Conception d'applications pour systèmes transactionnels coopérants

Gilles Bogo

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR ES SCIENCES

"Informatique"

par

Gilles BOGO

CONCEPTION D'APPLICATIONS POUR SYSTEMES TRANSACTIONNELS COOPERANTS

Thèse soutenue le 18 Juin 1985 devant la commission d'examen :

Président : L. Bolliet

Examinateurs : M. Adiba

J.C. Chupin

M. Léonard

C. Rolland

M. Schlumberger

UNIVERSITE SCIENTIFIQUE ET
MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

(RANG A)
SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE :

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AVANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo. de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Maths appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (Mr)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (Mr)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique
GASTINEL Noël	Analyse numérique Mathématiques appliquées

.../...

GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André' (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
NOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE :

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie
AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (Mr)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique

.../...

DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (Mr)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (Mr)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (mr)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président: Daniel BLOCH

Vice-Présidents: René CARRE
Hervé CHERADAME
Jean-Pierre LONGEQUEUE

Année universitaire 1983-1984

Professeur des Universités

ANCEAU	François	E.N.S.I.M.A.G
BARIBAUD	Michel	E.N.S.E.R.G
BARRAUD	Alain	E.N.S.I.E.G
BAUDELET	Bernard	E.N.S.I.E.G
BESSON	Jean	E.N.S.E.E.G
BLIMAN	Samuel	E.N.S.E.R.G
BLOCH	Daniel	E.N.S.I.E.G
BOIS	Philippe	E.N.S.H.G
BONNETAIN	Lucien	E.N.S.E.E.G
BONNIER	Etienne	E.N.S.E.E.G
BOUVARD	Maurice	E.N.S.H.G
BRISSONNEAU	Pierre	E.N.S.I.E.G
BUYLE BODIN	Maurice	E.N.S.E.R.G
CAVAIGNAC	Jean-François	E.N.S.I.E.G
CHARTIER	Germain	E.N.S.I.E.G
CHENEVIER	Pierre	E.N.S.E.R.G
CHERADAME	Hervé	U.E.R.M.C.P.P
CHERUY	Arlette	E.N.S.I.E.G
CHIAVERINA	Jean	U.E.R.M.C.P.P
COHEN	Joseph	E.N.S.E.R.G
COUMES	André	E.N.S.E.R.G
DURAND	Francis	E.N.S.E.E.G
DURAND	Jean-louis	E.N.S.I.E.G
FELICI	Noël	E.N.S.I.E.G
FONLUPT	Jean	E.N.S.I.M.A.G
FOULARD	Claude	E.N.S.I.E.G
GANDINI	Alessandro	U.E.R.M.C.P.P
GAUBERT	Claude	E.N.S.I.E.G
GENTIL	Pierre	E.N.S.E.R.G
GUERIN	Bernard	E.N.S.E.R.G
GUYOT	Pierre	E.N.S.E.E.G
IVANES	Marcel	E.N.S.I.E.G
JALINIER	Jean-Michel	E.N.S.I.E.G
JAUSSAUD	Pierre	E.N.S.I.E.G
JOUBERT	Jean-Claude	E.N.S.I.E.G
JOURDAIN	Geneviève	E.N.S.I.E.G
LACOUME	Jean-Louis	E.N.S.I.E.G
LATOMBE	Jean-Claude	E.N.S.I.M.A.G
LESIEUR	Marcel	E.N.S.H.G
LESPINARD	Georges	E.N.S.H.G
LONGEQUEUE	Jean-Pierre	E.N.S.I.E.G
LOUCHET	François	E.N.S.E.E.G
MASSELOT	Christian	E.N.S.I.E.G
MAZARE	Guy	E.N.S.I.M.A.G
MOREAU	René	E.N.S.H.G
MORET	Roger	E.N.S.I.E.G
MOSSIERE	Jacques	E.N.S.I.M.A.G
PARIAUD	Jean-Charles	E.N.S.E.E.G
PAUTHENET	René	E.N.S.I.E.G
PERRET	René	E.N.S.I.E.G
PERRET	Robert	E.N.S.I.E.G
PIAU	Jean-Michel	E.N.S.H.G
POLOUJADOFF	Michel	E.N.S.I.E.G
POUPOT	Christian	E.N.S.E.R.G
RAMEAU	Jean-Jacques	E.N.S.E.E.G
RENAUD	Maurice	U.E.R.M.C.P.P
ROBERT	André	U.E.R.M.C.P.P
ROBERT	François	E.N.S.I.M.A.G
SABONNADIERE	Jean-Claude	E.N.S.I.E.G
SAUCIER	Gabrielle	E.N.S.I.M.A.G
SCHLENKER	Claire	E.N.S.I.E.G
SCHLENKER	Michel	E.N.S.I.E.G
SERMET	Pierre	E.N.S.E.R.G
SILVY	Jacques	U.E.R.M.C.P.P
SOHM	Jean-Claude	E.N.S.E.E.G
SOUQUET	Jean-Louis	E.N.S.E.E.G
VEILLON	Gérard	E.N.S.I.M.A.G
ZADWORNY	François	E.N.S.E.R.G

Professeurs Associés

BLACKWELDER	Ronald	E.N.S.H.G
HAYASHI	Hirashi	E.N.S.I.E.G
PURDY	Gary	E.N.S.E.E.G

Professeurs Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S

FRUCHART	Robert	Directeur de recherche
JORRAND	Philippe	Directeur de recherche
VACHAUD	Georges	Directeur de recherche
ALLIBERT	Michel	Maître de recherche
ANSARA	Ibrahim	Maître de recherche
ARMAND	Michel	Maître de recherche
BINDER	Gilbert	Maître de recherche
BORNARD	Guy	Maître de recherche
CARRE	René	Maître de recherche
DAVID	René	Maître de recherche
DEPORTES	Jacques	Maître de recherche
DRIOLE	Jean	Maître de recherche
GIGNOUX	Damien	Maître de recherche
GIVORD	Dominique	Maître de recherche
GUELIN	Pierre	Maître de recherche
HOPFINGER	Emil	Maître de recherche
JOUD	Jean-Charles	Maître de recherche
KAMARINOS	Georges	Maître de recherche
KLEITZ	Michel	Maître de recherche
LANDAU	Ioan-Dore	Maître de recherche
LASJAUNIAS	Jean-Claude	Maître de recherche
MERMET	Jean	Maître de recherche
MUNIER	Jacques	Maître de recherche
PIAU	Monique	Maître de recherche
PORTESEIL	Jean-Louis	Maître de recherche
THOLENCE	Jean-Louis	Maître de recherche
VERDILLON	André	Maître de recherche
SUERY	Michel	Maître de recherche

Personnalités habilitées à diriger des travaux de recherche
(Decision du Conseil Scientifique)

E.N.S.E.E.G.

ALLIBERT	Colette
BERNARD	Claude
BONNET	Roland
CAILLET	Marcel
CHATILLON	Catherine
CHATILLON	Christian
COULON	Michel
DIARD	Jean Paul
EUSTATHOPOULOS	Nicolas
FOSTER	Panayotis
GALERIE	Alain
HAMMOU	Abdelkader
MALMEJAC	Yves (CENG)
MARTIN GARIN	Régina
NGUYEN TRUONG	Bernadette
RAVAINE	Denis
SAINFORT	(CENG)
SARRAZIN	Pierre
SIMON	Jean Paul
TOUZAIN	Philippe
URBAIN	Georges (Laboratoire des ultra-réfractaires ODEILLO).

E.N.S.E.R.G.

BARIBAUD	Michel
BOREL	Joseph
CHOVET	Alain
CHEHIKIAN	Alain
DOLMAZON	Jean Marc
HERAULT	Jeanny
MONLLOR	Christian

E.N.S.I.E.G.

BORNARD	Guy
DESCHIZEAUX	Pierre
GLANGEAUD	François
KOFMAN	Walter
LEJEUNE	Gérard
MAZUER	Jean
PERARD	Jacques
REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine
BOIS	Daniel
DARVE	Félix
MICHEL	Jean Marie
OBLED	Charles
ROWE	Alain
VAUCLIN	Michel
WACK	Bernard

E.N.S.I.M.A.G.

BERT	Didier
CALMET	Jacques
COURTIN	Jacques
COURTOIS	Bernard
DELLA DORA	Jean
FONLUPT	Jean
SIFAKIS	Joseph

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean
COEURE	Philippe (LETI)
DELHAYE	Jean Marc (STT)
DUPUY	Michel (LETI)
JOUVE	Hubert (LETI)
NICOLAU	Yvan (LETI)
NIFENECKER	Hervé
PERROUD	Paul
PEUZIN	Jean Claude (LETI)
TAIEB	Maurice
VINCENDON	Marc

Laboratoires extérieurs :

C.N.E.T.

DEMOULIN	Eric
DEVINE	R.A.B.
GERBER	Roland
MERCKEL	Gérard
PAULEAU	Yves

I.N.S.A. Lyon

GAUBERT	C.

****

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : **Monsieur M. MERMET**
Directeur des Etudes et de la formation : **Monsieur J. LEVASSEUR**
Directeur des recherches : **Monsieur J. LEVY**
Secrétaire Général : **Mademoiselle M. CLERGUE**

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique - Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique Industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

Je tiens à remercier tout particulièrement

Monsieur Louis Bolliet, Professeur à l'Université de Grenoble, qui m'a fait l'honneur de présider le jury, mais aussi pour les encouragements qu'il m'avait donnés pour commencer mon troisième cycle universitaire.

Madame Colette Rolland, Professeur de Paris I-Sorbonne pour l'intérêt qu'elle a porté à notre travail, les critiques constructives qu'elle a fait à la lecture du manuscript et pour sa participation au jury.

Monsieur Michel Léonard, Professeur à l'Université de Genève, pour ses remarques, ses conseils mais aussi les encouragements qu'il m'a donné après une lecture approfondie du document original ainsi que pour sa participation au jury.

Monsiéur Michel Adiba, Professeur à l'Université de Grenoble. Ses lectures successives du manuscript et ses conseils m'ont permis d'aboutir. Je le remercie d'avoir accepté de participer au jury.

Monsieur Serge Abiteboul, chercheur à l'INRIA qui a bien voulu juger ce travail.

Monsieur Maurice Schlumberger, responsable de CAP SOGETI Innovation qui a accepté de jouer le rôle de "candide" dans le jury.

J'aurais beaucoup à dire pour remercier Jean Claude Chupin, Directeur Technique à Bull SEMS et responsable de ce travail. Depuis dix ans, son soutien amical, ses conseils, ses encouragements m'ont permis de progresser et de mener à bien ce travail.

Mes remerciements vont également à Edouard André avec qui j'ai commencé cette étude. Son expérience et ses idées bouillonantes m'ont enrichi.

Il me faut également remercier tous les participants au projet SCOT et tout particulièrement Paul Decitre et Roland Balter avec qui nous avons surmonté toutes les difficultés du projet. Ce projet a été collectif, sans les uns et les autres, il n'aurait pas abouti.

Je ne peux pas oublier Brigitte et Daphné qui ont du faire preuve d'une grande patience, ainsi que mes amies et amis qui m'ont encouragé et supporté pendant la rédaction de cette thèse. Qu'ils soient tous remerciés.

Gilles Bogo

RÉSUMÉ :

Les moyens offerts par les systèmes de gestion de base de données et les systèmes transactionnels pour maintenir la cohérence et l'intégrité des systèmes d'information sont tout d'abord analysés tant en centralisé qu'en réparti.

La seconde partie est consacrée à l'étude de deux grandes classes de méthodes de conception, l'une fondée sur les modèles de description de données, l'autre sur les types abstraits. Dans chaque cas, une méthode particulière est présentée et analysée.

Après présentation de l'application bancaire pilote, la troisième partie définit un modèle pour la description des applications transactionnelles. Celui-ci est appliqué et confronté à l'application pilote.

La dernière partie décrit la réalisation de ce modèle dans le langage ADA. Un environnement de conception est construit et se présente comme un sur-ensemble du langage ADA. Enfin, cet outil est comparé à d'autres propositions du domaine de la recherche.

AVANT-PROPOS

Le projet SCOT dans lequel s'insère ce travail s'est déroulé au Centre de Recherche BULL de Grenoble entre 1979 et 1982.

Il n'est pas vain de rappeler que ce projet est un projet collectif qui a vu la participation à des titres divers et pour des durées plus ou moins longues de bon nombre de personnes.

Si le projet a été suivi dans son ensemble par le groupe composé de Roland Balter, Jean-Claude Chupin, Paul Decitre et Gilles Bogo, il est bon de rappeler que son lancement et la première phase ont grandement été influencés par Edouard André et Juan Andrade du Centre de Recherche Bull.

Jean-Louis Cheval, Michel Delaunay, Pierre Laforgue et Joëlle Couttaz-Raymond du Laboratoire IMAG, ainsi que Paul Bérard du centre de Recherche BULL ont participé à ce travail. Il ne faut pas oublier les étudiants de l'ENSIMAG qui ont apporté leur contribution au projet.

Une équipe de la Société Générale et de SG2 composée de MM. Propheta, Bodet, Dumaine, Crystal et Heerebout nous a apporté toute l'information concernant l'application traitée dans le projet. Cette coopération s'est concrétisée par une convention entre le Centre de Recherche Bull, la Société Générale et SG2.

Le projet SCOT a été partiellement financé dans le cadre du projet pilote base de données réparties SIRIUS.

CONCEPTION D'APPLICATIONS POUR

SYSTEMES TRANSACTIONNELS COOPERANTS

- TABLE DES MATIERES -

CHAPITRE 1

INTRODUCTION

1. INTRODUCTION

1.1. Problèmes à résoudre

Un système d'information est constitué d'un ensemble de données, éventuellement regroupées dans une base de données, de programmes d'application (ou de transactions) et d'outils de synchronisation (assurant l'accès concurrent aux données par exemple), l'ensemble étant utilisé pour modéliser un système du monde réel. L'un des principaux problèmes posés par les systèmes d'information est le maintien de la cohérence des données qu'ils gèrent avec le monde réel qu'elles sont sensées modéliser.

Un certain nombre d'outils de synchronisation sont nécessaires aux systèmes d'information. Ce sont d'une part les mécanismes internes comme ceux assurant l'accès concurrent aux données, et d'autre part les mécanismes de synchronisation externes permettant, entre autres, les interactions avec le monde extérieur (usagers, autres systèmes d'information, ...).

Les bases de données et les systèmes transactionnels apportent chacun des solutions partielles à ce problème. Plus précisément, les systèmes transactionnels prennent en compte un environnement non fiable et de concurrence d'accès aux données. Les systèmes de gestion de base de données (SGBD), au travers du modèle de description de données structurant ces données et du langage de manipulation s'appuyant sur cette structure tentent de maintenir l'intégrité sémantique du système d'information.

Le maintien de la cohérence pourrait être transparent au niveau de la conception des applications et dans ce cas c'est le système (transactionnel ou SGBD) qui doit prendre en charge ce maintien. Un seul niveau de cohérence (généralement forte) est alors disponible pour les applications, ce qui ne satisfait pas nécessairement tous

les besoins induits par les nécessités du monde réel. En effet, les mécanismes mis en oeuvre par les SGBD aussi bien que par les systèmes transactionnels pour maintenir la cohérence face aux pannes et à l'accès concurrent rendent incompatibles les opérations de longue durée et la disponibilité (le partage) de ces données pour d'autres opérations.

Des outils doivent donc être mis à la disposition des concepteurs pour gérer un niveau de cohérence propre à chaque application. Le concepteur aura alors la charge de définir le découpage de son application en unités d'exécution préservant la cohérence.

Beaucoup d'efforts sont actuellement investis dans la réalisation d'outils de programmation, tant en ce qui concerne les langages de programmation (Ada, Pascal, ...) que dans les ateliers logiciels. On peut citer, rien qu'en France, de nombreux projets dont "Concerto" au CNET, "Adele" au laboratoire IMAG, "Alpage" à CII-HB. Ces ateliers de développement de logiciel comprendront beaucoup d'outils d'aide à la programmation, à la mise au point de programmes et au stockage des programmes et de documentations (gestion de versions).

Ces efforts sont nécessaires. En effet, la programmation n'est plus considérée maintenant comme un art ou un artisanat réservé à un petit nombre, mais une véritable production industrielle et à ce titre, doit correspondre aux règles imposées aux produits industriels (qualité, fiabilité, maintenance, etc...).

Cependant, il nous semble que ces efforts seront insuffisants si on n'associe pas, en amont des outils de programmation des ateliers logiciel, des outils adaptés à la conception et la spécification des systèmes informatiques (en particulier des systèmes d'information et des grandes applications informatiques). Dans certains ateliers, ces outils sont d'ailleurs envisagés. En effet, les nouveaux langages de programmation introduisent des concepts puissants dont doivent tenir compte les outils de conception.

Les méthodologies de spécification proposées jusqu'à maintenant (Z, Special, Dream, Pie,...) se veulent générales c'est à dire qu'elles prétendent traiter tous les types de problèmes de conception de logiciel. Leur généralité les rend parfois difficile d'utilisation dans un contexte particulier. D'autre part, aucune d'entre elles ne prend en compte le problème particulier des applications réparties.

L'objet de l'étude développée dans les chapitres qui suivent, est de proposer une méthodologie adaptée à un problème particulier dans un environnement donné : l'expression et le maintien de la cohérence dans les applications réparties. Mais, tout d'abord, présentons le contexte de cette étude.

1.2. Présentation du projet SCOT

Le projet SCOT a pour but d'étudier et de réaliser un système transactionnel permettant de garantir une certaine cohérence des données dans un environnement réparti où il existe une certaine concurrence d'accès aux données et où la fiabilité des composants (sites et réseau) n'est pas garantie.

Il est le résultat de l'expérience d'un projet antérieur - POLYPHEME - [POLY79] ayant abouti à la réalisation d'un Système de Gestion de Bases de Données Réparti (SGBD-R) avec schéma global et langage de requête relationnels, permettant la coopération de SGBD hétérogènes existants. Cette recherche a permis de mesurer la distance séparant un prototype d'un produit et de déceler les limites actuelles d'une approche uniquement fondée sur la répartition d'un SGBD. Elle a montré qu'un outil très général ne peut prétendre offrir une solution satisfaisante à tous les types de problèmes. De plus, le problème de la gestion de la cohérence des données distribuées n'avait été abordé que très partiellement [ANDR80].

Pour le projet SCOT, nous avons pensé que, plutôt que de valider (qualifier) la réalisation du système par une application (tel qu'il avait été fait dans POLYPHEME), il était plus réaliste de piloter la conception du système par une application de manière à obtenir une convergence entre ce qui était nécessaire pour les applications et ce qu'il était possible d'offrir par le système. Sous réserve de précautions lors du choix de l'application, cette démarche ne restreint pas la généralité des solutions proposées.

Le projet SCOT comporte donc deux parties :

- D'une part, l'étude des mécanismes d'exécution répartie, de maintien de la cohérence en environnement réparti, concurrent

et non fiable. Cette étude a conduit à la proposition de protocoles dont la réalisation constitue le noyau du système transactionnel coopérant SCOT.

- D'autre part, l'étude d'une application "pilote" qui permette, en offrant un outil d'analyse et de spécification adapté,

  + de mieux cerner les problèmes

    - de décomposition d'une application,
    - de recherche des concepts spécifiques à l'application,
    - d'expression de la répartition,
    - d'expression de la cohérence.

  + de confronter les mécanismes étudiés pour le système à une utilisation par une application réelle,

  + de tirer des enseignements de cette expérience concernant l'étude des applications réparties ainsi que des enseignements concernant la conception des systèmes sur lesquels s'exécuteront les applications ainsi conçues. L'étude de l'application a permis de proposer des scénarios d'application qui ont influencé la conception des protocoles du système SCOT en mettant en évidence des problèmes spécifiques aux applications transactionnelles. Ces scénarios ont également permis une évaluation des protocoles.

La première partie s'est concrétisée par la proposition d'un ensemble complet de protocoles permettant la coopération de systèmes transactionnels. Ces protocoles ont été introduits progressivement dans l'architecture réseau (DSA) de BULL. Une maquette a montré qu'il était possible de les réaliser sur un système transactionnel existant.

La seconde partie a permis l'analyse d'une application réelle existante dont la réalisation est transactionnelle. Cette analyse a été effectuée à partir du cahier des charges fourni par ses concepteurs (Société Générale et SG2). Elle a été étudiée et spécifiée avec leur collaboration. Elle a permis de mettre en évidence les problèmes spécifiques aux applications transactionnelles et d'élaborer une méthode de conception adaptée à ces applications. L'application a en-

suite été spécifiée selon la méthode proposée et une partie a été réalisée sur la maquette du système SCOT.

1.3. Présentation de la thèse

Cette thèse concerne essentiellement la seconde partie du projet SCOT, c'est à dire l'étude, la conception et la réalisation d'une application répartie garantissant une certaine cohérence des données. Cependant, tout au long de la présentation, il sera fait référence aux mécanismes et aux protocoles du système SCOT. Cette référence est faite à titre d'exemple car tout autre système de même type, c'est à dire offrant les mêmes garanties en ce qui concerne le maintien de la cohérence, serait tout aussi adapté. Tout au long du document nous donnons des exemples en utilisant le langage ADA autant pour les concepts qu'il propose que pour sa bonne lisibilité.

Le premier chapitre est consacré à l'analyse du problème de l'expression et du maintien de l'intégrité et de la cohérence tant en environnement centralisé que réparti. Des solutions partielles existent dans les SGBD et dans les systèmes transactionnels conventionnels. Elles sont examinées et confrontées aux objectifs d'une application réelle. Le système SCOT est un exemple de système offrant une solution complète au maintien de la cohérence stricte en environnement transactionnel coopérant. Les protocoles utilisés par le système SCOT sont décrits et comparés à d'autres approches du domaine de la recherche.

Dans le second chapitre, des méthodes de conception et de spécification sont étudiées. Deux grandes classes de méthodes sont abordées : l'une fondée sur les modèles de base de données, l'autre fondée sur les types abstraits. Dans chaque cas, une méthode particulière est détaillée et discutée.

Dans le troisième chapitre, un modèle pour la conception des applications transactionnelles est proposé. Il est fondé sur les notions de type abstrait et de transaction en tant qu'opérateur sur un type. La notion d'opération est proposée comme une extension permettant de tenir compte des actions de longue durée. Toutes ces notions sont justifiées par des exemples tirés d'une application bancaire décrite dans cette partie.

Enfin, le dernier chapitre présente une réalisation du modèle utilisant le langage Ada. On donne la syntaxe et la sémantique des extensions faites au langage. L'utilisation du langage ainsi défini est montrée sur de nombreux exemples issus de l'application. Des conclusions sont tirées concernant l'utilisation de la méthodologie proposée et aussi des implications qu'elle a sur la conception des systèmes transactionnels coopérants qui supporteront des applications ainsi conçues.

Pour terminer, la méthode est comparée à d'autres propositions qui restent du domaine de la recherche.

La conclusion tire les leçons de cette étude et propose un certain nombre de perspectives qui sont ouvertes tant pour la conception des applications transactionnelles que pour leur réalisation à l'aide d'un langage adapté (extension du langage ADA). Il est également montré quelles répercussions cette approche peut avoir sur les modèles de base de données utilisés par les systèmes d'information ainsi conçus.

CHAPITRE 2

BASES DE DONNEES ET SYSTEMES TRANSACTIONNELS

2. BASES DE DONNÉES ET SYSTÈMES TRANSACTIONNELS

Notre objectif, dans cette partie, est d'analyser les moyens dont il est possible de disposer pour maintenir la cohérence et l'intégrité dans les systèmes d'information. Par système d'information nous désignons les applications informatiques composées d'un ensemble de données structurées (généralement une base de données), d'un ensemble de programmes (requêtes à la base de données ou transactions) permettant la manipulation des données et d'un ensemble d'outils de synchronisation assurant le contrôle de l'exécution des programmes sur les données.

Notre intérêt s'est porté essentiellement sur les systèmes d'information de gestion mais les résultats de cette étude peuvent être étendus à tous les systèmes d'information nécessitant la garantie d'une certaine cohérence.

La majorité des systèmes d'information présentent une "couche base de données" (incluant éventuellement un langage de requête) et une couche pour la gestion des programmes d'application ou des transactions. Les produits récents des constructeurs tendent à imbriquer les deux niveaux, soit en ajoutant des fonctionnalités algorithmiques ou transactionnelles aux langages de manipulation des SGBD (PL/1 pour system R, COBOL pour IDS), soit en offrant à l'utilisateur terminal ou aux programmes d'application une interface de nature exclusivement transactionnelle (TDS-IDS, CICS-DL1).

Pour mieux comprendre le rôle joué par chaque fonctionnalité (transactionnel ou base de données) dans les systèmes d'information, on se propose de les distinguer.

D'une part, le module de gestion des données travaille à partir de schémas de description (conceptuel, interne) accèdant aux données. Il a essentiellement un rôle de structuration de l'information. Si un langage de manipulation lui est associé, il est fondé sur la ma-

nipulation des structures définies dans le modèle. Ce module gère donc l'aspect statique (l'état permanent) du système d'information.

D'autre part, le module de gestion des transactions contrôle l'exécution des programmes d'application. Il gère l'aspect dynamique du système d'information, c'est à dire les règles de changement d'état.

On appelle transaction une unité d'exécution regroupant un ensemble indivisible d'actions élémentaires sur des données. On entend par action, toute opération qui affecte l'environnement de travail de la transaction, qu'elle soit directement visible (édition d'un bordereau, envoi d'un message sur le terminal de l'usager, ...) ou indirectement (consultation ou modification d'une donnée de la base). Le système transactionnel garantit aux transactions la propriété d'atomicité. Ceci signifie que les actions d'une transaction sur l'environnement de travail sont toutes exécutées, ou qu'aucune ne l'est. En anticipant sur la suite, on peut dire que dans le premier cas, le système d'information passe d'un état cohérent à un autre sans transition visible et que dans le second cas, il ne change pas d'état.

La propriété d'atomicité permet de préserver la cohérence des données en dépit des pannes et de l'accès concurrent.

2.1. Cohérence et intégrité dans les systèmes d'information

L'un des rôles essentiels des systèmes d'information est le maintien de la cohérence des données qu'ils ont à gérer.

Pour la cohérence, il faut envisager deux notions :

- D'une part, l'intégrité du système d'information par rapport au monde réel qu'il est sensé modéliser. Il s'agit d'abord de la qualité de la modèlisation, c'est à dire les capacités du modèle à représenter le monde réel et de l'utilisation qu'en fait le concepteur. Cette cohérence s'exprime généralement par des règles (que ce soit le modèle de description de données ou les contrain-

tes d'intégrité) qui doivent être vérifiées lors de tout changement d'état du système d'information. De plus, la capacité du système supportant l'application à maintenir la cohérence des données avec le monde réel en dépit des facteurs indépendants de la modèlisation (les pannes et l'accès concurrent) joue un rôle essentiel.

- D'autre part, la qualité physique des données, c'est à dire la validité des états pris par les données lues ou écrites sur leur support physique. Cette qualité est assurée par la sauvegarde et la restauration de ces supports dans un état cohérent.

Dans ce qui suit, quand nous parlerons d'intégrité sémantique, ou plus simplement d'intégrité, il s'agira de la modèlisation du monde réel et du maintien de l'intégrité du système d'information avec le monde réel en liaison avec cette représentation (erreurs de conception, règles d'intégrité insuffisantes, incapacité du modèle à prendre en compte les possibilités du monde réel, etc).

Nous examinons d'abord les différentes causes pour lesquelles un système d'information peut devenir incohérent. Nous verrons quelles solutions apportent les bases de données d'une part et les systèmes transactionnels d'autre part, d'abord en environnement centralisé, puis en réparti.

2.1.1. Les pannes

On entend par panne, tout incident dans l'environnement où s'exécute le système d'information, c'est à dire les défaillances du matériel ou du système d'exploitation, mais aussi celles du système d'information lui-même dues, par exemple, à une erreur de programmation conduisant à ne pas réaliser entièrement un changement d'état du système d'information.

Dans la manipulation de données correlées, les pannes peuvent compromettre la cohérence en arrêtant une séquence d'opérations portant sur ces données. Une partie seulement des données correlées sont modifiées et les règles d'intégrité peuvent ne plus être respectées.

C'est le principal problème auquel le mode transactionnel apporte des solutions.

La transaction définissant une opération atomique est complètement exécutée ou ne l'est pas du tout. Pour celà, les données permettant soit de terminer, soit de revenir à l'état antérieur sont sauvegardées sur un support secondaire (journalisation) et la terminaison de la transaction est enregistrée à l'aide d'une seule entrée-sortie (assurant l'exécution ou la non exécution) en un point appelé point de validation (ou "commitment").

Si un incident a eu lieu avant le point de validation, à la reprise, la base est éventuellement remise dans un état cohérent précédent à l'aide de mécanismes de retour arrière dépendants du type de journalisation (avant, après, ...), puis la transaction est réexécutée.

Si l'incident survient après, et que l'étape de validation n'est pas encore terminée, alors seule l'étape de validation est reprise. Pour assurer l'atomicité, la phase de validation, consécutive à ce point qui consiste à consolider les modifications dans la base, est une opération idempotente, c'est à dire qu'elle peut être réexécutée autant de fois que celà est nécessaire sans modifier l'état final. Il est montré dans [SCOT20] que cette propriété n'est pas toujours vérifiée et que la réexécution de la phase de validation peut engendrer des effets de bord tels que la duplication de messages vers l'usager. Celà ne remet toutefois pas en cause la cohérence des données.

2.1.2. L'accès concurrent

L'exécution simultanée de requêtes ou de transactions dans un système d'information a été introduite principalement pour permettre à un grand nombre d'usagers interactifs de partager simultanément (de manière optimale si possible) les ressources du système. Ceci implique le partage d'un même ensemble de données pouvant conduire à des incohérences (pertes de mises à jour par exemple). Le but du contrôle d'accès concurrent est de contrôler l'exécution simultanée de requêtes (à la base de données) ou transactions accèdant à des données partagées, afin de garantir que chacune d'elles a un résultat analogue à celui obtenu si elle s'était exécutée seule dans le système.

Les systèmes transactionnels aussi bien que les SGBD qui offrent un accès partagé, disposent de mécanismes pour contrôler la concurrence

d'accès aux données. Sans entrer dans les détails de ces mécanismes, on peut noter que la plupart d'entre eux sont fondés sur le verrouillage des ressources [GRAY78]. L'acquisition des ressources peut être statique (au début de la requête) ou dynamique (au moment de l'utilisation) et l'accès à une ressource peut être partagé (lecture seule) ou exclusif (modification).

La manipulation, par les techniques évoquées ci-dessus, de données correlées, peut provoquer des interblocages, ce qui oblige à associer aux techniques de verrouillage des mécanismes de détection ou de prévention de ces interblocages.

Dans les systèmes centralisés, ce sont généralement des mécanismes de détection des interblocages qui sont utilisés. Par exemple, dans les techniques de détection par temporisation, à chaque transaction est associé un délai maximal d'exécution. Quand ce délai est atteint, une des transactions bloquées est abandonnée. Le choix de cette transaction peut se faire sur l'âge de la transaction, sur le type de ressources détenues (critiques ou non), etc.

Nous verrons que pour les systèmes répartis, on utilise plutôt des mécanismes de prévention des interblocages.

2.1.3. Les erreurs sémantiques

Il s'agit des erreurs qui, à la suite de modifications des données, conduisent le système d'information à un état incompatible avec le monde réel qu'il modélise. Ces erreurs sont dues soit à une mauvaise conception du système d'information (absence de contrôles, etc), soit à un manque de protection vis à vis des usagers (saisie de données incorrectes par exemple) ou des programmes d'application (erreurs de programmation). La détection mais surtout la prévention de telles erreurs sont essentielles pour le fonctionnement correct des systèmes d'information.

Dans les bases de données, ce problème est pris en compte, d'une part, par le modèle de description de données et d'autre part, par le contrôle d'intégrité sémantique du SGBD [BADA79], [HAMM75], [FERR82].

- Intégrité contrôlée par le modèle :

Le modèle permet de décrire les règles de dépendance (dépendances fonctionnelles, associations, unicité des clés,...) entre les entités modélisées. Les modèles sont nombreux mais peuvent se diviser en quelques grandes classes : hiérarchique (IMS), réseau (CODASYL, SOCRATE), relationnel, entité-association, fonctionnel (DAPLEX, Data Semantics), etc. On a cherché à faire entrer de plus en plus de sémantique dans les modèles (introduction de la notion de ROLE [BACH78][CODD79], modèle fonctionnel [SHIP79]) sans parvenir à exprimer complètement la sémantique du monde réel. En particulier, ces modèles, bien adaptés à la description de structures et de dépendances statiques, deviennent inaptes à la description des comportements, c'est à dire de toute la dynamique du monde réel.

Par exemple, définir un compte bancaire par la seule structure (relation) suivante :

relation COMPTE (numéro_cpt, proprio, solde, ...)

est insuffisant. Il est aussi nécessaire de préciser quelles opérations permettent de changer son état et l'algèbre relationnelle ne permet d'exprimer ni le fait que les seules opérations possibles sont CREDIT et DEBIT, ni les liens qui existent entre ces opérations.

