Identity Management - A System Design

Jon Colombo & Keith Awcock

Introduction

This is the second of two articles addressing the practicalities of Identity Management. The first, which appeared in the October edition of Information Security Bulletin (ISB0710), explained:

- *The context* – the development of a centralised infrastructure to manage Identities across 100 plus systems.
- *The overall model* – effectively double-entry book keeping – "Should" records of authorised Identities are reconciled against "Actuality" records built directly from systems information.
- *Pitfalls* – the surprising number of hazards that were encountered.
- *A general set of requirements* for an Identity Management System (IDMS).

The current article describes a solution based on these requirements. Its purpose is to provide help for those developing an IDMS, and provide a general understanding of one approach for those considering the implementation of commercial solutions.

Constraints

The design philosophy was influenced by three environmental factors:

1. *Restricted Resources* – both human and financial.
2. *Dynamic Environment* – any system built would need to adapt and change.
3. *Management Structure* – the organisational structure meant that there was no possibility of forcing changes to other systems. Therefore, the IDMS interfaces had to adapt to the systems.

These factors produced three responses:

1. *Small Targeted Components* – the IDMS is made up of discrete single purpose components; e.g. one extracts a list of users from a NT domain controller, another separate component extracts the rights of the users.
2. *Use of Meta-Layers* – where external systems cannot be controlled, an abstraction layer is inserted that allows mitigation of the effects of any change.
3. *Use an Array of Inexpensive Machines* – small machines in an expandable array create low start up costs, cheap incremental expansion, adaptability and fault tolerance. The downside is that this can create complex dependencies.

Model

The IDMS consists of five blocks, see Figure 1:

1. *Raw System Data* – each system under administration reports on users in its own way. Thus the data comes in all shapes and sizes and with differing reliabilities.
2. *Import of Data* – collects the data and imports it. Runs imports automatically at predetermined times, deals with all the complexities of importing from unreliable sources and reports on problems.
3. *Operations* – organises the imports and maintains the IDMS. Has user interfaces to manage import and report processes, view and resolve problems.
4. *Identity Administration* – the main use of the IDMS takes place here; management, monitoring and manipulation of identity data. From the administrators perspective this is Identity Management.
5. *Management Information* – provides metrics for management and reports for the owners of systems.

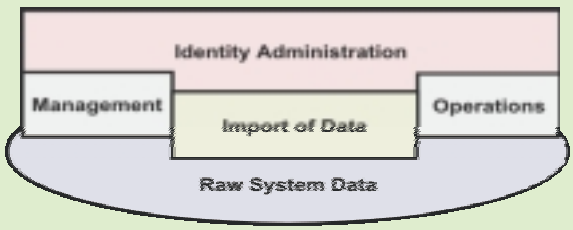

Figure 1 - **IDMS Blocks**

The Components

Figure 2 on the following page shows the components overlaid on the IDMS block model:

Data Stores

There are four separate 'stores' – not individual tables, but logical collections of data. Working clockwise from the top:

