Migrating from AMS to UAPM using API’s

Contents

Acknowledgements

Special thanks to the following people for reviewing this document and contributing to it with their comments and suggestions: Hood Khalifa, Mohammed Alaliwat and Shardul Kothari. This document was written by Ali Almossawi <almossawi@gmail.com> on June 6, 2007.

1 Introduction

This report documents the experiences of a small team that worked on converting the data from a legacy asset management system (AMS) to Computer Associates' Unicenter Asset Portfolio Management (UAPM). It will hopefully benefit anyone thinking about making a similar kind of migration, even if it's in a very small way. Since the report wasn't a project requirement, it is being released as an unofficial document[1]; the company name Silkworm Inc. used herein is fictitious.

As part of Silkworm Inc.'s move towards using more state-of-the-art software systems to support their operations, they were phasing out a number of legacy systems, one of which was their asset management system that was to be replaced with a new state-of-the-art one. For one reason or another, the third-party that was responsible for migrating the data had had problems doing so and therefore the decision was made to have a couple of in-house developers work on the migration. The previous half-completed work had resulted in almost all the software and hardware models and a small number of assets being created[2].

2 Planning the work

The work was carried out in the following stages:

Break-in: None of the developers had any experience with UAPM or its API's, so the first thing to do was to get up to speed with the software system by reading through its documentation and playing around with sample code.

Mapping: The next step was to determine the mappings between our AMS and UAPM models. The former uses simple tables to store data about assets, which it classifies as either PCs, peripherals or software systems. The latter however has a much richer model that not only allows for more data to be stored about assets, but also allows one to easily construct much more complex queries because of the way an asset's different data are tied together. The process of mapping AMS fields to UAPM ones provided a good way to become more comfortable with UAPM. It also brought to light a few potential pitfalls and discrepancies that needed to be taken care of, such as missing fields. Details of the mappings were documented in a report that was sent to the concerned manager, system administrator and a member of the Enterprise Management team for comments. This step ended once the go-ahead had been given by the mentioned stakeholders to proceed and the recommendations outlined in the document had been implemented.

Scripting: Next up was to write the scripts. The design stage took about a day; thereafter, coding took about a week and implementing changes due to modified requirements took another week. The current asset data had been exported from AMS into Excel spreadsheets, refined and normalized by the system administrator and then imported into MS SQL tables via Microsoft's SQL Enterprise Manager. A copy of UAPM, which by now was an effective mirror of the one in production, resided on the same staging machine. Because the "software script" requires that all the assets already be defined for it to create entitlements on them, the two scripts had to be run consecutively.

Post-scripting: The final step involved verifying the converted data, which we had defined as a confirmation that a subset of the data in UAPM of an arbitrary size "looked right". Hey, if management was happy with that kind of metric so were we!

---

[1] Releasing it as an official report is a lengthy process so we thought it would be best to get it out in its current form for now. This was done with the consent of our manager of course.

[2] In UAPM, an asset is based on a model. For example, a model may be a particular type of PC and all PCs of that type currently deployed in the environment are assets that are based on it.

3 Mapping AMS to UAPM

The content of this section and its accompanying appendices have been adapted from the previously mentioned mappings document. As a result of doing this exercise, we were able to uncover a number of discrepancies between the UAPM installations on the staging and production servers including issues such as differently named, missing or unpopulated fields. The following list explains the notation that's used throughout this section to describe the mappings. The footnotes have been kept intact to give an idea of the types of things that the exercise brought to light and subsequently corrected. Note that AMS stores data about assets in three tables: PCs, Peripherals and Software Applications whereas UAPM has notions of models, assets, contacts, relationship objects, software entitlements, etc.

- field abc ≡ field xyz: UAPM field on the left-hand-side is equivalent to the AMS field on the right-hand-side
- Object.*Generic Component*[<abc>] ≡ abc: The AMS field maps to a UAPM asset as a generic component
- Object.<abc> abc ≡ abc: <abc> on the lhs is the value of abc on the rhs
- Object.<abc def> ≡ abc, def: The UAPM field on the lhs maps to two AMS fields
- ~~text~~: Seemingly unnecessary field; safe to ignore

The UAPM names are the ones shown in the application and not the one used in the API. The full lists of mappings are available in the appendix.

