Validation dans les outils de l'ingénierie des besoins orientée objet

Samba Baldé(1), Christian Clercin(2) , Oumar Sarr(1)
(1)ESP - Département Génie Informatique BP 5085 Dakar Sénégal
(2) Ampopoka Villa Kay BP 1449, Fianarantsoa 301 Madagascar
e-mail: sarro@ensut.ensut.sn

Résumé:

Nous nous intéressons dans cet article aux problèmes de validation dans les méthodes d'analyse de système d'information orientées objet. Partant du fait que l'ingénierie des besoins comprend les activités d'élicitation, de modélisation, de synthèse et de validation, nous proposons des outils couvrant tout le champ de l'ingénierie des besoins. Nous nous appuyons sur la méthode de spécification O* mais notre démarche peut être facilement étendue aux autres méthodes orientées-objet et même à toute méthode d'analyse de système d'information. Nous proposons notamment parmi nos outils:

- un éliciteur des besoins à deux axes, le premier concentré sur la capture des scénarios utilisateurs et le second déduisant de la spécification textuelle une esquisse de spécification formelle à valider
- un prototypeur traduisant les spécifications orientées objet vers un langage orienté objet exécutable
- un assistant du processus intégrant des heuristiques liées aux sémantiques des méthodes de développement.

Mots-clé : Méthodes analyse SI orientées-objet, AGL OO, Méthode O*, Eliciteur

Abstract :

In this paper, we are dealing with validation issues in object-oriented information system analysis methodologies. Based on grounds that requirements engineering include elicitation, modelisation, synthesis and validation activities we propose tools covering all the field of requirements engineering. We rely on the specification method O* but our approach can be easily extended to other object-oriented methodologies and even to any other information system methodology. We propose among the set of our tools :

- an elicitor of requirements with two axis : the first focusing on the user scenario capture and the second infering from the textual specifications a blueprint of formal specifications to be validated
-a prototyper translating object oriented specifications in an object-oriented executable langage
-a process assistant integrating heuristics to the semantics of sofware methodologies

Key-words: object-oriented methods for IS, OO CASE, O* Method, elicitor

INTRODUCTION

Peu d'outils permettent de prendre en compte toute l'étape d'analyse des systèmes d'information. Les Ateliers de Génie Logiciel (AGL) se concentrent sur la capture des spécifications écrites dans un langage de modélisation donné et sur une exploitation peu ou prou intelligente de ces spécifications.

Pourtant, dans l'enseignement des méthodes des Systèmes d'Information (SI), nous commençons paradoxalement par l'étude de l'existant. Nous enseignons comment il faut recueillir les informations à partir des entrevues issues du système de pilotage ou du système opérant. Après la consolidation de ces entrevues, nous avons l'habitude de recommander de faire la

synthèse des différents éléments recueillis et de valider cette synthèse auprès du système de pilotage. Les spécifications sont écrites par la suite comme une manière de traduire l'existant informationnel. Les outils proposés couvrent en général le deuxième tronçon de la partie analyse. Aucun outil ne couvre tout le champ de l'analyse. Des travaux antérieurs [Gros 91] avaient déjà décrit ce " gap " et proposé des solutions partielles. C'est seulement ces deux dernières années que le problème semble être perçu dans ses dimensions principales avec l'émergence des méthodes d'analyse et de conception orientées OO, notamment avec Objectory [Jacobson 92] et OMT [Rumbaugh 91][Rumbaugh 94], avec l'utilisation des cas d'utilisation.

D'une manière générale, on distingue dans l'ingénierie des besoins [Dubois 94] les activités suivantes:

- une activité d'élicitation qui permet de capturer précisément les besoins.
- une activité de modélisation qui permet de représenter les descriptions informelles du client en utilisant des concepts formels.
- une activité de synthèse qui permet de déduire les insuffisances et les contradictions.
- une activité de validation qui permet de valider la description du système auprès des utilisateurs.

La plupart des méthodes ne prennent pas en charge toutes ces activités et exigent de l'utilisateur la compréhension des concepts, ce qui peut être une contrainte trop forte. Nous avons déjà travaillé sur les trois dernières activités [Balde94] [Clercin95] mais l'intégration des perspectives tracées par la première activité constituait une faiblesse réelle dans la panoplie des outils que nous préconisions dans la phase d'ingénierie des besoins.