Quelques tentatives ont été faites pour associer opérations et données : Data Semantics [ABRI74] qui n'a pas été poursuivie, TYP [CHAB80] qui reste un système expérimental.

Le modèle de base de données généralisées développé dans le cadre du projet TIGRE [TIGR83] est une extension du modèle entité-association. En plus de la généralisation de la notion de donnée à des types de données de structure complexe (tel que le document), l'extension conduit à considérer les données comme des types abstraits, pour lesquels, en plus de la structure, doivent être définies des opérations de manipulation (éditer un document par exemple).

- Les contraintes d'intégrité :

Pour compléter les modèles, les SGBD peuvent disposer d'un sous système de contrôle de l'intégrité sémantique. Celui-ci permet de définir a priori un ensemble de contraintes (assertions ou prédicats) sur le schéma de description de la base de données.

Les contraintes expriment des restrictions sur les domaines des valeurs prises par les données, ou bien des restrictions sur les changements d'état possibles. A chaque changement d'état de la base (modification), le SGBD vérifie soit que les données satisfont bien cet ensemble de contraintes d'intégrité, soit que, compte tenu des valeurs des données de la base, le changement d'état est autorisé. Cependant, dans cette technique, le contrôle ne s'exerce que sur le résultat de l'opération au moment de l'affectation de la valeur à la donnée, jamais sur le processus qui a permis de parvenir à cette valeur, sauf en ce qui concerne les pré-conditions qui autorisent l'opération de se dérouler.

En ce qui concerne les changements d'états, la difficulté majeure de cette méthode provient du fait que ces contraintes sont définies indépendamment de l'utilisation des données. En effet, l'usager de la base de données est autorisé à définir ses requêtes au fur et à mesure de ses besoins et qu'elles sont analysées indépendamment les unes des autres. Les requêtes n'étant pas connues à l'avance, il est nécessaire de définir un ensemble complet et cohérent de contraintes, c'est à dire pouvant "parer à toute éventualité". Il y a alors fort à parier que cet ensemble sera trop restrictif, trop lourd ou au contraire insuffisant.

La vérification du respect des contraintes peut se faire statiquement lors de la "compilation" de la requête (pour certaines contraintes portant sur des valeurs, par exemple : la vérification de la valeur de remplacement dans une simple modification). Elle peut également se faire dynamiquement lors de l'exécution de la requête, la transgression d'une contrainte entrainant l'abandon de celle-ci, ou encore après exécution de la requête. On vérifie, avant consolidation des modifications dans la base, que l'état de la base est toujours cohérent. Ces deux dernières méthodes nécessitent des mécanismes de journalisation, de retour-arrière et de reprise analogues à ceux utilisés pour la prise en compte des pannes.

Cette méthode de contrôle de l'intégrité sémantique, très coûteuse, est la seule possible dans les SGBD interactifs dans lesquels les requêtes sont inconnues à l'avance.

Le mode transactionnel ne fournit aucun outil pour contrôler l'intégrité sémantique. Ici, les contraintes s'expriment non plus statiquement sur les données, mais dynamiquement dans les programmes. Cependant, l'utilisation d'un ensemble limité et prédéfini de requêtes que constituent les transactions autorisées pour la modification du système d'information, permet de mieux les maîtriser. Le problème se pose donc différemment et pourra être résolu en utilisant une bonne méthodologie pour la conception de l'application.

En effet, les contraintes d'intégrité doivent être prises en compte lors de la conception du système d'information. Leur expression constitue même une part importante de la spécification de celui-ci. De même que la définition des contraintes d'intégrité d'un SGBD est un des rôles essentiels du gérant de la base de données, c'est une lourde responsabilité pour le concepteur d'application que de définir les contraintes d'intégrité et de les introduire dans les transactions.

Ceci pourrait nous amener à penser que cette technique nous conduit à l'écriture d'une application manipulant des fichiers correlés comme si elle était réalisée avant l'introduction des bases de données. Cependant, il faut noter que la plupart des systèmes d'information reposent sur l'utilisation d'un SGBD (comme une super méthode d'accès aux données) pour lequel un ensemble de contraintes d'intégrité peut être défini au moins pour les données élémentaires de l'application (certainement plus). Il reste donc au concepteur d'application à introduire les contraintes d'intégrité à un niveau plus global, celles qui portent sur des objets correlés mis en jeu dans les transactions.

2.2. Problèmes spécifiques à l'environnement distribué

2.2.1. La répartition des données

Dans un système distribué, les différents sites qui coopèrent ne disposent pas de mémoire commune et communiquent par messages à travers un réseau dont la fiabilité n'est pas garantie. Du fait de la durée de circulation des messages, il est impossible de garantir la synchronisation de tous les sites. Il sera donc impossible d'évaluer un état instantané global du système d'information.

Dans un SGBD réparti, une requête d'un usager ou une transaction accèdant à des données distribuées sur plusieurs sites (qu'on appellera requête ou transaction globale) devra donc être éclatée en plusieurs sous-requêtes ou sous-transactions qui s'exécuteront sur ces différents sites et qu'on apellera requête ou transaction locale. De nombreux travaux [POLY79] [CALE78] [LEBI79] [SDD180] ont été consacrés à la décomposition des requêtes globales en sous requêtes locales et à leur exécution.

La plupart des méthodes de décomposition s'appuient sur l'existence d'un schéma global de description de données (généralement relationnel) qui intègre les différents schémas locaux [ADIB78]. Ce schéma décrit l'ensemble des données de la base de données ainsi que leur répartition. Le schéma global constitue la référence pour la compilation des requêtes à la base de données répartie, c'est à dire que les requêtes sont exprimées dans les termes de ce schéma global. Les requêtes globales sont ensuite décomposées en requêtes locales en fonction des informations de répartition contenues dans le schéma. Chacune des requêtes locales est exprimée dans les termes du schéma local auquel elle est destinée.

L'aspect statique du schéma global et le mode assertionnel du langage d'interrogation de ces systèmes, conduisent à de nombreuses difficultés dans l'expression des requêtes globales. En effet, il est très difficile de séparer les données de localisation des données propres au système d'information (comme il est proposé dans le projet POLYPHEME), la localisation étant souvent le résultat d'un calcul sur les données comme il a été montré dans [BOGO78] ou le résultat de la consultation de dictionnaires de données (system R*).

Afin de garantir la cohérence des données, les requêtes au SGBD réparti sont décomposées en un ensemble de transactions.

Deux approches de cette décomposition des requêtes ont été faites :

- Laisser à l'utilisateur final le soin d'enchainer les requêtes globales [LEBI79], les transactions locales apparaissant comme des sous-requêtes (résultat de la décomposition), dans des plans d'exécution fixés lors de la décomposition, et destinés aux SGBD locaux,

- Intégrer le langage de manipulation à un langage de programmation (PL/1 pour le système R*), les transactions apparaissant explicitement dans les programmes d'application, indépendamment des requêtes d'accès à la base de données, ces transactions pouvant elle-même être décomposées en transactions locales au vu des accès aux données qu'elles manipulent et après consultation d'un dictionnaire de données.

Nous allons voir maintenant comment ces nouveaux paramètres (répartition, décomposition,...) interviennent sur le problème du maintien de la cohérence et de l'intégrité sémantique. On indiquera ensuite les solutions choisies dans le système SCOT.

2.2.2. Les pannes en environnement réparti

La participation de plusieurs sites et l'existence d'un réseau de communication augmentent le risque de pannes. En particulier, une panne du réseau peut conduire à l'isolement d'un site participant à l'exécution d'une transaction globale, créant, dans certains cas, une incertitude sur la terminaison des actions. De plus, la validation d'une requête ou d'une transaction ne peut plus s'effectuer en une seule entrée-sortie puisque plusieurs sites interviennent.

Afin de respecter la propriété d'atomicité d'une transaction au niveau global, il est nécessaire de synchroniser toutes les opérations de validation sur les sites intervenant dans la transaction et de n'effectuer une validation partielle (locale) que si on est certain que tous les autres feront de même. Du fait de la durée de transit

des messages dans le réseau, la validation ne sera donc pas instantanée ; on n'aura plus un point de validation comme en centralisé, mais une période pendant laquelle les données sont (globalement) dans un état incohérent et doivent, par conséquent, ne pas être visibles pour les autres transactions.

Si une panne (de site ou de réseau) intervient après qu'une transaction ait terminé son exécution et avant la réception d'un message de validation, elle peut laisser une indétermination quant à la terminaison de la transaction globale. L'indétermination ne pourra être levée et les ressources, acquises par l'ensemble des transactions locales de la globale concernée, ne pourront être restituées que lors de la reprise du site en panne ou du rétablissement de la communication. Cette période d'attente peut être longue si la panne persiste et nous verrons plus loin (c.f. 2.4.4.) des techniques qui permettent de minimiser cette période.

Les techniques de validation et de reprise après panne doivent tenir compte de ces incertitudes. Un protocole de validation et de reprise après panne doit être mis en oeuvre entre tous les sites coopérant à l'exécution de la requête ou transaction globale. Des solutions existent [GRAY78], [LIND79] [SCOT14] et sont détaillées dans la section 2.4.

2.2.3. L'accès concurrent en environnement réparti

Dans les systèmes d'information répartis, les méthodes de contrôle de l'accès concurrent utilisent soit des techniques de verrouillage des ressources, soit des techniques d'estampillage des données. Les verrous peuvent être partagés (généralement en consultation) ou exclusifs garantissant alors l'accès à la ressource par une seule transaction. L'acquisition dynamique des ressources introduit le risque d'apparition d'interblocages entre les transactions globales.

Les techniques permettant de résoudre le problèmes des interblocages en environnement réparti sont généralement des techniques de prévention [ROSE78], [BERN79]. En effet, les techniques de détection par temporisation imposent de connaître la durée d'exécution d'une transaction globale (afin de définir une horloge de garde adaptée à chacune d'elles) ou l'état instantané global, ce qui est très difficile à évaluer du fait de l'intervention de sites différents et de

la durée de transit des messages sur le réseau. Les techniques fondées sur la maintenance d'un graphe des attentes [STON77] [MENA79] sont complexes et conduisent à des temps de détection longs, à des échanges de messages nombreux sur le réseau et à des abandons de transactions à tort.

Les techniques de prévention visent à éviter les situations qui peuvent conduire à un interblocage. Une transaction est abandonnée quand elle est susceptible de conduire à un interblocage. Les solutions se distinguent par les règles qu'elles utilisent pour déterminer les transactions qui peuvent conduire à un interblocage. La technique généralement adoptée consiste à ordonner les transactions par un estampillage de celles-ci. A chaque transaction globale est associée une valeur qu'on appelle estampille de la transaction et qui est liée à son âge et au site sur lequel elle est créée. Ainsi, chaque estampille est unique dans tout le système et l'ensemble des estampilles constitue un ensemble ordonné. Dans le techniques à estampillage, les transactions qui peuvent conduire à un interblocage sont, au vu des estampilles respectives des transactions actives et en attentes, soit abandonnées, soit mises en attente. Les transactions abandonnées seront reprises ultérieurement avec la même estampille. Cette méthode assure un vieillissement des transactions dans le système. Les transactions les plus anciennes ayant priorité sur les plus jeunes, elle écarte ainsi le risque de famine, c'est à dire qu'une transaction ne risque pas, à cause du contrôle d'accès concurrent de rester indéfiniment dans le système.

Comme dans le cas de la validation, le contrôle d'accès concurrent implique une coopération des différents sites qui rend un protocole nécessaire. Un exemple de protocole de contrôle d'accès concurrent est donné dans la section 2.4.

2.2.4. Intégrité sémantique en environnement réparti

La difficulté du contrôle de l'intégrité sémantique en réparti provient du fait qu'il est impossible d'évaluer un état global instantané de la base de données. Comme dans le cas centralisé, examinons les moyens dont on dispose pour exprimer les règles d'intégrité et les problèmes qui se posent.

- **Tout d'abord au niveau du modèle.**

L'établissement d'un schéma global intégrant les schémas locaux pouvant être hétérogènes, s'avère nécessaire. Le modèle MOGADOR, par exemple, [ADIB78] permet d'établir des correspondances entre le modèle relationnel et les modèles hiérarchique et réseau. Dans le cas où les modèles locaux sont homogènes (cas de SIRIUS DELTA où le modèle est relationnel), la solution se trouve simplifiée.

Si la solution est possible, des problèmes subsistent. Il est nécessaire en particulier, que chaque SGBD local réalise complètement les correspondances entre le modèle local et le modèle global (toutes les opérations relationnelles si le modèle global est relationnel). Ceci est une limitation importante si on veut faire coopérer des machines très différentes (gros centraux et micro-ordinateurs). Cette coopération n'est pas illusoire compte tenu des progrès de la micro-électronique et des nécessités de la décentralisation. En particulier, on ne voit pas comment on pourrait implanter un SGBD relationnel dans une carte de crédit à micro-processeur (CP8) qui interviendra bientôt dans les réseaux bancaires. Inversement, il serait possible d'y implanter les transactions locales à cette carte, c'est à dire la gestion des quelques données qu'elle contient, le lancement de transactions à distance et les mécanismes de validation transactionnelle.

- **Les contraintes d'intégrité.**

En environnement distribué, deux situations peuvent se présenter :

1- la base de données réparties est toujours accédée au travers du schéma global. Elle est vue comme une seule et même base. Il n'existe pas de distinction entre les données strictement locales (sans corrélation avec des données d'une autre base locale) et des données pour lesquelles cette corrélation existe. Dans ce cas, les contraintes ne s'appliquent qu'aux données globales.

2- Les données sont accédées soit au travers du schéma global, soit au travers des schémas locaux à chaque base participante (coopération de bases préexistantes). Dans ce cas, il existe des contraintes globales et des contraintes régissant les accès locaux.

Du point de vue définition, les contraintes peuvent porter soit sur le schéma local, soit sur le schéma global [ANDR80]. La notion de

contrainte globale est importante puisqu'elle permet d'exprimer des restrictions sur des données correlées résidant sur des sites distincts. Ces contraintes n'ont de sens sur aucun site pris séparément.

Deux démarches permettent d'aborder le problème de l'expression des contraintes :

- Ascendante :
  Définition de contraintes locales qui, par composition en fonction des corrélations de données appartenant à des sites distincts, permettent d'obtenir des contraintes globales.

- Descendante :
  Définition de contraintes globales qui sont décomposées en fonction des schémas locaux.

Dans le premier cas, les contraintes globales sont obtenues par union ou intersection de contraintes locales, ce qui peut conduire, selon les cas, soit à une permissivité trop grande (si on choisit par exemple la contrainte la plus faible), soit inversement à une impossibilité d'accès à certaines données.

La seconde démarche est délicate. Une contrainte d'intégrité globale est un prédicat qui doit décomposé en contraintes (prédicats) s'exprimant sur les schémas locaux qui permettront les vérifications dynamiques. De plus, dans le cas où chaque base locale est aussi accédée localement, des contraintes purement locales peuvent avoir été définies indépendamment des contraintes globales. Ces différentes contraintes ne doivent pas entrer mutuellement en conflit.

Le problème de la définition est accentué d'une part par la coopération de plusieurs bases pouvant avoir chacune leur propre ensemble de contraintes locales et d'autre part, par la nécessité de définir un jeu exhaustif de contraintes pour permettre un accès interactif de la base à l'aide d'un langage d'interrogation.

Du point de vue vérification, une requête globale est décomposée en requêtes locales (devant s'exécuter sur un seul site). Les contraintes statiques peuvent être contrôlées de deux façons :

- au niveau global ; et s'il est fait référence à des données locales, la base concernée est interrogée. Les requêtes locales ne sont exécutées que si les contraintes statiques sont vérifiées.

- au niveau local ; chaque requête locale a la charge de vérifier les contraintes statiques concernant les données qu'elle met en jeu. Dans ce cas, des mécanismes de synchronisation et de reprise sont nécessaires pour annuler les effets des requêtes locales qui ont pu s'effectuer alors que l'une d'elles a détecté une erreur. La vérification des contraintes statiques peut ici s'apparenter à celle des contraintes dynamiques.

La vérification des contraintes dynamiques ne peut s'effectuer qu'au niveau local, soit au cours de l'exécution, soit en fin d'exécution des requêtes. Des mécanismes de validation, de reprise et de retour-arrière analogues à ceux utilisés pour le contrôle de cohérence externe (pannes) sont nécessaires. Au coût de vérification des contraintes, analogue à ce qu'on a en centralisé, s'ajoute ici le coût des messages échangés entre les sites locaux et le site global.

Dans la première situation (accès global seul), les deux schémas de vérification sont acceptables. Dans la seconde (accès global et local), afin que les même contrôles soient effectués sur les requêtes globales et lcales, seules les vérifications locales sont acceptables.

Dans la mesure où les mécanismes de synchronisation et de reprise sont imposés par la vérification des contraintes dynamiques, la vérification statique au niveau global ne se justifie que si les mécanismes d'activation d'une action locale et de synchronisation sont coûteux.

En conclusion, le contrôle d'intégrité sémantique en environnement distribué pose deux problèmes :

- Difficulté à définir de manière cohérente et exhaustive les contraintes d'intégrité,

- Coût élevé du contrôle systèmatique de l'intégrité sémantique.

Or, dans de nombreuses applications,le concepteur connait les contraintes à respecter en fonction du contexte. Il peut déterminer

dès la conception les opérations qui risquent de mettre en défaut la cohérence et donc mettre en oeuvre les mécanismes nécessaires uniquement dans ces cas [ROLL81d].

Dans le cas réparti plus encore qu'en centralisé, les systèmes transactionnels n'offrent aucun outil pour définir et assurer le maintien de l'intégrité sémantique. De plus, ici, les programmes s'exécutent sur des sites différents, peuvent avoir été conçus par des équipes différentes. Cependant, dans le cas du transactionnel, le concepteur maitrise l'ensemble des actions réalisables sur les données (transactions). Il n'a donc à déterminer que les règles d'intégrité des données liées aux seules actions possibles. D'autre part, il dispose des outils de synchronisation et de reprise qui sont nécessaires au maintien de la cohérence. Il maitrise donc les règles à respecter et les outils pour les respecter.

Pour la réalisation d'applications transactionnelles réparties, une méthodologie de conception globale s'impose donc.

2.3. CONCLUSION

Il existe une certaine analogie entre les moyens mis en oeuvre pour maintenir la cohérence dans les systèmes centralisés et dans les systèmes répartis.

Si les mécanismes de contrôle de cohérence vis à vis des facteurs externes (exécution, pannes, accès concurrent) sont maitrisés, aussi bien en centralisé qu'en réparti, ceux liés à l'expression des contraintes d'intégrité sémantique et à leur vérification restent, en réparti comme en centralisé, à un niveau tout à fait insuffisant. De plus, les systèmes d'information tendent à intégrer de plus en plus de sémantique du monde réel (automatisation de certaines tâches, courrier électronique, etc...) et l'expression des contraintes d'intégrité sémantique devient un facteur prépondérant de ces systèmes.

La section suivante décrit la mise en oeuvre d'un système transactionnel coopérant : le système SCOT. On y montre comment la notion de transaction et les protocoles utilisés dans le système SCOT pour

faire face aux pannes et à l'accès concurrent en environnement transactionnel coopérant, permettent de définir un premier niveau de cohérence (forte). Le système SCOT est montré comme un exemple complet de système transactionnel coopérant. Tout autre système offrant les même fonctionnalités en ce qui concerne le maintien de la cohérence aurait pu être le support de cette étude sur la conception des appplications en vue du contrôle et du maintien de l'intégrité. Nous utiliserons SCOT pour nos exemples, mais également pour montrer comment cette approche peut être intégrée à un langage et une méthodologie de conception des applications transactionnelles réparties.

Dans les chapitres suivants, nous montrons comment, en s'appuyant sur cette base indispensable, on peut décrire la sémantique d'une application en intégrant les aspects statiques aussi bien que dynamiques.

2.4. Le système SCOT

Cette présentation est largement inspirée d'une partie du rapport de recherche [SCOT16]. Elle est importante pour la compréhension du chapitre 5 où la méthode de conception est appliquée aux mécanismes développés dans le système SCOT.

2.4.1 Présentation du système

Le système SCOT [SCOT8] permet la coopération, à travers un réseau général d'ordinateurs, de systèmes transactionnels résidant sur des sites distincts et présentant une certaine autonomie.

Sur chaque site, un système transactionnel contrôle l'exécution cohérente de transactions qui sont des programmes qui résident et s'exécutent sur un seul site et ne manipulent que des données locales (gérées par le système transactionnel). Cette approche est notablement différente de celle des SGBD-R pour laquelle on envisage le transport d'information pour effectuer le traitement sur un autre site (jointure de deux relations résidant sur des sites distincts).

Lorsqu'on connecte entre eux plusieurs systèmes transactionnels, on peut réaliser une application distribuée en faisant coopérer les transactions. Cette coopération se traduit par le lancement de transactions à distance et l'échange de messages entre transactions s'exécutant sur des sites distincts.

On étend le concept de transaction à cette coopération en la désignant sous le terme de transaction globale. L'exécution des transactions coopérantes doit vérifier dans son ensemble la propriété d'atomicité.

Dans la suite, on désignera sous le nom d'agent chaque transaction coopérante et on réservera le terme transaction pour désigner le groupe d'agents pour lesquels les systèmes transactionnels garantissent la propriété d'atomicité.

La programmation d'un agent fait apparaître explicitement les relations possibles avec les autres agents (activation, dialogue). Le contrôle des schémas d'exécution (exécution parallèle ou exécution procédurale) reste sous la responsabilité du programmeur d'application.

Une transaction (globale) est initialisée sur un site par l'exécution d'un agent (appelé agent initial de la transaction) qui peut demander l'exécution, sur des sites différents ou non, d'autres agents dont l'identité et le site d'exécution sont dictés par l'application et peuvent être éventuellement le résultat d'un algorithme. Ce schéma est transitif et engendre un arbre d'invocation.

Par la suite, deux agents peuvent dialoguer en fonction des besoins de l'application.

Il est donc clair que sur un tel système, une application distribuée ne se réalise pas comme un programme unique que l'on répartit ensuite, mais que celle-ci provient (elle est la conséquence) de la coopération d'agents s'exécutant sur des sites différents.

Le système SCOT fournit au concepteur d'application les commandes suivantes pour contrôler l'exécution des transactions coopérantes. Ces commandes sont de deux types : exécution et dialogue.

Commandes d'exécution :

EXEC : (nom de site, nom d'agent, paramètres)
demande d'exécution d'un agent sur un certain site pour exécuter un programme avec des paramètre initiaux. Ceci correspond à l'initialisation d'une transaction (activation de l'agent initial).

INIT : (nom de site, nom d'agent, paramètres)
Demande d'activation d'un agent (autre que l'agent initial d'une transaction) à distance ou local pour exécuter un programme avec des paramètres initiaux.
Cette commande n'est pas bloquante pour l'agent invocateur autorisant ainsi les traitements parallèles à l'intérieur d'une transaction.

FIN : Signal de fin d'exécution d'un agent.
L'agent demande l'exécution de la procédure de validation globale selon le protocole défini plus loin (2.4.4).

Commandes de dialogue :

ENVOI : (identification d'agent récepteur, texte).
Envoi d'un message (texte) à un agent distant.
Cette commande n'est pas bloquante et suppose que l'agent destinataire a déjà été activé.

RECEVOIR : (identification d'agent émetteur).
Attente d'un message en provenance d'un agent distant.
Cette commande est bloquante jusqu'à la réception d'un message émis par une commande ENVOI de l'agent cité.

Entre les commandes INIT et EXEC qui demandent toutes les deux l'activation d'un agent, il existe une différence fondamentale. EXEC

lance l'agent initial d'une transaction, c'est à dire qu'elle initialise une unité d'exécution cohérente. INIT lance un agent quelconque d'une transaction qui s'insère dans une unité de cohérence déjà initialisée.

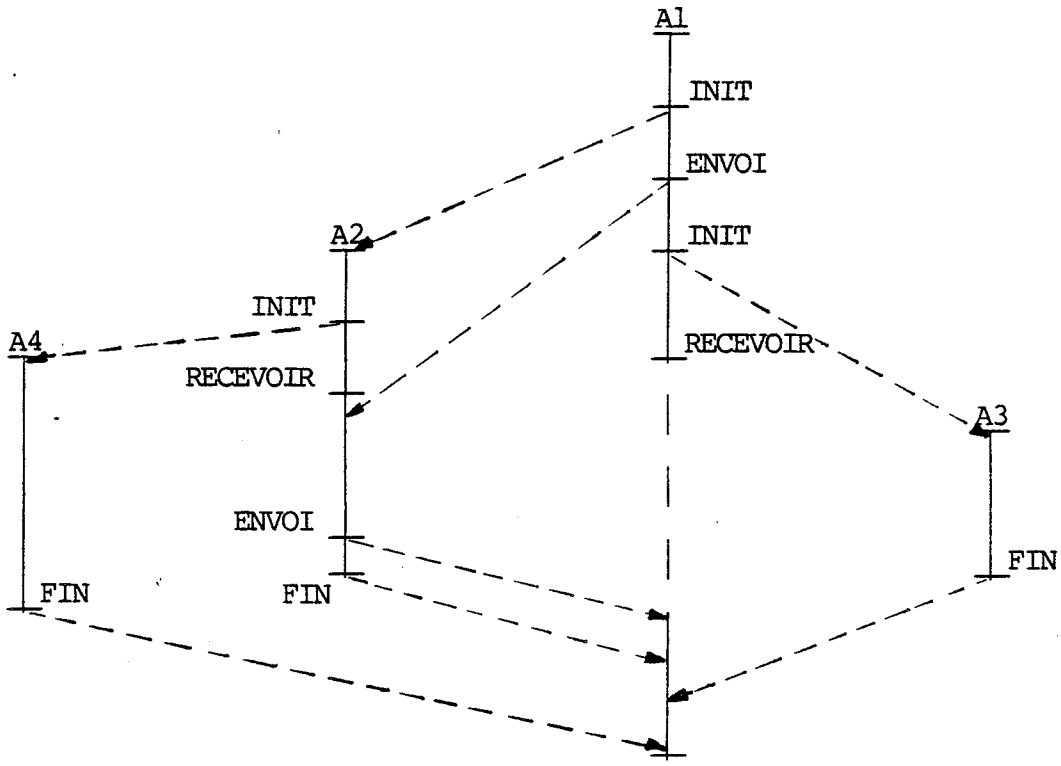


Figure 2.1. Exemple de transaction

Les systèmes transactionnels de chaque site coopèrent, par des échanges de messages, pour assurer les services décrits ci-dessus. La coopération des systèmes transactionnels se fait selon des protocoles du niveau "application" au sens de l'ISO [ISO 79] ou de DSA (Distributed System Architecture). Les protocoles SCOT sont classés en quatre catégories :

- Le protocole d'exécution répartie qui régit le lancement d'un agent à distance et la synchronisation entre agents.

- Le protocole de contrôle d'accès concurrent qui contrôle l'accès simultané de par des transactions à des données partagées.

- Le protocole de validation globale qui assure l'atomicité des transactions.

- Le protocole de reprise après panne qui, complément indispensable du protocole de validation, permet de synchroniser les actions des systèmes transactionnels après une panne d'un site ou remise en état du réseau de communication.

2.4.2. Protocole d'exécution répartie [SCOT9]

Le protocole d'exécution répartie contrôle le lancement et la synchronisation des agents coopérants. Il y a trois types de relations entre les différents agents d'une même transaction,

l'arbre d'invocation (ou de filiation),
le schéma d'exécution,
le schéma de validation.

Nous avons choisi de permettre à chacune de ces relations de s'établir de manière indépendante pour être aussi efficace que possible. D'autres approches imposent aux schémas d'exécution et de validation d'utiliser le chemin établi par l'arbre d'invocation [COLL79].

Le protocole d'exécution répartie établit l'arbre d'invocation et le schéma d'exécution alors que le schéma de validation est établi par le protocole de validation. Ces deux schémas peuvent être différents selon les accès faits aux données par les différents agents (des agents qui ne font que des accès en lecture à des données apparaissent dans le schéma d'exécution mais pas dans le schéma de validation).

Alors que le schéma d'exécution suit simplement l'arbre d'invocation et les échanges de messages, dans le schéma de validation, la connaissance par chaque agent de l'agent contrôlant le processus de validation entraine des échanges directs entre tous les agents et celui-ci en particulier.

- L'arbre d'invocation.

La commande INIT provoque le lancement d'un agent sur un système distant, pour le compte d'une transaction. L'agent émetteur de cette commande continue son exécution en parallèle avec l'exécution de l'agent invoqué. On a donc la possibilité d'exécuter plusieurs agents en parallèle pour le compte d'une même transaction. Des paramètres initiaux peuvent être passés par cette commande.

Un agent invoqué peut à son tour lancer d'autres agents. On construit ainsi un arbre dont la racine est l'agent initial (celui qui a été lancé par une commande EXEC).

Nous montrerons aux paragraphes 5.4.2 et 5.4.3 comment ces deux commandes peuvent être utilisées par le concepteur d'application et en particulier comment il est possible d'utiliser une commande EXEC au cours de l'exécution d'un agent.

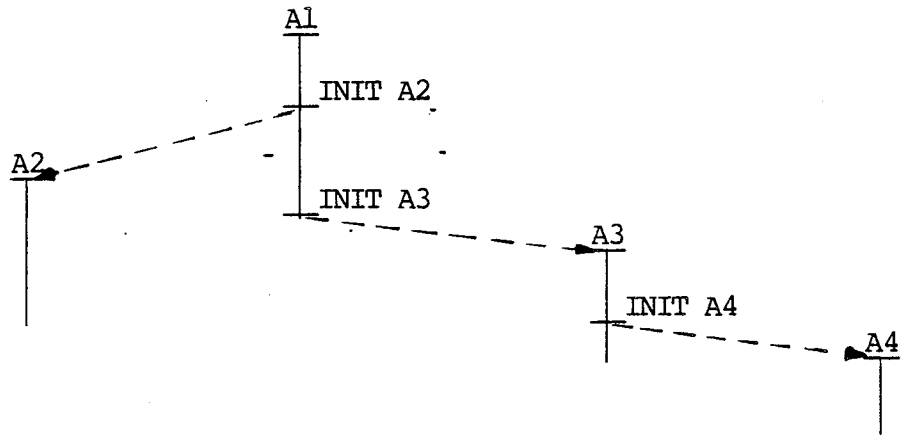


Figure 2.2 Exemple d'arbre d'invocation

- **L'adressage :**

Une transaction est nommée de manière unique par tous les systèmes transactionnels à l'aide d'un identificateur de transaction. Cet identificateur est créé dynamiquement au lancement de la transaction initiale : c'est l'estampille (timestamp) de la transaction. Une estampille est composée de deux parties :

- La valeur d'un compteur local incrémenté lors d'événements significatifs tels que l'initialisation d'une transaction ou d'un agent à distance (poids forts de l'estampille).

- Un numéro de site (poids faibles) destiné à assurer l'unicité des estampilles. En effet, deux sites peuvent parfois engendrer des valeurs de compteurs identiques.

La méthode de Lamport [LAMP78] est utilisée pour synchroniser, en environnement réparti, les compteurs locaux. Cette synchronisation permet de ne pas privilégier un site dans la génération de estampilles. Cette propriété est importante car l'estampille est également utilisée dans le contrôle d'accès concurrent et la reprise après panne (cf. 2.4.3 et 2.4.5).

A l'intérieur d'une transaction, un agent est désigné par un identificateur constitué par le couple :

- estampille courante sur le site initiateur au moment de la prise en compte de la commande INIT,

- numéro de site sur lequel doit s'exécuter l'agent invoqué.

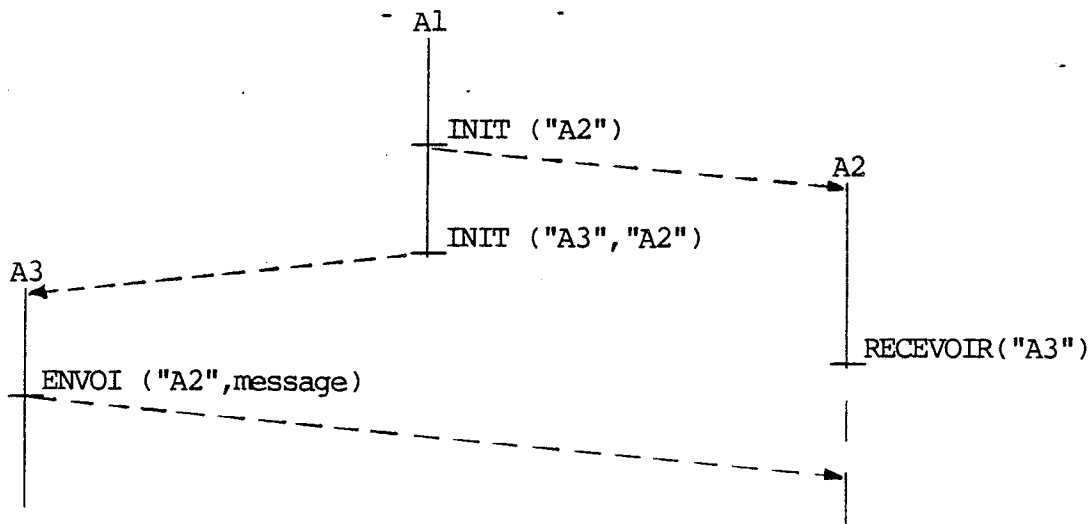
Ce choix permet d'identifier de manière unique un agent avant son lancement. Un agent connait donc l'identité de ses fils sans message en retour des systèmes transactionnels qui en contrôlent l'exécution.

