Les Outils.

1)Scanners.

Une série de programmes analyseurs forment les outils d'analyse. Chacun traite un langage spécifique. L'objectif est de :

- décomposer un langage particulier ;
- reconstituer les éléments, qui auraient pu être modifiés au cours du processus précèdent..

1.1)Définition du langage.

Il est défini par un fichier qui comporte une série de paramètres, modifiable par l'utilisateur. Ces paramètres ont les trois aspects suivants :

- ils définissent les classifications du langage : la classification se fait sur trois niveaux, permettant une définition très fine, au cas échéant, par exemple, MOVE peut être classifié comme (RESERVED WORD, VERB) en COBOL, MVC comme OP CODE en Assembleur, etc. ;
- ils définissent chaque élément du langage, au moyen de ces classifications, qui permet d'éviter des modifications dans les outils pour pouvoir traiter des langages différents ;
- ils spécifient la routine externe (s'il y en a une) à appeler pour traiter un élément spécifique : les routines externes peuvent être appelées d'une manière ciblée (tous les MOVEs dans un programme COBOL) ou d'une manière plus générale (tous les verbes).

1.2)Les programmes de la machine source.

Il s'agit d'un seul programme source, ou une bibliothèque de sources, dans le format de la machine source - c'est à dire sans traduction vers les caractères de la machine cible.

1.3 Règles.

Une facilité facultative, permet une application rapide des aspects 'Scanners' aux sources.

Chaque article dans ce fichier est composé de deux parties :

- le format du langage source, employé pour les recherches dans toutes les sources ;
- la nouvelle structure

1.4 Routines utilisateur.

Il s'agit d'une bibliothèque de sous-programmes (routines externes) qui peuvent être appelés, à la demande de l'utilisateur, afin de modifier la sortie du processus.

IL existe une bibliothèque de routines externes qui répondent aux besoins déjà rencontrés sur d'autres migrations. Par exemple, la création d'un inventaire, l'insertion de code de capture de statistiques, etc. Ces routines sont compris dans les outils standards auxquels sont rajoutés tous ceux nécessaires pour un client spécifique.

Des utilisateurs peuvent écrire leurs propres routines pour des traitements particuliers, en faisant référence aux interfaces prévues.

1.5) Langage intermédiaire.

Ceci représente la sortie ultime du processus. Normalement, dans le cas d'une migration d'un langage vers un autre (COBOL VS vers COBOL Microfocus, par exemple), la sortie sera déjà du COBOL et on pourrait la considérer comme « terminée » (n'ayant pas besoin d'autre processus autre que quelques réglages ponctuels à la main).

Par contre, dans le cas où les environnements source et cible sont totalement différents (la conversion des maps CICS vers Microfocus, ou de PL1 vers COBOL ou d'un JCL vers un autre, par exemple) la sortie sera sous une forme exploitable par un processus de génération, vraisemblablement basé sur des macros GENIUS (se réfère aux Outils.).

1.6) Les rapports facultatifs.

Il est possible d'obtenir du processus un inventaire, une liste des programmes source traités et des messages issus des routines 'utilisateurs'.

2) GENIUS

L'outil GENIUS est basé sur un langage de macros avec comme objectif la génération de sources soit de n'importe quelle nature (JCL, COBOL, etc.). Le but est de générer des sources comme des fichiers de caractères - indépendant des besoins du langage cible, ceux-ci étant traités par des macros utilisateur.

2.1) Le langage Genius.

Le générateur de macros, Genius, a les éléments linguistiques suivants :

- les variables définies par l'utilisateur, soit localement (valable à l'intérieur d'un macro), soit globalement (disponible à toutes les macros dès sa définition ;
- la codification des attributions, qui permettent l'attribution de valeurs aux variables utilisateur ;
- la codification des conditions, permettant une génération variante selon les consignes de l'utilisateur ;
- des facilités d'assistance, qui montrent le procédure de génération telle qu'elle se de ;
- des fonctions standards pour la manipulation des chaînes de caractères de longueurs variable ;
- des fonctions utilisateur qui font adapter le langage de macros.

2.2) Source Genius.

Un source Genius a trois types de commandes :

- une commande Genius, qui sera traitée par l'outil et qui peut faire sortir dans le fichier en sortie ;
- des commandes du langage cible (JCL ou COBOL, par exemple), qui peuvent être modifiées dynamiquement, avant d'être écrites dans le fichier en sortie ;
- des appels aux macros.

Une commande du langage cible doit suivre les règles de ce langage (JCL ou COBOL, par exemple), sauf dans le cas d'un point dans la première position. Ce caractère est réservé pour les macros Genius.

