Conception et réalisation d'outils horizontaux complétant les méthodologies de Systèmes d'Information orientées–objet

S. BALDE(1), C. CLERCIN(1), T. DE VALLOIS(2), O. SARR(1)

(1)
ENSUT Département Informatique
BP 5085
DAKAR
Sénégal
fax : (221) 25 75 28
clercin@ensut.ensut.sn, tusne@ensut.ensut.sn

(2)
ENSETP
Camp Claudel
DAKAR
Sénégal

Résumé :

Des outils horizontaux sont proposés pour guider le développeur dans le processus d'analyse, de conception et de réalisation d'un SI avec une méthodologie OO. Une approche OO du cycle de vie et des processus de développement est discutée et des outils dans un environnement tout objet sont proposés. La démarche insiste sur un processus de développement incrémental à base de prototypes et s'inspire, en les adaptant, des recherches les plus récentes dans le domaine. L'implémentation des outils est effectuée dans l'environnement Windows/C++.

Mots–clés : Méthodes SI orientées OO, AGL OO, Méthode O*, outils horizontaux

INTRODUCTION

Les approches orientées–objet (OO) sont de plus en plus reconnues dans la définition des problèmes informationnels et couvrent aujourd'hui des domaines variés (langages, SGBD, systèmes d'exploitation etc.). Toutefois ces approches sont aujourd'hui incomplètement maîtrisées dans l'analyse et la conception des Systèmes d'Information (SI). Certes des propositions commerciales commencent à voir le jour (OOA, OOD, HOOD, MACH2, SYS–PO) mais aucune d'entre elles ne couvre tout le cycle de vie du SI. La plupart font un mapping d'approches pour aborder deux niveaux différents du cycle (notamment approches cartésiennes et approches OO). Or ces approches sont totalement opposées dans leur démarche et le produit obtenu n'est pas convaincant. [SUT91] fait remarquer dans une étude comparative des méthodes cartésiennes et des méthodes OO qu'il n'est pas réaliste de croire que l'utilisation d'une méthode cartésienne ne gauchira pas l'implémentation d'un système OO. [WIR91], après avoir développé un exemple détaillé suivant la méthode SA et une méthode OO démontre que les spécifications obtenues sont tellement différentes qu'elles ne devraient pas être utilisées en même temps (les méthodes sont certes comparables mais incompatibles).

Nous nous intéressons ici à une approche OO qui couvre tout le cycle de vie du SI et travaillons à la réalisation d'outils appuyant cette approche. Ces outils ont toutes les caractéristiques des éléments d'un véritable atelier de génie logiciel (AGL) OO.

Nous expliquons d'abord notre approche du problème avant de présenter l'architecture fonctionnelle de nos outils et de détailler l'implémentation de nos outils.

I. NOTRE APPROCHE METHODOLOGIQUE

I.1 Sur les outils

La plupart des AGLprésents sur le marché présentent des solutions partielles au problème de la productivité du développement. Leur noyau permet une modélisation conforme aux formalismes des méthodes, une production de maquettes et de tables de bases de données et autorise souvent une vérification des produits.

Un AGL, système d'outils logiciels pour le développement d'applications, ne saurait toutefois se résumer en la saisie de spécifications textuelles et la production de diagrammes, production statique de textes "morts" et de schémas passifs.

On constate que les aspects de gestion du projet, de gestion du processus préconisés par la méthode sous-jacente sont souvent absents.

Un AGL doit d'abord intégrer des aspects structurels dans son dictionnaire (l'univers du discours, le domaine de l'implémentation, l'administration de l'objet en développement, le projet de développement) [WIN79].

Au-delà, l'intégration de l'aspect dynamique de la gestion du processus exige que le modèle de processus (la sémantique du processus de développement) soit décrit dans l'outil, qu'il serve d'assistant et de régulateur de l'activité du développeur.