- Le schéma d'exécution.

Contrairement à l'arbre d'invocation qui est strictement hiérarchique, le schéma d'exécution peut être représenté par un graphe quelconque. En effet, tout agent peut communiquer avec tous les autres agents de la même transaction, dans la mesure où il en connait les identificateurs. Cette communication s'établit par échange de messages (commandes ENVOI et RECEVOIR).

Avec la technique d'adressage décrite précédemment, un agent peut démarrer deux autres agents et les mettre en rapport en leur communiquant leur identité respective (schéma d'exécution triangulaire, cf figure 2.3).

L'agent émetteur n'attend pas que le récepteur ait reçu et traité le message pour continuer son exécution. Cette technique évite les acquittements de niveau application (comme ceux du "rendez-vous" du langage ADA) et permet, de plus, de construire des schémas non procéduraux avec les synchronisations les plus complexes.



A2 : Nom de programme
"A2" : Identification interne d'agent.

Figure 2.3. Exemple de schéma d'exécution triangulaire.

2.4.3. Protocole de contrôle d'accès concurrent. [SCOT12]

L'objectif du "contrôle d'accès concurrent" est de permettre l'exécution parallèle de plusieurs transactions en leur garantissant un fonctionnement équivalent à celui d'une exécution séquentielle (propriété de sérialisabilité). On trouve dans la littérature un grand choix de techniques pour effectuer ce contrôle. Nous avons choisi une approche fondée sur le verrouillage des ressources et la prévention des interblocages [ROSE78] [BERN79] [STON78].

Sur chaque site, les agents acquièrent dynamiquement des ressources. L'accès à ces ressources est contrôlé à l'aide d'un verrou. On distingue deux types de verrous : les verrous "partagés" autorisent l'accès simultané à une ressource ; les verrous "exclusifs" garantissent l'accès à une ressource par une seule transaction. Une ressource est vérouillée jusqu'à la phase de validation de la transaction. L'acquisition dynamique de verrous et la possibilité de transformer un verrou "partagé" en verrou "exclusif" introduisent la création eventuelle d'interblocages (deadlock) entre les transactions. Un interblocage est provoqué par une chaine de transactions globales T_i en attente telles que :

$$T_0 \longrightarrow T_1 \longrightarrow T_2 \ \ldots \ \longrightarrow T_n \longrightarrow T_0$$

La transaction T_i attend la transaction T_j ($T_i \longrightarrow T_j$) si au moins l'un des agents de T_i attend une ressource détenue par un agent de T_j.

L'algorithme de contrôle d'accès concurrent de SCOT empêche la formation de boucles dans les chaines d'attente en utilisant les estampilles. On a vu en 2.4.1 qu'une transaction possède une estampille unique dans tout le système. Chaque agent a l'estampille de sa transaction. L'estampillage introduit une relation d'ordre entre les transactions. Cette relation d'ordre est très proche de l'ordre de lancement des transactions (l'âge d'une transaction globale est inversement proportionnel à la valeur de son estampille).

L'algorithme retenu est dérivé du "WOUND-WAIT" proposé par Rosenkrantz et al. [ROSE78]. Il peut s'exprimer de la manière suivante :

Une transaction qui entre en conflit avec une autre transaction attend la libération de la ressource détenue par sa concurrente si elle est plus jeune, ou tue sa concurrente si elle est plus ancienne. Le système assure qu'il n'y a pas de conflit entre deux agents d'une même transaction. Ceci signifie que si deux agents ayant même estampille accèdent à la même donnée, on exécute le second accès sans attente. C'est au concepteur d'application de s'assurer que ces accès multiples sont bien corrects. Une autre solution consiste à interdire l'accès à une donnée par plusieurs agents d'une même transaction, considérant que l'accès multiple est une erreur de conception puisque les données peuvent être échangés par des messages.

D'une façon plus formelle, on peut également l'énoncer ainsi :

la transaction Tj (estampille tj) possède une ressource R,
la transaction ti (estampille ti) demmande cette ressource,

si ti < tj alors Tj est abandonnée,
sinon Ti attend la terminaison de Tj.

Ainsi, tout au long d'une chaine d'attente, les estampilles sont toujours strictement décroissantes, ce qui interdit la formation de boucles. Cet algorithme prévient donc la formation d'interblocages en éliminant les transactions suceptibles de les provoquer.

Une transaction abandonnée est redémarrée ultérieurement avec la même estampille. Cette méthode assure à une transaction abandonnée un "vieillissement" qui augmente ainsi ses chances de succès. Cet algorithme n'est donc pas sujet à la famine. En effet, une transaction est certaine de s'exécuter (d'obtenir une certaine ressource) lorsque toutes les transactions d'estampilles inférieures (plus anciennes) seront terminées.

On reproche généralement à cette méthode d'abandonner à tort des transactions qui ne sont pas réellement impliquées dans un interblocage. C'est pourquoi de nombreuses améliorations, visant à réduire le taux d'abandons, ont été proposées. En fait, les évaluations menées sur des modèles simulés ont montré que ces abandons préventifs, si leur fréquence demeure raisonnable, peuvent avoir un effet bénéfique [SCOT14]. En effet, en éliminant quelques transactions du groupe de transactions en concurrence, la prévention des interblocages diminue le taux de multiprogrammation effectif. Cette autorégu-

lation prend tout son sens quand le taux de conflits devient trop grand et évite ainsi un écroulement du système. On retrouve ici un résultat bien connu dans le cas des systèmes paginés, la prévention de l'écroulement du système (ou "trashing") par limitation du degré de multiprogrammation.

Ce mécanisme de contrôle d'accès concurrent présente aussi l'avantage de préserver l'autonomie des systèmes transactionnels locaux. Le protocole correspondant se limite à l'échange de messages d'abandon pour tuer la filiation d'une transaction et de reprise pour demander le redémarrage d'une transaction.

2.4.4. Protocole de validation globale [SCOT14]

Par opposition au précédent, ce protocole nécessite une coopération importante des systèmes transactionnels.

La technique adoptée est une procédure de validation (commitment) mixte dans laquelle cohabitent les techniques généralement présentées sous l'appellation validation à "1-phase" et validation à "2-phases" [GRAY78, LIND79].

Quelle que soit la technique utilisée, le processus de validation est contrôlé par un agent privilégié, appelé "supérieur". Les validations des autres agents, appelés "inférieurs", sont subordonnées à la validation du supérieur. Le choix du supérieur peut être négocié lors du lancement des agents ou décidé une fois pour toutes. Dans le prototype SCOT, c'est la seconde solution qui a été adoptée, le supérieur étant l'agent initial. En effet, cet agent est en liaison avec l'utilisateur terminal qui pourra, dans tous les cas, être informé du déroulement de la validation.

Pour garantir l'atomicité des transactions contre les effets des pannes de site ou du réseau, les techniques de validation (commitment) sont fondées sur l'utilisation de journaux. Chaque système écrit sur des journaux des informations qui, en cas de redémarrage après panne, permettront de reprendre et terminer des transactions en cours de validation, ou, au contraire, de défaire des actions effectuées par des transactions partiellement exécutées.

Le principe de la validation est le suivant :

- un agent inférieur qui a terminé son exécution envoie un message de terminaison au supérieur. Il entre dans l'état "prêt-à-valider",

- lorsqu'il a reçu tous les messages de terminaison, le supérieur valide localement et diffuse un ordre de validation aux inférieurs,

- Un agent inférieur qui reçoit ce message valide localement (on notera qu'il n'y a pas d'accusé de réception envoyé au supérieur).

L'état "prêt-à-valider", qui caractérise les agents inférieurs, est une période critique (appelée zone grise), pendant laquelle le système local ne peut pas prendre unilatéralement la décision de valider ou d'abandonner un agent. En effet, en l'absence de message du supérieur, un agent inférieur n'a aucun moyen de savoir si le supérieur a passé on non le point de validation. Une panne du supérieur ou du réseau, qui isole le site inférieur du site supérieur, laisse les agents inférieurs dans l'incertitude quant à l'issue de la transaction. Le dialogue avec d'autres agents inférieurs de la même transaction permet, dans certains cas, de lever l'ambiguïté. En règle générale, il est nécessaire d'attendre la reconnexion du supérieur pour conclure. Pendant cette attente, les ressources détenues par les agents inférieurs restent bloquées, à moins d'exécuter une intervention manuelle.

Aucun algorithme ne permet de prévenir totalement cette situation. Il est seulement possible de réduire la durée de la zone grise pour minimiser ce risque. C'est le rôle du protocole à 2-phases. On peut cependant noter que la négociation du supérieur peut également minimiser le risque d'attente en désignant comme supérieur l'agent qui détient les ressources critiques pour l'application.

En utilisant SCOT, le concepteur a, pour un agent inférieur, le choix entre trois stratégies :

- Pas de validation (figure 2.4.a) :

  envoi d'une commande "fin sans validation" indiquant au supérieur qu'il n'y a pas d'autre phase au processus de validation.

- Validation à une phase (figure 2.4.b)

  Envoi d'un message "prêt à valider" au supérieur. Le supérieur diffuse le message "valider" lorsqu'il a reçu tous les messages "prêt à valider".

- Validation à deux phases (figure 2.4.c) :

  envoi d'une commande "fin" au supérieur. Quand le supérieur a reçu toutes les commandes de terminaison, il demande aux inférieurs de se préparer à valider (message "pré-valider").
  Les inférieurs acquittent cette commande en renvoyant au supérieur le message "prêt à valider". Comme dans le cas précédent, le supérieur valide localement et diffuse l'ordre de validation quand il a reçu tous les messages "prêt à valider".

  Cette méthode exige un aller-retour de plus mais elle permet de synchroniser et, selon les scénarios, de raccourcir les zones grises.

L'étude de différents scénarios d'application montre que :

- Le protocole de validation à 0-phase (pas de validation) est indispensable pour les applications qui ne requièrent aucune garantie de cohérence ou qui manipulent des données en "lecture seule".

- Certaines applications sont très pénalisées par le protocole à 2-phases, en particulier celles qui ont un schéma d'exécution très simple (par exemple, les transactions constituées de deux agents) pour lesquelles l'utilisation du protocole à 1-phase est bien adapté.

- Les applications qui requièrent un grand degré de disponibilité des ressources nécessitent l'utilisation du protocole à 2-phases.

Nous pensons que les trois protocoles sont nécessaires, le choix de l'un d'entre eux étant un compromis entre le coût supplémentaire induit par une synchronisation plus sophistiquée et une meilleure disponibilité des ressources.

Le protocole de validation globale proposé dans SCOT :

- fournit un mécanisme général dans lequel les trois méthodes cohabitent,
- Laisse le choix de la technique appropriée au programmeur d'application.

Ce protocole de validation mixte fonctionne de la manière suivante :

- Chaque agent de voit affecter un type de protocole par le concepteur d'application,
- Cette information est transmise au supérieur en paramètre du message de terminaison,
- Le supérieur procède ensuite avec chaque agent inférieur selon le protocole indiqué.

Le protocole de validation est complété par un protocole de traitement des erreurs mis en oeuvre lorsque l'exécution d'une transaction est interrompue par une erreur sémantique. Dans ce cas, une demande d'abandon est diffusée à tous les agents de la transaction et un message explicitant la cause et l'origine de l'erreur est envoyé à l'initiateur de la transaction.

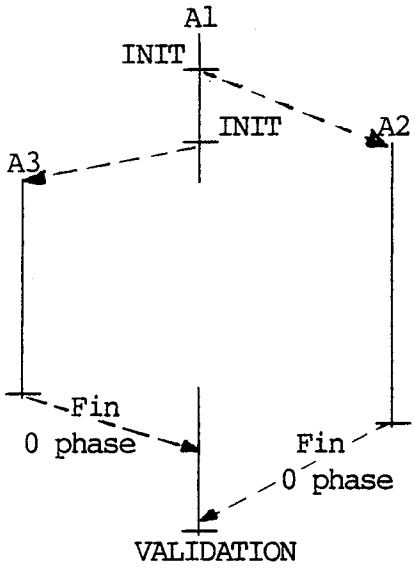


figure 2.4.a.

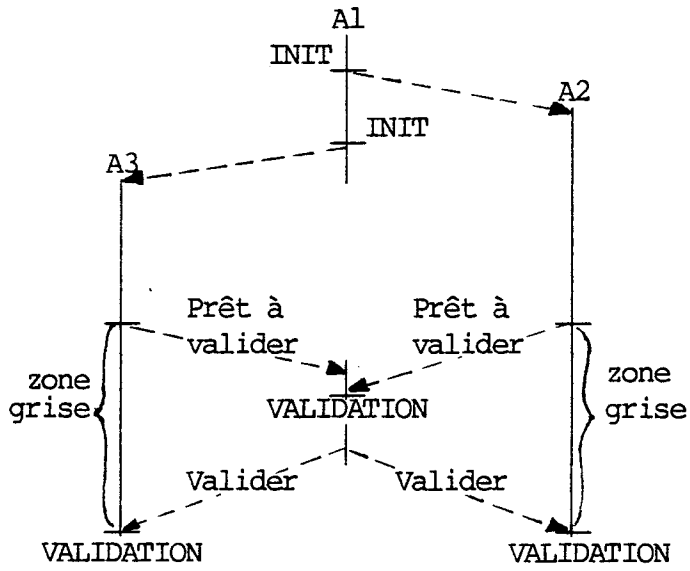


figure 2.4.b.

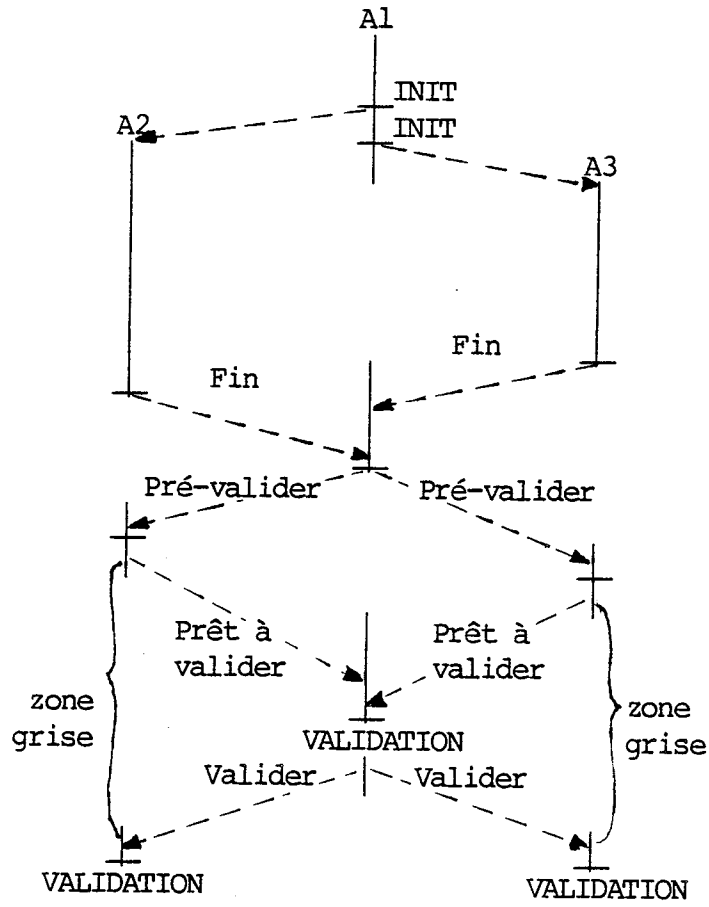


figure 2.4.c.

2.4.5. Protocole de reprise [SCOT9, SCOT14]

La cohérence des données distribuées dépend en grande partie de la capacité des systèmes coopérants à prendre la même décision pour tous les agents d'une même transaction. Cette coordination est assurée par :

- Le protocole de validation (cf 2.4.4) lorsque la transaction se termine normalement,

- Le protocole de traitement des erreurs lorsque l'exécution de la transaction est interrompue par une erreur sémantique,

- Le protocole de reprise, après une panne d'un site ou du réseau de communication.

Un système transactionnel exécute une procédure de reprise dans les trois cas suivants :

- Détection d'un incident affectant un système transactionnel coopérant. (Par incident on entend panne d'un site ou du réseau). Le système local abandonne tous les agents qui coopéraient avec un agent du site disparu, à l'exception de ceux qui étaient dans l'état "prêt à valider", pour lesquels une information complémentaire est nécessaire.

- Reconnexion d'un système (les communications avec lui sont rétablies). Si ce système est susceptible de détenir des informations utiles à la terminaison d'un agent local, il est interrogé (procédure d'enquête sur l'issue d'une transaction.

- Redémarrage après une panne du système local. Le système s'efforce de trouver une issue à tous les agents qui s'exécutaient au moment de la panne. Le journal local est utilisé pour déterminer le traitement propre à chaque agent.

En fonction de l'état de chaque agent, un système local peut, soit décider seul de l'issue à lui donner (validation ou avortement), soit avoir besoin de l'avis d'un système coopérant (pour les agents dans l'état "prêt à valider").

Ce protocole d'accord nécessite la réalisation :

- d'une procédure de surveillance destinée à détecter la reconnexion d'un système transactionnel disparu. Cette procédure peut être réalisée au niveau de chaque système transactionnel ou au niveau session.

- d'une procédure d'enquête destinée à connaître la décision prise par le supérieur pendant une panne.

La procédure d'enquête fonctionne de la manière suivante : dans l'état "prêt à valider" et sans nouvelles du supérieur, un agent inférieur diffuse aux partenaires une commande ENQUETE accompagnée de l'estampille de la transaction. Si au moins un partenaire répond par

un message VALIDER, l'agent local est validé. Si au moins un des partenaires répond par un message AVORTER, l'agent est avorté. Si personne ne répond ou si toutes les réponses font état de partenaires dans la zone grise, l'agent local est suspendu en attente de la reconnexion du supérieur. Cette procédure permet, dans certains cas, de trouver une solution à un agent inférieur isolé du supérieur (par exemple, lorsqu'un des agents inférieurs a reçu le message valider tandis qu'un second ne l'a pas reçu).

2.4.6. Support de la coopération [SCOT23]

L'architecture des systèmes ouverts de l'ISO [ISO 79] incite à ouvrir une session par paire d'entités communiquantes. Il y a une correspondance bi-univoque entre une session et une paire d'agents coopérants. Mais, dans le cas du transactionnel coopérant, contrairement au transfert de fichier ou du traitement par lots, la durée du dialogue inter-agents est très brève. Il y a de nombreux établissements de connexions pour peu de messages échangés. Il faut donc veiller à ce que la charge système (en messages et en temps de traitement) induite par le lancement et l'arrêt d'un agent soit faible.

En conséquence, nous avons adopté dans SCOT une méthode qui consiste à multiplexer les messages des différents protocoles sur des sessions bi-directionnelles, établies de manière durable entre les systèmes transactionnels eux-mêmes. On évite ainsi l'allocation et la désallocation d'une session à chaque démarrage-arrêt d'un agent. L'adressage d'une entité coopérante utilise l'identificateur d'agent présenté en 2.4.2. La seconde partie de cet identificateur désigne la session à utiliser, la première partie identifie l'agent auquel le message est destiné et sert au démultiplexage des messages sur chaque site.

La prise en compte du critère "performance" dans la définition descendante de l'architecture du prototype SCOT nous a amenés à remettre en cause les services offerts par le niveau session défini par l'ISO et à proposer une session dédiée au transactionnel coopérant. Les fonctionnalités de cette session sont très élémentaires et orientées vers un mode "sans connexion".

2.4.7. Conclusion - liens application / système

Nous avons présenté l'ensemble des techniques utilisées dans le système SCOT. Ceci nous a permis de montrer que l'étude de scénarios d'application a influencé les choix faits pour les protocoles (en particulier pour la validation), mais aussi comment le concepteur d'application peut intervenir sur le choix d'un protocole de validation pour chaque agent d'une transaction. Ceci montre bien que la réalisation d'une application sur un système transactionnel coopérant donné tient compte des particularités de ce système. Les liens entre l'application et le système sont donc assez importants.

CHAPITRE 3

CONCEPTION D'APPLICATIONS - SPECIFICATION

3. CONCEPTION D'APPLICATION - SPECIFICATION

3.1. Présentation des problèmes

Avant de réaliser un système (on ne parle ici que des systèmes informatiques), il est nécessaire d'en analyser tous les mécanismes (internes) et toutes ses relations avec le monde extérieur. Pour des systèmes simples dont la conception ne met en jeu qu'une ou au plus quelques personnes, trop souvent cette connaissance reste dans la tête des concepteurs ou des réalisateurs. Bien qu'il ne semble pas absolument nécessaire de disposer d'une description écrite et détaillée, celle-ci peut être très utile pour la maintenance ou l'évolution du programme.

Pour les systèmes de grande taille mettant en jeu plusieurs dizaines, voire plusieurs centaines de personnes travaillant pendant plusieurs années à différents niveaux, une description précise, détaillée, non ambiguë est indispensable à tous les stades de la conception, aussi bien pour la description des besoins des usagers, que pour l'architecture du système ou sa réalisation.

Il ne s'agit pas ici de faire un état de l'art des méthodes de spécification. Il existe trop de méthodes, qu'elles soient générales ou particulières (adaptées à un type d'application bien défini). Ce que nous essayons de montrer dans ce chapitre, c'est le chemin que nous avons suivi dans ce domaine, au travers de toutes ces techniques, en insistant sur celles qui ont retenu notre attention et que nous avons le plus étudié.

En aucun cas, il ne s'agit d'un jugement de valeur, pour ou contre les méthodes retenues ou non. Notre démarche nous conduit d'études

sur les bases de données réparties [POLY79] à la conception d'applications devant être réalisées au moyen de ces même bases. Le critère essentiel ayant aiguillé notre démarche est l'adéquation de la méthode aux problèmes posés par les applications réparties.

Avant d'examiner les méthodes existantes, il est utile de préciser à quel type de système d'information on s'est intéressé. Il s'agit des grands systèmes d'information des administrations tels que banques, assurances, de gestion de stocks, etc. Dans ce qui suit, on gardera à l'esprit les caractères spécifiques de ces systèmes, en particulier, la taille, la complexité, la durée de vie et surtout la cohérence des données qu'ils ont à gérer. Il est bien évident que tous les systèmes d'information n'ont pas besoin du même degré de cohérence. C'est à chaque système et à chaque application de déterminer le niveau de cohérence souhaité.

3.2. Des méthodes de spécification

3.2.1. Différentes approches

Sans vouloir classifier à tout prix, on peut distinguer deux grandes tendances ou classes de méthodologies. La première est fondée sur les modèles de description de données (schéma conceptuel), la seconde a priori plus dynamique, est fondée sur les types abstraits de données (abstract data types).

Cependant, dans tous les cas, les problèmes peuvent être abordés de manière ascendante (bottom-up) ou descendante (top-down). Dans une méthode ascendante, on commence par la description des éléments les plus fins, puis en les composant progressivement, on trouve les éléments les plus construits de l'application. Inversement, une méthode descendante permet, en raffinant les objets du plus haut niveau, de décrire les objets de base de l'application.

En fait, la spécification d'une application est rarement faite de manière complètement ascendante ou descendante, mais partant des objets les mieux connus ou perçus intuitivement, on peut préciser en raffinant, ou les composer de manière à construire d'autres objets de l'application.

3.2.2. Qu'est-ce qu'une spécification?

- C'est une description des fonctions, indépendante des choix faits pour sa réalisation [WEBE78].

- Spécifier un système c'est faire une description formelle qui peut être traitée par un automate [ROLL81].

- L'objectif d'une spécification est d'offrir une description mathématique d'un concept, la preuve du programme étant établie par l'équivalence avec la spécification [LISK75].

Pour synthétiser ces définitions, nous dirons qu'une spécification est une description formelle, précise, non ambiguë d'une machine à réaliser. C'est un énoncé des caractéristiques (propriétés) tant statiques que dynamiques qui permet la vérification et la réalisation.

On peut ajouter que c'est un moyen de communication entre le concepteur et le réalisateur. On aura différents niveaux de spécification suivant qu'on cherche à exprimer les besoins de l'utilisateur final ou du réalisateur (informatique), ou suivant le degré de liberté laissé au réalisateur (choix techniques). Toutes les études distinguent plusieurs phases de spécification au cours du processus de conception.

Pour [WILE79], en ce qui concerne le développement de logiciel, on aura une succession de descriptions de plus en plus détaillées, chacune d'elles ayant une orientation, un vocabulaire, des intérêts propres.

Il distingue :

- La spécification des besoins (requirements), orientée vers l'utilisateur (souvent appelée cahier des charges); elle est écrite dans le vocabulaire du domaine. Il s'agit essentiellement de définir l'enveloppe du système, c'est à dire son comportement vu de l'extérieur. Quelquefois, il est tenu compte des performances et des contraintes économiques.

- La spécification d'architecture (design) qui est orientée

vers la réalisation. Cette phase consiste à conceptualiser les objets, les fonctions permettant de les manipuler et les interactions entre les objets. Elle s'appuie essentiellement sur la notion de module ou de type abstrait. Ce domaine a donné lieu à de nombreuses recherches.

- La spécification de réalisation qui est orientée vers l'exécution et qui constitue la dernière phase avant la programmation. C'est dans cette étape que sont effectués la plupart des choix techniques qui seront associés à la réalisation.

Pour [ROLL81], le processus de conception de systèmes d'information comprend deux phases :

- D'abord la modélisation du monde réel avec des concepts et outils théoriques (un schéma conceptuel).

- Puis l'expression des aspects techniques et de réalisation dans un langage formel.

Ces deux approches des problèmes sont trés différentes, cependant on notera deux points de convergence. Une des phases de la conception est faite en dehors de tout contexte de réalisation. La méthode de spécification comporte un langage qui permet des manipulations, transformations et vérifications.

Toutes les études récentes sur ce sujet convergent sur un autre point important [WILE79, GUTT80, ROLL81, BROD81, WEBE78]. Dans la description d'une application, on ne peut pas séparer la description des entités ou données, des opérations ou actions qui permettent de les manipuler. Les données représentent l'aspect statique (permanent) du système, c'est à dire son état observable. Les opérations, ou actions, ou programmes, expriment les transitions entre les différents états possibles. La relativité de ces deux descriptions s'exprime par le fait que l'état, à un instant donné, est le résultat de l'enchaînement des opérations depuis l'état initial.

A chaque approche correspond un certain nombre d'outils dont au moins un langage de description quelquefois associé à une

méthodologie, celle-ci ayant d'autant plus de chances d'être utilisée qu'elle est supportée par un langage [LISK77].

Nous allons maintenant examiner plusieurs approches différentes, les comparer et les confronter à nos objectifs.

3.3. Méthodes fondées sur les modèles de description de données

3.3.1. Généralités

Le concept de base de données a été le premier pas vers l'expression de la cohérence et du contrôle de celle-ci par un système plutôt que par l'utilisateur (dans les programmes). Dans une base de données, on exprime la sémantique des données au travers d'un modèle (hiérarchique, réseau, relationnel, ...), la manipulation de ces mêmes données étant réalisée par l'utilisation d'un langage mettant en oeuvre les méta-entités du modèle (record, owner dans le modèle CODASYL ; relation, attribut, domaine dans le modèle relationnel ; entité et association dans le modèle E-R).

Afin de décrire des applications plus complexes, beaucoup d'efforts ont été faits pour élaborer des modèles de description de données de plus en plus sophistiqués avec une sémantique de plus en plus riche. Par exemple, la notion de rôle joué par une entité ou un attribut d'une entité [BACH78], [CODD79], le concept d'association, etc.

Le modèle offre, au moyen du schéma, un outil pour décrire un système d'information sous la forme d'un ensemble de données dont les valeurs définisssent à chaque instant l'état du système. Des contraintes d'intégrité peuvent exprimer des restrictions sur les états ou les changements d'états possibles, et par là, enrichir la sémantique de la description.

Par contre, les propriétés liées à l'évolution du système (la dynamique), qui s'expriment par des restrictions sur les transitions entre les états, sont d'une expression peu naturelle dans les modèles de données qui sont, par contre, bien adaptés à la description des propriétés statiques (états).

Quelques tentatives ont été faites pour prendre en compte les aspects dynamiques de la gestion des données :

- en introduisant le modèle relationnel (concept de relation et l'algèbre relationnelle) dans le langage PASCAL (PLAIN [WASS79], ou ASTRAL) ou un modèle fonctionnel (DAPLEX) dans le langage ADA (ADAPLEX [SMIT81]).

- en offrant un nouveau modèle comprenant des concepts et des outils (langage) de description et de prise en compte des aspects dynamiques.

Cette dernière approche, davantage orientée vers la conception que vers la réalisation est largement décrite dans [ROLL79, ROLL81a, b, c]. Nous allons la décrire pour en rechercher les avantages et inconvénients.

3.3.2. Analyse d'une méthode

La méthode proposée est une approche qui se veut structurale et qui conduit à une description formelle appelée schéma conceptuel. On entend par schéma conceptuel une représentation unique et complète des structures statiques (composants) et dynamiques (inter-relations et transformations des structures).

Les outils associés à cette méthode de conception sont :

- Un modèle, c'est à dire un jeu de concepts et de règle associées, pour structurer,

- Un langage pour décrire la modélisation.

- Le modèle conceptuel

Il est fondé sur le modèle relationnel en tant que formalisme mais également comme langage de description. Deux autres formalismes de description sont utilisés, l'un graphique, l'autre langage. Ce modèle intègre des aspects statiques et dynamiques. On peut le représenter par le schéma de la figure 3.1.

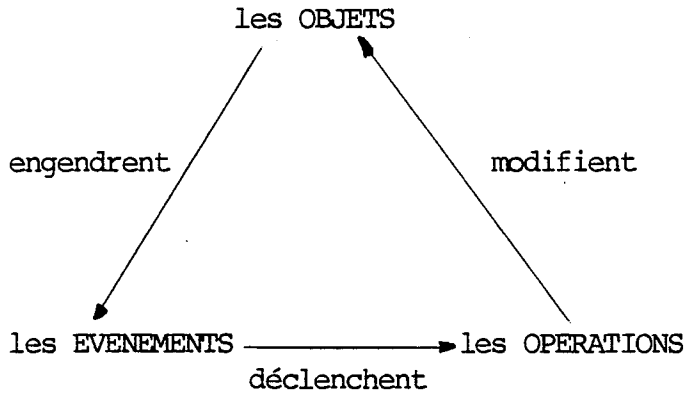


Figure 3.1. Le modèle conceptuel.

Aux méta-entités du modèle :

C-OBJET, C-EVENEMENT, C-OPERATION (C- pour concept)

sont associées des représentations qui permettent de modéliser le système d'information sous forme d'un graphe.

Un C-OBJET définit l'aspect temporel d'une classe d'objets réels, c'est à dire l'ensemble maximal des propriétés d'un objet ayant un comportement dynamique identique,

Un C-EVENEMENT définit un changement d'état du système d'information et engendre une ou plusieurs transformations des objets en déclenchant des C-opérations éventuellement conditionnelles et/ou interactives,

Une C-OPERATION est une modification élémentaire du système d'information c'est à dire qu'elle ne porte que sur un seul C-OBJET.

Exemple :

L'opération bancaire de virement d'un compte à un autre peut être décrite par le schéma suivant :

```
C-OBJETS : VIREMENT (ref_virement, comptel, compte2, montant);
           COMPTE (num_de_c, solde,...);

C-EVENEMENT : ARRIVEE-VIREMENT (ref_virement);

C-OPERATIONS : CREDITER (#cred, num_de_c, montant);
               DEBITER  (#deb, num_de_c, montant);
               ANNULER (#annul, ref_virement);

CONDITIONS :(C1)   si VIREMENT.montant > compte2.montant alors
                      ANNULER (VIREMENT).
```

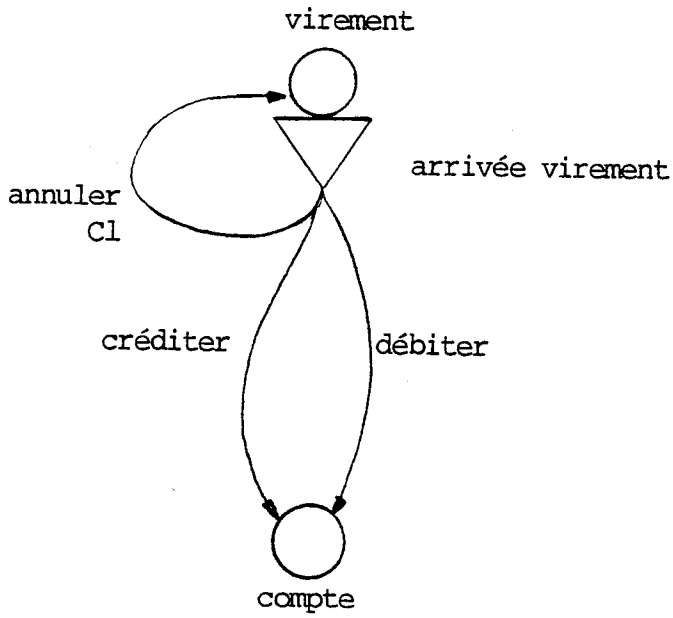


Figure 3.2. Schéma conceptuel d'un virement.