2.3) La bibliothèque de macros.

La bibliothèque de macros est un nombre de macros Genius, chacun ayant un traitement particulier à effectuer. Un macro peut être considéré comme un sous-programme de Genius, qui peut être appelé plusieurs fois, le traitement étant conditionné par les valeurs des arguments qui lui sont fournis à chaque appel.

2.4) Les fonctions utilisateur.

Les utilisateurs peuvent adapter le langage en écrivant des programmes « normaux » et en se servant de l'interface standard fournie avec Genius. Ces programmes peuvent être appelés à partir du source Genius et peuvent rendre des valeurs à l'environnement Genius.

Par exemple, un utilisateur écrit un programme, SQRT, en COBOL, pour calculer la racine carrée d'un chiffre. On y accède par un source Genius de la façon suivante :

```
var  SET    SQRT(16)
```

Le sous-programme SQRT est appelé avec la valeur 16. Le sous-programme rend la valeur 4 qui est attribuée à la variable var.

Les utilisateurs ont besoin uniquement de mettre leurs sous-programmes dans la bibliothèque de fonctions pour pouvoir les référencer.

2.5) Le source généré.

Ce source, en images carte de 80 octets, est créé par les commandes Genius , soit dans le source lui-même, soit dans les macros.

2.6) Les rapports facultatifs.

Il existe deux types de rapports fondamentaux dans le système :

- des rapports d'erreurs et d'avertissements ;
- des rapports de trace, qui, à la demande de l'utilisateur, et a un niveau de son choix, démontrent le passage du processus. Il est prévu qu'un utilisateur sorte une telle trace pour le premier passage d'une nouvelle macro mais pas d'une manière régulière.

3) FILCON

FILCON est un outil de conversion de données qui :

- permet un libre échange de données entre es environnements source et cible, dans les deux sens, d'une manière acceptable par l'environnement récepteur ;
- fournit une méthode pour définir les traductions nécessaires ;
- vérifie le cohérence de toutes les données numériques pour avec l'image prévue de ces données, vu que selon l'environnement, le langage de traitement etc., certains cas d'utilisation arithmétique risque de rendre les données incompatibles (à la demande du client, les données de ce type peuvent être « réparées »).

Même si le besoin de la conversion de données est évidente, d'autres traitements doivent être effectués :

- le traitement et création des labels des deux environnements ;
- le blocage et déblocage des enregistrements logiques à l'intérieur d'un bloc ;
- l'alignement spécial nécessaire pour certains type de données (des octets 'slack' COBOL).

3.1) Les bibliothèques de 'clause copy' COBOL.

Afin de simplifier le travail de la définition du processus de conversion, on se sert des clause copie COBOL pour la spécification des images des enregistrements avant et après conversion.

Ces clauses copy ont le format et le contenu conformé à l'environnement source. Ils auront les même noms de champs, dans le même ordre, pour les clause copie source et cible. Seulement les PICTUREs peuvent être changés, car certains formats de données peuvent exister dans un environnement et pas dans l'autre.

Les 'clauses copy' restent inchangées à partir de l'environnement source, et servent pour la création des clauses copie cible. De cette façon, les octets 'slack' n'apparaissent pas au programmeur. Cependant, la définition de champs peut obliger à introduire ou retirer des octets 'slack', le cas échéant. Un tel repositionnement implicite est indiqué dans les rapports d'exécution.

Un type d'enregistrement, ou un champ, ou la redéfinition d'un champ peut être choisi selon les données dans l'enregistrement , d'une façon facilement lisible et compréhensible dans un style COBOL :

```
SELECT_RECORD PAYROLL-DETAILS
        IF      RECORD-TYPE EQUAL 'D'
        AND     RECORD-DELETED-FLAG NOT EQUAL 'Y'.
```

3.2) Les routines externes.

Les utilisateurs peuvent écrire des routines externes, appelées aux moments importants, qui peuvent modifier le traitement et modifier le contenu du fichier en sortie.

La routine appelée après la traduction, par exemple, peut accéder l'enregistrement par les clauses copie, peut supprimer des enregistrements, le modifier, etc. Des telles routines sont des sous-programmes

standards COBOL. Le contextes de la traduction, le sens, les compteurs de blocs et d'enregistrements, etc. sont rendus disponibles à ces routines.

N.B. Les routines externes ne fonctionnent que dans un seul sens. C'est à dire, les autres mécanismes de FILCON sont réversibles en échangeant les noms des bibliothèques de clause copie COBOL du source et de la cible, mais cette possibilité n'est pas valable pour les routines externes.