Pour prendre en compte de toutes les activités de l'ingénierie des besoins, l'analyste utilisant une méthode OO, doit disposer des outils suivants :

-un éliciteur permettant la capture des besoins de l'utilisateur ,
-un outil de production des schémas de classe permettant la saisie et la modification des classes,
-un outil de production du graphe statique, donnant une vue globale des hiérarchies d'héritage et des liens de composition et de référence,
-un outil de production de graphes dynamiques (graphes de déclenchement des opérations, graphes de transition d'états).
-un outil de trace qui est le moyen de capitaliser l'expérience de développement en cours et de revenir, le cas échéant, à des versions passées du développement,
-un assistant de processus qui guide le développeur durant les spécifications,
-un prototypeur qui réalise une application partielle à partir des spécifications.

La méthode O* [Brunet 93][Nature 92], utilisée à titre expérimental pour le développement des outils, est une méthode OO issue des travaux de REMORA [Rolland 86] et des techniques de génie logiciel appliquées à EIFFEL [Meyer90]. Nous noterons ici que la méthode permet notamment de tenir compte des aspects statiques et dynamiques dans la description des classes.

Dans l'architecture des outils proposés pour l'ingénierie des besoins, certains outils ont bien entendu un aspect horizontal par rapport à ces activités et m^me par rapport aux étapes d'analyse et de conception d'un système..

Le papier est organisé comme suit: le premier chapitre présente l'éliciteur , le second présente le prototypeur et le troisième présente l'assistant, outil horizontal, notamment utilisé dans les synthèses des besoins.

1. L'ELICITEUR

L'éliciteur est chargé de capturer dynamiquement l'existant informationnel et les objectifs des décideurs. Il est composé de deux parties qui peuvent être utilisées de façon exclusive:

- l'inquisiteur (ici un gestionnaire de scénarios d'utilisation)
- le déducteur

Celui-ci est un système expert qui a des traits largement inspirés de CARE [Grosz 91] [Rolland 93] [Rolland 94]. L'interaction avec l'analyste peut se faire soit par l'intermédiaire d'un langage naturel soit par l'intermédiaire d'interfaces graphiques. Une base des faits contient toute la base d'information en cours et une base des règles contient les connaissances requises pour transformer la description textuelle en modèle conceptuel. L'interface de CARE a été initialement réalisée avec le langage Objective C, la base des règles en Prolog. Nous réalisons ici tous nos travaux dans l'environnement C++/Windows.

Dans ce papier, nous présentons l'inquisiteur (gestionnaire de scénarios d'utilisation). Nous travaillons avec le concept de cas d'utilisation introduit dans [Jacobson 92] en tenant compte de l'amélioration et de la formalisation apportées par [Regnell 95]. Le modèle des cas d'utilisation spécifie la fonctionnalité que le système doit offrir du point de vue de l'utilisateur[Jacobson 92] Ce modèle utilise des acteurs pour représenter les rôles que l'utilisateur peut jouer, et des cas d'utilisation pour représenter ce que les utilisateurs devraient être capables de faire avec le système. Chaque cas d'utilisation est une suite complète d'événements au sein du système, vue du point de vue de l'utilisateur. Dans [Regnell 95] il est proposé un processus d'ingénierie des besoins orienté utilisation UORE (Usage Oriented Requirements Engineering), visant à améliorer l'original UCDA (Use Case Driven Analysis) de la méthode Objectory [Jacobson 92]. UORE est constitué de deux phases : la phase d'analyse et la phase de synthèse. La phase d'analyse a comme entrée une description informelle des besoins et produit le modèle des cas d'utilisation contenant la description des acteurs et des cas d'utilisation. Ce modèle à son tour est utilisé comme entrée à la phase de synthèse qui formalise les cas d'utilisation, les intègre et crée le modèle d'utilisation synthétisé

Les spécifications des cas d'utilisation sont exprimées d'une façon plus condensées en utilisant des mécanismes d'abstraction des actions utilisateurs et du système. Chaque spécification formelle de cas d'utilisation est transformée en scénario d'utilisation abstrait décrit sous forme d'une séquence d'actions utilisateur et d'actions système interconnectées par des transitions qui représentent les messages résultant de chaque action. Les contextes d'invocation et de terminaison d'un scénario d'utilisation abstrait sont indiqués.