La deuxième phase de conception consiste à exprimer le schéma ainsi défini dans un langage (ISDEL) proche de PASCAL associé à SEQUEL. Ce langage permet d'une part, la réalisation des relations et d'autre part, la génération de modules de traitement à partir des expressions assertionnelles provenant des conditions et contraintes d'intégrité exprimées sur le modèle conceptuel.

Cette étape permet de rapprocher la spécification très près de la réalisation du système d'information.

- Avantages et inconvénients

Si dans la première phase, le modèle offre un bon support de réflexion, souple, permettant une expression aussi bien formelle qu'informelle, le passage à la deuxième phase ne s'opère pas par transformation de la première expression, provoquant ainsi une discontinuité dans le processus de spécification.

On dipose d'un méta-modèle utilisable pour toutes les spécifications. Il n'est donc pas nécessaire de redéfinir un nouvel environnement pour décrire chaque système d'information. Ce méta-modèle est fondé sur un modèle théorique bien connu (le relationnel) et bénéficie donc de tout l'apport (algèbre relationnelle) de ce modèle. Il offre des possibilités de démonstration et de vérification de la spécification. De plus, ce nouveau modèle apporte la prise en compte des aspects dynamiques des applications (événements, opérations) au niveau du modèle, sur le même plan que la description des structures (objets).

Dans les exemples donnés dans [ROLL81, ROLL82], on constate que l'utilisation de diagrammes (schémas) pour décrire un système est très vite limitée par la complexité qui rend le schéma difficile à maîtriser et à lire. On retrouve ici la même difficulté que celle rencontrée dans SCOT pour l'expression des protocoles en termes de réseaux de Nutt [SCOT 9]. En particulier, il est difficile de cerner ce qui est lié à un même objet et donc d'en définir la sémantique précise et complète. Ceci est sans doute dû à l'utilisation d'un modèle plus causal que structural (les événements engendrent les opérations sur les objets).

Il est, par ailleurs, difficile de procéder par raffinements successifs et par exemple, seule la connaissance de tous les C-OBJETS permet de définir les événements et opérations associées. Il est cependant possible de décrire des sous ensembles du système d'information constitués de C-objets ayant un comportement relativement indépendant et d'exprimer ensuite leurs inter-relations.

L'expression du temps est particulièrement difficile puisque rien ne permet, sauf de manière informelle, de définir des notions telles que la séquentialité d'opérations activées par un même événement. Cette difficulté se répercute sur la prise en compte des exceptions (retours arrières en cas d'anomalie). Or le temps et les exceptions sont des éléments essentiels dans la définition des systèmes d'information opérationnels d'une certaine taille.

Pour illustrer ces difficultés, on peut sur l'exemple précédent essayer de décrire la séquentialité des opérations de crédit et de débit, ou bien exprimer que si le débit est impossible (solde insuffisant) alors annuler le crédit.

En fait, le virement décrit dans l'exemple est simplifié et pour être précis, on doit dire que cette opération bancaire se décompose en deux parties :

-1 le débit du compte et la création sur l'autre compte d'une provision (qui n'est pas le crédit effectif) et le crédit d'un compte de la banque.

-2 Après un certain délai (date de valeur), le débit du compte de la banque, l'annulation de la provision et le crédit effectif du compte du client.

L'expression dans ce modèle de "après un certain délai" n'est pas possible. Une tentative d'introduction du concept de C-TRIGGER [ROLL79] n'a pas été retenue par la suite.

3.3.3. Autres expériences

Le langage de spécification Z [ABRI77] fondé sur les mathématiques (ensemblistes) et la logique est un outil puissant pour la description d'applications et la transformation progressive de cette description vers la réalisation.

Un système est décrit en termes des ensembles qui le composent, des fonctions qui relient ces ensembles et des conditions d'application (restrictions) de ces fonctions exprimées sous la forme de prédicats.

Une expérience menée a l'IMAG, nous a permis de construire un schéma conceptuel de bureautique [ABH 79a, b] comme un schéma global représentant la coopération de schéma locaux, chacun d'eux décrivant un service bureautique (traitement de texte, messagerie, classement).

Par exemple, le service de messagerie met en jeu deux ensembles : DOCUMENT et ABONNE qui sont reliés par les fonctions suivantes :

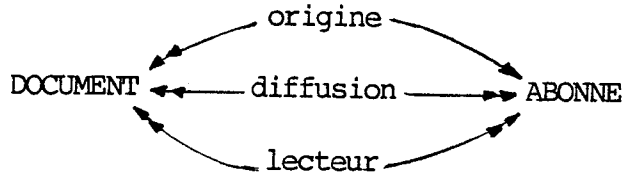
DOCUMENT	origine(1)	ABONNE	définissant l'émetteur du document
DOCUMENT	DIFFUSION(0,-)	ABONNE	définissant la liste des abonnés qui recevront le document
DOCUMENT	LECTEUR(0,-)	ABONNE	indiquant les abonnés de la diffusion qui n'ont pas retiré le document

Les fonctions peuvent être mono-valuées (en minuscules : origine (1) dans l'exemple) ou multi-valuées (en majuscules : DIFFUSION (0,-)) et les valeurs données entre parenthèses indiquent le domaine de valuation. Dans l'exemple, la diffusion d'un document peut comprendre 0 ou un nombre indéterminé d'abonnés.

De la même façon, on pourrait définir les fonctions inverses telles que :

DOCUMENT ——— CORBEILLE(0,-) ABONNE qui est l'ensemble des documents qu'un abonné n'a pas encore lu.

On obtient le schéma conceptuel de messagerie suivant :



Un ensemble d'outils (langage, règles de transformations) permet de définir des prédicats exprimant les conditions sur les fonctions et par transformations successives de celles-ci, de se rapprocher de la réalisation. Dans l'exemple ci dessous, les conditions ont été exprimées au travers des opérations permettant de manipuler chacune de ces notions. Par exemple, un abonné a peut diffuser un document "d" à une liste d'abonnés par l'opération

```
DIFFUSER (a ∈ ABONNE, d ∈ DOCUMENT, LISTE ⊂ ABONNE)
   if origine (d) = a then for x ∈ LISTE
                              DIFFUSION (d)<--- x
                              LECTEUR  (d) <--- x  end
```

De ce schéma conceptuel, on a pu déduire :

- les entités coopérant au sein d'un système bureautique intégré (documents, usagers, descripteurs) dont l'un d'eux - le document - constitue le lien entre toutes les applications,

- les opérations permettant la manipulation de chacune des entités identifiées,

- les protocoles qu'il est nécessaire d'établir pour assurer la coopération de tous les services composant l'application.

Ce schéma a également montré comment il était possible d'introduire un nouveau service utilisant les concepts définis dans le premier schéma global. Ce nouveau service fait apparaitre de nouveaux liens

avec les autres services. Ces liens peuvent alors se décrire dans le nouveau schéma conceptuel global.

En revanche, cette méthode qui offrait un bon outil d'analyse a montré ses limites quant aux possibilités de raffinement de la spécification et à la difficulté à se transformer en réalisation, en particulier à cause du manque d'outils d'aide au concepteur, toutes les transformations successives nécessaires se faisant "à la main".

Il faut noter également la difficulté à exprimer les conditions dans un langage riche dont la sémantique des outils est complexe. Cette technique reste l'apanage de concepteurs - réalisateurs de très haut niveau. En particulier, il semble difficile de donner une telle spécification à un programmeur pour la réaliser.

3.4. Méthodes fondées sur les types abstraits

3.4.1. Présentation générale

Il ne s'agit pas ici de présenter la théorie des types abstraits mais de montrer comment leur utilisation peut nous aider à concevoir les applications. Nous ne retiendrons ici des types abstraits que les aspects description et nous ne nous intéresserons pas aux aspects démonstration et vérification.

[GUTT78] présente un type abstrait comme l'association d'une classe de valeurs et d'une collection d'opérations sur ces valeurs définies sous forme axiomatique. Une représentation des types abstraits algèbriques [GUTT78, GUTT80a] est l'association dans une même unité de description de :

- la définition syntaxique des opérations du type introduisant les données sur lesquelles portent ces opérations (type, domaines de valeur),

- la spécification des propriétés de ces opérations (axiomes écrits sous forme d'equations) définissant la sémantique des objets du type.

D'autres approches des types abstraits se distinguent par la présentation des axiomes et les méthodes de validation des descriptions [HOAR72] [GOGE75].

Les méthodes utilisant les types abstraits sont bien adaptées à la description du comportement puisqu'elles s'appuient sur les notions d'opérateur et de changement d'état plutôt que sur celles de donnée et d'état. L'état d'une donnée peut toujours être évalué à partir de l'état initial et de la séquence d'opérateurs qui lui a été appliquée.

On trouve des applications de ces concepts dans différents domaines. Si la spécification reste le domaine principal [GUTT80b] [BERT79], on trouve des réalisations des types abstraits dans les nouveaux langages de programmation (Clu, Alphard, Typ) [LISK75] [ICHB79] [MINO79]. La réalisation du système SMALLTALK [BYTE81] a mis en oeuvre de façon intensive la notion de type abstrait. SMALTALK est un système comprenant un langage et les outils associés (compilateur, metteur au point, éditeur interactif, etc) qui sont eux même construits avec les notions proposées dans le langage. Les types abstraits apelés classes, méta-classes, super-classes sont manipulés par des opérateurs désignés comme des méthodes. Classes et méthodes sont les entités connues et manipulables par les outils du système (compilation ou édition d'une classe, exécution d'une méthode). L'ensemble du système présente une très grande homogènéité.

Une utilisation plus récente de l'abstraction se trouve] dans la conception des bases de données, pour lesquelles, la prise en compte des aspects dynamiques est devenue un besoin exprimé par de nombreux concepteurs. On trouve des applications purement formelles, comme la définition d'un modèle dans [LOCK79] dans lequel il est proposé un modèle pour la conception des bases de données en plusieurs étapes.

Les premières étapes sont destinées à la description des données et correspondent à la définition du schéma à l'aide du langage de description de données (DDL). Schéma et sous-schémas définissent la sémantique de chaque type de données et les inter-relations des données de même type sous forme de types abstraits algèbriques. La dernière étape se distingue des autres car elle conduit à décrire les inter-relations entre les types de données. Ces inter-relations sont engendrées par les opérations qui sont au-delà du modèle de données (opérations portant sur plusieurs types). Cette étape défi-

nit le langage de manipulation de données (DML). Des langages de conception de base de données associant l'aspect structurel des données à l'expression de la dynamique (comportement) donné par le concept de type abstrait ont été proposés [BROD81] [CHAB80] [WEBE78].

3.4.2. Description d'une méthode

Pour illustrer notre discussion, on donne tout d'abord la description d'un compte bancaire selon le formalisme décrit dans [GUTT80a]. La spécification comprend trois phases : syntaxique, sémantique et restrictive.

syntaxe

OUVRIR $\mathbb{N} \longrightarrow$ compte

FERMER compte $\longrightarrow$

DEBIT compte $\times \mathbb{N}^* \longrightarrow$ compte

CREDIT compte $\times \mathbb{N}^* \longrightarrow$ compte

SOLDE compte $\longrightarrow \mathbb{Z}$

sémantique

declare c : compte, v : entier

SOLDE (OUVRIR (0)) = 0

SOLDE (CREDIT (c,1)) = SOLDE (c) + 1

OUVRIR (v) = CREDIT (OUVRIR (0),v)

CREDIT (CREDIT (c,v),1) = CREDIT (c,v+1)

DEBIT (DEBIT (c,v),1) = DEBIT (c,v+1)

CREDIT (DEBIT (c,1),1) = c

DEBIT (CREDIT (c,1),1) = c

restriction

failure (DEBIT (c,v)) ==> v > SOLDE (c)

La partie syntaxique introduit les opérateurs et les objets sur lesques ils s'appliquent (noms, types, domaines des valeurs prises, etc...). Dans cet exemple on n'utilise que des comptes et des valeurs entières, positives ou négatives. Aucune référence n'est faite à la sémantique propre des données sinon au travers des opérateurs qui peuvent les manipuler. Le choix de la structure de ces données est laissé au réalisateur.

La seconde partie décrit la sémantique des opérateurs qui s'exprime

par les relations entre ces opérateurs. Par exemple, SOLDE s'exprime à partir de OUVRIR, CREDIT et DEBIT ; CREDIT et DEBIT s'expriment l'un par rapport à l'autre. On peut noter que l'opérateur SOLDE est l'unique moyen de distinguer les valeurs du type abstrait entre elles puisque tous les autres opérateurs donnent un résultat dans l'ensemble des comptes.

La partie restriction donne les exceptions au fonctionnement du type abstrait. Plus générales que les préconditions, les restrictions laissent au réalisateur le choix de déclencher ou non une exception en cas de fonctionnement anormal (par exemple, la restriction donnée est une restriction sur le fontionnement de l'opérateur DEBIT).

On peut également imposer une exception systématique, et dans cet exemple on aurait indiqué :

v > SOLDE (c) ----> failure (DEBIT (c,v))

La démonstration de la validité de cette description (cohérence et complétude au sens de Guttag) due à Pierre Laforgue est donnée dans [SCOT15] pp 67-70.

3.4.3. Avantages et inconvénients

L'avantage essentiel des méthodes de description fondées sur les types abstraits est la possibilité de valider, de vérifier les descriptions. Chaque type ainsi validé peut être réutilisé dans un type construit, en conservant toutes ses propriétés démontrées au pas précédent. Cependant, il faut noter que les descriptions axiomatiques et les démonstrations, qu'elles soient algébriques ou logiques, sont d'une manipulation difficile. De plus cette description se situe très loin de la réalisation. Des systèmes d'aide à la conception et à la démonstration des propriétés des types abstraits apparaissent (système AFFIRM).

La possibilité de décomposition d'une application en objets est aussi une caractéristique intéressante de ces méthodes. L'association des opérateurs et des objets donne les changements d'états autorisés pour un objet donné. Ces changements d'états définissent les contraintes d'intégrité de l'objet qui, s'ils sont correctement spéci-

fiés, ne pourront pas être transgressés. Les constructions d'objets définissent, à chaque pas de la construction, de nouvelles règles d'intégrité qui sont des restrictions par rapport à celles définies sur les objets utilisés.

3.5. Motivations du choix de la méthode

Dans ce chapitre, nous venons de survoler l'ensemble des méthodes que nous avons explorées en vue de l'étude des applications transactionnelles. Cette analyse est loin d'être exhaustive et nous n'avons présenté que les aspects qui ont retenu notre attention.

Le choix de la méthode a été dicté par deux objectifs essentiels de notre étude :

- permettre de dégager l'aspect transactionnel de l'application,

- aider à la définition de la répartition.

Le mode transactionnel privilégie l'aspect dynamique par rapport à la structuration des données. C'est le programme qui est la partie essentielle et non pas la structure de données que manipule la transaction. Les types abstraits sont orientés vers la description de la dynamique des objets, sans toutefois négliger la structure de données composant l'objet.

D'autre part, il est clair qu'il existe une analogie entre la notion d'opérateur d'un type abstrait qui effectue un changement "atomique" de l'objet qu'il représente et une transaction qui est une opération atomique. Cette analogie est particulièrement utilisée dans une extension du langage CLU présentée dans [WEIH83].

La répartition exige l'existence d'entités qui puissent être réparties. Dans un modèle de description de données, ces entités sont liées au modèle. Par exemple, dans le modèle relationnel, c'est la relation qui pourra être répartie, ou une partition de la relation en fonction de certains critères. Un mauvais choix dans la répartition pourra conduire, dans certaines opérations, à transférer le contenu total de relations d'un site à l'autre.

La notion d'objet défini par un type abstrait offre une entité logique facilitant la conception de la distribution de l'application. En particulier, la possibilité d'agrégation d'objets que permet le concept de type abstrait autorise le choix des objets et le niveau des objets à répartir. Dans cette méthode, la répartition n'est pas contrainte par le modèle mais elle est liée à la logique de l'application. Elle offre une grande souplesse dans le choix de la répartition des objets.

Les deux principaux objectifs sont pleinement atteints par l'utilisation des types abstraits. Ils offrent en plus d'autres avantages comme par exemple la construction par agrégation ou par raffinements.

Les types abstraits sont à la base de la méthode de conception que nous avons choisie et qui est décrite dans les chapitres suivants.

CHAPITRE 4

UN MODELE TRANSACTIONNEL POUR LES APPLICATIONS

4. UN MODÈLE TRANSACTIONNEL POUR LES APPLICATIONS

4.1. Objectifs

Ce qui suit suppose l'existence d'un système transactionnel coopérant (tel qu'il a été décrit au chapitre 2) garantissant un premier niveau de cohérence (forte) vis-à-vis des facteurs externes (pannes et accès concurrent). Il reste à établir l'interface existant entre l'application et le système transactionnel.

Comme nous l'avons vu, la transaction constitue l'unité d'exécution et l'unité de validation (de maintien de la cohérence). On va montrer comment elle peut devenir, au niveau de la conception de l'application, l'unité de modélisation. Cependant, notre démarche n'a pas été d'analyser une application dans les termes transactionnels tels qu'ils existent dans les systèmes actuels (par exemple COBOL / TDS), mais plutôt de donner une spécification fonctionnelle d'une application et ensuite de montrer que les concepts transactionnels (éventuellement nouveaux) sont adaptés à sa réalisation.

Nous prendrons des exemples dans l'application bancaire étudiée dans le cadre du projet SCOT et que nous présentons succintement avant le modèle afin d'illustrer notre démarche avec des exemple issus de cette application. Le modèle n'a pas été étudié pour la description de cette application, mais l'application a servi à piloter la définition puis à valider le modèle qui pourrait très bien s'appliquer à toute autre application de type transactionnel. Pour commencer, nous justifions le choix de cette application.

4.2. L'application

4.2.1. Choix d'une application pour le projet

Dans le cadre du projet, nous souhaitions étudier une application qui réponde aux critères suivants :

1) Répartie

L'objectif de SCOT n'étant pas d'étudier les raisons conduisant à la répartition d'une application, nous avons choisi une application dont la répartition s'imposait a priori, c'est à dire dont la répartition découlait des structures d'organisation.

Les différents services d'une même grande banque (agences, siège, etc...) coopèrent pour réaliser les opérations bancaires. Même si actuellement, les services ne sont pas gérés de manière disjointe (gestion centralisée des agences) ou s'ils ne coopèrent pas directement, la répartition fonctionnelle existe. Elle peut et doit être prise en compte.

2) Transactionnelle

Le projet SCOT proposant le maintien de la cohérence d'objets correlés résidant sur plusieurs sites, une application de type "base de renseignements" n'aurait pas pu faire apparaitre les problèmes. Les opérations bancaires sont transactionnelles par nature (en dehors de leur réalisation informatique). En effet, elles s'exécutent sous la forme de procédures interactives (éventuellement par échange de courrier) déclenchées par des événements (issus du client, de la banque ou extérieurs) tels que des synchronisations. Elles sont prédéfinies et manipulent concurremment des objets bancaires (comptes de clients et de banques, cours de devises, etc).

3) Représentative

La notion de cohérence est assez universelle et l'application montre qu'elle résout le problème. De plus, les échanges bancaires comprennent des aspects commerciaux (import/export), financiers et administratifs (gestion de comptes) dans lesquels la cohérence a une grande importance, aussi bien sur le plan commercial (confiance

des clients et des autres banques) que juridique (régularité des opérations).

4) Opérationnelle

En dehors des caractéristiques précédentes, une application déjà opérationnelle présente des aspects rarement considérés dans les cas d'école. Par exemple :

- Emission de documents contractuels interdisant des retours arrière.
- Existence de transactions dont la durée est incompatible avec le blocage des ressources.
- Importance du nombre d'exceptions possibles à prendre en compte.
- Volume de transactions traitées et de données manipulées.

Cette étude a été menée en collaboration avec des spécialistes du milieu bancaire impliqués dans la réalisation de l'application actuellement existante.

4.2.2. Description informelle de l'application

Dans un premier temps, on peut décomposer le modèle bancaire (vu au travers de la Société Générale [SOGE 1] [SOGE 2]) en plusieurs grands services disposant d'une certaine indépendance dans leur fonctionnement. Le seul lien existant entre eux réside dans le transfert ou l'échange d'informations en vue de la réalisation d'opérations bancaires mettant en jeu plusieurs de ces services.

On a identifié les services suivants :

- les agences,
- le bureau central des changes (BCC),
- la direction des affaires internationales (DAIT),
- le siège central,
- la trésorerie.

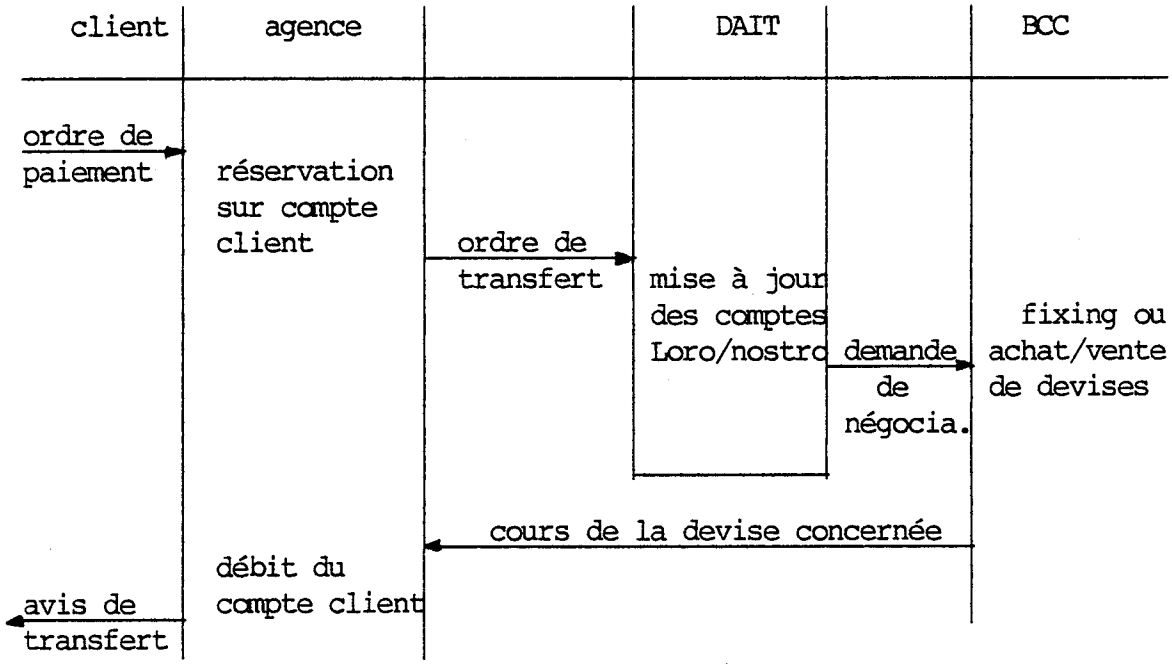
Pour les opérations avec l'étranger, seuls les trois premiers services sont impliqués de la façon suivante :

Les agences sont à l'origine de toutes les opérations (liaison avec les clients) : gestion des comptes clients, opérations de caisse, transferts étrangers, etc... Les objets manipulés par les agences sont les comptes des clients et ceux de la banque, les portefeuilles étrangers composés des remises import et export en liaison avec les clients de l'agence.

Le bureau central des changes est principalement concerné par les achats et ventes de devises étrangères, par le "fixing" (cotation journalière des devises), et par la tenue de la position de change pour chaque devise. Les objets correspondants sont les positions de change et les demandes d'achat et de vente de devises.

La DAIT est responsable des relations avec les correspondants étrangers (banques étrangères et filiales) avec qui la banque entretient des relations de compte. Il s'agit de la gestion des comptes Loro (leurs comptes chez nous) et Nostro (nos comptes chez eux) ainsi que la tenue des livres comptables.

Des opérations telles que le paiement d'une remise import mettent en jeu ces trois services selon le scénario simplifié suivant :



Pour plus de détails concernant la définition de l'application, on pourra se reporter à [SCOT15].

4.3. ELEMENTS DU MODELE

Grossièrement, on peut décomposer une application en grands services disposant d'une relative indépendance. Les liens entre services s'établissent par échange ou transfert d'information, soit pour archivage, soit pour demander l'exécution d'une fonction propre au service concerné, cette fonction rendant ou non des résultats (sous forme de transfert d'information) au service demandeur. Ainsi, certaines grandes fonctions mettent-elles en jeu plusieurs services.

Chaque service gère des données qui lui sont propres, et au cours des opérations, ne sont échangées que les données strictement nécessaires à la coopération. On ne parle pas ici des applications qui s'articulent autour d'une base de données centrale et qui ne présentent donc pas de services autonomes.

Considérant maintenant non plus l'aspect organisationnel mais l'aspect fonctionnel, une appplication se décompose en grandes fonctions qu'on appelle opérations (par exemple, le transfert de fonds entre banques étrangères, le remboursement d'un emprunt, etc). Très souvent, une opération met en jeu plusieurs services de l'organisation, séquentiellement ou en parallèle.

Chaque opération forme un tout dont le résultat n'est cohérent (du seul point de vue des opérations) que lorsque celle-ci est complètement exécutée. Une opération définit donc un changement d'état cohérent du système d'information, et on pourrait être tenté de le matérialiser (de le réaliser) par une transaction. Cependant, la durée de certaines opérations est incompatible avec le type de blocage des ressources nécessaire au contrôle d'accès concurrent qui existe aujourd'hui dans les systèmes transactionnels, et qui limite la disponibilité des informations pour les autres opérations.

> Par exemple, un transfert de fonds entre banques étrangères peut nécessiter un délai d'au moins 24 heures pour obtenir le cours de la devise. Pendant ce délai, on ne peut pas envisager de bloquer ni le compte du client, ni le compte de la banque qui sont mis en jeu par l'opération bancaire.

Etant donné la durée d'exécution de certaines opérations, un usager peut souhaiter connaitre les étapes intermédiaires de l'exécution afin de les analyser ou de les consulter. Pour celà, les étapes ne doivent pas être choisies au hasard. Elles correspondent à des états intéressants pour l'usager, bien que présentant une certaine incohérence du point de vue (global) de l'application. Ces étapes correspondent à des états dans lesquels les objets manipulés par l'opération sont dans un état cohérent (du point de vue du seul objet). Par exemple, un compte doit être toujours complètement crédité ou débité.

Le concepteur d'application est donc contraint de décomposer chaque opération dépassant une certaine durée incompatible avec la disponibilité de l'information, en plusieurs transactions dont le système, dans ce cas, garantit l'atomicité.

Evidemment, entre l'exécution des transactions, les données qu'elles manipulent sont visibles, et donc modifiables, par d'autres transac-

tions appartenant à d'autres opérations. Les données sont alors, du point de vue des opérations, globalement dans un état incohérent.

> Exemple : Un transfert entre deux banques est composé de plusieurs transactions dont une de débit d'un compte dans la banque émettrice, une transaction de transfert effectif, une dernière transaction de crédit d'un compte dans la banque réceptrice. Il est clair que les transactions de crédit dans une banque et de crédit dans l'autre banque ne sont pas liées de la même façon que ces même transactions dans une compensation intra-banque. Que l'opération globale de transfert ne se présente pas comme une transaction n'est pas nécessairement gènant. Ce qui est important, c'est d'assurer que l'ensemble des transactions sera réalisé à un moment donné.

Il est donc nécessaire de fournir au concepteur d'application un outil ou une méthode permettant de recréer la notion d'opération indivisible détruite par ce découpage, ou un moyen pour contrôler les états intermédiaires du système d'information au cours du déroulement des opérations. Dans ce dernier cas, le système d'information passera par des états dont la cohérence forte (au sens des opérations) n'est plus garantie.

> Actuellement, dans les applications bancaires, la reprise de ces incohérences est réalisée manuellement, principalement du fait de la non coopération des différents services (coopération par envoi de bordereaux).

Chaque service peut être vu comme un site logique responsable de la gestion d'un ensemble de ressources et de données, exécutant des actions sur les données et coopérant avec les autres services du système d'information. Même si physiquement l'application est réalisée par un système centralisé, il est utile de déterminer la coopération entre les services en définissant les responsabilités de chaque service donc les données gérées par chacun d'eux, celles qui sont accessibles aux autres services et comment, par quelles interfaces.

En dehors de toute répartition physique de l'application, il est donc intéressant d'analyser la coopération des différentes parties d'une même application à la manière d'un système réparti. Pour celà, on considère une répartition logique correspondant à un découpage

fonctionnel de l'application. Cette manière d'analyser permet de définir les interfaces entre les différentes entités coopérantes, ce qui rendra par la suite, une répartition physique plus aisée.

Reprenant la terminologie de SCOT, introduite à la section 2.4, sur chaque site participant à l'application, la manipulation des données d'un service (locales) se fait à l'aide d'agents. Lorsque les données sont correlées, leur manipulation implique la coopération d'agents dans une même transaction (globale) dont l'atomicité est garantie. La coopération peut être interne à un service ou mettre en jeu plusieurs services si l'application le nécessite.

> Par exemple, la compensation entre deux comptes est réalisée d'une manière générale, par un agent de débit de l'un des comptes et par un agent de crédit de l'autre compte. L'ensemble de ces deux agents s'exécutant constitue la transaction de compensation. Cette façon de décrire le fonctionnement de la compensation est général, que les comptes mis en jeu soient gérés par une même agence ou non, sur un même site physique ou sur des sites différents.

Le concepteur d'application définit donc les agents en fonction des besoins de manipulation des données locales, et aussi leur coopération qui engendre les transactions.

Pour réaliser cette triple décomposition - **opération, transaction agent** - on se propose d'adopter une même démarche en offrant un outil de conception adapté aux applications transactionnelles.

Pour celà, nous définissons les trois notions d'objet, de transaction et d'opération.

4.3.1. La notion d'objet

Un OBJET est caractérisé par un ensemble de propriétés qui doivent être conservées quelles que soient les manipulations ou transformations qu'il subit. Plus précisément, dans un système d'information, un objet correspond à un ensemble de données ayant des propriétés propres (domaine de valeurs ...) et des propriétés qui s'expriment par des relations entre les données, c'est à dire des propriétés qui mettent en jeu plusieurs données. C'est l'ensemble des propriétés individuelles et des relations qui caractérise l'objet. Ces propriétés expriment son intégrité sémantique (par exemple, la relation entre un numéro de compte, son propriétaire et le solde du compte).

Lorsqu'un ensemble de propriétés s'applique à plusieurs objets, il définit une classe ou type d'objet (par exemple tous les comptes clients d'une même banque ont les mêmes propriétés). On peut, par ailleurs, définir des collections d'objets lorsqu'on dispose d'un moyen pour les répertorier ou pour les énumérer (sur la valeur prise par une propriété par exemple). Une collection est un sous-type qui se distingue par une propriété particulière ou une restriction sur une propriété du type. Bien que souvent type et collection soient confondus, dans le cas des applications réparties (en particulier, ceci étant vrai pour d'autres applications) la distinction est importante. Par exemple, le type compte client est unique dans une banque mais chaque agence gère une collection de comptes clients. On peut comparer la notion d'objet, de type d'objets et de collection aux notions de classe, méta-classe et super-classe de SMALLTALK [BYTE81].

> Pour un objet, on peut distinguer des propriétés statiques qui s'expriment par des structures de données (rélations entre données, domaine de valeur que peuvent prendre les données, etc) et des propriétés dynamiques liées à la manipulation, c'est à dire aux changements d'état de l'objet.

Il s'agit ici des noms des opérateurs permettant de changer l'état de l'objet et des relations entre ces opérateurs (par exemple l'ordre dans lequel ils doivent être appliqués ou les conditions qui

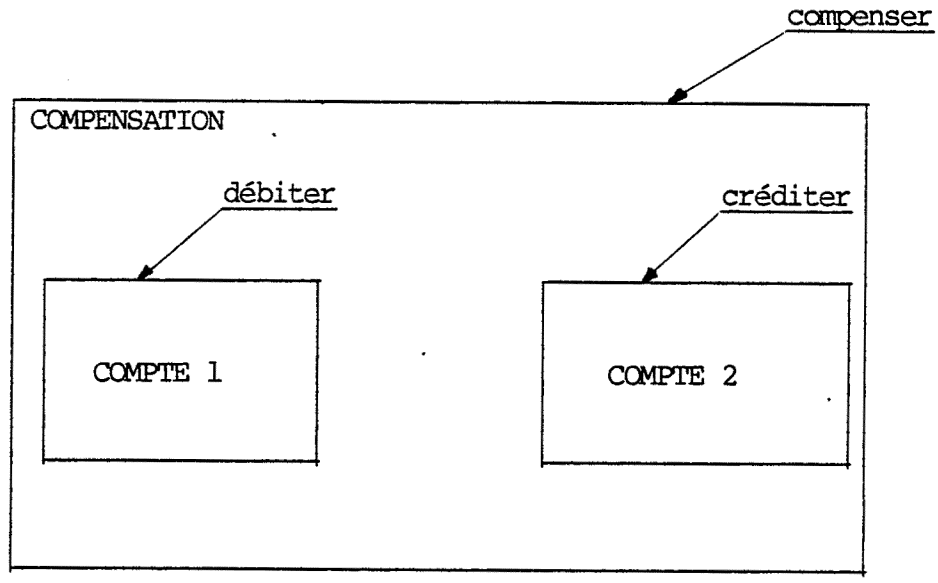
doivent être remplies pour qu'ils puissent être appliqués). En définissant toute la sémantique d'un objet, on est amené à déterminer tous les états qu'il peut prendre et quels sont les opérateurs et les conditions qui conduisent à ces états.