Nous proposons d'intégrer un gestionnaire du processus dans notre prototype d'outil ; il doit être davantage qu'un compagnon méthodologique renvoyant à l'utilisateur un diagnostic sur la violation des règles de la méthode [VES&AL92]. Il sera visible de l'interface et composé de trois éléments :

- un élément actif gérant les aspects permissifs (à la demande, il propose des décisions en fonction du contexte),
- un élément d'arrière-plan gérant les aspects proscriptifs (lié aux diagnostics interactifs et différés : sur les contrôles et la vérification/validation),
- un élément passif gérant l'aide en ligne (sur les concepts, l'utilisation de l'outil, la connaissance de la démarche).

I.2 Sur les modèles de processus.

La force d'une méthode de développement réside en particulier dans les règles et les contraintes qu'elle impose sur

- les produits du développement ("deliverable"),
- le déroulement du travail (les étapes ou activités),
- les décisions à prendre à chaque moment.

Les modèles de processus sont des représentations abstraites des actes d'un projet : le développeur doit transformer des phénomènes réels et des besoins exprimés en des spécifications conceptuelles générales puis détaillées du SI.

Différents modèles de processus ont été présentés depuis le "code and fix" qui est en soi l'absence de modèle : le modèle cascade [ROY70, de la spirale [BOE87], de la spirale hiérarchisée[IIV90], de la fontaine [HEN90].

Les voies de recherches récentes tentent de prendre encore plus de recul sur la théorie du projet logiciel, en élaborant un méta-modèle de

processus. Ces recherches amènent à considérer trois types de modèles [DOW87], qui dépassent en l'englobant la typologie précédente :

Dans les **modèles centrés sur les produits**, le point de vue est celui de la progression du produit à livrer. Le processus, suite d'activités de transformation (des traitements), reste en arrière–plan et ne rend pas compte de la sémantique propre de ces activités.

Dans les **modèles centrés sur l'activité**, c'est un plan d'actions en vue d'un objectif qui est mis en vedette alors que le lien avec le produit à liver n'est pas explicite.

Ces deux premiers types de modèles recoupent les processus cascade, spirale et fontaine.

Une troisième approche centre le **modèle sur les aspects liés à la décision**. Ainsi, toute transformation du produit dans le cours du développement est–elle la conséquence d'un acte de décision. Cette approche explique non seulement ce qui est fait et comment on le fait, mais aussi pourquoi c'est fait. Cependant, la définition du produit reste en arrière–plan.

Le **Modèle Objet Evolutif** de [MOR&SOU93] veut rendre compte de l'évolution du produit, en s'appuyant sur le modèle de processus orienté–décision et en couplant fortement le produit et le processus.

[ROL&PRA93] postulent qu'un développement informatique particulier est une instance de situation générique de développement. Dans ce cadre, sont définies des correspondances entre macro–contextes de situations et stratégie d'une part, entre micro–contextes de situations et tactiques d'autre part. De telles considérations pourraient être étendues facilement pour fonder un macro–modèle de processus opérationnel au stade de la conception [ROL93], voire aux phases ultérieures.

Ces recherches sur la modélisation du processus apportent une meilleure compréhension et communication de tout le processus de création de logiciel et peuvent aider à concevoir les outils logiciels intégrés articulant des fonctionnalités liées aux produits (modélisations graphiques, documentations internes et externes, intermédiaires et finales) à une assistance dynamique pour suivre la démarche de la méthode utilisée.

I.3 Notre approche.

Toute modélisation doit intégrer les aspects statiques et dynamiques tout en évitant la séparation artificielle des données et des traitements. Les méthodes systémiques ont proposé des solutions intéressantes. Aujourd'hui, du point de vue de la réutilisabilité, les techniques OO ont montré leurs forces [BOO92]. Selon nous, une véritable

méthode OO dans les SI doit avoir bien entendu la philosophie objet (classes, héritage, encapsulation, polymorphisme) mais aussi s'inspirer des travaux antérieurs sur les méthodes systémiques.

A titre expérimental, nous appuyons nos travaux sur la méthode O*[BRU93] issue du projet BUSINESS CLASS du programme ESPRIT, intégrant les travaux sur REMORA [ROL&FOU86] et les techniques OO de MEYER [MEY88].

