Titre: Title:	Réalisation et évaluation d'un développement orienté objet pour la conception d'évacuateurs de crues
Auteur: Author:	Marc Barbet
Date:	1997
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Barbet, M. (1997). Réalisation et évaluation d'un développement orienté objet pour la conception d'évacuateurs de crues [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6697/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/6697/
Directeurs de recherche: Advisors:	Claude Marche, & Benoît Robert
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

RÉALISATION ET ÉVALUATION D'UN DÉVELOPPEMENT ORIENTÉ OBJET POUR LA CONCEPTION D'ÉVACUATEURS DE CRUES

MARC BARBET
DÉPARTEMENT DES GÉNIES CIVIL, GÉOLOGIQUE ET DES MINES
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.Sc.A)
(GÉNIE CIVIL)
JUILLET 1997

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

RÉALISATION ET ÉVALUATION D'UN DÉVELOPPEMENT ORIENTÉ OBJET POUR LA CONCEPTION D'ÉVACUATEURS DE CRUES

Présenté par: BARBET Marc

en vue de l'obtention du dîplome de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. KAHAWITA René, Ph.D., président

M. MARCHE Claude, D.Sc., membre et directeur de recherche

M. ROBERT Benoît, Ph.D., membre et codirecteur de recherche

M. HAUSLER Robert, Ph.D., membre

REMERCIEMENTS

L'écriture de ce mémoire n'aurait pas été possible sans le soutien financier et les conseils prodigués par M. Benoît Robert. Ce sujet a fait vibrer en moi une fibre informatique et m'a permis de réaliser un projet concret.

Je tiens également à remercier M. Claude Marche et M. Jean Joannette qui m'ont aidé à éclaircir certains problèmes au niveau de l'hydraulique du développement.

Les encouragements perpétuels de mes amis et de ma famille ont facilité la recherche et la rédaction de ce rapport. Merci à vous, parents, pour votre soutien et votre confiance. Merci à toi Isabelle, pour ton amour et ta compréhension.

Une pensée spéciale à tous ceux qui ont su rester patients...

SOMMAIRE

La programmation structurée, ou objet, est utilisée depuis de nombreuses années par les grandes compagnies informatiques. Des gammes complètes de logiciels objets sont aujourd'hui disponibles sur le marché et sont développées afin de répondre à certains besoins. La rapidité de production et les possibilités de réutilisation des parties de programmes existants sont des contraintes qui obligent les concepteurs à s'orienter vers des outils informatiques puissants.

Pour exploiter cet engouement informatique et en faire une critique, un développement orienté objet a été réalisé. Il permet la conception informatique d'évacuateurs de crues. Ce projet a été réalisé en parallèle au développement HYDRAM. Il a permis de regrouper les connaissances d'Hydro-Québec reliées aux domaines des évacuateurs de crues. Un système expert basé sur des règles de connaissances a été développé par un groupe d'experts et de cogniticiens. Un logiciel, informatisé avec le langage Prolog, a été réalisé afin de permettre l'utilisation de ce système expert. Notre étude a débuté à partir de cette base de connaissances et de la structure de raisonnement utilisée.

Les objectifs initiaux étaient d'effectuer un développement informatique orienté objet à partir des règles de connaissances proposées par le système Hydram. L'écriture d'une modélisation complète s'est avérée indispensable afin d'utiliser tous les concepts offerts par l'orienté objet. Après l'étude des méthodes disponibles, une modélisation de la conception des évacuateurs de crues a été réalisée. Elle a permis de séparer le projet en plusieurs modules. Chacun d'eux est formé par des objets qui représentent divers éléments physiques et structuraux d'un évacuateur. Ces objets contiennent des variables qui les identifient et des méthodes qui permettent des opérations. Le langage orienté objet utilisé, Smalltalk, a permis d'écrire le programme désiré. Les objectifs initiaux étaient d'assurer un entretien facile, une évolution et une réutilisation possibles du logiciel tout

en utilisant les principes de base de l'orienté objet. Une utilisation soignée des principes orientés objet nous a permis de développer un logiciel qui a été validé.

Ce logiciel permet la lecture et l'enregistrement d'un axe d'évacuation, fournit les dimensions d'un évacuateur de crues et d'un canal d'amenée. Des bibliothèques ont été développées afin de permettre le fonctionnement et une certaine réutilisation du développement. Elles contiennent des valeurs par défaut, les formules utiles, des méthodes de gestion de données et d'interfaces. Une étude comparative entre le logiciel produit et le développement HYDRAM initial a permis de soulever certaines différences, comme les possibilités de réutilisation et les modélisations faites. La grande différence se situe dans la structure des logiciels. Un langage orienté objet ne permet pas de commencer la programmation sans avoir prévu une structure rigide et la reconnaissance de tous les objets du développement.

Dans le choix d'un style informatique, la notion de temps est très importante. Selon le type de développement, un concepteur débutant aura beaucoup de difficulté avant d'exploiter pleinement les possibilités de l'orienté objet. Par contre, le temps perdu lors de la première utilisation est compensé par une réutilisation possible pour de nouveaux développement dans le même domaine. Il faut donc considérer la notion d'investissement dans une analyse de temps. Un premier développement orienté objet sera plus long qu'un autre, car le programmeur et l'analyste doivent offrir une future réutilisation de certaines parties du produit. Ce concept de réutilisation est très prometteur, mais il est difficile d'en obtenir les bénéfices instantanément. Sa maîtrise reste une affaire d'expert.

Il est nécessaire d'investir dans la création de bibliothèques d'objets afin d'optimiser l'utilisation de l'orienté objet. Les méthodes de modélisation doivent évoluer afin de répondre aux nombreux développements auxquels les méthodes conventionnelles ne s'appliquent pas. Le virage orienté objet est amorcé par le marché informatique. Pour que

le simple programmeur puisse y participer, des progrès restent à faire au point de vue de la compréhension des méthodes de modélisation et de l'utilisation des langages de programmation. Si une suite de développements informatiques dans un même domaine devait être réalisée, l'orienté objet se révélerait un atout indispensable afin de maximiser la qualité et l'uniformité des logiciels produits.

Mots clés: orienté objet, modélisation, étude comparative, évaluation et évolution, Smalltalk, évacuateur de crues.

ABSTRACT

Structured programming, also know as object-oriented programming, has been used for a number of years by principal software firms. A complete range of software products is now available on the market and have been developed to meet particular needs. Computer software designers have to move towards powerful computer tools to ensure mass production and to increase the reusability of previously-written code on new projects.

To expand and review this trend in system design, an object-oriented program has been developed that is devoted to the conceptual design of spillways. The development was realized in parallel with the development of HYDRAM, a program that permits consolidation of all the knowledge and acquired experience of Hydro-Québec related to spillways. A knowledge-based system was presented and realized by cognitive scientists. A software package was developed using the Prolog language for this expert system. Our study is based upon this knowledge and the reasoning behind it.

The initial objectives were to develop the object-oriented software from the knowledge rules proposed in HYDRAM. The development of a complete model was found to be invaluable in order to use the concepts of object-oriented programming. After a study of current object-oriented methods, a model for this design of spillways was created. The modeling technique allowed the project to be separated into discrete modules. Each module is formed by the object to represent different physical and structural elements for a spillway. These objects have certain attributes associated with their permitting identification and operating rules. The language used was Smalltalk and this permitted the development of the desired program. The initial objectives were to ensure easy maintenance while being able to provide continuous development using object-oriented

techniques. A computer program was developed based upon the findings of object-oriented principles and subsequently validated.

This software allows reading of the data that describes the main axis of evacuation of a spillway and supplies its dimensions. A library is created to permit storage of the functions and a certain updating of information so that reuse of data is possible. It contains the necessary attributes to correct itself, useful formulations and methods of managing the inputs and interfaces. An initial study comparing this product with the development of HYDRAM indicates certain differences, such as the possibility of reuse of the designs. The principal difference remains in the structure of the software. An object-oriented language does not permit the start up of a program sequence without defining a rigid structure and a comprehensive specification of all the objects in the development.

With respect to choice of style in a computer program, the notion of time is an important factor. In accordance with the type of development envisaged, a beginner would encounter lots of difficulties before starting to fully exploit all the possibilities of object-oriented programming. However, this initial startup loss is compensated for by the possible reusability for a new development in the same field of expertise. The investment factor must be considered while analyzing time. A first object-oriented development would take much longer than another, because the computer programmer and analyst have to supply future reusability of certain parts of the product. This concept of reusability is very promising but it is very difficult in obtaining fast results. The control remains in the hands of the experts.

It is necessary to invest in a library of objects to optimize the use of the object-oriented. The methods of designs should allow expansion so that existing programs in which conventional methods do not apply could be transferred. The turning point of the object-

oriented has begun on the computer market. If a simple programmer wants to join this trend, there is still progress to be made in the comprehension of modeling technique and in the use of languages in programming. If a follow up in development in computers in the same idea should be an asset in order to maximize the quality and regularity of the software produced.

Key words: objet-oriented, modeling technique, comparative study, evaluation and evolution, Smalltalk, spillways.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES ABREVIATIONS

Abréviations	Description
ACE :	Aide au choix des évacuateurs de crues
Bc :	Largeur des contreforts
Bca :	Largeur du canal d'amenée
Bcde :	Largeur du canal d'évacuation
Bpa :	Largeur d'une passe
Cd :	Coefficient de débit
Ce :	Coefficient de débit effectif
CIGB :	Commission internationale des grands barrages
CMP :	Crue Maximale Probable
CradierAmont :	Élévation de la base du radier amont
DSD :	Design of Small Dams (1987)
DH :	Dénivellation hydraulique
f :	Facteur de rayon
fct1 :	Facteur de correction 1
fct2 :	Facteur de correction 2
fra :	Facteur radier aval
fsub :	Facteur de submergence
GB3 :	Grande Baleine 3
H :	Horizontal
ha :	Énergie cinétique
H_D :	Hauteur de conception de la lame d'eau
hd :	Écart entre la ligne de charge et la ligne d'eau au pied du coursier
He :	Hauteur de charge au dessus du coursier
H_{eau} :	Hauteur d'eau

K :	Coefficient pour le calcul du profil du seuil
Kc :	Coefficient de contraction due aux contrefort
Kpl :	Coefficient de perte due à la présence des piliers
Lb :	Largeur brute de l'écoulement
Le :	Largeur effective de l'écoulement
LG2 :	La Grande 2
N :	Coefficient pour le calcul du profil du seuil
Nc :	Nombre de contreforts
Nettoyage :	Profondeur de nettoyage du roc
NiMaxAmont :	Niveau d'eau maximal du réservoir en amont de l'évacuateur
Npa :	Nombre de passes
Npl :	Nombre de piliers
OMT :	Object Modeling Technic
P :	Hauteur du parement amont
Pente :	Pente du coursier
Q :	Débit
Q conception :	Débit de conception
Q vérification :	Débit de vérification
R :	Rayon du pied du coursier
Roc :	Niveau du roc
SM3 :	Sainte Marguerite 3
TN :	Terrain naturel
V :	Vertical
Vc :	Vitesse à la base du coursier
Vca :	Vitesse dans le canal d'amenée
X :	Coordonnée horizontale du profil du seuil
xa :	Coordonée horizontale du pied du coursier
xm :	Coordonée horizontale du point d'intersection

xtg :	Coordonée horizontale du point de tangence
Y :	Coordonnée verticale du profil du seuil
ya :	Coordonée verticale du pied du coursier
Ya :	Hauteur de l'eau au pied du coursier
ym :	Coordonée verticale du point d'intersection
ytg :	Coordonée verticale du point de tangence
Zroc :	Élévation du niveau du roc

INTRODUCTION

De tous temps, la conception d'un barrage a requis la présence d'un évacuateur de crues. Cet ouvrage hydraulique permet la protection du barrage et des digues situées autour du réservoir, en déversant en aval l'excédent d'eau accumulée. Il est de plus utilisé lors de la première mise en eau pour permettre le maintien de plusieurs paliers de niveau au cours du remplissage du réservoir. Avant la restitution de l'eau en aval, son énergie doit être réduite afin de respecter des obligations environnementales, d'érosion, etc. Des dissipateurs d'énergie accompagnent ainsi la plupart des évacuateurs de crues.

Selon les données statistiques de la Commission Internationale des Grands Barrages (Sokolov, 1979), une des causes principales de la plupart des ruptures de barrages et des accidents sérieux est le déversement des débits de crue catastrophiques par la crête des barrages en enrochement. Cette cause est imputable à divers facteurs, dont de multiples erreurs de conception et la justification insuffisante des solutions adoptées. Plus de 40 % des accidents survenus au cours de la dernière décennie découlent d'évacuateurs non adaptés aux crues réelles. Beaucoup d'ouvrages, aujourd'hui en exploitation, ont été conçus il y a plus de cinquante ans. Bien que les méthodes de conception n'aient pas été révolutionnées depuis, le calcul des débits de conception a évolué. La CMP (Crue Maximale Probable) est devenue un standard pour le calcul du débit maximal. Pourtant, l'évaluation des crues maximales a été, pendant longtemps, d'une précision douteuse. Ce n'est qu'après le développement des lois probabilistes que la prévision des crues s'est améliorée. La sécurité est ainsi augmentée pour éviter de sous-dimensionner les évacuateurs de crues.

C'est avec la volonté de standardiser la conception des évacuateurs de crues et de conserver les connaissances et l'expertise déjà acquises que l'étude de faisabilité

HYDRAM a vu le jour. Ce projet a permis le développement d'un cheminement uniforme de conception et l'écriture de toutes les règles de connaissance qui y sont reliées.

L'objectif de notre projet est de concevoir un logiciel informatique qui reproduise le cheminement de conception d'un évacuateur de crues. Ce logiciel doit profiter de la montée sur le marché des langages orientés objet. Ces langages d'une « nouvelle génération » semblent offrir de nombreux avantages au niveau de la modélisation d'un logiciel. La modélisation est la représentation logique et structurée d'un développement. Il est donc intéressant de construire un logiciel en profitant des concepts orientés objet en pleine évolution, et de faire une évaluation critique de cet engouement informatique.

Ce projet de conception informatique d'évacuateurs de crues requiert le développement de trois parties. La première concerne la modélisation des cheminements hydrauliques et structuraux afin de mener à bien la réalisation du logiciel. La modélisation est une étape du développement d'un logiciel qu'il est important de ne pas négliger. Elle permet de poser les bases de structuration et guide l'étape de programmation. Les concepts orientés objets se doivent d'être exploités afin d'en évaluer les avantages et les faiblesses. Une étude sur les moyens de modélisation doit donc être faite. La deuxième partie est la construction même du logiciel avec l'ensemble de la programmation et une validation du produit obtenu. Cette étape sera illustrée d'exemples afin de saisir l'essence de l'orienté objet. La dernière partie présente une étude comparative entre le logiciel orienté objet produit et le développement informatique initial du projet HYDRAM. Un langage de programmation d'intelligence artificielle nommé Turbo Prolog (Bratko, 1990) a été utilisé. Les résultats de ce produit ont été comparés avec les résultats des experts du groupe de développement du projet au service hydraulique de Hydro-Québec. La comparaison des deux logiciels produits permettra d'évaluer le concept orienté objet et de porter un jugement sur son utilité et son évolution.

CHAPITRE I
CONTEXTE ET OBJECTIFS DU PROJET

1.1 Engouement orienté objet

Depuis quelques années, le monde de la programmation subit une profonde évolution avec la montée sur le marché de nouveaux instruments de programmation. L'utilité de l'informatique était encore mise en doute il y a quelques dizaines d'années. Aujourd'hui, les besoins au niveau informatique sont très importants. Il est toujours nécessaire d'aller plus vite et plus loin. L'expansion de tous les marchés requiert un soutien logistique immense. Ce soutien doit souvent être géré de façon minutieuse par des outils informatiques. Le marché semble être à la recherche de langages informatiques pouvant faire face à cette nouvelle demande. Les limites des langages de développement conventionnels, tel que Pascal ou C, sont-elles atteintes? L'émergence de nouveaux langages et le succès de leur commercialisation, tel Borland C++, tendent à confirmer ce fait. Les phases de modélisation et de programmation d'un logiciel doivent être plus rapides, plus simples, et le résultat produit doit être plus performant que ses prédécesseurs.

Cette recherche d'outils supérieurs a engendré un véritable engouement pour les langages de programmation dits orientés objet. En quelques années, ils ont évolué d'un stade marginal et de curiosité à une phase de grand intérêt. Les gestionnaires s'interrogent sur les progrès réels de ces langages. Le pas pour se lancer dans une telle technologie est immense. La décision doit donc être prise en connaissance des avantages, des faiblesses et des conditions optimales d'utilisation des langages orientés objet. Un choix non éclairé peut s'avérer source de nombreuses pertes de temps. Il est donc important de s'appuyer sur des exemples pratiques de développements afin de guider un utilisateur vers une orientation informatique précise.

1.2 Domaine Hydraulique

De grands réservoirs d'eau contrôlés par l'homme sont présents dans de nombreux pays. Leurs utilités sont multiples: le contrôle du débit d'une rivière, l'irrigation, le stockage, la production hydroélectrique... La gestion de tels réservoirs est délicate et doit tenir compte de plusieurs paramètres. L'un d'eux est la quantité d'eau provenant des précipitations naturelles. Elle est prévisible mais parfois trop forte, ou incompatible avec les besoins en eau en aval du réservoir, ou avec les besoins en énergie de l'aménagement hydraulique. Afin de pallier une montée trop importante des eaux dans le réservoir, des installations sont prévues. Le contrôle du niveau de l'eau se fait à partir d'un évacuateur de crues.

Afin d'assurer une conception fiable et rapide d'un évacuateur de crues, un projet de développement a été instauré, en 1990, sous la tutelle d'Hydro-Québec. Ce projet avait pour but de regrouper et d'uniformiser toutes les connaissances acquises dans le domaine de conception d'évacuateurs de crues et de réduire le temps de traitement de ces connaissances. Pour réaliser ce développement, les cogniticiens et les experts du groupe de développement ont approfondi les méthodes de conception et les critères en usage au service Hydraulique. L'informatisation de ce développement a suivi pour rendre utilisable le cheminement de conception produit, facilement et rapidement.

L'informatisation d'un tel système permet l'optimisation de certains critères de conception. Des calculs rapides et fiables peuvent être exécutés et les résultats analysés immédiatement. Différentes combinaisons de critères peuvent être essayées afin de vérifier leurs effets sur l'écoulement de l'eau lors de son passage par un évacuateur de crues.

1.3 Étude des modèles existants

Cette partie est destinée à faire le point sur les études faites dans le domaine de l'automatisation de la conception des évacuateurs de crues. Il y a un manque de modèles mathématiques complets dans le domaine qui nous intéresse. Plusieurs modèles informatisés existent pour le calcul des réservoirs et/ou du laminage. À titre d'exemples, voici trois modèles largement éprouvés :

- Le modèle HEC5 (Hydrologic Engineering Center), développé par le *US Army Corps of Engineers*, et dont l'efficacité porte sur le contrôle des crues contre les inondations et sur l'exploitation des réserves d'eau ;
- Le modèle SSAR (Streamflow Synthesis and Reservoir Regulation) dont l'efficacité se résume à la simulation du processus hydrologique du cycle de l'eau et au calcul du laminage à travers les rivières et réservoirs ;
- Le modèle HEC1 qui est un modèle pluie - ruissellement et qui permet, entre autres, de calculer l'hydrogramme laminé à travers un déversoir.

Comme beaucoup d'autres, ces trois modèles développés dans les années 80 ne traitent pas directement de la conception des évacuateurs de crues.

En 1991, les partenaires du groupe CASTOR ont acquis un logiciel d'aide à la conception des évacuateurs de crues. Ce logiciel, identifié sous le terme HEV, a été conçu par Mohamed Taleb (Taleb, 1991) dans le cadre d'une thèse de doctorat à l'École Polytechnique de Montréal. De plus, un système-expert, identifié sous le terme de ACE (aide au choix des évacuateurs de crues) accompagne et complète ce logiciel.

Une description plus précise de ces deux études suit.

1.3.1 Le logiciel HEV

HEV est un logiciel d'aide à la conception des évacuateurs de crues. Il couvre la totalité du cheminement à suivre pour la conception d'un évacuateur. Il se compose de trois modules principaux. Le premier module permet de déterminer la capacité et la cote normale du réservoir. Le deuxième module effectue le calcul du laminage des crues à travers la retenue, il détermine la capacité d'évacuation et la loi de déversement pour un hydrogramme de crue et un type d'évacuateur donnés. Le dernier module est composé de deux fonctions principales. L'une est conçue pour le calcul du profil de la crête, du coursier et du dissipateur d'énergie. L'autre a pour but l'étude du comportement hydraulique de l'évacuateur.

C'est un logiciel interactif qui réserve aux communications graphiques une place prépondérante. Son interface usager est relativement claire bien que n'étant utilisable que par des professionnels formés.

Cette étude est très détaillée et touche la plupart des aspects de la conception d'un évacuateur de crues. Le seul module qui peut être utilisé dans le cadre de notre projet est le module "Évacuateur". Il permet le calcul géométrique et hydraulique des évacuateurs de crues les plus utilisés dans la pratique (évacuateur Creager, Parabolique et déversoir Tulipe). Les caractéristiques géométriques des évacuateurs sont largement documentées. Le logiciel traite le cas des évacuateurs équipés de vannes.

Ce logiciel ne fait pas de choix sur le type d'évacuateur à utiliser. L'utilisateur doit lui-même indiquer l'évacuateur requis pour une étude particulière. Une fois ce choix fait, le logiciel débute la conception de l'évacuateur de crues et fournit les résultats à l'usager.

Pour aider l'utilisateur dans le choix de l'évacuateur, Benoît Robert, de l'École Polytechnique de Montréal, a développé un système expert appelé ACE (Robert, 1992).

1.3.2 Le système expert ACE

ACE (Aide au Choix des Évacuateurs) est un système expert qui permet d'aider l'ingénieur dans son travail de conception. Il détermine le type d'évacuateur de crues le plus intéressant à associer à un ouvrage de retenue. Cette étape très importante ne nécessite aucun calcul spécifique mais oblige plutôt à combiner des notions techniques, spécifiques à chaque étude. Il n'y a pas de règle précise de combinaison de ces notions. Des expertises sont acquises en travaillant dans le domaine, en observant les conceptions existantes, tout en respectant une base théorique. Ces considérations ont été regroupées dans le système expert ACE, à l'aide de critères de conception qui sont traités selon des facteurs de pondération.

Ce système utilise un certains nombre de critères (le rôle de l'évacuateur, le type de risque, la topographie du site, l'hydrologie, ...) et y accorde une certaine pondération. La valeur de cette pondération est fixée selon les connaissances des experts et traduit l'importance relative de chacun de ces critères. Ce facteur de pondération dépend de chaque cas de conception. Il est donc modifiable par l'usager mais les experts ont déterminé une marge limitant ces modifications.

Le système expert ACE ne fournit pas une réponse unique à l'utilisateur. Il présente à l'ingénieur les types d'évacuateurs les plus utilisés dans les conditions de l'étude en cours. Chaque conception est accompagnée d'un facteur de pondération. Si ce facteur n'est pas concluant, le système présente la pondération pour chacun des critères de conception afin de permettre une comparaison entre les diverses possibilités.

La validation de ce système semble avoir été concluante. Ce système n'a pas été complètement automatisé.

1.3.3 Le développement HYDRAM

HYDRAM réalise la conception d'évacuateurs de crues dans un stade préliminaire. Le cheminement fait la distinction entre divers modules principaux, à savoir l'enregistrement de l'axe d'évacuation, la conception de l'évacuateur et celle du dissipateur d'énergie. Chacun de ces modules possède son propre cheminement pour aboutir à sa conception s'il est réalisable. Selon des critères établis avec des règles de connaissance, des choix sont faits sur le type d'évacuateur ou de dissipateur à retenir. Les résultats correspondent à la présentation des dimensions possibles des ouvrages, le profil des écoulements sur les évacuateurs et les dissipateurs, et le cheminement suivi pour aboutir à ces valeurs.

Les évacuateurs de type parabolique et de type creager sont développés sous des dénivellations hydrauliques variables. Les systèmes de dissipation possibles sont le bassin et la fosse de dissipation, le canal d'évacuation et en rivière. Les calculs d'un canal d'évacuation, des courbes de remous et des volumes excavés complètent ce développement. Le logiciel produit est implémenté à l'aide du langage de programmation Turbo-Prolog (Bratko, 1990).

1.3.4 Analyse et comparaison

Le logiciel HEV nécessite l'entrée du type d'évacuateur retenu. Le système expert ACE fournit au logiciel cette information. Le choix du type d'évacuateur se fait à l'aide de pondérations successives sur des critères retenus par des experts dans le domaine. La compilation des résultats de ces critères indique à l'usager les choix possibles, par ordre de faisabilité. Les systèmes n'ont aucun lien informatique entre eux.

Le système qui vient d'être conçu est différent des précédents. Avec des principes hydrauliques précis fournis par des experts, il détermine le type d'évacuateur requis selon le site. Une fois le type d'évacuateur déterminé, il permet la conception complète de l'évacuateur et du dissipateur dans le cadre d'une étude de faisabilité. La conception est moins précise que celle fournie par HEV mais l'objectif de la recherche est différent.

Le projet en cours de modélisation répond à un besoin de la compagnie Hydro-Québec. Un choix d'évacuateur rapide et une conception fiable sont importants pour tous les types d'études, de petites ou de grandes envergures.

1.4 Objectifs visés

Cette étude poursuit plusieurs objectifs dans deux domaines qui semblent bien distincts, l'informatique et l'hydraulique. Ces deux domaines sont liés depuis de nombreuses années avec l'émergence des méthodes de résolution numérique qui requièrent un grand soutien informatique. Le lien qui est ici mis en évidence est la structuration des connaissances dans un domaine scientifique comme le Génie Civil. L'objectif principal est d'offrir une aide dans le choix d'un outil informatique à des fins de développement scientifique. Tous développements engendrent la manipulation d'un nombre important de connaissances et de données. Afin de simplifier ces manipulations, une structuration claire est indispensable. La structuration et le traitement de ces connaissances se font en fonction de divers critères, comme les besoins, les attentes, les délais et les ressources disponibles. Les systèmes experts ont connu leur émergence à partir de cette volonté de créer un cheminement simple et fiable de prise de décisions.

Dans le but d'éclairer nos commentaires quant aux choix informatiques à prendre, un développement orienté Objet dans le domaine de l'hydraulique a été réalisé. Des objectifs ont été fixés afin de mener à bien notre étude.

➡ **Concepts orientés objet**

L'apprentissage et l'utilisation d'un langage informatique orienté objet sont des étapes incontournables lorsqu'un jugement doit être porté sur l'utilité et la pertinence d'un tel langage. Cette logique informatique et le langage employé doivent être étudiés en profondeur afin que l'ensemble de leurs ressources soient utilisées. L'apprentissage fournit les bases de la structuration des connaissances et de la réflexion quant à la possibilité de prendre le virage orienté objet pour un développement similaire à celui réalisé.

➡ **Structuration des connaissances et modélisation hydraulique**

La structuration des connaissances est la clé du bon fonctionnement d'un modèle. Cette étape de développement nécessite beaucoup de rigueur et le suivi de règles très précises afin de la mener à bien. Ce n'est qu'une fois la modélisation du développement terminé que l'étape de programmation débute. La modélisation doit suivre certains modèles afin d'exploiter au maximum les possibilités de l'orienté objet. Ses caractéristiques peuvent varier selon la méthode utilisée. Le langage informatique employé est important afin de bien suivre la modélisation effectuée. Les règles de structuration d'un modèle orienté objet diffèrent d'un modèle conventionnel. La structuration des connaissances sera faite en fonction d'un modèle orienté objet.

➡ **Production d'un logiciel validé**

Un logiciel de conception informatique d'évacuateurs de crues doit être réalisé à l'aide d'un langage orienté objet. Pour effectuer une comparaison efficace avec HYDRAM, le logiciel produit doit utiliser la plupart des ressources disponibles dans le langage choisi. Il sera validé avec des projets récents afin de vérifier sa fiabilité. Ses fonctions doivent être similaires avec la partie *'Évacuateur'* du logiciel HYDRAM.

➡ **Étude comparative entre deux développements parallèles**

Pour évaluer un développement, il est pertinent de le comparer avec une étude similaire réalisée avec un soutien informatique différent. Une comparaison entre les deux logiciels produits, HYDRAM en prolog et en orienté objet, doit être effectuée dans plusieurs domaines. Le premier est au niveau de la modélisation informatique. La programmation, la validation et l'optimisation représentent les étapes suivantes. Enfin, l'exploitation et la réutilisation constituent les dernières étapes de comparaison.

➡ **Évaluation de l'orienté objet**

À l'aide du développement réalisé, une évaluation de l'orienté objet et du langage informatique utilisé sera faite. À ce stade, les possibilités de l'orienté objet seront éprouvées, facilitant ainsi la critique de cet environnement informatique et de ses concepts de base. Ses avantages et inconvénients seront présentés et la notion de temps inhérent à ce genre de modélisation informatique sera évaluée.

➡ **Réflexion sur l'utilisation de l'orienté objet et sur son évolution possible**

Une réflexion sur l'utilisation de l'orienté objet comme plate-forme informatique pour de futurs développements sera portée. Cette partie permettra d'effectuer un choix plus éclairé quant à la méthode de modélisation et de programmation à employer.

CHAPITRE II
DOMAINE INFORMATIQUE: L'ORIENTÉ OBJET

Depuis quelques années, les termes « orienté objet » apparaissent fréquemment dans de nombreuses publications, informatiques ou scientifiques, et dans un grand nombre de publicités de produits de logiciel. De plus, plusieurs congrès ou symposiums se sont consacrés au développement du concept orienté objet. Le marché des systèmes orientés objet en Amérique du Nord et en Europe est très important, et pourrait atteindre des sommets d'ici quelques années. Plus de 80 % des applications développées dans les organisations le seront en utilisant une technologie reliée à l'orienté objet. Le Roux (1993) confirme que l'orienté objet commence à s'installer de façon durable, et que les organisations qui ne se préparent pas aujourd'hui à ce virage risque de prendre un retard technologique.

2.1 Présentation

2.1.1 Évolution

Le principe de l'objet n'est pas entièrement nouveau. Il découle d'un besoin réel d'alléger la structure des programmes, et de les rendre plus performants. La difficulté de concevoir et de maintenir des programmes de taille importante a imposé la notion de programmation structurée, représentée dans des langages tel que Pascal. Avec l'émergence de l'intelligence artificielle, il est devenu indispensable de structurer la masse de connaissance manipulée, en regroupant dans une entité unique les informations et les propriétés relatives à un même concept. Cette volonté a été le point de départ aux principes de base de « l'objet ».

Après le développement d'un premier langage informatique, appelé Simula, le langage Smalltalk (Goldberg 1983) est apparu à la fin des années 1970 et représente le premier

véritable langage de programmation objet. D'autres langages sont apparus dans les années 80, comme C++, suivis par de nombreuses recherches sur l'orienté objet. Il a fallu attendre la fin des années 80 pour voir apparaître les premières méthodes d'analyses et de modélisations orientées objet, comme la méthode OMT (Rumbaught, 1991) ou la méthode Coad/Yourdon (Coad et Yourdon, 1991).

2.1.2 Définition

Il est difficile de donner une définition exacte d'une approche orientée objet. Selon les auteurs, cette définition varie. De manière générale, le terme « orienté objet » signifie que le logiciel est organisé comme une collection d'objets indépendants comprenant les données et les opérations possibles. Le concept orienté objet prend toute son importance et son utilité dans la phase précédant la programmation du logiciel. Il s'agit d'une nouvelle manière de penser le logiciel en s'appuyant sur le domaine d'application du développement. Les bases d'un développement de ce type sont l'identification et l'organisation des concepts des domaines d'applications. Nous aborderons la façon d'identifier et de manipuler ces concepts. Il est donc indispensable de bâtir une conception indépendante du langage de programmation retenu.

Ce paragraphe ne constitue qu'une définition possible de l'orienté objet. À la lecture de ce document, le lecteur comprendra la difficulté de définir précisément ce concept. La compréhension, quant à elle, sera beaucoup plus simple.

2.1.3 Illustration

Afin d'illustrer les principales caractéristiques de l'orienté objet, il suffit de se rapporter au système d'interface utilisé par la compagnie Microsoft dans un environnement Windows. Une icône, ou une fenêtre (objet) est sélectionnée avec un pointeur ou une souris. En choisissant un élément dans un menu (envoi de message), l'objet effectue une

opération ou change d'apparence. Certains des principes de base de l'orienté objet viennent d'être utilisés. Un objet particulier a été activé, et par l'envoi d'un message, une opération (ou méthode) s'est déclenchée pour faire réagir l'objet en conséquence. Un système orienté objet est composé d'objets distincts et indépendants. Les objets communiquent entre eux par l'envoi de messages. Un objet qui reçoit un message de la part d'un autre objet réagit en déclenchant la méthode appropriée. Un objet est dit indépendant car lui seul peut manipuler ces données et déclencher ses méthodes.

2.2 Approche orientée objet

Certaines notions théoriques sont importantes à comprendre afin de pouvoir exploiter de façon constructive les possibilités de l'orienté objet. Les propriétés d'une approche orientée objet sont nombreuses. La plupart de ces propriétés sont abstraites pour un utilisateur non familier. À cette étape, il suffit de présenter les propriétés de base pour pouvoir comprendre l'essence d'un développement orienté objet. Selon la littérature et mon expérience, quatre aspects de base méritent d'être présentés: l'entité, les classes, l'héritage et la réutilisation.

Une entité est définie comme un objet dans une approche orientée objet. Elle peut représenter une chose concrète, comme un évacuateur de crues, un document, une fenêtre de résultats, ou par un concept abstrait, comme une vérification, une acceptation ou un rejet. Chaque objet est une entité, et est indépendant de tous autres objets. Un objet est constitué par des attributs et des méthodes. Les attributs identifient les caractéristiques d'un objet. Par exemple, un canal d'amenée est caractérisé par sa largeur, sa position, ses longueurs et ses dénivellations. Les méthodes sont les opérations que l'objet peut exécuter. Par exemple, une méthode permet le calcul de l'intersection entre la longueur uniforme du canal d'amenée et le terrain naturel. Un objet peut posséder un grand nombre d'attributs et de méthodes.

Les objets qui ont la même liste d'attributs et de méthodes sont regroupés dans un même groupe d'objets. Dans le langage orienté objet, ce groupe est appelé une classe. Elle représente un modèle pour un ensemble d'objets qui ont des propriétés communes. La seule différence entre deux objets d'une même classe se situe au niveau de la valeur que prendront les attributs. La présence de classes allège la structure d'un programme en liant les divers objets dont les caractéristiques sont semblables. Chaque objet d'une même classe est appelé instance de cette classe. Par exemple, si le canal d'amenée est considéré comme un objet, chaque canal d'amenée sera un objet différent car les valeurs des attributs de chacun sont différentes, mais ils appartiendront à la même classe. Ainsi, le canal d'amenée du site SM3 et celui du site LG2 appartiennent à la même classe car ils ont la même forme (attributs) et le même comportement (méthodes). Les valeurs que prennent les attributs sont toutefois différentes, les canaux d'amenée pour différents sites sont donc des objets différents.