4 Writing the scripts

Though the implementation was done in C#, the finished artifacts are referred to as scripts throughout. A class DataManager is used by all three scripts to connect to our data sources and sinks. Of course, the API dll's need to be referenced in a project before you create an object of type iconnect.ArgisConnect. Note that the source code should have accompanied this report. If it wasn't please contact the author.

4.1 Converting PCs and peripherals to hardware assets

This script is made up of a number of checks, a number of operations and a logger. The checks account for all the different situations that came about, such as missing models or users. The operations appropriately add or remove assets, models, individuals and relationships. Note that although the AMS data had been refined to a great extent, meaning that names such as "e-Trust", "eTrust", "E Trust" had been standardized, a few discrepant names had been missed and were therefore handled in the code. Code Listing 1 shows the main portion of one of the scripts.

```
iconnect.ArgisConnect oArgis = DataManager.connectToUapm();
System.Data.SqlClient.SqlConnection sqlConnection = DataManager.connectToAms();

//select everything from the PCs table
System.Data.SqlClient.SqlDataAdapter sqlDataAdapter = new
       System.Data.SqlClient.SqlDataAdapter();
System.Data.SqlClient.SqlCommand sqlSelectCommand = new
       System.Data.SqlClient.SqlCommand();
System.Data.DataSet ds = new System.Data.DataSet();
sqlSelectCommand.CommandText = "SELECT * FROM uapmadmin.AssetHardware";
sqlDataAdapter.SelectCommand = sqlSelectCommand;
sqlSelectCommand.Connection = sqlConnection;
sqlDataAdapter.Fill(ds);

for(int i=0;i<ds.Tables[0].Rows.Count;i++) {
       Console.WriteLine((i+1)+". currently processing PC with tag # " +
              (!ds.Tables[0].Rows[i]["Tag No"].ToString().Equals("") ?
              ds.Tables[0].Rows[i]["Tag No"].ToString() : "(missing tag #)"));

       //package current record (PC) as an asset and do all noted checks
       packageAmsPCAsUapmAssetAndInsertIntoUAPM(oArgis, ds.Tables[0].Rows[i]);
}

Console.WriteLine("All done with PCs cap'n!");
```

Code Listing 1: AmsPcToUapm shown iterating through all AMS records and calling a method with a ridiculously long name; similar to the AmsPeripheralToUapm one. For details of that method, please take a look at the accompanying source code.

`ds.Tables[0].Rows[i]`, which is passed to `packageAmsPCAsUapmAssetAndInsertIntoUAPM()`, represents a single AMS record and is used for setting the fields for that record's corresponding UAPM object and all dependent objects. Note that initially, the script looked for and removed existing versions of an asset before adding a new one, but that was then changed when it was realized that certain third-party applications like Crystal Reports depended on those assets' UID's. The script was therefore modified so that it instead only overwrites the fields it is interested in. As for an asset's generic components, such as it's operating system, CPU, RAM, HDD, etc. it clears them all by deleting the relationship that associates the two together and then recreates them. This is to ensure that only the items defined in the script are listed as generic components for assets[3]. Generic components are added using the HWConfiguration dependent object (HWConfigurationObj).

4.2 Associating software applications with PCs

What this script does is create associations between software and hardware assets. Since the AMS data stores a PC tag number for each software application, for each AMS record , it looks up that PC by its tag number, then the software by its name and uses the Software License dependent object (SWLicenseObj) to store the software entitlement details such as its name. Finally, the Software License object is linked to the previously retrieved software and hardware assets. As with the previous script, this one keeps a log of everything that is going on. Code Listing 2 shows the part of the script that links the Software License with the hardware asset (PC) associated with it. Because of the number of checks the script ended up having to do, it took slightly longer than expected to finish processing all of its records.