La modélisation O* consiste à regrouper, par le mécanisme de classification, les objets perçus dans l'univers du discours en des schémas d'objets (classes). L'étude locale permet l'affinage d'une classe indépendamment des autres, tandis que l'étude globale permet de modéliser les liens et les contraintes statiques (héritage, composition, référence) puis les liens et les contraintes dynamiques (opérations, événements).

La dynamique de la méthode O* [BRU93] correspond au modèle fontaine, qui est une itération d'activités sur les classes en cours de développement. On distingue six activités (étapes) flexibles qui constituent autant de points de vue du développeur sur l'application–cible : inventaire des objets du monde réel, description initiale des objets, étude locale des classes, étude globale des aspects statiques, puis des aspects dynamiques, validation et vérification.

Dans cette démarche, le développeur peut travailler à ses classes en profondeur ou en largeur sans être contraint de terminer une tâche avant de passer à une autre (il n'y a que des classes et elles se trouvent à des états divers de développement). Le processus ressemble sous certains aspects au "code and fix" appliqué aux stades de l'analyse et de la conception ; le modèle de processus est en quelque sorte "immergé", caché par l'aspect incrémental et itératif du prototypage de la méthode. De ce fait, on pourrait ne pas se préoccuper de la démarche. Cependant, un type de processus existe qu'il s'agit de modéliser avant de l'implémenter.

L'atelier guidera le développeur dans ses productions grâce à des rappels de la méthode O*, à des propositions de tâches réalisables en fonction du contexte ; il permettra ainsi un pilotage souple et intelligent du cycle de vie du logiciel–cible.

II. ARCHITECTURE FONCTIONNELLE DES OUTILS

L'activité du développeur est perçue comme un processus incrémental de prototypage progressif pour parvenir à l'application–cible. Dans ce cadre, il prend des décisions contextuelles qui instancient un modèle de processus fondé sur le quadruplet <situation–décision–argument–action>.

Il part d'une spécification floue et détaille progressivement les besoins logiciels. On intégrera l'aspect décisionnel comme élément central de la démarche suivant l'approche de modélisation de processus du projet NATURE [NAT92], [SCH93]. Le développeur parviendra rapidement à des maquettes de ses fonctionnalités (description statique et dynamique de ses classes) puis à des prototypes (ébauches d'IHM liés aux classes décrites).

Le modèle de processus que nous implémentons sur la méthode O* s'appuie sur les trois pôles "produit", "activité" et "décision", mais il privilégie le point de vue décisionnel.

Pôle produit.

L'activité conceptuelle consiste à réaliser deux produits : un modèle statique et un modèle dynamique. Ce pôle met donc en avant la production du travail de conception : l'output est la construction de deux modèles.

Pôle activités.

Les produits sont élaborés au cours des six étapes présentées plus haut, dont trois étapes centrales, itératives et incrémentales : l'étude locale des classes (spécification en profondeur d'aspects statiques et dynamiques locaux d'une classe), l'étude globale des aspects statiques (graphe statique) et l'étude globale des aspects dynamiques (graphes dynamiques).

Une méthode définit l'ordre dans lequel les diverses activités de développement doivent être poursuivies. L'outil offre donc les facilités de navigation qui aident le développeur à respecter la démarche O* ; il peut, dans ce cadre incrémental, choisir de développer une classe en profondeur ou de développer en largeur sur toutes les classes. Cette souplesse nécessite une vérification différée du degré de complétude du travail.

Pôle décisions.

Toute transformation d'une classe est la conséquence d'une décision du développeur. Cette décision est permise ou prescrite par la méthode employée ; elle dépend du contexte dans lequel il se trouve, i.e. de l'étape ; on couple donc fortement des ensembles de décisions–type à des contextes.

L'élément central du processus suivi par le développeur est la décision contextuelle. Les décisions–type contextuelles sont d'abord les concepts utilisés dans la méthode. Concrètement, les actions sont tout ce qui est pris en charge par les commandes présentes dans les menus de l'outil. D'un point de vue macroscopique, le processus suivi est le modèle fontaine ; d'un point de vue microscopique, à l'intérieur de chaque étape, il suit le modèle proposé par [NAT92].