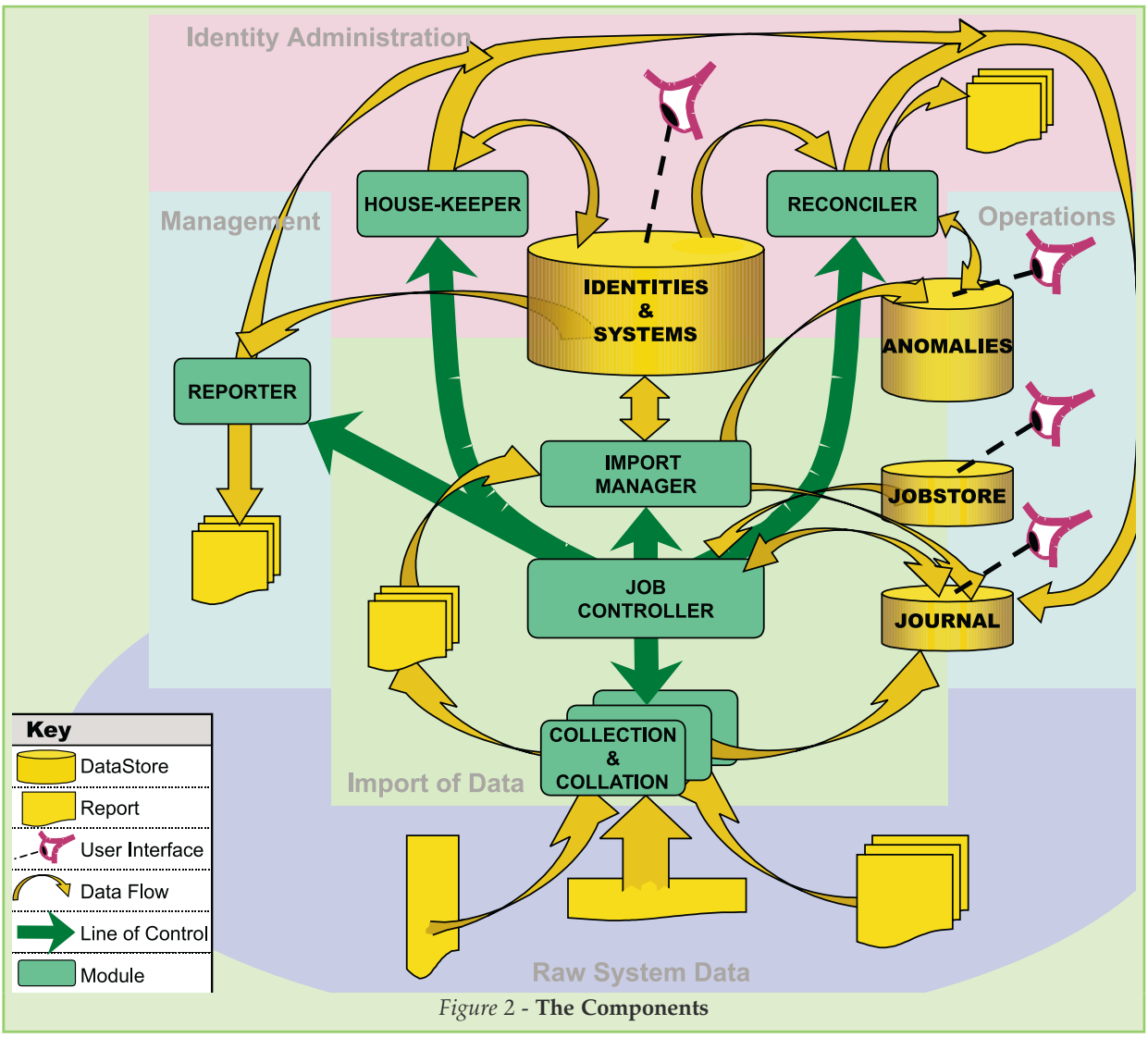

Figure 2 - **The Components**

1. *Identities and Systems* – the main data store, contains details of people, their identities and the systems administered.

2. *Anomalies* – records each problem encountered by the IDMS, both operational and procedural.

3. *Jobstore* – stores the jobs to run, their parameters, and the order in which they need to be run. Also provides documentation for the IDMS.

4. *Journal* – the IDMS's 'memory', recording what has run and when, and provides evidence of activity for the auditors.

Modules

There are six functional modules:

1. *Collection and Collation (C&C)* – a number of programs, driven by the Job Controller module, that transport and convert Raw System Data from its original location and format to a common CSV one. Progress is recorded to the Journal.

   Generally there will be at least one C&C programme per data source. This modular approach both provides complete insulation from changes in target systems and means that new requirements need no amendments to existing code, thus providing greater reliability.

2. *Job Controller* – the control module for the IDMS. Reads jobs from the Jobstore, manages C&C Import Manager Reporter Housekeeper and Reconciler modules.

   Uses the Journal to track and manage progress.

3. *Import Manager* – controlled by the Job Controller module, intelligently injects data into the Identities and Systems datastore from the CSV files produced by the C&C jobs. It is here that the filters described in the first article are implemented. The Import Manager reports normal progress to the Journal and problems with imports by creating 'Anomalies'.