Les différentes classes créées peuvent avoir un lien entre elles. Ce lien est appelé lien d'héritage. Lorsque des classes peuvent se diviser en sous-classes, les classes de niveau inférieur (sous-classe) héritent des attributs et des opérations d'une classe parente. Une classe peut ainsi être définie d'une manière générale pour ensuite être raffinée dans des sous-classes de plus en plus précises. Par exemple, un canal d'amenée peut posséder une longueur à pente variable selon le profil du terrain. Ainsi tous les canaux ont certaines longueurs, largeurs et positions. La classe parente pourrait regrouper ces attributs communs, tandis qu'une classe de niveau inférieur prendrait en compte le calcul de la pente variable. Cette sous classe hériterait de tous les attributs de la classe parente avec ses attributs et ses méthodes de calculs en plus. Cette propriété de regrouper les classes identiques pour créer des «super-classes» permet de réduire les répétitions dans la conception et dans les programmes. C'est l'un des principaux avantages d'un système orienté objet.

La réutilisation est un des objectifs fondamentaux de tous développements orientés objet. Elle peut être définie par la qualité d'un langage permettant de réutiliser l'existant pour le développement d'autres applications (Beaudoin 1992). La modification et l'évolution d'un développement orienté objet se doivent d'être facilitées par les propriétés de réutilisation qu'offrent toutes modélisations objets. La conception de classes générales ou de dictionnaires de données complets dans un développement permet de les réutiliser en totalité ou en partie dans un futur projet. Par exemple, un dictionnaire contenant toutes les formules hydrauliques utilisées pour la conception d'un évacuateur de crues pourra être réutilisé pour une future application ou des formules hydrauliques seront requises. Pour pouvoir profiter de cette possibilité de réutiliser une partie d'une application, des efforts doivent être fait pour créer des bibliothèques de classes réutilisables.

2.3 Modélisation par objet

Le cycle de vie d'un logiciel se divise en trois étapes qui sont l'analyse, la modélisation et l'implémentation. Les phases d'analyse et de modélisation sont les plus importantes. Dans la littérature informatique courante, l'accent semble être mis sur les problèmes liés à l'étape de programmation d'un logiciel. Les avantages d'un développement orienté objet résident dans les deux premières étapes du cycle de vie, qui sont l'analyse et la modélisation. Si ces deux étapes sont négligées, il est futile d'espérer obtenir un développement orienté objet qui soit clair, évolutif et réutilisable. Son fonctionnement en lui même restera hypothétique. Les faiblesses dans l'étape de modélisation sont beaucoup plus coûteuses que celles survenant dans l'étape de programmation. Les imperfections dues à cette dernière étape se corrigent rapidement.

2.3.1 Principes de modélisation

Selon la méthode de modélisation utilisée, les étapes menant à la réalisation du logiciel varient. Cependant, la plupart de ces méthodes respectent les trois phases du cycle de vie

d'un logiciel. Bien que les phases d'analyse et de conception soient les étapes les plus importantes, il est néanmoins difficile d'appliquer une méthode de modélisation convenant parfaitement au besoin du développement en cours. Chacune des méthodes existantes s'accorde sur certains aspects, notamment la façon de reconnaître les objets et d'organiser les classes. Voici les étapes nécessaires pour aboutir à une organisation des objets et des classes:

- Identification des objets et des classes,
- Identification des attributs et des méthodes,
- Organisation des classes d'objets en utilisant l'héritage,
- Groupement des classes en modules,
- Préparation d'un dictionnaire de données.

Il n'y a pas de recettes infaillibles pour identifier les objets qui seront utilisés pour l'implémentation. Il faut procéder par essais en énumérant tous les objets susceptibles d'exister en fonction du domaine d'application. En les regroupant par la suite selon leurs liens, les classes d'objets apparaissent. La figure 2.1 indique la suite logique des événements.

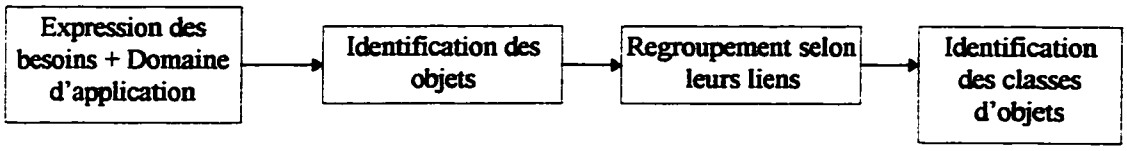


Figure 2.1 : Identification des objets et des classes

La préparation d'un dictionnaire de données n'a pas la même signification d'une méthode de modélisation à l'autre. Il est toutefois important de prévoir le cheminement, l'échange et le stockage des différentes données qui seront manipulées. Ces données peuvent être des valeurs de départ, des résultats intermédiaires et finaux, ou des variables temporaires. Le dictionnaire peut être partagé par différentes classes et différents modules. Pour

obtenir une des données dans le dictionnaire, un message devra être envoyé à l'objet d'où provient la donnée.

Ces quelques paragraphes ne sont qu'une introduction aux principes de bases orientés objet. Le lecteur intéressé pourra consulter les différents volumes qui traitent de ce sujet (Beaudouin, 1992 - Rumbaught, 1991 - Booch, 1991). Un simple exemple est présenté afin d'illustrer ces principes.

2.3.2 Illustration

Dans le logiciel produit en orienté objet, prenons l'exemple du module concernant l'axe d'évacuation.

Dans le module « axe d'évacuation », l'utilisateur doit pouvoir enregistrer un axe d'évacuation pour un évacuateur de crues dans un éditeur d'axe. Cet axe peu comprendre un nombre illimité de points et peut être modifié, complété ou éliminé. Chacun des points de l'axe est identifié par son chaînage et différentes altitudes. L'axe en lui même est identifié par un nom, un site et différentes valeurs (longueur, largeur...). Une aide doit être offerte à l'utilisateur afin de le guider dans son cheminement. Enfin, l'axe doit pouvoir être visualisé à l'aide d'un éditeur graphique. Cet éditeur permet de vérifier les caractéristiques de chaque point de l'axe, et de les modifier si nécessaire.

Pour identifier les objets pertinents et les classes, il faut vérifier les besoins du développement et les différents domaines d'application.

Différents objets se démarquent, à savoir l'éditeur d'axe, l'éditeur graphique, la gestion des opérations effectuées sur l'axe et l'identification de l'axe. Plusieurs de ces objets partagent les mêmes opérations. L'éditeur d'axe et l'éditeur graphique ont des comportements qui sont semblables aux niveaux des caractéristiques de l'interface, de

certaines opérations de modifications, et du traitement du même axe. Il est possible dès à présent d'effectuer certains rapprochements et d'établir certains liens entre ces objets. L'orienté objet offre le concept d'héritage pour éclaircir le programme et permettre de faire des rapprochements logiques entre des objets partageant les mêmes comportements et les mêmes attributs. La figure 2.2 présente un cheminement possible.

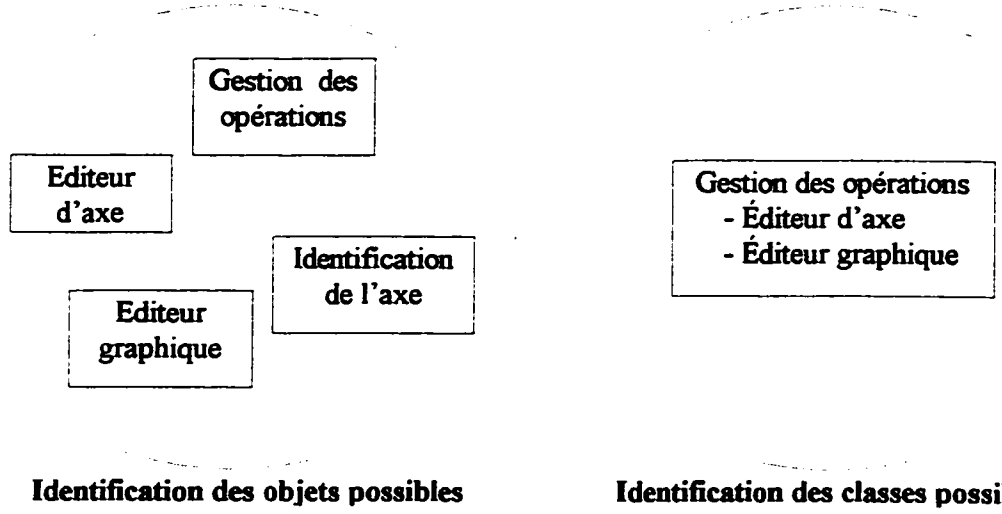


Figure 2.2 : Cheminement orienté objet

La classe « Gestion des opérations » est une classe parente qui transmet toutes les opérations possibles de modifications, de sauvegardes ou de rappels à ses deux sous-classes. Elle inclut également les attributs de l'axe, c'est à dire son identification et ses différentes variables. Les liens d'héritage sont clairs. Les classes « Éditeur graphique » et « Éditeur d'axe » héritent des attributs et du comportement de la classe parente. À ce stade, les termes classes ou objets deviennent synonymes.

Toutes les données recueillies seront transmises à un dictionnaire de données afin d'être utilisées pour la suite du développement. Ce cheminement proposé n'est pas le cheminement unique. Plusieurs autres peuvent être possibles, selon la compréhension du problème par l'utilisateur, ou l'élargissement du domaine.

2.3.3 Méthode de modélisation

Les auteurs de langages orientés objet ne se sont pas préoccupés d'élaborer des méthodes de modélisation afin de mener à bien des développements. Selon des spécialistes du monde de l'orienté objet, la conception d'un logiciel orienté objet est si naturelle, qu'il suffit de considérer le domaine d'étude pour découvrir les objets, les classes et les méthodes pertinentes. Pour quelqu'un de très expérimenté, ce moyen est sûrement possible, mais les concepteurs de logiciel ont besoin d'une aide et de modèle pour exploiter au maximum les possibilités de l'orienté objet. Depuis quelques années, des méthodologies sont proposées comme une solution aux besoins des concepteurs.

Seules les méthodes de modélisation objets nous intéressent à ce stade du développement. Ils en existent plusieurs, et les plus reconnues sont la méthode OMT, la méthode Coad/Yourdon, et la méthode Booch. Les deux premières méthodes préconisent le développement d'une modélisation indépendamment du langage de programmation qui est utilisé.

La méthode Booch est orientée vers le développement de logiciels implantés avec le langage informatique ADA (Booch 1991). Ce langage offre des caractéristiques orientées objet, mais ne permet pas l'héritage. Malgré ses grandes qualités, il ne peut donc être considéré comme un langage orienté objet. Une méthode développée pour un langage spécifique ne permet pas une modélisation complète et fiable, quelque soit le type de problème.

La méthode OMT (Object Modeling Technique) semble être la plus populaire. Elle a été développée au centre de recherche de *General Electric* par J. Rumbaugh et couvre les phases d'analyse, de conception et d'implémentation du cycle de développement (Rumbaugh 1991).

La phase d'analyse consiste à comprendre le modèle d'application et à modéliser le problème suivant les trois aspects suivants: Le modèle *objet*, le modèle *dynamique* et le modèle *fonctionnel*.

- Le modèle *objet* décrit la structure statique en terme d'objets et de relations existant entre eux. Le modèle est représenté graphiquement par des diagrammes d'objet (diagramme de classes et d'instance).
- Le modèle *dynamique* décrit l'environnement actif des objets, c'est à dire les changements qui se produisent dans le temps sur les objets et les relations. Les objets s'échangent des messages, et la réception de ces messages entraîne des changements dans l'état d'un objet. Les concepts du modèle dynamique sont modélisés dans des diagrammes d'états.
- Le modèle fonctionnel sert à décrire les traitements qui constituent l'application. La circulation des données y est représentée. Ce modèle sert à représenter le fonctionnement du logiciel dans son ensemble. Le modèle est représenté graphiquement par des diagrammes de flux de données.

Les phases de conception et d'implémentation du système ont pour objectif de choisir une approche globale pour l'architecture du système. Elles consistent à prendre des décisions concernant la séparation du système en sous-systèmes, la gestion des données... À des fins d'optimisation, de nouvelles classes et associations peuvent être créées.

La méthode Coad/Yourdon a été développée par P. Coad et E.Yourdon (Coad & Yourdon 1991). Le tableau 2.1 présente le cheminement de cette méthode. Trois phases principales sont envisagées, à savoir une phase d'analyse, de conception et d'implantation.

La phase d'analyse débute par la conception du modèle objet en considérant les 5 items : sujet, classe & objet, structure, attribut et service.

Sujet **Classe & Objet** **Structure** **Attribut** **Service**	Composante **Domaine du Problème**	Composante **Interface Usager**	Composante **Gestion des Tâches**	Composante **Gestion des Données**

Tableau 2.1: Modèle de base de la méthode Coad/Yourdon

L'item *sujet* n'est utilisé qu'en cas de système d'envergure. Il sert à sectionner le modèle original en modèles de tailles inférieures plus facilement exploitables. L'item *Classe & objet* est celui qui représente les classes du système sans s'intéresser aux attributs, services et relations entre objets. L'item des *Structures* représente les constructions de type "Généralisation-Spécialisation" et les constructions de type "Tout-Partie". La notion d'héritage s'applique aux constructions de type "Généralisation-Spécialisation". L'item des *Attributs* identifie pour chaque classe les informations qui la caractérisent et pour lesquelles chaque objet de la classe possède ses propres valeurs. L'item final Service décrit le comportement des objets de chaque classe du modèle. Ce niveau correspond aux services requis par la classe et aux interactions entre les objets qui communiquent par des messages.

La phase de conception ajoute les détails d'implantation au modèle objet en considérant les quatre composantes: domaine du problème, interface usager, gestion des tâches et des données. La première activité consiste à revoir le modèle objet dans le but de répondre aux critères de réutilisation et de performance. L'aspect *interface-usager* concerne la présentation et la communication. L'aspect *gestion des tâches* correspond à la définition et à la coordination des tâches, tandis que la *gestion des données* concerne la représentation et la gestion des données (Harvey et Moulin, 1993).

2.3.4 Modélisation retenue

Harvey et Moulin (1993) ont effectué une comparaison entre ces différentes méthodes de modélisation. Le tableau résumé se trouve en annexe 1. Les conclusions de cette étude présentent la méthode de Coad/Yourdon comme une méthode intéressante pour des projets de petites à moyennes envergures. La phase d'analyse est très satisfaisante au niveau de la conception du modèle objet. La phase de conception est quant à elle beaucoup moins réussie. Sur la méthode OMT, les auteurs précisent que c'est une méthode applicable sur des projets de différentes envergures. Le formalisme est suffisamment riche, bien que certaines améliorations peuvent être apportées. La documentation sur cette méthode est assez claire bien que manquant d'exemples.

Les critères de choix de la méthode de développement sont le respect des objectifs du développement du logiciel, à savoir d'utiliser les concepts de base orientés objet, tout en permettant la réutilisation, la modification et l'évolution du logiciel. Le problème majeur dans le choix d'une méthode est qu'il est très difficile de suivre une méthode comme OMT ou Coad/Yourdon sans avoir préalablement une bonne expérience dans l'orienté objet. Les cours n'étant pas très répandus pour faciliter mon adaptation à l'environnement orienté objet, la modélisation a été faite en suivant partiellement la méthode OMT (bien documentée) et en se basant sur les principes de l'orienté objet. Une fois le développement terminé, la méthode utilisée sera commentée.

2.4 Langages informatiques

2.4.1 Langages disponibles

Il est possible de faire du développement orienté objet avec plusieurs types de langages informatiques qui sont disponibles sur le marché. Les langages dits dédiés sont le premier type. Ils ont été développés pour le concept orienté objet. Les principaux sont Smalltalk

et Eiffel. Leurs utilisations obligent à respecter le concept objet. Ils prennent en compte les objets et ses principales propriétés comme l'héritage et l'association de données avec les opérations. Le deuxième type représente les langages hybrides comme le C++. Ces langages permettent de réaliser facilement des développements orientés objet, mais ils offrent également la possibilité de contourner les concepts objets pour faire de la programmation procédurale normale. L'utilisation du C++ exige ainsi un encadrement très strict pour obtenir un développement entièrement objet. Le dernier type de langages sont ceux qui n'ont pas été développés en fonction de l'orienté objet. C, Pascal et Prolog sont de ce type. Il peuvent toutefois être exploités pour des développements objets, mais l'exercice peut se révéler très lourd et très difficile.

Rumbaugh et al. (1991) ont procédé à une comparaison entre ces différents langages en établissant plusieurs paramètres. Ces paramètres de comparaison représentent toutes les caractéristiques du concept orienté objet. Le tableau résumé se trouve en annexe 2. Les principales conclusions de cette étude sont que le langage Smalltalk est remarquable pour son environnement de développement et sa bibliothèque de classes étendue. Il offre les meilleures caractéristiques, à la fois des langages et des environnements de développement. Il est idéal pour le développement rapide de prototype. C++ est un langage hybride fortement fondé sur le langage C, mais offre la programmation orientée objet. Son efficacité est obtenue au détriment d'une certaine souplesse de conception. Son accessibilité et la disponibilité de certains compilateurs font de lui un langage orienté objet très répandu.

2.4.2 Langage informatique retenu

Seul un type de langage purement objet peut être envisageable pour le développement. Afin d'effectuer une comparaison fiable entre les deux développements, il faut utiliser au maximum les propriétés orientées objet. De plus, la faiblesse dans les méthodes de modélisation disponibles, rend impensable l'utilisation d'un langage qui ne serait pas

orienté objet; le développement obtenu ne respecterait pas les principes de l'objet. À la lumière de l'évaluation faite sur divers langages objets, Smalltalk sera utilisé pour l'implémentation du développement. La version disponible est Smalltalk\V Digitalk. Cette version présente toutes les caractéristiques de Smalltalk 80, mais offre un environnement de travail très supérieur, et diminue le temps de programmation. Smalltalk est le premier langage orienté objet. Il a été développé par Xerox au Palo Alto Research Center dans les années 70.

CHAPITRE III
DOMAINE SCIENTIFIQUE - LES ÉVACUATEURS DE CRUES

Un évacuateur de crues est un ouvrage permettant l'évacuation d'une montée d'eau dans un réservoir. C'est un ouvrage de sécurité qui protège le barrage principal et les digues secondaires d'un écoulement d'eau au dessus de leur crête. Un déversement aurait des conséquences catastrophiques sur la résistance de ces digues.

Les principes hydrauliques menant à la conception des évacuateurs de crues sont largement couverts par une littérature abondante. Les notions hydrauliques sont connues et éprouvées, alors que l'écriture d'un cheminement structuré pour dimensionner un tel ouvrage n'est guère développée. L'objet de cette partie n'est donc pas de rechercher et de classifier les différents moyens disponibles pour la conception, mais de relever tous les éléments pertinents à la description hydraulique et structurale d'un évacuateur de crues.

3.1 Description de l'évacuateur de crues

Deux principaux types d'évacuateurs de crues se distinguent suivant la position de leur entonnement par rapport au niveau normal de la retenue: les évacuateurs de surface et les évacuateurs de fond ou demi-fond. L'entonnement de ce dernier type d'évacuateur est situé sous la surface du réservoir. L'écoulement est sous pression dans une grande partie de son développement. Le débit est contrôlé par une vanne placée à l'extrémité de la partie en charge. Les évacuateurs de surface peuvent être ou non contrôlés par des vannes ou des portes. L'eau excédentaire est prélevée proche du niveau normal de la retenue. L'écoulement se fait à surface libre et est dirigé vers un point de restitution. Ces deux variantes d'évacuateurs sont les plus utilisées. Il existe également des évacuateurs de surface en charge, comme les siphons de décharge (CIGB 1987).

Un canal d'amené permet d'acheminer l'eau du réservoir vers l'évacuateur de crues. La longueur de ce canal varie en fonction de la position de l'évacuateur et de la topographie du site.

Un évacuateur de surface est normalement composé de trois parties (voir figure 3.1):

- Un seuil déversant: Il est le point le plus élevé de l'ouvrage et contrôle ainsi le débit évacué. L'écoulement transite par ce seuil avant d'atteindre la deuxième partie de l'ouvrage.
- Un coursier: Il achemine l'eau vers le point de restitution. Sa longueur et sa configuration dépendent du type de barrage, des conditions d'écoulement et des coupes topographiques.
- Un ouvrage terminal à partir duquel le flot évacué est restitué au lit naturel. Ce dernier ouvrage permet également de dissiper une trop forte énergie de l'eau avant l'entrée dans la rivière. Cet ouvrage est appelé dissipateur d'énergie.

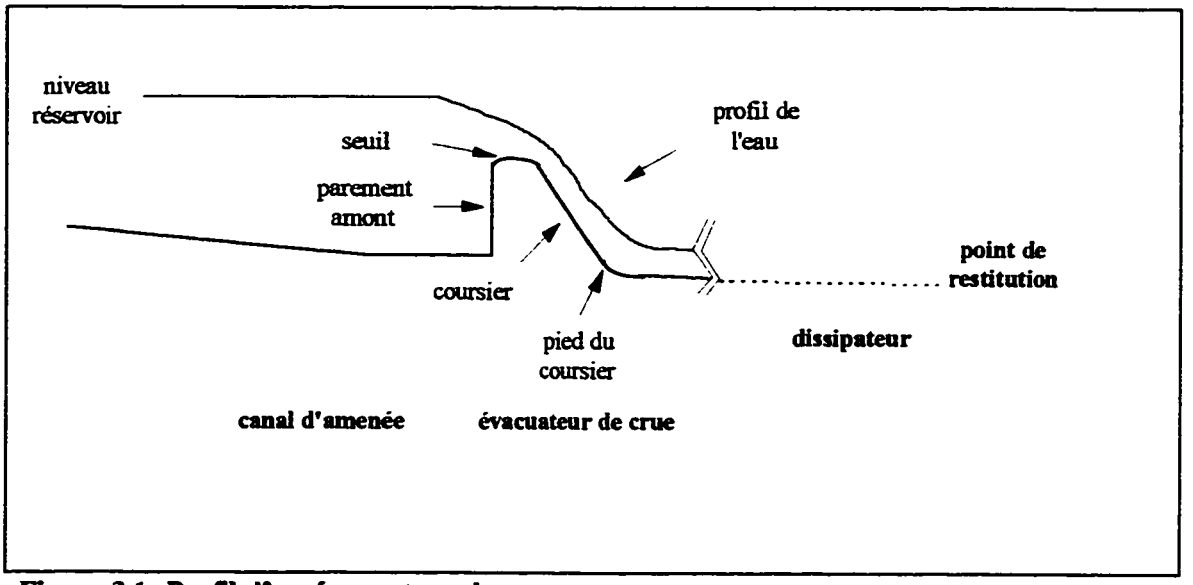


Figure 3.1: Profil d'un évacuateur de crues

3.2 Considérations hydrauliques et structurales

La conception d'un évacuateur de crues est guidée par des considérations hydrauliques et par des choix à faire au niveau structural.

3.2.1 Profil de l'évacuateur

Dans le design d'un profil d'évacuateur, il est important de s'assurer que la nappe d'eau suive le profil établi (CIGB 1987). Si la conception est inadéquate, plusieurs problèmes peuvent apparaître: si la crête est sous-dimensionnée (courbe du seuil trop forte), le jet d'eau aura tendance à se décoller de la surface, et ainsi créer des zones de pression négative. Dans ces régions, il y a de forts risques d'arrachement du matériel constituant la crête. Si la crête est surdimensionnée à outrance (courbe du seuil trop faible), des problèmes peuvent apparaître à la base du coursier avec un risque de dommages importants (Sokolov, 1979).

Le profil déversant du radier prend la forme de la surface libre inférieure d'une lame déversante d'épaisseur H_D (Hauteur de design de la lame d'eau au dessus de la crête). Deux types de profils sont disponibles, le type creager et le type parabolique. L'expression analytique des profils déversants d'après Design of Small Dams (1987) est donnée par:

Creager : $\frac{Y}{H_D} = K * \left(\frac{X}{H_D}\right)^N$ **Parabolique :** $\frac{Y}{H_D} = 0.25 * \left(\frac{X}{H_D}\right)^2$

avec : H_D : Hauteur de design de la lame d'eau au dessus de la crête,

X,Y : Coordonnées horizontales et verticales,

K,N :Coefficients dépendants du rapport ha/H_D et de l'angle du parement amont,

ha : Énergie cinétique au-dessus de la crête.

La sélection d'une des deux crêtes envisagées est guidée par certaines considérations, à savoir le débit provenant du réservoir et la hauteur de chute. Ainsi, il semble être préférable d'utiliser une crête de type creager pour des débits élevés ou de fortes hauteurs de chutes sur des évacuateurs dépourvus de vannes. Par contre, il est conseillé d'employer une crête de type parabolique pour des ouvrages avec des vannes opérant à de petites ouvertures. Le profil parabolique est moins plongeant que le creager pour une même hauteur d'eau au dessus de la crête. Les graphiques utiles pour dimensionner ces profils sont tirés de Design Of Small Dams (1987) et de différents ouvrages réalisés par la Société d'Énergie de la Baie James (SEBJ).

Les coursiers sont nécessaires pour conduire le flot déversé jusqu'au point de restitution à l'aval du barrage. Leurs longueurs et leurs configurations dépendent du type de barrages, des conditions d'écoulement et des coupes topographiques. Les coursiers sont généralement à l'air libre et ont un tracé rectiligne, ils sont prévus pour des écoulements à surface libre torrentiels. Une fois le point de tangence relevé entre la crête et le coursier, une droite est prolongée suivant une certaine pente qui est de l'ordre de 1V:0,7H.

Le pied du coursier se situe donc à la fin de la droite formant le coursier, et est constitué par un simple arc de cercle. Son rôle est de permettre à l'écoulement de prendre une certaine direction pour ensuite entrer dans le dissipateur d'énergie.

3.2.2 Paramètres de conception

Cette partie se veut un résumé des recherches et analyses portées sur les divers coefficients utiles à notre étude. Il existe un grand nombre de coefficients dans le domaine de l'hydraulique, autant dans la caractérisation de l'écoulement que la conception pure d'ouvrages hydrauliques. Les renseignements qui vont suivre viennent non seulement d'une littérature généreuse, mais également de toute la recherche effectuée par M. Benoît

Robert dans le cadre de la structuration de règles pour la conception d'évacuateurs (Robert, 1996).

Les coefficients hydrauliques sont souvent dépendant les uns des autres. De nombreux diagrammes sont utilisés dans Design Of Small Dams (1987). Des graphiques supplémentaires provenant de Hydro-Québec ont permis de compléter notre savoir grâce à leurs expériences en matière d'ouvrages hydroélectriques. Le premier offre une aide pour la recherche du coefficient de débit et le deuxième apporte un facteur de correction à ce coefficient pour les évacuateurs de type parabolique. Une liste des courbes utilisées est présentée en annexe 3.

L'utilisation de ces coefficients a un but précis, celui de choisir un type d'évacuateur et de faire sa conception. Le déroulement utilise ou produit deux types de caractéristiques :

➡ Les caractéristiques physiques :

- nombre de passes,
- largeur d'une passe,
- nombre de piliers,
- largeur d'un pilier,
- nombre de contreforts,
- largeur d'un contrefort,
- largeur de l'évacuateur,
- hauteur de pelle,
- inclinaison du parement amont,
- profil du radier aval,
- profil et largeur du canal d'amenée,
- profil et largeur du dissipateur d'énergie.

➡ Les caractéristiques hydrauliques :

- débit de conception,
- débit de vérification,
- niveau maximal d'exploitation amont,
- niveau d'eau aval pour le débit de conception,
- chaînage de l'évacuateur,
- vitesse d'écoulement en rivière en crue.

Les valeurs déterminées sont les suivantes :

- dénivellation hydraulique, DH
- hauteur d'eau, H_{eau}
- hauteur de charge, He
- hauteur de conception, H_D
- coefficient de débit, C_D
- coefficient de débit effectif, Ce
- hauteur d'énergie cinétique, ha.

Les équations nécessaires pour l'obtention de ces coefficients se trouvent dans la littérature hydraulique citée précédemment. Les liens entre les différentes équations, et donc le cheminement pour la conception a été produit par Benoît Robert en collaboration avec des experts d'Hydro-Québec.

3.2.3 Considérations hydrauliques

Ce projet faisant référence à une étude privée, il est important d'identifier les paramètres hydrauliques qu'il faut considérer. Les différents symboles sont représentés sur la figure 3.2.

Les évacuateurs de crues envisagés sont généralement dimensionnés pour pouvoir faire face à une crue maximale probable. Le débit provenant de cette CMP (Crue Maximale Probable) est fixée par le concepteur de l'ouvrage à la suite des études appropriées.

Selon les dénivellations amonts et avals, deux types d'aménagements sont envisagés, à savoir un aménagement de type « basse chute » et un de type « haute chute ». Une dénivellation hydraulique totale, différence entre le niveau d'eau amont et le niveau d'eau aval, de 7,5 mètres au débit de conception, représente la limite entre les deux types d'aménagements. Pour un aménagement de type « basse chute », seul un profil parabolique est considéré, alors que pour un aménagement de type « haute chute », les profils paraboliques et creagers sont envisagés.

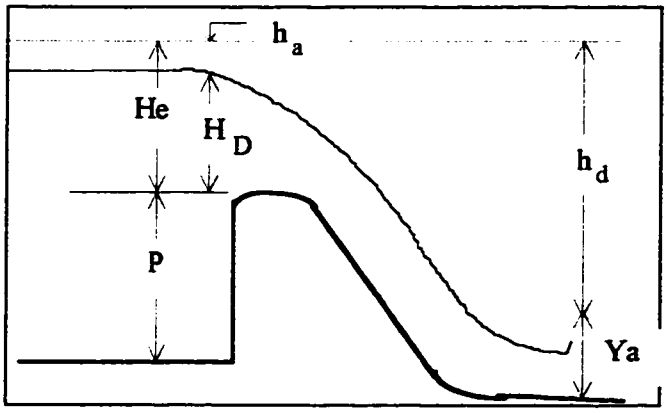
La capacité d'évacuation d'un évacuateur de crue est connue, elle est fonction de ses caractéristiques physiques. Elle dépend du coefficient de débit effectif, de la largeur effective et de la hauteur de charge au dessus du seuil. La formule est donnée par :

Débit : $Q = Ce.Le.He^{3/2}$ avec Ce : Coefficient de débit effectif

Le : Largeur effective

He : Hauteur de charge au dessus du seuil

Selon le type d'aménagement, le calcul du coefficient de débit est différent. Pour un aménagement de type « basse chute », il faut tenir compte de la corrélation entre l'influence du radier aval et le facteur de submergence. Ces liens sont illustrés aux figures 9.27 et 9.28 de Design of Smalls Dams (1987). Pour des dénivellations hydrauliques totales supérieures à 7,5 mètres, le facteur de submergence est pratiquement toujours négligeable.



avec

He : Charge au dessus du seuil

H_D : Hauteur de la lame d'eau

P : Hauteur du parement amont

h_a : Énergie cinétique en amont

h_d : Énergie cinétique en aval

Ya : Hauteur d'eau en aval

Figure 3.2: Coefficients hydrauliques utilisés

3.2.4 Considérations structurales

Un évacuateur de crues est souvent alimenté par plusieurs voies d'accès séparées par des piliers. Ces piliers imposent à l'écoulement une direction déterminée afin que le débit soit distribué uniformément tout le long du déversoir. Ils supportent généralement un pont d'accès traversant le déversoir, et permet de loger des installations mécaniques comme des vannes. Pour tenir compte de l'influence de ces piliers sur l'écoulement, il faut travailler avec une largeur effective. Cette largeur englobe le nombre, la largeur et la forme des piliers. La figure 3.3 illustre les voies d'accès.

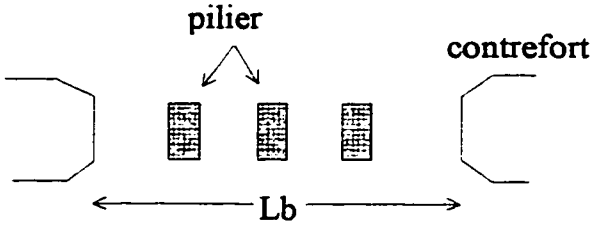


Figure 3.3: Influence des piliers

La formule de la largeur effective est donnée par:

Longueur effective : $Le = Lb - 2\left(Npl.K_{pl} + K_c\right).He$

avec : Le : largeur effective

Lb : largeur réelle

Npl : nombre de pilier

Kpl et Kc: coefficients de perte associés aux piliers et aux contreforts

He : Charge au dessus de la crête

Sur toute la largeur de la crête de l'évacuateur, la présence de piliers délimite un certain nombre de passes. Chacune de ces passes est caractérisée par sa largeur (Bpa). Le nombre de passe (Npa) dépend de l'importance du débit provenant du réservoir et entrant dans l'évacuateur de crues.

3.2.5 Canal d'amenée

Le canal d'amenée relie l'évacuateur de crues au réservoir en amont. Pour différentes raisons hydrauliques et économiques, l'évacuateur peut se situer à une certaine distance du réservoir. Il est donc important de prévoir un canal d'amenée dont la longueur, la largeur et la pente permettent à l'eau de s'acheminer vers l'évacuateur. Le canal débute au point de chaînage 0+000 sur un axe et se termine au parement amont de l'évacuateur. Plusieurs points définissent l'axe (point 1, point 2...). Il est constitué d'une partie horizontale et d'une partie inclinée si requise. Cette partie inclinée est nécessaire si la partie horizontale rejoint le sol avant le premier point de l'axe. La figure 3.4 illustre cette situation. Les formules utilisées sont des calculs d'intersection de droites.

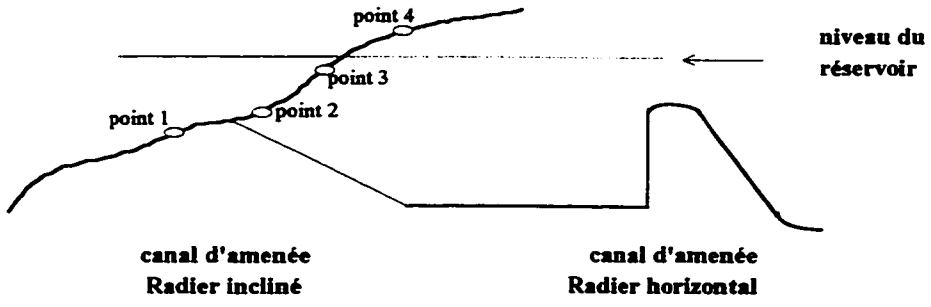


Figure 3.4: Canal d'amenée

3.3 Conception

La conception d'un évacuateur de crues requiert l'emploi de règles précises de conception et le suivi d'un cheminement fiable. Ce cheminement respecte tous les cas de figures possibles et tout les aménagements envisageables.

3.3.1 Cheminement

Le diagramme de conception a été élaboré par Benoît Robert en collaboration avec J. Joannette et S. Colameo pour le service hydraulique de Hydro-Québec. Tout le cheminement hydraulique est présenté dans le rapport « HYDRAM - Un système à base de connaissance pour la conception des évacuateurs de crues (Robert, 1996) ».

Les différentes étapes de conception sont présentées dans un ordre précis. La figure 3.5 présente une partie du cheminement. Elle est valide pour des conditions de hautes chutes, et pour des seuils de type parabolique et de type creager. Chaque étape est constituée par des règles précises qui orientent la conception selon les résultats précédents et les formules de bases.