De fait, le modèle de processus symbolisé par le quadruplet <situation–décision–argument–action> constitue un raffinement du modèle

global (ici le modèle fontaine), sujet à description plus détaillée pour arriver à prescrire des unités de décisions et des actions atomiques.

Exemple dans la spécification locale de classe :

situation : classe dispose d'attributs statiques
décision possible : modification du nom d'un attribut statique
argument : l'attribut doit préexister, le nouveau nom ne doit pas préexister
action : changer le nom ancien dans le nouveau nom

Spécifications de l'Aide.

Si le gestionnaire de processus est l'assistant actif du CASE, l'aide en est l'assistant passif ; elle est en effet déclenchée sur l'initiative du développeur (la gestion du processus est sous le contrôle de l'outil). L'aide n'est pas un supplément d'âme du logiciel ; dans le cas d'un CASE, c'en est l'âme même. Elle est conçue de manière intrinsèque à l'outil, comme une fonctionnalité indispensable au même titre que l'interface graphique ou le dictionnaire. Elle constitue donc une classe logicielle.

L'aide à la démarche.

L'aide à la démarche est un guide pédagogique du le processus de développement. Elle présente un cours raisonné de la démarche préconisée par la méthode et implémentée par le logiciel.

Elle peut être activée à tout moment, ou intervenir suite à des diagnostics sur les produits réalisés (vérification de règles). La fonction d'aide au processus présente donc au développeur des remarques génériques sur ses produits et des possibilités de travail en fonction du point du développement déjà atteint.

L'aide sur les concepts.

L'ensemble des concepts est répertorié dans l'aide. Il est toujours possible d'obtenir les définitions de ceux–ci.

La trace du processus réel.

Par le gestionnaire de trace, on dote l'outil d'une mémoire des événements du développement ; l'outil dispose de la connaissance des produits réalisés, leur configuration (incomplétude, problèmes de cohérence etc.) ; la trace et les produits permettent à l'outil de pronostiquer contextuellement des tâches réalisables dans la période suivante, en

cohérence avec la démarche de la méthode O*. Elles permettent d'envisager la réutilisation.

