have taken place in the mean time. Thus, the only purpose of the first part and of #SubscriptionTransactions is to find the values of the subscriptions.

# Business Intelligence Reports

There are many other dimensions on which the data can be sliced and diced, and in future some of it could be included into the operational system, but we are not at that point yet.

A better way to do queries, than using the SQL hierarchical classification is via a cube in the SubsDW data warehouse. This has been done in the past, but the data warehouse is not currently updated on a regular basis. The ETL tools to do that is availible, should the need arise.

See Subscription.pptx.

# Diagnostic reports

The tools directory contains the SQL used for migration, plus diagnostic scripts.

## Stored procedures

There are cases where we keep redundant data for the sake of performance and end user simplicity. But this opens a consistency danger, which has to be checked for .

### Diagnostic tools

**MAdministrationData001**. Used to check the severity 1 exceptions.

**MLedgerdata003** – tally liability between Transactions and Customer

**MLedgerData009** – Payments without transaction records. Also checks for transaction pointing to non-existing subscriptions.

**MLedgerData007** – Check for discrepancies on customers or products between transactions and subscriptions.

**MLedgerData008** – Compare ledger with issue on StockDelivered

**MSubscriptionData036** – Verify Unitsleft. Also verify that it does not become negative? If no units are left, the sub should be cancelled or expired.

**MSubscriptionData011, 016** – Verify DeliveryMethod and DeliveryAddress.

**MLedgerData016** Tally DebitValue of deliveries with subscriptions

**MProductData018** – Identify phantom promotions.

**MIMSCustomerSpecificData002**  - Tally sum of disks received in MimsCustomer with delivery operations in the transaction table. You need a CustomerId here?

OPTIONAL and OPERATIONAL

**MCustomerData030** – Find all customers who owe us more than the credit limit allows. ON 15 Nov there was only 1. On April 2009 there were 68

**UQuery015**  - Possible dormant subscriptions. Undelivered subscriptions.

**CustomerDoc.CreditLimitViolatins**

### Rectification tools

MLedgerdata005 – used to **update** (sic) liability in Customer and CreditValue in Ledger on the Initialise operation.

MSubscriptionData012 – used to **reverse** deliveries, using the delivery XML.

You should list overlapping subscriptions to check if they are not duplicates.

MCustomerData011 – used to consolidate duplicate customers.

MPaymentData005 – Check for duplicate references in payments.

MProductData008 – Check for duplicate deliveries.

## Utilities

1 -1 records from the capture screen.

CompanyMerge.sql to locate ids of duplicate companies.

CompanyFix.sql to consolidate duplicate companies.

CustomerDelete.sql To delete customers if nothing points to them.

# Batch processing

There are three kinds of documents that are regularly distributed to customers:

Invoices

Statements/Receipts

Credit notes

There are a number of reasons why I want the generation of reports to run as a service on the server that hosts the database.

* The jobs are resource intensive, so I do not want to run it on workstations.
* Since the tasks take a long time, I want to be able to activate it asynchronously.

Generating reports for distribution occurs in three steps:

* Initialise the batch. i.e. create the XML
* Activate the batch. This is the batch job that runs on the WCF service.
* Distribute the EMAIL.

A problem with this approach was that, should an error occur in the batch job, it is difficult for the normal user to see what went wrong. For this reason, the document creation has been moved back to run in an inter-active environment.

Once a proper interactive monitoring of the service is in place, the original approach might be resumed.

It is assumed that these interactive jobs only read the database and that they are restartable. They interact via XML on a share called MIMSData on the server. You can run only a subset of the XML by editing the XML with Notepad. If I remember correctly, if the job is terminated pre-maturely, the system is clever enough not to reprocess XML entries that have been processed already.

# Service Orientation

There is a need for the CPD system to interact with the MIMS system, and this has been implemented via a WPF service.

At the moment MobiMims interacts with the MIMS system via a Web service. It would be nice to consolidate these two.

In future, there will be a need for the MIMS shop to interact with the MIMS system.

I have already implemented a RESTFUL service in WPF that can be invoked from remote devices, such as Android.

## WCF service

Implemented as the project called: MIMS.WcfService

## Connection to database

The system is based on the assumption that the data tier objects get their connection string from the static Settings object that is initialised by the host.

## NT service host

Implemented as Subs.NTService

I will try to run this under the user called MIMSBATCH. I think this is a local account on the server. Password = Akk3d1s. Member of Administrators group. Owner of MIMS database.

Is this all that is required to connect to the database?

## NT ServiceHost Setup

You have to run a fix here if you want to use it on Visual Studio 2013 or 2017.

# Adaptation points

## Settings

Because the reusable code is used against different databases, you need a mechanism to ensure that the system points to the correct database.

The connection string is set in the Properties/Settings part of the startup object. Then, in Mainform, in the constructor, that string is used to set a field in a static class that is also called Settings, and resides in the Data Tier where everybody can access it. Whenever Adapters are used, you have to execute AttachConnection, in order to get the right connection.

In the development environment in Subs.Data, there is also a Settings that provides a connection string for development purposes. To prevent this from being accessed in runtime, I change the Catalog to MIMSx before I deploy. IN this way, any runtime code that attempts to bypass the correct connection string convention, will result in an error.

