Le Projet Y[1]

RJ Chevance
Bull SA
rjc@chevance.com

Résumé

Le projet de gamme Y a été conduit par CII (Compagnie Internationale pour l'Informatique) entre 1972 et 1975. Cette gamme était destinée à remplacer à partir de 1981, au sein d'UNIDATA, les systèmes de nom de code gamme X.
La fusion de CII et de HONEYWELL-BULL en 1975 et la nécessaire convergence des gammes de produits entraîna l'arrêt de ce projet.
Le rationnel de l'architecture et ses caractéristiques majeures sont présentées. La gamme Y était fondée sur un adressage par capacités avec un niveau de mémoire unique (one level store). L'adressage utilisait des adresses virtuelles de 64 bits et l'interface de programmation était fondé sur le concept de machine langage.

1 - Introduction

Après avoir développé ses propres systèmes avec notamment IRIS 50/60, IRIS 80 et les mini-ordinateurs MITRA, CII (Compagnie Internationale pour l'Informatique) rechercha des alliances internationales au début des années 1970. Après une première alliance infructueuse avec ICL et CDC (Multinational Data), CII forma UNIDATA avec PHILIPS et SIEMENS.
En termes d'architecture de système, la convergence se fit autour de l'architecture Siemens. Cette architecture était très voisine de l'architecture IBM System/370 et provenait d'une licence RCA. Cette alliance se concrétisait par le développement en commun d'une gamme de produits compatibles : la gamme X avec des modèles allant de X0 à X4. La plupart de ces modèles ont été commercialisés.

Dès 1972, avant même qu'aucun des nouveaux systèmes issus de la coopération entre les trois Compagnies n'ait été commercialisé, le besoin d'étudier la génération suivante se faisait jour. Ainsi le projet d'une future ligne de produits, de nom de code Gamme Y, a été développé à la CII de 1972 à 1975.
Cette étude fût menée de façon indépendante à l'origine puis, à partir de la seconde moitié de 1974, partagée avec les partenaires. La commercialisation des premiers systèmes de la Gamme Y était envisagée pour le début des années 80.

L'architecture de la gamme Y proprement dite n'ayant jamais fait l'objet d'une publication, il m'a semblé utile de la décrire ainsi que d'évoquer les acteurs de cette étude.

Cet article présente en particulier la problématique telle qu'elle était perçue à ce moment ainsi que le rationnel des choix majeurs.

2 - Contexte de l'étude

Au sein de la CII, parallèlement à la poursuite des développements tant sur la future gamme X d'UNIDATA que sur les gammes existantes, un petit groupe de réflexion sur la gamme future, naturellement baptisée gamme Y, fût créé dès la fin de 1972.

Il convient de mentionner, qu'au début de cette étude, la presse spécialisée se faisait l'écho du développement d'une nouvelle gamme de systèmes d'IBM : la FS (Future Series). Très peu de choses

[1] à paraître dans les actes du 4ième Colloque « Histoire de l'Informatique » Rennes Novembre 1995

concrètes filtraient sur cette nouvelle gamme mais il se disait qu 'elle se caractériserait par l'abandon de la stricte compatibilité avec la gamme 370.

Les besoins exprimés pour cette future génération de systèmes étaient, schématiquement, les suivants :

- protection et sécurité accrues ;
- séparation des mécanismes et de la stratégie ;
- minimisation de la complexité de la programmation des systèmes ;
- longévité de l'architecture ;
- minimisation des dépendances vit à vis de l'architecture des machines (c'est à dire portabilité du logiciel) ;
- architecture multiprocesseur symétrique ;
- capacité à gérer de grands volumes de données.

On avait constaté que la difficulté à faire d'évoluer les systèmes tenait, en grande partie, au fait que leur réalisation ne faisait pas suffisamment la distinction entre les mécanismes de base et l'application de ces mécanismes à l'implémentation d'une stratégie particulière. Par exemple, pour l'édition de liens, l'un des mécanismes de base est la résolution d'une référence à un objet (désigné de façon symbolique) dans un contexte déterminé. Pour la plupart des systèmes, cette résolution ne pouvait se faire que statiquement (i.e. avant l'exécution du programme) par l'éditeur de liens qui était -de ce fait- un point de passage obligé. Ainsi, on se privait de la souplesse de l'édition dynamique de liens (cf. MULTICS) et le traitement de l'appel de procédure dynamique de COBOL conduisait à une réalisation assez complexe et peu élégante. En revanche, en considérant que les mécanismes de base étaient la résolution d'une référence symbolique ainsi que l'invalidation d'une référence résolue, on disposait de la panoplie nécessaire à l'implémentation de différentes stratégies d'édition de liens.
Ce constat nous a conduit à exploiter, de façon systématique, le principe de séparation des mécanismes et de la stratégie.
En ce qui concerne les difficultés liées au développement du logiciel, il convient de rappeler que le logiciel système (i.e. système d'exploitation proprement dit, compilateurs, gestionnaires de données, protocoles de communication,....) était le plus souvent programmé en langage d'assemblage. Le système MULTICS [ORG72] qui était réalisé dans un sous-ensemble du langage PL/1, constituait une exception à cette époque. Toutefois, certaines expériences avaient été réalisées autour des systèmes de la CII. Ainsi, le système SIRIS 7 était programmé dans un langage de programmation structuré implémenté au moyen d'un méta-assembleur. Deux systèmes d'exploitation expérimentaux de type temps partagé, pour la machine 10070, avaient été développés à la fin des années 1960 dans un langage inspiré du PL360 (syntaxe de type Algol mais relativement proche de l'architecture de la machine) : l'un par l'IRIA (devenu INRIA depuis) : le système ESOPE et l'autre par le Centre de recherche de la CII : le système SAM (Système à Accès Multiple). Un langage de programmation ciblé vers le développement du logiciel système, le LPS (Langage de Programmation Système), a été créé à la CII au début des années 1970. Ces quelques faits montrent que l'on était prêt à franchir le pas pour le développement du logiciel système en langage de haut niveau. On doit aussi noter la connexion de ces études avec les travaux conduits par l 'équipe de Jean ICHBIAH sur le langage LIS (Langage d'Implémentation de Software) et qui allait donner naissance, plus tard, à ADA.
Le besoin de systèmes fiables était l'une des demandes majeures. Les causes des défaillances étaient, pour partie, liées à la complexité du logiciel système et la relative rusticité des mécanismes de protection de certaines architectures. L'intégration, au niveau de l'architecture, de mécanismes de protection élaborés semblait alors nécessaire.