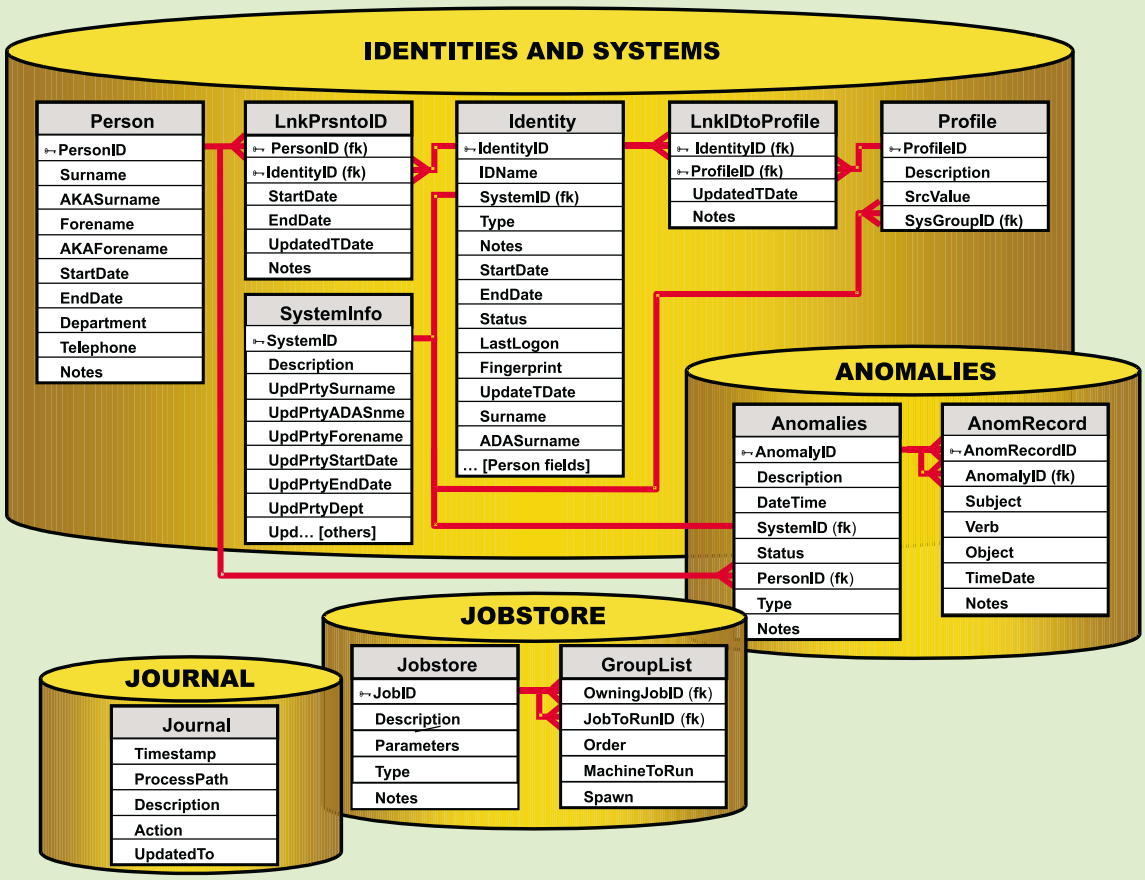

Figure 3 – **Entity Relationships**

4. *Reconciler* – compares the 'Should' records against 'Actuality' records for each system, and searches for dangerous combinations of rights under Job Controller control. Creates an anomaly record or paper report whenever it finds a mis-match. Progress is recorded in the Journal.

5. *House-Keeper* – under Job Controller control, runs grooming processes on the data to update the personnel data, (e.g. telephone numbers, department, leaving date) apply data retention/deletion rules, and maintain the database. Progress is recorded in the Journal.

6. *Reporter* – produces and distributes (via email or hard copy) management reports, statistics etc. Is controlled by Job Controller and records progress in the Journal.

User Interfaces

The IDMS has four user interfaces that allow the system to be used and maintained by non technical people:

1. Identity Administration - an application that provides a way to enter and modify data, for example manually add a new user, or move ownership of an account from one person to another.

The Identity Administration application needs granular access control itself; e.g. read-only access to all the information for auditors and views of the data limited to single systems for system owners.

2. *Anomalies* – an interface to the anomalies created by Import Manager and Reconciler. Provides the functionality necessary to deal with the anomaly; e.g. if the anomaly indicates a user has left then the 'Should' accounts can be closed and emails sent requesting closure of the 'Actual' accounts at the touch of a button.

3. *Jobstoreal* – lows users to manage the jobs run by Job Controller, supply or change parameters and add and schedule new jobs.