Vu de l'extérieur, un objet se présente donc comme un ensemble d'opérateurs. Si ces opérateurs sont le seul et unique moyen de changer son état et si chaque opérateur conserve l'intégrité de l'objet, l'intégrité sémantique de l'objet sera conservée quelles que soient les manipulations réalisées sur celui-ci.

on peut aussi distinguer dans un objet des sous ensembles de propriétés et isoler ainsi des composants qui sont aussi des objets. La description de l'objet englobant peut alors s'exprimer par une liste de composants et par les propriétés de l'association de ces composants. Si dans un assemblage, des objets distincts ont des composants communs, alors ceci introduit une contrainte d'intégrité supplémentaire

Un objet composé offre un sous-schéma (c'est à dire une restriction) des objets qu'il englobe. Ces objets ne sont accédés qu'au travers des nouveaux opérateurs définis sur l'objet englobant ne mettant pas nécessairement en oeuvre tous les opérateurs des objets composants.

> Exemple : une compensation entre deux comptes bancaires est un objet composé de deux comptes. L'opérateur "compenser" de cet objet demande l'exécution d'un opérateur "débiter" sur l'un des comptes et l'exécution d'un opérateur "créditer" sur l'autre compte. Au travers de l'opérateur "compenser", on voit donc partiellement et différemment les deux comptes.



Cependant, un objet composé ne doit pas être vu que comme une simple restriction de l'utilisation des opérateurs portant sur chacun des objets le composant. En effet, les opérateurs portant sur l'objet composé sont une composition des opérateurs applicables à chacun des objets composants mais sous certaines conditions, dans un certain ordre, etc. Cette composition définit de nouvelles contraintes d'intégrité sur les objets qui sont ainsi correlés. Il s'agit donc bien d'un enrichissement de la sémantique de l'application.

Dans l'exemple précédent, l'opérateur "compenser" est sémantiquement plus riche que chacun des opérateurs qui le composent.

Dans la description d'une application, les objets qui correspondent au plus petit granule logique manipulable au niveau de l'application, c'est à dire ceux pour lesquels il n'est pas utile de définir de composant, sont désignés comme des objets élémentaires (de l'application). Les objets composés apparaissent comme la coopération d'objets élémentaires ou d'autres objets composés.

Exemple : Un compte client peut être vu comme un objet élémentaire si au niveau de l'application on ne connait que les opérateurs permettant de le manipuler (créditer, débiter, ...) sans jamais connaitre la structure sous-jacente (c'est à dire qu'il est constitué d'un numéro,d'un solde, d'un certain

nombre de compteurs, etc), ainsi que les opérateurs qui modifient les valeurs de cette structure.

Exemples d'objets composés :

Un guichet gère une collection de comptes clients, une collection de remises import/export, une collection d'opérations de change.

Une compensation est un objet composé de deux comptes, dont la durée de vie est limitée à la durée de l'opération compenser qui consiste en un débit de l'un des comptes et en un crédit de l'autre.

4.3.2. Représentation des objets

Dans le contexte d'une application, la sémantique des objets manipulés ne peut être complètement décrite par une seule structure de données (statique) car une partie de celle-ci provient de leur comportement (dynamique). Comme nous l'avons vu au chapitre 3, les modèles de base de données sont mal adaptés à la description des aspects dynamiques des applications. De plus, il est difficile dans ces modèles, de cerner la notion d'objet. Par exemple, dans le modèle relationnel, la notion de relation permet de définir des classes d'objets, mais les opérations de manipulation relationnelles ne portent pas seulement sur les éléments de la classe mais permettent aussi de modifier chacun des attributs de la classe. Ceci signifie que chacun des attributs (c'est à dire la structure des objets) est manipulable dans l'interface des requêtes.

De ce point de vue, il semble donc plus adéquat de représenter un objet par un type abstrait qui permet de regrouper dans une même entité cette double description de structure de données et opérateurs associés. Un objet n'est donc visible, pour les autres objets, que par l'intermédiaire des opérateurs. Les relations inter-objets s'expriment donc par des demandes d'exécution de ces opérateurs.

Dans ce contexte, les opérateurs sont le seul et unique moyen de manipuler l'objet. Ils expriment donc tous les changements d'états de l'objet et doivent en conséquence en conserver les propriétés, c'est

à dire garantir son intégrité. Ceci signifie que chaque opérateur conserve l'intégrité de l'objet qu'il manipule et ainsi, des séquences autorisées d'opérateurs garantissent, elles aussi, cette intégrité.

4.3.3. La notion de transaction

Les opérateurs expriment les propriétés liées à la manipulation des objets. Chaque transaction définit un changement d'état du système d'information. L'ensemble des opérateurs liés à un objet et leurs inter-relations dans l'objet définissent tous les changements d'états possibles pour cet objet.

Si les objets du système d'information doivent être partagés (comme le sont les comptes dans une banque), afin de préserver leur cohérence, les opérateurs doivent être réalisés comme des opérations atomiques sur cet objet, que celui-ci soit élémentaire ou composé.

> Nous appelons ces opérateurs atomiques des transactions. en effet, chaque transaction préserve la cohérence d'un objet ou d'un ensemble d'objets correlés. Toute coopération d'objets dans un objet composé s'effectue au moyen des transactions définies sur les objets concernés.

Un ensemble de transactions est associé à chaque objet et constitue le seul et unique moyen d'accès et de modification de l'objet. Les transactions expriment donc les règles d'intégrité liées à la manipulation de l'objet auquel elles sont attachées. Cet ensemble de transactions doit donc être complet et cohérent (au sens des propriétés exprimées par ces transactions).

L'imbrication de transactions (nested transactions) exprime des inter-dépendances entre les objets (de même type ou de types différents) correspondant des actions devant se dérouler en coïncidence. Dans d'autres approches, cette propriété est vérifiée par des post-conditions [LOCK79]. L'exécution d'une transaction rend

obligatoire l'exécution des transactions imbriquées dans celle-ci. Cette contrainte d'exécution des transactions imbriquées est une contrainte très forte. Elle n'est justifiée que lorsque la cohérence forte est nécessaire. Nous verrons plus loin comment, dans les cas où cette cohérence forte n'est pas absolument nécessaire, on peut rejoindre les propositions faites par J. Gray [GRAY81] concernant les imbrications de transactions. Il est proposé de définir des mécanismes de retour arrière pour des transactions bien que celles-ci aient été validées. Si l'idée de ce mécanisme est générale, son application dans un contexte donné n'est pas simple. Elle nécessite de connaitre tous les opérateurs applicable à l'objet concerné par le retour arrière.

Par exemple, si un compte a été crédité, le retour arrière serait un débit qui, lui, n'est pas toujours applicable.

4.3.4. Représentation des transactions

> Les opérateurs d'un objet devant garantir sa cohérence malgré un environnement partagé et non fiable, leur exécution doit être atomique. Un opérateur d'un objet peut donc être une transaction.

La notion d'agent telle qu'elle est définie dans SCOT est utilisée pour représenter les opérateurs sur les objets élémentaires. La manipulation d'un objet composé se fait par la manipulation des différents objets élémentaires le composant. L'opérateur d'un objet composé est donc une coopération entre agents. Comme il doit être atomique, c'est une transaction au sens de SCOT.

Si SCOT permet l'imbrication de transactions (schémas d'exécution procéduraux), il permet également le parallèlisme d'exécution qui se trouve souvent dans les applications. Or, aucun modèle de base de données ne permet actuellement de décrire de telles représentations incluant les objets et les opérateurs associés, ainsi que les imbrications et les exécutions parallèles d'opérateurs.

Ainsi, la cohérence forte définie dans [GRAY78] est garantie pour les objets par la transaction (locale et globale) telle qu'elle est réalisée dans le système SCOT.

4.3.5. La notion d'opération

Nous avons vu au chapitre 3 que le maintien de la cohérence forte est coûteux et particulèrement en environnement réparti :

- blocage des ressources utilisées,
- abandon de transactions et réexécution,
- synchronisation par échange de messages sur le réseau,
- etc.

Pour certaines applications, le prix à payer pour maintenir cette cohérence peut être incompatible avec d'autes objectifs :

- temps de réponse,
- impossibilité de bloquer certaines ressources,
- durée d'exécution,
- etc.

Ceci ne signifie pas que le concepteur d'application n'ait à sa disposition qu'une politique du "tout ou rien" en matière de maintien de la cohérence. Nous proposons donc la notion d'opération qui permet au concepteur de maitriser les objets mis en présence et les transactions nécessaires à la réalisation de l'opération. Ceci lui permettra de de décider du niveau de cohérence qu'il souhaite maintenir aux objets concernés.

Une **opération** est un ensemble de transactions dont l'exécution réalise un changement d'état complexe du système d'information et surtout dont la durée est incompatible avec le verrouilage qui assure le partage cohérent des données.

Ces changements d'états portent sur un ensemble d'objets dont la corrélation n'est pas nécessairement très forte. Les objets composés étant manipulés par des transactions, une opération est donc un ensemble de transactions pouvant se dérouler séquentiellement ou en parallèle. De la même façon que pour les transactions, les schémas d'exécution des opérations sont laissés à la charge du concepteur d'application.

Bien qu'indispensable, la notion de transaction fournit un niveau de cohérence inadéquat (pour les besoins des applications) pour tenir compte de cette contrainte de durée. Cependant, à ce niveau, il est difficile de définir la cohérence une fois pour toutes, celle-ci étant très liée à l'application elle même, certaines applications exigeant une cohérence plus forte que d'autres. En particulier, l'absence de retour arrière automatique (pris en charge par les mécanismes de reprise du système) en cas de déroulement anormal de l'exécution d'une opération ne signifie pas que l'intégrité des données ne pourra pas être maintenue mais seulement qu'au niveau des opérations l'intégrité qui sera maintenue sera du niveau propre à l'application, connu et maitrisé par celle-ci.

En effet, dans le cas ou seulement une partie d'une opération est terminée de manière satisfaisante (certaines transactions seulement sont exécutées), le système présente un état incohérent du point de vue de la logique de l'application, et même un risque de divergence à partir de cet état incohérent si d'autres transactions le prennent comme point de départ. Ceci doit être un risque calculé par le concepteur qui, pour une opération donnée réalise sa décomposition en transactions.

Cet état peut même être intéressant pour l'usager de l'application puisqu'il pourra connaitre les objets et donc évaluer l'état, même incohérent du système d'information.

- Par exemple, le banquier pourra connaitre le montant des avoirs, faire des prévisions sur les achats et ventes de devises en fonction de opérations en cours.

Les transactions composant une opération peuvent se dérouler :

- <u>séquentiellement</u> et dans ce cas le contrôle de leur terminaison (qui ne porte à un instant donné que sur une transaction) ainsi

que les retours arrière sont faciles à réaliser : toutes les transactions de l'opération qui se sont exécutées avant celle en cours sont terminées.

- En parallèle. Dans ce cas, il est nécessaire d'envisager toutes les combinaisons (possibles au sens de la logique de l'application) de transactions terminées ou non, et pour chaque cas, prévoir les retours arrières éventuels.

Le seul contrôle nécessaire au concepteur est un résultat de terminaison (positif ou négatif) de chaque transaction participant à l'opération. En effet, une transaction est soit complètement exécutée, soit pas du tout. Elle ne reste jamais dans un état intermédiaire. L'ensemble des résultats permet de décider si une opération a terminé son exécution. Si tel n'est pas le cas, la combinaison des résultats positifs et négatifs peut être utilisée pour prévoir les reprises correspondantes ou une interprètation du résultat.

Celles-ci peuvent consister en :

-1- Une exécution de transactions inverses, c'est à dire des transactions annulant les effets des transactions terminées, ramenant ainsi, si c'est possible, le système d'information à l'état cohérent précédant l'exécution. Par exemple, la transaction inverse d'un débit sur un compte pourrait être un crédit, la réciproque n'étant pas nécessairement vraie si on tient compte des préconditions (ou gardes) d'exécution des transactions.

Il faut noter que les transactions inverses sont quelquefois difficiles à déterminer du fait des exceptions qu'elles peuvent produire à nouveau dans l'état d'exception qu'est la reprise (effet domino).

Ce type de reprise est de même nature que la reprise mise en oeuvre par les systèmes transactionnels pour le maintien de la cohérence face aux pannes et à l'accès concurrent. Pour celà, il est nécessaire de tenir des journaux contenant les informations indispensables à la reprise des transactions. Pour toute transaction, il faut définir la transaction inverse. Cette solution est suggérée dans [GRAY80].

-2- Une réexécution des transactions avortées avec des paramètres d'activation différents. Par exemple, forcer le débit d'un compte en dépit d'un solde négatif.

Ce type de reprise conduit, sous des conditions différentes, à un état cohérent prévu initialement. Pour cette technique, il est nécessaire de spécifier toutes les exceptions susceptibles de se produire et de consigner dans un journal les informations concernant le lancement de chaque transaction.

-3- L'exécution de transactions ad hoc dont le seul but est de ramener la base sans un état cohérent du seul point de vue de l'application, cet état pouvant éventuellement être différent de celui visé initialement.

Ceci conduit à enrichir l'ensemble des états cohérents admissibles pour la base. Cette méthode laisse une grande liberté d'action au concepteur et en particulier, rend possible l'intervention d'un usager du système.

Chacune des méthodes présente ses avantages mais aussi ses difficultés pour le concepteur. Dans la première solution, la difficulté provient du fait qu'il est nécessaire de spécifier pour chaque transaction, la transaction inverse et de minimiser le risque qu'une transaction inverse ne puisse pas se dérouler du fait d'un changement d'état de la base dû à une autre transaction et mettant en défaut le mécanisme de retour arrière.

La seconde solution n'est pas toujours applicable. En effet, il n'existe pas nécessairement plusieurs états cohérents pour un ensemble de données. Dans tous les cas, il n'est pas garanti qu'un état cohérent puisse être atteint automatiquement en toutes circonstances. Une assistance de l'usager peut s'avérer nécessaire. L'opération peut aussi être abandonnée dans l'état, le système donnant un résumé de la situation pour l'usager.

La troisième proposition ouvre la voie à toutes les actions possibles. En particulier, elle offre la possibilité de l'intervention manuelle d'un opérateur qui pourrait prendre des décisions exceptionnelles.

Le choix d'un type de reprise est fait par le concepteur en fonction de différents critères come par exemple :

- le niveau de cohérence souhaité,
- le coût de la reprise,
- un aspect strictement "politique",
- etc.

Bien que la solution ne soit pas entièrement satisfaisante sur le plan du maintien de la cohérence, ces mécanismes permettent une certaine automatisation des reprises dans les opérations, ce qui est loin d'être la situation actuelle.

Ce qui est important, c'est que cette méthode laisse au concepteur de l'application la charge du choix du niveau de cohérence pour lequel il est le seul à pouvoir décider.

> Une opération est donc complètement définie par l'ensemble des transactions qui participent et des transactions permettant d'amener le système d'information dans un état acceptable du point de vue de la cohérence en cas de déroulement anormal d'une au moins des transactions participantes.

Toute réalisation du modèle doit offrir un support (outil) au concepteur d'application pour qu'il puisse décider du niveu de cohérence choisi pour l'opération concernée. Ce modèle, fondé sur la notion de transaction, offre tous les moyens pour réaliser la notion d'opération.

4.3.6. Conclusion

Alors que les modèles de base de données classiques (CODASYL, relationnel, entité-association) tentent d'exprimer la sémantique des applications au travers de la description des données, ce modèle privilégie les aspects dynamiques en exprimant la sémantique de l'application au moyen des manipulation des objets (transactions et opérations).

Les schémas de description de données ne sont qu'une partie de l'application, l'autre partie étant représentée par les programmes ou requêtes d'interrogation de la base, même si ceux-ci peuvent être composés dynamiquement par un usager interactif.

A l'inverse des modèles de description de données qui séparent nettement la structure des données de leur manipulation, le modèle qui vient d'être décrit lie intimement les concepts d'objet et de transaction. Cette imbrication ne nuit pas à l'indépendance des données vis à vis des programmes (data independence) car elle se situe à un niveau tout différent, beaucoup plus logique.

L'intérêt de la notion d'objet est de limiter les actions possibles sur un ensemble de données, donc de mieux en maitriser la cohérence.

La possibilité de choix entre des transactions maintenant une cohérence forte et les opérations qui laissent toute liberté de choix dans le niveau de cohérence voulu, offre au concepteur d'applications transactionnelles un outil complet.

Si la notion de transaction est indispensable, le concept d'opération apporte un complément qui se montre tout aussi nécessaire pour le type d'application que nous visons. Il offre la possibilité, ayant délimité un objet composé, de choisir ou de décider du niveau de cohérence en fonction de critères propres à l'application, à chaque fois que la cohérence forte ne pourra pas être maintenue.

CHAPITRE 5

MISE EN OEUVRE DU MODELE

5. MISE EN OEUVRE DU MODELE

5.1. Introduction

L'objectif de ce chapitre est de montrer l'applicabilité du modèle défini dans le chapitre précédent en définissant une méthode de conception fondée sur ce modèle. Toute méthode de conception s'appuie sur un langage qui permet d'exprimer la spécification. Plutôt que de définir un nouveau langage (un de plus!), ce qui est hors de notre propos et de nos compétences, nous avons préféré utiliser un langage qui offre les concepts dont nous avons besoin. A ce langage nous ajoutons les concepts propres au type d'application que nous visons. Les choix que nous avons faits sont expliqués et justifiés au fur et à mesure de leur introduction.

En application des idées discutées dans les chapitres précédents, nous allons, dans ce chapitre, associer une représentation à chaque élément du modèle, c'est à dire, les objets, les transactions et les opérations.

Comme nous l'avons montré dans la présentation du modèle, la sémantique d'un objet est plus complètement décrite par un type abstrait et afin de garantir aux opérateurs du type abstrait la propriété d'atomicité, ceux-ci doivent être représentés par des transactions ou des agents impliqués dans les transactions.

En dehors des langages de spécification qui s'expriment complètement en termes de types abstraits algébriques ou axiomatiques présentés au chapitre 3, les nouveaux langages de programmation

(MESA, CLU, ADA, etc...) introduisent cette notion sous différentes formes :

- Module dans MESA [XERO78],
- Cluster dans CLU [LISK78],
- Task et Package dans ADA [ADA 80],
- Atomic type et Guardian dans ARGUS [WEIH83].

Ces langages mettent en oeuvre essentiellement la partie syntaxique du type abstrait qui est définie par l'interface du module, cluster, package ou task. La sémantique doit être exprimée en termes de programmes et reste sous la responsabilité du programmeur. Cependant, ces langages offrent des outils plus ou moins complets pour décrire les aspects sémantiques :

- synchronisation par message ou rendez-vous,
- ordonnancement des opérations (moniteurs, ...).
- commandes gardées,
- etc...

Le langage ADA regroupe des concepts suffisamment riches et nombreux pour décrire l'application. Hormis les aspects purement langage que nous développerons plus loin, le choix de ADA a été dicté par les bonnes chances qu'il possède d'être largement diffusé comme produit dans un avenir proche et par l'existence d'un document de référence à la disposition de tous et surtout à notre disposition lors du commencement de cette étude. La section suivante présente les concepts du langage qui sont utilisés en les justifiant et en donnant des exemples de leur utilisation. Il est évident que le langage contient d'autres concepts qui ne seront pas décrits, soit parce qu'ils sont communs à de nombreux langages et donc bien connus, soit parce qu'ils ne sont pas utilisés ici. Ensuite nous montrerons leur utilisation pour la conception de l'application.

5.2. UTILISATION DU LANGAGE Ada

5.2.1. Types, modules et tâches

Le langage ADA [ADA80] est fortement typé, c'est à dire qu'à partir des types prédéfinis (integer, real, string, ...) il est possible de définir de nouveaux types complexes, leur donner un nom et les réutiliser dans de nouvelles constructions. Cette propriété garantit qu'on ne pourra pas ajouter, dans l'exemple suivant, le numéro de compte au solde même si ceux-ci sot définis par des types entiers. Cette caractéristique du langage, déjà connue dans PASCAL, permet également de donner des noms dans le langage de l'application, à tous les types et objets manipulés.

Exemple :

```
type  TYPE_COMPTE_BANQUE is
    record
        NUMERO : NUMERO_DE_COMPTE;
        SOLDE : VALEUR;
    end record;
```

Avec les notions de package (ou module) et de tâche (task), le langage ADA offre une certaine réalisation des types abstraits. Un package est un regroupement de déclarations (de types, de procédures ou fonctions portant sur les types, dse packages, ...). Une tâche est un processus susceptible de se synchroniser avec d'autres tâches par un mécanisme unique : le rendez-vous.

Les packages comme les tâches sont définis en deux temps :

-1- Définition de l'interface, c'est à dire des objets et opérations accessibles de l'extérieur (on parle de la partie visible). Ceci représente la partie syntaxique du type abstrait.

Dans le cas des packages, l'interface comprend un sous ensemble des objets qu'il regroupe, une procédure ou une fonction n'y étant représentée que par son en-tête.

Dans le cas d'une tâche, l'interface ne comprend que la liste des rendez-vous (entry) utilisés dans le corps de celle-ci, en

précisant pour chacun les paramètres permettant d'échanger des messages typés à l'occasion de rendez-vous avec d'autres tâches.

-2- On définit ensuite le corps (body) du package ou de la tâche, c'est à dire la représentation des objets "non visibles" de l'extérieur, ainsi que les corps des opérations (procédures, fonctions pour les packages ; rendez-vous pour les tâches). Ceci représente la partie sémantique du type abstrait.

Par exemple, si on veut représenter un compte de banque par un package, on aura :

```
package COMPTE_BANQUE is

   procedure CREDITER (MONTANT : in integer);
   procedure DEBITER (MONTANT : in  integer);
   function SOLDE return integer;

end COMPTE_BANQUE;
```

```
package body COMPTE_BANQUE is

   COMPTE : TYPE_COMPTE_BANQUE;  -- ce type est défini
                                 -- par un record
   procedure  CREDITER (MONTANT) is

        COMPTE.SOLDE := COMPTE.SOLDE + MONTANT;

   end CREDITER;

   procedure DEBITER is ...   end DEBITER;

   function SOLDE is   ... end SOLDE;

end COMPTE_BANQUE;
```

Cet exemple montre comment on peut protéger la structure de données COMPTE de toute manipulation en dehors des opérateurs définis dans l'interface puisque la structure COMPTE n'est pas définie dans a

partie visible du package. On aurait pu associer à chaque opérateur le nom du compte à manipuler,

```
procedure CREDITER (C : in TYPE_COMPTE_BANQUE, MONTANT : in integer);
```

Ceci assure la même définition des opérations pour tous les comptes de type COMPTE_BANQUE. Dans ce cas, la structure de données associée à chaque compte doit être définie soit dans la partie visible, soit à l'extérieur de cette description, ce qui diminue la puissance de l'abstraction.

On donne maintenant la représentation de l'objet COMPTE_CLIENT sous la forme d'une tâche.

```
task COMPTE_CLIENT is

   entry OUVRIR;
   entry FERMER;
   entry CREDITER (MONTANT : in integer);
   entry DEBITER  (MONTANT : in integer);
   entry SOLDE (S : out integer);

end COMPTE_CLIENT;
```

```
task body COMPTE_CLIENT is
   C : TYPE_COMPTE_CLIENT;
begin
   accept OUVRIR;
   loop
      select
        accept CREDITER  ...  do  ... end;
      or
        accept DEBITER   ...  do  ... end;
      or
        accept  SOLDE    ...  do  ... end;
      or
        accept  FERMER  do  exit  end;
      end select;
   end loop;
end COMPTE_CLIENT;
```

L'apport de cette description par rapport à la précédente (package) est l'expression des relations existant entre les différents opérateurs. En effet, la combinaison des instructions

- accept qui autorise un rendez-vous et

- select qui autorise un choix parmi plusieurs rendez-vous possibles, un seul pouvant intervenir à la fois,

permet de décrire la séquentialité des opérateurs, la possibilité de répétition d'un ou d'un ensemble d'opérateurs sous la forme d'une expression de chemin. Cet ordonancement des opérateurs d'un type correspond à une propriété couramment rencontrée dans les systèmes d'information concernant l'ordonnancement des événements sur un objet ou un ensemble d'objets. Ceci signifie qu'un événement ne peut se produire que si un (ou un ensemble) événement s'est produit dans le passé.

Dans l'exemple ci-dessus, on indique que le compte doit d'abord être ouvert, puis il peut être indifféremment crédité, débité ou communiquer son solde, ceci jusqu'à sa fermeture. Cet exemple montre une dépendance des opérateurs internes à l'objet. Il serait également possible d'exprimer des dépendances inter-objets. Par exemple, un client n'existe effectivement que lorsque son compte est ouvert.

> Ainsi, l'état de l'objet est défini, d'une part par les valeurs que prend la structure de données qu'il représente (description statique), c'est à dire le résultat obtenu par l'application des opérateurs qui ont précédé l'état actuel, et d'autre part, par les opérateurs dont l'application (exécution) est autorisée (description dynamique).

Le rendez-vous n'est pas une simple synchronisation. Il permet en même temps aux objets d'échanger un message typé représenté par les paramètres du rendez-vous. La synchronisation autorise de séquentialiser l'exécution des objets, l'appelant attendant la fin de l'exécution de l'opération pour continuer la sienne. Cette technique permet d'échanger un message en retour.

exemple :

accept DEBITER (MONTANT : integer; RESULT : boolean) do ...

Elle permet également le déroulement en parallèle des deux objets se synchronisant.

exemple :

accept CREDITER (MONTANT : integer);

Les activations de rendez-vous peuvent être rendues conditionnelles par une instruction when (commande gardée). Le rendez-vous ne peut être activé que si la condition est vérifiée.

Par exemple, on peut faire opposition sur un compte client et une opération de débit s'exprime par :

```
when C.OPPOSITION = "NORMAL" =>
accept  DEBITER  ... do  ...  end;
```

Les préconditions expriment généralement des dépendances inter-objets lorsque la condition porte sur une variable évaluée lors de l'occurrence d'un événement. En effet, tout événement est provoqué par la synchronisation d'un autre objet.

exemple :

```
when C.OPPOSITION = 'normal' =>

accept DEBITER (MONTANT) do

  if C.SOLDE - MONTANT < MAX_DECOUVERT
  then C.OPPOSITION := 'bloque';
  ...
```

La synchronisation, l'ordonnancement des opérateurs et les préconditions sont des caractéristiques essentielles du langage. Tout au long de ce chapitre nous les utiliserons et nous montrerons toute la sémantique qu'ils apportent à la description des systèmes d'information. Ce sont ces notions qui permettront d'exprimer la dynamique des objets, pour chaque objet (ordonnancement et préconditions) et pour un ensemble d'objets (synchronisation).

5.2.2. Généricité et type de tâches

On peut enrichir le niveau d'abstraction en autorisant le regroupement au sein d'une même description des objets ayant des propriétés communes, mais dont certaines de ces propriétés sont paramétrées par des types formels. Cette notion s'appelle la généricité. Elle permet de créer des objets (occurrences ou instances du type générique) ayant des comportements analogues (propriétés dynamiques) mais se rapportant à des types différents.

Dans ADA, la généricité se présente comme une sorte de macro-générateur, remplaçant les types génériques par les types formels lors de la création des occurrences de l'objet. Le langage ADA ne nous permet de disposer que d'un niveau de généricité, c'est à dire qu'un objet générique ne peut pas être décrit dans un autre objet générique.

La généricité qui nous intéresse s'applique aux packages. Elle autorise la création statique des objets au niveau de la description et non pas dynamiquement au niveau du fonctionnement du système.

Par exemple, dans la banque, la notion de compte est assez générale et signifie qu'un tel objet ne peut être manipulé que par des opérations de crédit et de débit. Ces opérations peuvent avoir une sémantique différente suivant qu'il s'agit d'un compte client, d'un compte banque ou du compte d'un correspondant étranger. La description d'un compte générique aura donc pour paramètre le type du compte à partir duquel on distinguera les opérations.

Exemple : description d'un compte générique.

```
generic (TYPE_COMPTE, OUVERTURE, FERMETURE, ACTION)
package COMPTE_ABSTRAIT  is
|
|  task COMPTE is
|
|    entry  OUVERTURE ...
|    entry  FERMETURE ...      -- propriétés paramètrables
|    entry  ACTION ...
|
|    entry  CREDITER ...       -- propriétés standard
|    entry  DEBITER  ...
|
|  end COMPTE;
|
end COMPTE_ABSTRAIT;
```

La création d'un compte client à partir de cette définition se fait par l'opération suivante :

```
COMPTE_CLIENT is new COMPTE_ABSTRAIT (COMPTE_CL, OUVRIR, FERMER,
                                                    SOLDE);
```

Si, on avait défini COMPTE comme un task type plutôt que comme une tâche, la même opération de création aurait engendré un type d'objets dynamique que sont les COMPTE_CLIENT.

La puissance d'une telle notion réside dans deux faits :

- Tous les objets possédant les mêmes propriétés sont décrits en une seule fois, même si ces propriétés portent sur des types différents. Toute évolution d'une des propriétés est automatiquement reportée sur tous les objets représentés par cette abstraction. Ceci est très important dans la phase de conception.

- Cette description va permettre de créer plusieurs objets du même type et d'affecter un nom à chaque occurrence.

La notion de type de tâche (task type) rend possible, en plus de la notion de tâche (objet dynamique), la définition d'une classe d'objets générables suivant un modèle déterminé, non paramètré. Chaque occurrence d'objet est créée dynamiquement (dans des tâches) en lui assignant une référence au lieu d'un nom (notion d'access type).

Par exemple, si on définit le type de tâche COMPTE_CLIENT,

```
task type  COMPTE_CLIENT is ...
```

on pourra créer un compte par l'opération suivante :

```
C1 := new COMPTE_CLIENT;
```

C1 définit alors le moyen d'accès au compte et pour réaliser un crédit au compte C1, on écrira :

```
C1.CREDITER ( ... );
```

Les facilités de création dynamique offertes par la notion de type de tâche nous incitent à représenter la plupart des objets de l'application par de telles descriptions.

Les packages (génériques ou non) seront utilisés soit :

- pour représenter des objets passifs, pouvant être utilisés par les objets actifs que sont les tâches,

- pour regrouper des déclarations ou des définitions de types.

5.2.3. En résumé,

- Ada offre deux possibilités pour décrire des objets sous forme de types abstraits : le package et la tâche, chacune d'elles offrant un moyen pour nommer les abstractions construites et donc permettre de les réutiliser. Cette possibilité de nommage offre au concepteur le moyen de décrire l'application dans les termes de l'utilisateur facilitant le dialogue concepteur/utilisateur.

- La notion de rendez-vous définie dans les tâches permet de synchroniser deux objets avec échange d'un message typé.

- La généricité et les types de tâches permettent de créer et de nommer des objets statiquement (pour les génériques) ou dynamiquement (pour les types de tâches) suivant un modèle.

- La notion de tâche rend possible la description d'objets vivants (actifs) par l'expression de leur état, non seulement vis à vis des valeurs qu'ils peuvent prendre, mais aussi des opérations qui leur sont applicables.

- Avec la généricité on pourra également décrire un environnement de conception adapté à l'application (introduction de nouveaux concepts).

- Enfin, le langage Ada est complètement documenté et a de bonnes chances d'être largement diffusé. De plus, il existe un groupe de travail sur l'utilisation de ce langage pour la spécification.

5.3. Un premier niveau de spécification

5.3.1. Réalisation des objets

On va donner ici des indications sur les choix qui ont été faits et des exemples de leur utilisation. Un exemple complet et commenté est donné en annexe. On trouvera dans [SCOT15] [SCOT21] une spécification complète de l'application bancaire.

A ce niveau, on veut décrire les objets en dehors de toute contrainte de réalisation, qu'elle soit informatique, transactionnelle ou autre. On se limite donc aux propriétés générales des objets. La méthode que l'on utilise peut être aussi bien ascendante que descendante. Partant des objets les mieux maitrisés (ou les mieux connus) de l'application, on procède soit par raffinements successifs précisant ainsi le fonctionnement des objets, soit par composition des objets afin de construire l'application.

Les objets actifs sont représentés par des tâches.

Si une classe d'objets actifs doit être définie, on la décrira par un task type.

Par exemple, l'objet compte client sera défini par

```
task type COMPTE_CLIENT is ...
```

Les objets passifs ou les regroupements d'objets actifs peuvent être définis comme des packages.

Par exemple, la banque peut être définie comme le regroupement de plusieurs services qui seront eux même complètement décrits par ailleurs (separate).

```
package BANQUE is
|
|   package AGENCE_DE_BANQUE is separate;
|   package DAIT is separate;     -- Dir. des Affaires Internat.
|   package BCC  is separate;     -- Bureau Central des Changes
|
end BANQUE


package body BANQUE is
|
|   package body AGENCE_DE_BANQUE is separate;
|   package body DAIT is separate;
|   package body BCC is separate;
|
|   task  SIEGE is
|     ...
|   end SIEGE
|
end BANQUE
```

La description d'un objet ou d'un type peut être incomplète,

- soit parce que les éléments non décrits se définissent simplement par leur nom (élément connu de tous sans ambiguïté),

  exemple :

  ```
  type MONAIE_ETRANGERE is ($, £, LIT, FB, DM, ...);
  ```

- soit parce que seule la connaissance de leur interface est nécessaire à la définition du système.