La longévité est une donnée constante au niveau de la conception de nouvelles architectures. Le coût de développement d'une nouvelle architecture ne pouvait, dès cette période, se justifier qu'en fonction de la période durant laquelle l'architecture permettait de soutenir la compétition. Bien évidemment, l'indépendance de la programmation système vis à vis de la structure des machines est un facteur qui contribue à la longévité de l'architecture.

L'architecture de type multiprocesseur symétrique avait été déjà exploitée au niveau de l'IRIS 80 avec le système d'exploitation SIRIS 8 et poursuivie dans le cadre de la gamme X. Ce type d'architecture, qui

permet une croissance aisée de la performance des systèmes par simple ajout de processeurs, était donc l'une des hypothèses de base du projet.

L'un des besoins essentiels des entreprises en matière de systèmes d'information était (et est toujours) la possibilité de disposer «en ligne » de toute l'information nécessaire (i.e. sans recourir à tous les mouvements de bandes magnétiques qui étaient à la fois source de coût et d'erreur). Sans trop préjuger de la solution au niveau du matériel (notons que les premiers dispositifs de stockage avec robots d'accès faisaient leur apparition e.g. 3850 d'IBM), la possibilité d'adresser l'ensemble des données directement était un objectif de l'étude. Là encore, le système MULTICS avait montré la voie avec la notion de fichier « couplé » c'est à dire de fichier connecté à l'espace d'adressage d'un processus et qui peut donc être directement accédé par les programmes. De cette façon, il n'y a plus d'entrées-sorties explicites ; les mouvements de données entre les dispositifs de stockage externes et la mémoire étant tout simplement pris en compte par le dispositif de pagination à la demande.

Il convient de souligner la prédominance du logiciel au niveau du démarrage de l'étude. Alors que l'approche qui prévalait en général à cette époque était une claire séparation des rôles des équipes de développement matériel et logiciel autour de l'interface de programmation (définie par ce que l'on appelait le « manuel de référence ») avec peu d'interaction entre les équipes et bien souvent une prédominance du matériel sur le logiciel (il convient de rappeler que les possibilités de la technologie étaient limitées), la priorité a été clairement donnée au logiciel lors du démarrage des études de la gamme Y.

3 - Description de l'architecture de la gamme Y

3 - 1 - Genèse de l'architecture Y

Si la compatibilité binaire -au moins au niveau des applications- était une contrainte de l'étude, une certaine liberté était accordée -au moins au niveau de l'étude- en matière d'architecture.

En matière d'architecture, deux approches étaient possibles :

- machine virtuelle permettant de restituer plusieurs machines ayant le même répertoire d'instructions (avec par exemple des systèmes d'exploitation différents) au-dessus d'une seule et même machine physique ;
- machine abstraite fondée sur un niveau de virtualisation plus élevé permettant de supporter différents environnements d'exécution.

Le concept de machine virtuelle avait été initialement exploré par IBM avec CP/CMS développé sur la machine 360/67. Ce projet de recherche a ensuite donné le jour au produit VM. CII, pour sa part, avait développé un système comparable dans le cadre de ses activités de recherche au sein de l'unité de recherche CII/IMAG à Grenoble : le projet Gémeaux/SPS développé sur IRIS 80 qui permettait de supporter le concept de machine virtuelle.

L'intérêt de cette approche est de pouvoir supporter, sur une même machine, plusieurs environnements simultanément, chacun de ces environnements disposant d'une machine virtuelle réalisée sur la machine « physique » au moyen d'un hyperviseur. La difficulté majeure avec ce type d'approche est qu'elle ne permet pas de s'affranchir facilement des contraintes de l'architecture de base et plus particulièrement des capacités d'adressage de la machine (e.g. 2**24 octets).