4. *Journal* – a control panel type interface that reads the data from the Journal and indicates the state of the jobs, green for successful, red for failed and Amber for running.

   Gives direct access to the Journal data so that deeper analysis can be carried out if necessary.

Component Architecture

Although the model is simple, and the number of components relatively small, this provided no guide to the complexity of implementation. There were difficult design decisions necessary for virtually every component. These are examined below.

Data Stores

These are discussed with reference to the Entity Relationship diagram, Figure 3.

Identities and Systems

The storage of identities had five major design decisions:

1. *References to People* – the IDMS was not seen as a database of computer accounts, but a collection of references to people's identities on other systems. These take the form of a copy of the important information in a normalised form; *not* a vector or pointer to the data.

   Thus every Identity record, whether it was gathered to provide HR data, 'Actuality' data, or 'Should' data potentially contain the same fields, from which the full Person record can be created. This approach creates tremendous flexibility by allowing the combination of data across systems to bridge gaps. e.g. details of a person can be gathered from more than just the HR system. A more complete picture can be achieved by importing data from HR's system, the door access system the telephone list and the helpdesk.

2. *Data source trust scoring* – if data is to be combined from different sources, then the problem of which takes precedence when entries on two systems conflict. Some form of trust scoring has to be implemented.

3. *Use of Double Entry Book-keeping* – the first article explained that the whole approach is based on the principles of double entry book-keeping. A complete record of a person's identity on a system has two parts; a 'Should' entry and an 'Actuality' entry. To distinguish them a 'Type' field is used.

4. *Support for Sharing Accounts* – the sharing of accounts has to be represented in any system that mirrors the real world. One account may be linked to many people, and each person may have access to the account for different periods. All these changes have to be recorded.

5. *Depth of Model* – a decision was made to keep things simple by only recording Identities and Profiles, not lower level rights (i.e. menu options, books, curves etc.)

These design decisions produced the structure shown in Figure 3:

Person – contains anybody authorised to use any of the banks systems. Staff, suppliers with access for maintenance purposes, clients, customers, and 'virtual people' used for system accounts.

LnkPrsonToID – allows the connection of X Persons to Y Identities, thus solving the shared account support problem, holds the dates each Person has access to the Identity.

Identity – information on a particular Identity, the name, the system that the Identity belongs to etc.

Note: The last login field is Identity specific, not dependant on the people who have access to the Identity if it is shared.

LnkIDToProfileal – lows for security models where a single Identity can have more than one profile; e.g. NT Unix.

Profile – records privileges an identity has on a system. Only one level is considered, (see 5 above) so for systems that implement a multi-layer security model, the data is 'de-normalised'. The trick here is to determine the lowest level of 'distinct permissionable right'. However, it was found that on some systems dissimilar low level rights share a name. Therefore it is prudent to record the full path to them; e.g. the 'Curve A' low level rights belonging to an Input group is fundamentally different to the 'Curve A' rights that belong to the Release group, even though they share a name.

SystemInfo – records information about systems. It should be noted though that a system in this context does not necessarily equate directly to computer systems under administration. Datasets pulled in for information purposes (HR, Cardswipe, Telephone Directory etc.) also require system records.

There is one UpdPrtyxxx field for each Person field, each contains a trust score. i.e. HR End-Dates are assigned a higher trust score than Telephone Directory End-Dates. However, telephone numbers from the Telephone Directory are assigned higher trust scores than those from HR data. If the most trusted source contains no data, then the next most trusted source is used. When these are netted together the Person record ends up consisting of the most trusted source for each field.

Anomalies

An anomaly is created when either the Import Manager or the Reconciler encounters something that does not match its rules. Management of these anomalies is easier if:

Anomalies refer to people – provision of information about the Person an anomaly is associated with helps the administrators investigate the anomaly efficiently.

Anomalies are directly fixable – when an anomaly is created there should be enough information

stored with it for an operator to make a reasonable decision on how the handle it.

The Anomalies database consists of two tables:

Anomalies – describes the anomaly e.g. loading data into NT System could not distinguish the owner of an identity between two people.

AnomRecord – a child table storing information required for the operator to resolve the anomaly e.g. the information on the two people (names, departments phone numbers).

Jobstore
The Jobstore has four design criteria:

Identification – each job and every program that runs must be identifiable.

Sequencing – it must record the sequence in which jobs will be run.