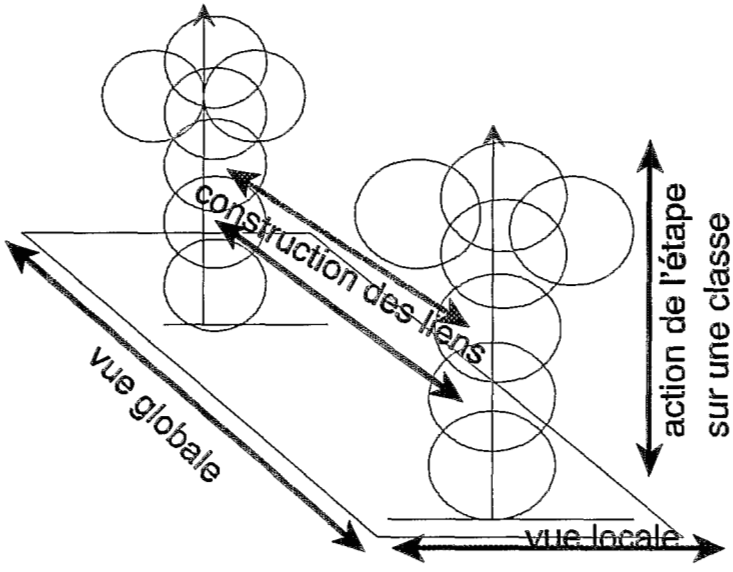

Figure 1 : le modèle fontaine sur O*

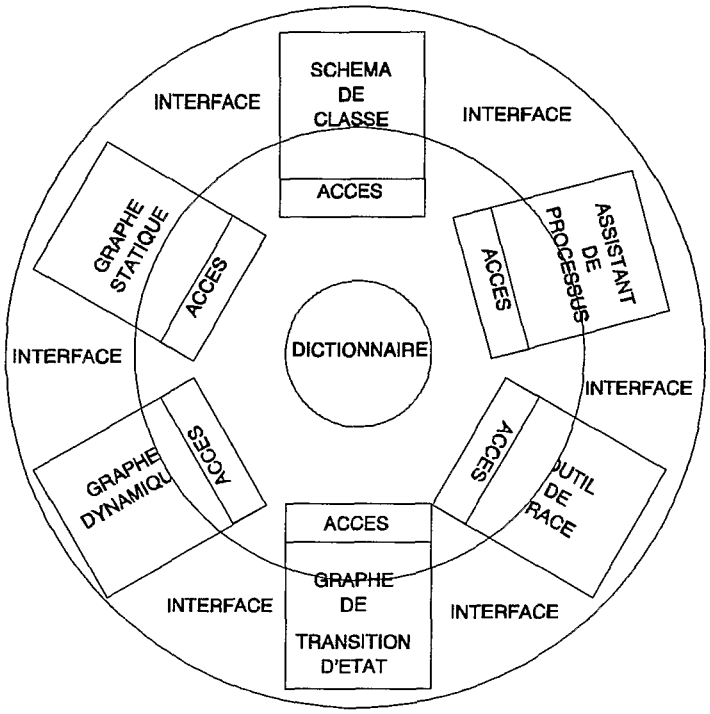

Figure 2 : l'architecture de l'outil

III. IMPLEMENTATION DES OUTILS

"L'objet logiciel s'obtient par la convergence d'un objet conceptuel, d'un objet interface et d'un objet mémoire" [VAU91].

Nous avons séparé en quatre étapes notre démarche de réalisation :

a. choix d'un environnement de développement

b. Implémentation des concepts de la méthode O* en langage C++

c. Conception de l'interface :

- Associer aux classes logicielles un ensemble de classes d'interface.
- Créer les ressources associées.
- Définir la communication entre les classes d'interfaces et les classes logicielles.

d. Conception interne du système :

- Organisation du projet en modules.
- Conception détaillée des méthodes complexes.
- Description des classes mémoires

• Assemblage des classes mémoire et logicielles.

Chaque étape de la démarche inclut des tests unitaires et d'intégration.

III.1. Choix d'environnement de développement

L'implémentation nécessite un environnement graphique moderne. Windows a été choisi à cause de sa grande distribution et de sa disponibilité. Pour l'outil de programmation, nous avons été confrontés au choix entre un méta-CASE (Graphtalk par exemple) et le langage C++.

Le choix de C++, malgré le temps de développement, s'explique par :

• l'accès à un grand nombre de bibliothèques d'objets,

• l'intégration avec des bibliothèques objets gérant d'autres types de programmation (ex. logique des prédicats),

• le contrôle des aspects matériels,

• la pérennité,

• le coût,

• l'interfaçage avec d'autres logiciels.

III.2. Quelques cas d'implémentation de concepts O*

Les modèles sémantiques de la méthode O* et du langage C++ s'appuient sur le paradigme objet. Mais la méthode O* utilise des concepts supplémentaires qui devront être simulés dans le langage C++.

a. la notion de domaine de valeur

Nous faisons les propositions générales suivantes :

- Si la sémantique du domaine de valeur correspond à un type de donnée prédéfinie du langage C++, nous adoptons ce type pour ce domaine de valeurs;

Exemple : les domaines ENTIER et REEL de O* peuvent être rendus respectivement par les types long et double du C++.

- Si elle correspond à la notion d'agrégation de domaines de valeurs élémentaires, la notion de classe du langage C++ permet de les implémenter (données + opérations).

Exemple : le domaine adresse devient la classe adresse.

- La notion d'ensemble de domaines de valeurs sera rendue par le concept de classe conteneur du langage C++.

Exemples : sacs, ensembles et listes.

b. le lien de composition

La sémantique du lien de composition est bien rendue par la notion d'objet membre en C++ : ainsi la création (destruction) d'un objet membre d'un autre objet se fait en même temps que la création (destruction) de l'objet lui même.