Le modèle d'utilisation synthétisé consiste en une collection de vues d'utilisation par acteur. Une vue d'utilisation est synthétisée à partir de tous les scénarios d'utilisation abstraits produits pour un acteur spécifique. Une vue d'utilisation est créée en trouvant les parties similaires des scénarios d'utilisation abstraits et en les fusionnant. Le résultat est un graphe orienté avec trois types de noeuds : les actions de l'utilisateur, les actions du système, et les contextes d'invocation et de terminaison. Ces noeuds ont le même poids sémantique que ceux des scénarios d'utilisation abstraits.

La puissance de la représentation sémantique de O*[Brunet 93] nous permet de transformer simplement les vues d'utilisation en graphe dynamique.

Pour chaque vue d'utilisation nous spécifions deux classes O*:
-une classe dérivée de la classe O* ACTEUR représentant l'acteur de la vue d'utilisation et
-une classe SYSTEME encapsulant les objets d'analyse intervenant dans le déroulement de la vue d'utilisation

Les règles de passage de la vue d'utilisation vers le graphe dynamique brut sont les suivantes:
- à chaque action système d'une vue d'utilisation nous associons une opération O* spécifié sur la classe SYSTEME correspondante
- à chaque action utilisateur d'une vue d'utilisation nous associons une opération O* spécifié sur la classe ACTEUR correspondante

- à chaque action utilisateur, nous associons un événement externe O* correspondant à la fin de l'action utilisateur; cet événement est spécifié à l'intérieur de la classe acteur correspondante;le message O* de l'événement externe correspond au libellé du message UORE de la vue d'utilisation; les opérations déclenchées sont les opérations associées aux actions systèmes cible des messages UORE provenant de l'action utilisateur; les conditions O* de déclenchement respectives des opérations correspondent aux conditions UORE d'emprunt du chemin du message UORE dans le graphe de la vue d'utilisation
-à chaque action système, nous faisons correspondre un événement interne correspondant à la fin de l'action système;cet événement est spécifié sur la classe SYSTEME; le prédicat O* de l'événement interne est systématiquement vérifié; la partie déclenchement de l'événement interne est obtenue de façon similaire à la partie déclenchement de l'événement externe correspondant à une action utilisateur ci-dessus.

Nous présentons ci-dessous (figure 1) une illustration de cette mise en correspondance sur une vue d'utilisation d'un client d'un guichet automatique de banque (GAB) tiré de [Regnell 95].

L'éliciteur que nous construisons permet
- d'éditer les modèles d'utilisation synthétisés,
- de contrôler que chaque scénario d'utilisation abstrait est un chemin possible dans la vue d'utilisation corespondante
- de découvrir de nouveaux scénarios d'utilisation grâce à un parcours systématique des graphes des vues d'utilisation
- de traduire automatiquement une vue d'utilisation en un graphe dynamique O* brut que l'analyste pourra ensuite affiner en distribuant les actions système sur les objets de la base de spécification.

L'assistant au processus est utilisé comme vérificateur dans l'éliciteur. En effet, le dialogue peut ne pas être achevé, des contradictions peuvent se déduire de la description des vues. En outre, l'éliciteur pourra imprimer le texte de la description informelle induite.