Parameter Values – the storage of parameters available must be flexible enough to allow for virtually any type of parameter.

Self Documenting – job has a unique name and Notes field where the purpose of the job is explained. A report can be created documenting the system in its entirety.

These resulted in two distinct types of job:

1. *A list of Jobs* – used by the Job Controller for scheduling.
2. *Action* – interpreted by the Job Controller and passed to one of the other modules for implementation.

They were implemented using two tables:

Jobstore – the Parameters field is free text, the content varying dependant on the module to be called. The Notes field allows documentation of the job, whilst the Type field tells the Job Controller how to handle the instructions.

GroupList – used to organise jobs, the MachineToRun field is used to implement the 'Array of Inexpensive Machines' concept – the Job Controller on any particular machine will only run jobs with the name of its host in this field. The Spawn field tells the Job Controller whether to wait for completion before initiating the next job.

Journal
The journal is very straightforward: a single table recording the date and time of an action and the job that caused it.

The Journal provides three key functions.

1. *Operational data* – to manage day to day production, monitor and, if necessary, debug day to day job flows.
2. *Job Controller data* – provide information that the Job Controller can use to manage its automated processes. i.e. to 'chain' imports and reports ensuring that the same data is not picked up or processed twice.
3. *A history* – to provide evidence that the work has been done.

A single table performs all three functions. The ProcessPath field contains the complete job path of the Jobscript or programme (Process) that created it, thereby allowing code re-use whilst retaining traceability. i.e. if Job 203:= Job Controller, 4 =Housekeeper, 15 = NT cleanup, and 5 = Programme to remove old closed Identities then the process path would be 203.4.15.5. But if the same process was used in the Unix group (JobID 100) to remove Unix Identities, it might have a process path of 203.4.100.5. It is clear where the process was running in the cycle and which set of data it was being worked on, (see sample Journal, Figure 5.)

The Description field generally contains a human readable summary of the activity to meet the operational data requirement. It can also be used by the Job Controller; e.g. a job may record the number of rows updated for a particular import. A Reporter process can then periodically check these figures and report on any variations.

The UpdatedTo field is important where imports and reports are going to be 'chained'. Job Controller locates the last time a job was run using the process path, it then knows that it only has to pick up data that is more recent than the UpdatedTo date.

Modules
Each module called for some specific design decisions:

Collection and Collation
Collection and collation provide the transport layer and consist of two logical processes.

1. *Collection* – transports the raw data from each system as found. Collects all the data for the selected system in a given time period and presents it for collation; e.g. in a multi-host application, the process collects the files from each host in that system and concatenates them into one file.
2. *Collation* – The raw data is normalised – converted into two standard CSV formats. Having one file for Identities and one for Profiles is not strictly necessary but does save considerable time when matching. The standard CSV formats allow for all the useful data that any system might be able to provide, for most systems they will be partially empty.

C&C engines are built using various tools dependant on the source data; e.g. Perl, C++, MSExcel macros, MSAccess.

The standard Identity CSV file contains eleven fields (Identity, Surname, Forename, Telephone,

Department, ID Status, Start-Date, End-Date, Last Logon, Unique ID, Fingerprint) of which only two have non-obvious uses:

1. *Unique ID* – used where the data source has a unique primary key that can be guaranteed never to be repeated, even over time; e.g. National Insurance Number.
2. *Fingerprint* – is created by the collation process by joining system data to create what is hoped will be a unique key; e.g. the users name may not be unique, but can be made so by concatenating it with the account creation time to create a statistically probable unique value.

The standard Profile CSV file contains only two fields, *Identity* and *Profile*. This is all that is needed provided the profile file is derived at the same time as the Identity file and the Profile import is run after the Identity one. If this is the case, a direct match using just the Identity/System combination is safe. Normalisation allows existing systems reports to be used – rather than having special reports created or using agents.

Job Controller

At the centre of the IDMS is the Job Controller. It is responsible for starting jobs in the correct order; supplying the modules with the correct parameters and reporting problems to the operation control systems. The Job Controller itself was written in Perl, a corporate scripting standard.

The Jobstore data is structured such that the Job Controller only needs to be given the Unique ID of a starting job and can then follow a 'tree' calling the relevant modules and programs as it goes. The basic flow for this procedure is shown in Figure 4.

