Livret - 9

Programmation par événements

Programmation visuelle, IHM
&
Modèle RAD

RM di scala

Cours informatique programmation

Rm di Scala - http://www.discala.net

9 : Programmation événementielle et visuelle

Plan du chapitre:

Introduction

1. Programmation visuelle basée sur les pictogrammes

2. Programmation orientée événements

3. Normalisation du graphe événementiel

le graphe événementiel *arcs et sommets*
les diagrammes d'états UML *réduits*

4. Tableau des actions événementielles

5. Interfaces liées à un graphe événementiel

6. Avantages et modèle de développement RAD visuel

le modèle de la spirale *(B.Boehm)*
le modèle incrémental

1. Programmation visuelle basée sur les pictogrammes

Le développement visuel rapide d'application est fondé sur le concept de programmation visuelle associée à la montée en puissance de l'utilisation des Interactions Homme-Machine (IHM) dont le dynamisme récent ne peut pas être méconnu surtout par le débutant. En informatique les systèmes MacOs, Windows, les navigateurs Web, sont les principaux acteurs de l'ingénierie de l'IHM. Actuellement dans le développement d'un logiciel, un temps très important est consacré à l'ergonomie et la communication, cette part ne pourra que grandir dans un avenir proche; car les utilisateurs veulent s'adresser à des logiciels efficaces (ce qui va de soi) mais aussi conviviaux et faciles d'accès.

Les développeurs ont donc besoin d'avoir à leur disposition des produits de développement adaptés aux nécessités du moment. A ce jour la programmation visuelle est une des réponses à cette attente des développeurs.

La programmation visuelle au tout début a été conçue pour des personnes n'étant pas des programmeurs en basant ses outils sur des manipulations de pictogrammes.
Le raisonnement communément admis est qu'un dessin associé à une action élémentaire est plus porteur de sens qu'une phrase de texte.

A titre d'exemple ci-dessous l'on enlève le "fichier.bmp" afin de l'effacer selon deux modes de communication avec la machine: utilisation d'icônes ou entrée d'une commande textuelle.

Effacement avec un langage d'action visuelle (souris)

Action :	**Réponse :**
	

Effacement avec un langage textuel (clavier)

Action :	**Réponse :**
del c:\Exemple\Fichier.bmp	| ?

Nous remarquons donc déjà que l'interface de communication MacOs, Windows dénommée "bureau électronique" est en fait un outil de programmation de commandes systèmes.

Un langage de programmation visuelle permet "d'écrire" la partie communication d'un programme uniquement avec des dessins, diagrammes, icônes etc... Nous nous intéressons aux systèmes RAD (Rapid Application Development) visuels, qui sont fondés sur des langages objets à bases d'icônes ou pictogrammes. Visual Basic de MicroSoft est le premier RAD visuel a avoir été commercialisé dès 1991, il est fondé sur un langage Basic étendu incluant des objets étendus en VB.Net depuis 2001, puis dès 1995 Delphi le premier RAD visuel de Borland fondé sur Pascal objet, puis actuellement toujours de Borland : C++Builder RAD visuel fondé sur le langage C++ et Jbuilder, NetBeans RAD visuel de Sun fondés sur le langage Java, Visual C++, Visual J++ de Microsoft, puis Visual studio de Microsoft etc...

Le développeur trouve actuellement, une offre importante en outil de développement de RAD visuel y compris en open source. Nous proposons de définir un langage de RAD visuel ainsi :

Un **langage visuel** dans un RAD visuel est un **générateur de code source** du langage de base qui, derrière chaque action visuelle (dépôt de contrôle, click de souris, modifications des propriétés, etc...) engendre des lignes de code automatiquement et d'un manière transparente au développeur.

Des outils de développement tels que Visual Basic, Delphi ou C# sont adaptés depuis leur création à la programmation visuelle pour débutant. Toutefois l'efficacité des dernières versions a étendu leur champ au développement en général et dans l'activité industrielle et commerciale avec des versions "entreprise" pour VB et "Architect "pour Delphi et les versions .Net.

En outre, le système windows est le plus largement répandu sur les machines grand public (90% des PC vendus en sont équipés), il est donc très utile que le débutant en programmation sache utiliser un produit de développement (rapide si possible) sur ce système.

Proposition :

Nous considérons dans cet ouvrage, la programmation visuelle à la fois comme une **fin** et comme un **moyen**.