2. LE PROTOTYPEUR

Notre outil se situe dès l'analyse dans la perspective d'un processus incrémental de prototypage conduisant au produit final. il comporte donc la production de prototypes. Ces prototypes constituent des moyens privilègiés de communication développeur-client. Ils ont pour objectifs:
- de faciliter l'obtention d'informations sur la pertinence et l'adéquation de la spécification et de la conception du système informatique à venir.
- de servir de précurseur à l'écriture effective du système final [Krief 91].

```
classe Client GAB
 événements
        carte Insérée
          message
           (numéro de compte)
          déclenche
                ValidationCarte sur SYSTEME
        finEntrerCode
          message
           (Code)
          déclenche
                ValidationCode sur SYSTEME
                si CodeEntré
                PrendreCarte sur Client GAB
                si Avorté
        finEntrerMessageOuSelectCode
          message
           ()
          déclenche
                ValidationMontant sur SYSTEME
                si MontantEntré
                PrendreCarte sur Client GAB
                si Avorté
                CollecterInfosSolde sur SYSTEME
                si solde sélectionné
        finRéentrerCode
          message
           (Code)
          déclenche
                RetenterValidation sur SYSTEME
                si CodeEntré
                PrendreCarte sur Client GAB
                si Avorté

        finRéEntrerMontant
          message
           (Montant)
          déclenche
                ValidationMontantsur SYSTEME
                si MontantEntré
                PrendreCarte sur Client GAB
                si Avorté
        finPrendreEspèces
          message
           ()
          déclenche
                PrendreCarte sur Client GAB
        finPrendreCarte
          message
           ()
          déclenche
                CollecteInfoReçu sur SYSTEME
```

```
classe SYSTEME
 événements
        FinValidationCarte
          prédicat
           (TRUE)
          déclenche
                EntrerCode sur Client GAB
                si CarteOk
                PrendreCarte sur Client GAB
                si CarteInvalide
        FinValidationCode
          prédicat
           (TRUE)
          déclenche
                EntrerMessageOuSelectCode sur Client GAB
                si CodeOk
                RéentrerCode sur Client GAB
                si CodeInvalide
        FinValidationMontant
          prédicat
           (TRUE)
          déclenche
                CollecteEspèces sur SYSTEME
                si MontantOk
                RéentrerMontant sur Client GAB
                si MontantInvalide
        FinRetenterValidation
          prédicat
           (TRUE)
          déclenche
                RejeterCarteInvalide sur SYSTEME
                si TropErreur
                RéentrerCode sur Client GAB
                si CodeInvalide
                EntrerMessageOuSelectCode sur Client GAB
                si CodeOk
        FinCollecteEspèces
          prédicat
           (TRUE)
          déclenche
                PrendreEspèces sur Client GAB
        FinCollecteInfoSolde
          prédicat
           (TRUE)
          déclenche
                PrendreCarte sur Client GAB
        FinCollecteReçu
          prédicat
           (TRUE)
          déclenche
                PrendreReçu sur Client GAB
```

Figure 1: Classes O* de description d'un cas d'utilisation classique (GAB).

Du point de vue des fonctionnalités, chaque prototype obtenu devra permettre de :
-créer, modifier et supprimer des instances de classes,
-accéder aux attributs et aux instances de liens de composition ou de référence, et les manipuler,
-saisir les événements externes et simuler les événements temporels du SI,
-suivre la propagation des événements internes dans un graphe dynamique.

Un modèle de conception de dialogue permet à l'outil de générer l'interface utilisateur. Pour la définition des écrans, on utilise les éléments de données (attributs et liens statiques) définis dans la métabase.

Le modèle fonctionnel global du prototype est le suivant (Figure 2)

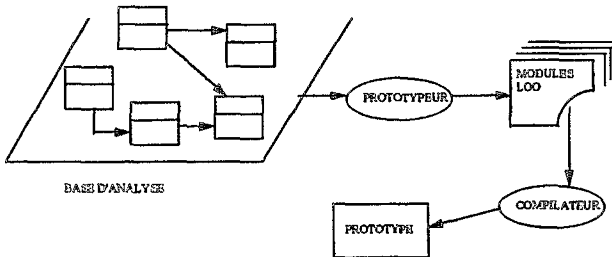

figure 2 : Schéma de principe du prototypeur

Le prototypeur génère à partir de spécifications conceptuelles O* un ensemble de modules logiciels écrits dans un langage orienté-objet. Dans cet ensemble de modules, on trouvera essentiellement :

. un module contenant les squelettes des classes logicielles,

. un module composé des classes de stockage et de manipulation des instances,

. un module de gestion de l'interface personne-machine du prototype (aspects statiques et dynamiques).

Nous avons notamment réalisé pour le prototypeur:

-l'interface d'aide à la spécification de concepts O*[ARIAL D/ 95-1],

-le passage des spécifications O* vers C++

-la génération des squelettes de classes C++[ARIAL D/ 95-2]

La persistance[ARIAL D/ 95-3] sera gérée grâce à des outils proposés dans les différents SGBD présents dans l'environnement Windows.

Nous présentons ici le passage des spécifications O* en C++

Passage des concepts O* en C++

Nous développons ci-après quelques règles d'implémentation de concepts O* en C++, bien que notre ambition soit de proposer un modèle pivot orienté-objet intermédiaire qui permettrait et faciliterait le passage du modèle O* vers différents langages orientés-objet (Eiffel, C++, Smaltalk, SGBDOO, etc..).

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 par la définition de mécanismes génériques écrits en C++.

Passage direct : héritage

La relation 'est-un" entre classes O* peut être rendue par l'héritage entre classes C++. Dans C++, en effet, la création d'un objet d'une classe dérivée s'accompagne automatiquement de celle des parties d'objets des classes généralisées correspondantes. Il est possible de restreindre un objet au point de vue de n'importe quelle classe appartenant à sa hiérarchie d'héritage. Ceci nous permet de retrouver la sémantique de l'héritage O* qui se fonde sur le constat que les objets identifiés par l'analyste sont parfois des perspectives différentes d'un même phénomène du monde réel. Par exemple les objets "personne DIOUF" et "salarié DIOUF" ont conceptuellement les mêmes identités puisqu'ils sont tous les deux une vue particulière d'un même phénomène.