  Par exemple, il n'est pas nécessaire de connaitre tout le comportement d'un client mais seulement de le connaitre vis à vis des opérations bancaires.

```
task type CLIENT is
|
|  type CLIENT_INFO is
|    record
|     NOM : string;
|     ADRESSE : string;
|      ...
|    end record;
|
|  entry DEMANDE_INSTRUCTION
|  entry AVIS_DE_SORT
|  entry NOUVELLE_ADRESSE
|  entry LISTE_DES_COMPTES ...
|
end  CLIENT
```

Par inclusion d'objets on obtient des objets composés.

Par exemple, une agence est composée de guichets réalisant les opérations d'import/export ou de gestion des comptes clients, d'une caisse, d'un ou plusieurs comptes de banque (en francs ou monnaies étrangères), etc... De plus, une agence a un fonctionnement qui est le suivant : elle est ouverte avec un certain crédit, peut recevoir les cours de la bourse, ... et enfin elle est close en donnant ses résultats.

Sa description est la suivante :

```
task type  AGENCE is
|
|  entry OUVERTURE (CREDIT : in integer);
|  entry FERMETURE (MONTANT_CAISSE : out integer);
|  entry COURS_DEVISE (COTATION : in COURS_DEVISE_INFO);
|
end AGENCE
```

```
task body  AGENCE is
   COTATION : COURS_DEVISE_INFO;
begin
   -- création des objets inclus qui sont définis comme des
   -- task type

   CAISSE := new COMPTE_BANQUE;
   SGSA := new COMPTE_BANQUE;
   GUICHET_1 := new GUICHET;
   GUICHET_2 := new GUICHET;

   -- fonctionnement de l'agence

   loop
     accept OUVERTURE do  ...  end;
        .
        .
        .
   end loop;
end AGENCE
```

5.3.2. Résultats de la première étape

A l'issue de cette première étape, il est important de montrer ce que cette démarche nous a apporté, en connaitre les limites, les insuffisances et les imperfections.

Le premier point positif important est la réalisation d'une spécification assez formalisée d'un grand système réel. On a constaté que la possèder sous une forme permettant de la consulter, de la modifier, de la manipuler, de la transformer ou de la communiquer était essentiel pour parvenir à une description complète et cohérente.

On a pu conserver tout au long du travail un bon niveau de généralité, c'est à dire ne pas noyer les points importants dans les détails relevant plus de la réalisation que de ce type de description.

La plus grande partie des termes utilisés est tirée des termes même de l'application, le langage n'apportant un support que pour décrire les mécanismes non inhérents à l'application (synchronisation, structure de la description, création d'objets).

L'éclatement de l'application en de nombreuses parties décrites indépendamment n'a posé aucun problème et permet une lecture et une compréhension plus aisée de chacune des parties du fait de sa petite taille.

Le regroupement, dans une même description, d'une structure de données et des opérateurs permettant de la manipuler (type abstrait) est important tant sur le plan de la fiabilité que sur le plan de la lisibilité.

En séparant la partie déclarative (visible) de la partie réalisation (body), on peut offrir très rapidement des interfaces aux utilisateurs des objets ainsi définis, sans que la partie réalisation soit compètement spécifiée.

- Choix techniques

Dans cette étape de la conception, aucun choix technique lié à la réalisation (informatique) de l'application n'a été effectué. La description que nous avons donné ci-dessus reste une description fonctionnelle. Dans la méthode, elle constitue la spécification du niveau le plus haut. A aucun moment ne s'est posé le problème de la représentation physique des objets, de l'existence ou non de fichiers ou d'une base de données. La description des objets reste formelle : des structures de données et des opérateurs qui leurs sont associés. Seul le découpage de l'application en objets est lié à la méthode d'analyse utilisée (fonctionnelle). En particulier, la répartition, si elle apparait possible, n'a pas été explicitée. Il ne s'agit pas d'un aspect négatif, au contraire, car la description garde toute sa généralité. Toutes les réalisations, centralisées ou réparties, utilisant ou non une base de données restent possibles.

- Vérification

Bien qu'on ait pu utiliser le "test translator" installé sur le système "Multics", celui-ci ne nous a rendu que peu de services. La syntaxe du langage prise en compte est celle de ADA provisoire alors que notre description utilise largement les concepts introduits dans la version définitive du langage (le concept de "task type" en particulier).

L'utilisation intensive des types de données permet de rendre la spécification très lisible, même pour un lecteur non informaticien et limite l'utilisation des données hors des valeurs ou états possibles.

Les interfaces (messages de synchronisation) entre les différents objets, c'est à dire les noms des opérateurs et les paramètres passés lors d'un appel, sont complètement décrits avec pour chacun d'eux un type. Une vérification "manuelle" est donc extrêmement aisée. On a pu ainsi détecter et corriger des incohérences introduites dans les interfaces entre objets.

5.4. Introduction des éléments du modèle

Telle qu'elle est présentée dans le modèle, la transaction est une opération atomique sur un objet (lui même représenté par un type abstrait).

> La transaction définit une unité de cohérence, c'est à dire une unité d'exécution préservant la cohérence. Ainsi, un opérateur sur un objet composé, qui manipule des objets élémentaires, est une transaction.

Cet opérateur peut à son tour être utilisé dans un opérateur d'un objet composé englobant, toutes ces transactions appartenant à la même unité de cohérence.

C'est au concepteur de définir la décomposition de l'application en objets d'une part et en unités de cohérence d'autre part, en fonction de critères tels que, par exemple, la disponibilité de l'information ou le temps de réponse du système. Cette décomposition n'est pas statique, c'est à dire qu'on ne peut pas définir a priori des ensembles de transactions constituant des unités de cohérence. Au contraire, elle est dynamique, c'est à dire que le concepteur définit de début d'une unité de cohérence et qu'ensuite, toutes les actions de la filiation appartiendront à cette unité de cohérence. Une unité de cohérence n'est complètement définie que lorsque toutes les actions qui la composent ont été activées.

Par contre, il est possible de connaitre toutes les actions appartenant potentiellement à une transaction. Certaines, par exemple, peuvent appartenir à la même alternative, une seule étant activée en fonction de la valeur d'une donnée, d'un événement ou d'un message. Cette connaissance peut être utilisée pour des simulations ou pour des applications spécifiques (c.f. confidentialité au § 6.5).

Afin de décrire cette décomposition, le concepteur aura à sa disposition un certain nombre de notions nouvelles dont nous décrirons d'abord la syntaxe, puis la sémantique précise.

Pour celà, un certain nombre de règles ont été ajoutées à la grammaire du langage ADA. D'autres extensions sont faites en utilisant le langage lui-même.

Nous allons examiner successivement la déclaration des objets, des transactions et des agents, puis leur activation. Ensuite nous examinerons les notions de transaction et d'opération. Pour terminer, nous établirons la correspondance entre les règles définies dans le langage et les mécanismes du système SCOT (cf. section 2.4).

5.4.1. Définition des objets

Nous avons conservé, pour la déclaration des objets, la syntaxe du langage ADA. Bien que les termes task et package ne correspondent pas exactement au sens de la définition des objets, la réalisation de ces deux notions correspond à ctte définition.

Partout où la sémantique du langage n'a pas été modifiée, la syntaxe a été conservée telle quelle.

Pour la définition des objets, on s'en tient à la présentation faite à la section 5.3.

5.4.2. Déclaration des agents

Pour conserver l'homogènéité avec le langage ADA [CII 80], nous utilisons une terminologie en anglais pour la définition syntaxique des extensions. En ce qui concerne les mots clé introduits (transaction et agent) leur signification est la même dans les deux langues.

Tout opérateur sur un objet est déclaré comme un agent, tant dans la partie interface (visible) que dans le corps de l'objet. Ceci est réalisé par l'ajout des deux règles suivantes dans la grammaire du langage :

```
entry_declaration ::= ...
        | agent identifier [(discrete_range)][formal_part];

accept_statement ::= ...
        | agent agent_name [formal_part]
              [do sequence_of_stmts end [identifier]];
              [sequence_of_stmts] end agent;
```

"agent_name" est un identificateur, éventuellement indicé, déclaré comme agent dans la partie spécification.

L'activation de l'agent n'est considérée comme terminée que sur le mot clé "end". La séquence d'instructions entre end et end agent se déroule en parallèle avec l'agent appelant. Il est donc possible de spécifier des schémas d'exécution procéduraux (end et end agent consécutifs), ou parallèle (sans do et end comme l'autorise ADA).

exemple : Compte client.

```
task type COMPTE_CLIENT is
   agent OUVRIR (C : in COMPTE_CLIENT_INFO);
   agent CREDITER (M : in integer);
   agent DEBITER (M : in integer);
      .
      .
end COMPTE_CLIENT


task body COMPTE_CLIENT is
   C : COMPTE_CLIENT_INFO;
begin
   agent OUVRIR (CI) do
      -- vérifications puis affectation des paramètres
      C := CI;
   end;
   end agent;                    -- exemple d'exécution
                                 -- procédurale d'un agent

   loop
      select
        agent CREDITER (M)
          C.SOLDE := C.SOLDE + M;

                                 -- exemple d'agent se déroulant
                                 -- en parallèle avec l'appelant

        end agent;
      or
        agent DEBITER (M)
          ...
        end agent;

      end select;
   end loop;
end COMPTE_CLIENT
```

5.4.3. Activation des agents et des transactions

Les activations d'agents dans une transaction déterminent l'arbre d'invocation de cette transaction. Deux règles sont ajoutées au langage pour l'activation des agents et des transactions. L'une a pour but de définir de début d'une unité de cohérence (transaction) et ceci correspond au lancement de l'agent initial de la transaction. L'autre définit l'activation d'un agent quelconque (autre que l'initial) au sein d'une transaction.

1) **Initialisation d'une transaction :**

```
entry_call ::= ...
| start transaction agent_name [actual_parameter_part]
```

2) Activation d'un agent quelconque :

```
entry_call ::= ...
| start agent agent_name [actual_parameter_list]
```

Comme on l'a vu, la déclaration d'un agent définit les règles d'ordonancement de celui-ci vis à vis de l'agent appelant, c'est à dire le déroulement procédural ou parallèle. Les raisons conduisant un concepteur à choisir l'un ou l'autre des scénarios sont diverses. Elles dépendent beaucoup de l'application. Pour une application donnée, la recherche de certaines synchronisations telles que le lancement d'une action après dialogue avec l'usager ou après l'exécution complète d'un calcul ou au contraire le lancement de toutes les actions en parallèle pour réaliser la transaction dans un délai aussi court que possible.

Cependant, l'une des principales raisons conduisant à choisir l'un ou l'autre des schémas d'exécution est la synchronisation pour transfert d'information en fin d'exécution d'un agent. On montrera plus loin que ce problème peut être résolu par l'activation d'un

agent ayant pour paramètres d'initialisation les valeurs à transmettre. L'activation d'un agent est alors le seul mécanisme de synchronisation, alors que SCOT propose deux mécanismes qui sont l'activation d'agent et l'attente et envoi de messages.

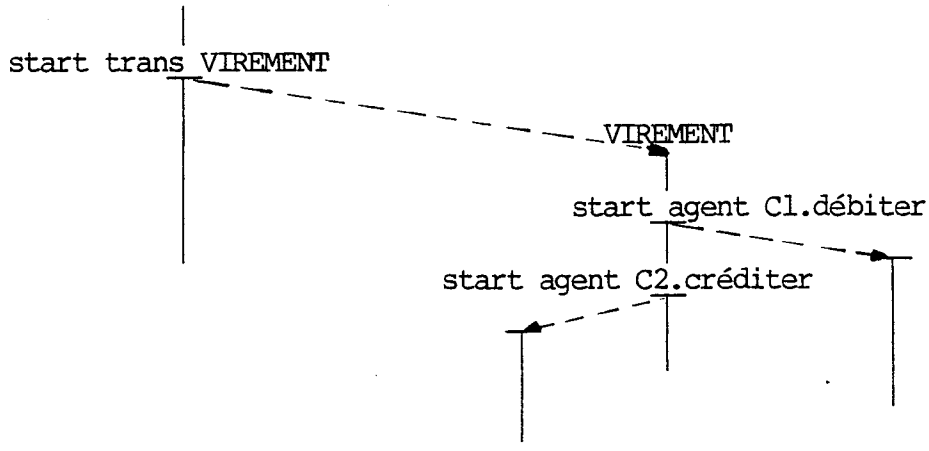
Exemple d'activation d'agents :

```
agent VIREMENT (C1, C2 : COMPTE ; M : integer);
|
|   start agent C1.DEBITER (M);
|
|   start agent C2.CREDITER (M);
|
end VIREMENT
end agent
```

Dans une opération, on trouvera l'activation de l'agent initial VIREMENT d'une transaction :

```
start transaction VIREMENT (CA, CB, MONTANT_V);
```

L'arbre d'invocation de cette transaction est le suivant :



Il est nécessaire de relier ces termes avec ceux introduits dans la présentation du système SCOT donnée dans la section 2.4. Selon les

protocoles, pour le premier agent de la transaction, on parlera d'agent initial ou supérieur ; pour le second, on parlera d'agent inférieur.

5.4.4. La notion d'opération

Une opération est constituée par un ensemble de transactions dont l'exécution constitue un tout pour la logique de l'application. Il est donc nécessaire d'offrir au concepteur un outil pour définir et contrôler cet ensemble de transactions. En particulier, les transactions étant validées, le système ne fournit aucun moyen pour défaire les modifications qu'elles ont produites. Le concepteur doit prévoir les retours arrière en cas de terminaison anormale de l'une des transactions de l'opération.

Pour permettre au concepteur de contrôler les opérations, nous proposons trois primitives.

La première sert à indiquer le début d'une opération :

```
DEBUT_OPERATION;
```

Elle signifie que toutes les transactions qui seront activées après ce point appartiendront à la même opération. Aucun paramètre n'est nécessaire.

Les deux autres primitives servent à contrôler la fin d'une opération :

```
ATTENTE_FIN_UNE_TRANSACTION (J : TYPE_JOURNAL);

ATTENTE_FIN_TOUTES_TRANSACTIONS (J : TYPE_JOURNAL_MULT);
```

La première sert à attendre la fin de chaque transaction et permet de contrôler son résultat dans un journal. La seconde attend la fin de toutes les transactions appartenant à l'opération, les résultats étant consignés dans un journal multiple.

Après chacune de ces primitives, le concepteur à les moyens de contrôler l'exécution de l'opération et si nécessaire d'envisager les retours arrière du niveau application.

5.4.5 Les mécanismes de SCOT

Dans cette partie on se place sous l'angle de l'application, d'où le terme agent est vu comme un opérateur sur un objet élémentaire, et le terme transaction définissant un ensemble d'agents coopérants préservant la cohérence et l'intégrité globales des objets manipulés.

A partir des définitions qui viennent d'être données, on peut essayer d'établir des correspondances avec les protocoles définis dans le système SCOT. Pour celà, il faut raffiner la spécification et décrire ces mécanismes en appliquant les choix techniques de ces protocoles (décrits dans la partie 2.4).

Une transaction couvre deux aspects distincts : l'application elle-même qui se traduit en termes- de manipulation de données par les agents, et les aspects systèmes qui se traduisent dans les termes des protocoles de SCOT.

On choisit de distinguer ces deux aspects en représentant l'agent par deux tâches. Une tâche système (SCOT) gère les protcoles, et une tâche application réalise les aspects application, c'est à dire qu'elle exécute un opérateur de l'objet sur lequel elle agit. Les synchronisations entre ces deux tâches sont fixées par les protocoles. Il s'agit essentiellement de l'activation de l'agent avec passage des paramètres, du signal de terminaison de la tâche application et des demandes d'activation d'autres agents (fils) ou de transactions (nouvelles unités de cohérence).

Les tâches systèmes doivent prendre en compte ces événements. Le protocole de validation de SCOT prévoit deux types d'agents. L'agent initial appelé agent supérieur ; c'est lui qui contrôle le processus de validation. Les autres agents sont appelés agents inférieurs ; ils rendent compte à l'agent supérieur de leur déroulement (terminaison, validation, ou abandon).

L'interface d'une tâche gérant un agent inférieur est la suivante :

```
task type INF_SCOT_TASK is

      -- événement de fin d'initialisation de l'agent
      -- courant TI au sein du supérieur TS

   entry D_INIT (TS : SUP_SCOT_TASK; TI : INF_SCOT_TASK);

      -- Demande de démarrage d'un agent fils

   entry  START (T : out INF_SCOT_TASK);

      -- Demande de démarrage d'une nouvelle unité de cohérence
      -- indépendante

   entry START_GLOBAL (T : out SUP_SCOT_TASK);

      -- événements de fin d'agent application

   entry LOCAL_END (CODE : TYPE_CODE_RETOUR);

   entry ABORT (CODE_ABORT : TYPE_CODE_ABORT);
      .
      . -- et autres événements du protocole SCOT
      .
end INF_SCOT_TASK
```

On trouvera une description particulière du corps de cette tâche dans [SCOT15] pp. 58-59.

La tâche déroulant l'application (le corps de l'objet concerné) devra contenir les "entry-calls" de synchronisation avec la tâche inférieure SCOT à laquelle elle est attachée et en particulier, le mot clé "end agent" engendrera :

```
T.LOCAL_END (C);
```

où T est l'accès à la tâche SCOT inférieure qui contrôle cette transaction et C le code de retour. A la réception de ce signal, si le code de retour est satisfaisant, la tâche SCOT inférieure enverra à la supérieure le message du protocole "fin d'agent" et se préparera à exécuter le processus de validation. Sinon, elle enverra un message d'abandon (T.ABORT).

L'interface de la tâche SCOT supérieure est la suivante :

```
task type  SUP_SCOT_TASK  is

      -- événement de fin d'initialisation de l'agent
      -- supérieur TS dans l'opération OP.

   entry O_INIT (TS : SUP_SCOT_TASK; OP : OPERATION_TASK);

      -- Demandes d'activation d'agents et de transactions

   entry START (T : out INF_SCOT_TASK);

   entry START_GLOBAL (T : out SUP_SCOT_TASK);


      -- événement de fin de tâche application initiale

   entry LOCAL_END (CODE : TYPE_CODE_RETOUR);

      -- événement de fin de l'agent fils. Activé pour
      -- chaque terminaison d'un agent de la filiation

   entry D_END (T : INF_SCOT_TASK);

      -- événement d'abandon programmé d'un agent

   entry ABORT (CODE : TYPE_CODE_AB; T : INF_SCOT_TASK);

end SUP_SCOT_TASK
```

On trouve dans cette interface des liens avec :

- l'opération dont dépend la transaction,

- l'agent application initial de la transaction,

- les tâches SCOT inférieures de tous les agents de la filiation.

Exemple :

Nous donnons la séquence engendrée pour l'activation des agents dans l'agent VIREMENT dont la description est donnée ci-dessous.

```
agent VIREMENT (C1, C2 : COMPTE; M : integer)  do
|
|     start agent C1.DEBITER (M);
|     start agent C2.CREDITER (M);
|
end VIREMENT;
end agent;
```

L'agent VIREMENT est contrôlé par une tâche SCOT (supérieure ou inférieure) possédant la référence TV. La séquence engendrée est la suivante :

```
accept VIREMENT (C1,C2 : COMPTE ; M : integer) do
|
|    TV.START (TD);    -- création d'une nouvelle tâche SCOT
|                      -- inférieure qui retourne un accès TD
|                      -- et contrôlera l'agent DEBIT
|    C1.DEBIT (M; TD);
|                      -- TD est transmis à la tâche activée
|                      -- pour synchronisation en fin d'exécution
|
|    TV.START (TC);
|    C2.CREDITER (M; TC);
|
|    T.LOCAL_END;
|
end VIREMENT
```

Nous présentons maintenant la description (partielle) en langage Ada d'une tâche SCOT relative à un agent supérieur.

```
task body  SUP_SCOT_TASK  is

    -- sont décrits les événements de l'interface usager : O_INIT,
    -- START, LOCAL_END, ABORT, START_GLOBAL
    -- Les événements inter systèmes transactionnels : DIN, D_END

    -- Variables SCOT :
    ENDLIST,COMMITLIST,SONLIST,OKLIST : TYPE_LISTE;

    -- variables nécessaires à la description :
    MOI_MEME : SUP_SCOT_TASK;
    ID_OPER : OPERATION_TASK;

begin

    accept O_INIT (T : SUP_SCOT_TASK; OP : OPERATION_TASK);
      -- démarrage de la transaction
      MOI_MEME := T;
      ID_OPER := OP;
    end O_INIT;

    loop
      select
        accept START (T : out INF_SCOT_TASK) do
          -- démarrage d'une tâche SCOT inférieure à partir de
          -- l'agent initial
          T := new INF_SCOT_TASK;
          T.D_INIT (MOI_MEME; T);
          -- mise à jour des listes SONLIST, COMMITLIST etc
          return T;
        end START;

      or
        accept START_GLOBAL;
          -- démarrage d'une nouvelle tâche SCOT supérieure
          ID_OPER.START_GLOBAL (T : out SUP_SCOT_TASK);
        end START_GLOBAL;

      or
        accept LOCAL_END (CODE : TYPE_CODE_RETOUR) do
          -- La tâche application associée est terminée.
          -- 1) attendre la fin des inférieures.
```

```
        While ENDLIST /= COMMITLIST do
             accept D_END (T : INF_SCOT_TASK) do

                -- mise à jour ENDLIST
             end D_END;
         end while;


         -- 2) effectuer la validation (non décrit)

         -- 3) Retourner à l'opération le code retour
         ID_OPER.D_END (MOI_MEME, CODE_RET);

       end LOCAL_END;

     or
       accept ABORT ( CODE_ABORT : TYPE_CODE_ABORT) do
         -- réception d'un avortement programmé d'une inférieure

       end ABORT;
     end select
   end loop;

end SUP_SCOT_TASK
```

- Réalisation de la notion d'opération

Une opération est constituée de deux tâches :

Une tâche T1 définie par le concepteur (déroulant l'application).

Une tâche T2 qui enregistre dans un journal les activations, validations ou avortements des transactions pour en rendre compte à T1.

La tâche T2 est créée à partir d'un modèle prédéfini à la demande de T1 par la fonction

DEBUT_OPERATION

qui retourne à T1, l'accès à la tâche T2 qui vient d'être créée.

La fin d'une opération est déterminée par l'attente de la terminaison des transactions concernées. Cette attente peut être de deux types :

attente du signal de chaque terminaison ou abandon parl'appel de l'entrée

```
T2.ATTENTE_FIN_UNE_TRANSACTION (J : out TYPE_JOURNAL);
```

attente du signal de fin de toutes les transactions par l'appel de l'entrée

```
T2.ATTENTE_FIN_TOUTES_TRANSACTIONS (J : out TYPE_JOURNAL_MULT);
```

Les types TYPE_JOURNAL et TYPE_JOURNAL_MULT définissent les structures des informations associées à la fin des transactions : identité de la transaction globale, liste des transactions activées par celle-ci, compte-rendu de validation, etc...

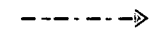
Nous présentons ci-dessous l'interface d'une tâche opération (T2).

```
task type OPERATION_TASK is

      -- événement de fin d'initialisation de la tâche pour
      -- passage de son identité

   entry FIN_INIT (T : OPERATION_TASK);

      -- demande d'activation d'une transaction

   entry START_GLOBAL (T : out SUP_SCOT_TASK);

      -- événement de fin de transaction  avec passage
      -- de la filiation (SON_LIST)

   entry D_END (T : SUP_SCOT_TASK; SON_LISTE : LISTE;
                                    CR : TYPE_CODE_RETOUR);

      -- enregistrement des attentes de fin d'opération

   entry WAIT_ALL_GLOBAL_ENDS (INFO : out TYPE_ARRAY_JOURNAL);

   entry WAIT_ANY_GLOBAL_END (INFO : out TYPE_JOURNAL);

end OPERATION_TASK
```

Le corps de cette tâche est détaillé dans [SCOT15] pp. 65-66.

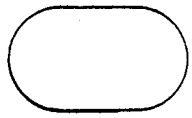
5.4.5. Architecture

les différentes transactions et opérations sont contrôlées par un ensemble de tâches reliées entre elles. Ces liens peuvent s'exprimer sur le graphe suivant où on représente par :


la création et activation d'une tâche SCOT contrôlant un agent


le contrôle d'une transaction application par une tâche SCOT

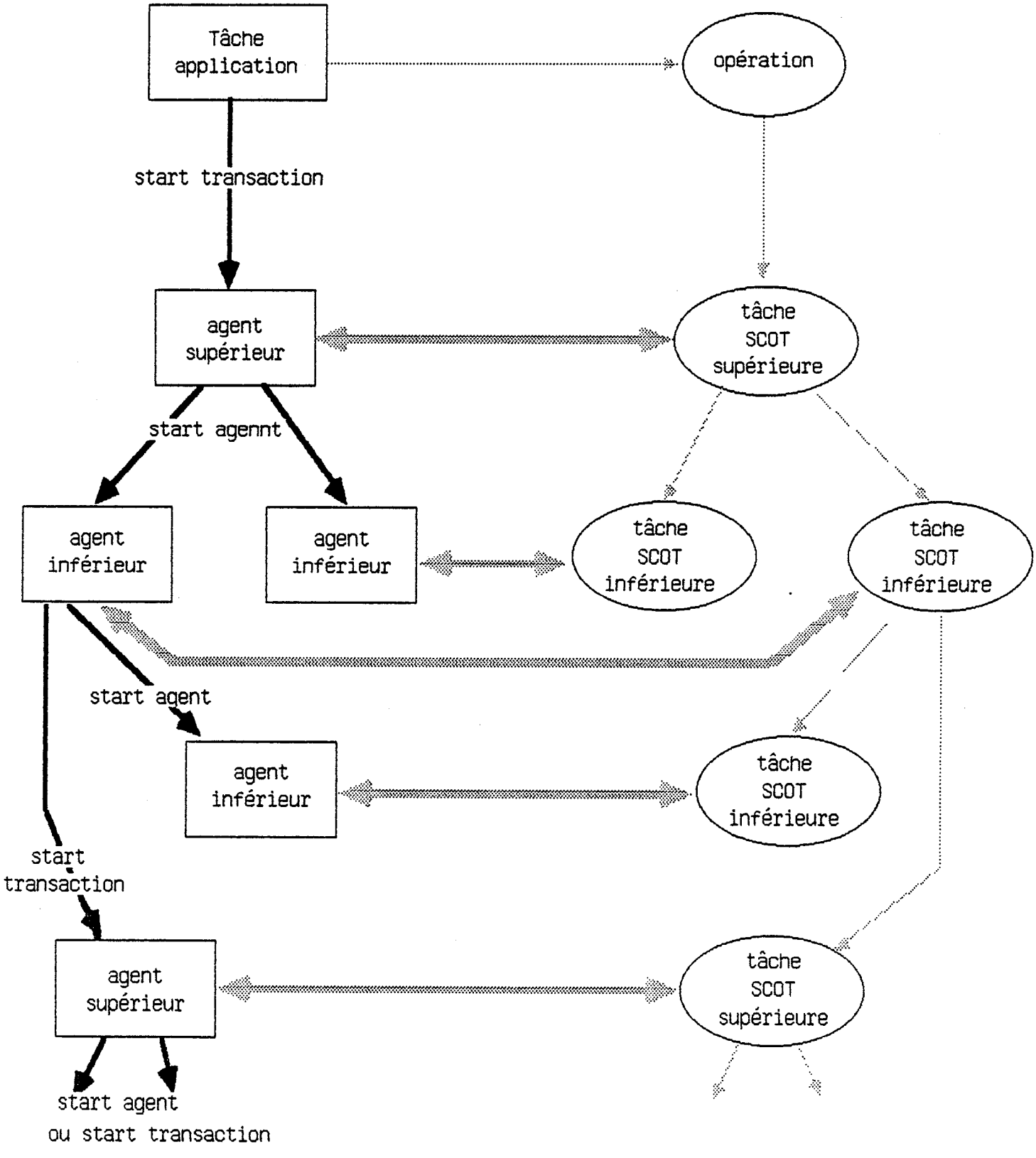

le lancement d'un agent (initial ou autre)



représente une tâche système pour un agent supérieur ou inférieur,

représente une tâche application pour une transaction ou un agent ou une opération

APPLICATION SYSTEME



5.5. Seconde description de l'application

5.5.1. Application du langage

Nous allons maintenant décrire l'application dans les termes introduits dans la section précédente (agent, transaction, opération). La principale évolution de cette description par rapport à la précédente (cf. 5.3) est la détermination, par le concepteur, des unités de cohérence. Le concepteur doit dans cette phase définir le découpage de l'application en opérations, transactions et agents. Pour chaque opération, il doit établir les retours arrière en cas de terminaison non satisfaisante d'une au moins des transactions la composant.

Remarque :

On pourra noter que le découpage en unités de cohérence n'est pas lié à une quelconque répartition, mais seulement à la manipulation de données correlées.

Nous allons illustrer les mécanismes de conception définis dans les sections précédentes par quelques exemples. Une description plus complète est donnée en annexe et dans [SCOT21].

Pour illustrer la déclaration des transactions dans un objet élémentaire, nous donnons ci-dessous la description d'un compte de banque. Ces comptes sont toujours ouverts et les seules opérations applicables sont CREDITER, DEBITER et CONSULTER.

```
task type COMPTE_BANQUE is
|
|    agent  CREDITER (M : in integer);
|    agent  DEBITER (M : in integer);
|    agent  CONSULTER (S : out integer);
|
end COMPTE_BANQUE;
```

```
task body COMPTE_BANQUE is

   CB : COMPTE_BANQUE_INFO;

 begin
   loop
     select
       agent  CREDITER (M)
       |
       | CB.SOLDE :=CB.SOLDE + M;
       end agent;

     or
       agent  DEBITER (M)
       |
       | CB.SOLDE := CB.SOLDE - M;
       end agent;

     or
       agent  CONSULTER (S) do
       |
       | S := CB.SOLDE;
       end CONSULTER;
       end agent;
     end select;
   end loop;

end COMPTE_BANQUE;
```

La description du VIREMENT montre la mise en oeuvre des mécanismes d'activation des agents. Cette opération bancaire peut être complètement réalisée dans une transaction puisqu'elle ne met pas en oeuvre d'autre transaction indépendante (pas de start_transaction). Elle pourra appartenir à la même unité de cohérence que l'agent guichet qui l'a lancée.

On peut imaginer un scénario différent. L'agent guichet lance une transaction indépendante de virement dans une opération. Ceci permet alors de gérer les cas d'anomalie de déroulement et d'envisager une reprise automatique du virement. Mais on peut noter que le choix de l'un ou l'autre des scénario est indépendant de la définition de l'agent VIREMENT.

```
task type VIREMENT is
    agent  INIT_VIREMENT (V : in VIREMENT_INFO);
end VIREMENT;

task body VIREMENT is
    CE,CR : COMPTE_CLIENT_A;
    AE,AR : AGENCE_A;
  begin
    agent  INIT_VIREMENT (V : in VIREMENT_INFO) do
      -- rechercher les comptes client CE et CR de l'émetteur
      -- et du récepteur et le compte SGSA par lequel
      -- transitera l'opération.
      -- calculer AE et AR les agences de l'émetteur et du
      -- récepteur.

      start agent  AE.CE.DEBITER (V.MONTANT);
      start agent  AR.CR.APPROVISIONNER (V.MONTANT;
                                         V.DATE_DE_VALEUR, SGSA);

      start agent  AR.SGSA.CREDITER (V.MONTANT);

    end INIT_VIREMENT;
    end agent;
end VIREMENT;
```

L'exemple d'un GUICHET illustre les mécanismes de définition d'opération et d'activation de transactions.

La tâche GUICHET est à l'origine de toutes les opérations initialisées dans les agences. On donne ici les exemples d'ouverture d'un compte client, de virement interne et de demande de transfert étranger.