Ne souffrant pas des même contraintes, l'approche machine abstraite fût donc retenue. Le système MULTICS fût, comme pour beaucoup des architectures de système créées à ce moment, le point de départ.
Le concept MULTICS de fichier connecté à l'espace d'adressage des processus conduisit au concept d'un espace unique d'adressage et à un seul niveau (one level store). L'hypothèse de base de la gamme Y était un espace d'adressage unique de 64 bits (2**64 octets) : l'ensemble des objets que le système pouvait manipuler appartenait à un espace d'adresses -représentées sur 64 bits- potentiellement accessible à tout processus. Le lieu de résidence des objets (permanents) était le dernier niveau de la hiérarchie de mémoire. Le mécanisme de pagination à la demande était généralisé : il remplaçait les entrées-sorties. Notons que l'espace d'adressage n'était pas étendu au réseau.

Pour faciliter le développement et la gestion des versions des logiciels, le concept d'édition dynamique de liens popularisé par le système MULTICS avait été retenu et étendu.

Une première architecture directement dérivée de MULTICS fût étudiée. Cette première architecture retenait la même structuration de l'espace d'adressage que MULTICS avec un espace commun à l'ensemble des processus et un espace privé par processus ainsi que le mécanisme de protection fondé sur les anneaux.
Des difficultés liées à la structuration de l'espace d'adressage et au mécanisme de protection nous ont conduit assez rapidement à nous orienter vers une architecture fondée sur les capacités (capability-based addressing) [FAB74]. Ce type d'architecture, assez populaire alors dans le monde de la recherche, n'avait pas encore connu d'exploitation industrielle. Il convient toutefois de mentionner, bien que nous l'ignorions à l'époque, que le système 6000 d'HONEYWELL (dont les successeurs sont actuellement commercialisés par BULL sous le nom de DPS9000) intégrait dans sa définition, à partir de 1973, une notion de capacité (architecture dite NSA New System Architecture) qui ne fût exploitée qu'à partir des années 1980.

3 - 2 - Principes

L'architecture de la Gamme Y se caractérisait par les éléments suivants :

- un espace unique d'adressage virtuel et à un seul niveau de 64 bits. Espace d'adressage unique signifie que l'adresse d'un objet est la même quel que soit le processus qui l'adresse. A un seul niveau signifie que les objets sont adressés comme de la mémoire quel que soit leur lieu de résidence ;
- un espace structuré en termes d'objets indépendants les « Data Sets » regroupés en volumes logiques. Chacun des 2**32 data sets pouvait contenir 2**32 octets. Les data sets étaient potentiellement accessibles par tous les processus ;
- un data set était un conteneur de segments (ou windows dans la terminologie Y) qui étaient définis sur le data set par une origine et une longueur. La taille maximale d'un segment était de 2**32 octets. La notion de segment était gérée par le logiciel ;
- les objets (de quelque nature qu'ils soient) étaient désignés par des noms symboliques. Cet espace de désignation n'avait pas d'organisation fixée *a priori*, un mécanisme général permettait d'implémenter différentes organisations de cet espace de noms (séparation des mécanismes et de la stratégie). Ainsi, l'organisation de cet espace sous forme de répertoires hiérarchisés (cf. MULTICS) n'était qu'un cas particulier ;
- un mécanisme d'adressage et de protection fondé sur les capacités ;
- l'utilisation du principe de partition pour la protection des capacités ;
- une architecture à deux niveaux avec une architecture de base et une notion d'architecture virtuelle (voir ci-après). Cette partie de l'étude était supportée par le contrat DRME 74/450 ;
- un répertoire d'instruction « natif » fondé sur une architecture de type « machine-langage » ;
- une architecture du logiciel de base fondée sur ce que l'on conviendrait aujourd'hui d'appeler une approche objet. Le logiciel de base était structuré en types d'objets et d'opérations sur ces objets.

Les sous-paragraphes ci-après décrivent les traits les plus caractéristiques et les plus originaux de l'architecture.

3 - 3 - Adressage par capacités

Une capacité était représentée sur 16 octets :

- 4 octets décrivant le type de l'objet désigné (pointeur, windows ou C-list), l'état de la capacité et les droits d'opération ;
- 4 octets identifiant le volume logique et le data set ;
- 4 octets indiquant l'origine de l'objet désigné dans le data set ;
- 4 octets indiquant la longueur de l'objet ou un déplacement identifiant un pointeur.

Il est essentiel, dans une architecture fondée sur les capacités, de protéger les capacités contre toute manipulation intempestive. Deux approches étaient possibles : partition ou préfixes.
Le principe de partition s'oppose au concept de préfixe ou «tag ». Un préfixe n'est rien d'autre qu'un descripteur (un petit nombre de bits) associé à chacun des éléments de la mémoire et qui est interprété uniquement par le matériel. De cette façon, le matériel peut vérifier la validité des opérations (i.e. prévenir les modifications intempestives des capacités) ou bien donner dynamiquement une interprétation de la signification des instructions en fonction des préfixes associés aux opérandes. Les machines BURROUGHS B5000 et B6500/7500 constituent une excellente illustration de l'architecture à préfixe (ou tag), les préfixes étant dans ce cas utilisés pour décrire le type des données (e.g. entier, flottant simple ou double, pointeur, voire même appel de fonction,....).