La programmation visuelle est sous-tendue par la réactivité des programmes en réponse aux actions de l'utilisateur. Il est donc nécessaire de construire des programmes qui répondent à des sollicitations externes ou internes et non plus de programmer séquentiellement (ceci est essentiellement dû aux architectures de Von Neumann des machines) : ces sollicitations sont appelées des événements.

Le concept de **programmation dirigée ou orientée par les événements** est donc la composante essentielle de la programmation visuelle.

Terminons cette présentation par 5 remarques sur le concept de RAD :

Utiliser un RAD simple mais puissant

Nous ne considérerons pas comme utile pour des débutants de démarrer la programmtion visuelle avec des RAD basés sur le **langage C++**. Du fait de sa large permissivité ce langage permet au programmeur d'adopter certaines **attitudes dangereuses** sans contrôle possible. Seul le programmeur confirmé au courant des pièges et des subtilités de la programmation et du langage, pourra exploiter sans risque la richesse de ce type de RAD.

Avoir de bonnes méthodes dès le début

Le RAD Visual C# inclus dans Visual Studio, a des **caractéristiques très proches de celles de ses parents Java et Delphi,** tout en apportant quelques améliorations. L'aspect fortement typé du langage C# autorise la prise en compte par le développeur débutant de bonnes attitudes de programmation.

C'est Apple qui en a été le promoteur

Le premier environnement de développement visuel professionnel fut basé sur **Object Pascal** et a été conçu par Apple pour le système d'exploitation **MacOs**, sous la dénomination de **MacApp** en 1986. Cet environnement objet visuel permettait de développer des applications MacIntosh avec souris, fenêtre, menus déroulants etc...

Microsoft l'a popularisé

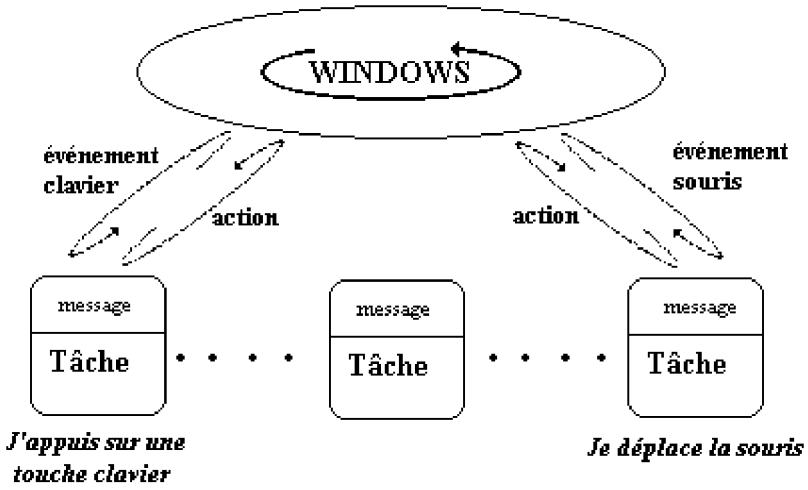
Le RAD Visual Basic de MicroSof conçu à partir de 1992, basé sur le langage Basic avait pour objectif le développement de petits logiciels sous Windows par des **programmeurs non expérimentés et occasionnels**. Actuellement il se décline en VB.Net un langage totalement orienté objet faisant partie intégrante de la plate-forme .Net Framework de Microsoft avec Visual C#.

Le concept de RAD a de beaux jours devant lui

Le métier de développeur devrait à terme, consister grâce à des outils tels que les RAD visuels, à prendre un "caddie" et à aller dans un supermarché de composants logiciels génériques adaptés à son problème. Il ne lui resterait plus qu'à assembler le flot des événements reliant entre eux ces logiciels en kit.

2. Programmation orientée événements

Sous les versions actuelles de Windows, système multi-tâches préemptif sur micro-ordinateur, les concepts quant à la programmation par événement restent sensiblement les mêmes que sous les anciennes versions.



Nous dirons que le système d'exploitation passe l'essentiel de son " temps " à attendre une action de l'utilisateur (événement). Cette action déclenche un message que le système traite et envoie éventuellement à une application donnée.

Une définition de la programmation orientée événements

Logique de conception selon laquelle un programme est construit avec des objets et leurs propriétés et d'après laquelle les interventions de l'utilisateur sur les objets du programme déclenchent l'exécution des routines associées.