3.3.2 Hypothèses et valeurs de départs

Afin d'uniformiser la conception d'évacuateurs de crues, il est indispensable d'utiliser un certain nombre de valeurs de conception fixées par des experts ou par le concepteur. Ces valeurs de départ correspondent à des critères de conception de départ basés sur les connaissances des experts et sur des calculs hydrauliques et structuraux.

Les hypothèses et valeurs de départ peuvent être répertoriées selon plusieurs types:

- Valeurs dépendant du débit de conception: Les largeurs des piliers et des contreforts dépendent du débit de conception.

- ➡ Limites: Pour chaque valeur modifiable par le concepteur, une limite inférieure et supérieure sont présentées.
- ➡ Valeurs par défaut: Des données hydrauliques ou structurales sont proposées au concepteur. Ces valeurs correspondent aux pratiques habituelles et connues. Elles sont toutefois modifiables pour s'adapter aux différentes situations.
- ➡ Rapport d'initialisation: Différents rapports d'initialisation sont imposés au concepteur. Ces valeurs de départ se doivent d'être respectées afin d'assurer une conception fiable de l'évacuateur de crues. La plupart de ces rapports proviennent de l'expérience des experts. Trois rapports sont proposés, P/H_D (Hauteur du parement amont/hauteur de la lame d'eau au dessus du seuil), Bpa/He (largeur d'une passe/ hauteur de charge au dessus du seuil) et le facteur radier aval.

Le tableau 3.1 présente un résumé des valeurs de départ utilisées dans cette étude.

Tableau 3.1: Critères de conception des évacuateurs

Critère de conception	Valeurs par défaut	Limite inférieure	Limite supérieure
Vitesse base du coursier	15 m/s	7 m/s	30 m/s
Caractéristiques des piliers:			
• Largeur		2,5 m	4,5 m
➡ $Q < 1500\ m^3/s$	2,5 m		
➡ $1500 m^3/s \leq Q < 4000 m^3/s$	3,5 m		
➡ $Q \geq 4000 m^3/s$	4,25 m		
• Coefficient de contraction	0,01	0,00	0,02
Caractéristiques des contreforts:			
• nombre	2	0	2
• Largeur		1,25	2,125
➡ $Q < 1500\ m^3/s$	1,25 m		
➡ $1500 m^3/s \leq Q < 4000 m^3/s$	1,75 m		
➡ $Q \geq 4000 m^3/s$	2,125 m		
• Coefficient de contraction	0,1	0,0	
caractéristiques du radier aval:			
• facteur multiplicateur du rayon	4	4	100
• angle d'ouverture maximal	35^0	0	35^0
• pente maximale du coursier	0,7	0	0,7

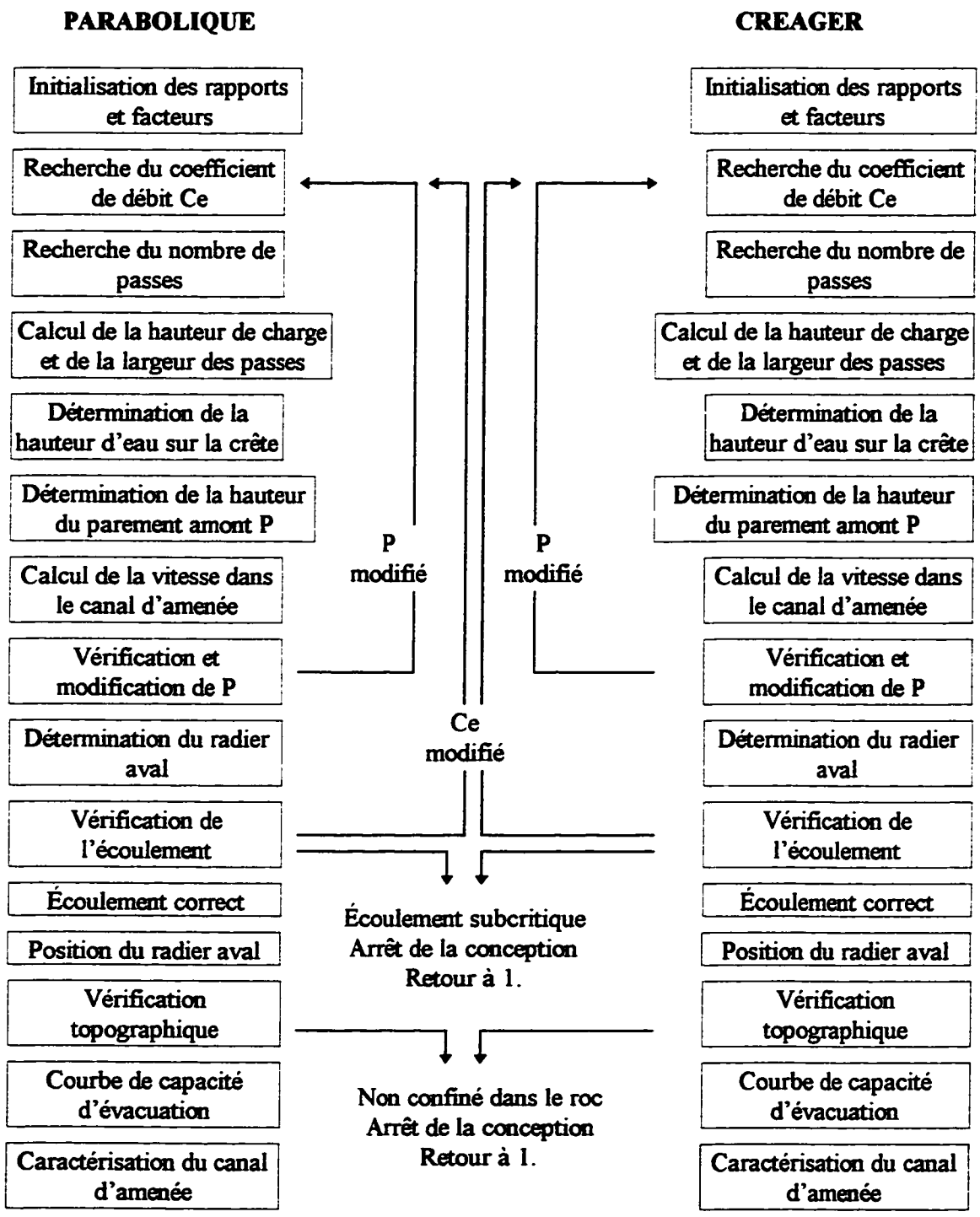


Figure 3.5 : Conception d'un évacuateur en haute chute

3.5 Modèle de référence

HYDRAM est un système expert d'aide à la conception d'évacuateurs de crues. C'est un système à base de connaissance qui est utilisé en appliquant des règles et des raisonnements d'experts. Pour réaliser cette étude, les cogniticiens et experts du groupe de développement ont approfondi point par point les méthodes et les critères en usage au service hydraulique. Les raisonnements invoqués par les experts ont été codifiés pour être exploitables informatiquement.

Un première version du logiciel produit était présentée en 1991. Depuis, la base de connaissance a été accrue et modifiée en profondeur.

Les règles de connaissance sont la transcription des connaissances formulées par les experts internes et externes, à Hydro-Québec, consultés lors du développement du système. Plusieurs règles de connaissance sont présentes pour chaque étape de la conception. Par exemple, à l'étape « Initialisation des rapports et facteurs », quatre règles peuvent être utilisées en fonction de certaines variables du projet à l'étude.
Les règles sont exprimées sous la forme de règles de production:

Si : conditions d'application

Alors : conclusions.

Elles suivent le raisonnement de l'esprit humain par leur simplicité. Voici un exemple des règles de connaissance pour l'étape « Initialisation des rapports et des facteurs »:

Règle 1: **Si** : - évacuateur de type «parabolique»

- Q conception < 4000 m^3/s

Alors: - rapport $P/H_D = 0.4$

- rapport Bpa/He = 0.7

- facteur radier aval = 1.0

Règle 2: **Si** : - évacuateur de type parabolique
- Q conception ≥ 4000 m^3/s

Alors: - rapport $P/H_D = 0.5$
- rapport Bpa/He = 0.8
- facteur radier aval = 1.0

Règle 3: **Si** : - évacuateur de type creager
- Q conception < 4000 m^3/s

Alors: - rapport $P/H_D = 0.32$
- rapport Bpa/He = 0.7
- facteur radier aval = 1.0

Règle 4: **Si** : - évacuateur de type creager
- Q conception ≥ 4000 m^3/s

Alors: - rapport $P/H_D = 0.5$
- rapport Bpa/He = 0.8
- facteur radier aval = 1.0

Lorsque la conception des évacuateurs de crues est rendue à cette étape, les conditions d'applications sont évaluées et la règle correspondante est appliquée pour obtenir les conclusions utiles à la suite du cheminement.

3.5 Modules retenus pour le développement orienté objet

Le développement Hydram comprend plusieurs modules. Seulement certains d'entre eux seront modélisés sous forme orienté objet. Notre but n'est pas de produire une version complète en orienté objet, mais de produire un logiciel objet qui pourra être comparé avec une partie de ce développement. Les différents modules initiaux de HYDRAM sont :

- **Axe d'évacuation,**
- **Évacuateur de crues:**

- ⇨ Conception en basses chutes
  - ➡ Parabolique
  - ➡ Creager
- ⇨ Conception en hautes chutes
  - ➡ Parabolique
  - ➡ Creager

- **Dissipateur d'énergie**
- **Canal d'amené**
- **Calcul des volumes d'excavation.**

Pour effectuer une comparaison valable et pour produire un logiciel utilisable, différents modules ont été modélisés en orienté objet :

- **Axe d'évacuation,**
- **Évacuateur de crues, conception en hautes chutes:**
  - ➡ Parabolique
  - ➡ Creager
- **Canal d'amené**

La conception en conditions de basses chutes, la partie sur les dissipateurs et le calcul des volumes d'excavation ne seront pas abordées dans le développement objet.

CHAPITRE IV
DÉVELOPPEMENT HYDRAM EN ORIENTÉ OBJET

Cette partie de l'étude présente le développement orienté objet réalisé à partir des règles de connaissances du projet HYDRAM. L'analyse et la modélisation sont des étapes du développement qui ont été réalisées avant et pendant l'étape d'implémentation. Le but du mémoire n'étant pas de présenter les lignes de codes (une centaine de pages), seulement une certaine partie de ces lignes est présentée en annexe 5, à titre d'exemple.

4.1 Objectifs de développement

Cette étude nous permet de faire une évaluation de l'orienté objet, autant dans la modélisation que dans l'implémentation. Pour mener à bien ce projet, il est important de manipuler les concepts orientés objet en respectant ses principes de base. Le respect de ces principes permet d'obtenir une modélisation complète et une implémentation fiable. Une comparaison est faite entre deux développements parallèles. Pour que cette comparaison soit faisable, les possibilités de l'orienté objet doivent être éprouvées en grande partie. Les avantages et les défauts de ces possibilités pourront ainsi être évalués. Le développement produit doit pouvoir permettre plusieurs fonctions principales.

L'entretien du logiciel doit pouvoir être facilité par une structure adaptée et résistante, et par des objets clairement identifiés. Une éventuelle modification du raisonnement peut survenir ou une modification en fonction du projet traité peut être apportée. La clarté dans le raisonnement, le regroupement des règles de conception et un cheminement clair sont indispensables pour permettre un entretien aisé. Le suivi de la modélisation et de l'implémentation permet, non seulement de s'assurer du bon cheminement de la programmation mais, aussi, de la compréhension de l'utilisateur. Avec un cheminement clair, il peut suivre et comprendre les procédures de calculs et de modélisation. L'orienté

objet permet à un développement de demeurer évolutif à la fin de son cycle de vie. Hydram, version orientée objet, se doit d'atteindre cet objectif. L'évolution possible d'un logiciel permet de lui ajouter des composantes et de le mettre à jour. Enfin, le principe de réutilisation doit pouvoir s'appliquer en développant certaines librairies de classes. Il est difficile de juger le potentiel de réutilisation d'un développement. Avec une modélisation claire et une programmation soignée, un tel résultat devrait pouvoir être atteint.

4.2 Structure interne

Le chapitre 3 présente les grandes lignes du développement HYDRAM et les modules retenus pour le développement orienté objet. La structure interne et la structure de raisonnement sont présentées et sont illustrées à l'aide d'un exemple.

Dans cette étude, les développements généraux qui ont été retenus sont l'axe d'évacuation, l'évacuateur de crues en hautes chutes et le canal d'amenée. Il n'existe pas de solution unique quant à la modélisation de ce développement. Elle dépend des objectifs initiaux de développement au niveau de l'évolution, de l'entretien, du suivi et de la réutilisation.

Pour assurer le respect de ces objectifs, plusieurs modules d'importance ont été retenus sous des noms différents :

- Hydram Gestion,
- Hydram Evacuateur,
- Hydram Interface,
- Axe Opérateur,
- Librairies.

Ces différents modules couvrent l'intégralité du développement et chacun d'eux contient différents objets et plusieurs classes. Les liens entres ces modules seront illustrés à l'aide d'un exemple.

4.2.1 Hydram Gestion

Le module Hydram Gestion assure L'initialisation de l'ensemble du développement, au niveau de la conception des évacuateurs de crues, des dictionnaires et des variables. Deux types d'évacuateurs sont disponibles, le parabolique et le creager. Les méthodes de conception sont identiques pour les deux types d'évacuateurs. Seuls certains rapports d'initialisation et certaines formules varient. Le module Hydram Gestion permet d'orienter la conception suivant le type d'évacuateur à l'étude. L'initialisation des valeurs par défaut et la création de certains dictionnaires de travail se font à cette étape.

Les objets initiaux dans ce module sont la gestion des évacuateurs de type parabolique et la gestion des évacuateurs de type creager. Il est possible de rassembler ces objets afin de détailler et d'éclaircir la structure. Les évacuateurs de type parabolique et de type creager partagent certaines opérations, comme le cheminement de calcul et l'initialisation de divers dictionnaires. Les liens d'héritage peuvent ici être utilisés pour respecter les concepts de l'orienté objet. La figure 4.1 présente les objets possibles et l'organisation retenue.

La représentation finale de ce module est simplifiée par les classes Creager et Parabolique qui héritent du comportement et des attributs de la classe parente Hydram Gestion. Le tableau 4.1 présente un résumé des différentes opérations disponibles (méthodes) et des variables (attributs) dans ces classes.

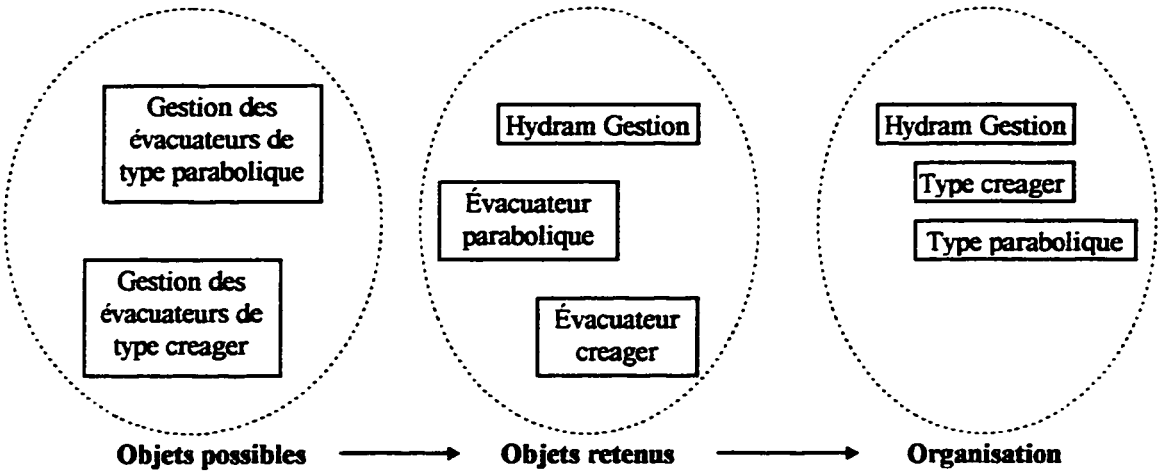


Figure 4.1 : Représentation du module Hydram Gestion

Les positions des objets possibles et retenus dans la figure 4.1 ne sont pas alignées avant que l'organisation ne soit faite par les liens d'héritage.

Tableau 4.1 : Description du module Hydram Gestion

CLASSE	VARIABLES	METHODES	DESCRIPTION
Hydram Gestion	variables de travail	- Init de dictionnaire résultats - Init de dictionnaire travail - Init de la suite de calculs - Orient de la suite de calcul	Initialisation des dictionnaires et de la suite de calcul, orientation des calculs.
Type Creager		- Init de coefficients - Init de dictionnaire travail	
Type Parabolique		- Init de coefficients - Init de dictionnaire travail	

Lecture du tableau:

Init : Initialisation & Orient : Orientation

Ce module respecte le concept évolutif de l'orienté objet. D'autres types d'évacuateurs ou d'autres types de chutes (Basse chute) peuvent être considérés dans cette partie du développement.

4.2.2 Hydram Evacuateur

Un évacuateur de crues est un ouvrage constitué de plusieurs parties comme le seuil, les piliers, les passes... Chacun de ces items peut exister en dehors d'un Évacuateur de crues, mais un évacuateur ne peut exister sans le rassemblement de ces parties. Le module Hydram Évacuateur est le module le plus important du développement. Il assure le lien entre tous les objets permettant la conception finale d'un évacuateur de crues. Les différents calculs structuraux et hydrauliques sont gérés à partir de ce module.

Les objets initiaux étant associés à un évacuateur de crues sont nombreux. Ils concernent les éléments structuraux et les considérations hydrauliques. Il est possible et souhaitable de rassembler ces objets afin d'utiliser les notions d'héritage de l'orienté objet. Par définition, un objet est indépendant. Il doit donc pouvoir être utilisé indépendamment des autres objets si nécessaire. Dans le choix des objets, il est important de respecter ce principe, peu importe la façon dont ils seront implémentés. La figure 4.2 présente les objets initiaux et les objets retenus du module Hydram Evacuateur.

Tous les items se rapportant au radier, comme la crête, le coursier et le pied, ont été regroupés dans un objet nommé **Seuil**. Ces items sont dépendants les uns des autres. Les différentes composantes d'un évacuateur sont représentées par des objets, dont la classe parente est Hydram Evacuateur. Cette classe initialise un dictionnaire de valeurs par défaut qui est utilisé par le reste des classes. La figure 4.3 présente l'organisation des objets avec la notion d'héritage.

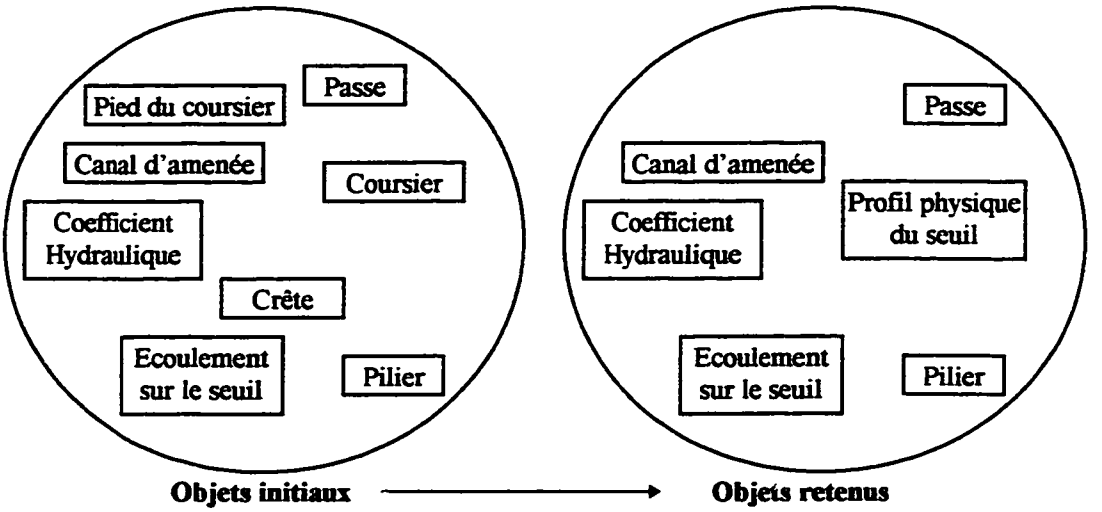


Figure 4.2 : Objets initiaux et retenus dans le module Hydram Evacuateur

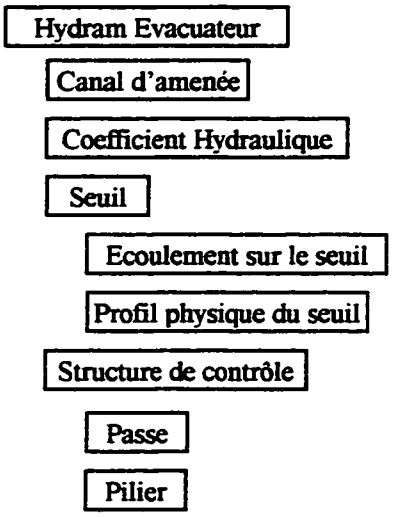


Figure 4.3 : Organisations du module Hydram Evacuateur

Le tableau 4.2 présente un résumé des différentes opérations disponibles (méthodes) et des variables (attributs) dans ces classes.

Tableau 4.2 : Description du module Hydram Evacuateur

CLASSE	VARIABLES	METHODES	DESCRIPTION
Hydram Evacuateur		- Init dictionnaire défaut - Init dictionnaire travail	Fournit les diction - naires aux sous-classes
Canal d'amenée	- Longueurs - Largeurs - Pentes - Vitesse - Chaînage - Élévation	- Calcul de largeurs - Calcul de longueurs - Calcul de pentes - Calcul de vitesse - Points d'intersection	Contient les étapes de calcul, procède à la conception et aux modifications.
Coefficient hydraulique	- Ce - He - H_D - ha	- Calcul coefficient débit - Hauteur de charge - rapport Bpa/He	Effectue les calculs de certains coefficients hydrauliques
Seuil	- Le	- Calcul largeur effective	Rassemble les deux sous-classes
Ecoulement	- hd - Ya - ha	- Hauteur d'eau en aval - Énergie cinétique - Courbe de capacité	Effectue les calculs de l'écoulement sur le radier et des coefficients
Physique	- P - Rayon - Chaînage - Élévation - Coefficients	- Profil du radier aval - Hauteur du parement - Coordonnées de points - Calcul de rayon - Coefficient k et n (seuil)	Produit le profil du radier aval et calcule la géométrie de certains éléments
Structure de contrôle	- Le	- Calcul largeur effective	Rassemble les deux sous-classes
Passe	- Npa - Bpa	- Calcul du nombre - Calcul des largeurs	Effectue les calculs pour les passes
Pilier	- Npl - Bpl - Bc - Kc et Kpl	- Calcul du nombre - Calcul largeur - Calcul coefficients	Effectue les calculs pour les piliers et les contreforts

Lecture du tableau:

Init : Initialisation

La séparation des divers objets composant l'évacuateur de crues favorise la réutilisation, une certaine clarté et un entretien facile. La séparation de la classe seuil en deux sous-classes est faite pour assurer un meilleur entretien et un suivi aisé.

4.2.3 Axe Opérateur

La description du module Axe Opérateur est faite au chapitre 2, à titre d'exemple, au point 2.3.2. Les termes « Axe Opérateur » sont alors remplacés par « Gestion des opérations » pour simplifier la lecture. En résumé, l'éditeur d'axe et l'éditeur de graphique, tous deux indépendants l'un de l'autre, sont des sous-classes de la classe parente Axe Opérateur. Ils héritent de cette classe diverses opérations de sauvegarde et de récupération d'axes, des textes d'aide générale, et toutes les fonctions gérant l'interface et la gestion de données. Le tableau 4.3 présente un résumé des différentes opérations disponibles (méthodes) et des variables (attributs) dans ces classes.

Tableau 4.3 : Description du module Axe Opérateur

CLASSE	VARIABLES	METHODES	DESCRIPTION
Axe Opérateur		- Construction interface - Ecriture d'une aide - Fonction de sauvegarde - Fonction d'appel - Stockage de données	Gestion des éditeurs, Gestion de l'interface, Gestion des fichiers.
Editeur Axe	- Longueurs - Largeurs - Altitude - Chaînage - Elévation	- Prise de données - Vérifications - Corrections - Modification - Construction interface	Prise de données, Gestion des données, Production interface.
Editeur Graphique	- Point - Courbe	- Conception courbe - Affichage courbe - Affichage suivi	Construction des courbes, Permet la visualisation du profil du sol et des points

L'éditeur d'axe et de graphique est un objet en lui même, toutes les opérations et les données sont regroupées. L'interface très claire permet un suivi rapide. La réutilisation de ce module est limitée aux opérations de gestion des courbes et de fichiers.

4.2.4 Hydram Interface

Hydram Interface est le module qui gère toute l'interface du logiciel. Toutes les différentes fenêtres sont conçues dans cette partie du développement. Créer une interface utilisable et professionnelle demande beaucoup de fonctions d'écriture, de couleur, et de gestion. Il est donc préférable de rassembler la gestion et l'écriture des interfaces dans un même module afin d'éviter les répétitions inutiles et de favoriser la réutilisation. L'interface concerne la prise de données, l'affichage des résultats généraux et des résultats du canal d'amenée, la conception d'écran de présentation, de localisation de fichiers et de visualisation des valeurs par défaut. Toutes ces possibilités sont regroupées dans le même module Hydram Interface sous forme d'objets indépendants. Ils héritent tous des propriétés de la classe parente Axe Opérateur. La figure 4.4 présente l'organisation de ce module et les liens d'héritage qui s'y rattachent.

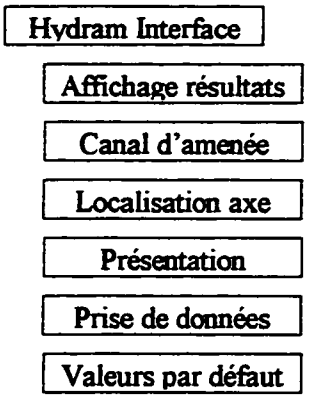


Figure 4.4 : Organisation du module Hydram Interface

Le tableau 4.4 présente un résumé des différentes opérations disponibles (méthodes) et des variables (attributs) dans ces classes.

Tableau 4.4 : Description du module Hydram Interface

CLASSE	METHODES	DESCRIPTION
Hydram Interface	- Construction écritures - Initialisation couleurs - Initialisation fenêtres - Initialisation menus - Initialisation imprimer - Initialisation variables de travail	Gestion des écritures, Gestion des couleurs, Gestion des fenêtres.
Canal Amenée	- Ouverture fenêtre - Lecture dans dictionnaire - Affichage valeurs - Sauvegarde, imprimer les valeurs	Affichage des résultats, Gestion des résultats, Gestion des menus, Validation et retour.
Localisation Axe	- Ouverture fenêtre - Lecture réponse - Ouverture fenêtre de recherche - Message à Opérateur Axe	Localisation de l'axe, enregistré.
Affichage Résultats	- Ouverture fenêtre - Lecture dans dictionnaire - Affichage valeurs - Sauvegarde, imprimer les valeurs	Affichage des résultats, Gestion des résultats, Gestion des menus, Reprise ou arrêt.
Présentation	- Ouverture fenêtre - stockage identification	Affichage message et prise d'identification
Prise données	- Ouverture fenêtre - lecture et stockage des données - Vérification des données - Aide à l'utilisateur	Affichage message, Prise données hydraulique avec vérification.
Valeurs Défaut	- Ouverture fenêtre - Lecture dans dictionnaire - Affichage valeurs - validation modification - enregistrement	Présentation des valeurs par défauts, Gestion des valeurs.

Le module d'interface est dédié au développement du logiciel. Il est possible de réutiliser quelques méthodes de la classe parente comme la gestion de l'écriture, des couleurs et des fenêtres, mais il est futile de pouvoir réutiliser davantage de ce module. Lors de la phase de programmation, les fenêtres seront personnalisées selon le modèle requis. Chaque interface a son propre cachet et ses propres caractéristiques. Les possibilités d'évolution et d'entretien d'un tel système sont toutefois garanties.

4.2.5 Librairies

Dans ce type de développement, la quantité de données manipulées et de résultats fournis est très dense. Beaucoup de valeurs peuvent être regroupées et exploitées ensuite par divers modules. Il est important de respecter la provenance et la destination de chaque groupe formé. Chacun d'entre eux a son utilité, ses caractéristiques et ses formats d'écriture et d'appel.

Il est donc nécessaire d'utiliser des outils pour aider à la gestion de ces valeurs. Plusieurs librairies ont ainsi été construites afin de faciliter ces manipulations.

Les données et les valeurs produites ou gérées par le logiciel appartiennent à différents groupes:

- **Valeurs par défaut**: Pour faciliter la conception des évacuateurs de crues, beaucoup de valeurs par défaut sont disponibles. Elles sont regroupées dans un dictionnaire de façon à pouvoir être utilisées à tout instant lors du cheminement. Les valeurs par défaut utilisées sont présentées au chapitre 3. Ces valeurs ont été posées pour le développement de HYDRAM. Il est donc peu probable que cette librairie puisse être réutilisée. Elle peut néanmoins être modifiée très facilement par l'utilisateur. L'utilisation de cette librairie est très simple. Chaque élément est identifié par son nom. Pour faire appel à l'un de ces éléments, il suffit de préciser son nom et la valeur correspondante est retournée.

➡ **Courbes**: Beaucoup de courbes sont utilisées tout au long de la conception. Ces figures permettent d'identifier des coefficients et de faire des vérifications sur les résultats obtenus. Elles proviennent, pour la plupart, de Design Of Small Dams (1987), et deux d'entre elles sont fournies par les experts du département hydraulique d'Hydro-Québec. L'annexe 3 présente les différentes courbes utilisées avec leurs origines. Toutes ces courbes ont dû être reproduites numériquement afin d'être utilisables par le logiciel. La librairie résultante présente une identification par courbe, et chacune de ces courbes est représentée par une multitude de points. L'utilisation de cette librairie est simple. Chaque élément est identifié par le nom de la courbe. Pour l'appeler, un message permet le retour d'une série de points représentant la courbe. Le module de gestion des ensembles de points permet de manipuler facilement les courbes et d'effectuer une multitudes d'opérations.

➡ **Formules**: La conception des évacuateurs de crues requiert le maniement d'une multitude de formules autant dans le domaine hydraulique que dans le domaine structural. Plusieurs types de formules sont nécessaire, des formules de base et modifiées pour la conception de ce genre d'ouvrages. Cette librairie de formules peut être utilisée par différents objets à tout moment du cheminement. La conception d'une telle librairie permet de rassembler toutes les formules en un même endroit et, ainsi, d'améliorer la clarté du logiciel, de faciliter l'entretien du programme, d'assurer l'évolution du système et de rendre possible la réutilisation de cette librairie de formules. Une description détaillée de cette librairie se trouve en annexe 4. L'appel des formules varie selon les variables en jeu. Il suffit de vérifier dans la formule les variables requises pour lui permettre de fournir un résultat.

➡ **Gestion des collections:** Au cours du cheminement pour la conception d'un évacuateur de crues, des ensembles de points ou de valeurs sont manipulés pour en

vérifier le contenu, obtenir des valeurs ou modifier certains éléments, et effectuer certaines opérations. Dans un langage orienté objet, ces ensembles se nomment collections ou dictionnaires. Il est important de développer un gestionnaire pour ces collections afin de réduire le temps de manipulation et de programmation. Les actions possibles sur un ensemble de données sont des opérations de lecture, d'interpolation, d'ajout, de retrait et de modification de valeurs, de recherche de toutes sortes et de transformation. Il est nécessaire de rassembler toute les opérations possibles dans une même librairie afin d'assurer un entretien et une évolution rapides du logiciel. Pour faire appel à ces différentes opérations de gestion, il faut vérifier le type de collection attendu, et les organiser en conséquence. Une fois le type reconnu, les opérations s'effectuent en fonction des variables requises.

- **Dictionnaire de travail et de résultats:** Afin de stocker les différents résultats de calcul et de permettre une plus grande rapidité dans le traitement des données, il est utile de créer des dictionnaires de travail et de résultats dans lesquels les différentes variables seront stockées momentanément durant les diverses phases de la conception. Cette collection n'est qu'un outil de travail et seule sa structure est réutilisable. Son existence permet au concepteur de faire un suivi constant sur les échanges de données et les résultats produits durant le cheminement.

4.3 Structure de raisonnement

Les différents objets échangent entre eux de nombreux messages. Pour obtenir une information ou un résultat de la part d'un objet, un message doit activer une de ses méthodes. Lorsque cette méthode est activée, l'opération requise se produit si tous les paramètres d'appel sont respectés. Afin d'éviter une programmation trop complexe, il est utile de produire un graphique pour visualiser les différentes interactions qui se produisent dans le logiciel. La figure 4.5 présente un résumé de ces échanges de messages.

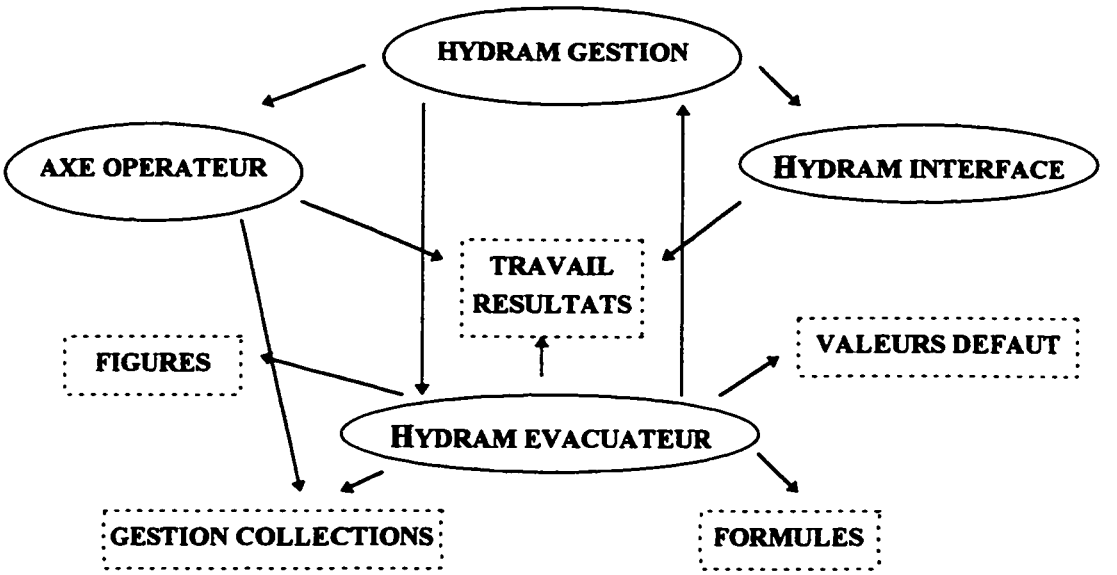


Figure 4.5 : Diagramme d'échange de messages

Légende:
- Librairies
- Module
- Envoi de message de la source au receveur

Un message est envoyé d'une source à un receveur pour demander le retour d'une valeur ou l'activation d'une méthode. Une fois le message accepté, l'opération s'effectue. Le sens des flèches indique la source du message et le receveur. Afin de respecter les notions d'entité et de relations entre les objets, il est nécessaire de suivre un tel diagramme d'échange de données. Sur cette figure, le module Hydram Évacuateur envoie des messages à toutes les librairies pour obtenir des informations, au module Hydram Gestion pour initialiser certaines valeurs...Le module Hydram Interface envoie des messages au dictionnaire Hydram Résultats pour obtenir certains résultats pour l'affichage. Ce ne sont que quelques exemples qui expliquent la lecture de la figure 4.5.