[3] PCs of course and not peripherals

```
iconnect.SWLicense softwareLicense = asset.SWLicenseObj();
softwareLicense.addNew();
softwareLicense.SWLicenseName = "Workstation-based";
softwareLicense.SWLicenseDate = getDate();
softwareLicense.SWLicenseTime = getTime();

//creating link object to link the software entitlement to the hardware asset
iconnect.Link link = softwareLicense.LinkObj();
link.addNew();
link.LinkName = "Licensed to";
link.LinkToObjectName = hwcoNa; //HW asset name
link.LinkToObjectID = hwcoID; // HW asset ID
link.update();
softwareLicense.update();

result = softwareLicense.PrimaryObjectName + " is Licensed to " +
       link.LinkToObjectName+"("+ds.Tables[0].Rows[i]["Tag_No"].ToString()+
       ")"+"Linking date: "+ softwareLicense.SWLicenseTime;
writeToLog(result+"\t---Successful!");
```

Code Listing 2: Linking a Software License to a hardware asset after having looked up the specified software by name

5 Minor annoyances and how they were resolved

- The Manufacturer ID field can't be null for newly created assets, which is fine, except that the error message that is displayed when it's null is "Model version is a required field", which is somewhat ambiguous. So after some head scratching, we figured out what it was complaining about and started passing a Manufacturer ID to every call to `asset.addNew(...)`

- When assigning a value to a field that is a dropdown list, such as Asset Classification, its integer ID must be referenced in the assignment and not its name.

- We wondered for a while why we couldn't lookup objects using an SQL guard in `individual.lookupFirst(...)`; the error message that was returned didn't make it all that clear. So we came up with a workaround to search for contacts by last name, compare their staff ID to the one we had on hand and look up the next one if they don't match. Our egos took a toll when we later figured out that we need to have "WHERE" in the string that's passed as an argument for it to work.

- Some fields such as Manufacturer and CompanyBoughtFor, which are actually foreign keys in the database, are read only. So, to update, say, an asset's owner, what you have to do is look up the company's ID using its company name (asset.CompanyBoughtFor), then set the asset's CompanyBoughtForID field to the returned object's CompanyID field; something like this:

```
iconnect.Company company = oArgis.createCompany();
company.lookupFirst(ienums.CEnums.enLookupBy.eARGIS_LOOKUP_BY_FIELD,
ienums.CEnums.enLookupCompany.eARGIS_LOOKUP_COMPANY_BY_SQL_WHERE, "WHERE
company_name = '" + newOwner + "'", ref rows);
asset.CompanyBoughtForID = company.CompanyID;
```

Code Listing 3: Updating read-only fields

Appendices

A1 Records for PCs in AMS[4]

Asset.Asset Classification ≡ “IT Asset”
Asset.Asset Type ≡ “Hardware”
Asset.Class ≡ Type
Asset.Asset Tag No. ≡ Tag No[5]
Asset.Model Name.Model Name ≡ Model
Asset.Asset Serial Number ≡ Serial No
Asset.Subclass ≡ CPU Type
Asset.*Generic Component*[<CPU_Speed CPU Type>] ≡ CPU_Speed, CPU Type[6]
Asset.Model Name.Model Version.Available From Date ≡ Warranty_From[7]
Asset.Warranty Expiry Date ≡ Warranty_To
~~Monitor_Type~~
~~Monitor_Size~~
Asset.*Generic Component*[<RAM> RAM] ≡ RAM[8]
Asset.*Generic Component*[<HDD> HDD] ≡ HDD
Asset.*Generic Component*[<CD_ROM_Type>] ≡ CD_ROM_Type
Asset.Owner ≡ Asset Owner
Asset.Purchase Order ID ≡ Sanction_LPO_No
Asset.Quantity ≡ Quantity
Asset.Seller Company.Company Name ≡ Vendor Name
Asset.Host Name ≡ Host Name
Asset.DNS Name ≡ Domain_Name
Asset.Contact.Company.Company Name ≡ Emp Organization
Asset.Contact.Staff No. ≡ Owned_By_Emp_Id[9]
Asset.Contact.First Name ≡ Owned_By_Emp_Name[10]
Asset.Contact.Last Name ≡ Owned_By_Emp_Name[11]
Asset.Contact.Job Title ≡ employee_Title
Asset.Department ≡ Department[12]
Used By[13]
Asset.Contact.Telephone Number ≡ Tel_Ext
Asset.Floor Location ≡ Station, Building, Wing_Floor
Asset.Cabinet Location ≡ Location
Remarks[14]