O* offre la possibilité de spécifier des contraintes sur les différentes instances des classes spécialisées

La contrainte de disjonction

Une contrainte de disjonction peut être définie entre schémas spécialisés d'un même schéma généralisé. L'intersection des schémas spécialisés est vide.

Exemple : classe VEHICULE
assertions
disjonction (VOITURE,CAMION)
classe VOITURE hérite de VEHICULE
classe CAMION hérite de VEHICULE

Les schémas CAMION et VOITURE héritent du schéma d'objet VEHICULE avec une contrainte de disjonction. En C++, ce type de contrainte semble inutile puisqu'un objet instance de CAMION ne peut être instance de VOITURE. La contrainte de disjonction est traduite par un héritage classique.

La contrainte de couverture

La contrainte de couverture entre un ensemble de classes spécialisées exprime qu'à tout objet de la classe généralisée doit nécessairement correspondre au moins un objet appartenant à une des classes spécialisées.

Exemple:

classe PERSONNE
assertions
couverture(ETUDIANT, ENSEIGNANT)
classe ETUDIANT hérite de PERSONNE
classe ENSEIGNANT hérite de PERSONNE

On peut traduire cette contrainte en C++ par une classe PERSONNE abstraite (non instanciable), alors que ETUDIANT et ENSEIGNANT sont des classes concrètes persistantes. Pour tenir compte du fait que l'intersection n'est pas vide, nous pouvons être amenés à créer des classes de niveaux inférieurs en utilisant l'héritage multiple : on peut avoir des étudiants qui sont en même teps des enseignants.

La contrainte de partition

Une contrainte de partition entre un ensemble de classes spécialisées exprime à la fois une contrainte de disjonction et une contrainte de couverture. A tout objet de la classe généralisée doit nécessairement correspondre exactement un objet spécialisée appartenant à une des classes spécialisées.

Exemple : classe PERSONNE
assertions
partition(HOMME,FEMME)
classe HOMME hérite de PERSONNE classe FEMME hérite de PERSONNE

Les instances de PERSONNE sont soit des femmes, soit des hommes. Ils ne peuvents être ni l'un ni l'autre (couverture) et pas non plus les deux à la fois (disjonction). En C++ cette contrainte peut se traduire par une classe abstraite PERSONNE non-instanciable et deux classes FEMME et HOMME concrètes persistantes.

Ajout à C++:GTE, contrainte d'attribut graphe de transition d'état

Nous associons à chaque opération une méthode d'autorisation : **BOLL OpérationEstApplicable().**

Cette méthode, invoquée lors du déclenchement de l'opération, évalue l'assertion booléenne disjonctive dont les prédicats sont les classes d'états initiales de l'opération. L'opération n'est exécutée que si cette assertion est vraie.

contraintes d'attribut

Nous traduisons la contrainte d'attribut par des méthodes dites d'autorisation, appelées en tête des méthodes de création de la classe concernée.

Exemple

```
classe Salarie
{
char *NomSlarie;
int AgeSalarie;
// constructeur et destructeur
public:
Salarie(char*,int);
~Salarie();
// fonction d'autorisation
BOOL Salarie::b_ContrainteAttributSalarie();
}
BOOL Salarie::b_ContrainteAttributSalarie()
{
    if (AgeSalarie < 18) return FALSE;
    else return TRUE;
}
Salarie::Salarie(char*,int)
{
if (!b_ContrainteAttributSalarie())
~Salarie;
}
```

Le prototypeur sera capable de générer automatiquement les méthodes d'autorisation.

La gestion des événements

Evénements externes

La gestion des occurrences d'événements externes est faite grâce à une classe que nous faisons hériter d'une classe interface fenêtre (Sous Windows Borland C++ la classe TWindow - voir figure3) mettant à profit la capacité de cette dernière à réagir aux messages et événements de l'environnement de programmation. Ainsi la simulation de l'occurrence d'un événement externe consistera pour le développeur à selectiooner un item dans un menu. ce menu possède la structure que nous avons donnée aux événements externes. La notion d'acteur de l'organisation a été utilisée comme critère : l'ensemble des événements externes de l'application est partitionné en sous-ensembles, chacun à un rôle particulier joué par un acteur de l'organisation vis-à-vis du système d'information,.