## Presentation tier

Customer properties:

Make them all nullable, and handle the superset in the code.

You can mark them as Customer\_specific on the customer form.

Reports – have different .rpt files. We use Reporting Serrvices. But Crystal Reports allows you to import different report definitions at run time!

Preferences – no of issues – make this a property of the product.

## Business tier

References to user privileges – this can go to the database.

## Data tier

References to folders – this can be standardised to ‘c:\Subs’

For the batch jobs, this can be a command line parameter.

References to database – this should be driven by a config file?

## Administration data - static members and virtualisation

In order to subsequently analyse data, and in order to up the quality of the data, many entry fields are restricted to specific values. The collection of valid values is collectively called Adaptation Data.

It stands to reason that it should be possible for the Administrator of the system to edit and adapt this data. That explains the necessity for the ‘Administration’ menu item in the top left of the main screen.

Due to the magnitude of this data, performance problems were encountered, i.e. it took time to load this data into the application on demand. To alleviate this problem, the data were loaded into static classes, so that it needs to be loaded only once in the application, and then be available for subsequent use. Hence the gradual move to the class called ApplicationData2.

When the application data is changed by a user, the changes are immediately available to the rest of the application. The problem is that it is not directly available to other applications that rely on a cached copy of the data – unless they refresh the cached copy.

It turned out that the time is consumed not so much with the loading of the data, than with the binding of the loaded data to the controls. Fortunately, with Dotnet4, a new feature, called virtualisation became available, by which the binding occurs only on the current entries.

This new approach is not rolled out across all the adaptation data as yet.

## Tools

Survey it.

# Deployment

Since there are only 4 users, Click-Once works perfectly well in this environment.

# Suspended functionality - MAPIT

There is some code left in SourceSafe, but removed from the project, since it causes confusion. It includes DXS and MAPIT.

I've attached the documentation on how the Search tables are constructed (The TYPE field in RSearch is obsolete). Most named roads fall within towns where we have town boundaries. Hence an entry in TOWN. Some roads with names and numbers extend beyond the town boundary, hence one entry without a town. In some cases the Suburb has the same name as the town and this is not repeated in the suburb field. Please note that many roads with names do not have extensions (e.g. street etc.). For road names it is best to use the RD\_NAME\_F field which combines everything including a route number if present.

## Mapping to Medical Aids

One can specify these mapping declaratively, procedurally or via enumeration.

Declaratively would include:

* Not applicable
* Unknown
* None
* Some
* SomeNot

The latter two options suggests a hybrid, because you have to enumerate the some or the some not.

Furthermore, in the enumeration, you have to cater for three options:

* Contracted in = doctor will pay medical aid fees
* Contracted out = the doctor will submit to the medical aid, but ask his own fees
* Private = The doctor does not want to deal with the medical aid at all.

# Suspended functionality - Archiving

This has now been discontinued, because the operational system has a dependence on the history.

The immediate need arises from the fact that some statements become larger than 900 pages. Because it takes this long, we have to archive old data from time to time.

Even so, we have to provide a facility to query the archived data as well.

The archived data goes into a separate database, called MIMS Archive. This database currently contain 4 tables:

Transactions

Subscription

Comments

SubscriptionIssue

These tables are identical to the original tables, except that they do not have primary keys. Also referential integrity is relaxed?

The affected stored procedures are:

* MLedgerData011 – used for the ad hoc transaction reports
* MLedgerData022 – used to do the archiving of the transaction records: Maintenance/ Archive
* MLedgerData025 – Used to GetNetUnits for a specific user and receiver.

## Temporary bypass

Up to 6 April 2010, EMIMS disk subscriptions were archived. Thereafter, they have been left in the operational database.

In order to see whether someone has received an EMIMS disk, the system has to access the archived records. This establishes a dependency between the online database and the archive database that we do not want.

The plan is to keep the status queue, but as soon as a new disk is sent to everybody, or as soon as e-mims is stopped, this dependency can be dropped. ( Hein Reitmann ).

## Transactions

It is easy to remove transactions, since nothing is dependent on them.

It is dependent on itself, however, i.e. you have to keep the balance of units and the balance of the money from the archive in the operational database.

What happens on the second round – update if present.

## Subscriptions

To archive the subscriptions, you have to archive the subscription plus the two tables dependent on it, i.e. Comments and SubscriptionIssue.

## Customers

To be done. At the moment, some customers are just flagged as cancelled or active.

## Data warehouse

Somehow the data warehouse will have to be populated from the archive and from the live database.

# Roadmap

Upgrade SQL server. We are currently still on 2008.

Standardise the naming of the stored procedures.

Remove redundant stored procedures.

Revisit Archiving and Data warehouse.

At the moment the system does not cater for explicit Credit Note lines on the statement. This can be resolved only after the archiving issue has been resolved.

Change the deliverymethod of an existing subscription – in sito. This is problematic, because the subscription is a contract, and the deliverycost has already been factored into the UnitPrice – on which everything else depends.

Create a reminder system based on Comments and linked to Continuous Debitorder.

# To do

Convert diagrams from Enterprise Architects and Visio to Draw.io.

Try to get the BI report that we sent to Barbara. C:Subs?