Avec le principe de partition, Il y a deux classes de segments : segments de code ou de données d'une part et segments de capacités (ou C-lists) d'autre part. La manipulation des capacités requiert donc de disposer de droits particuliers sur des C-lists (i.e. d'avoir une capacité appropriée pour accéder à l'objet). Le concept de tag avait été écarté car il conduisait à une trop grande complexité du matériel.

Ainsi, la possibilité de créer ou de modifier des capacités était matérialisée par la possession d'une capacité avec des droits appropriés sur un objet de type C-list.

La figure 1 illustre l'adressage par capacités.

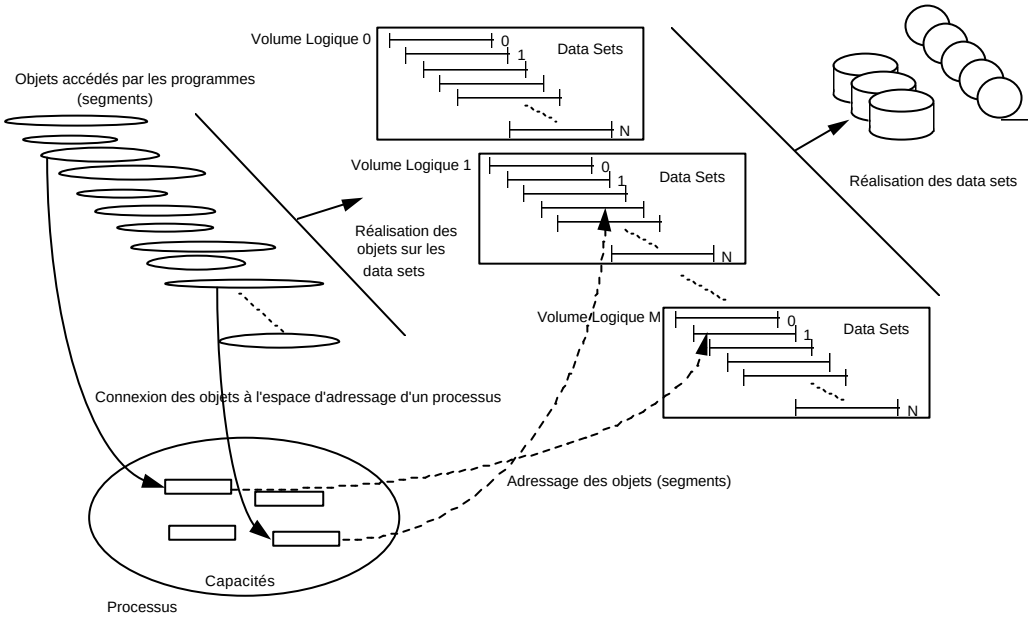

Figure 1 - Adressage par capacités

Ce schéma montre les différents niveaux d'abstraction de l'adressage : réalisation des data sets sur les supports physiques (derniers niveaux de la hiérarchie de mémoire), réalisation des objets de la programmation (segments) sur les data sets, connexion des segments à l'espace d'adressage des processus au moyen des capacités et accès (protégé) aux segments par l'intermédiaire des capacités.
Dans un but de simplification, on n'a pas représenté l'espace de désignation symbolique des objets de la programmation.

3 - 4 - Gestion de l'espace d'adressage

Comme on l'a indiqué, la transformation des noms symboliques d'objets en adresses pouvait s'opérer au moyen d'une hiérarchie de désignation inspirée de MULTICS (que l'on retrouve d'ailleurs dans la plupart des systèmes contemporains). Rappelons que les mécanismes de base permettaient d'implémenter d'autres modes d'organisation des noms symboliques. Une telle transformation permettait donc d'obtenir une capacité générique (contenant la description de la réalisation du segment sur un data set) ainsi que

les informations de protection. A partir de ces informations, une capacité pouvait être crée pour le processus demandant l'accès à l'objet.
Si ce mécanisme ne présentait pas de caractère original, en revanche la mise en œuvre des capacités mérite d'être évoquée.
Dans les adressages à capacités décrits dans la littérature scientifique, il était classique d'utiliser une indirection pour accéder à la réalisation de l'objet (e.g. son adresse logique ou physique). Cette indirection utilisait le nom de l'objet figurant dans la capacité (dans le cas de Y, cela eut été le champ volume logique/data set) comme index dans une table globale généralement appelée Master Object Table. Cette approche conduit à une inefficacité certaine et à une gestion spécifique de cette table globale. Pour contourner cette difficulté, nous avions retenu de faire appartenir cette table à l'espace d'adressage proprement dit en spécialisant des data sets. Ainsi, le data set 0 du volume logique 0 permet d'accéder à la description de la réalisation des différents volumes logiques et le data set 0 de chacun des autres volumes logiques permet d'accéder à la description de la réalisation des data sets du volume logique concerné. De cette façon, l'accès aux tables décrivant la réalisation des data sets sur le dernier niveau de la hiérarchie de mémoire n'était pas différent de l'accès à n'importe quel ensemble de données (pour assurer la protection, des capacités sur de tels data sets étaient bien sûr nécessaire).
Il convient de noter que seule la description de la réalisation sur le dernier niveau de mémoire était nécessaire au fonctionnement du système, tous les autres niveaux de mémoire n'étant que des caches qui n'avaient d'autre besoin que la description de leur contenu (i.e. des fragments d'objet qu'ils contiennent). En fait, la hiérarchie de mémoire était gérée en termes de volumes logiques, les notions de segments et de data sets n'étant visibles que des processeurs au moment de la résolution de l'adressage et des tests de protection. Ainsi les adresses qui étaient présentées à la hiérarchie de mémoire étaient des adresses logiques du type : <volume logique, déplacement>. De cette façon, la taille des granules (élément échangé entre deux niveaux de la hiérarchie) pouvait être déterminée en fonction des caractéristiques physiques des niveaux concernés et l'on faisait l'économie d'une transformation complète d'adresse au niveau des processeurs.
Au niveau d'un processus, l'espace d'adressage était matérialisé par l'ensemble des capacités auxquelles le processus avait accès.