In order for this to function correctly three conventions have to be observed:

1. All processes are stored in a single directory, the location of which is stored in an environment variable.
2. Jobscripts are written in perl, with a naming convention of P<uniqueid.pl>. A Jobscript may call lower level programmes which will have their own ID's.
3. Each Jobscript has an entry function, main(), that is in a namespace derived from the JobID, e.g. the Jobscript for process number 17 is shown in Listing 1.

This allows a library of perl subroutines to be built up to handle the more common requirements; e.g. Getting a named parameter from the command line or changing the format of a date string.

Each Jobscript or programme (Process) records its own progress in the Journal.

If any process fails it records the relevant facts in the journal. The calling process will also fail and record that in the journal, leaving a complete audit trail.

By convention the a number of rules are applied

1. Each process has its own Unique ID. (ProcessID)
2. Each Process must be passed the 'ProcessPath' from its parent. The full ProcessPath will then consist of the parental one concatenated with the processes own ProcessID. It is this full path that is used in the Journal.
3. A process always records its start and completion in the journal.
4. A failing process will always record the reason it failed in the Journal
5. A successful process records the date and time of the data used where required for allow 'chaining'.

Listing 2 provides an example of these rules implemented.

Figure 5 shows an extract from the Journal, from which it can be seen that the Import Manager fails when used the second time to import normalised NT data. The Journal shows how this failure flows through the process that call the Import Manager.

Import Manager

The IDMS is based on the premise that the only thing it imports are identities, irrespective of whether these come from administered systems,

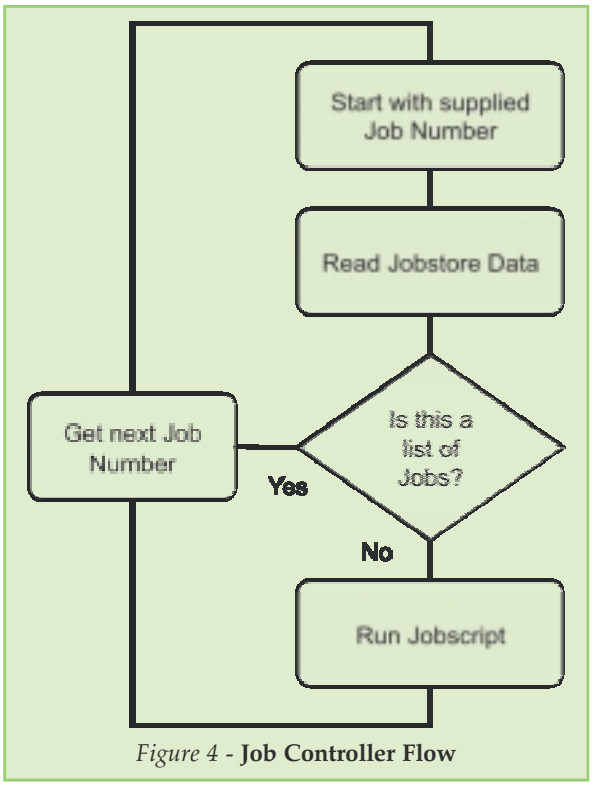

Figure 4 - **Job Controller Flow**

personnel records or any other source. Person data is derived from that Identity data. Thus the Import Manager, written in C++ reads incoming data a line at a time and attempts to match each line to existing data. Matching has already been covered in the previous article to a reasonable level. However, there are two additional considerations:

1. *Acronyms* – to support the fact that people may use nicknames or abbreviations, the Person table has two fields to hold these, AKASurname and AKAForename. When matching on a Name, the queries always 'OR' on the associated AKA field as well.

```
require "utils.pl";
package P17;

sub main()
{
   my $args = shift;
   my $src = GetNamedParam($args, "source");
   my $dest = GetNamedParam($args, "dest");
   # More code that does something useful
}
```

Listing 1

2. *Updating an Existing Identity* – once an incoming Identity has been matched to one on the IDMS, then the Import Manager has to determine what action to take based on the Status (Open, Closed, Disabled) of both pieces of data. The rules used are shown in Table 1.

Note: The Closed/Disabled to Open transitions create a new identity. This builds up an audit trail of account activity; e.g. the records of an Identity belonging to someone returning after a period of absence will still show the gap in use.

Housekeeper

The Housekeeper, built using Perl and Stored Procedures performs two major functions, updating person information from identity data and cleaning out redundant identities. It can also be used for various small grooming tasks such as closing links where the Identity or Profile has been closed or deleted.