Si l'objet membre comporte des parties dynamiques, le langage prévoit leur création (destruction) dans le constructeur (destructeur) de l'objet.

Exemples :

Langage O*	Langage C++
classe PERSONNE **propriétés** nom : CHAINE	**class** PERSONNE { CHAINE nom ; ENTIER age : // ..... } ;

classe SUCCURSALES
propriétés
jours ouvrables : ensemble(JOUR DE LA SEMAINE
classe OUVRAGE
propriétés
auteurs : liste (CHAINE)
	class SUCCURSALES {
	Ensemble<JourDeLaSemaine>
	JoursOuvrables;
	} ;
	class OUVRAGE {
	Lliste<CHAINE> auteurs;
	//....
	} ;

N.B : Ensemble et Liste sont des classes C++ paramétrées par un type (classes template C++)

c. Le lien de référence

Le lien de référence sera traduit par la syntaxe associée à un pointeur membre d'une classe (la classe référençante). On associera à l'objet référencé une liste de pointeurs vers les objets le référençant.

L'objectif visé ici est d'assurer la propriété d'inclusion des cycles de vie : le cycle de vie d'un objet référençant est inclus dans le cycle de vie de l'objet référencé; un mécanisme de gestion d'exceptions placé dans le destructeur de l'objet référencé permet d'autoriser ou non la désallocation de la mémoire de l'objet référencé selon que la liste des objets référençants est vide ou pas.

d. Assertion : la contrainte d'attribut

Elle est traduite par un ensemble de méthodes booléennes appelées méthodes autorisations. Celles-ci sont appelées en tête des méthodes de création et de modification de la classe.

e. Evénement interne à un objet

Suite à une méthode de modification, une méthode événement interne constate ou non un changement d'état remarquable du SI [BRU93].

L'occurrence d'événement interne entraîne l'invocation par la méthode événement interne d'un certain nombre de méthodes sur d'autres objets.

f. Evénement externe

Un événement externe de l'étape d'analyse se traduit dans le logiciel par un plusieurs événements utilisateurs sur les objets d'interface. L'objet logiciel et les objets d'interface collaborent pour contrôler la validité du message de l'événement externe avant sa prise en compte effective.

g. Un exemple d'implémentation la métaclasse classe

Nous illustrons ici, à travers l'exemple de la métaclasse **classe**, notre démarche de conception. Les squelettes de classes C++ ci–dessous montrent la structure logicielle résultant de l'implémentation de l'objet de conception classe. L'objet conceptuel s'est enrichi de nouvelles classes à savoir des classes interfaces et une classe mémoire :

```
class TSchemaClasse : public TStreamable
{
char  NomClasse[LG_MAXI_NOMCLASSE];
TFenSchemaClasse FenSchemaClasse ;
TIconeClasse rIconeClasse ;
//...
public :
BOOL CycleDsGrapheHeritage(TSchemaClasse*, HWND) ;
//...} ;

class TFenSchemaClasse : public TWindow
{//..
void Trt_AjouterLienHeritage(RTMessage Msg) = [
CM_FIRST+IDM_ADD_HERIT] ;
//...} ;
class TCreeLienHeritDial : public TDialog
{} ;
```

Nous constatons notamment que les aspects liés à la persistance sont gérés par un lien d'héritage entre la classe conceptuelle TSchemaClasse et la classe flux TStreamable de Borland C++. Les aspects interfaces apparaissent à travers un lien de composition entre la classe TSchemaClasse et les classes TFenSchemaClasse et TIcone :

La classe TIconeClasse matérialise à l'écran les icônes de classe (un rectangle contenant le nom de l'icône).

La classe TFenSchemaClasse est dérivée de la classe TWindow de la bibliothèque de classes OWL et permet la spécification interactive des caractéristiques locales à une classe ;

Exemple de coopération entre les objets de ces classes :

Lorsque le développeur déroule l'option de menu Ajouter/Lien/Héritage, le méta événement externe "Ajouter un lien d'héritage" se produit et déclenche l'opération (ici la fonction membre) Trt_AjouterLienHeritage de l'objet d'interface TFenSchemaClasse associé à l'objet TSchemaClasse. Une boîte de dialogue s'ouvre alors pour la saisie du nom de la super classe (voir Figure 3).