3 - 5 - Communications

Deux systèmes de communication pouvaient être utilisés entre processus : communication au moyen des objets de l'adressage ou bien un système de boîte à lettres avec des ports de communication.
Dans le cas de la communication au moyen de l'adressage, les processus devaient connecter des objets de l'adressage (des windows) dans leur espace d'adressage, c'est à dire avoir des capacités pour ces objets.
Les boîtes à lettres et les ports étaient utilisés pour la communication avec l'extérieur du système aussi bien que pour les communications entre processus.

3 - 6 - Répertoire d'instructions

Comme on l'a indiqué ci-dessus, l'architecture comprenait -en application du principe de séparation des mécanismes et de la stratégie- deux niveaux : architecture de base et architecture virtuelle.
L'architecture de base, donnée stable de l'architecture Y, définit les structures essentielles : notion de processus et objets associés (e.g. pile de données et pile de capacités), registres de base pour l'adressage des objets et un jeu d'instruction limité à la manipulation de ces structures (e.g. chargement d'un registre de base avec une capacité, appel ou retour de procédure).

Au-dessus de cette architecture de base, différentes architectures -dites architectures virtuelles- visibles au niveau de la programmation pouvaient être définies. Ces architectures virtuelles étaient supportées par l'architecture de base. Ainsi, la notion classique de répertoire d'instruction correspondait à une architecture virtuelle. Il convient de noter que l'approche « machine virtuelle » consistait à engendrer des machines virtuelles au-dessus d'une machine physique et où toutes ces machines avaient la même architecture. En revanche, l'approche prise dans la gamme Y permettait de supporter des architectures différentes au-dessus d'une même architecture de base. De cette façon, on assurait la compatibilité avec les gammes existantes sans toutefois se « bloquer » sur les contraintes (en particulier la limitation de la taille de l'espace d'adressage) de ces architectures.

Suite à une première étude sur une machine langage : la machine COBOL [CHE73], une étude plus générale fût entreprise pour l'architecture virtuelle «native » de la gamme Y. Les objectifs de cette architecture -dite machine HLL (High Level Language) [BAT79]- étaient de supporter une grande variété de langages de programmation tout en assurant un degré de protection élevé. Les langages COBOL, FORTRAN et PL/1 furent retenus. Notons que si le succès de PL/1 était alors loin d'être assuré, ce langage rassemblait les caractéristiques de beaucoup de langages et aussi la plupart de leurs difficultés d'implémentation. On pouvait donc raisonnablement penser qu'une architecture permettant un support efficace de PL/1 était susceptible de supporter la plupart des autres langages. On entreprit donc de réaliser des mesures statiques et dynamiques de programmes COBOL et PL./1 au niveau « source » i.e. mesures exprimées en terme des constructions du langage source et donc indépendantes de toute implémentation. On disposait déjà des mesures de programmes FORTRAN publiées par D.E. KNUTH. Les mesures statiques de programmes consistaient à enregistrer les caractéristiques des programmes lors de la compilation (e.g. occurrence types d'ordres, des types des opérandes,.....). A l'aide des résultats de ces mesures, on pouvait optimiser le répertoire d'instructions en fonction de l'encombrement des programmes compilés. Les mesures dynamiques permettaient d'optimiser le répertoire d'instructions en fonction de la rapidité d'exécution. Ces mesures ont été réalisées par modification du compilateur COBOL de l'IRIS 50 et du compilateur PL/1 de l'IRIS 80.
Les résultats des mesures COBOL ont été publiés [CHE78]. Les résultats des mesures PL/1 n'ont pas fait l'objet d'une publication mais ils ont permis toutefois de constater que les profils des programmes de gestion étaient assez voisins de ceux relevés avec COBOL et que les programmes «scientifiques » étaient voisins de ceux relevés avec FORTRAN confirmant ainsi une relative indépendance vis du langage de programmation.
Une attention toute particulière avait été accordée aux problèmes de protection. Avec PL/1, comme dans les langages autorisant la manipulation des pointeurs, le risque de références pendantes ou dangling references -c'est à dire la possibilité d'utiliser des pointeurs sur des objets détruits- existe. Pour résoudre ce problème des structures particulières furent introduites au niveau de l'architecture virtuelle [BAT78], en d'autres termes on ne se servait pas des capacités pour résoudre ce problème en évitant ainsi la consommation d'une ressource précieuse : l'espace d'adressage unique et ses adresses universelles.