4.4 Illustration

Afin d'illustrer les notions orientées objet dans le contexte de la conception des évacuateurs de crues, plusieurs exemples sont présentés. Ils sont choisis pour représenter différents concepts orientés objet.

Les classes Editeur Axe et Editeur Graphique sont des sous-classes de leur parent Axe Opérateur. L'ensemble des lignes de programmation est présenté en annexe 5. Certains passages sont ici présentés à titre d'exemples.

4.4.1 Création d'un objet

En Smalltalk, tous les éléments sont des objets, comme une fenêtre, un dictionnaire ou un fichier. L'exemple suivant est une méthode qui permet le classement d'un dictionnaire de travail en ordre croissant selon le chaînage de points.

```
| collecTampon ele collecChainage taille taille2 valeurChainage
collecOrdonnee index axeTravailCopie |

collecChainage:= OrderedCollection new: self size.                    (1)
taille:= AxeTravail size.
AxeTravail do: [:element | valeurChainage:= ((element at:2) asNumber ).
                          collecChainage add: valeurChainage].        (2)
collecOrdonnee:=(collecChainage hydramCollectionOrdreCroissant ).
index:= 0.
ele:= nil.
collecTampon:= OrderedCollection new: self size.
taille2:= collecChainage size.
1 to:taille2 do:[:valeur |  [(collecOrdonnee at:valeur) = ele]
                              whileFalse:[index:= index+1.
                                          ele:=(collecChainage at:index)].
                            collecTampon add: index.
                            index:= 0 ].
axeTravailCopie:= OrderedCollection new:self size.
collecTampon do:[ :elem | axeTravailCopie add:(AxeTravail at:elem)].
AxeTravail removeAll.
axeTravailCopie do:[ :eleme | AxeTravail add:eleme ].
```

La variable *collecChainage* devient un objet lorsque elle est identifiée par *OrderedCollection new* à la ligne (1). *OrderedCollection* est une classe de l'environnement objet de Smalltalk qui permet de gérer différents ensembles de points. En ajoutant le terme *new*, la variable collecChainage est désormais un objet appartenant à cette classe, et hérite ainsi de toutes ses méthodes de manipulation de collections. Par exemple, à la ligne (2), l'objet *collecChainage* utilise l'opération *add:*, qui est une méthode appartenant à la classe *OrderedCollection.* Cette méthode permet l'ajout d'un item dans une collection.

4.4.2 Envoi de message

Plusieurs moyens sont disponibles pour envoyer un message dans le langage Smalltalk. Voici l'exemple d'une méthode permettant de corriger la largeur de l'axe d'évacuation:

```
| message valeurLargeur point|
message:= 'Opération de saisie en cours !'.
(enregistrement=false) ifTrue:[ self affichageMessage: message. ^nil ].
(ajout=true) ifTrue:[ ^nil ].
(modification=true) ifTrue:[^nil].
point:= (Prompter titre: 'Éditeur axe - Correction'
        prompt: 'Nouvelle largeur:'                    (3)
        valeurDefaut: '20').
(point=nil) ifTrue:[^nil].
valeurLargeur:= point asNumber.
 AxeDico at:'Largeur' put:valeurLargeur.
```

La ligne (3) présente un exemple pour l'envoi de message. *Prompter* est une classe qui gère l'affichage d'écrans d'informations ou de prises de données. Elle possède une méthode qui permet l'ouverture d'une petite fenêtre avec un champs pour entrer de l'information. Il suffit d'écrire le nom de la méthode avec les attributs nécessaires, précédés du nom de la classe. La figure 4.6 présente le résultat de cette simple requête.

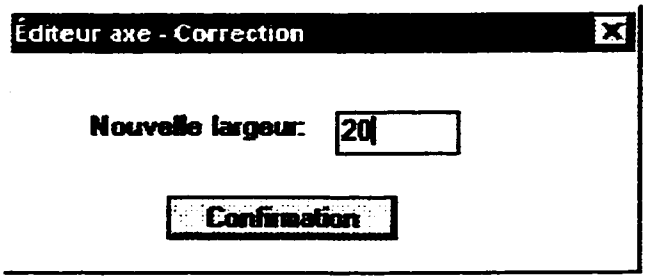


Figure 4.6 : Envoi de message

4.4.3 Écriture d'une méthode

La méthode présentée calcule la pente minimale de fond nécessaire pour le canal d'amenée variable. Elle appartient à la classe **Canal Amenée** qui gère les calculs du canal d'amenée. La formule non informatisée est la suivante:

$$\text{Pente minimale} = \frac{z1 - zDcau}{chDcau - ch1} * 100$$

Avec z1: cote du premier point de l'axe
zDcau: cote du début du canal d'amenée uniforme
chDcau: Chaînage du début du canal d'amenée uniforme
ch1: Chaînage du premier point de l'axe

La méthode devient:

```
penteFondMinimale
   " Retourne la pente minimale du fond du canal d'amenée. Cette pente relie
     le premier point de l'axe au début du canal d'amenée uniforme. ( % )      "

   | ch1 z1 zDcau chDcau pente |
   ch1:= ((ValeursPointsAxe at:1) at:2).
   z1  := (ValeursPointsAxe hydramElevationVsChainageMT: ch1).
   chDcau:= ((HydramTravail at:'Canal Alimentation') at:'chDcau').
   zDcau:= ((HydramTravail at:'Canal Alimentation') at:'zDcau').
   pente:=((((z1 - zDcau) / (chDcau - ch1)) * 100)).
   ^pente
```

L'écriture informatique de base reste sensiblement la même que l'écriture non informatisée. Lorsque cette méthode est appelée, elle prend les renseignements dans un dictionnaire de travail, et renvoie la valeur de la pente par le message *^pente*.

Les avantages de la modélisation objet n'est pas de simplifier l'écriture des équations de base, mais de proposer une organisation claire, et favorisant l'évolution du système et sa réutilisation.

4.4.4 Réutilisation

Un objet, une classe ou une méthode peut être réutilisable s'il est indépendant. La librairie de la gestion des collection en est un exemple. Voici l'exemple d'une méthode qui retourne la valeur minimale d'une collection et sa position:

```
hydramCollectionIndexValeurMin
   " retourne la valeur minimale d'une collection et son index "

 | final taille compte nombre comparaison position collecTampon |
  collecTampon:= OrderedCollection new:self size.
  taille:= self size.
  position:=1.
  compte:=0.
  final:=false.
  [final=false] whileTrue:[
     self do: [: valeur |    nombre:= (self at: position).
                              comparaison:= (nombre > valeur).
                              (comparaison=false) ifTrue:[compte:=compte + 1]].
        (compte=taille) ifTrue:[ final:= true.
                                 collecTampon add: nombre.
                                 collecTampon add:position.
                                 ^collecTampon]
                        ifFalse:[position:= (position + 1).
                                 compte:=0.
                                 final:=false]].
```

Lorsque que cette méthode est appelée, il faut que la source soit une collection. La méthode retourne un ensemble de deux nombres: la valeur minimale de l'ensemble et sa position.

4.5 Critiques

Idéalement, l'implémentation d'un développement ne doit s'effectuer qu'une fois la modélisation terminée. Pour modéliser de façon optimale, il est nécessaire d'avoir un aperçu des possibilités d'un langage orienté objet. Il est donc difficile d'obtenir une bonne modélisation lors du premier contact avec un environnement orienté objet. Je reste fidèle au développement de HYDRAM en orienté objet mais, pour un futur développement, des précisions seront apportées sur les divers liens d'héritage, le programme sera allégé en supprimant certaines variables, toutes les opérations de gestion seront séparées des différents modules pour permettre un entretien plus aisé, les liens entre les différents modules et objets seront clarifiés. La réutilisation sera accentuée en établissant des librairies de classes soignées et plus générales dans leurs modes d'appel.

Dans cette modélisation, le rapprochement entre les diverses classes aurait pu se faire différemment. En incluant le module de gestion générale dans le module Hydram Evacuateur, beaucoup de messages pourraient être supprimés pour alléger la structure. Par contre, la clarté de cette structure en serait diminuée. La classe permettant la conception du seuil est séparée entre le côté hydraulique et le côté structural. La partie hydraulique est en fait indissociable de la partie physique puisque ses résultats sont influencés par le profil du seuil. La liaison de ces deux objets serait préférable pour respecter le concept de réutilisation.

En ce qui concerne la réutilisation, tout le développement n'est, bien sûr, pas réutilisable. Seules certaines parties le sont entièrement, et certaines pourraient le devenir avec peu d'effort. Le manque d'expérience et de temps justifient ces lacunes dans la réutilisation des différents objets. La réutilisation demande une programmation très structurée et une modélisation complète et précise. Par exemple, les liens entre le dictionnaire de travail et la librairie des formules devraient être supprimés pour assurer une indépendance totale de cette librairie. De façon informatique, une librairie se crée en regroupant tous les

éléments, méthodes et dictionnaires, dans une même entité informatique appelée librairie. L'opération est ardue car il faut s'assurer de n'oublier aucun élément dont l'absence nuirait au comportement du reste de la librairie. Les objets spécifiques à la conception d'un évacuateur de crues ne pourront que très difficilement être réutilisés. Ils pourraient être utiles à un nouveau développement sur différents évacuateurs de crues ou sur des ouvrages similaires. Il est toutefois discutable de tout faire en fonction de la réutilisation. Le temps investi doit être justifiable.

La méthode de modélisation employée est une méthode proche de la méthode OMT (Rumbaugh 1991). Elle présente un modèle objet (lien entre classes et objets) et un modèle fonctionnel (circulation des messages et des données). Ces deux modèles développés sont moins structurés et moins détaillés que ceux de la méthode OMT. Il est difficile de faire une bonne modélisation lors d'un premier développement. De plus, les exemples de modélisation objet n'abondent pas dans la littérature et cela, pour toutes les méthodes disponibles.

CHAPITRE V
RESULTATS ET VALIDATION

La modélisation a été informatisée à l'aide du langage Smalltalk. Le logiciel obtenu comprend donc un éditeur d'axe, permet la conception d'un évacuateur de crues et d'un canal d'amenée. Des librairies de classes accompagnent le logiciel. Afin de présenter les résultats obtenus, de façon graphique et numérique, une conception va être faite à partir d'une étude d'avant projet développé par le département hydraulique de Hydro-Québec. Les différents modules du développement sont détaillés avec les données initiales et les résultats obtenus.

Les données utilisées dans cette étape de validation proviennent d'études préliminaires. Les valeurs hydrauliques et les caractéristiques du terrain utilisées ne peuvent être considérées comme des données exactes ou utilisables à d'autres fins que cette étude.

Afin de s'assurer du bon fonctionnement du logiciel orienté objet, une validation va être faite à partir de plusieurs projets déjà réalisés. Les valeurs obtenues sont comparées avec les valeurs retenues par les experts.

5.1 Présentation du site

Une étude d'avant projet a été réalisée sur le site de Grande Baleine. La carte du site du projet GB3 (Grande Baleine 3) est présentée en annexe 6. La conception d'un évacuateur de crues est au centre de cette étude. Les données de base correspondent aux valeurs suivantes:

Débit de conception:	1994 m^3/s	Largeur des piliers:	3.5 m
Débit de vérification:	2260 m^3/s	Nombre de contreforts:	2
Niveau maximal du réservoir:	389 m	Largeur des contreforts	1.75 m
Niveau d'eau aval pour le débit de conception:	354.5 m	Vitesse à la base du coursier:	18 m/s

Chaînage de l'évacuateur: 105 m

Le terrain naturel (TN) est la surface de terre au dessus de la couche de roc. La topographie de l'axe d'évacuation du site GB3 est présenté dans le tableau 5.1 .

Tableau 5.1 : Axe GB3

POINT	CHAINAGE	ELEVATION ROC	ELEVATION TN
1	0+000	378 m	382 m
2	0+050	380 m	384 m
3	0+100	384 m	387 m
4	0+250	378 m	380 m
5	0+400	372 m	374 m
6	0+630	351 m	351 m

5.2 Présentation des résultats

5.2.1 Editeur d'axe

La première étape du développement consiste à inclure l'axe d'évacuation dans le logiciel afin que le profil du sol soit enregistré. L'éditeur d'axe offre la possibilité à l'utilisateur de manipuler un nombre illimité de points et d'inclure la largeur de l'axe. Chaque point de l'axe est défini par ses valeurs de chaînage et d'élévation, et par les caractéristiques de la largeur de l'axe en ce point. Les valeurs peuvent être modifiées, des axes enregistrés peuvent être récupérés, et les axes étudiés peuvent être visualisés par l'éditeur graphique. Dans cet éditeur, les profils du roc et du terrain naturel sont dessinés, et les caractéristiques de chaque point peuvent être vérifiées. Les figures 5.1 et 5.2 présentent l'interface offerte au concepteur. Toutes les opérations de contrôles et de vérifications de données sont activées lors de l'ouverture de l'éditeur.

Lorsque la conception de l'axe d'évacuation est terminée, un ensemble de points est disponible pour la suite du développement. Cet ensemble contient la description de l'axe

d'évacuation et les caractéristiques de chacun de ses points. L'interface créée permet à l'utilisateur de progresser rapidement dans la conception.

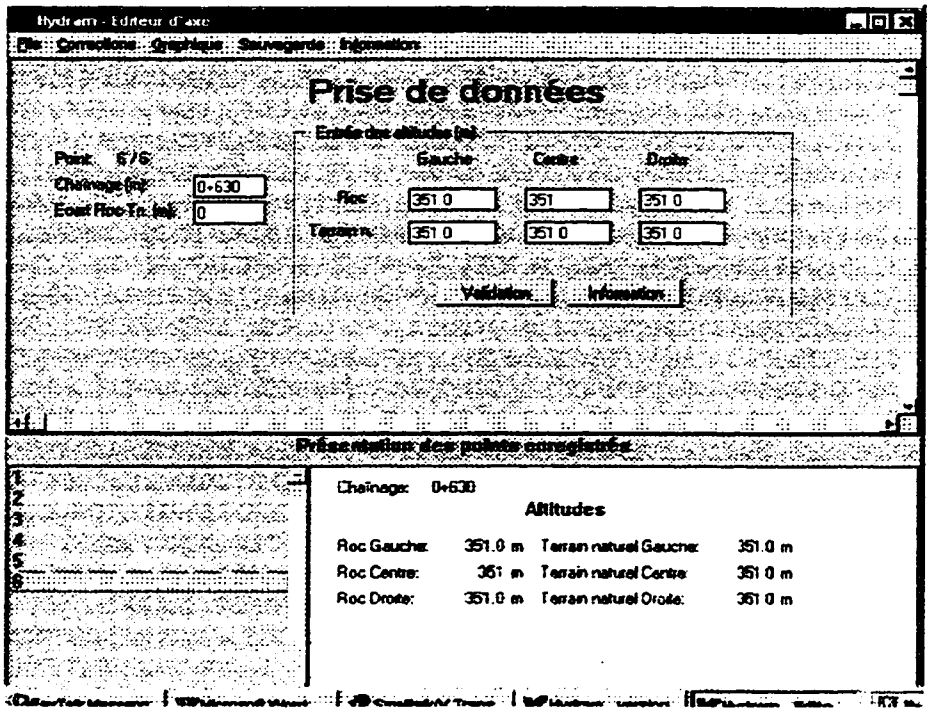


Figure 5.1 : Éditeur d'axe

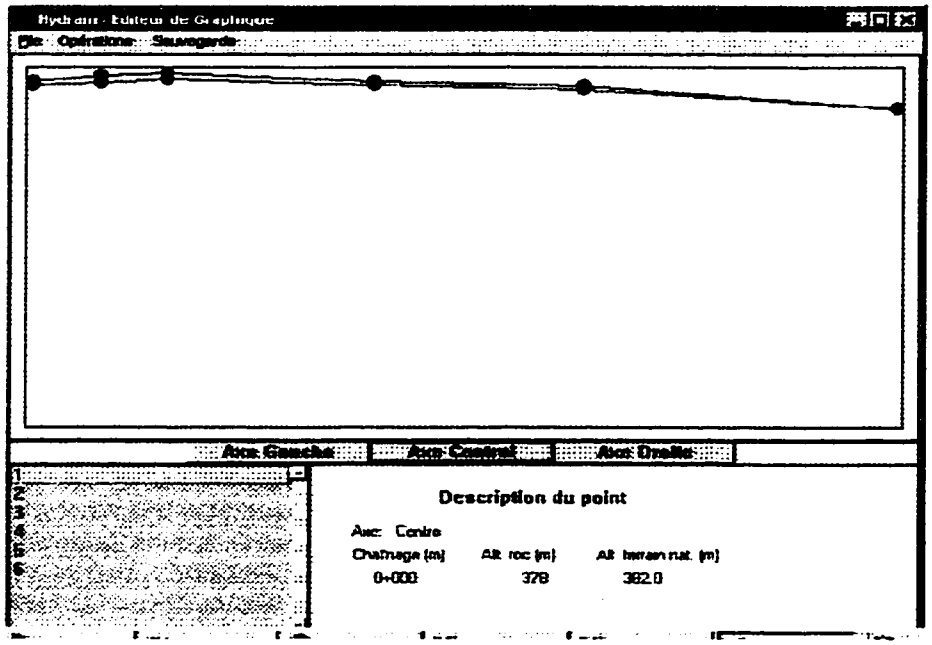


Figure 5.2 : Éditeur de graphique

5.2.2 Évacuateur de crues

Lorsque que la partie sur l'axe d'évacuation est complétée, l'entrée des données de base est requise. Ces valeurs vont permettre au logiciel de débuter les calculs de conception. La figure 3.5 du chapitre 3 présente la structure de raisonnement qui caractérise les différentes étapes de calcul suivies. La figure 5.3 présente la fenêtre de prise de données de base qui est offerte au concepteur.

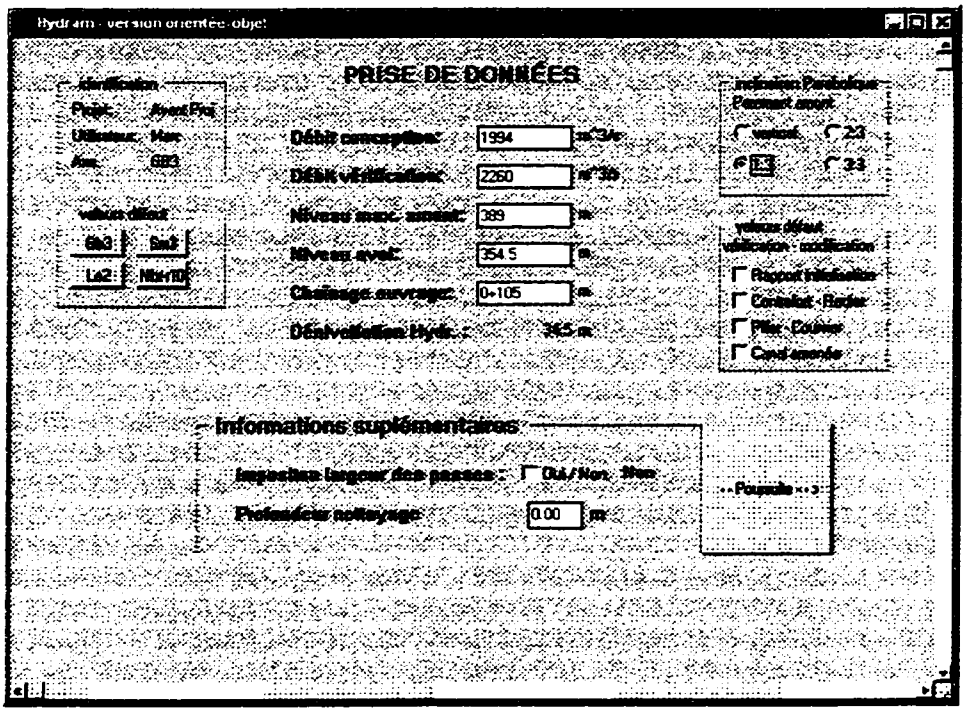


Figure 5.3 : Prise de données

La prise de données inclut l'inclinaison du parement amont, la vérification des valeurs par défaut utilisées et l'imposition de la largeur des passes si nécessaire. Une fois la validité des données confirmée, la conception des évacuateurs de crues débute.

Pour cet exemple, l'inclinaison du parement amont est fixé à 1:3, la largeur des passes n'est pas imposée, et les valeurs par défaut ne sont pas modifiées.

Après quelques secondes de calcul, une fenêtre présente l'ensemble des résultats à l'usager. Ces résultats sont divisés en plusieurs parties, à savoir le rappel des données initiales, l'ensemble des résultats de l'évacuateur de type parabolique et de type creager, et le choix des opérations disponibles. Ces opérations sont l'affichage des résultats pour le canal d'amenée, la possibilité de générer le profil complet du radier aval et la courbe de capacité. La figure 5.4 présente la fenêtre offerte au concepteur.

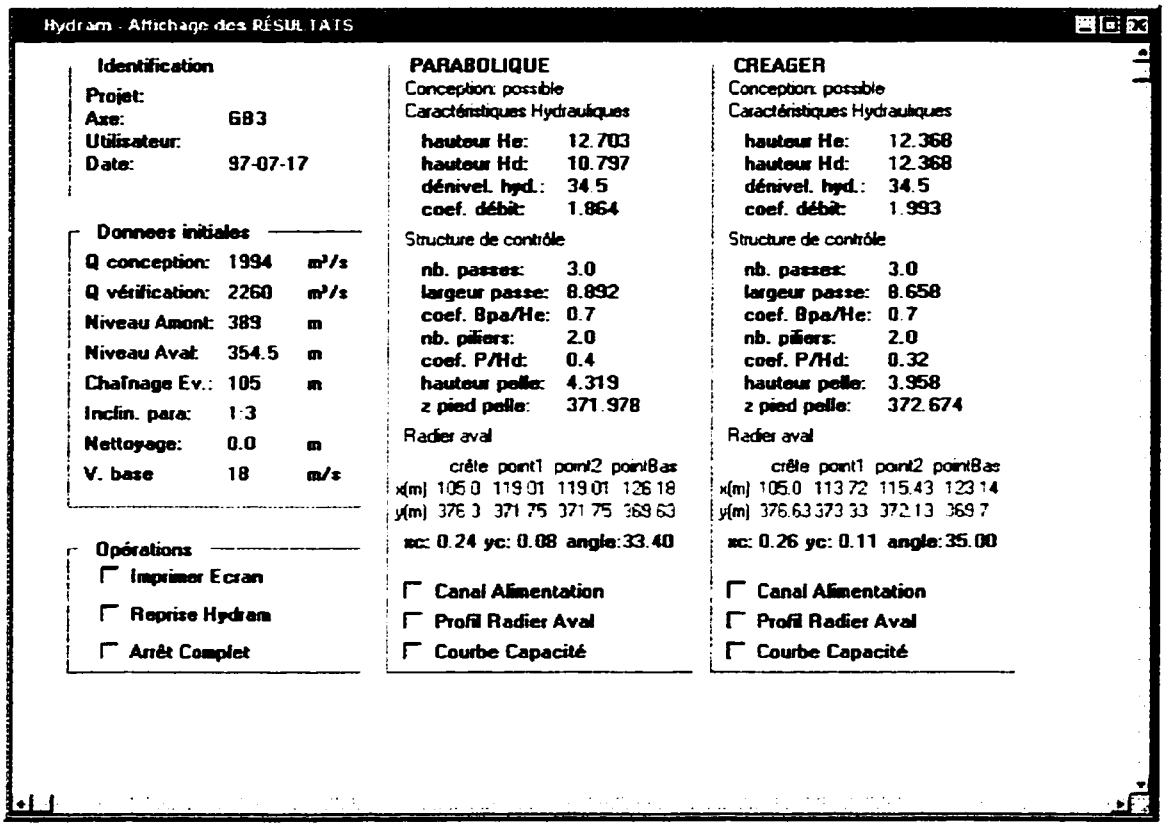


Figure 5.4 : Résultats de conception

Lorsque le profil du radier aval est activé, le logiciel génère tous les points du profil du radier aval, comprenant le seuil, le coursier et le pied du coursier. La figure 5.5 présente le profil obtenu pour l'évacuateur de type parabolique. De la même façon, lorsque la courbe de capacité est activée, le logiciel génère tous les points de cette courbe. La figure 5.5 présente la courbe obtenue pour l'évacuateur de type parabolique.

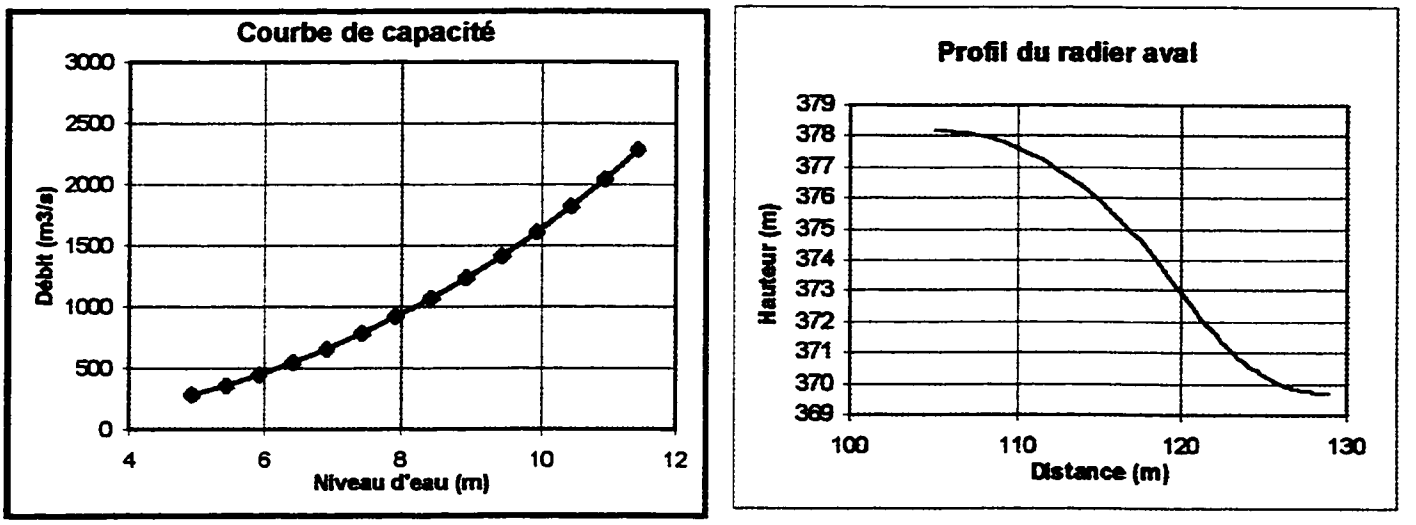


Figure 5.5 : Courbe de capacité et profil du radier aval

Les résultats fournis par le logiciel consistent ainsi en une série de valeurs identifiées et de courbes illustrant le profil de l'évacuateur et la courbe de capacité.

5.2.3 Le canal d'amené

Le canal d'amenée est un ouvrage annexe à l'évacuateur de crues. Ses règles de calculs sont indépendantes du reste de la conception de l'évacuateur de crues. Seules les coordonnées de son point de départ au bas du parement amont, altitude et chaînage, sont fournies lors de la conception de l'évacuateur de crues. La figure 5.6 présente les divers résultats qui accompagnent la fenêtre ouverte par l'usager. Ces valeurs représentent deux solutions possibles avec les longueurs et largeurs variables et uniformes, les chaînages et les altitudes des divers points, la pente latérale et la pente de fond, la vitesse de l'eau dans le canal... Deux alternatives sont présentées, l'une avec la pente minimale calculée, et l'autre avec la pente fournie par l'usager si celle ci est inférieure à la pente maximale permise.

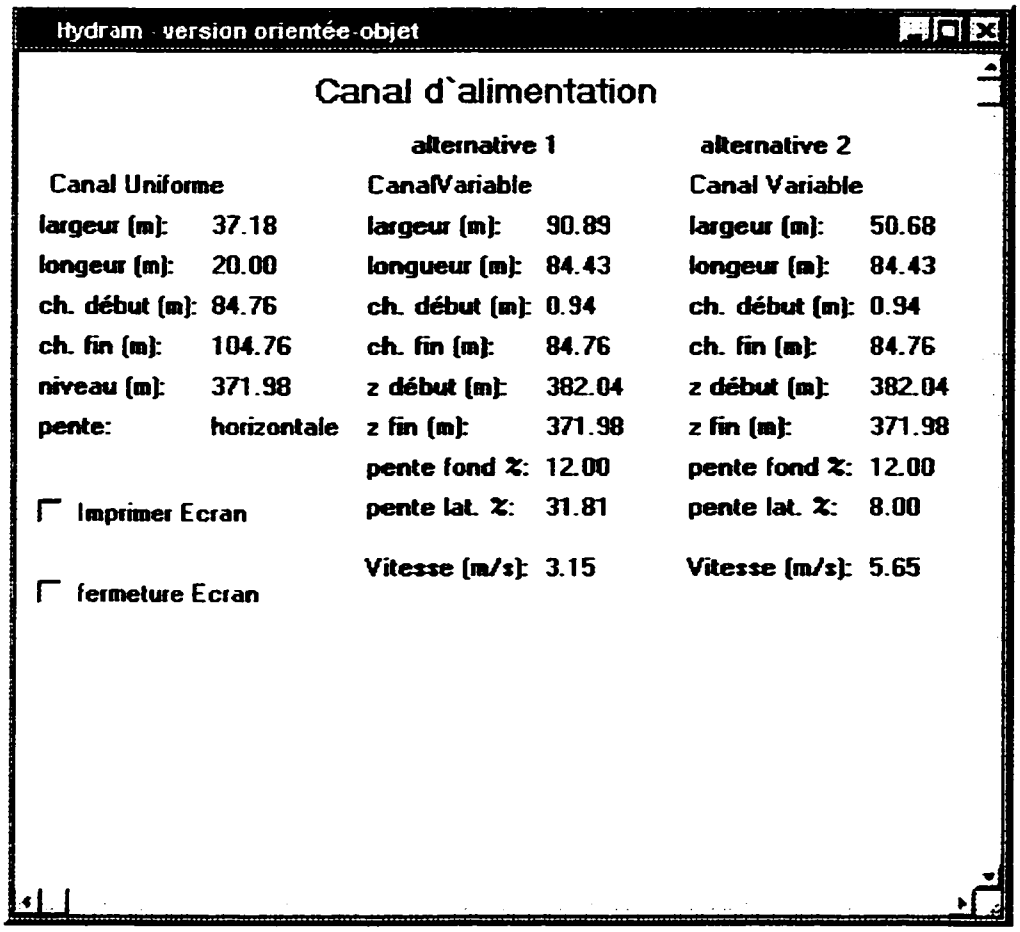


Figure 5.6 : Résultat du canal d'amenée

5.2.4 Librairies

La seule librairie que le concepteur peut consulter et modifier est le dictionnaire des valeurs par défaut. Chacune de ces valeurs a une influence sur la conception obtenue. Dans la partie validation, l'influence de certaines valeurs par défaut est vérifiée. La possibilité de les modifier rapidement facilite la vérification de la conception. La figure 5.7 présente la fenêtre des rapports d'initialisation. Ces rapports ne sont qu'une partie de la librairie des valeurs par défaut.

Le dictionnaire des formules et des courbes utilisées, ainsi que le gestionnaire des collections sont des librairies réservées à un informaticien.

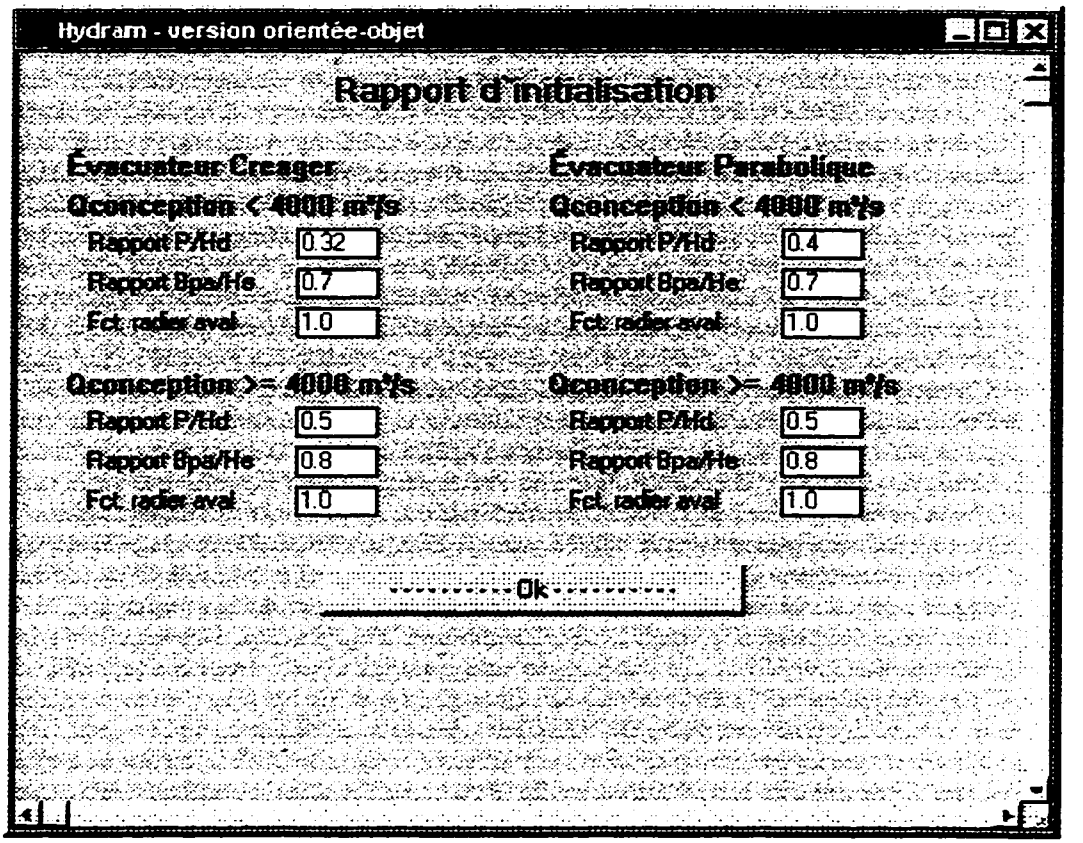


Figure 5.7 : Rapport d'initialisation

La présentation des différents écrans de prise de données et d'affichage de résultats permet également d'exposer les possibilités d'interface. Chacun de ces écrans est géré par le module Hydram Interface. L'initialisation des écritures, des couleurs, des boutons et des champs de prise de données se fait dans la classe parente. Les différentes fenêtres héritent de ces méthodes pour se personnaliser par la suite.

5.3 Validation du logiciel

Cette étape de validation concerne le logiciel orienté objet. Afin de vérifier la modélisation et la programmation, une série de simulations est effectuée. La validation a pour but de vérifier la concordance entre les règles à la base du logiciel et les résultats fournis par ce dernier. Quelques commentaires et explications sont apportés afin de justifier certains des résultats obtenus. Toutes les données utilisées lors des simulations proviennent de cas d'étude réels. Les sites retenus sont Sainte-Marguerite (SM3), et Grande Baleine (GB3).