Evénements temporels

Nous devons implémenter un mécanisme de reconnaissance des événements temporels.Notre environnement nous permet de générer des "timers" pour envoyer à notre application des messages à intervalles de temps prédéterminés, simulant ainsi les occurrences des événements temporels.

Evénements internes et cycle dynamique

La notion de cycle dynamique intègre (analogue à la notion de transaction atomique dans le vocabulaire des bases de données est importantes) est importante en O*. Elle correspond à la survenance d'une occurrence d'événement déclenchant un ensemble d'opérations qui font passer le SI d'un état cohérent à un autre état cohérent Pour assurer une cohérence complète d'un SI, il faut prendre en compte les relations d'inférence entre événements : celles-ci n'ont de sens que si

une occurrence d'événement inférée suit directement l'occurrence d'événement inférant correspondant.

Le gestionnaire d'événements

Le gestionnaire des événements temporels ainsi que le gestionnaire des événements externes permettent la simulation, par l'utilisateur, de ces types d'événements, dont les occurrences sont placées dans une FIFO en entrée du gestionnaire d'événements (Figure 3).

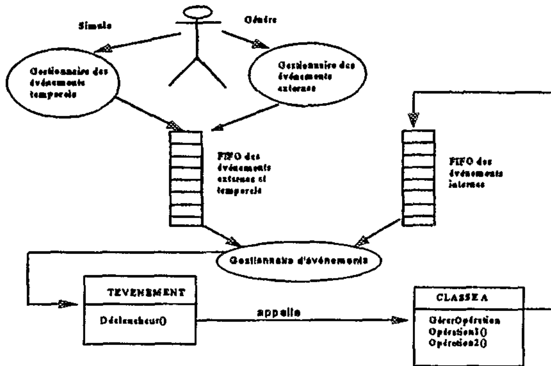

Figure 3 : le gestionnaire d'événements.

Pour chaque événement externe ou temporel, le gestionnaire d'événements appelle la fonction déclencher() de la classe TEVENEMENT. Cette fonction appelle la fonction GérerOpération() des classes concernées en spécifiant les opérations à déclencher.
La fonction GérerOpération() exécute les opérations demandées si c'est possible, puis identifie les événements internes valides de la classe et les placent dans la pile des événements internes. Le gestionnaire des événements applique ensuite le même traitement aux événements internes jusqu'à ce que la pile des événements soit vide; il reprend alors l'événement suivant de la pile des événements temporels et externes.
On remarquera que le gestionnaire des événements doit identifier les bouclages sur des événements internes lors de l'éxécution et proposer une éventuelle intervention.

3. L'ASSISTANT

Nous proposons d'intégrer dans notre prototype d'outil un **assistant de processus**, visible de l'interface et chargé des contrôles et de l'aide à la modélisation. Il comprend quatre degrés:

. **le dégré moins un** ou Assistant-Documentaliste : c'est un compagnon méthodologique [Vessey 92] rappelant les concepts, la connaissance de la démarche, l'utilisation de l'outil à l'aide de documents hypertextes; il pourra connaître une extension par des connaissances sur les domaines,

. **le degré zéro** ou Assistant-Contrôleur : il gère le proscription, la prescription, la permission et la contradiction des produits du développement,

. **le degré un** ou Assistant-Conseil en méthode : il assure l'assistance sous la forme de propositions d'actions et d'activités contextuelles en s'appuyant sur l'état des spécifications et la methode O*,

. le degré deux ou Assistant-Expert en domaines : il propose au développeur des parties de schémas existants à partir d'informations sur le sujet ou/et de l'état des spécifications (travail d'identification).

Quelque soit le degré, l'Assistant opère soit dynamiquement (en arrière-plan), soit à la demande de l'utilisateur. La phase d'ingénierie des besoins est caractérisée par un nombre élevé de tentatives et d'itérations; celles-ci impliquent le besoin d'une ergonomie de qualité pour que l'utilisateur accepte l'outil. L'Assistant doit alors être personalisable : à la demande, le développeur doit pouvoir définir des niveaux d'aide et de contrôle relâchés ou stricts.