---

[4] The UAPM API requires that every asset have a non-null manufacturer, so the code currently sets the manufacturer for all added assets to an arbitrarily selected one. The AMS data doesn’t have such a field for us to use so, if required, we’ll have to come up with one on our own and determine the mappings between assets and their respective manufacturers in the code.
[5] The field AltAssetID is used in the staging UAPM setup
[6] e.g. Asset.*Generic Component*[3.4 GHz Pentium-IV] ≡ 3.4 GHz Pentium-IV
[7] Not used, got the ok to ignore this
[8] e.g. Asset.*Generic Component*[512 MB RAM] ≡ 512 MB
[9] The rest of a contact’s fields are inherited from the model. In the staging UAPM setup, since models don’t exist, a new one is created if one can’t be found.
[10] All but Owned_By_Emp_Name’s last token
[11] Owned_By_Emp_Name’s last token
[12] To be ignored, as agreed upon
[13] To be added, can be ignored for now
[14] To be added, as before

A2 Records for peripherals in AMS

If peripherals don't already exist in UAPM as assets, they should be added using the mappings shown below first before associating them with PCs (i.e. assets of type "Hardware" and of class "Desktop") on the Attached_PC field.

Asset.Asset Classification ≡ "IT Asset"
Asset.Asset Type ≡ "Hardware"
Asset.Asset Class ≡ {"Printer", "Monitor", "External Storage", "External Modem", "Scanner"}
Asset.Asset Subclass ≡ {{"Bubblejet", "Deskjet", …}, …}[15]
Asset.Asset TAG No. ≡ Tag_No
Asset.Serial Number ≡ Serial No
Asset.Warranty Expiry Date ≡ Warranty_To
Asset.Floor Location ≡ Building, Wing_Floor
Asset.Cabinet Location ≡ Location_Cabin
Asset.Seller Company.Company Name ≡ Vendor Name
Asset.Contact.Job Title ≡ employee_Title
Asset.Contact.Staff No. ≡ Owned_By_Emp_No[16]
Asset.Contact.First Name ≡ Owned_By_Emp_Name[17]
Asset.Contact.Last Name ≡ Owned_By_Emp_Name[18]
Asset.Model Name.Model Version.Available From Date ≡ Warranty_From
Asset.Department ≡ dept_Descriptions[19]
Remarks[20]
Asset.Owner.Company Name ≡ Onwer_Name (sic)
Asset.Contact.Company.Company Name ≡ Emp Organization
Asset.Model Name.Model Name ≡ Peripheral Model

A3 Records for software applications in AMS

Asset.Asset Type ≡ "Software"
Asset.Asset Classification ≡ "IT Asset"
Asset.Class ≡ "Unknown"[21]
Asset.Asset Name ≡ Software_Name
Asset.Serial Number ≡ Sr_No
Asset.Seller Company.Company Name ≡ vendor name
Asset.Owner.Company Name ≡ Organization
Asset.Model Name.Model Version ≡ Version

[15] See any asset in UAPM on production server; missing in staging UAPM setup, so ignored
[16] The rest of a contact's fields are inherited from the model as before
[17] All but Owned_By_Emp_Name's last token (as before)
[18] Owned_By_Emp_Name's last token (as before)
[19] To be ignored, as agreed upon
[20] To be added, same as before
[21] To be used as the default value for now