The major functions are:

Updating People – built up from the Identity data; e.g. the name from HR or the Door card database; the telephone number from the telephone system or the HR database using the mechanism explained under Datastores – Identities and Systems above. This allows more than one system to update a particular Person field. If a person has a Department entry in both HR (priority 10) and the telephone directory (priority 45) then the HR value is used, if however there is no HR entry the telephone directory value is used.

Data Retention – to comply with Principle 5 of the Data Protection Act, the IDMS cannot hold personal data for longer than is necessary. For this reason automatic deletion of people and anomalies is required. This process requires some rules beyond a simple if-the-person-has-left-for-more-than-1-year-delete-it one because there may be an outstanding anomaly (investigation) in which case as much information about that person must be kept.

Reconciler

The aim of the IDMS is to ensure that all access to the managed systems has been authorised. The reconciler uses Perl and Stored Procedures to apply the double entry book-keeping principles described in

```
require "utils.pl";
require "Journal.pl";
package P17;

sub main()

{
my $args = shift;
my $procpath= GetNamedParam($args, "proc");
$procpath .= ".17";
# journal status 4 is started
WriteJournal($procpath,4,"Start HR Import");
my $src = GetNamedParam($args, "source");
my $dest = GetNamedParam($args, "dest");
# More code that does something useful
my $err=HRCandC($procpath,$src,$dest);
if ($err == 0) {
   Serr=ImportManager($procpath,$dest,$otherparams); }
if ($err == 0) {
   # journal status 1 is successful
   WriteJournal($procpath, 1, " HR Import Succeeded",
                                          $file_date); }
else {
   # journal status 3 is failed
   if ($err == 1) {
   WriteJournal($procpath, 3, "HR Import - Source file
                             not found", $file_date); }
   else {
   WriteJournal($procpath, 3, "HR Import - Reason
                               Unknown", $file_date); }
   }
   # journal status 5 is completed
   WriteJournal($procpath,5,"Completed HR Import");
}
```

Listing 2

Timestamp	Process Path	Status	Description	Time
2002-07-013 00:01:00	1	start	Start Main Process	
2002-07-013 00:01:05	1.17	start	Start HR Import	
2002-07-013 00:01:09	1.17.11	start	Start HR Collection and Collation	
2002-07-013 00:02:45	1.17.11	complete	Completed HR Collection and Collation	
2002-07-013 00:02:50	1.17.5	start	Start Import Manager	
2002-07-013 00:07:00	1.17.5	complete	Completed Import Manager	
2002-07-013 00:07:05	1.17	run ok	HR Import Succeeded	2002-07-12 23:30:00
2002-07-013 00:07:13	1.17	complete	Completed HR Import	
2002-07-013 00:07:20	1.58	start	Start NT Import	
2002-07-013 00:07:23	1.58.23	start	Start NT Collection and Collation	
2002-07-013 00:07:32	1.58.23	complete	Completed NT Collection and Collation	
2002-07-013 00:07:32	1.58.5	start	Start Import Manager	
2002-07-013 00:07:34	1.58.5	failed	Import Manager failed to connect to DB	
2002-07-013 00:07:34	1.58.5	complete	Completed Import Manager	
2002-07-013 00:07:35	1.58	failed	NT Import - Import Manager failed	
2002-07-013 00:07:35	1.58	complete	Completed NT Import	
2002-07-013 00:07:36	1	failed	Main Process - Child process 58 failed	
2002-07-013 00:07:38	1	complete	Completed Main Process	

Figure 5 – **Sample Journal**

the first article, and creates anomalies or reports in the following cases.

Note: Anomalies serve to identify both security and service issues.