3.3) Les données.

Les données sont transformées, par un outil qui fonctionne de bande à bande - ceci étant le moyen le plus facile sur un site, et, en plus, il est le plus souvent indépendant du gestionnaire de catalogues. Les données portées ainsi peuvent être introduites sur le système cible, suivant les mécanismes standards de ce système récepteur.

3.4) Les rapports facultatifs.

Ces rapports, bien que facultatifs, jouent un rôle important dans le processus FILCON.

Au niveau (pas d'erreurs ni anomalies), il écrit les compteurs par type d'enregistrement le plus bas.

Pour chaque erreur détectée (une valeur qui dépasse sa taille maximale ou caractères non-numériques dans un champ numérique, par exemple), le rapport indiquera :

- le numéro de séquence logique de l'enregistrement en erreur ;
- le type d'enregistrement (le nom du groupe de niveau 01) ;
- le nom du champ en erreur ;
- la valeur du champ source et cible (dans le cas d'une « réparation »).

4) INVENTAIRE

L'inventaire est une base de données qui contient de l'information sur les objets à traiter dans le cadre d'un projet de migration.

La source principale de cette information est fournie par les analyseurs (ou plus exactement les routines externes). Un analyseur (Scanner), recherche quatre éléments dans chaque source traité :

4.1) Calls.

Ce sont des appels aux modules compilés indépendamment. Ce peut être des appels TP suivant le moniteur, les appels dynamiques dans le mesure ou ils peuvent être déchiffrés, etc..

4.2) Copies.

Cette classe contient toutes les commandes subsidiaires référencées par une compilation - tous les clauses copy, includes, macros trouvées dans le source.

4.3) Fichiers.

Les fichiers utilisés par le source, leur méthode d'accès, leur type d'accès (entrée, sortie ou mise à jour).

4.4) Particularités.

Les particularités du source, qui ne figurent pas dans le système cible ou nécessiteront de l'intervention manuelle.

Les données de la base sont accessibles, par les routines d'interface, aux outils eux-mêmes. Par exemple, la génération de JCL aura besoin de connaître, en plus des fichiers référencés par le source lui-même, les fichiers référencés par ses sous-programmes et/ou des sous-programmes de ces sous-programmes.

L'inventaire est essentiellement un moyen de contrôle et planification, et s'effectue normalement en deux étapes au début du projet.

L'inventaire de contrôle, qui se concentre sur les CALLS et COPY est fait pour s'assurer :

- de l'homogénéité - pour vérifier que tous les programmes principaux référencés par les JCLs et les tables TP sont présents dans l'inventaire ;
- de la cohérence - pour vérifier que toutes les modules appelées par les programmes principaux, toutes les clauses copie etc. sont présents dans l'inventaire, et idem pour les modules appelées par les modules etc. ;
- de la redondance - chaque objet dans la base doit être référencé par autre chose en considérant que les JCLs et les tables TP font des appels du plus haut niveau ;
- de l'uniformité - pour vérifier qu'il n'existe qu'une seule version de chaque objet.

L'inventaire de planification est basé sur la population définie par l'inventaire de contrôle, cette fois en prenant en compte tous les détails, y compris les particularités qui pèsent sur la migration.

Cet inventaire donne une base pour l'estimation du travail et le délai nécessaire, car :

- le nombre de source à traiter est connu ;
- les difficultés à traiter sont connus ;

En tant de documents de contrôle, des rapports sont disponible en exécutant des outils spécifiques à la capture de données à partir de l'inventaire. Ces rapports peuvent :

- définir un unité de conversion (un lot) étant donné une liste de programmes ;
- sortir un organigramme d'un programme indiquant tous les CALLS et COPIES ;
- sortir un organigramme d'une application, basé sur le passage de fichiers.

5) METERS

Le système Meters est une méthode par laquelle on valide l'exhaustivité d'un test, en insérant des commandes additionnelles, transparentes au programme sous test. Le système est mis en place pour la plupart par des routines externes d'un analyseur.

Un chemin est une série de commandes, exécutées l'une après l'autre, se terminant par une commande de :

- débranchement - GO, EXIT, CALL etc. ;
- condition - IF, ELSE, etc. ;
- impérative - AT END, INVALID KEY, etc.
- terminaison - STOP, EXIT PROGRAM, GOBACK.

Une longue série de MOVEs sera considérée comme un seul chemin, car toutes les commandes MOVE doivent être exécutées.

Chaque procédure peut être considérée comme un nombre de chemins, exécutés dans une séquence déterminé en fonction des données.