La validation est faite en plusieurs étapes. Plusieurs facettes du logiciel et de son utilisation sont à considérer afin d'effectuer une validation complète. De son utilisation générale à l'évaluation des résultats obtenus, plusieurs simulations illustrent ces différentes facettes. Pour ce faire, toutes les ressources du logiciel seront exploitées.

5.3.1 Validation des résultats

Pour valider les résultats produits par le logiciel, les résultats obtenus sont comparés avec les résultats obtenus par le développement HYDRAM en Prolog et les résultats provenant des experts pour certains cas. Le premier cas de comparaison est l'avant projet GB3. Le tableau 5.3 présente les résultats de la conception d'un évacuateur de crues pour le site étudié. Tous les résultats obtenus ne sont pas dans ce tableau. Seuls les plus importants sont utiles pour cette étape de validation.

Les résultats de la conception préliminaire d'un évacuateur de crues sur les sites GB-3 et SM3 sont présentés. Les valeurs proviennent du département hydraulique d'Hydro-Québec.

Résultats des experts, site : **GB-3**

Débit de conception (m^3/s):	1994
Nombre de passes :	3
Hauteur de charge (m) :	12.5
Largeur des passes (m) :	9
Cote du seuil (m) :	377.7
Cote à la sortie du coursier (m) :	367.4
Longueur canal d'amenée (m) :	100
Largeur canal d'amenée (m) :	37.5
Pente fond canal d'amenée (%) :	12
Cote du radier amont (m) :	372

Tableau 5.3 : Validation pour le site GB-3

	HYDRAM OBJET		HYDRAM PROLOG	
Site: GB-3	**parabolique**	**creager**	**parabolique**	**creager**
	HYDRAULIQUE ET STRUCTURE			
Charge He (m)	12.70	12.36	12.6	12.4
Hauteur Hd (m)	10.79	12.36	10.7	12.4
Coeff de débit	1.86	1.99	1.9	1.99
Nombre passes	3	3	3	3
Largeur Passe (m)	8.9	8.65	8.8	8.7
Nombre piliers	2	2	2	2
Hauteur Pelle (m)	4.31	3.95	4.3	4
Elév pied pelle (m)	372	372.6	372.1	372.6
	RADIER AVAL			
Chaînage crête (m)	105	105	105	105
Elévation crête (m)	376.3	376.6	376.4	376.6
Ch bas coursier (m)	126.2	123.14	129.7	125.4
Elev bas coursier (m)	369.6	369.7	368.87	368.81
	CANAL D'AMENEE			
Largeur uniforme (m)	37.2	36.5	36.9	36.6
Longueur uniforme (m)	20	20	20	20
Largeur variable (m)	90.89	87.7	85.2	86.8
Longueur variable (m)	84.4	80	80	79
Pente fond (%)	12	12	12	12
Pente latérale (%)	31.8	32	30.2	31.8

Il est simple de constater que les résultats du développement objet sont identiques à ceux de Hydram Prolog. Les caractéristiques hydrauliques et structurales sont les mêmes pour

les deux développements. Une différence maximale de 10 cm est observée au niveau de la charge He au dessus du seuil. La balance des écarts se limite à quelques centimètres. Ces maigres différences s'expliquent par deux causes principales. La première concerne le nombre de chiffres considérés après une virgule pour une valeur. Le développement orienté objet conserve tous les chiffres après la virgule, tandis que le développement Hydram n'en conserve que deux. Cette différence est insignifiante sur quelques opérations, mais pour une conception, une multitude de calculs et d'itérations est requise. Une différence de quelques centimètres est donc possible. La deuxième raison est la lecture des coefficients sur les figures. L'informatisation de ces figures ne peut être aussi précise que les originales. Selon la reproduction produite, des imprécisions sont possibles.

La conception du radier aval présente des différences plus fortes. Les caractéristiques de la crête sont identiques, tandis que les points bas du coursier sont séparés par quelques 0.7 mètres. Le point bas du coursier est identifié après la conception d'une parabole représentant le seuil, d'une droite pour le coursier, et d'un arc de cercle représentant le pied du coursier. Le grand nombre de calcul engendre une certaine différence dans les résultats. L'écart restant est dû à une différence dans le calcul des divers points de tangence entre les courbes. Cette différence se répercute jusqu'aux caractéristiques du point bas du coursier.

Il n'y a pas de conception unique pour le canal d'amenée. Les largeurs dépendent de la pente latérale, et les longueurs dépendent de la pente du fond. Plusieurs pentes de fond sont possibles selon les points de l'axe. Dans le développement orienté objet, plusieurs alternatives sont offertes à l'usager. Les différences constatées ne sont donc pas des erreurs, mais sont des alternatives valables mais différentes.

L'avant projet du site SM3 est utilisé pour la deuxième validation. Les données de départ sont indiquées dans le tableau 5.4 .

Tableau 5.4 : Données de base - Site SM3

Point	Chaînage(m)	Elev roc (m)	Elev TN (m)
1	0+025	410	411
2	0+075	424	425
3	0+102	435	436
4	0+135	424	425
5	0+149	408	409
6	0+200	400	401

Q conception (m3/s)	2615
Q vérification (m3/s)	2871
Niveau amont (m)	428.7
Niveau aval (m)	415.9
Chaînage eva. (m)	0+106

Tableau 5.5 : Validation pour le site SM3

	HYDRAM OBJET		HYDRAM PROLOG	
Site: SM3	**parabolique**	**creager**	**parabolique**	**creager**
	HYDRAULIQUE ET STRUCTURE			
Charge He (m)	14.15	13.78	14.1	13.8
Hauteur Hd (m)	12	13.78	12	13.8
Coeff de débit	1.86	1.99	1.9	1.99
Nombre passes	3	3	3	3
Largeur Passe (m)	9.91	9.64	9.9	9.7
Nombre piliers	2	2	2	2
Hauteur Pelle (m)	4.8	4.41	4.8	4.4
Elév pied pelle (m)	409.7	410.5	409.8	410.5
	RADIER AVAL			
Chaînage crête (m)	106	106	106	106
Elévation crête (m)	414.6	414.9	414.6	414.9
Ch bas coursier (m)	127.6	124.3	130	127.2
Elev bas coursier (m)	408.7	408.7	407.9	407.92
	CANAL D'AMENEE			
Largeur uniforme (m)	40.2	39.45	40.2	39.6
Longueur uniforme (m)	20	20	20	20
Largeur variable (m)	56.61	55.45	54.6	54.4
Longueur variable (m)	46	44.07	41.5	41.5
Pente fond (%)	12	12	12	12
Pente latérale (%)	17.7	18.15	17.3	18

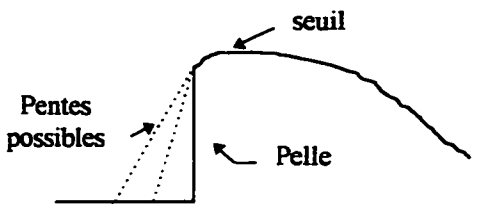
La comparaison des deux développements pour le site SM3 est identique à celle portée pour le site GB3. Les variations sont similaires et les explications le sont aussi.

5.3.2 Vérification du comportement

Le logiciel orienté objet offre la possibilité de faire varier plusieurs données autres que les valeurs de base (débit, niveau, valeur de base...). Ces valeurs sont l'inclinaison du parement amont et l'imposition de la largeur des passes. La modification d'une de ces valeurs influence la conception de l'évacuateur. Afin de vérifier la programmation, et d'évaluer l'influence respective de chacun de ces paramètres, plusieurs simulations sont faites et les résultats sont présentés.

➡ **Influence de l'inclinaison du parement amont:**

Le parement amont est aussi appelé « pelle ». C'est la partie de l'évacuateur situé à l'extrémité amont du radier. La figure jointe présente le profil d'un évacuateur. Pour des raisons économiques, quatre pentes sont possibles, à savoir la position verticale pour un évacuateur de type creager, et une pente 1:3, 2:3, et 3:3 pour un évacuateur de type parabolique. Le choix d'une pente est fait par l'utilisateur au moment de la prise de données. En fonction des graphiques disponibles dans DSD (Design of Small Dams, 1987), seul l'évacuateur de type parabolique peut être influencé par une variation de la pente du parement amont.



La valeur du coefficient de débit varie en fonction de cette pente. La courbe 9.25 de DSD présente les corrections à apporter en fonction du parement amont d'un évacuateur de type parabolique. Par exemple, le facteur correctif sera plus important pour une pente 2:3 qu'avec une pente 1:3. Le coefficient de débit se retrouve ainsi augmenté, et est donc accompagné d'une diminution de la hauteur de charge au dessus du seuil. Des répercussions sont également notables au niveau du profil du radier aval. L'équation de la

courbe du seuil est de la forme: $\frac{Y}{H_D} = K * \left(\frac{X}{H_D}\right)^N$. Les coefficients N et K dépendent de l'inclinaison du parement amont.

En résumé, une modification de l'inclinaison du parement amont affecte divers coefficients, et donc une partie de la conception de l'évacuateur de crues. Il est dès à présent possible de prédire de faibles modifications. Les coefficients en jeu ne varient que de quelques centièmes. À l'aide d'une simulation avec le logiciel orienté objet, voici les valeurs obtenues:

Site GB-3

Inclinaison :	**1:3**	**2:3**
Coefficient de débit :	1.864	1.88
Hauteur de charge (m) :	12.70	12.64
Hauteur d'eau (m) :	10.79	10.74
Largeur passe (m) :	8.89	8.85
Hauteur pelle (m) :	4.32	4.3
Cote pied de la pelle (m) :	371.97	372.05

La tendance prévue est respectée. Les différences sont de l'ordre de quelques centimètres, avec un maximum de 8 cm pour la hauteur de pelle. Cet écart est faible, mais il est toutefois intéressant de noter l'influence de l'inclinaison du parement amont.

➡ **Imposition de la largeur des passes:**

La largeur des passes se calcule automatiquement à partir de la hauteur de charge He et du rapport d'initialisation Bpa/He. Selon les caractéristiques du terrain et les équipements

en place, il est parfois nécessaire de fixer une largeur de passe avant de débuter la conception.

Le débit est fixé avant de débuter les calculs. Si une largeur de passe est imposée plus étroite que la largeur requise, la hauteur d'eau sur la crête va augmenter pour permettre le passage du même débit. Le nombre de passe ne peut être qu'un chiffre pair. Ainsi, si les largeurs de passes sont réduites jusqu'à une certaine limite, le nombre de ces passes sera normalement augmenté, et la hauteur d'eau diminuera. La simulation suivante présente ce cas.

	Largeur	**Largeur**
Site GB-3	**Non imposée**	**imposée**
Débit de conception (m^3/s)	1994	1994
Largeur de passe (m)	**8.89**	**5**
Hauteur de charge (m) :	12.7	13.56
Hauteur d'eau (m) :	10.79	11.52
Hauteur pelle (m) :	4.31	6.16
Cote pied de la pelle (m) :	371.97	369.27
Nombre de passes :	3	5
Coefficient de débit :	1.86	1.88

Ces différentes évaluations du logiciel orienté objet produit confirment la validité de son développement. Les résultats obtenus selon les cas sont validés par le logiciel Hydram en Prolog. Le chapitre 6 présente une comparaison détaillée entre les deux développements produits.

CHAPITRE VI
ÉTUDE COMPARATIVE ENTRE DEUX DEVELOPPEMENTS PARALLELES

Afin de mener à bien une comparaison entre deux développements informatiques parallèles, il est nécessaire d'établir diverses phases de comparaison. Ces différentes phases englobent tous les aspects du développement, de la modélisation initiale à l'exploitation finale du produit obtenu. Quatre domaines de comparaison sont retenus, tels que la modélisation ou le développement, la programmation, l'optimisation, et l'exploitation ou la réutilisation.

Quelque soit le langage informatique employé, tout développement doit transité par ces différents domaines. Selon les langages informatiques utilisés, les différences dans le cheminement seront plus prononcées dans un certain domaine que dans un autre. Ces différences peuvent se situer au niveau de la syntaxe, de la structure, de la modélisation des connaissances, ou au niveau des concepts suivis par le langage. Afin d'effectuer une comparaison complète, il est nécessaire d'envisager tous les cas de figures. Pour chacun des domaines envisagés, des critères regroupant l'ensemble des points de comparaison sont posés et servent de base à cette étude. Elle repose donc sur le concept informatique suivie par le langage, mais également sur le langage informatique retenu, en l'occurrence TurboProlog et Smalltalk/V pour win32. Une trop grande généralisation sur tous les langages informatiques dépasserait notre domaine d'expertise.

Les critères de comparaison vont dans un premier temps être explicités, et seront suivis par la comparaison proprement dite.

6.1 Critères de comparaison

6.1.1 Modélisation - Développement

Cette étape regroupe tout le processus de modélisation des règles de connaissance. Elle peut varier selon la philosophie des divers langages: un langage dit « procédural » nécessite une modélisation différente d'un langage dit « objet ». La façon de modéliser le développement est propre à tout langage. Pour des langages optant pour une philosophie identique, les moyens de modélisation sont assez similaires. Pour des langages aux concepts différents (Procédural vs Objet), les moyens employés varient généralement.

Cette étape est importante lors du développement d'une quelconque étude. La structure de raisonnement se doit d'être posée de façon précise, l'utilisation et la gestion de divers modules ou dictionnaires doivent être envisagées. Il faut donc définir la structure interne du logiciel dans sa globalité, et dans toutes ses spécificités. Pour prendre en compte l'ensemble de la modélisation, nous nous proposons de suivre une liste de points de comparaison:

- ➡ Mise en place de la structure de raisonnement,
- ➡ Modélisation des règles de calculs,
- ➡ Modélisation de la structure du logiciel:
- ➡ Temps investi dans cette étape de modélisation.

La structure de raisonnement englobe toutes les règles de connaissance développées par M. Benoît Robert et des experts de Hydro-Québec. Cette structure est formée d'une multitude de règles, et doit donc être mise en place de façon à pouvoir être exploitable de façon simple et rapide par le logiciel. Cette structure fournie le cheminement nécessaire à la conception d'un évacuateur de crues. Elle est la base d'un système expert.

Les règles de calculs sont une partie des règles de connaissances. Elles sont directement alimentées par des équations hydrauliques et mathématiques utiles à la conception d'un évacuateur de crues. Ces équations sont nombreuses et variées. Leurs utilisations sont fréquentes et répétitives, il est donc utile de prévoir un mode de gestion de moyens de calculs.

La structure interne du logiciel regroupe l'ensemble des moyens requis pour mener à bien la conception, mais également divers modules permettant la gestion de ces moyens. La modélisation de la structure interne se doit de respecter les objectifs du développement. Les composantes de cette structure sont variées, la liste suivante présente les points de comparaison qui sont proposés:

- Partition et gestion des modules,
- Utilisation de dictionnaires dits «par défaut»,
- Utilisation et gestion des équations,
- Utilisation et gestion des figures,
- Gestion et stockage de données,
- Utilisation et gestion de collections de points.

La durée de cette étape est directement reliée au temps investi dans la mise en place de la structure de raisonnement et dans la modélisation de la structure interne du logiciel. Les règles de calculs constituent une base utilisable par les deux développements. La connaissance initiale des moyens de modélisation est un facteur de temps qui n'est pas négligé.

6.1.2 Programmation

Tous les points de comparaison liés à l'utilisation du langage informatique sont groupés dans l'étape de programmation. Cette étape qui suit la phase de modélisation ne se limite pas au codage des divers composants du logiciel. Elle englobe toutes les spécificités du

langage et les utilisations possibles. Cette utilisation est caractérisée par plusieurs éléments qui sont évalués:

- Langage informatique,
- Retour, correction,
- Manipulation structurale.

Plusieurs éléments du langage informatique doivent être évalués:

- Syntaxe,
- Maniement et utilisation,
- Programmation et écriture,
- Aide interne et détection d'erreurs.

Les moyens mis à la disposition de l'usager varient d'un langage informatique à un autre. Il devient donc utile de s'attarder sur ces moyens et sur les retombées, notamment en terme de gain de temps. La syntaxe n'offre que peu d'intérêt dès qu'elle est maîtrisée par le programmeur. En revanche, les possibilités offertes par le langage peuvent être d'une grande utilité. Ses possibilités se trouvent dans le maniement, l'aide offerte, et la clarté. Lors du codage d'un développement, le programme est sujet à de multiples retouches et corrections. Il s'agit d'une étape inévitable qui se doit d'être soutenue par une structure informatique adéquate. Les moyens mis à la disposition du programmeur sont donc évalués.

Les manipulations structurales sont les modifications possibles de la structure informatique de base. Afin d'alléger ou d'améliorer le programme, de telles manipulations peuvent être nécessaires. Encore une fois, les moyens disponibles seront évalués.

Le temps attribué à cette étape de programmation dépend de plusieurs facteurs:

- Modélisation initiale: précision et modification,

➡ Connaissance initiale du langage et de ses possibilités.

Cette étape de programmation est en constante évolution. Il est donc utile de vérifier les moyens mis à la disposition d'un programmeur afin de satisfaire à ces exigences. L'utilité de s'attarder à une telle étape est ainsi nécessaire. Les moyens de réutilisation sont abordés dans une section ultérieure.

6.1.3 Optimisation

Lorsque l'étape de modélisation et de programmation aboutissent à la conception d'un logiciel, une phase d'optimisation débute. C'est au cours de cette phase qu'une validation du logiciel survient, et que le langage utilisé est mis à l'épreuve face aux modifications et corrections éventuelles. Les possibilités évaluées pour chacun des langages informatiques sont celles fournies avec la version utilisée lors de la programmation, et non celles provenant des versions ultérieures. Les points de comparaisons sont variés pour effectuer une étude pertinente:

➡ Résultats numériques,
➡ Aide et indications offertes,
➡ Vitesse d'exécution,
➡ Interface,
➡ Maniabilité de la structure,
➡ Modifications éventuelles.

Les résultats numériques sont les valeurs de base qui sont recherchées et se doivent d'être exacts. Les items suivants concernent les possibilités que le langage employé offre au programmeur. L'aide fournie et l'interface représentent les possibilités offertes par les langages, mais non pas les résultats présentés à l'écran. La vitesse d'exécution ne dépend pas que de la lourdeur de la structure informatique. Cette vitesse est directement proportionnelle à la capacité de l'ordinateur sur lequel le programme est exécuté. Il est

donc difficile d'évaluer la performance en terme de vitesse. Toutefois, une comparaison succincte est effectuée.

La maniabilité de la structure et la possibilité d'effectuer des modifications éventuelles sont deux éléments d'importance dans cette phase d'exploitation. Ces deux éléments constituent les points de comparaisons d'importance. La structure informatique, telle que fixée après l'étape de modélisation peut être sujette à des modifications. Sa maniabilité se doit donc d'être mise à l'épreuve. Les moyens offerts pour y parvenir sont évalués et comparés. La maniabilité d'une structure en facilite les modifications éventuelles.

6.1.4 Exploitation - Réutilisation

Une dernière étape est indispensable afin de compléter la comparaison entre les deux développements. Lorsque l'optimisation est achevée, le logiciel est normalement terminé. Il faut toutefois pousser notre étude au delà. Un logiciel n'est pas un produit final en soi. Au fil du temps, les besoins évoluent, des modifications et l'apport de complémentarités deviennent nécessaires. La façon dont la modélisation et la programmation sont faites influe ainsi sur les développements futurs. Une comparaison sur l'exploitation éventuelle du logiciel est effectuée. Elle porte sur deux points:

- Opérations sur le logiciel,
- Réutilisation sur de nouveaux développements.

La réutilisation est un concept exploité par les concepts orientés objet. Elle permet de scinder le programme en Objets, ou modules, directement réutilisables pour des nouveaux développements. À l'aide des deux développements parallèles, une vérification sera effectuée sur ce concept de réutilisation, et sur son impact.

Cette partie de l'étude fait référence à l'ensemble du développement effectué.

6.2 Développement HYDRAM en Turbo Prolog

Le développement orienté objet est conçu à partir de la base de connaissances développée pour le projet HYDRAM. La programmation initiale du projet HYDRAM a été faite à l'aide du langage Turbo Prolog. Des exemples de cette étape de programmation sont donnés dans le rapport « HYDRAM, un système à base de connaissances pour la conception des évacuateurs de crues (Juin, 1996) ».

6.2.1 Langage et environnement Prolog

Prolog est un langage de programmation d'intelligence artificielle basé sur l'utilisation de règles de production. Il est caractérisé par une programmation dite logique. Initialement, la partie théorique fut développée par Robert Kowalski, l'expérimentation par Maarten van Emden, et l'implémentation par Alain Colmerauer à Marseilles (France) au début des années 70 (Bratko, 1990). Prolog est un langage centré autour de quelques mécanismes de base, comme la structuration de données et le *backtracking*, ou la manipulation de boucles. Prolog est reconnu comme une langage puissant pour l'intelligence artificielle et les développements non numériques en général.

L'environnement de programmation avec Prolog est différent des modèles connus procéduraux comme ceux de Pascal ou de C. Dans Prolog, une suite de faits constitue la base de connaissances du développement. Tout au long du développement, une série d'instanciations de variables permet d'effectuer les opérations prévues. La compréhension est facilitée par une écriture simple et facilement identifiable.

6.2.2 Modélisation

Le développement HYDRAM en Prolog est très fidèle à la structure de raisonnement initiale. La figure 3.5 dans le chapitre 3 présente une partie de cette structure, celle concernant la conception des évacuateurs de crues en haute chute. Le cheminement présenté est complet et est directement programmable. Différents modules sont isolés, tels que la conception des évacuateurs, la conception des dissipateurs, et un éditeur d'axe permettant l'enregistrement d'axe d'évacuation. Ces différents éléments sont séparés lors de la programmation et de l'exécution du logiciel mais sont dépendants les uns des autres par rapport à certaines valeurs de départ et à des ensembles de points. Par exemple, la conception des dissipateurs requiert les dimensions et les résultats de quelques valeurs de l'évacuateurs de crues et la conception de l'évacuateur utilise les données de l'axe d'évacuation provenant de l'éditeur d'axe.

Certaines valeurs sont représentées par des variables dites générales. Ces variables constituent la base de connaissances et sont représentées par des faits. Par exemple, différentes valeurs par défaut et l'ensemble des figures utilisées sont ainsi représentées. Certaines formules sont également accessibles par l'ensemble des règles de calcul.

La structure de raisonnement constitue la base de la modélisation. La modélisation est simple et suit ce cheminement. Lorsqu'une requête est faite, comme la recherche du coefficient de débit, les règles de connaissances reliées à cette étape sont lues de façon informatique et, lorsque les conditions d'application sont respectées, les conclusions sont enregistrées et le cheminement se poursuit. La figure 6.1 généralise cet exemple. Il n'y a donc pas de place précise pour les règles de connaissance. Elles sont toutes réparties dans le programme et elles sont parcourues lorsque la requête par une autre règle est faite.

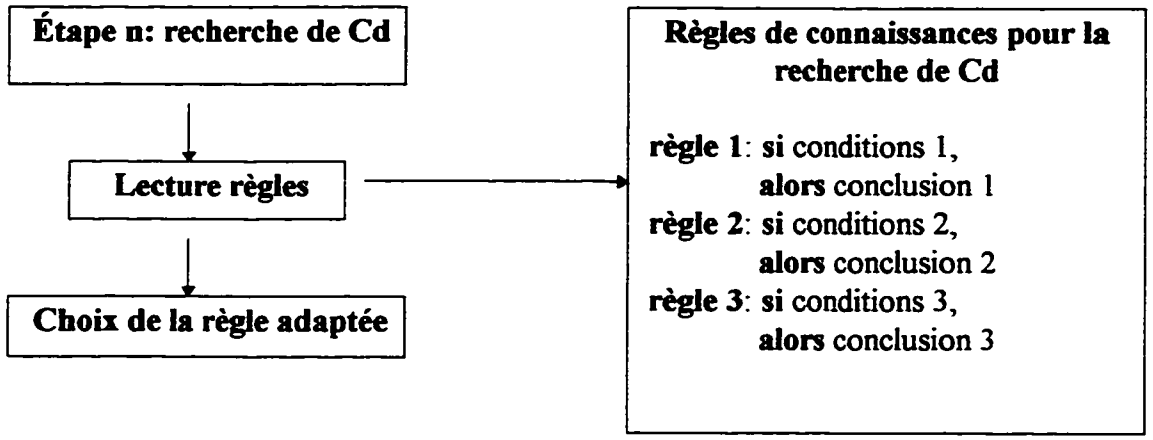


Figure 6.1: Lecture des règles de connaissances

Les équations sont incluses dans les règles de connaissances. Quelques formules sont généralisées comme le calcul de pentes. Les données circulent selon les opérations. Les ensembles de points ou de données sont informatisés de façon à être accessibles pour l'ensemble du développement.

Le temps investit dans l'étape de modélisation est faible. La répartition des règles de connaissances se fait lors de la programmation. Pour un environnement comme celui de prolog, une fois la structure de raisonnement établie de façon complète et fiable, la modélisation est presque inexistante.

6.2.3 Programmation

Le langage informatique utilisé présente une syntaxe simple et identifiable rapidement. L'utilisation et le maniement sont plus délicats. Prolog n'est pas un langage procédural classique. La programmation de sa base de faits, l'instanciation des variables et les retours en arrière possibles en font un environnement particulier auquel un programmateur débutant doit s'habituer. Une fois ces particularités apprivoisées, le maniement devient simple et logique. L'aide interne pour la détection d'erreurs est puissante et accessible.

Chaque fait est une sorte d'entité informatique et est donc facilement manipulable. Le temps investi dans cette étape est un temps normal de programmation selon l'expérience du programmeur.

6.2.4 Optimisation

Les résultats du logiciel prolog répondent aux attentes des clients. Bien que n'étant pas reconnus pour ses applications numériques, les divers calculs et itérations produisent des résultats fiables et se font très rapidement. Les indications fournies à l'utilisateur permettent un suivi du cheminement de conception, les différents résultats sont présentés à l'écran et sont très facilement imprimables. Les modifications éventuelles à apporter dans la programmation se font assez simplement pour le concepteur. Pour un informaticien externe à la conception et au logiciel, toutes modifications restent plus délicates. L'identification de chaque règle et sa description lui sont utiles avant d'effectuer une quelconque recherche. À partir de là, les modifications mineures sont possibles et assez rapides.

6.2.5 Exploitation - Réutilisation

Au cours du projet HYDRAM, des modifications majeures ont été faites sur le logiciel produit en Prolog avec succès. Des modifications majeures sont donc possibles. Le temps nécessaire à ces travaux dépend de l'importance de ceux-ci, mais semble être normal.

Les principes de réutilisation sont possibles. Les courbes enregistrées et les prédicats généraux peuvent être réutilisés sans modifications dans de nouveaux développements. La répartition aléatoire de la plupart des règles de connaissance rend difficile le regroupement et la réutilisation. Pour qu'un élément respecte les principes de réutilisation, il faut que sa structure et ses modes d'appels soient assez généraux. À l'aide

de plusieurs modifications, il est toutefois possibles d'utiliser une partie de ce développement pour de futurs projets dans le même domaine.

6.3 Comparaison

La modélisation représente la plus grande différence entre les deux développements. Le développement objet offre une structuration basée sur les éléments composant l'ouvrage étudié, et sur les divers ensembles permettant la conception. Le développement prolog est basé sur la structuration des connaissances et sur le mode de raisonnement menant à la conception désirée. L'utilisation d'un langage orienté objet requiert une très grande rigueur dans les étapes d'analyse et de programmation. Il est inutile de se lancer dans la programmation sans avoir préalablement analysé et modélisé les librairies et de leurs modes de gestion, les échanges de messages, et la représentation des objets. Le propre d'un développement objet est de faire une modélisation complète et fiable afin de simplifier la programmation et de permettre un entretien, une évolution et une réutilisation importante du logiciel produit. La programmation conventionnelle est impossible dans un langage comme Smalltalk. Il est donc inutile s'aventurer dans un développement en orienté objet sans respecter et appliquer les contraintes d'une telle utilisation. L'étape de modélisation n'est donc pas simplement souhaitable, mais est indispensable. En prolog, les librairies ne sont pas complètement séparées du reste du développement, et les règles sont réparties mais ne sont pas regroupées par utilisation. Une modélisation initiale exigeante n'est pas requise pour un développement de ce genre.

Les environnements des deux langages utilisés sont différents. Il est plus facile et surtout plus rapide de s'adapter à l'utilisation d'un langage Prolog. Smalltalk présente un environnement tout à fait nouveau. Une interface complexe permet à l'usager de se déplacer dans les différentes classes et d'accéder aux méthodes correspondantes. Mis à part cette difficulté d'adaptation, le langage Smalltalk offre à l'utilisateur une bibliothèque de classes et de méthodes intéressantes. L'étape de programmation est ainsi largement

facilitée par la multitude de fonctions pré-programmées. Ce genre de bibliothèque n'existe pas dans le langage prolog. Les modifications de la programmation ou de la structure sont possibles dans les deux types de logiciels. Chacun d'eux permet des modifications mineures rapides pour un utilisateur expert. Une aide doit toutefois lui être apportée s'il n'est pas le concepteur.

L'étape d'optimisation du logiciel ne présente que peut de différence entre les deux développements. L'interface, utilisant les possibilités du système d'exploitation Windows, pour le logiciel objet présente de grandes qualités de présentations, de couleurs et de manipulations. Ces avantages sont amoindris par une vitesse d'exécution plutôt lente. Cette vitesse est beaucoup plus rapide pour le développement en prolog qui présente une interface simple, mais appropriée.

Les domaines de la réutilisation et d'exploitation présentent une certaine différence entre les deux logiciels. Beaucoup d'éléments semblent être réutilisables dans le développement orienté objet. Une modélisation structurée et détaillée facilite l'exploitation du logiciel. Elle lui assure une évolution et une réutilisation possibles. L'étape de modélisation décide des éléments réutilisables, mais l'étape de programmation peut contredire la première étape si les divers éléments ne sont programmés de façon parfaite. Il est possible dans smalltalk d'éviter de programmer de façon idéale. Des efforts sont nécessaires, et une attention particulière doit être portée sur les échanges entre les objets, et leurs modes d'appel. Avec un minimum d'expérience, il est toutefois aisé de s'assurer de la réutilisation de chaque élément créé. La rapidité de l'étape de modélisation avec le langage prolog hypothèque les possibilités de réutilisation. La base de faits peut être utile à nouveau, ainsi que différentes librairies.

CHAPITRE VII
ÉVALUATION ET ÉVOLUTION

7.1 Notion de temps

Dans le cadre d'un développement informatique comme Hydram, le client s'attend à recevoir un produit de qualité à moindre coût. Le temps devient donc une notion très importante. Le concepteur du logiciel doit donc choisir un langage informatique et une approche logique qui lui permettront de maximiser son temps. Cette notion de temps peut devenir le critère de décision final.

Nous pouvons dés à présent évaluer les possibilités de gain de temps engendrées par l'utilisation d'un langage orienté objet. Il est important de tenir compte de l'expérience informatique d'un utilisateur dans le domaine du projet. La notion de réutilisation dépend de cette expérience.

7.1.1 Aucune expérience

Un débutant connaît beaucoup de difficultés en se lançant dans la programmation objet. Ces difficultés d'adaptation et de compréhension semblent entraîner une perte de temps importante avant de pouvoir démarrer la phase de programmation. Il faut s'habituer à une logique, une interface et un environnement tout à fait différents des langages conventionnels. Ainsi, avant de connaître et d'exploiter les bases d'un langage, un utilisateur débutant aura besoin de plus de temps dans un environnement orienté objet. Cette réflexion est valable avec les langages structurés de la génération de Smalltalk. Ils présentent une approche complexe et sont peu documentés quant à leur utilisation.

Il est toutefois possible d'utiliser un langage comme Smalltalk sans comprendre l'intégralité des notions orientées objet. Toutefois, les difficultés d'adaptation à

l'environnement orienté objet excluent l'emploi d'un tel langage comme un langage procédural de base, le temps investit ne serait plus justifiable. Si un cheminement procédural est indiqué pour le projet informatique, un langage procédural est donc fortement recommandé.

7.1.2 Expert informatique, débutant dans le domaine

Smalltalk est un langage orienté objet puissant et simple d'utilisation pour un programmeur qui y est familier. Ses avantages sont nombreux lorsque toutes les possibilités de ce langage sont exploitées. Elles ne peuvent l'être que par un utilisateur aguerri à ce langage.

La possibilité la plus attrayante des langages orientés objet est la réutilisation. Il est ainsi possible d'utiliser des objets ou des parties de programmes d'anciens développements ou de logiciels annexes avec une certaine facilité. Ces divers compléments peuvent être présents dans des bibliothèques informatiques. Ils peuvent également être créés par l'utilisateur dans d'autres développements. Il est donc important de différencier un utilisateur ayant développer des outils informatiques dans le domaine du projet à l'étude. Un utilisateur expert sans expérience informatique dans le domaine d'étude ne pourra pas gagner du temps avec des outils déjà réalisés.

Les outils développés ne le sont toutefois pas uniquement dans le domaine scientifique à l'étude. Des outils pour faciliter l'interface, gérer des dictionnaires ou des graphiques ne sont pas spécifiques à un domaine particulier d'étude. Ces développements pourront donc être facilement réutilisés par un programmeur familier avec le langage employé.

Des modules informatiques favorisant la programmation et la gestion de l'interface sont de plus accessibles et peuvent engendrer un gain de temps non négligeable.

7.1.3 Expert informatique, expérience dans le domaine

L'utilisation optimale d'un langage orienté objet comme Smalltalk survient lorsque le programmeur entame un nouveau développement dans un domaine familier. Lors de tous développements sérieux, un programmeur objet se créé des bibliothèques de formules, de données, et de procédures réutilisables dans un développement futur. La création de ces objets peut être longue et coûteuse, mais leurs réutilisations entraînent un gain de temps immense.

La notion de réutilisation prend donc toute son importance et son utilité lorsque le programmeur est un expert dans ce genre de langage et a déjà oeuvré dans le domaine.

7.1.4 Conclusion

Le temps d'apprentissage semble donc être supérieur pour un langage orienté objet. Les gains de temps qui découlent de l'utilisation de ce type de langage dépend de l'expérience que le programmeur a dans le domaine de développement. Un programmeur d'expérience peut créer des objets qui faciliteront et accéléreront de nouveaux développements. La création d'outils pour faciliter l'interface et la gestion de bibliothèques est non seulement utile, mais surtout indispensable. Ils permettent à l'utilisateur de diminuer un temps important accordé à la manipulation de paramètres et de données.

Pour engendrer un gain de temps dans la programmation d'un nouveau développement, le programmeur doit donc être un expert en orienté objet, et posséder une expérience dans le domaine à l'étude. Si ces conditions ne sont pas remplies, il est illusoire espérer économiser du temps dans la phase de modélisation et de programmation.

Les langages orientés objet offre une structuration claire et détaillée d'un développement informatique. Les différents éléments constituant le programme sont en général distincts dans leur comportement et dans la programmation. C'est le propre d'un développement orienté objet. Cette séparation des divers modules présente un grand avantage dans la phase d'optimisation d'un programme. Cette phase concerne toutes les modifications aux niveaux de l'interface, de l'aide à l'usager et de la maniabilité du logiciel produit. La plupart des modifications apportées durant cette phase ne concernent que l'interface. La séparation des divers modules permet la modification de l'un sans jamais affecter les autres. Il devient très simple de modifier l'interface ou d'apprécier la maniabilité du logiciel. Un gain de temps est donc ajouté à ce stade de la programmation.