1. *Unauthorised Identity* – 'Actual' identity with no 'Should' value. This could be indicative of intruders on the system.
2. *Unauthorised privileges* – 'Actual' identity has profiles with no corresponding profiles on the 'Should' identity. This could be indicative of intruders on the system.
3. *Leaver not dealt with* – Person data shows person has left but 'Should' identity is open. Indicates the verification process has failed.
4. *Leaver deletion not happened* – Person data shows person has left but 'Actual' identity is open. Indicates where the deletion process has failed.
5. *Poor Identity creation service* – 'Should' Identity exists with no corresponding 'Actual' identity. Created only after a defined number of days, aims to identify problems with the execution of account creation.
6. *Poor Profile creation service* – 'Should' profiles exist with no corresponding 'Actual' profiles. This anomaly is thrown only after a certain number of days have passed.
7. *Poor Identity deletion service* – 'Should' identity is closed but the 'Actual' identity is open. Created only after a defined number of days to identify problems with the execution of account deletion.
8. *Poor Profile deletion service* – An 'Actual' identity has open profiles with corresponding closed profiles on the 'Should' identity. This anomaly is created only after a defined number of days and identifies problems with the execution of profile deletion.

Source \ IDMS	Open	Closed	Disabled
Open	Update database values	Create new identity	Create new identity
Closed	Update database values and close identity	If values have changed create an anomaly	Update database values and close identity
Disabled	Update database values and close identity	Create an anomaly	If values have changed create an anomaly

Table 1 – **Input Decision Matrix**

Reporter

A combination of MSAccess, Excel and Stored Procedures creates and distributes via email or hardcopy IDMS management reports. These serve two purposes, reporting to system owners on the state of their systems and the provision of management statistics on the operation of the IDMS. Reports are:

1. *Accounts not used in X- Days* – allows the system owner to monitor for unused accounts, saving on licensing fees.
2. *Recertification* – displays all Identities and profiles for all users of a particular system. This is used as a periodic check (quarterly) to ensure users have not incrementally gained rights that would not be given if requested in one application.
3. *System users email* – bonus functionality allowing system wide emails to be sent; e.g. for systems that do not automatically enforce password change rules, an email can be sent every 30 days to remind the users of that system to change their passwords.
4. *Anomaly statistics* – a weekly report giving an account of how many anomalies of each of the Reconciler anomaly Types. This is used to prove security service levels to upper management.

Conclusion

The first article set out thirteen requirements that an IDMS must meet, whilst the current article describes the solution implemented. It is now possible to bring these together to demonstrate how each requirement was met:

1. *No Agent Technology* – the C&C routines are passive, pull based technology requiring read-only access to standard reports produced by the administered systems. All programmes are run on the IDMS systems own hardware.
2. *Intelligent Updating of Datastore* – the Import Manager updates Identities and Profiles using a sophisticated set of filters, whilst the C&C routines can be engineered to cope with the vagaries of different transport mechanisms.
3. *Intelligent Scheduler* – the Job Controller combines Jobstore information of what is required with Journal data about what has already happened. It uses this to sequence and control imports, reconciliations, reports and housekeeping functions.
4. *Imports Different Data Shapes* – the normalisation function embedded in the C&C routines allows the IDMS to adapt to any data format.
5. *Incremental Data Importing* – the Job Controller passes the information in the UpdatedTo field of the Journal to the C&C module as a parameter. The C&C Module then only extracts the correct data.
6. *Out of Sequence Data Importing* – The updated to values in the Journal allow the C&C to identify out of sequence data. C&C modules are coded to handle this eventuality appropriately for each system.
7. *Partial File Importing* – can either be dealt with in the C&C layer or by passing the right parameters to the Import Manager.
8. *Report All Import Anomalies* – one of the prime design criteria of the Import Manager.
9. *It Must Journal Its own Activity* – activity is recorded in depth in the Journal data store.
10. *Datastore Comparison Reporting* – the dedicated Reconciliation Module handles this.
11. *Choice of Output Formats* – management reports are available as printout or email, whilst the Reconciler can either create anomalies, or reports (or both).
12. *Automated Housekeeping* – the dedicated Housekeeper handles this.
13. *Provide Operational Tools* – there are three separate user interfaces to meet operational requirements.

References

[1] Colombo, J & Awcock, K, *Identity management Experiences*, Information Security Bulletin, Vol 7, Issue 10, October 2002, pp 25-32.

Jon Colombo started work as an archaeologist with degrees from London and Oxford. From there he moved into IT, gaining an MBA from City University in 1993. For the last 10 years he has worked entirely in Information Security, setting up InfoSec functions at United Friendly Insurance and at WestLB AG, London, where he still works. He is a qualified CISSP.

Keith Awcock, graduated as an engineer from Kings College, London in 1989, moved directly into software engineering with Phillips. For the last 5 years has specialised in Information Security, working at WestLB AG in London. He is a qualified CISSP.