La figure 2 illustre l'articulation entre la machine HLL et l'architecture de base.

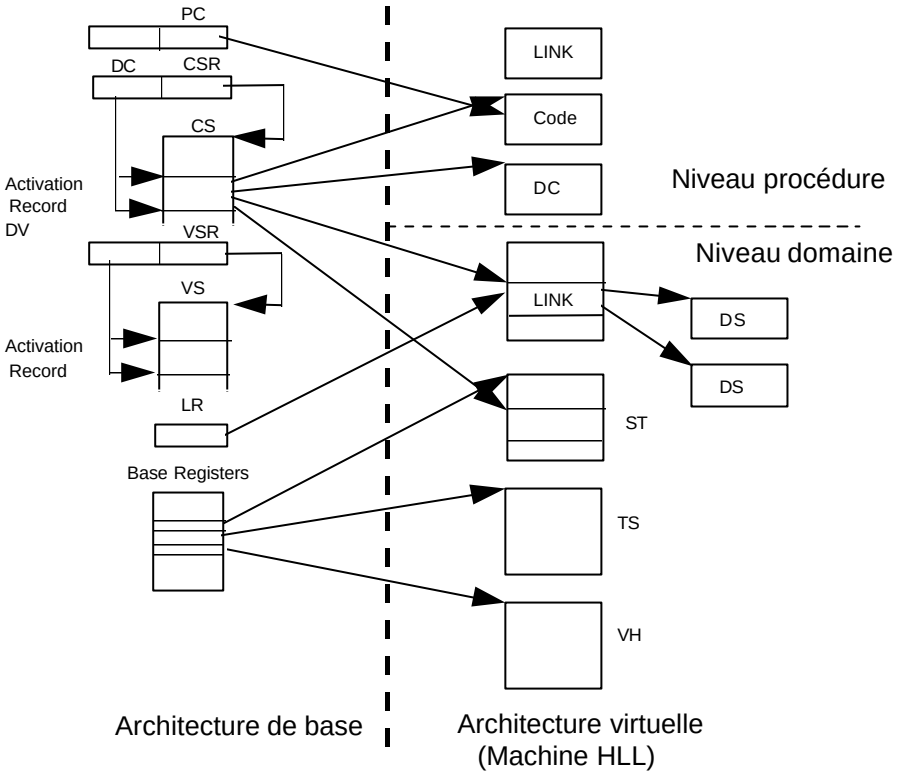

Figure 2 - Architecture de base et Machine HLL

Ce schéma met en évidence la partie architecture de base et la partie architecture virtuelle. Dans la partie architecture de base, on trouve les éléments suivants :

- le compteur ordinal (Program Counter PC) identifiant l'instruction en cours d'exécution dans le segment de code associé à la procédure ;
- la pile d'enchaînement de capacités (Capability Stack CS) identifiée par un registre spécifique (Capability Stack Register CSR) et dont la partie de la pile associée à l'activation de la procédure (Activation Record) est identifiée par un champ de ce registre (Display Capabilities DC). C'est à partir de cette partie de la pile que sont identifiés les différents segments auxquels le processus a accès ;
- la pile d'enchaînement de données ou valeurs (Value Stack VS) identifiée par un registre spécifique (Value Stack Register VSR) et dont la partie de la pile associée à l'activation de la procédure (Activation Record) est identifiée par un champ de ce registre (Display Values DV) ;
- le segment contenant les capacités vers les objets auxquels la procédure en cours d'exécution à accès. Ce segment est identifié par un dispositif accélérateur : le registre LR (LINK Register);
- un ensemble de registres de base permettant d'accéder rapidement aux différents objets manipulés par la procédure.

Tous ces registres ont le format des capacités.

Dans la partie architecture virtuelle, on trouve une partie procédure et une partie dite domaine. La partie procédure est composée de trois segments : un segment de code qui contient les instructions, un segment contenant les descripteurs et les constantes (Descriptors&Constants DC) et une définition générique des objets auxquels la procédure peut faire référence (LINK). La partie domaine correspond au contexte d'exécution du processus, on y trouve les éléments suivants :

- la définition des objets auxquels le processus fait ou peut faire référence. Au sein de ce segment, une zone LINK (en fait un segment) identifiée par le LINK Register contient les capacités vers les objets accessibles par la procédure en cours d'exécution. Cette zone LINK