Dans la demande de transfert, le début d'une opération est matérialisé par l'instruction "start_opération" [1], donnant une identifi-

cation d'opération dans TRANS_PILOT. Cette identification sera transmise à chaque transaction appartenant à cette opéra-tion. En [2] on attend la fin de toutes les transactions constituant l'opération en vue, si nécessaire, de réaliser des reprises.

```
task type GUICHET is

    agent  VIREMENT_INTERNE (CL_ORIGINE : TYPE_IDENTIFICATION;
               CL_DESTIN : TYPE_IDENTIFICATION; M : integer);
    agent  VIREMENT_EXTERNE (CL_ORIGINE : TYPE_IDENTIFICATION,
                        BANQUE_DESTIN : BANQUE; M : integer);
    agent  REMISE_CHEQUE;
    agent  DEPOT_ESPECES;
    agent  RETRAIT_ESPECES;
    agent  REMISE_EXPORT;
    agent  REMISE_IMPORT;
    agent  DEMANDE_TRANSFERT;
    agent  OUVRIR_COMPTE_CLIENT;

end GUICHET;
```

```
task body GUICHET is

    -- Phase d'initialisation.
    -- Un guichet est associé à un terminal. Tous les agents
    -- exécutés ici correspondent à des opérations bancaires
    -- Il est nécessaire d'assurer le suivi de l'opération jusqu'à
    -- sa fin et exécuter des reprises en cas de problèmes.


    CPT_CL : COMPTE_CLIENT_INFO;
    V_INFO : VIREMENT_INFO;
    T_INFO : TRANSFERT_INFO;
    TRANS_PILOT : operation_task;
    J : TYPE_ARRAY_JOURNAL;
```

```
begin

  loop
    select
      agent  OUVRIR_COMPTE_CLIENT do

        -- saisie des  paramètres y compris le numéro
        -- de compte et initialisation de CPT_CL.

        C := new COMPTE_CLIENT;

        -- conserver la référence C et le numéro de compte par
        -- exemple dans un dictionnaire de références de comptes.

        start agent  C.OUVRIR (CPT_CL);

      end OUVRIR_COMPTE_CLIENT;
      end agent;


    or
      agent  VIREMENT_INTERNE (CL_ORIGINE;CL_DESTIN,M) do

        -- initialisation de V_info
        V := new VIREMENT;

        start agent  V.INIT_VIREMENT (V.INFO);

      end VIREMENT_INTERNE;
      end agent;

    or
```

```
        agent  DEMANDE_TRANSFERT do
          |
          |   -- saisie des informations nécessaires à T_INFO.
          |
          |   T := new TRANSFERT_SIMPLE_EMIS;
          |
    [1]   |   TRANS_PILOT := début_opération; -- lancement d'une
          |                                   -- opération
          |
          |   start transaction T.PRISE_EN_CHARGE (T_INFO);
          |
          |   if TR.MONTANT*COURS_DU_JOUR (TR_INFO.DEVISE) >
          |                                     MAX_COURS_DU_JOUR
          |   then
          |     start transaction DEMANDE_NEGOCIATION (T_INFO);
          |
          |   -- ici on doit attendre la réception du résultat de la
          |   -- négociation et agir en fonction de son résultat.
          |   -- Si négociation acceptée :
          |
          |
          |     start agent  T.IMPUTER_TIRE (T_INFO);
          |
          |   -- sinon faire une reprise.
          |  end if;
          |
    [2]   |   attente fin toutes transactions (J);  -- attente de la
          |                                         -- fin des transactions.
          |  -- consultation du journal J pour connaitre le résultat
          |  -- de l'exécution des transactions. Si l'une d'elles
          |  -- s'est avortée, reprendre son exécution ou défaire
          |  -- les modifications de celles qui sont validées.
          |
          |  end DEMANDE_TRANSFERT;
          |  end agent;
          |
          |  end select
          | end loop;
          |
        end GUICHET;
```

5.5.2. Résultats et conclusions

La méthode nous a permis de décrire l'application en termes transactionnels, de déterminer les unités de cohérence et d'adapter à chacune d'elles un niveau de cohérence correspondant aux besoins de l'application.

Les raffinements successifs de la spécification ont conduit à l'ébauche de la définition des protocoles de transactionnel copérant. Nous avons présenté ceux décrits dans la section 2.4.

Ces protocoles sont du niveau "application", c'est à dire le niveau 7 du modèle OSI de l'ISO [ISO 79]. Pour les grandes applications (banques, assurances, administration, ...) qui constitient des mondes relativement fermés, les protocoles peuvent être ajustés, adaptés aux besoins spécifiques de chacune d'elles. Par exemple, il serait possible d'introduire des mécanismes propres au contrôle de la notion d'opération, ou encore ne pas contrôler l'accès concurrent à un objet par des agents d'une même transaction, etc).

Il est toutefois nécessaire, pour conserver une passerele avec des applications extérieures, de maintenir dans ces protocoles le minimum assurant la coopération de transaction et garantissant la cohérence forte.

Dans [GRAY80], il est proposé une extension des protocoles transactionnels à un niveau qui correspond approximativement à celui d'opération. Le mécanisme de reprise repose sur la fourniture pour chaque transaction, d'une transaction inverse (cf. §4.3.5). Si pour une application donnée, ce mécanisme était généralisé, la méthode que nous proposons permet d'introduire celui-ci dans les protocoles. Ceci déchargerait le concepteur de la gestion de ce niveau de cohérence, toute son attention étant portée sur la définition des unités sur lesquelles doit porter cette cohérence.

De ce fait, la méthode va beaucoup plus loin que la simple conception d'applications transactionelles puisqu'elle permet également de décrire les protocoles et les "couches système" nécessaires à la réalisation des applications.

CHAPITRE 6

BILAN DE LA METHODE DE CONCEPTION

6. BILAN DE LA MÉTHODE DE CONCEPTION

Il est nécessaire maintenant de tirer les leçons de l'utilisation de la méthode pour concevoir les applications transactionnelles.

Nous avons proposé de procéder en deux phases. Or il peut sembler que le concepteur puisse décrire son application directement dans le langage proposé pour la seconde phase. Le rôle de la première phase est cependant essentiel. Il a permis de dégager les objets et leurs interactions au travers des opérateurs. Cette description peut se faire sans qu'il soin nécessaire de définir ou de connaitre les unités de cohérence. Au contraire, c'est la connaissance des objets, acquise dans la première phase de description, qui permettra un découpage plus aisé en unités de cohérence.

6.1. La prise en compte de la cohérence

Sur la décomposition en objets réalisée dans la première étape de conception, on a appliqué un choix technique important : le découpage de l'application en unités de cohérence. Ce découpage a été rendu possible par l'introduction des notions de transaction, d'agent et d'opération.

Bien que ces concepts soient très puissants (sémantique très riche), leur introduction dans le langage reste simple et la description conserve une bonne concision.

Ainsi, on a défini un environnement de conception bien adapté à l'étude des applications transactionnelles coopérantes. Cette métho-

de est avant tout orientée vers la définition de la cohérence dans les applications. Bien qu'elle ait été développée dans le cadre d'une étude concernant les systèmes coopérants (distribués), la méthode est complètement applicable pour la conception des systèmes centralisés. En effet, la répartition n'est prise en compte qu'en fin de la phase de conception. Si les méthodes de maintien de la cohérence en centralisé sont différentes de celles utilisées dans les systèmes répartis, la notion même de cohérence est générale et commune aux deux types de systèmes d'information. Les différences apparaissent dans les outils systèmes de maintien de la cohérence et non pas dans la conception de l'application.

Bien que tous les exemples utilisés pour illustrer notre démarche aient été extraits de la même application, on peut penser que celle-ci est suffisamment générale pour que les problèmes abordés lors de son étude se retrouvent dans la plupart des applications de type transactionnel.

A ce stade de la conception, aucun autre choix technique n'a été effectué. Nous allons maintenant aborder quelques-uns des choix qui sont maintenant possibles : la répartition, la réalisation sur un système transactionnel précis, la communication entre entre agents d'une même transaction.

6.2. La répartition

A partir de la connaissance du découpage en objets et de la description des liens entre les objets (opérations), on peut maintenant envisager la répartition de l'application. La règle fixée par le transactionnel coopérant, tel qu'il est défini dans ce qui précède, est que les données locales ne sont manipulées que par des agents locaux. On n'envisage pas le transport d'algorithmes ni le transport des données pour un traitement sur un autre site. Dans notre description, ceci se traduit par le fait qu'une occurrence d'un objet simple (données et agents s'y appliquant) réside sur un seul site.

Différentes stratégies de répartition sont possibles. Elles pourront être fondées sur des décisions politiques (volonté de décentraliser ou de donner plus d'autonomie à des services ou des agences), ou sur

des évaluations de performances à réaliser (temps de réponse d'une transaction, disponibilité locale des données), ou encore sur une architecture matérielle incluant des machines spécialisées (terminaux bancaires, distributeurs de billets, par exemple).

On peut donner quelques exemples de règles de répartition :

- Tous les objets d'un même type résident sur un seul site.

  exemple : les comptes loro/nostro de tous les correspondants sont centralisés sur un site gérant les relations internationales.

- Les objets d'un même type sont répartis sur plusieurs sites suivant un critère de localisation (création de collections en fonction de la localisation).

  exemple : -Les comptes clients sont répartis dans les agences en fonction du numéro de compte.

- La répartition se fait par service.

  exemple : Chaque service BCC, DAIT, chaque agence est géré sur un site distinct.

Le système transactionnel devra connaître la localisation des objets (en utilisant des dictionnaires de données par exemple) afin d'activer les agents sur les sites correspondants aux données qui doivent être manipulées. De plus, il aura à tenir compte du fait que plusieurs agents d'une même transaction s'exécutent sur le même site, voire sur le même objet. Dans ce cas, des simplifications pourront être trouvées dans les protocoles (pas d'échange de messages, pas de contrôle d'accès concurrent entre agents d'une même transaction).

6.3. Impact de la méthode de conception sur la réalisation

Une partie de l'application présentée ayant été réalisée sur le système SCOT, on peut donner quelques indications sur l'influence que la méthode de conception a sur la réalisation.

Le système SCOT est construit autour d'un noyau de système transactionnel conventionnel (DTF : Distributed Transactional Facility) offrant une interface de programmation en COBOL. La distance séparant ADA de COBOL peut sembler grande. Cependant, la décomposition en objets fournit un cadre pour définir la modularité de la réalisation de l'application. Chaque objet est réalisé par un module contenant une structure de données (article de fichier par exemple) et l'ensemble des transactions qui pourront manipuler ces données.

Dans chaque module il est également possible de définir un sous schéma d'une base de données ne donnant l'accès qu'aux données nécessaires à ce module, garantissant ainsi que ces données ne sont manipulables que par les transaction ou les agents définis dans le module, à condition que les sous schémas relatifs à des modules différents n'aient pas d'intersection.

Ayant complètement réalisé les objets de base du système, il est alors possible de les utiliser dans d'autres modules. Dans ce cas, seule la connaissance de l'interface est nécessaire, c'est à dire la connaissance des noms des transactions avec leurs paramètres. La sémantique interne d'une opération peut donc évoluer (changement de la structure d'un fichier ou de la programmation d'une transaction) sans perturber la réalisation des objets qui l'utilisent, tant que l'interface n'évolue pas.

6.4. Communication entre transactions

A aucun moment de la conception, on n'a utilisé les commandes ENVOI et RECEVOIR proposées dans le système SCOT pour la communication entre agents d'une même transactions. Toute la communication a pu être réalisée par activation d'agents avec passage de paramètres.

Cette solution, si elle apparait plus canonique, peut présenter quelques difficultés lors de la réalisation. En effet, pour chaque échange de message, on procède à l'activation d'un nouvel agent. Ceci conduit, dans certains cas, à des transactions composées d'un grand nombre d'agents, dont le rôle de certaines est très réduit. Ceci entraîne un processus de validation plus long et plus coûteux (en terme d'échange de messages).

Ce découpage peut également amener à exécuter plusieurs agents d'une même transaction sur un objet. Le protocole de contrôle d'accès concurrent doit autoriser ce type d'accès dont le contrôle reste alors sous la responsabilité du concepteur ou du programmeur d'application.

Nous avons pu constater que, d'une manière gégérale, comme le note M. Brodie dans [BROD81], ce type de méthode conduit à un morcellement de l'application en un grand nombre d'objets et d'opérateurs de taille réduite. La technique consistant à activer un agent pour communiquer accentue ce défaut.

Au moment de la réalisation, il faut donc choisir entre la canonicité des notions mises en oeuvre et l'efficacité (augmentation du nombre d'agents activés et du nombre de messages éhangés pour réaliser le protocole). Quelle que soit la méthode de communication choisie, le mécanisme d'activation d'un agent doit être très efficace.

Un autre problème important n'est pas abordé, celui de la décomposition de l'application en objets appropriés. Le choix des objets reste intuitif et en partie subjectif. La méthode, comme d'autres, n'offre qu'un outil pour décrire cette décomposition.

6.5. La confidentialité dans une application

Une étude sur la confidentialité dans le applications transactionnelles coopérantes a été menée en relation avec le projet SCOT. Dans une base de données, surtout répartie, de même que pour le maintien de l'intégrité, le fait de pouvoir définir une requête dydnamiquement, sans relation avec les requêtes précédentes ou suivantes, pose des problèmes pour le contrôle de la confidentialité des données. De ce point de vue, les applications transactionnelles sont mieux adaptées à ce contrôle. En effet, les requêtes sont toutes connues à l'avance. Elles peuvent être analysées et modifiées pour y inclure un contrôle de la confidentialité.

Cependant, ce seul point de vue est insuffisant, car la confidentialité s'exprime en fonction des corrélations entre les données

manipulées. Certaines données sont critiques car elles donnent accès à d'autres données (données charnières).

La méthode d'analyse que nous avons proposée permet de connaitre toutes les corrélations entre les données en déterminant l'arbre d'activation des agents dans toutes les transactions. Il est possible de construire automatiquement ce graphe et d'en déduire les données charnières. Ensuite, en fonction de la confidentialité de chaque donnée, un superviseur de confidentialité détermine la confidentialité de chaque transaction. Tout le travail de détermination de la confidentialité est effectué statiquement aun moment de la conception de l'application.Il ne reste au système transactionnel qu'à contrôler les activations des transactions par les usagers, en fonction des droits d'accès de chacun d'eux.

Une maquette de cette méthode de détermination et de contrôle de la confidentialité a été réalisée au dessus du système SCOT et de l'application décrite dans les termes de notre proposition.

6.6. Résultats et Réalisation

L'objectif initial de l'étude d'une application dans le cadre du projet SCOT était de mettre en évidence, dans une application réelle, des scénarios qui permettent d'évaluer les protocoles définis dans le projet. Cette étude est allée au delà de cet objectif et les résultats les plus importants sont les suivants :

- la nécessité d'une méthodologie.

  Nous avons montré que la conception des application transactionnelles exigeait une méthodologie particulière adaptée à la recherche des unités d'exécution cohérente, que ce soit en centralisé ou en réparti.

- la proposition d'un modèle et d'une méthodologie.

  Le modèle présenté met en lumière les différents niveaux de cohérence adoptés lors de la conception d'une application transactionnelle. La méthodologie et les outils proposés aident le concepteur

à déterminer les unités de cohérence et à choisir les outils de maintien de celle ci qu'il doit mettre en oeuvre.

- la spécification d'une aplication réelle.

  Partant du cahier des charges d'une application bancaire, celle ci a été spécifiée à l'aide de l'outil proposé en collaboration avec des équipes de la Société Générale et de SG2 [SIRI81]. Une grande partie de cette spécification est donnée en annexe.

- L'étude de la répartition.

  Alors que la répartition des bases de données présente une certaine raideur liée au faible nombre de concepts des modèles et permettant la distribution, au contraire, le modèle transactionel développé ici pour la décomposition de l'application offre une grande souplesse dans la distibution des objets, qu'elle soit organisationnelle ou fonctionelle.

  Des scénarios d'exécution extraits de l'application ont pu montrer l'intérêt des systèmes transactionels coopérants.

Les réalisations sont :

- Le langage de conception qui peut être utilisé pour la conception de toute application transactionnelle, répartie ou non.

- La spécification de l'application qui a permis d'évaluer le langage et la méthodologie proposés et qui a également été utilisée pour une étude sur la confidentialité des applications transactionnelles.

- La réalisation partielle de l'application spécifiée. Cette réalisation a été effectuée sur le prototype (incomplet) du système SCOT.

6.7. Comparaison avec d'autres approches

D'autres approches des applications ont été faites, soit en vue de leur répartition [LISK79], soit de leur réalisation sur des bases de données [BROD81] [CHAB80]. En ce qui concerne la réalisation des applications transactionnelles, un modèle transactionnel a été proposé par Gray [GRAY80].

Dans ce qui suit, on se propose de comparer notre approche avec le modèle transactionnel proposé par J. Gray et l'introduction dans le langage CLU de primitives permettant la prise en compte de la répartition [LISK79, LISK81].

6.7.1. Le modèle transactionnel

Cet article [GRAY80] distingue les problèmes en centralisé et en réparti. La notion de transaction est proposée comme base pour la fiabilité (le maintien de la cohérence). A la transaction sont associés des journaux (avant et après) autorisant la validation et la reprise de cette transaction. Pour la gestion de l'accès concurrent, une méthode de sérialisation des transactions est utilisée.

Dans un système réparti, la transaction migre de site en site, chacun des sites conservant dans ses journaux la trace des actions locales sur les données. La fiabilité (validation) est assurée par un mécanisme de validation à deux phases analogue à celui utilisé dans le système SCOT. Le contrôle de l'accès concurrent est réalisé par des mécanismes de verrouillage et les interblocages sont détectés et non pas prévenus comme il est proposé dans SCOT.

Ce modèle propose, à partir de la notion de transaction, un ensemble de mécanismes garantissant une cohérence forte des données des applications. Certains de ces mécanismes sont analogues à ceux du système SCOT (validation), d'autres différents (contrôle d'exécution répartie, interblocages).

Il est ensuite proposé d'introduire ces notions dans un langage possédant la notion de module. On associe alors au module des attributs indiquant si les opérations définies dans ce module doivent ou non

engendrer des articles dans les journaux, si les modifications des données doivent être validées, et si le module peut être partagé.

Pour chaque opération d'un module, il est alors nécessaire de définir une opération inverse (le retour arrière) et une opération de réexécution, qui tiennent compte des données écrites dans les journaux. Les opérations de validation et d'avortement sont sous le contrôle du programmeur.

Ce modèle est limité au transactionnel "pur". Il est de même niveau que le modèle du système transactionnel SCOT. Il ne tient pas compte des opérations de longue durée pour lesquelles la notion de transaction est insuffisante. Aucun outil pour gérer la cohérence des données manipulées par ces opérations n'est proposé. Le seul niveau de cohérence disponible pour le programmeur est celui offert par le système (cohérence forte).

6.7.2. Proposition de langage [LISK79, LISK81, WEIH83]

La base de départ de cette proposition est le langage CLU [LISK77]. Cete extension concerne la programmation d'applications réparties. A la notion d'objet ayant une durée de vie longue (donnée) on associe la notion de GUARDIAN.

Un guardian est un ensemble d'objets fortement typés et de processus résidant sur un seul site. Plusieurs guardians peuvent appartenir à un même site. Les processus manipulent directement le objets de leur guardian. A l'intérieur d'un guardian, les objets sont partagés par les processus, mais le partage n'est pas possible entre guardians. Chaque guardian est accessible via un ensemble d'opérateurs appelés handlers. Tout argument d'un handler est passé par valeur (toute référence ne peut se faire qu'à l'intérieur d'un handler. Le guardian offre un service sûr pour une ressource et dispose de moyens pour le garantir (sauvegarde, restauration, reprise, ...). Une exécution consiste en plusieurs guardians et les processus peuvent communiquer par messages.

Le langage offre des outils de communication entre processus de différents guardians sur des voies unidirectionnelles, entre "PORTS". Le mécanisme retenu est un envoi de message typé, sans attente.

La réalisation de ce modèle est faite sur le langage ARGUS [WEIH83] qui offre la notion de type atomique. En plus des propriétés des types abstraits classiques, un type atomique garantit la sérialisabilité et l'atomicité des opérateurs du type. Les types atomiques sont soit prédéfinis (tableau, enregistrement, ...), soit définis par le programmeur.

Les mécanismes garantissant la cohérence des types sont entre autres, le verrouillage à deux phases pour réaliser l'ordonnancement des actions, la conservation d'informations redondantes (journaux) pour assurer la reprise.

On retrouve là des notions analogues à celles de notre proposition. La notion d'objet résidant sur un seul site et celle de guardian, gérant d'une ressource, l'ensemble de processus représentant les transactions actives sur l'objet.

Par contre le mécanisme de communication proposé par B. Liskov est beaucoup plus général et offre plus un outil pour la réalisation de mécanismes spécifiques à certaines classes d'applications. Dans la proposition SCOT, les mécanismes sont liés au mode transactionnel. Une fois ce niveau complètement décrit (syntaxe et sémantique), son utilisation est beaucoup plus simple que le mécanisme port - message.

D'autre part, aucun outil n'est offert pour garantir la cohérence globale sur des ressources gérées par plusieurs guardians. La notion de guardian se situe à un niveau intermédiaire entre la notion de tâche du langage Ada et notre proposition d'objet composé et de transaction. A partir de la notion de guardian, on pourrait construire un environnement de conception du même type que celui que nous proposons.

CHAPITRE 7

CONCLUSION

7. CONCLUSION

7.1 Rappel des objectifs

Le projet SCOT avait pour but l'étude du maintien de la cohérence dans les systèmes d'information. Dès le départ, les objectifs de l'étude comprenaient à la fois les aspects systèmes et application. Le système comprenait les outils à mettre en oeuvre pour maintenir la cohérence. Les aspects application concernaient d'une part l'utilisation des outils offerts par le système et d'autre part la définition des besoins de cohérence des applications et leur expression.

De plus, l'étude des applications devait fournir des informations sous la forme de scénarios d'exécution, de validation, de reprise qui, nous l'espèrions, orienteraient la définition des mécanismes de maintien de la cohérence. La plupart des scénarios ont pu être trouvés a priori, l'application ne servant à montrer qu'ils existent réellement et qu'ils apparaissent tous dans une même application. Cette étude a également permis de montrer l'importance de chacun de ces scénarios dans une même application. Par exemple, en ce qui concerne la concurrence, il a été montré que certaines données sont partagées par de nombreuses transactions alors que d'autres le sont très peu, ce qui justifie des scénarios différents.

L'étude des applications a été une composante importante du projet SCOT. Elle a permis de mieux définir la notion de cohérence, du point de vue global pour un système d'information, et en particulier

de montrer que la cohérence forte n'est pas toujours nécessaire ou même possible à réaliser. Il faut souvent trouver un compromis entre la possibilité de partage des données et la pénalisation induite par le verouillage des données pour réaliser la concurrence d'accès.

Le projet a aussi mis en évidence le fait que le niveau de cohérence requis par les applications est très variable. Si la cohérence forte est nécessaire au niveau des objets élémentaires, elle ne l'est pas (ou très rarement) au niveau macroscopique (celui des opérations) où un grand nombre d'objets sont mis en jeu et pour une durée longue. Il est apparu que le plus important est de contrôler les risques d'incohérences et d'éviter qu'elles ne se propagent.

7.2. Le transactionnel coopérant : quel avenir?

Après avoir participé à deux projets : Polyphème dont l'objectif était la réalisation d'un SGBD réparti, et SCOT qui visait la réalisation d'un système transactionnel coopérant, il est maintenant possible de faire quelques comparaisons et de tirer des conclusions sur ces deux approches.

A la fin du projet Polyphème (1979), la faisabilité d'un SGBD réparti était démontrée mais de nombreux points restaient à étudier. Ce projet avait montré en particulier qu'un modèle de base de données réparties était réalisable, même en présence de bases de données régies par des modèles différents. Des points essentiels comme le maintien de la cohérence, le contrôle d'accès concurrent et le contrôle de l'intégrité sémantique n'avaient pas (ou très peu) été abordés.

Si l'objectif final est d'arriver à la réalisation d'un SGBD-R ayant au moins toutes les fonctionnalités d'un SGBD centralisé, le projet SCOT est une étape indispensable. Des projets plus récents de SGBD-R (INGRES, SIRIUS-delta) ont montré que le SGBD devait être construit sur une couche transactionnelle garantissant la cohérence. SCOT est donc un maillon dans la construction d'un tel SGBD.

D'autre part, la coopération signifie très souvent l'interconnexion de matériels et de systèmes hétérogènes d'origines diverses. Pour

parvenir s.à une telle coopération, il est nécessaire de définir les règles, les protocoles qui la régissent.

De ce point de vue, les systèmes transactionnels coopérants apportent une solution à court et moyen terme pour la réalisation de systèmes d'information répartis. En effet, une normalisation peut intervenir rapidement sur des concepts simples tels que la notion de transaction, l'activation à distance d'une transaction, la communication, la validation et l'accès concurrent. C'est bien sur des protocoles analogues à ceux de SCOT que doit porter une telle normalisation.

La réalisation d'un SGBDR nécessite la normalisation de ces même protocoles plus un accord sur le modéle de description de données, le langage d'interrogation, les opérateurs, les contraintes d'intégrité, ...

Alors que les protocoles du transactionnel coopérant sont pratiquement acquis, les protocoles pour les bases de données réparties sont loin de voir le jour.

7.3. Le langage de conception

L'intérêt du mode transactionnel est d'offrir un outil pour contrôler et garantir la cohérence des données dans une application. La difficulté pour le concepteur est de définir quelle cohérence, pour quelles données.

La méthodologie que nous proposons aide le concepteur à définir les objets de l'application ; non pas les données, mais les associations de données significatives pour l'aplication. C'est ce que nous appelons objets. Ils n'ont d'existence que parce qu'ils sont manipulés, modifiés par un ensemble d'opérateurs (agents ou transactions). Le fait de disposer d'une définition précise des objets (structure et opérateurs) permet de déterminer la cohérence nécessaire pour chacun d'eux et de l'exprimer au travers des trois notions d'agent, de transaction et d'opération.

Le langage que nous avons défini permet d'exmpimer de façon précise et claire la cohérence voulue pour chaque objet de l'application. Il permet également de différencier les niveaux de cohérence. La cohérence forte est assurée par la notion de transaction. La notion d'opération laisse au concepteur le choix du niveau de cohérence propre à son application mais aussi le soin de construire les outils permettant de maintenir la cohérence ainsi définie.

On peut penser que l'utilisation d'un langage conçu pour la programmation n'est pas adaptée à la spécification. Dans notre cas, il s'agit plus précisément d'un langage d'aide à la conception. Dans le domaine d'utilisation, la distance entre ce langage de conception et la réalisation (généralement en COBOL et pour longemps) est encore très grande.

La notion d'objet est générale et se prête à diverses réalisations en fonction des outils à la disposition du programmeur d'application.

Si le langage ADA impose des contraintes d'écriture, le résultat permet un contrôle de la cohérence de la spécification, sans toutefois aller jusqu'à offrir un outil de vérification ou de démonstration. Le langage que nous avons défini est beaucoup plus un langage de définition d'architecture de système mettant en évidence les interactions entre les objets.

La présentation algorithmique de la spécification entraine souvent une écriture manquant de concision. En contrepartie, elle est facilement compréhensible par un programmeur. L'utilisation des types avec des noms clairs garantit, après une courte phase d'apprentissage, la lecture par un spécialiste du domaine d'application.

L'utilisation du langage ADA pour la réalisation d'applications transactionnelles n'est pas pour un avenir proche. Cependant, dans un tel cas, notre approche resterait valable car elle conduirait, par raffinements successifs à la réalisation de l'application elle même. En attendant, l'utilisation de la méthode permettrait de familiariser les programmeurs avec ce type de langage. En effet, si l'écriture reste difficile, après un court apprentissage, la lecture est aisée. Ce serait une formation des programmeurs par l'exemple.

7.4. L'avenir : un système de gestion de base d'objets

Un des principaux résultats de ce travail a été de montrer que les objets gérés par les systèmes d'information ne sont pas de simples données statiques évoluant au fur et à mesure des transactions exécutées sous la seule responsabilité d'un usager. Au contraire, ces objet ont un comportement dynamique, c'est à dire qu'ils évoluent et que leur évolution est fonction des actions exécutées sur l'objet. A chaque changement d'état, le comportement de l'objet peut être différent. Ceci signifie qu'en plus de l'état que constitue la valeur des données composant l'objet, doit s'ajouter l'état lié à son comportement. Il inclut les opérateurs de l'objet exécutables dans l'état actuel et les conditions d'exécution de ces opérateurs.

Dans cette approche, chaque objet contient ses règles d'intégrité sémantiques propres exprimées au travers des opérateurs et de leur enchainement. L'étude des applications a montré qu'il n'est pas nécessaire de définir tous les états imaginables pour un objet et toutes les conditions sur ces états, mais seulement les états qui ont un sens pour l'objet (et l'application), c'est à dire ceux qui peuvent être atteints par l'exécution des opérateurs définis sur l'objet. Dans les systèmes d'information, les opérateurs sur chaque objet sont connus à l'avance ; leur nombre est limité. Ce sont ces opérateurs qui définissent les règles d'intégrité sémantique sur l'objet.

Les modèles de base de données actuels ne sont pas adaptés à de telles descriptions. En effet, ils séparent description et manipulation des données. Or pour réaliser facilement un système d'information, il serait intéressant de disposer d'un modèle adapté à la définition et la manipulation d'objets : un modèle de base d'objets.

Un article dans ce sens a été publié par P. Lockeman et al [LOCK79]. Cette proposition théorique, fondée sur les types abstraits algèbriques, ne semble pas avoir été poursuivie ou reprise pour une expérimentation.

REFERENCES BIBLIOGRAPHIQUES

BIBLIOGRAPHIE GENERALE

[ABH79a] André E., Bogo G., Haméon J.
"Conceptual approach to office automation".
Séminaire international sur les systèmes intégrés de burotique, Versailles Novembre 1979.

[ABH79b] André E., Bogo G., Haméon J.
"La bureautique : un exemple d'approche conceptuelle".
Rapport de recherche n° 176, Laboratoire IMAG, Novembre 1979.

[ABRI74] Abrial J.R.
"Data Semantics".
IFIP TC2 W conf. Cargese 1974.

[ABRI77] Abrial J.R.
"Z : a specification language".
Paris 1977.

[ADIB78] Adiba M.
"Un modèle relationnel et une architecture pour les systèmes de bases de données réparties".
Thèse d'état, Université de Grenoble, Septembre 1978.

[ANDR80] Andrade J.M.
"L'intégrité et la mise à jour dans un SGBD-Réparti. projet Polyphème".
Thèse Docteur-ingénieur, Université de Grenoble, Nov. 1980

[BACH78] Bachman C.W.
"The role concept in data models".
VLDB Conference, Berlin 1978.

[BADA79a] Badal D.Z. et al.
"Cost and performance analysis of semantic integrity validation methods".
Computer Science dept UCLA Los Angeles CA.

[BERN79] Bernstein P. A. et al.
"Concurrency control in SDD1 : a system for distributed databases. Part 1".
C.C.A. Cambridge Mass. Technical report CCA03-79, Jan. 79

[BERT79] Bert D.
"La programmation générique. Construction de logiciel, spécification algébrique et vérification".
Thèse doctorat d'état, Université de Grenoble, Juin 1979.

[BERT82] Bert D.
"Software component construction".
Laboratoire IMAG, Grenoble 1982.

[BOGO78] Bogo G.
"Agora : un système de courrier électronique sur le SGBD réparti POLYPHEME".
Rapport de DEA, Université de Grenoble, Septembre 1978.

[BOGO81] Bogo G., Chupin J.C.
"Conception des applications transactionnelles réparties".
Projet SIRIUS, Journées de présentation des résultats, Paris, novembre 1981.

[BROD81] Brodie M.L.
"On modelling behavioural semantics of databases".
IEEE Transactions on Software Engineering 2/1981, pp 32-41.

[BYTE81]
"SMALLTALK 80".
Byte, Aout 81.

[CALE78] Caleca J.Y.
"L'expression de la décomposition des transactions dans un SGBD-Réparti".
Thèse de 3ème cycle, Université de Grenoble, Sept. 1978.

[CHAB80] Chabrier J.J., Henry P.
"Un système de programmation pour la construction et la manipulation des systèmes de bases de données".
Bulletin GROPLAN n° 11. 1980

[CHE 81] Cheval J.L., Delaunay M., Laforgue P., Raymond J.
"Un outil de spécification d'applications transactionnelles et une application pilote pour le projet SCOT".
Rapport de recherche IMAG, Février 1981.

[CII 80] Cii Honeywell Bull
"Reference Manual for the ADA Programming Language".
CII-HB, 68 route de Versailles, Louveciennes, Juillet 1980

[CODD79] Codd E.F.
"Extending the database relational model".
ACM Sigmod Conference, Boston May 1979.

[COLL79] Colliat G., Bachman C.
"Commitment in a distributed database".
Database Architecture, G. Bracchi and G Nijsen edts.
North Holland 1979.

[FERR82] Ferrat L.
"Le sous-système ISIS : une méthode pour contrôler l'intégrité sémantique dans MICROBE".
Laboratoire IMAG, Septembre 1982.

[GOGE75] Gogen J.A. et al.
"Abstract datatypes as initial algebra and correctness of data representation".
Proc. conf on computer graphics pattern recogn. May 1975.

[GRAY78] Gray J.N.
"Notes on database operating systems".
Research report RJ2188, IBM Research laboratory, San Jose, February 1978.

[GRAY80] Gray J.N.
"A transactional model".
8th international conference on languages and programming, 1980.

[GRAY81] Gray J.
"The transaction concept : virtues and limitations".
VLDB, Cannes, Septembre 1981.

[GUTT78] Guttag J.V. et al.
"Abstract data types and software validation".
CACM December 78, Vol 21, n°12, pp 1048-1064.

[GUTT80a] Guttag J.V.
"Notes on type abstraction".
IEEE Transactions on Software Engineering, January 1980.

[GUTT80b] Guttag J., Horning J.J.
"Formal specification as a design tool".
Xerox Palo Alto Research Center, January 1980.

[HAMM75] Hammer H.M., McLeod D.J.
"Semantic integrity ina relational database system".
VLDB Conf. Framingham Mass September 1975.