Il faut considérer la notion d'investissement dans cette analyse de temps. Un développement orienté objet peut être plus long qu'un autre, car le programmeur et l'analyste doivent offrir une future réutilisation de certaines parties du développement. Le temps passé à permettre la réutilisation et à prévoir la maintenance du logiciel représente cette différence de temps. La durée de codage est quant à elle identique pour un programmeur expérimenté.

7.2 Notion de réutilisation

Dans l'étude effectuée, nous n'avons qu'effleuré le concept de réutilisation Des classes de références et de consultations ont été créées, mais leurs réutilisations futures restent hypothétiques. En manipulant ce concept, il devient rapidement évident que l'accès et la compréhension restent très difficiles pour un débutant.

Lors des phases de modélisation et de programmation, il est important de vérifier la pertinence de rendre l'outil réutilisable. Si tel est le cas, sa mise en oeuvre doit s'effectuer dés le début de l'étude. Transformer un module informatique en module réutilisable semble être assez difficile. Pour permettre la réutilisation d'un logiciel, d'une base de

données, ou d'un simple dictionnaire, de nombreux éléments doivent être pris en compte. Tout d'abord, l'accès à cet outil doit être simple, les données d'entrées claires et son utilisation documentée. La réutilisation d'un objet nécessite une information précise sur le type d'entrée attendue. Les résultats fournis par cet objet sont précis et doivent donc pouvoir être acceptés par l'utilisateur. Enfin, les opérations effectuées par l'objet en question ne doivent présenter aucune variable ou contexte spécifique au projet en cours, car il serait inutilisable pour un futur projet. Ce ne sont que quelques notions du concept de réutilisation qu'il est très important de respecter.

La création d'outils réutilisables nécessite une grande expérience dans le domaine de l'orienté objet et d'une bonne connaissance du domaine de l'étude et de ses besoins.

La réutilisation permet toutefois d'utiliser des objets qui ont été créés par d'autres experts en ce domaine. Il existe des bibliothèques de classes (ou d'objet). Ces bibliothèques présentent l'avantage d'offrir à un informaticien un large éventail de modules informatiques déjà développés, avec les difficultés reliées à leurs utilisations. La quantité de ces bibliothèques augmente chaque jour. Elles traitent pour la plupart de facilités informatiques mis à la disposition d'un utilisateur. Dans le domaine du génie civil, ou de la science en général, ces bibliothèques n'en sont qu'au stade de développement. Il est donc illusoire d'espérer utiliser des modules informatiques déjà développés dans notre domaine d'étude.

Le langage Smalltalk possède des objets facilement utilisables. Ces objets facilitent la programmation, améliorent l'interface et offrent plusieurs options pour rendre plus abordable la syntaxe de ce langage.

La réutilisation est donc une affaire d'expert dans ce domaine. Un utilisateur débutant ne peut compter sur elle pour accélérer son étape de programmation. La création d'objets

réutilisables reste difficile sans une grande expérience. Toutefois, une fois ce concept assimilé, il offre un gain de temps inestimable pour le développement de futurs projets.

Les coûts engendrés par la réutilisation sont nombreux mais peuvent être comptabilisés en tant qu'investissements. Le programmeur doit consacrer du temps pour la création de classes d'objets. Il faut donc accepter de dépenser davantage pour les premiers développements afin d'investir dans la réutilisation des classes. Les futures applications en bénéficieront. Un gain de temps important entraînera des réductions de coûts. Moulin et Nguyen (1993) proposent à certaines compagnies et organisations d'investir dans le développement de bibliothèques pour que des objets soient disponibles lors d'utilisations futures. Il semble toutefois improbable que des compagnies se lancent dans de tels investissements à moins d'en obtenir des bénéfices.

7.3 Concept orienté objet

Dans une conception orientée objet, la décomposition est basée sur les objets du domaine du problème. Les développeurs de différents programmes dans le même domaine auront tendance à découvrir les mêmes objets. Ce fait augmente la réutilisation des composantes d'un projet à l'autre dans le même domaine.

Le concept orienté objet est une nouvelle façon d'aborder un problème. Il s'appuie sur la modélisation d'objets concrets et abstraits pour représenter le développement. Des objets bien conçus peuvent servir de bases à de nouveaux logiciels. C'est le principe de la réutilisation. L'orienté objet en général est une approche très intéressante car elle oblige les concepteurs et les experts à réfléchir en terme d'entités et non pas en terme d'ensembles. Le principe d'individualiser chaque partie d'un projet rend ce dernier beaucoup plus structuré et mieux équipé face au temps. Le temps représente la durée d'utilisation d'un logiciel. Des développements qui ne seraient pas objets ne pourront pas atteindre de grandes tailles sans connaître des problèmes avec le temps. Tous les logiciels

provenant de la compagnie Microsoft (Windows, Microsoft office...) se doivent d'être évolutifs, et conçus complètement en objets. Une modification quelconque ou une mise à jour sont fréquentes, et doivent pouvoir être fait très rapidement. Par exemple, si une opération de la souris est défectueuse, il suffit de réinstaller le module qui gère les opérations de la souris. La firme Borland prétend aujourd'hui qu'elle peut produire plus vite que ses concurrents, grâce à l'implantation de la technologie orientée objet. La plupart des logiciels provenant de ces grandes compagnies se ressemblent dans les interfaces, les modes de gestion, les opérations disponibles. Avec des outils objets, la création d'un nouveau logiciel est accéléré par l'utilisation des développement effectués.

Ces avantages sont évidents pour de grands développements, mais le concepteur peut se questionner sur l'utilisation de l'orienté objet pour des développements de tailles petites à moyennes. Pour un développement unique de petite taille, il n'est pas utile de s'aventurer dans un développement orienté objet si le concepteur n'y est pas familier. Le temps investit ne serait pas justifié. Il est préférable d'utiliser un langage connu avec le programmeur se sent bien pour ce genre de développement. Par contre, si les développement sont fréquents dans un même domaine, il est alors important de considérer l'orienté objet dans ses choix d'orientation informatique.

7.5 Choix informatiques pour futurs développements

Cette partie sera développée en deux étapes, l'une sur les questions à soulever d'une manière générale, et la deuxième sur mon expérience personnelle.

7.5.1 Problématique générale

Afin de faire un choix éclairé sur le langage informatique pour un futur développement, il est indispensable d'évaluer certains paramètres:

- type de développement: le domaine d'étude, les types de traitements, le besoin de simulations, la taille du développement...
- Attentes exécution: la vitesse d'exécution, le type d'interface, le type de gestion informatique...
- Attentes exploitation: l'apport de modifications, les possibilités de faire de nouveaux développements, la réutilisation, la prévision de domaines évolutifs.
- Ressources humaines: Le temps disponible et l'expérience du programmeur, la connaissance du domaine de modélisation, l'aide de développements précédents.

Avant de se lancer dans une modélisation et une programmation orientée objet, il est préférable de vérifier que le type de développement à résoudre y soit adapté. La mode ne peut justifier le fait de se lancer dans un modèle informatique sans vérifier que le moyen utilisé soit le plus performant pour ce genre d'application.

7.5.2 Choix personnels

L'apprentissage de l'orienté objet a été assez long. Un manque d'exemple réel et de littérature adaptée à ralenti mon cheminement. Après avoir effectué un premier développement avec le langage Smalltalk, je suis désormais familier avec les moyens de modélisation, la syntaxe du langage, et les différentes librairies disponibles. Toutefois, un seul développement ne suffit pas à créer des objets facilement réutilisables. Pour un développement futur, les objets crées au cours de ce projet devront être modifiés et adaptés, et ce, même si le domaine d'étude reste identique.

La réutilisation reste donc difficile avant le développement de plusieurs projets. Il est toutefois possible d'utiliser différentes librairies à conditions que leurs modes d'appel et d'utilisation soient clairement présentés. Les librairies dans le langage Smalltalk sont claires et détaillées.

En faisant abstraction des cas spécifiques, il est préférable de programmer dans un langage avec lequel on se sent le plus à l'aise. Les cas spécifiques sont les développements dont les paramètres correspondent à un langage ou à un type de langage particulier: simulations, vitesse d'exécution, interface, plate forme requise (windows, dos), type d'exploitation, type de réutilisation ...Pour une suite de développements dans un même domaine, quel qu'il soit, l'orienté objet serait inévitable pour s'assurer d'un entretien aisé à long termes, et d'un gain de temps appréciable. Il est donc probable qu'une modélisation orienté objet soit mon choix personnel pour un développement futur. Une fois maîtrisé, l'orienté objet peut être utilisé pour de nombreux types d'informatisations, de grandes ou de petites tailles.

7.6 Avenir de l'orienté objet

L'orienté objet présente un avenir qui va dépendre de la direction que prendront les concepteurs de langages orientés objet. Le langage qui a été utilisé pour le développement du logiciel, reste difficile d'approche pour un programmeur sans expérience. Smalltalk a été retenu comme langage de programmation car il pousse le programmeur à utiliser et comprendre toutes les facettes de l'orienté objet, dans ses principes de base et sa syntaxe complète. Cet avantage présente cependant une grande difficulté pour le programmeur débutant. Il existe aujourd'hui des versions supérieures qui permettent une utilisation optimale de l'orienté objet. Différents modules sont largement développés et facilitent la programmation. Le codage de l'interface est très exigeant et requiert beaucoup de patience dans la version de Smalltalk utilisée. Enormément de lignes de codes, et donc de temps, sont nécessaires pour obtenir un bon résultat. Cet investissement est trop important pour cette partie du développement. Les versions supérieures améliorent cet aspect de la programmation en offrant une création simplifiée de l'interface. Des outils sont à la disposition de l'utilisateur afin d'accorder un temps raisonnable à ce genre de construction artistique...

L'orienté objet est une approche informatique remplie de promesses. Elle offre des possibilités immenses et de grands avantages dans le monde du développement informatique. Il est toutefois important de constater que l'accès à ce progrès n'est pas à la porté de tous. La compréhension en profondeur des ressources que les langages informatiques actuels offrent est difficile. La logique veut que tous progrès soient accompagnés par une certaine dose d'efforts. Ici, L'effort doit être fait par les concepteurs de langages de programmation, et non pas par les utilisateurs de ces langages. Le virage orienté objet ne pourra être pris que lorsque les langages disponibles seront d'approche plus simple.

Les langages orientés objet doivent à tout prix se développer pour assurer une approche plus aisée et une compréhension de leurs mécanismes internes plus rapide. Le futur de l'orienté objet dépendra en grande partie de ces améliorations.

La phase cruciale d'un développement orienté objet est la modélisation de la problématique. Pour permettre une approche plus aisée, des méthodes de modélisation supplémentaires doivent être développées afin de couvrir un large éventail de domaines. L'analyste doit être bien guidé dans cette étape afin d'exploiter tous les avantages d'une modélisation orientée objet. Le virage objet se prendra ainsi plus facilement dans un avenir proche.

CONCLUSION

Un développement unique de petite taille ne requiert pas l'utilisation d'un langage orienté objet. Un utilisateur débutant passera plus de temps à comprendre les principes objets que le temps de programmation disponible. Toutefois, un utilisateur expert produira un logiciel de très bonne qualité en un temps raisonnable.

Les principes orientés objet se doivent d'être appliqués lorsqu'il sera possible d'exploiter certaines caractéristiques, comme la réutilisation et l'évolution possibles des produits réalisés. Une suite de développements informatiques dans un même domaine offre les conditions idéales pour l'exploitation des avantages de l'orienté objet. Avec certains efforts initiaux, l'utilisateur peut fournir un produit de très grande qualité, avec un entretien et une évolution possible. Les efforts initiaux concernent la modélisation des développements. Elle doit être organisée de façon à privilégier le principe de réutilisation. L'expression « dans le même domaine » est très large. Elle ne porte pas seulement sur un domaine scientifique ou un milieu de gestion. Cette expression considère le partage de certains éléments, qui peuvent être au niveau de la structure, de l'interface ou des bases de données...

Tout langage informatique peut permettre le développement de n'importe quel projet. Toutefois, en fonction du domaine ou des attentes, les efforts requis peuvent être plus importants. Par exemple, Prolog n'est pas un langage performant pour des applications purement numériques. Fortran a connu sa popularité par la qualité de ce genre d'application. Les langages orientés objet commencent à être reconnus et le seront encore davantage pour le développement de série de logiciels.

Le virage orienté objet est pris par de nombreuses entreprises qui produisent des logiciels. Des compagnies comme Microsoft ou Borland en sont des exemples concrets. La vitesse

de production des logiciels et les nombreuses mises à jour disponibles sont possibles grâce aux vertus d'une structuration ordonnée et à un grand effort d'investissement dans des bibliothèques d'objets. Il est, pour l'instant, très difficile pour un concepteur occasionnel de prendre ce virage. Les méthodes de modélisation aujourd'hui disponibles manquent d'exemples pratiques. Elles sont difficiles d'utilisation avant d'avoir une bonne expérience dans le domaine de l'orienté objet. Les langages disponibles, quant à eux, s'améliorent d'années en années pour faciliter la programmation objet. De nouveaux produits sont en train de percer le marché grâce à une syntaxe de programmation réduite au minimum. Les bibliothèques d'objets disponibles sont de plus en plus étendues afin de permettre l'accès à ce genre de langage à tous les utilisateurs, experts ou débutants.

Dans un domaine comme le génie civil, les possibilités sont immenses. Beaucoup d'objets peuvent être partagés entre une multitude d'applications. Les efforts requis pour utiliser la technologie de l'orienté objet ne sont pas justifiés pour un développement unique et isolé.

BIBLIOGRAPHIE

AUBIN, L., CHAMPOUX, R. "L'évacuateur de crue de LG 3 ", Congrès des grands barrages, New Delhi, 1979, Q.50 R.7.

AUBIN, L.,LEFEBVRE, D. " Les évacuateurs de crue des aménagements hydroélectriques LG 2 et LG 1 du complexe La Grande ", Congrès des grands barrages, New Delhi, 1979, Q.50 R.8.

LAFON, M. "Les langages à objets, Principes de base, techniques de programmation ", Édition Armand Colin, Paris 1992.

BELLIER, J. "Évacuateurs de crue « longs » ", Congrès des grands barrages, Istamboul, Turquie, 1967, Q.33 R.40.

BONNET, A. "L'intelligence artificielle, promesses et réalités ", InterEditions, Paris, 1984.

BONNET, A. "Systèmes Experts, vers la maîtrise technique ", InterEditions, IIA, Paris, 1986.

BOOCH, G. " Object-Oriented Design with Applications ", Redwood City, CA, 1991.

BOURGIN, A. " Considérations sur la conception d'ensemble des ouvrages d'évacuation provisoires et définitifs des barrages ", Congrès des grands barrages, Istamboul, 1967, Q.33 R.27.

BRATKO, I. " Prolog, programming for artificial intelligence ", second edition, Addison Wesley, 1990.

BRULÉ, J., BLOUNT, A. " Knowledge acquisition ", Artificial Intelligence Series, McGraw-Hill Publishing Company, 1989.

CASSIDY, J " Fluid mechanics and design of hydraulic structures ", Journal of Hydraulic Engineering, V116, p961-77, Août 1990.

CIGB-ICOLD, " Sécurité des barrages, recommandations ", Bulletin 59, 1987

CIBG-ICOLD, " Évacuateurs de crue de barrages ", Bulletin 58, 1987

Cinquième Congrés international des grands BARRAGES, supplément à la revue "Travaux" - N° 247, mai 1955

COAD P., YOURDON, E. "Object-Oriented Design ", Prenctice-Hall, N.J., 1991.

Comité ad hoc des méthodes de calcul des barrages, "Logiciels de calcul des barrages. Validation. Réflexions et propositions ", Commission internationale des grands barrages, Bulletin 94, 1994.

COULETTE, B. , MARCAILLOU, S. " Intégration de la notion de point de vue dans la modélisation par objets ", ICO Québec, p.11-21, Automne 1993.

DUBOIS, J.-C. "L'analyse du risque appliquée aux systèmes experts ", Applications de l'ICO, Intelligence artificielle et sciences cognitives au Québec, Volume 3, Numéro 2, p.27-31,été 1991.

FELDMAN, A.D., "HEC models for water Resources system simulation : Theory and experience", advances in hydroscience, vol.12, Academic Press, New York.

GOLDBERG A., "Smalltalk 80", Addison Wesley, 1983

Groupe de travail du comité Français " Ouvrages d'évacuation de grande capacité ", Congrés des grangs barrages, New Delhi, 1979, Q.50 R.61.

HAGER, W. H. "Écacuateurs. Ondes de choc et entrainement d'air. Synthèse et recommandations ", Sous-Comité n°4 du Comité de l'Hydraulique des Barrages, Bulletion 81, 1992.

HARVEY, G., MOULIN B. " Revue et comparaison de quelques méthodes de conception de systèmes orientés objet ", ICO Québec, p.38-50, Automne 1993.

MASINI, G., NAPOLI, A., ... "Les langages à objets, langages de classes, langages de frames, langages d'acteurs ", Informatique Intelligence Artificielle, Février 1990.

MOUELHI, G., MARINIER, J.P. "Évacuateur de crue du barrage de Sidi Saad ", Congrès des grands barrages, New Delhi, 1979, Q.50 R.5.

MOULIN, B., NGUYEN, C. L. " La technologie orientée objet: Évolution ou révolution ? ", ICO Québec, p.4-9, Automne 1993.

MOULIN, B., SIMIAN, G. " Informatique cognitive des organisations ", Colloque ICO'89, Québec, juin 1989.

OSKOLKOV, A.G. "Experience in designing and maintenance of spillway structures on large rivers in the USSR ", Congrès des grands barrages, New Delhi, 1979, Q.50 R.46.

OUÉDRAOGO, C. O. "Méthodologie orientée objet : L'approche intégrative est-elle une solution ? ", Séminaire orienté vers les applications en entreprises, Montréal, Mars 1994.

REESE, A., MAYNORD, S. " Design of spillway crests ", Journal of Hydraulic Engineering, V113, p476-90, Avril 1987

ROBERT, B., TALEB, M., MARCHE, C. "Développement et usage d'un système expert pour le choix des évacuateurs de crue ", Revue canadienne de génie civil, Volume 19, numéro 5, octobre 1992.

ROBERT B., "Un système à base de connaissance pour la conception des évacuateurs de crues", Rapport technique, Juin 1996.

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., LORENSEN, W. "Object-Oriented Modeling and Design ", General Electric Research and Development Center, New Jersey, 1991.

SOKOLOV, I.B, ROSANOV, N.S. " Rupture et accidents de barrages et recherche sur leur sécurité ", Commission internationale des grands barrages, New Delhi 1979, Q.49 R.36

Symposium ICO 94, " L'orienté objet, mode d'informaticiens ou nécessité pour les organisations ", Montréal, Mai 1994.

TALEB, M. "Approche logicielle à la conception d'évacuateurs de crues, dissipateurs d'énergie et vidanges de fond ", thèse Doctorat, École Polytechnique de Montréal, 1991.

THÉORET, A. " Bilan des systèmes experts au Québec ", Applications de l'ICO, Intelligence artificielle et sciences cognitives au Québec, Volume 3, Numéro 2, p.13-18, été 1991.

United States Departement of the Interior, Bureau of reclamation " Design of Small Dams, a water resources technical publication ",Denver, Colorado, 1987.

WETTER, V. "Evacuateurs de crue ", Congrès des grands barrages, New Delhi, 1951, Q.12 R.10.

ANNEXE 1

COMPARAISON DE MÉTHODES DE CONCEPTION

CRITÈRES	MÉTHODOLOGIES		
	Coad/Yourdon	**OMT**	**Booch**
1. Cycle de développement:			
- Analyse:	Oui	Oui	Oui
. description des activités	Bon	Bon	Mauvais
. modèles, spécifications	Bon	Bon	Moyen
- Conception:	Oui	Oui	Oui
. description des activités	Moyen	Bon	Mauvais
. modèles, spécifications	Moyen	Bon	Bon
- Implémentation:	Non	Oui	Non
. description des activités		Bon	
2. Formalisme			
. facile à maîtriser	Oui	Modérément	Non
. bon outil de communication	Oui	Oui	Modérément
. non ambigu	Oui	Non	Non
3. Outils automatisés:			
. modèles et graphiques	Oui	Oui	Oui
. validation et cohérence	Oui	?	?
. génération de code	Non	?	?
. intégration du dictionnaire	Non	?	?
4. Utilisation du prototypage	Oui	Oui	Oui
5. Contrôle de qualité	Non	Oui	Oui
6. Définitions des biens livrables:	Bon	Bon	Bon
. modèles	Moyen	Moyen	Bon
. spécifications			
7. Documentation de la méthodologie:			
. volume	Oui	Oui	Oui
. articles	Oui	Oui	Oui
. cours	Oui	?	Oui

Lecture de la table :

Bon : support satisfaisant
Moyen : support moyen
Mauvais : support insatisfaisant
? : manque d'information

ANNEXE 2

COMPARAISON DES LANGAGES ORIENTÉS OBJET

J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy et W. Lorensen ont effectué une comparaison entre différents langages utilisés pour des développements orientés objet. Voici, résumés et comparées, les principales caractéristiques de ces langages:

	C++ 2.0	Smalltalk 80	CLOS	EIFFEL	Objective C
Intégration des classes avec des types primitifs	hybride	pur	intégré	intégré	hybride
Typage fort	oui	non	non	oui	oui
Faculté de restreindre l'accès aux attributs					
Contrôle d'accès aux clients	oui	oui	non	oui	oui
Contrôle d'accès par les sous classes	oui	non	non	oui	non
Bibliothèque de classes standard	non	oui	non	oui	oui
Classes paramétrables	futur	-	-	oui	non
Héritage multiple	oui	non	oui	oui	non
Portée des noms de classes	non	non	oui	non	non
Modèle de messagerie					
Objet cible unique	oui	oui	non	oui	oui
Lien dynamique sur arguments mul.	non	non	oui	non	non
Caractéristique de combinaison des méthodes					
Concept SUPER	non	oui	non	oui	oui
Méthodes avant et après	non	non	oui	non	non
Assertions et contraintes	non	non	non	oui	non
Méta-données à l'exécution	non	oui	oui	non	oui
Ramasse-miettes	non	oui	oui	oui	non
Efficacité					
Lien statique là ou c'est possible	oui	non	non	oui	oui

Lecture de la table :

oui : la propriété est présente
non : la propriété est absente dans les implémentations courantes
futur : planifié dans une future version du produit
- : non applicable, les classes paramétrables ne sont pas nécessaires avec le typage faible.

ANNEXE 3

LISTE DES COURBES UTILISÉES

FIGURE	DESCRIPTION	ORIGINE
Courbe A recommandée	Coefficient de débit pour charge d'eau variable	Hydro-Québec
Courbe B recommandée	Coefficient de débit pour seuil parabolique à parement vertical	Hydro-Québec
Figure 288D2406, valeurs de k, inclinaison 1:3	Facteur k du seuil de l'évacuateur avec une inclinaison du parement amont de 1:3	DSD
Figure 288D2406, valeurs de k, inclinaison 2:3	Facteur k du seuil de l'évacuateur avec une inclinaison du parement amont de 2:3	DSD
Figure 288D2406, valeurs de k, inclinaison 3:3	Facteur k du seuil de l'évacuateur avec une inclinaison du parement amont de 3:3	DSD
Figure 288D2406, valeurs de n, vertical	Facteur n du seuil de l'évacuateur avec un parement amont vertical	DSD
Figure 288D2406, valeurs de n, inclinaison 1:3	Facteur n du seuil de l'évacuateur avec une inclinaison du parement amont de 1:3	DSD
Figure 288D2406, valeurs de n, inclinaison 2:3	Facteur n du seuil de l'évacuateur avec une inclinaison du parement amont de 2:3	DSD
Figure 288D2406, valeurs de n, inclinaison 3:3	Facteur n du seuil de l'évacuateur avec une inclinaison du parement amont de 3:3	DSD
Figure 288D2407, valeurs de xc, vertical	Facteur xc de la crête de l'évacuateur avec un parement amont vertical.	DSD
Figure 288D2407, valeurs de xc, inclinaison 1:3	Facteur xc de la crête de l'évacuateur avec une inclinaison du parement amont de 1:3.	DSD
Figure 288D2407, valeurs de xc, inclinaison 2:3	Facteur xc de la crête de l'évacuateur avec une inclinaison du parement amont de 2:3.	DSD
Figure 288D2407, valeurs de xc, inclinaison 3:3	Facteur xc de la crête de l'évacuateur avec une inclinaison du parement amont de 3:3.	DSD
Figure 288D2407, valeurs de yc, vertical	Facteur yc de la crête de l'évacuateur avec un parement amont vertical.	DSD
Figure 288D2407, valeurs de yc, inclinaison 1:3	Facteur yc de la crête de l'évacuateur avec une inclinaison du parement amont de 1:3.	DSD
Figure 288D2407, valeurs de yc, inclinaison 2:3	Facteur yc de la crête de l'évacuateur avec une inclinaison du parement amont de 2:3.	DSD
Figure 288D2407, valeurs de yc, inclinaison 3:3	Facteur yc de la crête de l'évacuateur avec une inclinaison du parement amont de 3:3.	DSD
Figure 288D2409	Coefficient de débit pour un creager avec un parement amont vertical	DSD

Figure 288D2410	Coefficient de débit en fonction de la hauteur de charge	DSD
Figure 288D2411, inclinaison 1:3	Coefficient de débit pour un parabolique avec un parement amont 1:3	DSD
Figure 288D2411, inclinaison 2:3	Coefficient de débit pour un parabolique avec un parement amont 2:3	DSD
Figure 288D2411, inclinaison 3:3	Coefficient de débit pour un parabolique avec un parement amont 3:3	DSD
Figure 288D2413	Coefficient de débit en fonction des conditions aval	DSD
Figure 288D2414	Coefficient de débit en fonction du degré de submergence	DSD

Lecture de la table :
DSD : Design of Small Dams, 1987

ANNEXE 4

LIBRAIRIE DE FORMULES

Chaque formule correspond à une méthode qui est placée dans la classe de développement. Cette classe fournit les caractéristiques de calculs nécessaires à la librairie de formules. Le tableau suivant présente une description sommaire des formules, ainsi que le nom de la méthode qui lui correspond.

DESCRIPTION	FORMULES	LOCALISATION
Coefficients de débit Ce	$Ce = Cd * fct1.Fra.(0.3048)^{1/2}$	*coefDebitCreaHCe*
Coefficients de débit Ce	$Ce = Cd * fct1.fct2.Fra.(0.3048)^{1/2}$	*coefDebitParaHCe*
Coefficients de débit Ce	$Ce = Cd * fct1.fct2.Fra.fsub.(0.3048)^{1/2}$	*coefDebitParaB*
Débit Q	$Q = Ce * Le * He^{3/2}$	*coefDebitQ*
Débit Q=fct(Bpa,He)	$Q = Ce(NpaBpa - 2(NplKpl + Kc)He)He^{3/2}$	*coefDebitQ2:valeurHe*
Hauteur de pelle P	$P = \frac{P}{H_D} * H_D$	*coefHauteurPelleP*
Niveau du Roc *chainage*	$Roc = zRoc - Nettoyage$	*coefNiveauRoc:*
Hauteur de charge He	$He = \frac{Bpa}{Bpa/He}$	*coefHauteurHeFctBpa*
Hauteur de charge He	$He = \frac{Q^{2/5}}{\left(Ce\left(Npa\frac{Bpa}{He} - 2(NplKpl + Kc)\right)\right)^{2/5}}$	*coefHauteurHe*
LargeurBrute L_B	$Lb = Npa * Bpa$	*coefLargeurBruteLb*
Largeur de passe Bpa	$Bpa = He * \frac{Bpa}{He}$	*coefLargeurBpa*
Nombre de passe	$Npa = \frac{Le - 2He(Kpl - Kc)}{Bpa - 2HeKpl}$	*coefNombrePasseNpa*
Nombre de pilier	$Npl = Npa - 1$	*coefNombrePilierNpl*
Largeur effective Le	$Le = \frac{Qconception}{CeHe^{3/2}}$	*coefLargeurEffecFctQ*

Largeur effective Le	$Le = Lb - 2*(Npl*Kpl + Kc)*He$	*coefLargeurEffecLe*
Largeur canal amené	$Bca = (NpaBpa) + (NplBpl) + (NcBc)$	*coefLargeurBca*
Vitesse canal amené	$Vca = \frac{Qconception}{Bca(He+P)}$	*coefVitesseVca*
Cote du radier Amont	$CradierAmont = NiMaxAmont - (H_D + P)$	*coefCradierAmont*
Hauteur P modifiée	$P = \frac{Q - VcaBcaHe}{VcaBca}$	*coefModifP*
Largeur canal évacuation	$Bcde = (Npa*Bpa) + (Npl*Bpl)$	*coefLargeurBcde*
Hauteur eau pied coursier	$Ya = \frac{Qconception}{Vc*Bcde}$	*coefEauPiedCoursier*
Energie cinétique ha	$ha = \frac{Vca^2}{2*g}$	*coefEnergieCinetiqueha*
Rayon pied coursier	$R = f*Ya$	*coefRayonPiedCoursier*
Parabole x=fct(y)	$X = \left[\frac{Y*Hd^{N-1}}{K}\right]^{1/N}$	*coefXparabole:y with:k*
Parabole y=fct(x)	$Y = \frac{K*X^N}{H_D{}^{N-1}}$	*coefXparabole:y with:k*

Point de tangence T $xtg = \left[\frac{H_D{}^{n-1}.Pente}{K.N}\right]^{(1/N-1)}$ $ytg = \frac{K.xtg^N}{H_D{}^{N-1}}$

coefXYtangente:pente with:k

Point de contact M $xm = xtg + \frac{l}{\sqrt{1+pente^2}}$ $ym = ytg - \frac{pente.l}{\sqrt{1+pente^2}}$

coefXYpointM:pente with:l

Point A pied coursier $xa = xm + R\sin\theta$ $ya = ym + R(1-\cos\theta)$ coefXYpiedA:xi with:yi