L'a mesure d'une procédure d'un langage du haut niveau (COBOL, par exemple) se base sur la reconnaissance de ces chemins individuels et sur l'insertion de commandes pour capturer le nombre de fois qu'une commande est exécutée.

Les compteurs des chemins sont utilisés dans trois manières différentes :

5.1) Passage.

Si le compteur indiquant le nombre de fois que le chemin a été exécuté est supérieur à zéro, il est évident que le chemin a été exécuté. En connaissant le nombre de chemins distincts le pourcentage des chemins exécutés peut être calculer. Un faible pourcentage indiquerait que les données de test n'ont pas été suffisant et ont besoin d'enrichissement.

5.2) Compteurs.

Les compteurs obtenus dans l'environnement sources peuvent être comparés avec les compteurs obtenus dans celui de la cible, pour avoir une preuve supplémentaire de la migration.

5.3) Les chemins morts.

Si un compteur est toujours zéro cela peut indiquer que le chemin est un traitement d'exception, exécuté uniquement dans le cas d'une erreur, ou que le chemin est mort, et ne sera jamais exécuté. Apres des années de maintenance, on peut arriver à cette dernière situation dans les cas où le chemin est dépassé, est obsolète, etc.

Meters a trois phases séparées :

- l'installation lui-même - l'insertion de commandes, par les routines externes d'un analyseur ;

- la capture des données issues de l'instrumentation, dans les deux environnements, par moyen un sous programme indépendant SPMETERS, dans un fichier indexé METERS ;
- l'édition des données par un programme spécifique au langage en question.

Le source.

Ceci est le source, après le traitement de la routine externe de l'analyseur. Il est la base du mécanisme du rapport, car il contient le numéro de chaque chemin.

Le fichier METERS.

C'est le fichier créé par les commandes insérées pendant l'exécution du programme. Deux versions de ce fichier sont possibles :

- la version cible, créée par la version cible du programme ;
- la version source (facultative), créée dans l'environnement source ou par une exécution antérieure.

5.4) Des rapports.

Le système peut fournir trois rapports différents :

- Sommaire - donne le pourcentage des chemins exécutés par le programme, et tous les sous-programme sujet de l'instrumentation ;
- Condensé - qui donne le compteur pour chaque chemin ;
- Complet - qui donne une listing de chaque source sujet de l'instrumentation, indiquant le nombre fois chaque chemin a été exécuté.

Si le fichier METERS est disponible dans les deux environnements, les compteurs des deux environnements sont montrés et des différences éventuelles entre les valeurs sont signalées.

6) JCLGEN

Le JCL de l'environnement cible est issu d'un outil de génération JclGen, basé sur l'outil Genius.

Au moins deux JCLs sera générés, pour deux objectifs divers et visant deux niveaux de connaissance différents.

6.1) JCL de test.

Le JCL de test est généré pour trois parties du processus

- la préparation des tests, où les fichiers de test sont reconstitués ou rechargés, dans un environnement de test, où les noms de fichiers se ressemblent à ceux de l'environnement ultime de production ;
- les programmes convertis sont exécutés (avec ou sans METERS) créant des fichiers en sortie ;
- les fichiers en sortie sont comparés avec ceux créés par les mêmes programmes dans l'environnement source.

6.2) JCL de production.

Le JCL de production est généré uniquement pour la production dans l'environnement cible. A ce stade du projet, la connaissance du personnel de JCL ne sera pas suffisante et le but est de générer JCL fonctionnel, facilement compréhensif en incorporant les nouvelles normes, dans une structure connue par le personnel.

6.3) JCL stabilisé.

Une troisième génération, beaucoup plus rare, est envisageable, quelques mois après le bascule vers l'environnement cible. Le but est de supprimer les traces introduites en liant le nouveau JCL à l'ancien.

La génération de JCL est basé sur trois entrées :

- les appels aux macros créés par les détails du programme à exécuter tirés on la base de conversion ;
- les normes de nomenclature des fichiers, définis au début de la migration (pour les noms symboliques et réels) ;
- une structures, tirées du JCL source pour le programme ou la série de programmes.

Pour la plupart, les appels aux macros et les normes restent inchangés travers les générations. Les changements demandés sont issus des changements des macros eux-mêmes, qui assure un traitement uniforme pendant toute l'opération de génération de JCL.

Quelques Références d'utilisation des outils de migration.

AEROSPATIALE

USINOR SACILOR

AGB

ANEP (GENUIS)

DISTRIPHOR

CAISSE D'EPARGNE DE RENNES(FILCON)

CNRS

AGB

RHENALU(GENUIS)

ROUSSEL-UCLAF