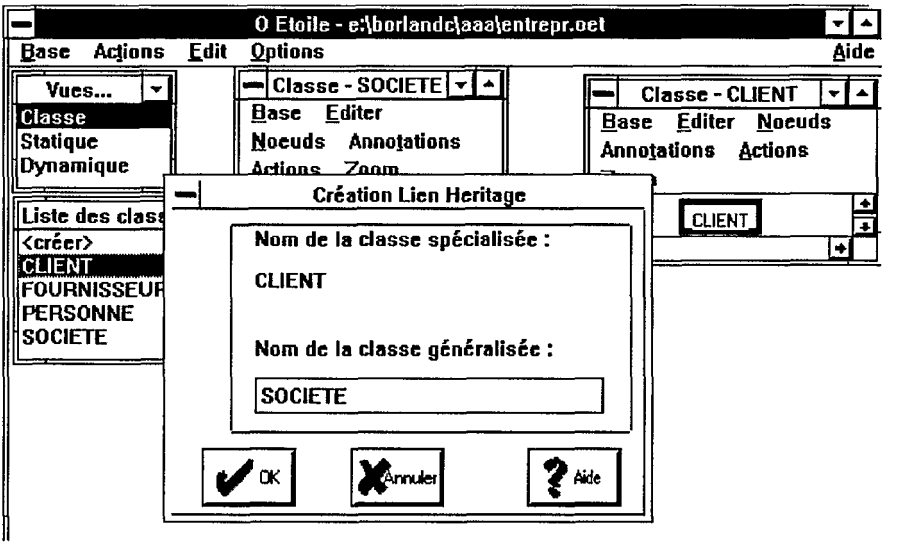

Figure 3 : Boîte de saisie d'une super classe.

Cette boîte de dialogue est créée dynamiquement par l'instanciation de la classe interface TCreeLienHeritageDial, dérivée de TDialog (classe OWL). Une fois le nom de la super classe saisie, l'objet interface TFenSchemaClasse dispose d'une demande complète de modification de l'état de l'objet TSchemaClasse ici la valuation de l'attribut "**classes généralisées**". L'objet interface active alors une méthode de contrôle (ici la fonction booléenne CycleDsGrapheHeritage) de TSchemaClasse pour s'assurer de la validité de la modification demandée et pour en conséquence autoriser ou refuser la demande de modification (Figure 4).

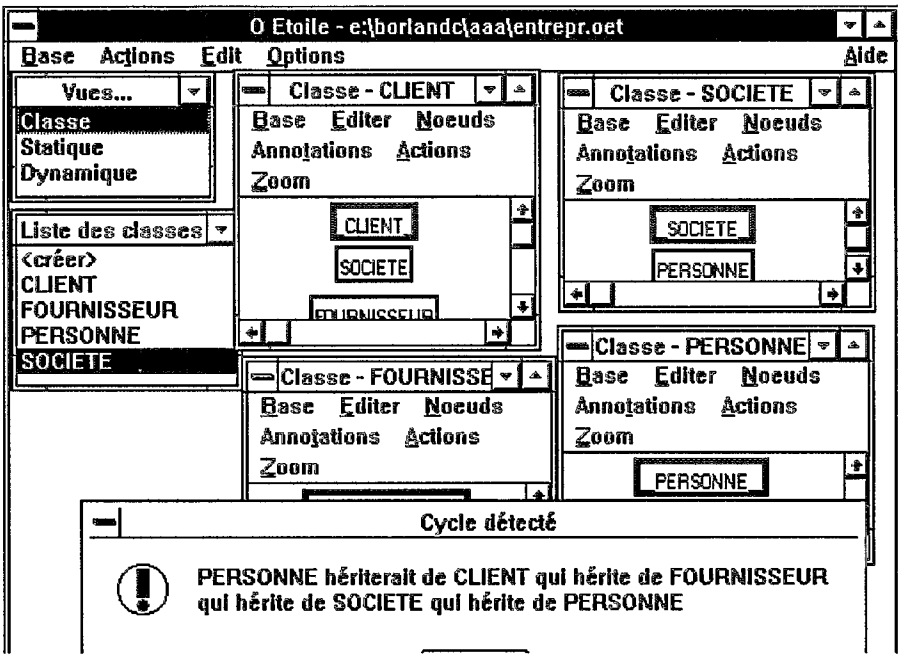

Figure 4 : Contrôle de validité interactif.