ANNEXE 5

PROGRAMMATION AXE OPÉRATEUR

Voici toutes les lignes de programmation du module Axe Opérateur :

```
ViewManager subclass: #AxeOperateur
 instanceVariableNames:
  ' font1029 font820 font513 couleurChamps couleurFond couleurCarac couleurPriseDonnee presentation
dialogue1 dialogueCourant mur vision '
 classVariableNames:
  ' AxeDico Index Recup '
 poolDictionaries:
  ' ColorConstants '

affichageDescriptionContenu: unEcran
"  affichage du contenu de l'écran de description des graphiques  "

"*** affichage pour 'Description du point' ***"

 self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           contents:'Description du point';
 framingBlock:[:box | (((box width * 1 // 3) + 90) @ (( box height * 2 // 3) + 40))
extent:(150@20)]).

"*** affichage pour axe ***"

  self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents:'Axe: ';
           framingBlock:[:box | (((box width * 1 // 3) + 30) @ (( box height * 2 // 3) + 70))
                    extent:(30 @ 20)]).

  self addSubpane:
        (FormattedStaticText new
           leftJustified;
           owner: self;
           font: font513;
           foreColor: couleurPriseDonnee;
           setName: 'AxeChoisi';
           framingBlock:[:box | (((box width * 1 // 3) + 60) @ (( box height * 2 // 3) + 70))
extent:(50 @ 20)]).

"*** affichage pour le Chainage ***"

  self addSubpane:
        (StaticText new
```

```
            leftJustified;
            owner: self;
            font: font513;
            contents:'Chaînage (m)';
            framingBlock:[:box | (((box width * 1 // 3) + 30) @ (( box height * 2 // 3) + 90))
                        extent:(70 @ 20)]).

   self addSubpane:
         (FormattedStaticText new
            rightJustified;
            owner: self;
            formatAsChainage;
            font: font513;
            foreColor: couleurPriseDonnee;
            setName: 'ChPoint';
            framingBlock:[:box | (((box width * 1 // 3) + 30) @ (( box height * 2 // 3) + 110))
extent:(45 @ 20)]).

"*** affichage pour le roc ***"

  self addSubpane:
         (StaticText new
            leftJustified;
            owner: self;
            font: font513;
            contents:'Alt. roc (m)';
            framingBlock:[:box | (((box width * 1 // 3) + 120) @ (( box height * 2 // 3) + 90))
extent:(70 @ 20)]).

   self addSubpane:
 (FormattedStaticText new
            rightJustified;
            owner: self;
            font: font513;
            foreColor: couleurPriseDonnee;
            setName: 'RocPoint';
            framingBlock:[:box | (((box width * 1 // 3) + 120) @ (( box height * 2 // 3) + 110))
        extent:(45 @ 20)]).

"*** affichage pour le terrain naturel ***"

  self addSubpane:
         (StaticText new
            leftJustified;
            owner: self;
            font: font513;
            contents:'Alt. terrain nat. (m)';
            framingBlock:[:box | (((box width * 1 // 3) + 200) @ (( box height * 2 // 3) + 90))
                        extent:(100 @ 20)]).

   self addSubpane:
         (FormattedStaticText new
            rightJustified;
            owner: self;
            font: font513;
            foreColor: couleurPriseDonnee;
```

```
            setName: 'TnPoint';
            framingBlock:[:box | (((box width * 1 // 3) + 200) @ (( box height * 2 // 3) + 110))
                        extent:(45 @ 20)]).

affichageMessage: unMessage
   " Retourne une fenêtre d'information avec unMessage "
   ^(MessageBox notify:'Editeur Axe - Information' withText: unMessage)

affichageMiniPoint: unEcran
   " Retourne l'interieur de la mini fenêtre Points : "

    self addSubpane:
        (StaticText new
           centered;
           owner: self;
           backColor: couleurFond;
           contents:'Présentation des points enregistrés';
           framingBlock:[:box | (box leftTop  down:((box height * 7 // 12) + 1))
          rightBottom:(( box width) @ ((box height * 7 // 12) + 25))]).

affichageRappelContenu: unEcran

"*** affichage pour chaînage ***"

   self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents:'Chaînage: ';
           framingBlock:[:box | (((box width * 1 // 3) + 20) @ (( box height * 7 // 12) + 35))
extent:(50 @ 20)]).

 self addSubpane:
        (FormattedStaticText new
           rightJustified;
           owner: self;
           formatAsChainage;
           font: font513;
           foreColor: couleurPriseDonnee;
           setName: 'Ch';
           framingBlock:[:box | (((box width * 1 // 3) + 70) @ (( box height * 7 // 12) + 35))
extent:(45 @ 20)]).

"*** affichage Altitudes ***"

   self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           contents:'Altitudes';
           framingBlock:[:box | (((box width * 1 // 3) + 150) @ (( box height * 7 // 12) + 50))
extent:(100@20)]).

"*** affichage StaticText pour Roc ***"
```

```
    self addSubpane:
        (StaticText new
            leftJustified;
            owner: self;
            font: font513;
            contents:'Roc Gauche: ';
            framingBlock:[:box | (((box width * 1 // 3) + 20) @ (( box height * 7 // 12) + 80))
                    extent:(70 @ 20)]).

    self addSubpane:
        (StaticText new
            leftJustified;
            owner: self;
            font: font513;
            contents:'Roc Centre: ';
            framingBlock:[:box | (((box width * 1 // 3) + 20) @ (( box height * 7 // 12) + 100))
                        extent:(70 @ 20)]).

    self addSubpane:
        (StaticText new
            leftJustified;
            owner: self;
            font: font513;
            contents:'Roc Droite: ';
            framingBlock:[:box | (((box width * 1 // 3) + 20) @ (( box height * 7 // 12) + 120))
                            extent:(70 @ 20)]).

"*** affichage des champs pour roc ***"

    self addSubpane:
        (FormattedStaticText new
            rightJustified;
            owner: self;
            font: font513;
            foreColor: couleurPriseDonnee;
            setName: 'Rgr';
            framingBlock:[:box | (((box width * 1 // 3) + 90) @ (( box height * 7 // 12) + 80))
                extent:(45 @ 20)]).

    self addSubpane:
        (FormattedStaticText new
            rightJustified;
            owner: self;
            font: font513;
            foreColor: couleurPriseDonnee;
            setName: 'Rcr';
            framingBlock:[:box | (((box width * 1 // 3) + 90) @ (( box height * 7 // 12) + 100))
                                    extent:(45 @ 20)]).

    self addSubpane:
        (FormattedStaticText new
            rightJustified;
            owner: self;
            font: font513;
            foreColor: couleurPriseDonnee;
            setName: 'Rdr';
```

```
            framingBlock:[:box | (((box width * 1 // 3) + 90) @ (( box height * 7 // 12) + 120))
extent:(45 @ 20)]).

"*** affichage de la lettre m ***"

 self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents: 'm';
           framingBlock:[:box | (((box width * 1 // 3) + 140) @ (( box height * 7 // 12) + 80))
extent:(10 @ 20)]).
 self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents: 'm';
           framingBlock:[:box | (((box width * 1 // 3) + 140) @ (( box height * 7 // 12) + 100))
extent:(10 @ 20)]).
 self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents: 'm';
           framingBlock:[:box | (((box width * 1 // 3) + 140) @ (( box height * 7 // 12) + 120))
extent:(10 @ 20)]).

"*** affichage des StaticText pour terrain naturel ***"

   self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents:'Terrain naturel Gauche: ';
           framingBlock:[:box | (((box width * 1 // 3) + 160) @ (( box height * 7 // 12) + 80))
                        extent:(120 @ 20)]).

   self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents:'Terrain naturel Centre: ';
           framingBlock:[:box | (((box width * 1 // 3) + 160) @ (( box height * 7 // 12) + 100))
                       extent:(120 @ 20)]).

 self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents:'Terrain naturel Droite: ';
```

```
            framingBlock:[:box | (((box width * 1 // 3) + 160) @ (( box height * 7 // 12) + 120))
                        extent:(120 @ 20)]).

"*** affichage Champs pour terrain naturel *"

  self addSubpane:
        (FormattedStaticText new
           rightJustified;
           owner: self;
           font: font513;
           foreColor: couleurPriseDonnee;
           setName: 'Tgr';
           framingBlock:[:box | (((box width * 1 // 3) + 280) @ (( box height * 7 // 12) + 80))
                        extent:(45 @ 20)]).

  self addSubpane:
        (FormattedStaticText new
           rightJustified;
           owner: self;
           font: font513;
           foreColor: couleurPriseDonnee;
           setName: 'Tcr';
           framingBlock:[:box | (((box width * 1 // 3) + 280) @ (( box height * 7 // 12) + 100))
                                extent:(45 @ 20)]).

  self addSubpane:
        (FormattedStaticText new
           rightJustified;
           owner: self;
           font: font513;
           foreColor: couleurPriseDonnee;
           setName: 'Tdr';
           framingBlock:[:box | (((box width * 1 // 3) + 280) @ (( box height * 7 // 12) + 120))
extent:(45 @ 20)]).

"*** affichage de la lettre m pour terrain naturel ***"

  self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents: 'm';
           framingBlock:[:box | (((box width * 1 // 3) + 330) @ (( box height * 7 // 12) + 80))
extent:(10 @ 20)]).
  self addSubpane:
        (StaticText new
           leftJustified;
           owner: self;
           font: font513;
           contents: 'm';
           framingBlock:[:box | (((box width * 1 // 3) + 330) @ (( box height * 7 // 12) + 100))
extent:(10 @ 20)]).
  self addSubpane:
        (StaticText new
           leftJustified;
```

```
            owner: self;
            font: font513;
            contents: 'm';
            framingBlock:[:box | (((box width * 1 // 3) + 330) @ (( box height * 7 // 12) + 120))
                        extent:(10 @ 20)]).

aideGenerale
   " Message d'information général "

| message |
message:='
L'éditeur d'axe permet de faire un relevé sur la
localisation d'un axe d'évacuation. La définition
se fait à l'aide de points représentant le profil
du terrain.
Pour chaque point, le chaînage, l'altitude du roc
et du terrain naturel sont utiles afin d'obtenir un
profil caractéristique. Au cours de la définition
de chaque point, des vérifications se font sur
les altitudes enregistrées.

Afin d'envisager diverses possibilitées, des
relevés seront effectués le long de l'axe central,
et sur les axes voisins (droite et gauche) situés
à une largeur déterminée. Des simplications et
vérifications d'usages guident l'utilisateur afin
de simplifier la démarche.'.

   self affichageMessage: message

effaceDialogueCourant
   " efface le dialogue courant "

   dialogueCourant notNil ifTrue:[ dialogueCourant mainView hideWindow]

enregistrementDonnee
   " Permet l'enregistrement des données pour chaque point dans la collection
     AxeTravail: orderedCollection ( point chainage rocG rocC rocD tnG tnC tnD ) "

   | collectionTampon rocG rocC rocD tnG tnC tnD chainage |
   collectionTampon:= OrderedCollection new: self size.
   rocG:=((self owner paneAt:'RocG') contents).
   rocC:=((self owner paneAt:'RocC') contents).
   rocD:=((self owner paneAt:'RocD') contents).
   tnG:=((self owner paneAt:'TnG') contents).
   tnC:=((self owner paneAt:'TnC') contents).
   tnD:=((self owner paneAt:'TnD') contents).
   chainage:=((self owner paneAt:'Chainage') contents).

   collectionTampon add: Index.
   collectionTampon add: chainage.
   collectionTampon add: rocG.
   collectionTampon add: rocC.
   collectionTampon add: rocD.
   collectionTampon add: tnG.
   collectionTampon add: tnC.
```

```
    collectionTampon add: tnD.

    AxeTravail add: collectionTampon.

initialisationChamps
    " Réinitialise tous les champs après l'enregistrement du point précédent "

    | addition |
    addition:= (Index + 1).
    (self owner paneAt:'Point') contents:(addition printString).
    ((self owner paneAt:'Chainage') contents:").
    ((self owner paneAt:'RocG') contents:").
    ((self owner paneAt:'RocC') contents:").
    ((self owner paneAt:'RocD') contents:").
    ((self owner paneAt:'TnG') contents:").
    ((self owner paneAt:'TnC') contents:").
    ((self owner paneAt:'TnD') contents:").


initialisationCouleur
    " Initialise les couleurs d'affichage "

    couleurFond:= ClrPalegray.
    couleurChamps:= ClrWhite.
    couleurCarac:= ClrDarkblue.
    couleurPriseDonnee:= ClrDarkred.

initialisationFont
    " Initialise les différentes écritures qui seront utilisées au cours
      de l'exécution "

    font1029 := Font face: 'Helv'
                    size: 10@29
                    fixedWidth: false
                    bold: true
                    hollow: false
                    italic: false
                    negative: false
                    strikeOut: false
                    underscore: false.
    font820 := Font face: 'Helv'
                    size: 8@20
                    fixedWidth: false
                    bold: true
                    hollow: false
                    italic: false
                    negative: false
                    strikeOut: false
                    underscore: false.
    font513 := Font face: 'Helv'
                    size: 5@13
                    fixedWidth: false.

initialisationPresentation

    presentation:= nil.
```

```
nonDisponible
   " Retourne une fenêtre d'information avec le message non disponible "

   ^(MessageBox notify:'Editeur Axe - Information' withText: 'Opération non disponible !')

nonDisponible: unBouton
   " Retourne une fenêtre d'information avec le message non disponible "

   ^(MessageBox notify:'Editeur Axe - Information' withText: 'Opération non disponible !')

ordreCroissantAxeTravail
  "  Classe les éléments de Axe Travail en ordre croissant selon leur chaînage  "

   | collecTampon ele collecChainage taille taille2 valeurChainage
   collecOrdonnee index axeTravailCopie |

   collecChainage:= OrderedCollection new: self size.
   taille:= AxeTravail size.
   AxeTravail do: [:element | valeurChainage:= ((element at:2) asNumber ).
                              collecChainage add: valeurChainage].
   collecOrdonnee:=(collecChainage hydramCollectionOrdreCroissant ).

   index:= 0.
   ele:= nil.
   collecTampon:= OrderedCollection new: self size.
   taille2:= collecChainage size.
   1 to:taille2 do:[:valeur |  [(collecOrdonnee at:valeur) = ele]
                                 whileFalse:[index:= index+1.
                                             ele:=(collecChainage at:index)].
collecTampon add: index.
index:= 0 ].

   axeTravailCopie:= OrderedCollection new:self size.
   collecTampon do:[ :elem | axeTravailCopie add:(AxeTravail at:elem)].
   AxeTravail removeAll.
   axeTravailCopie do:[ :eleme | AxeTravail add:eleme ].

recuperationAxe

   | boiteDialogue nom nomFichier taille index collectionTampon ecartRT code |
   boiteDialogue:= (FileDialog new openFile).
   nom:=((boiteDialogue) file).
   (nom isNil) ifTrue:[Recup:= false.^nil].
   nomFichier:= (File pathName: nom).
   code:= (nomFichier nextLine) asFloat.
   (code=13467982) ifFalse:[ self affichageMessage: 'Fichier inaccessible !'.
                             Recup:=false. ^nil].
   taille:= (nomFichier nextLine) asFloat.
   index:=1.
   AxeDico at:'Npt' put: taille.
   AxeDico at:'NomProjet' put:(nomFichier nextLine).
   AxeDico at:'NomAxe' put:(nomFichier nextLine).
   AxeDico at:'Largeur' put:(nomFichier nextLine).
   ecartRT:=(nomFichier nextLine).
   AxeDico at:'RocTn' put:(ecartRT asNumber).
```

```
[nomFichier atEnd ]
   whileFalse:[[index > taille]
                  whileFalse:[ collectionTampon:= OrderedCollection new: self size.
collectionTampon add: (nomFichier nextLine).                    collectionTampon add: (nomFichier
nextLine).                    collectionTampon add: (nomFichier nextLine).
collectionTampon add: (nomFichier nextLine).                    collectionTampon add: (nomFichier
nextLine).                    collectionTampon add: (nomFichier nextLine).
collectionTampon add: (nomFichier nextLine).                    collectionTampon add: (nomFichier
nextLine).
            AxeTravail add: collectionTampon.
                  index:= index + 1]].

nomFichier close.
```

sauvegardeAxeEditeur

```
| nom fichier index taille |
(mur=false) ifTrue:[self affichageMessage:'Saisie en cours !'.^nil].
(vision=false) ifTrue:[self affichageMessage:'Opération de correction en cours !'.^nil].
self ordreCroissantAxeTravail.

nom:=(FileDialog new sauvegardeAxeConsultation: 'sauvegarde Axe' fileName:'nomAxe') file.
(nom isNil) ifTrue:[ ^nil].
File remove: nom ifAbsent: [].
fichier:= File pathName: nom.

index:=1.
taille:= AxeTravail size.
fichier nextPutAll: '13467982' asString; cr.
fichier nextPutAll: taille asString; cr.
fichier nextPutAll:((AxeDico at:'NomProjet') asString); cr.
fichier nextPutAll:((AxeDico at:'NomAxe') asString); cr.
fichier nextPutAll:((AxeDico at:'Largeur') asString); cr.
fichier nextPutAll:((AxeDico at:'RocTn') asString); cr.

[index > taille] whileFalse:[
   fichier nextPutAll:(((AxeTravail at:index) at:1) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:2) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:3) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:4) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:5) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:6) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:7) asString); cr.
   fichier nextPutAll:(((AxeTravail at:index) at:8) asString); cr.
   index:= index + 1].

fichier close.
```

sauvegardeAxeHydram

```
" Cette méthode enregistre l'axe sous un nom.hyd pour l'exploitation
  sous hydram "

| fichier nom rechercheIndex collecTampon index taille collecRocCentral valeurRoc rocMax compteur position |
(mur=false) ifTrue:[self affichageMessage:'Saisie en cours !'.^nil].
(vision=false) ifTrue:[self affichageMessage:'Opération de correction en cours !'.^nil].
```

```
    self ordreCroissantAxeTravail.
    position:=1.
    rechercheIndex:=false.
    compteur:=nil.
    collecTampon:= OrderedCollection new: self size.
    AxeTravail do:[:element | valeurRoc:= ((element at:4) asNumber).
collecTampon add: valeurRoc].
    rocMax:=(collecTampon hydramCollectionValeurMax).
    [rechercheIndex=false]
          whileTrue:[ valeurRoc:=(((AxeTravail at:position) at:4) asNumber).
 (valeurRoc=rocMax) ifTrue:[compteur:= position.
rechercheIndex:=true].
position:=(position + 1)].

    nom:=(FileDialog new sauvegardeAxeHydram: 'sauvegarde Axe' fileName:'nomAxe') file.
    (nom isNil) ifTrue:[ ^nil].
    File remove: nom ifAbsent: [].
    fichier:= File pathName: nom.
    index:= 1.
    taille:= AxeTravail size.
    fichier nextPutAll: 'identificateur_projet('".
    fichier nextPutAll: ((AxeDico at:'NomProjet') asString).
    fichier nextPutAll: '")', cr.
    fichier nextPutAll: 'identificateur_axe('".
    fichier nextPutAll: ((AxeDico at:'NomAxe') asString).
    fichier nextPutAll: '")', cr.
    fichier nextPutAll: 'definition_axe('.
    fichier nextPutAll: ((AxeDico at:'Largeur') asString).
    fichier nextPutAll: ','.
    fichier nextPutAll: taille asString.
    fichier nextPutAll: ','.
    fichier nextPutAll:'['.

    [index < taille]
      whileTrue:[ fichier nextPutAll:'point_axe('.                    fichier nextPutAll:(((AxeTravail at:index) at:2)
asString).
fichier nextPutAll:',',
fichier nextPutAll:(((AxeTravail at:index) at:3) asString).
fichier nextPutAll:',',
fichier nextPutAll:(((AxeTravail at:index) at:4) asString).
fichier nextPutAll:',',
fichier nextPutAll:(((AxeTravail at:index) at:5) asString).
fichier nextPutAll:',',
fichier nextPutAll:(((AxeTravail at:index) at:6) asString).
fichier nextPutAll:',',
fichier nextPutAll:(((AxeTravail at:index) at:7) asString).
fichier nextPutAll:',',
                    fichier nextPutAll:(((AxeTravail at:index) at:8) asString).
                    fichier nextPutAll:'),'.
                    index:= index + 1].

    fichier nextPutAll:'point_axe('.
                    fichier nextPutAll:(((AxeTravail at:index) at:2) asString).
                    fichier nextPutAll:',',
                    fichier nextPutAll:(((AxeTravail at:index) at:3) asString).
fichier nextPutAll:',',
```

```
fichier nextPutAll:(((AxeTravail at:index) at:4) asString).
fichier nextPutAll:','.
fichier nextPutAll:(((AxeTravail at:index) at:5) asString).
fichier nextPutAll:','.
fichier nextPutAll:(((AxeTravail at:index) at:6) asString).
fichier nextPutAll:','.
fichier nextPutAll:(((AxeTravail at:index) at:7) asString).
fichier nextPutAll:','.
fichier nextPutAll:(((AxeTravail at:index) at:8) asString).
fichier nextPutAll:')]'; cr.

    fichier nextPutAll:'debut_axe(point_axe('.
    fichier nextPutAll:(((AxeTravail at:1) at:2) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:1) at:3) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:1) at:4) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:1) at:5) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:1) at:6) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:1) at:7) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:1) at:8) asString).
    fichier nextPutAll:'))'; cr.
    fichier nextPutAll:'fin_axe(point_axe('.
    fichier nextPutAll:(((AxeTravail at:taille) at:2) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:taille) at:3) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:taille) at:4) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:taille) at:5) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:taille) at:6) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:taille) at:7) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:taille) at:8) asString).
    fichier nextPutAll:'))'; cr.
fichier nextPutAll:'maximum_axe(point_axe('.
    fichier nextPutAll:(((AxeTravail at:compteur) at:2) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:compteur) at:3) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:compteur) at:4) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:compteur) at:5) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:compteur) at:6) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:compteur) at:7) asString).
    fichier nextPutAll:','.
    fichier nextPutAll:(((AxeTravail at:compteur) at:8) asString).
    fichier nextPutAll:'))'; cr.
```

```
    fichier close.


AxeOperateur subclass: #AxeEditeur
  instanceVariableNames:
   ' fenetrePrincipale list listPane enregistrement ajout modification numeroPoint '
  classVariableNames:
   ' Ancien '
  poolDictionaries:
   ' ColorConstants '


affichageDefautCentre: valeurDefaut
    "  Permet l'affichage des valeurs par defaut aprés l'entrée
       de l'altitude du roc sur l'axe central  "

    | altitudeRocCentral ecart |

    altitudeRocCentral:= ((self owner paneAt: 'RocC') contents asFloat).

    ecart:= ((self owner paneAt: 'Ecart') contents asFloat).
    (self owner paneAt:'TnC') contents: ((altitudeRocCentral + ecart) printString).
    (self owner paneAt:'RocG') contents: (altitudeRocCentral printString).
    (self owner paneAt:'RocD') contents: (altitudeRocCentral printString).

    self affichageDefautDroite: valeurDefaut;
         affichageDefautGauche: valeurDefaut.

affichageDefautDroite: valeurDefaut
    "  Modification de l'altitude du terrain naturel de droite  "

    | altitudeRocDroite ecart |
    altitudeRocDroite:= ((self owner paneAt: 'RocD') contents asFloat).
    ecart:= ((self owner paneAt: 'Ecart') contents asFloat).

(self owner paneAt:'TnD') contents: ((altitudeRocDroite + ecart) printString).


affichageDefautGauche: valeurDefaut
    "  Modification de l'altitude du terrain naturel de gauche  "

    | altitudeRocGauche ecart |
    altitudeRocGauche:= ((self owner paneAt: 'RocG') contents asFloat).
    ecart:= ((self owner paneAt: 'Ecart') contents asFloat).
    (self owner paneAt:'TnG') contents: ((altitudeRocGauche + ecart) printString).


affichageModificationEcart: valeurDefaut
    "  permet l'affichage des altitudes par défaut du terrain naturel  "

    (((self owner paneAt:'Ecart') contents asNumber) < 0)
       ifTrue:[(self affichageMessage:'l'écart entre le roc et le terrain naturel ne peut être négatif !').
  ((self owner paneAt:'Ecart') contents:'')].

    self
```

```
    affichageDefautCentre: valeurDefaut;
    affichageDefautDroite: valeurDefaut;
    affichageDefautGauche: valeurDefaut.

affichageSaisieDonnee: aPane
  " Retourne l'interieur de la fenêtre de saisie de donnée "
   self addSubpane:
       (StaticText new
          centered;
          font: font1029;
          backColor: couleurFond;
          contents:'Prise de données';
          framingBlock:[:box | ((box leftTop) + 10)  extent: (((box width) - 30 ) @ 25)]).

 self addSubpane:
       (StaticText new
          leftJustified;
          font: font513;
          backColor: couleurFond;
          contents:'Point:';
          framingBlock:[:box | ((box width * 1 // 20) @ 70) extent: (30 @ 15)]).

 self addSubpane:
       ( FormattedStaticText new
          backColor: couleurFond;
          font: font513;
          rightJustified;
          foreColor: couleurPriseDonnee;
          setName:'Point';
          contents: ('1' asString);
          framingBlock:[:box | (((box width * 1 // 20) + 30) @ 70) extent: (20 @ 15)]).

 self addSubpane:
       (StaticText new
          leftJustified;
          font: font513;
          backColor: couleurFond;
          contents:'/ ';
          framingBlock:[:box | (((box width * 1 // 20) + 53) @ 70) extent: (8 @ 15)]).

 self addSubpane:
       (FormattedStaticText new
          leftJustified;
          font: font513;
          backColor: couleurFond;
          setName:'Npt';
          contents:((AxeDico at:'Npt') printString);
          framingBlock:[:box | (((box width * 1 // 20) + 60) @ 70) extent: (20 @ 15)]).


 self addSubpane:
       (StaticText new
          leftJustified;
          font: font513;
          backColor: couleurFond;
          contents:'Chaînage (m):';
```

```
        framingBlock:[:box | ((box width * 1 // 20) @ 90) extent: (100 @ 15)]).

 self addSubpane:
      (FormattedEntryField new
         setName:'Chainage';
         font: font513;
         acceptChainage;
         contents: (0 printString);
         backColor: couleurChamps;
         foreColor: couleurPriseDonnee;
         framingBlock:[:box | (((box width * 1 // 20) + 100) @ 90) extent: (50 @ 17)]).


 self addSubpane:
      (StaticText new
         leftJustified;
         font: font513;
         backColor: couleurFond;
         contents:'Ecart Roc-Tn. (m):';
         framingBlock:[:box | ((box width * 1 // 20) @ 110) extent: (100 @ 15)]).

self addSubpane:
      (FormattedEntryField new
         setName:'Ecart';
         acceptFloat;
         font: font513;
         setTextLimit: 8;
         contents:((AxeDico at:'RocTn') printString);
         backColor: couleurChamps;
         foreColor: couleurPriseDonnee;
         when: #textChanged perform: #affichageModificationEcart:;
         framingBlock:[:box | (((box width * 1 // 20) + 100) @ 110) extent: (50 @ 17)]).

 self addSubpane:
    (GroupBox new
      contents: ' Entrée des altitudes (m) ';
      font: font513;
      backColor: ClrPalegray;
      foreColor: ClrBlack;
      framingBlock:[:box |  (((box width * 1 // 20) + 170) @ 50) extent: (350 @ 150)]).

 self addSubpane:
      (StaticText new
         leftJustified;
         font: font513;
         backColor: couleurFond;
         contents:'Gauche';
         framingBlock:[:box | (((box width * 1 // 20) + 255) @ 70) extent: (55 @ 20)]).

self addSubpane:
      (StaticText new
         leftJustified;
         font: font513;
         backColor: couleurFond;
         contents:'Centre';
         framingBlock:[:box | (((box width * 1 // 20) + 335) @ 70) extent: (55 @ 20)]).
```

```
self addSubpane:
      (StaticText new
         leftJustified;
         font: font513;
         backColor: couleurFond;
         contents:'Droite';
         framingBlock:[:box | (((box width * 1 // 20) + 415) @ 70) extent: (55 @ 20)]).

self addSubpane:
      (StaticText new
         leftJustified;
         font: font513;
         backColor: couleurFond;
         contents:'Roc';
         framingBlock:[:box | (((box width * 1 // 20) + 200) @ 100) extent: (50 @ 20)]).

self addSubpane:
      (StaticText new
         leftJustified;
         font: font513;
         backColor: couleurFond;
         contents:'Terrain n.';
         framingBlock:[:box | (((box width * 1 // 20) + 180) @ 125) extent: (50 @ 20)]).

self addSubpane:
      (FormattedEntryField new
         setName:'RocG';
         acceptFloat;
         setTextLimit: 8;
         font: font513;
         backColor: couleurChamps;
         foreColor: couleurPriseDonnee;
         when: #textChanged perform: #affichageDefautGauche:;
         framingBlock:[:box | (((box width * 1 // 20) + 250) @ 100) extent: (60 @ 17)]).

self addSubpane:
      (FormattedEntryField new
         setName:'RocC';
         acceptFloat;
         setTextLimit: 8;
         font: font513;
         backColor: couleurChamps;
         foreColor: couleurPriseDonnee;
         when: #textChanged perform: #affichageDefautCentre:;
         framingBlock:[:box | (((box width * 1 // 20) + 330) @ 100) extent: (60 @ 17)]).

self addSubpane:
      (FormattedEntryField new
         setName:'RocD';
         acceptFloat;
         setTextLimit: 8;
         font: font513;
         backColor: couleurChamps;
         foreColor: couleurPriseDonnee;
         when: #textChanged perform: #affichageDefautDroite:;
```

```
            framingBlock:[:box | (((box width * 1 // 20) + 410) @ 100) extent: (60 @ 17)]).

    self addSubpane:
            (FormattedEntryField new
                setName:'TnG';
                acceptFloat;
                setTextLimit: 8;
                font: font513;
                backColor: couleurChamps;
                foreColor: couleurPriseDonnee;
                framingBlock:[:box | (((box width * 1 // 20) + 250) @ 125) extent: (60 @ 17)]).

    self addSubpane:
            (FormattedEntryField new
                setName:'TnC';
                acceptFloat;
                setTextLimit: 8;
                font: font513;
                backColor: couleurChamps;
                foreColor: couleurPriseDonnee;
                framingBlock:[:box | (((box width * 1 // 20) + 330) @ 125) extent: (60 @ 17)]).

    self addSubpane:
            (FormattedEntryField new
                setName:'TnD';
                acceptFloat;
                setTextLimit: 8;
                font: font513;
                backColor: couleurChamps;
                foreColor: couleurPriseDonnee;
                framingBlock:[:box | (((box width * 1 // 20) + 410) @ 125) extent: (60 @ 17)]).

     self addSubpane:
        (Button new
            label: 'Validation';
            font: font513;
            when: #clicked perform: #validationSaisieDonnee:;
            framingBlock:[:box | (((box width * 1 // 20) + 270) @170) extent: (80 @ 20)]).

    self addSubpane:
        (Button new
            label: 'Information';
            font: font513;
            when: #clicked perform: #aideSaisieDonnee:;
            framingBlock:[:box | (((box width * 1 // 20) + 360) @170) extent: (80 @ 20)]).

     (Ancien:=true)
        ifTrue:[ (self owner paneAt:'Point') contents:((AxeTravail size) printString)].

aideEcart: unBouton
    "  Message d'aide pour l'écart "

    | message |
    message:=
'Afin de faciliter l`utilisation, une différence
d`élévation moyenne entre le roc et le terrain
```

```
naturel peut être fixée.
Cette valeur permet l'enregistrement du niveau
du roc ou du terrain naturel par défaut.'.

   self affichageMessage: message


aideIdentification: unBouton
   " Message d'aide pour la fenêtre d'identification "

   | message |
   message:=
'Cette fenêtre permet l'enregistrement du nom
du projet, de l'axe considéré et de la largeur
retenue.
La largeur est la distance entre l'axe central et
un axe voisin ( droite ou gauche). L'unité de la
largeur est le mètre (m). Une largeur doit être
fixée par l'usager. Les noms de l'axe et du
projet sont facultatifs.'.

 self affichageMessage: message


aideLargeur: unBouton
   " Message d'aide pour la largeur "

   | message |
   message:='
La largeur est la distance entre l'axe central et
un axe voisin ( à droite ou à gauche). L'unité de
la largeur est le mètre (m). '.

   self affichageMessage: message

aideNombrePt: unBouton
   " Message d'aide pour le nombre de point "

   | message |
   message:='
Un nombre illimité de points peut-être enregistré.
Il est préférable d'utiliser un nombre variant
entre 5 et 10 points. Un nombre minimal de 3
points est requis.'.

   self affichageMessage: message


aideSaisieDonnee: unBouton
   " Message d'aide pour la fenêtre de saisie de données "

   | message |
   message:='
Cette fenêtre permet l'enregistrement des points
caractéristiques des axes central, gauche, droite.
Afin de faciliter l'utilisation, une différence d'élé-
```

-vation moyenne entre le roc et le terrain naturel
peut être fixée par l'utilisateur.
Après avoir fixé le chainage du point considéré,
l'altitude du roc de l'axe central doit être fixé
avant toutes autres altitudes. Les valeurs sont
déterminées à partir des paramètres par défaut.

L'ordre dans lequel les points sont entrés est
sans importance. Le logiciel enregistre les
points en ordre croissant en fonction du
chainage'.

```
    self affichageMessage: message
```

correctionLargeur

```
    | message valeurLargeur point|
message:=
'Opération de saisie en cours !'.

    (enregistrement=false) ifTrue:[ self affichageMessage: message. ^nil ].
    (ajout=true) ifTrue:[ ^nil ].
    (modification=true) ifTrue:[^nil].
    point:= (Prompter titre: 'Éditeur axe - Correction' prompt: 'Nouvelle largeur:' valeurDefaut: '20').
    (point=nil) ifTrue:[^nil].
    valeurLargeur:= point asNumber.
    AxeDico at:'Largeur' put:valeurLargeur.
```

dialogue1

```
    " Permet l'ouverture de la fenêtre d'identification "

 self effaceDialogueCourant.
    ( dialogue1 notNil and:[ dialogue1 mainView isHandleOk])
        ifTrue:[ dialogue1 mainView showWindow.
                dialogueCourant:= dialogue1]
        ifFalse:[ dialogueCourant:= dialogue1:= AxeEditeurIdentification new openIdentification: self].
```

ecran0: unEcran

```
    (presentation= nil) ifTrue:[].
    (presentation='identification') ifTrue:[ self dialogue1 ].
```

ecran1: unEcran

```
    (presentation='saisieDonnee') ifTrue:[ self affichageSaisieDonnee: self].
```