est en fait une instanciation du segment LINK de la procédure dans le domaine du processus. Le segment, au niveau du domaine, contient donc les capacités vers les objets accessibles par le processus. Ces capacités peuvent être ou non résolues. Ce segment peut être construit soit statiquement (i.e. les liaisons vers les objets sont construites indépendamment de toute exécution du processus - c'est l'édition de liens statique) soit dynamiquement (i.e. les liaisons sont construites dynamiquement en fonction des références aux objets - c'est l'édition de liens dynamique et diverses politiques d'édition dynamique de liens peuvent être définies) ;
- des zones de données (Data Sets symbolisés par DS sur le schéma) peuvent être accédées au moyen de cette zone LINK, c'est en particulier le cas pour les fichiers. Notons que de tels objets peuvent aussi être accédés au moyen de capacités passées en paramètre dans la pile d'enchaînement de capacités (CS) ;
- un segment de données statiques (ST). Ces données appartiennent au processus et ont pour durée de vie l'exécution du processus. C'est par exemple le cas des données du FORTRAN et du COBOL ;
- un segment de données dynamiques (Value Heap VH) dont la durée de vie est contrôlée par certaines instructions du programme (e.g. ALLOCATE et FREE du PL/1). Notons que les variables crées automatiquement par l'activation d'une procédure ou d'un bloc appartiennent à la pile d'enchaînement de données VS ;
- un segment servant au traitement des références pendantes (pointeurs sur des objets ayant été détruits). Ce segment est appelé TS pour Tombstone. Lorsque qu'un objet -susceptible d'être référencé après sa destruction- est créé, une cellule dans ce segment est créée et toutes les références transiteront par cette cellule. Lors de la destruction de l'objet, la cellule est invalidée et les références à l'objet ne pourront donc plus être satisfaites. Ce mécanisme avait été élaboré de façon indépendante dans le cadre de l'étude de la machine HLL, lorsqu'un mécanisme similaire, désignant de telles cellules sous le nom de Tombstones, fit l'objet d'une publication [LOM75], nous adoptâmes cette terminologie.

On ne détaillera pas ici le répertoire d'instructions ni les mécanismes d'accès aux opérandes. Il convient toutefois de noter que la gestion des objets de l'architecture virtuelle (e.g. traitement des références pendantes) était résolue par des mécanismes propres à l'architecture virtuelle et non pas au niveau de l'architecture de base. Une grande simplification résultait de cette application du principe de séparation des mécanismes et de la stratégie (i.e. architecture de base = ensemble de mécanismes généraux, architecture virtuelle = mise en œuvre d'une stratégie).
On notera aussi qu'une attention toute particulière avait été accordée aux mécanismes d'édition de liens avec la possibilité de définir différentes stratégies (e.g. liaisons valides pour l'exécution d'une procédure ou bien reconstruites pour chaque exécution de la procédure dans le contexte d'un processus).

3 - 7 - Architecture du matériel

Des études préliminaires ont été menées sur l'architecture du matériel. Elles ont surtout concerné la hiérarchie de mémoire (qui était l'hypothèse de base sur laquelle reposait l'architecture), les transformations d'adresses et la cohérence des caches. La gestion des caches, en environnement multiprocesseur, avec une hiérarchie de caches incluant des caches privés et des caches partagés a fait l'objet d'analyses approfondies. Ainsi, un nouveau protocole de cohérence de cache fondé sur le concept de répertoire (directory) a été développé [CEN78] et un brevet a été déposé. Remarquons que ce concept est supporté par certains microprocesseurs de haut de gamme actuels.
La description des mécanismes de gestion de la mémoire sort du cadre de cet exposé. On doit toutefois noter que les interactions entre la gestion du matériel et la gestion des processus avait fait l'objet d'une analyse détaillée. On distinguait deux espaces pour les états des processus : logique et matériel. Dans le premier espace, les processus étaient (virtuellement) actifs ou bien en attente de ressources logiques alors que, dans le second espace, ces processus étaient (physiquement) actifs ou bien en attente de la résolution de défauts de présence dans les différents niveaux de la hiérarchie de mémoire.

4 - Les acteurs

Les études ont débuté en 1972 au sein de l'entité « Logiciel de base » dirigée par Claude BOULLE, François SALLE étant alors Directeur Technique. L'équipe était placée dans une entité dirigée par Denis DERVILLE. L'étude prit son essor durant l'année 1973. Georges LEPICARD rejoignit la CII durant l'été 1973 et s'intéressa de près à ce projet.
Avec Jean Louis MANSION, nous avions la responsabilité de l'architecture générale. Jean Louis MANSION assurait la direction de l'équipe logiciel qui comprenait notamment Jean Pierre ARMISEN, Jean Louis DUCHENE, Emmanuel MION, Michel PONTACQ, Jean Paul RISSEN et Richard WATREMEZ. Les travaux sur la machine HLL, réalisés par Gérard BATTAREL et Thierry HEIDET, étaient sous ma responsabilité.
Claude KAISER (INRIA) participa, à titre de conseiller, à des travaux relatifs à l'adressage et à la protection.
Les études du matériel ont été conduites par Lucien CENSIER assisté par les conseils de Paul FEAUTRIER (Université Paris 6). Alice RECOQUE suivait ces travaux sur l'architecture du matériel.

L'effectif de l'équipe a été au maximum d'une dizaine de personnes. Peu de travaux ont été effectués avec les équipes de SIEMENS et PHILIPS au sein d'UNIDATA. En effet, l'étude fût menée de façon indépendante jusqu'à la seconde moitié de 1974 car les développements, de la gamme X notamment, accaparaient la quasi-totalité des ressources disponibles. Dans la seconde moitié de 1974, ces travaux sur l'architecture Y furent présentés à nos partenaires et les groupes de travail furent formés. Le laps de temps entre la formation des groupes de travail et l'arrêt du projet ne permit pas une participation effective de nos partenaires d'UNIDATA.