[HOAR72] Hoare C.A.R.
"Proof of correctness of data representations".
Acta Informatica 1.4, 1972.

[ICHB80] Ichbiah J.D. et al.
"Rationale for the design of ADA programming language".
ACM Sigplan Notices, May 1980.

[ISO 79] ISO
"Reference Model of Open System Interconnection".
Note ISO/TC97/SC16/N227.

[LAMP78] Lamport L.
"Time, Clocks and ordering of events in distributed systems".
CACM vol 21 n° 7, July 1978.

[LEBI79] Le Bihan J. et al.
"Sirius-Delta : un prototype de bases de données distribuées".
International symposium on distributed databases, Paris,

[LIND79] Lindsay B.G. et al.
"Notes on distributed databases".
Research report RJ2571 IBM research laboratory San Jose July 1979.

[LISK75] Liskov B., Zilles S.
"Specification technique for data abstractions".
IEEE Transactions on Software Engineering, March 1975, pp 7-19.

[LISK77] Liskov B, Sydner A, Atkinson R, Schaffert J.
"Abstraction mechanism in CLU".
CACM 20,8 August 1977.

[LISK79] Liskov B.H.
"Primitives for distributed computing".
7th symposium on operating systems principles, Pacific Grove, Dec 1979.

[LISK81] Liskov B.H.
"On linguistic support for distributed programs".
Symposium on reliability in distributed software and database systems. Pittsburg, July 1981.

[LOCK79] Lockeman P. et al.
"Data abstractions for database systems".
ACM TODS Vol 4, n° 1. March 1979, pp 60-75.

[MENA79] Menasce D., Popek C.
"Locking and deadlock detection in distributed databases".
IEEE Trans. on Software Engineering, vol SE-5, N° 3,
Mai 1979, pp 195-202.

[MINO79] Minot R.
"ATM : un système de fabrication de programmes basé sur les concepts de modularité et de type abstrait".
Thèse 3ème cycle Université de Nancy, Mars 1979.

[POLY79] Cii Honeywell Bull, Laboratoire IMAG.
"Polyphème : un système de gestion de bases de données réparties".
Rapport de fin de contrat, Octobre 1979.

[RIDD80] Riddle W.E.
"An assessment of DREAM".
Software Engineering environment, North Holland pub Comp 1980.

[ROLL79] Rolland C., Leiffert S., Richard C.
"Tools for information system dynamics management".
IEEE Transactions on Software Engineering, 8/1979,
pp 251-261.

[ROLL81b] Rolland C.
"A methodology for information system design".
NCC 81, Chicago, Mai 1981.

[ROLL81c] Rolland C., Richard C.
"Une méthode de spécification conceptuelle".
Université Paris 1, 1981.

[ROLL81a] Rolland C., Thierry O.
"Constructions de base d'un langage de spécification de systèmes d'information".
Colloque sur les bases de données, Tunis, Avril 1981.

[ROLL81d] Rolland C.
"La conception des Bases de données réparties".
Projet SIRIUS, journées de présentation des résultats,
Paris, Novembre 1981.

[ROSE78] Rosenkrantz D.J. et al
"A system level concurrency control for distributed database systems".
ACM Transactions on Database Systems, vol 3 N° 2, Juin 78.

[SDD180] SDD1 Group.
"Technical reports".
Computer Corporation of America, Cambridge Mass. 1980.

[SIRI81]
"Une aplication bancaire".
Projet SIRIUS, Journées de présentation des résultats,
Paris, Novembre 1981.

[SMIT81] Smith J.M, Fox S., Landers T.
"Reference manual for ADAPLEX".
Technical report CCA-81-02, CCA Cambridge Mass. Jan.1981.

[SOGE 1] Société Générale
"Transfert étranger, Cahier des charges".
S.G. 112 av. Kleber, 75016 Paris, Mai 1977.

[SOGE 2] Société Générale
"Portefeuille étranger, Cahier des charges".
id [SOGE 1].

[SOGE 3] Société Générale.
"Transfert étranger, descriptif de saisie".
id [SOGE 1].

[SOGE 4] Société Générale.
"Portefeuille étranger, descriptif de saisie".
id [SOGE 1].

[STON77] Stonebraker M., Neuhold E.
"A distributed database version of INGRES".
Proc. 2nd workshop on Distributed Data Management and Computer Networks, Berkley, Mai 1977.

[STON79] Stonebraker M.
"Concurrency control and consistency of multiple copies of data in distributed INGRES".
IEEE Transactions on Software Engineering, Mai 1979.

[WASS79] Wasserman A.I.
"The data management facilities of PLAIN".
Lab of Medical Information Science, University of California San Francisco, 1979.

[WEBE78] Weber H.
"modularity in database system design : a software engineering view of database systems".
Proceedings VLDB. Berlin 1978, pp 65-91.

[WEIH83] Weihl W., Liskov B.
"Specification and implementation of resilient,atomic data-types".
Symposium on Programming Languages Issues in Software System, San Francisco Juin 83.

[WILE79] Wileden J.C.
"DREAM - an approach to designing large scale, concurrent software systems".
ACM National Conference, Octobre 1979.

[XERO78] Xerox.
"MESA language manual".
Xerox, Palo Alto Research Center, May 1978.

BIBLIOGRAPHIE DU PROJET SCOT

[SCOT 1]
"Présentation du projet transactionnel réparti SCOT".
RR SCOT n° 1, Mai 1979.

[SCOT 2]
"Projet SCOT : Système pour la COopération COhérente de Transactions".
Présentation au séminaire du Centre de Recherche CII-HB, 12-14 Septembre 1979. RR SCOT n° 2.

[SCOT 4]
"Transaction versus Database as a commercial approach to distributed information systems".
RR SCOT n° 4, Novembre 1979.

[SCOT 5]
"SCOT - a System for the COnsistent CO-operation of Transactions".
RR SCOT n° 5, Novembre 1979.

[SCOT 6]
"How the experience drawn from database leads to transaction co-operation".
RR SCOT n° 6, Novembre 1979.

[SCOT 7]
"An overview of update algorithms in distributed databases - Formalization with Nutt's evaluation nets".
RR SCOT n° 7, Mars 1980.

[SCOT 8]
"SCOT - Présentation générale des mécanismes de réalisation".
RR SCOT n° 8, Mai 1980.

[SCOT 9]
"Description du protocole SCOT, version 2".
RR SCOT n° 9, Janvier 1981.

[SCOT10]
"Méthode de contrôle de l'accès concurrent dans un système transactionnel réparti".
RR SCOT n° 10, Juin 1980.

[SCOT11] E. André, G. Bogo.
"Ada, abstract datatypes and distributed databases transactions".
RR SCOT n° 11, Aout 1980.
Proc. COMPSAC'80, Chicago 27-31 Octobre 1980.

[SCOT12] P. Decitre.
"A Concurrency control algorithm in a distributed environment".
RR SCOT n° 12, Octobre 1980.
Proc. NCC 81, Chicago, Mai 1981.

[SCOT13]
"Validation et reprise dans un système transactionnel coopérant".
RR SCOT n° 13, Novembre 1980.

[SCOT14] R. Balter.
"Selection of a commitment and recovery mechanism for a distributed transactional system".
RR SCOT n° 14, Janvier 1981.
Proc. Symposium on reliability in distributed software and database systems, Pittsburg, Juillet 1981.

[SCOT15]
"Un outil de spécification d'applications transactionnelles et une application pilote pour le projet SCOT".
RR SCOT n° 15, Février 1981.

[SCOT16]
"SCOT : un modèle transactionnel pour la gestion de données distribuées et la spécification d'applications".
RR SCOT n° 16, Mars 1981.

[SCOT17]
"Détection et prévention des interblocages dans un système transactionnel centralisé".
RR SCOT n° 17, Avril 1981.

[SCOT18]
"SCOT : A system for the consistent co-operation of transaction - overview -".
RR SCOT n° 18, Avril 1981.

[SCOT19]
"Adaptation du protocole SCOT à un réseau à diffusion fiable".
RR SCOT n° 19, Juin 1981.

[SCOT20]
"Prévention des interblocages dans un système transactionnel réparti".
RR SCOT n° 20, Juillet 1981.

[SCOT21]
"Conception d'applications pour systèmes transactionnels coopérants. Exemple : un système bancaire".
RR SCOT n° 21, Octobre 1981.

[SCOT22] R. Balter, P. Bérard, P. Decitre.
"Why control of the concurrency level in distributed systems is more fundamental than deadlock management".
RR SCOT n° 22, Mars 1982.
Proc.ACM Sigact-Sigops symposium on principles of distributed computing, Ottawa, Aout 1982.

[SCOT23]
"Design of a session level dedicated to co-operating transaction processing".
RR SCOT n°23, Octobre 1982.

[SCOT24] G. Bogo, J.C. Chupin.
"Application design for co-operative transaction processing systems".
Proc. NORDATA conf., Goeteborg, Juin 1982.

[SCOT25] R. Balter, G. Bogo, J.C. Chupin, P. Decitre
"Rapport final du projet SCOT".
Janvier 1983.

ANNEXE

UN EXEMPLE COMPLET :

SPECIFICATION D'UNE APPLICATION BANCAIRE

- ANNEXE -

Un exemple complet de spécification :

Une application bancaire

L'exemple que nous avons choisi de traiter décrit une partie de l'application bancaire présentée au chapitre 4. Il est extrait de la spécification complète donnée dans [SCOT15] et [SCOT21].

Cet exemple décrit tous les objets et les transactions impliqués dans un transfert entre comptes clients de banques résidant dans des pays distincts.

TYPES DE BASE

On a regroupé ici les définitions de toutes les structures de données utilisées dans cette description.

Il était aussi possible de définir ces structures uniquement dans les services où elles sont utilisées. Mais beaucoup apparaissent dans les échanges de messages (paramètres).

Ces structures sont simplifiées et ne représentent que les données nécessaires à notre spécification. Les valeurs affectées ainsi que les types associés le sont à titre d'exemple et peuvent être très différents des données réelles utilisées par les banques. En ce qui concerne les opérations étranger, on a essayé de se rapprocher le plus possible des structures de données réelles [SOGE3, SOGE4].

```
package TYPES_DE_BASE is

type NUMERO_DE_COMPTE is
   record
     NUMERO_AGENCE : string (1..4);
     NUMERO_CPT : string (1..6);
     CLE : string (1..2);
   end record;


type TYPE_DEVISE is (DUS,FF,LS,LI,DM,FB);


type COURS_INFO is
   record
     DEVISE : TYPE_DEVISE;
     COURS : integer;
   end record;

type COURS_DEVISE_INFO is
   array (TYPE_DEVISE'first .. TYPE_DEVISE'last) of COURS_INFO;


type TYPE_INSTRUCTION is (NORMAL,OPPOSITION_DEBIT,OPPOSITION_COMPTE,
                          CLIENT_DECEDE,COMPTE_FERME);


type TYPE_IDENTIFICATION is
   record
      NOM : string (1..30);
      ADRESSE : string (1..120);
      TELEPHONE : string (1..12);
   end record;


type VIREMENT_INFO is
   record
      CLIENT_EMETTEUR : TYPE_IDENTIFICATION;
      COMPTE_CLIENT_EMET : NUMERO_DE_COMPTE;
      CLIENT_RECEPTEUR : TYPE_IDENTIFICATION;
      COMPTE_CLIENT_RECEP : NUMERO_DE_COMPTE;
      DATE_DE_VALEUR : DATE;
      MONTANT : integer;
      RAISON  : string (1..40);
   end record;
```

```
type COMPTE_CLIENT;
type COMPTE_CLIENT_A is access COMPTE_CLIENT;

type PROVISION;
type PROVISION_A is access PROVISION;

type REMISE_EXPORT;
type REMISE_EXPORT_A is access REMISE_EXPORT;

type REMISE_IMPORT;
type REMISE_IMPORT_A is access REMISE_IMPORT;

type COMPTE_BANQUE;
type COMPTE_BANQUE_A is access COMPTE_BANQUE;

type COMPTE_CLIENT_INFO is
   record
      NUMERO : NUMERO_DE_COMPTE;
      IDENTIF_CLIENT : TYPE_IDENTIFICATION;
      INSTRUCTIONS : TYPE_INSTRUCTION;
      TARE : integer := 0;
      INDISPONIBLE : PROVISION_A;
      SOLDE : integer := 0 ;
      RESERVE : integer := 0;
      COMPTE_REF : COMPTE_CLIENT_A;  -- accès au compte lui même
   end record;


type PROVISION_INFO is
   record
      MONTANT : integer;
      DATE_DE_VALEUR : DATE;
      COMPTE_A_CREDITER : COMPTE_CLIENT_A := null ;
      COMPTE_A_DEBITER : COMPTE_BANQUE_A;
      PROVISION_SUIVANTE : PROVISION_A := null;
   end record;


type COMPTE_CORRESPONDANT;
type COMPTE_CORRES_A is access COMPTE_CORRESPONDANT;
```

```
type COMPTE_CORRES_INFO is
   record
      NUMERO : NUMERO_DE_COMPTE;
      IDENT_CORRESP : TYPE_IDENTIFICATION;
      CREDIT : integer;
      INDISPONIBLE : PROVISION_A;
      SOLDE : integer;
      RESERVATION : RESERVE_A;
      COMPTE_REF : COMPTE_CORRES_A;
   end record;


type TRANSFERT_INFO is
   record
      AGENCE_ORIGINE : AGENCE;
      CLIENT_EMETTEUR : TYPE_IDENTIFICATION;
      CPT_CL_EMET : NUMERO_DE_COMPTE;
      CLIENT_DESTINATAIRE : TYPE_IDENTIFICATION;
      CPT_CL_DEST : NUMERO_DE_COMPTE;
      BQ_DESTINATAIRE : BANQUE;
      BQ_COUVERTURE : BANQUE;
      MONTANT : integer;
      DEVISE : TYPE_DEVISE;
      MONTANT_FRAIS : integer;
      DATE_VALEUR : DATE;
      CPT_TRANSIT : NUMERO_DE_COMPTE;
      CPT_CORRESPONDANT : NUMERO_DE_COMPTE;
      OPER_ETRANGER : REF_OP_ETRANGER;
      CHANGE : boolean;
      CONTREVALEUR_FF : integer;
   end record;


end TYPES_DE_BASE;
```

compte_banque

Au § 5.5 est décrit le fonctionnement d'un compte de la banque. Aucune garde n'est associée aux transactions permettant de le manipuler. Toutes les opérations de crédit et de débit sont autorisées quel que soit le solde.

---

compte_client

On décrit ici le fonctionnement d'un compte de client. On peut constater que la transaction CREDITER n'est pas visible de l'extérieur du compte. Un crédit s'opère en deux temps : un approvisionnement qui crée un objet provision attaché au compte et qui à la date de valeur déclenchera la transaction CREDITER.

```
task type COMPTE_CLIENT is
|
    agent DEBITER (M : in integer);
    agent OUVRIR (C : in COMPTE_CLIENT_INFO);
    agent FERMER;
    agent MODIFIER_INSTRUCTION (I : in TYPE_INSTRUCTION);
    agent APPROVISIONNER (M : in integer; D : in DATE_VALEUR;
                          CB : in COMPTE_BANQUE);
    agent RESERVER (M : in integer);
    agent RESTITUER (M : in integer);
    agent CONSULTER (S : out integer);
|
end COMPTE_CLIENT;


task body COMPTE_CLIENT is
|
    use TYPES_DE_BASE;

    -- l'opération CREDITER n'est pas visible de l'exterieur.
    agent CREDITER (M : in integer, P : in PROVISION);

    CPT : COMPTE_CLIENT_INFO;
    procedure CHAINER (P : in PROVISION) is separate;
    procedure DECHAINER (P: in PROVISION) is separate;
    task PROVISION is separate;
    task body PROVISION is separate;

  begin

    agent OUVRIR (CPT : in COMPTE_CLIENT_INFO) do
    | -- initialisation des informations contenues dans CPT.
    end OUVRIR;
    end agent;

    loop
      select

        agent FERMER do
        | CPT.INSTRUCTIONS := "COMPTE_FERME";
        end FERMER;
        end agent;
      or
|
:
```

```
  when CPT.INSTRUCTIONS = "NORMAL" =>
  agent DEBITER (M : in integer) do
  |
  |  if CPT.SOLDE >= M then
  |     CPT.SOLDE := CPT.SOLDE - M;
  |  else   -- aviser opérateur pour forçage ou non
  |  endif;
  end DEBITER;
  end agent;

or when CPT.INSTRUCTIONS ="NORMAL" =>
  agent RESERVER (M :in integer)
  |
  |  CPT.SOLDE := CPT.SOLDE - M;
  |  CPT.RESERVE := CPT.RESERVE + M;
  end agent;

or
  agent RESTITUER (M : in integer)
  |  CPT.RESERVE := CPT.RESERVE - M;
  |  CPT.SOLDE := CPT.SOLDE + M;
  end agent;

or when CPT.INSTRUCTIONS /= "OPPOSITION_COMPTE" =>
 agent APPROVISIONNER (M : integer;D : DATE_DE_VALEUR;
  |                          CB : in COMPTE_BANQUE) do
  |
  |  P := new PROVISION;
  |  CHAINER (P);
  |
  |  start_agent P.ENREGISTRER (M,D,CPT.COMPTE_REF;P;CB);
  |
  end APPROVISIONNER;
  end agent;

or when CPT.INSTRUCTIONS /= "OPPOSITION_COMPTE" =>
   agent CREDITER (M : in integer, P : in PROVISION)
   |
   |  DECHAINER (P);
   |  CPT.SOLDE := CPT.SOLDE + M ;
   |
   end CREDITER;
   end agent;

or
```

```
        agent MODIFIER_INSTRUCTION (I : in TYPE_INSTRUCTION)
        |
        | CPT.INSTRUCTIONS := I;
        end agent;

    or
        agent CONSULTER (S :in integer)
        | S := CPT.SOLDE;
        | -- on peut aussi donner la liste ou le montant total
        | -- des provisions attachées au compte.
        end agent;

    else

        -- problème avec le compte (bloque ou autre pb)
        -- aviser opérateur qui prendra des décisions
    end select;
  end loop;

end COMPTE_CLIENT;
```

provision

```
task type PROVISION is
    agent ENREGISTRER (M : in integer; D : in DATE_VALEUR;
                       C : in COMPTE_CLIENT; P : in PROVISION;
                      CB :in COMPTE_BANQUE);
    agent VALIDER;
end PROVISION;

task body PROVISION is
    PROV : PROV_INFO;
    PR_PT : PROVISION_A
  begin
    agent ENREGISTRER (M;D;C;P) do
      PROV.MONTANT := M;
      PROV.DATE_DE_VALEUR := D;
      PROV.COMPTE_A CREDITER := C;
      PROV.COMPTE_A_DEBITER := CB;
      PR_PT := P;

      start_agent DATEUR.DEMANDER_REVEIL (P;D);

    end ENREGISTRER;
    end agent;

    agent VALIDER do
      start_agent PROV.COMPTE_A_DEBITER.DEBITER (M);
      start_agent PROV.COMPTE_A_CREDITER.CREDITER (M; PR_PT);
    end VALIDER;
    end agent;
end PROVISION;
```

agence

La description suivante donne les principales fonctionnalités de l'objet AGENCE. Une description complète sort du cadre de cet exemple.

```
task type AGENCE is

    -- description du fonctionnement de l'agence et non pas de sa
    -- structure.

    agent OUVERTURE_AGENCE (CREDIT :in integer);
    agent CLOTURE_AGENCE (MONTANT_CAISSE : out integer;
                          MONTANT_SGSA : out integer);
    agent FERMETURE_AGENCE;
    agent COURS_DEVISE (COTATION : in COURS_DEVISE_INFO);
    agent AVIS_TRANSFERT_RECU (T : in TRANSFERT_INFO;
                               T_REF : in TRANSFERT_A);

end AGENCE;

task body AGENCE is

    COTATION : COURS_DEVISE_INFO;

  begin

    -- lancement de toutes les activités de l'agence
    CAISSE := new COMPTE_BANQUE;
    SGSA := new COMPTE_BANQUE;
    GUICHET_1 := new GUICHET;
    GUICHET_2 := new GUICHET;

A:  loop
      agent OUVERTURE_AGENCE (CREDIT) do

        -- à l'ouverture, chaque agence reçoit un certain crédit
        -- du siège.
        start_agent SGSA.CREDITER (CREDIT);

        -- initialisation de la caisse.

      end OUVERTURE_AGENCE;
      end agent;
```

```
B:    loop
        select

          agent COURS_DEVISE (COTATION : COURS_DEVISE_INFO)
          | -- Réception une fois par jour du cours de chaque devi
          | -- Réveil de toutes les opérations en attente.
          end agent;
        or
          agent CLOTURE_AGENCE (MONTANT_CAISSE;MONTANT_SGSA)  do
          |
          | start_agent CAISSE.CONSULTER (MONTANT_CAISSE);
          | start_agent SGSA.CONSULTER (MONTANT_SGSA);
          |
          | -- ces valeurs vont pouvoir être enregistrées à l'agen
          | -- ainsi qu'au siège pour constitution du bilan.
          | exit B;
          |
          end CLOTURE_AGENCE;
          end agent;

        or
          agent FERMETURE_AGENCE do
          |
          | -- liquider les comptes et les activités de l'agence
          | exit A;
          |
          end FERMETURE_AGENCE;
          end agent;

        or
          agent AVIS_TRANSFERT_RECU (T, T_REF) do
          |
          |   TR := new TRANSFERT_RECU;
          |   start_agent TR.PRISE_EN_CHARGE(T, T_REF);
          |
          end AVIS_TRANSFERT_RECU;
          end agent;

        end select;
      end loop;
    end loop;

end AGENCE;
```

---

compte_loro

Les comptes LORO et NOSTRO sont les comptes des correspondants étrangers et ceux de la banque chez ces correspondants. Leur fonctionnement est très proche de celui des comptes clients (mécanisme de date de valeur).

```
task type COMPTE_LORO is
    agent OUVRIR;
    agent FERMER;
    agent CREDITER (M :in integer; P : in PROVISION);
    agent DEBITER (M : in integer; R : in RESERVE);
    agent APPROVISIONNER (M : in integer; D : in DATE_VALEUR
                          CB : in COMPTE_BANQUE);
    agent RESERVER (M :in integer; D : in DATE_VALEUR;
                    CB : in COMPTE_BANQUE);
    agent LIGNE_CREDIT (M : in  integer);
    agent CONSULTER (S : out integer);
end COMPTE_LORO;
```

guichet

```
task type GUICHET is
|
|agent VIREMENT_INTERNE (CL_ORIGINE : in TYPE_IDENTIFICATION;
|             CL_DESTIN : in TYPE_IDENTIFICATION; M : in integer);
|agent VIREMENT_EXTERNE (CL_ORIGINE : TYPE_IDENTIFICATION;
|                        BANQUE_DESTIN : in BANQUE; M : in integer)
|agent REMISE_CHEQUE;
|agent DEPOT_ESPECES;
|agent RETRAIT_ESPECES;
|agent REMISE_EXPORT;
|agent REMISE_IMPORT;
|agent DEMANDE_TRANSFERT;
|agent OUVRIR_COMPTE_CLIENT;
|
end GUICHET;

task body GUICHET is
|
|
|   -- Phase d'initialisation.
|   -- Un guichet est associé à un terminal. Tous les agents
|   --  exécutés  ici  correspondent  à  des  opérations  bancaires.
|   -- Il est nécessaire d'assurer le suivi de l'opération jusqu'à
|   -- sa fin et exécuter des reprises en cas de problèmes rencon-
|   -- trés à l'exécution.
|
|   CPT_CL : COMPTE_CLIENT_INFO;
|   V_INFO : VIREMENT_INFO;
|   T_INFO : TRANSFERT_INFO;
|   TRANS_PILOT : operation_task;
|   J : TYPE_ARRAY_JOURNAL;
|
begin
|   loop
|     select
|       agent OUVRIR_COMPTE_CLIENT do
|       |
|       |  -- saisie des  paramètres y compris le numéro
|       |  -- de compte et initialisation de CPT_CL.
|       |
|       |  C := new COMPTE_CLIENT;
|       |
|       |
|
```

```
    -- conserver la référence C et le numéro de compte par
    -- exemple dans un dictionnaire de références de comptes.

    start_agent C.OUVRIR (CPT_CL);

  end OUVRIR_COMPTE_CLIENT;
  end agent;

or
  agent VIREMENT_INTERNE (CL_ORIGINE; CL_DESTIN,M) do

    -- initialisation de V_info
    V := new VIREMENT;
    start_agent V.INIT_VIREMENT (V.INFO);

  end VIREMENT_INTERNE;
  end agent;

or
  agent DEMANDE_TRANSFERT do

    -- saisie des informations nécessaires à T_INFO.

    T := new TRANSFERT_SIMPLE_EMIS;

    TRANS_PILOT := start_opération;
                   -- lancement d'une opération
    start_transaction T.PRISE_EN_CHARGE (T_INFO);

    if TR.MONTANT*COURS_DU_JOUR (TR_INFO.DEVISE) >
                                          MAX_COURS_DU_JOUR
    then
      start_transaction DEMANDE_NEGOCIATION (T_INFO);

      -- ici on doit attendre la réception du résultat de
      -- la négociation et agir en fonction de son résultat.

      -- Si négociation acceptée :
      start_agent T.IMPUTER_TIRE (T_INFO);

      -- sinon faire une reprise.
    end if;
```

```
            attente_fin_toutes_transaction (J);
                        -- attente de la fin de trans. globales

            -- consultation du journal J pour connaitre le
            -- résultat de l'exécution des transactions. Si l'une
            -- d'elles s'est avortée, reprendre son exécution
            -- ou défaire les modifications de celles qui sont
            -- validées.

          end DEMANDE_TRANSFERT;
          end agent;

        end select
      end loop;

end GUICHET;
```

transfert_simple_emis

```
task type TRANSFERT_SIMPLE is
|
|    -- Traitement d'un transfert en agence
|    -- cette tâche assure le suivi de l'exécution du transfert
|    -- à l'agence, à la DAIT et au BCC.
|
|    agent PRISE_EN_CHARGE (T : in TRANSFERT_INFO);
|    agent COURS_DEVISE (C :in TYPE_COURS);
|    agent IMPUTER_TIRE (TR : in TRANSFERT_INFO);
|    agent PEC_DAIT_OK;
|
end TRANSFERT_SIMPLE

task body TRANSFERT_SIMPLE is
|
|    function COURS_DU_JOUR (D :in TYPE_DEVISE) return integer
|                                                    is separate;
|    function CALCUL_FRAIS (T : in TRANSFERT_INFO) return integer
|                                                       is separate;
|    procedure IMPUTATION is
|     |begin
|     |  start_agent TR.CPT_CL_EMET.DEBITER (MONTANT_TOTAL_DEBIT);
|     |
|     |  start_agent TR.CPT_TRANSIT.CREDITER (MONTANT_TOTAL_CREDIT);
|     |
|     |  start_agent TR.CPT_FRAIS.CREDITER (TR.MONTANT_FRAIS);
|     |
|     |
|     |  -- Avis a la DAIT pour enregistrement de l'opération dans
|     |  -- ses livres comptables
|     |
|     |  start_agent DAIT.DT.TRANSFERT_ENREGISTREMENT
|     |                                                        (TR);
|     | end IMPUTATION;
|
|    TR : TRANSFERT_INFO;
|    MONTANT_TOTAL_CREDIT : integer;
|    MONTANT_TOTAL_DEBIT : integer;
|    NON_IMPUTE : boolean  := true;
|
```

```
begin

  agent PRISE_EN_CHARGE (TR) do
  |
  |  TR.MONTANT_FRAIS := CALCUL_FRAIS (TR);
  |
  |  -- Création et initialisation d'une tâche DT à la DAIT
  |  -- assurant le transfert dans ce service. Cette tâche pourra
  |  -- être créée soit directement depuis l'agence soit par une
  |  -- agent activée par l'agence à la DAIT
  |
  |  -- Activation de cette tache DT pour réaliser l'application
  |  -- du transfert c'est à dire le crédit ou débit des comptes
  |  -- LORO ou NOSTRO.
  |
  |  start_agent DAIT.CREER_TRANSFERT_EMIS (TR);
  |  NON_IMPUTE := false;
  |
  end PRISE_EN_CHARGE;
  end agent;

 agent PEC_DAIT_OK do
  |
  |  start_agent DAIT.DT.TRANSFERT_APPLICATION (TR);
  |
  |  if TR.DEVISE = "FF" then
  |     if TR.CHANGE then
  |
  |        MONTANT_APPROX := TR.MONTANT*COURS_DU_JOUR (TR.DEVISE);
  |        CHANGE_INFO := ELABORER_INFO_BCC
  |
  |        if MONTANT_APPROX > MAX_COURS_DU_JOUR then
  |
  |           -- Pour une contrevaleur supérieure à 10000 FF
  |           -- l'opération de change doit être negociée au BCC
  |
  |           -- réservation sur le compte du client tiré
  |
  |         start_agent TR.CPT_CL_EMET.RESERVER
  |                              (MONTANT_APPROX+TR.MONTANT_FRAIS);
  |
  |           -- négocier le cours avec le BCC
  |           N := new NEGOCIATION;
  |           return;
  |
  :
```

```
        else
           -- si montant faible on fait l'opération de change
           -- au cours du jour
           -- et on avise le BCC qu'il doit prévoir un achat
           -- de devises

           TR.CONTREVALEUR := MONTANT_APPROX;
           start_agent BCC.AVIS_ACHAT (CHANGE_INFO)
        endif

        MONTANT_TOTAL_DEBIT := TR.CONTREVALEUR
                                          + TR.MONTANT_FRAIS;
        MONTANT_TOTAL_CREDIT := TR.CONTREVALEUR;

     else
        MONTANT_TOTAL_DEBIT := TR.MONTANT + TR.MONTANT_FRAIS;
        MONTANT_TOTAL_CREDIT := TR.MONTANT;

     endif
  else
     MONTANT_TOTAL_DEBIT := TR.MONTANT + TR.MONTANT_FRAIS;
     MONTANT_TOTAL_CREDIT := TR.MONTANT;
  endif

  IMPUTATION;

end PEC_DAIT_OK;
end agent;

 select
   when NON_IMPUTE =>
   agent IMPUTER_TIRE (TR) do

     start_agent TR.CPT_CL_EMET.RESTITUER
                              (MONTANT_APPROX+TR.MONTANT_FRAIS);
     IMPUTATION;

   end IMPUTER_TIRE;
   end_agent;

 or
   terminate;
 end select;

end TRANSFERT_SIMPLE_EMIS;
```

d_transfert_simple_emis

```
task type D_TRANSFERT_SIMPLE_EMIS is
|
|    -- Traitement du transfert à la DAIT :
|    -- application du transfert cad maj des comptes LORO ou NOSTRO
|    -- du correspondant (destinataire ou couverture)
|    -- Enregistrement du transfert dans les livres comptables
|
|    agent TRANSFERT_APPLICATION (T : in TRANSFERT_INFO);
|    agent TRANSFERT_ENREGISTREMENT (T : in TRANSFERT_INFO);
|
end D_TRANSFERT_SIMPLE_EMIS;


task body D_TRANSFERT_SIMPLE_EMIS is
|
|    T : TRANSFERT_INFO;
|
begin
|   start_agent T_REF.PEC_DAIT_OK;
|
|   agent TRANSFERT_APPLICATION (T) do
|   |
|   |  if T.DEVISE = "FF" then
|   |     -- approvisionner le compte LORO du correspondant en ff
|   |
|   |     start_agent T.CPT_CORRESPONDANT.APPROVISIONNER
|   |                 (T.MONTANT; T.DATE_VALEUR; T.CPT_TRANSIT);
|   |  else
|   |     -- débiter notre compte NOSTRO chez le correspondant pour
|   |     -- cette devise. On fait l'opération sur le compte miroir.
|   |
|   |     T.CPT_TRANSIT := NUMERO_COMPTE_PC (T.DEVISE);
|   |
|   |     start_agent T.CPT_CORRESPONDANT.RESERVER
|   |                 (T.MONTANT; T.DATE_VALEUR; T.CPT_TRANSIT);
|   |  endif
|   |
|   |  -- Création d'un avis de transfert et envoi à la banque
|   |  -- T.BQ_DESTINATAIRE  ou T.BQ_COUVERTURE
|   |
|   end TRANSFERT_APPLICATION;
|   end agent;
|
```

```
    agent TRANSFERT_ENREGISTREMENT (T)
    |
    |   -- A ce moment l'opération de transfert est terminée et T
    |   -- contient toute l'information pour l'enregistrement dans
    |   -- les livres comptables.
    |
    end agent;
|
end D_TRANSFERT_SIMPLE_EMIS;
```

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 5 de l'arrêté du 16 avril 1974

VU les rapports de présentation de

. Madame C. ROLLAND, Professeur
. Monsieur S. ABITEBOUL
. Monsieur J.C CHUPIN
. Monsieur M. LEONARD, Professeur

Monsieur BOGO Gilles

est autorisé à présenter une thèse en soutenance en vue de l'obtention du grade de DOCTEUR D'ETAT ES SCIENCES.

Fait à Grenoble, le 14 mai 1985

Le Président de l'U.S.M.G

22 MAI 1985

Le Président

M. TANCHE

UNIVERSITÉ SCIENTIFIQUE ET MÉDICALE DE GRENOBLE

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,