ecranListeChoix

```
    | valeurChoisie dico |
    valeurChoisie:= (listPane  getValue).
    dico:= (AxeTravail at: valeurChoisie).

    (valeurChoisie=nil)
    ifFalse:[  (self owner paneAt:'Ch') contents:  (dico at:2).
(self owner paneAt:'Rgr') contents: (dico at:3).
```

```
 (self owner paneAt:'Rcr') contents: (dico at:4).
 (self owner paneAt:'Rdr') contents: (dico at:5).
 (self owner paneAt:'Tgr') contents: (dico at:6).
(self owner paneAt:'Tcr') contents: (dico at:7).
 (self owner paneAt:'Tdr') contents: (dico at:8)].

ecranListeContenu: unEcran
   " Permet l'affichage de la liste de points dans l'écran de Liste "

   | position |
   list:=(AxeTravail collect: [:num | position:=(AxeTravail indexOf:num).
                                        ((AxeTravail indexOf:(AxeTravail at:position)) printString)]).
unEcran contents:list.

ecranMiniContenu: unEcran
 self affichageMiniPoint: self.

ecranRappelContenu: unEcran
 self affichageRappelContenu: self.

graphique

| message |
message:=
'La visualisation graphique pourra être
effectué lorsque l`opération de saisie
sera complétée !'.
   (enregistrement=false) ifTrue:[ self affichageMessage: message. ^nil ].
   (vision=false) ifTrue:[self affichageMessage: 'Opération de correction en cours !'.^nil].

   AxeGraphique new openAxeGraphique

information

   self aideGenerale

informationSauvegarde
   | message |
   message:=
'« Hydram - Prolog » permet le stockage des
valeurs dans un fichier *.hyd utilisable dans
-Hydram-. Ce fichier ne peut pas être consulté
dans l`éditeur.

L`item « Hydram - OO » permet le stockage
des valeurs dans un fichier *.axe utilisable
dans le logiciel Objet et dans l`éditeur . Ce
fichier est inutilisable par Hydram (Prolog).

Le même nom de fichier peut être employé pour
les différentes sauvegardes. Les extensions
par défaut s`inscrivent automatiquement. '.

   self affichageMessage: message.
```

```
initialisationAncien
   " la variable Ancien prend la valeur false par défaut  "
    Ancien:=false.

initialisationDictionnaireAxe
   "  initialise une variable dictionnaire DicoAxe et un dictionnaire système
      AxeTravail : OrderedCollection"

   | collection |

   collection:= OrderedCollection new: self size.
   Smalltalk add: (Association key:#AxeTravail value: collection).

   AxeDico:= Dictionary new.

initialisationIndex
   " Initialise la valeur de la variable index à 1 "
   Index:=1.

initialisationVariableSaisie


   presentation:= 'saisieDonnee'.
   mur:= false.
   enregistrement:= false.
   ajout:= false.
   modification:=false.
   numeroPoint:= nil.
   vision:=true.
   (Ancien=true) ifTrue:[ enregistrement:=true.
                          mur:= true].


initWindowSize
   ^(Display width * 9 // 10 @  Display height * 9 // 10 )

menuEcran0: ecranAffichage
   " menu pour le contrôle de la fenêtre d'affichage - ecran1 "

   ecranAffichage setMenu: ((Menu
      labels: '&Nouvel axe\&Consultation axe\&Information...' withCrs
      lines: #(2)
      selectors: #(nouveau ouverture information))
      title: '&Opérations Axe';
      yourself)

menuEcran1: ecranAffichage
   " menu pour le contrôle de la fenêtre d'affichage - ecran1 "

   ecranAffichage setMenu: ((Menu
      labels: '&Modification\&Ajout\&Retrait\Largeur' withCrs
      lines: #(3)
      selectors: #(pointModification pointAjout pointRetrait correctionLargeur))
      title: '&Corrections';
      yourself)
```

```
menuEcran2: ecranAffichage
   " menu pour le contrôle de la fenêtre d'affichage - ecran2 "

   ecranAffichage setMenu: ((Menu
      labels: '&Visualisation Graphique...' withCrs
      lines: #()
      selectors: #( graphique ))
      title: '&Graphique';
      yourself)

menuEcran3: ecranAffichage
   " menu pour le contrôle de la fenêtre d'affichage - ecran3 "

   ecranAffichage setMenu: ((Menu
      labels: 'Information...' withCrs
      lines: #()
      selectors: #(information))
      title: 'In&formation';
      yourself)

menuEcran4: ecranAffichage
   " menu pour le contrôle de la fenêtre d'affichage - ecran2 "

   ecranAffichage setMenu: ((Menu
      labels: 'Hydram - Prolog...\Hydram - Objet\information... ' withCrs
      lines: #(2)
      selectors: #( sauvegardeHydram sauvegardeEditeur informationSauvegarde))
      title: 'Sauvegarde';
      yourself)


nouveau

   presentation:='identification'.
   self changed: #ecran0:.

open
   " Ouverture fenêtre de départ "

   | ecran |

   self
      initialisationDictionnaireAxe;
      initialisationIndex;
      initialisationFont;
      initialisationCouleur;
      initialisationPresentation;
      initialisationAncien.


   self
      owner: self;
      foreColor: couleurCarac;
      labelWithoutPrefix: 'Hydram - Editeur d`axe'.
```

```
    self addSubpane:
      ((ecran:= StaticPane new)
        owner: self;
        setName: #ecran0: ;
        when: #needsMenu send: #menuEcran0: to:self with: ecran;
        when: #needsContents send: #ecran0: to:self with: ecran;
        framingBlock: [:box | box]).

      self openWindow

openSaisieDonnee
    " Fenêtre principale avec différents écrans "

    | ecran question |

    self
      owner: self;
      backColor: couleurFond;
      foreColor: couleurCarac;
      labelWithoutPrefix: 'Hydram - Editeur d'axe'.

    self initialisationFont;
        initialisationCouleur;
        initialisationVariableSaisie.

    self addSubpane:
      ((ecran:= StaticPane new)
        owner: self;
        setName: #ecran1: ;
        backColor: couleurFond;
        when: #needsMenu send: #menuEcran1: to:self with: ecran;
        when: #needsContents send: #ecran1: to:self with: ecran;
        framingBlock: [:box | (box leftTop)
                              rightBottom: (box rightTop down:(box height * 7 // 12))]).

    self addSubpane:
      ((ecran:= StaticBox new)
        owner: self;
        blackFrame;
        setName: #ecranMini;
        when: #needsMenu send: #menuEcran2: to:self with: ecran;
        when: #needsContents send: #ecranMiniContenu: to: self with: ecran;
        framingBlock: [:box | (box leftTop down: (box height * 7 // 12))
                                rightBottom:((box width ) @ ((box height * 7 // 12) + 25))]).

    self addSubpane:
      ((listPane:= ListPane new)
        owner: self;
        backColor: couleurFond;
        foreColor: couleurPriseDonnee;
        setName: #ecranListe;
        when: #needsMenu send: #menuEcran4: to: self with: listPane;
        when: #needsContents send:#ecranListeContenu: to: self with: listPane;
        when: #changed: send: #ecranListeChoix to: self;
        framingBlock: [:box | (box leftTop down: ((box height * 7 // 12) + 25))
                              rightBottom:((box width * 1 // 3) @ (box height))]).
```

```
    self addSubpane:
      ((ecran:= StaticBox new)
        owner: self;
        blackFrame;
        backColor: couleurFond;
        setName: #ecranRappel;
        when: #needsContents send: #ecranRappelContenu: to: self with: ecran;
        when: #needsMenu send: #menuEcran3: to:self with: ecran;
        framingBlock: [:box | ((box width * 1 // 3) @ ( (box height * 7 // 12) + 25))
                              rightBottom:(box width @ box height)]).

    self openWindow

ouverture

    Recup:=true.
    self recuperationAxe.
    (Recup=false)  ifTrue:[ ^nil ].
    self mainView close.
    Ancien:=true.
    AxeEditeur new openSaisieDonnee.


pointAjout
    "  permet d'ajouter un point  "

| message |
message:=
'L`ajout de points pourra être effectué lorsque
l`opération de saisie sera complétée !'.
    (enregistrement=false) ifTrue:[ self affichageMessage: message. ^nil ].
    (modification=true) ifTrue:[^nil].
    vision:=false.
    ajout:=true.
    (enregistrement=true) ifTrue:[ self affichageMessage:'Entrez le chaînage et les altitudes'.
self initialisationChamps.
 (self owner paneAt:'Npt') contents:(((AxeTravail size) + 1) printString).
(self owner paneAt:'Point') contents:(((AxeTravail size) + 1) printString)].

pointModification
    "  permet d'effectuer une modification sur un point enregistré  "

    | message taille point|
message:=
'La modification des points pourra être effectuée
lorsque l`opération de saisie sera complétée !'.

    (enregistrement=false) ifTrue:[ self affichageMessage: message. ^nil ].
    (ajout=true) ifTrue:[ ^nil ].
    vision:=false.
    point:= (Prompter titre: 'Éditeur axe - Correction' prompt: 'Point à modifier:' valeurDefaut: '1').
    (point=nil) ifTrue:[^nil].
    numeroPoint:= point asNumber.
    taille:= AxeTravail size.
```

```
   (numeroPoint > taille) ifTrue:[ self affichageMessage:'Point non enregistré !'. ^nil].
   (numeroPoint = 0) ifTrue:[ self affichageMessage:'Point non enregistré !'. ^nil].
   modification:=true.
   (enregistrement=true) ifTrue:[ self affichageMessage:'Entrez le chaînage et les altitudes'.
                                  self initialisationChamps.
                                  (self owner paneAt:'Npt') contents:((AxeTravail size) printString).
                                  (self owner paneAt:'Point') contents:( numeroPoint printString)].

pointModificationStockage
   " permet de stocker une modification sur un point enregistré "

  | collectionTampon rocG rocC rocD tnG tnC tnD chainage |
   collectionTampon:= OrderedCollection new: self size.
   chainage:=((self owner paneAt:'Chainage') contents).
   rocG:=((self owner paneAt:'RocG') contents).
   rocC:=((self owner paneAt:'RocC') contents).
   rocD:=((self owner paneAt:'RocD') contents).
   tnG:=((self owner paneAt:'TnG') contents).
   tnC:=((self owner paneAt:'TnC') contents).
   tnD:=((self owner paneAt:'TnD') contents).

   collectionTampon add: numeroPoint.
   collectionTampon add: chainage.
   collectionTampon add: rocG.
   collectionTampon add: rocC.
   collectionTampon add: rocD.
   collectionTampon add: tnG.
   collectionTampon add: tnC.
   collectionTampon add: tnD.

   AxeTravail at: numeroPoint put: collectionTampon.

pointRetrait
   " permet d'enlever un point de la liste "

| taille point message pointTampon tailleAxeTravail |
message:='
Le retrait de certains points pourra être effectué
lorsque l'opération de saisie sera complétée !'.
   (enregistrement=false) ifTrue:[ self affichageMessage: message. ^nil ].
   (ajout=true) ifTrue:[^nil].
   (modification=true) ifTrue:[^nil].
   tailleAxeTravail:= AxeTravail size.
   (tailleAxeTravail < 4) ifTrue:[self affichageMessage:'Nombre de points minimal atteint'. ^nil].
   pointTampon:= (Prompter titre: 'Éditeur axe - Correction' prompt: 'Point à retirer:' valeurDefaut: '1') .
   (pointTampon=nil) ifTrue:[^nil].
   point:= pointTampon asNumber.
   taille:= AxeTravail size.

   (point > taille) ifTrue:[ self affichageMessage:'Point non enregistré !'. ^nil].
   (point = 0) ifTrue:[ self affichageMessage:'Point non enregistré !'. ^nil].

   AxeTravail removeIndex: point.
   self changed: #ecranListe.
   AxeDico at:'Npt' put: (AxeTravail size).
   (self owner paneAt:'Npt') contents:((AxeTravail size) printString).
```

```
   self initialisationChamps.
   (self owner paneAt:'Point') contents:((AxeTravail size) printString).

sauvegardeEditeur

   self sauvegardeAxeEditeur.


sauvegardeHydram

   self sauvegardeAxeHydram.

validationIdentification: unBouton
   "  Vérifications d'usages sur les entrées et stockages  "

   | passage |
   passage:= self verificationIdentification.
   (passage=false) ifFalse:[
   AxeDico at:'NomProjet' put: ((self owner paneAt:'NomProjet') contents).
   AxeDico at:'NomAxe' put: ((self owner paneAt:'NomAxe') contents).
   AxeDico at:'Largeur' put: ((self owner paneAt:'Largeur') contents asFloat).
   AxeDico at:'RocTn' put: ((self owner paneAt:'RocTn') contents asFloat).
   AxeDico at:'Npt' put: ((self owner paneAt:'Npt') contents asNumber).
   self mainView close.
   AxeEditeur new openSaisieDonnee].


validationSaisieDonnee: unbouton
   "  Vérifications d'usages sur les entrées et stockages  "

   | passage taille tailleAvant |

   (mur=false) ifTrue:[
      passage:= self verificationSaisieDonnee.
      (passage=false) ifTrue:[^nil].
      taille:= AxeDico at:'Npt'.
      (Index=taille)
         ifFalse:[self enregistrementDonnee.
                  self changed: #ecranListe.
                  self initialisationChamps.
                  Index:= Index+1]
         ifTrue:[ self enregistrementDonnee.
                  self changed: #ecranListe.
                  enregistrement:= true.
                  self affichageMessage: 'Enregistrement terminé. Ne pas
oublier de sauvegarder !'.
                  mur:= true ]].

   (ajout=true) ifTrue:[
      passage:= self verificationSaisieDonnee.
      (passage=false) ifTrue:[^nil].
      self enregistrementDonnee.
      self changed: #ecranListe.
      AxeDico at:'Npt' put: (AxeTravail size).
      (AxeTravail at:(AxeTravail size)) at:1 put:(AxeTravail size).
      ajout:=false].
```

```
    (modification=true) ifTrue:[
        passage:= self verificationSaisieDonnee.
        (passage=false) ifTrue:[^nil].
        self pointModificationStockage.
        self changed: #ecranListe.
        modification:= false].

    vision:=true.

verificationChainage: unePosition
    " vérification chainage "

    | valeur |
        AxeTravail do:[:element |  valeur:=((element at:2) asNumber).
                                   (unePosition= (AxeTravail indexOf: element)) ifFalse:[
                                   (valeur= ((self owner paneAt:'Chainage') contents asNumber))
                                       ifTrue:[(self affichageMessage:
'Ce chaînage représente un point dont les
caractéristiques sont déjà enregistrées !').
^false]]].

verificationIdentification
    | nomProjet nomAxe largeur rocTn npt |

    nomProjet:=((self owner paneAt:'NomProjet') contents).
    nomAxe:=((self owner paneAt:'NomAxe') contents).
    largeur:=((self owner paneAt:'Largeur') contents).
    rocTn:=((self owner paneAt:'RocTn') contents).
    npt:=((self owner paneAt:'Npt') contents).
    (nomProjet='') ifTrue:[(self affichageMessage:'Nom du projet manquant !').
                           ^false].
    (nomAxe='') ifTrue:[(self affichageMessage:'Nom de l`axe manquant !').
                        ^false].
    (largeur='') ifTrue:[(self affichageMessage:'Largeur manquante !').
                         ^false].
    (rocTn='') ifTrue:[(self affichageMessage:'Écart entre le roc et le terrain naturel manquant !').
                       ^false].
    (npt='') ifTrue:[(self affichageMessage:'Nombre de points manquant !').
                     ^false].
    (((self owner paneAt:'Largeur') contents asNumber) < 0)
        ifTrue:[(self affichageMessage:'La largeur ne peut être négative !').
                        ((self owner paneAt:'Largeur') contents:'').
                        ^false].
    (((self owner paneAt:'RocTn') contents asNumber) < 0)
        ifTrue:[(self affichageMessage:'l`écart entre le roc et le terrain naturel ne peut être négatif !').
                        ((self owner paneAt:'RocTn') contents:'').
                        ^false].
    (((self owner paneAt:'Npt') contents asNumber) < 3)
        ifTrue:[(self affichageMessage:'Un nombre minimal de 3 points est requis !').
                        ((self owner paneAt:'Npt') contents:'').
                        ^false].
verificationSaisieDonnee

    |rocG rocC rocD tnG tnC tnD chainage valeur verifChainage|
    rocG:=((self owner paneAt:'RocG') contents).
```

```
rocC:=((self owner paneAt:'RocC') contents).
rocD:=((self owner paneAt:'RocD') contents).
tnG:=((self owner paneAt:'TnG') contents).
tnC:=((self owner paneAt:'TnC') contents).
tnD:=((self owner paneAt:'TnD') contents).
chainage:=((self owner paneAt:'Chainage') contents).

" *** vérification présence *** "
(rocG='') ifTrue:[(self affichageMessage:'Donnée manquante !'). ^false].
(rocC='') ifTrue:[(self affichageMessage:'Donnée manquante !'). ^false].
(rocD='') ifTrue:[(self affichageMessage:'Donnée manquante !'). ^false].
(tnG='') ifTrue:[(self affichageMessage:'Donnée manquante !'). ^false].
(tnC='') ifTrue:[(self affichageMessage:'Donnée manquante !'). ^false].
(tnD='') ifTrue:[(self affichageMessage:'Donnée manquante !'). ^false].
(chainage='') ifTrue:[(self affichageMessage:'Chaînage manquant !'). ^false].

" *** vérification positivité *** "
(((self owner paneAt:'RocG') contents asNumber) < 0)
   ifTrue:[(self affichageMessage:'Les altitudes ne peuvent être négatives !'). ^false].
(((self owner paneAt:'RocC') contents asNumber) < 0)
   ifTrue:[(self affichageMessage:'Les altitudes ne peuvent être négatives !'). ^false].
(((self owner paneAt:'RocD') contents asNumber) < 0)
   ifTrue:[(self affichageMessage:'Les altitudes ne peuvent être négatives !'). ^false].
(((self owner paneAt:'TnG') contents asNumber) < 0)
   ifTrue:[(self affichageMessage:'Les altitudes ne peuvent être négatives !'). ^false].
(((self owner paneAt:'TnC') contents asNumber) < 0)
   ifTrue:[(self affichageMessage:'Les altitudes ne peuvent être négatives !'). ^false].
(((self owner paneAt:'TnD') contents asNumber) < 0)
   ifTrue:[(self affichageMessage:'Les altitudes ne peuvent être négatives !'). ^false].

" *** vérification altitudes *** "
(((self owner paneAt:'TnG') contents asNumber) < ((self owner paneAt:'RocG') contents asNumber))
   ifTrue:[(self affichageMessage:'L`altitude du terrain naturel doit être supérieure à celle du roc !').
         ^false].
(((self owner paneAt:'TnC') contents asNumber) < ((self owner paneAt:'RocC') contents asNumber))
   ifTrue:[(self affichageMessage:'L`altitude du terrain naturel doit être supérieure à celle du roc !').
         ^false].
(((self owner paneAt:'TnD') contents asNumber) < ((self owner paneAt:'RocD') contents asNumber))
   ifTrue:[(self affichageMessage:'L`altitude du terrain naturel doit être supérieure à celle du roc !').
         ^false].

verifChainage:= true.
verifChainage:=(self verificationChainage: numeroPoint).
(verifChainage=false) ifTrue:[^false].
```

```
AxeOperateur subclass: #AxeGraphique
 instanceVariableNames:
  ' graphPane listPane list typeAxe crayon couleurBordure couleurGraphique couleurPoint
pointsGraphique graphPoints graphPoints2 couleurGraphique2 couleurPoint2 lastPoint '
 classVariableNames: ''
 poolDictionaries:
  ' ColorConstants '
```

```
affichageGraphique
   " Présente le graphique sélectionné "

   | bordure x y largeurDisponible hauteurDisponible deltaX deltaY taille
   yMax yMin collecTampon xi yi xr yr conversionX conversionY xo|

   self effaceCrayon.
   ((list size)=0) ifTrue:[^nil].
   self repartitionPointsGraphique.
   graphPoints:= OrderedCollection new: self size.
   graphPoints2:= OrderedCollection new: self size.
   taille:= pointsGraphique size.
   bordure:= graphPane rectangle insetBy: 10@10.

   crayon foreColor: couleurBordure;
            down;
            rectangle: bordure;
            foreColor: couleurGraphique;
            setFillColor: couleurGraphique.
   bordure:= bordure insetBy: 5@5.

   collecTampon:= OrderedCollection new: self size.
   pointsGraphique do:[ :valeur | collecTampon add: (valeur at:3)].
   yMax:= (collecTampon hydramCollectionValeurMax).
   yMin:= (collecTampon hydramCollectionValeurMin).
   largeurDisponible:= bordure width.
   hauteurDisponible:= bordure height.
   xo:= ((pointsGraphique at:1) at:1).
   deltaX:= (((pointsGraphique at:taille) at:1) - ((pointsGraphique at:1) at:1)).
   deltaY:= (yMax ).
   conversionX:= (largeurDisponible / deltaX).
   conversionY:= (hauteurDisponible / deltaY).
   xi:= bordure left.
   yi:= bordure bottom.

   1 to: taille do: [:element |  xr:= ((pointsGraphique at: element) at:1).
x:=  xi + (((xr - xo)  * conversionX) truncated).
yr:=((pointsGraphique at: element) at:3).
y:= (bordure bottom) up: ((yr * conversionY) truncated).
 (element=1) ifTrue:[ crayon place: x@y ]
ifFalse:[ crayon goto: x@y].
crayon circleFilled:5.
graphPoints add: x@y].

  crayon foreColor: couleurGraphique2;
            setFillColor: couleurGraphique2.

   1 to: taille do: [:element |  xr:= ((pointsGraphique at: element) at:1).
x:=  xi + (((xr - xo) * conversionX) truncated).
                                 yr:=((pointsGraphique at: element) at:2).
y:= (bordure bottom) up: ((yr * conversionY) truncated).
 (element=1) ifTrue:[ crayon place: x@y ]
ifFalse:[ crayon goto: x@y].
crayon circleFilled:5.
graphPoints2 add: x@y].
```

```
    lastPoint ~=nil
        ifTrue: [self eclairagePoint].

eclairagePoint

    crayon setFillColor: couleurPoint;
            place: (graphPoints at:(lastPoint := (self paneAt: #ecranListe2) selection));
            circleFilled: 5;
            setFillColor: couleurGraphique.
 crayon setFillColor: couleurPoint2;
            place: (graphPoints2 at:(lastPoint := (self paneAt: #ecranListe2) selection));
            circleFilled: 5;
            setFillColor: couleurGraphique2.


ecranDescriptionContenu: unEcran

    self affichageDescriptionContenu: unEcran


ecranListeAffichage: unEcran
    " Permet l'affichage de la liste de points dans l'écran de liste "

    | position |
    (typeAxe= nil) ifTrue:[^nil].
    list:=(AxeTravail collect: [:num | position:=(AxeTravail indexOf:num).
                                        ((AxeTravail indexOf:(AxeTravail at:position)) printString)]).

    unEcran contents:list.

ecranListeChangement
    " permet les changements lorsque un point est choisie "

    | valeurChoisie dico |
    valeurChoisie:= (listPane getValue).
    (valeurChoisie=nil) ifTrue:[ ^nil].
    dico:= (AxeTravail at: valeurChoisie).

    (typeAxe='Gauche') ifTrue:[ (self owner paneAt:'AxeChoisi') contents: (typeAxe).
                                (self owner paneAt:'ChPoint') contents: (dico at:2).
                                (self owner paneAt:'RocPoint') contents: (dico at:3).
 (self owner paneAt:'TnPoint') contents: (dico at:6)].
 (typeAxe='Centre') ifTrue:[ (self owner paneAt:'AxeChoisi') contents: (typeAxe).
 (self owner paneAt:'ChPoint') contents: (dico at:2).
 (self owner paneAt:'RocPoint') contents: (dico at:4).
 (self owner paneAt:'TnPoint') contents: (dico at:7)].
 (typeAxe='Droite') ifTrue:[   (self owner paneAt:'AxeChoisi') contents: (typeAxe).
 (self owner paneAt:'ChPoint') contents: (dico at:2).
 (self owner paneAt:'RocPoint') contents: (dico at:5).
 (self owner paneAt:'TnPoint') contents: (dico at:8)].

    lastPoint ~=nil
       ifTrue:[ crayon setFillColor: couleurGraphique;
 place: (graphPoints at: lastPoint);
 circleFilled: 5.
 crayon setFillColor: couleurGraphique2;
```

```
place: (graphPoints2 at: lastPoint);
circleFilled: 5].

self eclairagePoint.

effaceCrayon
   " nettoie l'écran  "

   crayon erase.
graphAxeCentre

   lastPoint:= nil.
   typeAxe:='Centre'.
   (self owner paneAt:'AxeChoisi') contents: typeAxe.
   (self owner paneAt:'ChPoint') contents: ''.
   (self owner paneAt:'RocPoint') contents: ''.
   (self owner paneAt:'TnPoint') contents: ''.
   self changed: #ecranListe2.
   self affichageGraphique.

graphAxeDroite

   lastPoint:= nil.
   typeAxe:='Droite'.
   (self owner paneAt:'AxeChoisi') contents: typeAxe.
   (self owner paneAt:'ChPoint') contents: ''.
   (self owner paneAt:'RocPoint') contents: ''.
   (self owner paneAt:'TnPoint') contents: ''.

   self changed: #ecranListe2.
   self affichageGraphique.

graphAxeGauche

   lastPoint:= nil.
   typeAxe:='Gauche'.
   (self owner paneAt:'AxeChoisi') contents: typeAxe.
   (self owner paneAt:'ChPoint') contents: ''.
   (self owner paneAt:'RocPoint') contents: ''.
   (self owner paneAt:'TnPoint') contents: ''.
   self changed: #ecranListe2.
   self affichageGraphique.



groupeBouton
   "  ajoute un groupe de bouton pour gérer l'exécution des graphiques "

   | groupPane |
   self addSubpane:
      (groupPane:= GroupPane new
        framingBlock: [:box | (box leftTop down: (box height * 2 // 3))
                              rightBottom:((box width ) @ ((box height * 2 // 3) + 20))]).

   groupPane addSubpane:
      (Button new
```

```
        label:'Axe Gauche';
        when: #clicked send: #graphAxeGauche to: self;
        framingBlock: [:box | (box leftTop right: (box width * 1 // 5))
                                    extentFromLeftTop: (box width // 5 @ box height)]).

    groupPane addSubpane:
      (Button new
        label:'Axe Central';
        defaultPushButton;
when: #clicked send: #graphAxeCentre to: self;
framingBlock: [:box |  (box leftTop right: (box width * 2 // 5))
extentFromLeftTop: (box width // 5 @ box height)]).


groupPane addSubpane:
 (Button new
label:'Axe Droite';
when: #clicked send: #graphAxeDroite to: self;
        framingBlock: [:box |  (box leftTop right: (box width * 3 // 5))
                                    extentFromLeftTop: (box width // 5 @ box height)]).
```

informationGraphique

```
  | message |

message:=
'Cet écran permet la visualisation des profils de
sol enregistrés. Pour chaque axe (droite, central
et gauche), les profils du roc et du terrain naturel
sont présentés. De plus les caractéristiques de
chaque point sont présentées dans les écrans
inférieurs.

Les trois boutons d'axe -gauche, central, droite-
permettent l'affichage de l'axe choisi.

L'écran inférieur gauche permet la présentation
du point sur le graphique et de ses
caractéristiques sur l'écran inférieur droit.

Pour apporter des modifications, choisir l'item
- Retour fenêtre saisie - du menu Opérations.'.

    self affichageMessage: message.
```

informationSauvegarde

```
    | message |
    message:=
'L'item « utilisation Hydram » permet le stockage
des valeurs dans un fichier *.hyd utilisable dans
-Hydram-. Ce fichier ne peut pas être consulté dans
l'éditeur.

L'item « consultation dans Editeur » permet le
```

```
stockage des valeurs dans un fichier *.axe non
utilisable dans -Hydram-. Ce fichier peut être
consulté et modifié dans l'éditeur.

Le même nom de fichier peut être employé pour
les différentes sauvegardes. Les extensions
s'inscrivent automatiquement. '.

    self affichageMessage: message.

initCouleurGraphique

    couleurBordure:= ClrBlack.
    couleurGraphique:= ClrDarkblue.
    couleurPoint:= ClrRed.
    couleurGraphique2:= ClrDarkred.
    couleurPoint2:= ClrBlue.

initVariable

    typeAxe:= nil.


initWindowSize
    ^(Display width * 9 // 10 @  Display height * 9 // 10 )


instrumentGraph
    "  designe un type d'instrument de dessin "

    crayon:=  graphPane pen.


menuEcranGraphique: ecranAffichage
    " menu pour le contrôle de la fenêtre de graphique "

    ecranAffichage setMenu: ((Menu
        labels: 'Retour fenêtre &saisie...\&Information' withCrs
        lines: #(1)
        selectors: #( retourFenetreSaisie informationGraphique))
        title: 'Opérations ';
        yourself)

menuEcranGraphique2: ecranAffichage
    " menu pour le contrôle de la fenêtre de graphique "

  ecranAffichage setMenu: ((Menu
        labels: '&utilisation Hydram...\&consultation dans Editeur...\information... ' withCrs
        lines: #(2)
        selectors: #( sauvegardeHydram sauvegardeEditeur informationSauvegarde))
        title: 'Sauvegarde';
        yourself)

openAxeGraphique
    " Fenêtre principale pour graphique avec différents écrans "
```

```
    | ecran |
    self
        owner: self;
        backColor: couleurFond;
        foreColor: couleurCarac;
        labelWithoutPrefix: 'Hydram - Editeur de Graphique'.

    self initialisationFont;
          initialisationCouleur;
          initVariable;
          initCouleurGraphique;
          ordreCroissantAxeTravail.

    self addSubpane:
        ((graphPane:= GraphPane new)
        owner: self;
        style: GraphPane noScrollbarsFrameStyle;
        setName: #graphPane;
        when: #needsContents send: #instrumentGraph to: self;
when: #display send: #affichageGraphique to: self;
framingBlock: [:box | (box leftTop)
rightBottom: (box rightTop down:(box height * 2 // 3))]).
self groupeBouton.
self addSubpane:
      ((listPane:= ListPane new)
          setName: #ecranListe2;
          backColor: couleurFond;
          foreColor: couleurPriseDonnee;
          when: #needsMenu send: #menuEcranGraphique: to: self with: listPane;
          when: #needsContents send: #ecranListeAffichage: to:self with: listPane;
          when: #changed: send: #ecranListeChangement to: self;
          framingBlock: [:box | (box leftTop down: ((box height * 2 // 3) + 20))
                                    rightBottom: ((box width * 1 // 3) @ (box height))]).

    self addSubpane:
        ((ecran:= StaticBox new)
          owner: self;
          blackFrame;
          backColor: couleurFond;
          setName: #ecranDescription;
          when: #needsMenu send: #menuEcranGraphique2: to: self with: ecran;
          when: #needsContents send:#ecranDescriptionContenu: to: self with:ecran;
          framingBlock: [:box | ((box width * 1 // 3) @ ((box height * 2 // 3) + 20))
                                    rightBottom:(box width @ box height)]).

    self openWindow

repartitionPointsGraphique

    | numero collectionTampon taille essai|

    numero:= 1.
    taille:= list size.
    pointsGraphique:= OrderedCollection new: self size.
```

```
    (typeAxe='Gauche') ifTrue:[
        [numero>taille] whileFalse:[ collectionTampon:= OrderedCollection new: self size.
                                     collectionTampon add:(((AxeTravail at:numero) at:2) asNumber).
                                     collectionTampon add:(((AxeTravail at:numero) at:3) asNumber).
                                     collectionTampon add:(((AxeTravail at:numero) at:6) asNumber).
                                     pointsGraphique add:collectionTampon.
numero:= numero+1]].

    (typeAxe='Centre') ifTrue:[
        [numero>taille] whileFalse:[ collectionTampon:= OrderedCollection new: self size.
                                     collectionTampon add:(((AxeTravail at:numero) at:2) asNumber).
                                     collectionTampon add:(((AxeTravail at:numero) at:4) asNumber).
                                     collectionTampon add:(((AxeTravail at:numero) at:7) asNumber).
                                     pointsGraphique add:collectionTampon.
numero:= numero+1]].

    (typeAxe='Droite') ifTrue:[
        [numero>taille] whileFalse:[ collectionTampon:= OrderedCollection new: self size.
                                     collectionTampon add:(((AxeTravail at:numero) at:2) asNumber).
                                     collectionTampon add:(((AxeTravail at:numero) at:5) asNumber).
                                     collectionTampon add:(((AxeTravail at:numero) at:8) asNumber).
                                     pointsGraphique add:collectionTampon.
numero:= numero+1]].


retourFenetreSaisie

    self mainView close.

sauvegardeEditeur

    self sauvegardeAxeEditeur.

sauvegardeHydram

    self sauvegardeAxeHydram.
```

ANNEXE 6

CARTOGRAPHIE DU SITE GB3

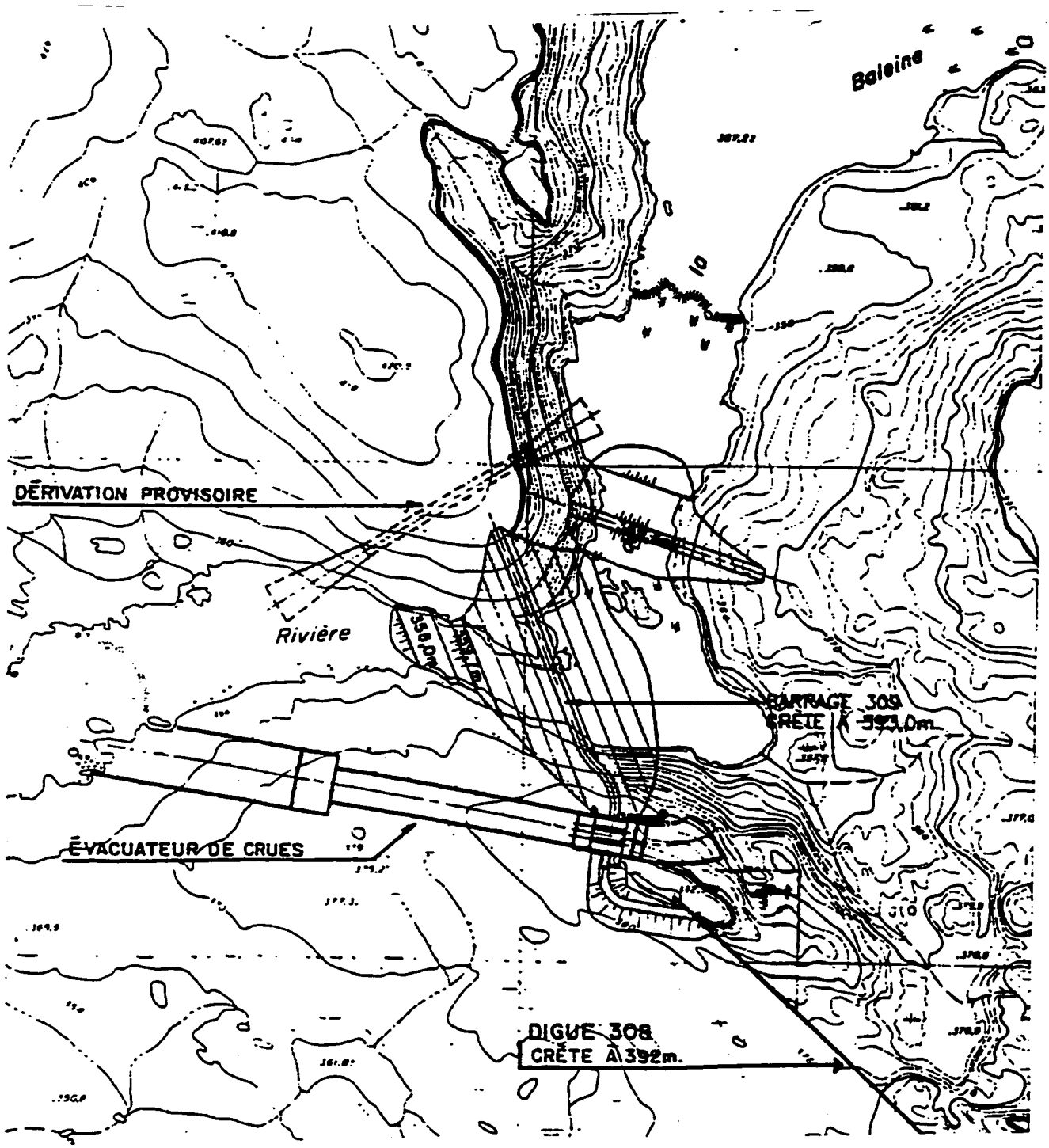
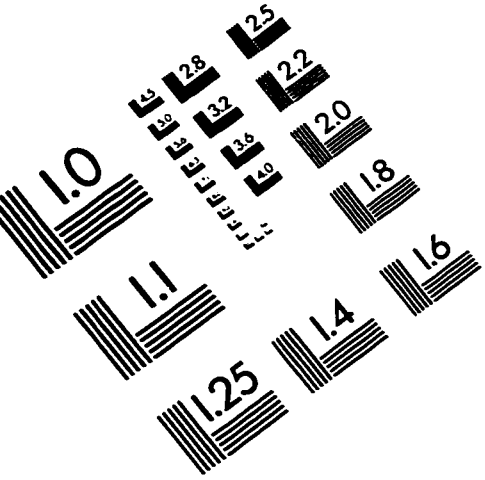
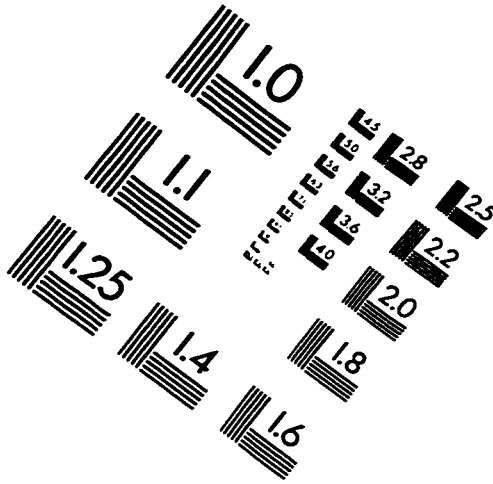
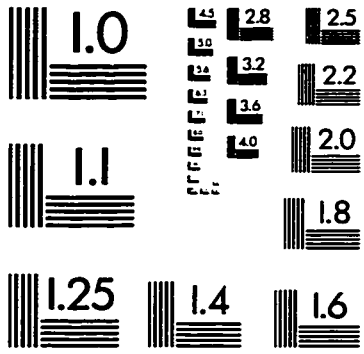
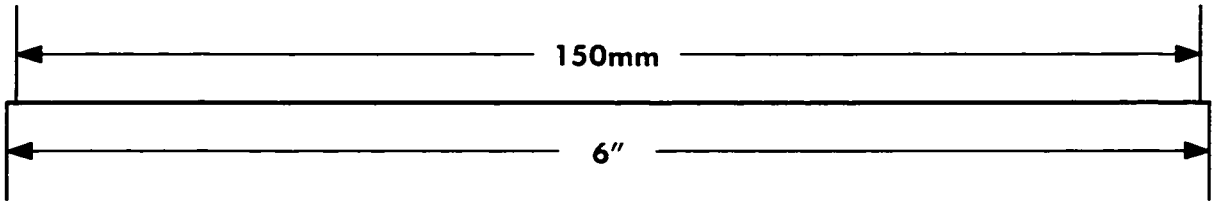


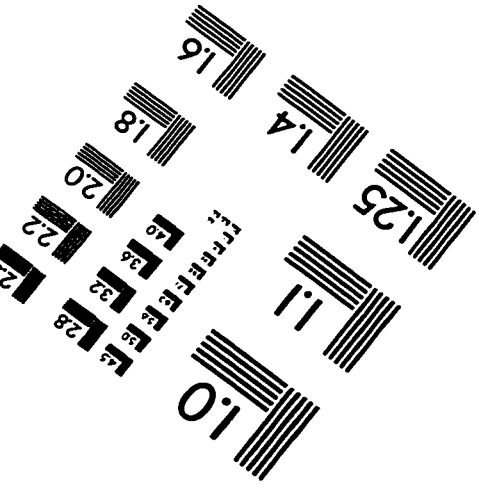
IMAGE EVALUATION TEST TARGET (QA-3)

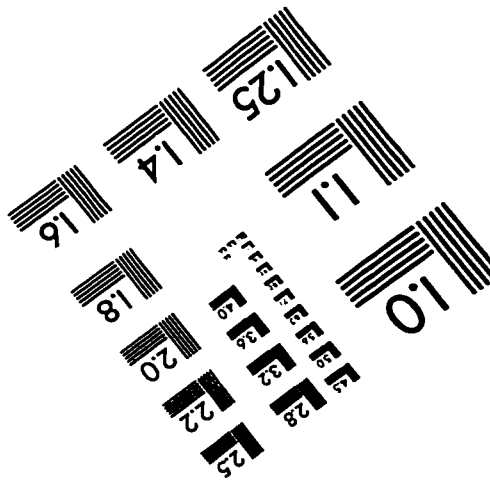












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989