5 - Epilogue

Le rapprochement de CII et de HONEYWELL-BULL, à partir de 1975, a conduit à l'arrêt des études de la Gamme Y. L'urgence des nécessaires travaux de convergence entre les différentes lignes de produits commercialisées d'une part et la date éloignée de commercialisation de cette gamme d'autre part ont été les raisons majeures qui ont conduit cette nouvelle Société à décider l'arrêt du projet. De ce fait, le développement de cette gamme n'a pas connu de prototype tant matériel que logiciel. Les nécessaires travaux de simulation, tant au niveau du matériel que du logiciel, n'avaient d'ailleurs pas encore débuté.

Bien que les études aient été menées en complète isolation, il existe une grande similitude entre les concepts du System/38 d'IBM (AS/400 de nos jours) annoncé en 1978 [IBM78] [SOL95] et l'architecture de la Gamme Y.
De fait, les architectures fondées sur les capacités n'ont pas connu la visibilité escomptée. On parle de visibilité et non de succès car l'AS/400 et le DPS/9000 ont connu et connaissent une commercialisation réussie, chacun dans leur domaine. En effet, bien peu d'utilisateurs et de programmeurs de ces systèmes savent que ce sont des architectures fondées sur les capacités.

On doit aussi noter qu'INTEL présenta, en 1980, le 432. Ce microprocesseur utilisait le concept de capacité mais les contraintes de réalisation des circuits intégrés (i.e. le budget exprimé en nombre de transistors) n'avaient probablement pas permis de résoudre les problèmes auxquels se heurtent les concepteurs de telles architectures. Les lecteurs intéressés trouveront dans [COL88] une analyse critique du 432.

Remerciements

Que Jean Louis MANSION et Michel PONTACQ trouvent ici l'expression de mes remerciements pour leurs remarques et commentaires qui ont permis d'améliorer la qualité et l'exactitude de cet article.

Bibliographie

[BAT78] G.J. BATTAREL, R.J.CHEVANCE « Requirements for a Safe PL/1 Implementation » ACM SIGPLAN Notices May 1978

[BAT79] G.J. BATTAREL, R.J.CHEVANCE « Design of a High-Level Language Machine » National Computer Conference New York 1979 pp649-655

[CEN78] L.M. CENSIER, P.FEAUTRIER «A New Solution to the Cache Coherence Problem in Multicache Systems »
IEEE Transactions on Computers December 1978 pp 1112-1118

[CHE73] R.J. CHEVANCE « A COBOL Machine »
ACM SIGPLAN/SIGMICRO Interface Meeting - Harriman New York 1973

[CHE78] R.J. CHEVANCE, T. HEIDET « Static Profile and Dynamic Behaviour of COBOL Programs »
ACM SIGPLAN Notices, April 1978

[COL88] R.P. COLWELL, E.F. GEHRINGER, E.D. JENSEN « Performance Effects of Architectural Complexity in the Intel 432 »
ACM Transactions on Computer Systems August 1988 pp 298-339

[FAB74] R.S. FABRY « Capability Based Addressing »
Comm. ACM 17:7 (July 1974) pp 403-412

[IBM78] IBM « IBM System/38 Technical Developments »
IBM Corporation Atlanta GA 1978

[LOM75] D.B. LOMET « Schema for Invalidating References to Freed Storage »
IBM Journal of Research and Development January 1975 pp26-35

[ORG72] E.I. ORGANICK « The Multics System: An Examination of Its Structure »
MIT Press 1972

[SOL95] F. SOLTIS « Inside the AS/400: An In-depth Look at the AS/400's Design, Architecture and History »
Duke Press 1995

L'auteur a abordé la programmation en 1963 par le calcul scientifique (sur la CAB500 installée au CNAM). Diplôme d'ingénieur CNAM en 1966, DEA d'informatique pratique à Paris 6 en 1968 et Doctorat d'Etat en 1970.
Après avoir pratiqué le calcul scientifique, dans le Laboratoire de Moteurs du CNAM puis dans une filiale de la Société BERTIN, il a rejoint la CII en 1968.
Après avoir contribué au contrôle de qualité des logiciels, il a participé au développement de compilateurs et a développé des outils associés.
Il s'est ensuite intéressé aux activités d'architecture de système : machines «langage », gamme Y, petits systèmes de gestion et processeurs de communication.
Il a rejoint TRANSAC-ALCATEL en 1980 pour les études d'architecture des systèmes. Après la fusion de TRANSAC avec CII-HB, il fût l'un des architectes des gammes de produits UNIX.
Au sein de la division Open Systems and Software, il est actuellement « Chief Scientist » en charge des activités prospectives en matière d'architecture de système.
En tant que chargé de cours, il a enseigné à l'Université de Paris VI en DEA de 1969 à 1972, en Maîtrise d'informatique (Compilation puis Génie logiciel) de 1971 à 1990 puis au CNAM de 1991 à 1993 en Intégration des systèmes.
Depuis septembre 1993, il est professeur associé au CNAM