CONCLUSION

A partir d'une méthode OO basée sur l'étape d'analyse d'un SI, nous avons défini et commencé à implémenter une panoplie d'outils ayant les caractéristiques d'un AGL OO. Les outils peuvent être utilisées dans toutes les étapes du cycle de vie et notamment tout au long du processus incrémental de modélisation d'un SI.

Le choix du "tout objet", malgré ses contraintes fortes dans l'environnement que nous avons choisi (Windows/C++), est porteur. Nous poursuivons l'implémentation en insistant sur le fait que nos outils doivent pouvoir être utilisés dans toutes les étapes d'un cycle de vie OO et dans une généralisation devraient pouvoir être paramétrés par les méthodes.

Bibliographie.

[BOE87] Boehm B.W. : "A Spiral Model of Software Development and Enhancement" Software Engineering Project Management, 1987.

[BOO92] Booch G. "Conception orientée–objet et applications", Addison–Wesley, Paris, 1992.

[BRU93] Brunet J. : "Analyse conceptuelle orientée–objet", thèse de doctorat de Paris VI, informatique, 1993, pp. 173–192.

[DOW87] Dowson M. :"Iteration in the Software Process". in Proc 9th Int. Conf. Software Engineering, Monterrey, CA, Mar. 30–Apr. 2 1987.

[HEN90] Henderson–Sellers B., Edwards J. M. : "The object–oriented systems life cycles", Com. of the ACM, Sept. 90, vol 33, n° 9, pp. 146–158.

[IIV90] Iivari J. : "Hierarchical spiral model for information system and software development", Information and Software technology, Feb. 1990.

[MOR&SOU93] Moreno M., Souveyet C. : "The Evolutionary Object Model (EOM)" in IFIP Trans. A–30, (Ed.) Prakash et alii pp. 41–58.

[NAT92] : "NATURE Esprit III project n° 6353".

[ROL&ALI86] Rolland C., Foucaut O., Benci G. : "Conception des systèmes d'information : la méthode REMORA", Eyrolles, Paris, 1986.

[ROL&PRA93] Rolland C., Prakash Naveen : "Reusable process Chunks".

[ROL93] Rolland C. : "Modeling the Requirements Engineering Process", 3d European–Japanese Seminar on Information Modeling and Knowledge Bases, Budapest, Hungary, 1993.

[ROY70] Royce W.W. : "Managing the development of large software systems" in Proc 9th Int. Conf. Softw. Eng., Monterey, 1987, pp. 328–338.

[SCH93] : Schmitt J.P. : "Product modeling for requirements engineering process modeling" in IFIP Transactions A–30, op. cit., pp. 231–245.

[SUT91] Sutcliffe A.G. :"Object–Oriented Systems Development: survey of structured methods", Jl of information and Softw. Tech., vol 33, n°6, 07/91.

[VAU93] Vauquier D. :"Développement orienté objet. Principes, processus, procédés", Eyrolles, 1993, 309 p.

[VES&AL92] Vessey Iris, Jarvenpaa S.L., Tractinsky N. :"Evaluation of vendor products : CASE tools as methodology companions" Com. of the ACM, Apr. 1992, vol 35, n° 4, pp. 90–106.

[WIN79] Winograd T. : "Beyond Programming Langages", Com. of the ACM, July 1979, vol 22, pp. 391–401.

[WIR91] R.J. WIERINGA : "Object–Oriented Analysis, Structured Analysis and Jackson System Development", Int. Conference on the Object–Oriented Approach in Information Systems, Quebec, Canada, Oct. 1991.