Nous présentopns cici quelques heureustiques propores au degré un de l'outil

Heuristiques, degré un de l'outil

L'outil offre au développeur, sur le constat des composants déjà produits (données du dictionnaire), des propositions de tâches, fonction d'heuristiques liées à la sémantique de la méthode. Des heuristiques liées à la sémantique du domaine bien que volontairement laissées de côté seront intégrées au degré deux. Nous montrons quelques heuristiques relevant de la fonction d'aide dynamique et interactive contextuelle.

L'Assistant-Conseil utilise un modèle restreint du modèle général de processus [Grosz 94].

Le modèle de processus

Le modèle processus s'appuie sur les concepts suivants :

. **La situation**, motif remarquable portant sur une partie ou sur la totalité des objets du dictionnaire de l'AGL. L'ensemble des situations à identifier est défini directement par l'ensemble des décisions possibles.

Une situation est reconnue à l'aide d'une heuristique d'identification.

. **l'heuristique d'identification**, définie par une conjonction d'expressions booléennes; ces expressions sont construites à l'aide des paramètres de l'AGL. L'identification se fera donc en recherchant une unification avec les objets de la base,

. **le paramètre**, élément d'information quantifiable extractible de la base de spéciffication (paramètre interne) ou d'une information donnée par l'analyste (paramètre externe),

.**l'action**, l'une des tâches possibles définies dans le processus de la méthode ou au niveau le plus fin par les fonctionalités de l'AGL

.**la décision**, possibilité de réponse de l'analyste aux choix offerts par la méthode.

Exemple de la factorisation

Cet exemple a pour objet de décrire notre modélisation d'une heuristique d'identification d'une situation d'héritage.

- **Situation** : Redondance entre deux classes
- **Décision** : Factoriser deux classes
- **Action** : création d'une superclasse, création d'attributs de la superclasse, suppression des attributs
redondants dans la hiérarchie

-**Paramètres** :

. Est_Classe(X) : renvoie VRAI si X est une classe

. Existe_Lien_Héritage(X,Y) : renvoie VRAI s'il existe un lien d'héritage direct ou indirect entre X et Y

. Distance_statique(X,Y) : nombre de propriétés identiques

.Distance_Dynamique(X,,Y) : nombre d'opérations homonymes manipulant les mêmes propriétés.

-**Heuristiques d'identification:**

Est_Classe(X)	Distance_statique(X,Y) > a
Est_Classe(Y)	Distance_Dynamique(X,Y) > b

nonExiste_Lien_Héritage(X,Y)	Distance_statique(X,Y) + Distance_Dynamique(X,Y) > c

Capitaliser les connaissances

L'Assistant-Conseil ne doit pas être un outil statique, il doit évoluer à partir des connaissances acquises au sein des projets. Les utilisateurs doivent pouvoir le personnaliser en fonction de leur expérience. D'une part les ensembles de décisions et de situations ne sont pas connus intégralement. Aujourd'hui, ils sont disponibles sous la forme de l'expérience des équipes, mais ne sont pas formalisés. D'autre part, les heuristiques d'identification sont aussi dépendantes d'une manière de travailler : deux équipes confrontées à un problème donné ne proposeront pas les mêmes modèles d'analyse. L'outil doit en partie s'adapter au savoir-faire de ses utilisateurs.

Notre outil permet donc d'ajouter, de modifier des heuristiques et de créer des paramètres externes ou agrégés. Il offre par ce biais une capitalisation du savoir-faire des équipes.

CONCLUSION

Nous avons réalisé les outils de Validation des spécifications de l'ingénierie des besoins orientée objet. dans l'environnement Windows/C++.

Sur l'éliciteur, nous travaillons à améliorer l'intégration des cas d'utilisation en définissant un modèle de cycle d'interrogation similaire à celui défini dans [POTTS 94] qui comprend la documentation des besoins, la discussion sur les besoins et l'évolution des besoins

De même , aujoud'hui les deux facettes de l'éliciteur sont séparées alors que l'inquisiteur et le déducteur utilisent des concepts similaires (acteurs, entités, événements et les interrelations pour le déducteur). Nous travaillons à définir un moyen de communication entre les outils de telle manière qu'on puisse les utiliser de façon efficiente.

Concernant l'assistant du processus, nous travaillons à la définition d'heuristiques plus globales dans l'outil de validation et à la généralisation aux méthodes de spécification orientées-objet plus aptes, selon nous, à capturer toutes les étapes du cycle de développement.

L'environnement informatique pauvre dans lequel nous travaillons ralentit le rythme de progression de nos résultats Cependant les réalisations déjà fonctionnelles (le prototypeur, certaines parties de l'éliciteur et du vérificateur) nous font penser que nous pourrons aboutir rapidement à un prototype d'un AGL portant des aspects déterminants pour les outils d'ingénierie des besoins.

Bibliographie

[Arial D / 95-1]	Réalisation d'un compilateur O* à l'aide des outils Lex et Yacc, Rapport interne ARIAL, DAK
[Arial D / 95-2]	Conception et Réalisation d'un prototypeur pour O*, Rapport interne ARIAL, DAKAR, 1995
[Arial D / 95-3]	Gestion de la persistance dans un AGL supportant une méthode orientée objet, Rapport intern DAKAR, 1995
[Balde 94]	Balde S., Clercin Chr., De vallois T., Sarr O. : " Conception et réalisation d'outils hor complétant les méthodologies de systèmes d'information orientées objet ", in " Actes - Proce CARI94, ,1994
[Boehm 84]	Boehm B. W. : "Verifying and Validating Software Requirements and Design Specifications" IEEE Software Volume 1, Number 1, January 1984,
[Brunet 93]	Brunet J. : "Analyse conceptuelle orientée-objet", thèse de doctorat de Paris VI, informatique, 1
[Chen 92]	Chen M., Norman R.J. : "A framework for integrated CASE", IEEE Software, vol.9, N°2, Marc
[Clercin 95]	Chr. Clercin, Samba Baldé,Oumar Sarr, Thiérry De Vallois "Vérification, validation et prototyp supportant une méthode de spécification orientée objet" in "Actes du congrès INFORSID XIIIe - 2 Juin 1995 Grenoble
[Dubois 94]	Eric Dubois et al Animating Formal Requirements Specifications of Cooperative Infor Systems Proceedings of the Second International Conference on Cooperative Information Syst CoopIS-94, Toronto(Canada), May 17- 20, 1994
[Fagan 86]	Fagan M.E. : "Advances in Software inspections" IEEE Trans. Softw. Eng. SE-12, 7 (July 1986
[Grosz 91]	Grosz G., Rolland C. : "Computer Aided Requirements Engineering" European-Japonese Seminar on Information Modeling and knowledge Bases,Tokyo, May 1991.
[Grosz 94]	Grosz G. : "A general framework for discribing the requirements engineering process", Int. Co Systems, Man and Cybernetics, San Antonio, Texas, USA, Octobre 1994.
[Henderson 90]	Henderson-Sellers B., Edwards J.M. : "The object-oriented systems life cycles", Com. of the AC Sept 90, Vol 33, n°9, pp. 146-158.
[Jacobson 92]	Jacobson I., et al. Object-Oriented Sofware Engineering, A Use Case Driven Approach, Add Wesley, 1992
[Krief 91]	Krief Ph. : "Utilisation des langages à objets pour le prototypage", Masson 1991, Paris.
[Meyer 90]	Meyer B. : "Conception et programmation par objets. Pour du logiciel de qualité", InterEditions,
[Moreno 93]	Moreno M., Souveyet C. : "The Evolutionary Object Model (EOM)" in IFIP Trans. A-30, (Ed.) Prakash et alii pp. 41-58.
[Nature 92]	Jarke M., Sutcliffe A., Vassiliou Y., Rolland C., Bubenko J. : "Deliverable NATURE.D-1, NATURE Esprit III project n° 6353", Dec 1992.
[Potts 94]	Colin POTTS et al. : "Inquiry-Based Requirements Analysis", IEEE Software, 1994
[Regnell 95]	[Regnell 95]Björn Regnell et al Improving the Use case Driven Approach to Requirements Engi
[Rolland 86]	Rolland C., Foucaut O., Benci G. : " Conception des systémes d'information : la méthode REMORA ", Eyrolles, Paris, 1986.
[Rolland 93]	Rolland C. : "Modeling the requirements Engineering process", 3d European-Japanese Seminar on Information Modeling and Knowledge Bases, Budapest, Hungary, 1993.
[Rolland 94]	Rolland C., Prakash Naveen : "Guiding the requirements engineering process" Asia-Pacific Software Engineering Conference, 1994.
[Rumbaugh 91]	Rumbaugh et al Object Oriented Modeling and Design Prentice Hall, Englewoods Cliffs, W.J.
[Rumbaugh 94]	J. Rumbaugh "Getting Started. Using use cases to capture requirements" Journal of Object Ori Programming", Sept 94
[Vessey 92]	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.