Par la suite, nous allons voir dans ce chapitre que la programmation d'une application " windows-like " est essentiellement une programmation par événements associée à une programmation classique.

Nous pourrons construire un logiciel qui réagira sur les interventions de l'utilisateur si nous arrivons à intercepter dans notre application les messages que le système envoie. Or l'environnement RAD (C#, comme d'ailleurs avant lui Visual Basic de Microsoft), autorise la consultation de tels messages d'un façon simple et souple.

Deux approches pour construire un programme

- *L'**approche événementielle*** intervient principalement dans l'interface entre le logiciel et l'utilisateur, mais aussi dans la liaison dynamique du logiciel avec le système, et enfin dans la sécurité.
- *L'**approche visuelle*** nous aide et simplifie notre tâche dans la construction du dialogue homme-machine.
- La combinaison de ces deux approches produit un logiciel habillé et adapté au système d'exploitation.

Il est possible de relier certains objets entre eux par des relations événementielles. Nous les représenterons par un graphe (structure classique utilisée pour représenter des relations). Lorsque l'on utilise un système multi-fenêtré du genre windows, l'on dispose du clavier et de la souris pour agir sur le système. En utilisant un RAD visuel, il est possible de **construire un logiciel qui se comporte comme le système sur lequel il s'exécute**. L'intérêt est que l'utilisateur aura moins d'efforts à accomplir pour se servir du programme puisqu'il aura des fonctionnalités semblables au système. Le fait que l'utilisateur reste dans un **environnement familier** au niveau de la manipulation et du confort du dialogue, assure le logiciel d'un capital confiance de départ non négligeable.

3. Normalisation du graphe événementiel

Il n'existe que peu d'éléments accessibles aux débutants sur la programmation orientée objet par événements. Nous construisons une démarche méthodique pour le débutant, en partant de remarques simples que nous décrivons sous forme de schémas dérivés des diagrammes d'états d'UML. Ces schémas seront utiles pour nous aider à décrire et à implanter des relations événementielles en C# ou dans un autre RAD événementiel.

Voici deux principes qui pour l'instant seront suffisants à nos activités de programmation.

Dans une interface windows-like nous savons que:

- Certains événements déclenchent immédiatement des actions comme par exemple des appels de routines système.
- D'autres événements ne déclenchent pas d'actions apparentes mais activent ou désactivent certains autres événements système.

Nous allons utiliser ces deux principes pour conduire notre programmation par événements.

Nous commencerons par **le concept d'activation et de désactivation**, les autres événements fonctionneront selon les mêmes bases. Dans un enseignement sur la programmation événementielle, nous avons constaté que ce **concept était suffisant** pour que les étudiants comprennent les fondamentaux de l'approche événementielle.

Remarque :

Attention! Il ne s'agit que d'une manière particulière de conduire notre programmation, ce ne peut donc être ni la seule, ni la meilleure (le sens accordé au mot meilleur est relatif au domaine pédagogique). Cette démarche s'est révélée être fructueuse lors d'enseignements d'initiation à ce genre de programmation.

Hypothèses de construction

Nous supposons donc que lorsque l'utilisateur intervient sur le programme en cours d'exécution, ce dernier réagira en première analyse de deux manières possibles :

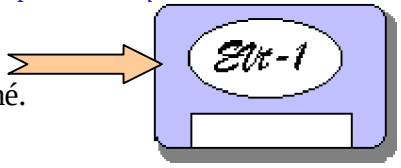
- soit il lancera l'appel d'une routine (exécution d'une action, calcul, lecture de fichier, message à un autre objet comme ouverture d'une fiche etc...),
- soit il modifiera l'état d'activation d'autres objets du programme et/ou de lui-même, soit il ne se passera rien, nous dirons alors qu'il s'agit d'une modification nulle.

Ces hypothèses sont largement suffisantes pour la plupart des logiciels que nous pouvons raisonnablement espérer construire en initiation. Les concepts plus techniques de messages dépassent assez vite l'étudiant qui risque de replonger dans de " la grande bidouille ".

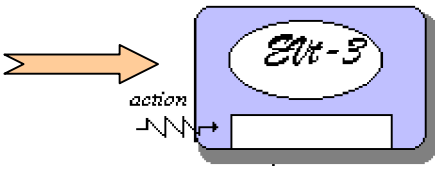
3.1 le graphe événementiel arcs et sommets

Nous proposons de construire un graphe dans lequel :

chaque **sommet** est un objet sensible à un événement donné.

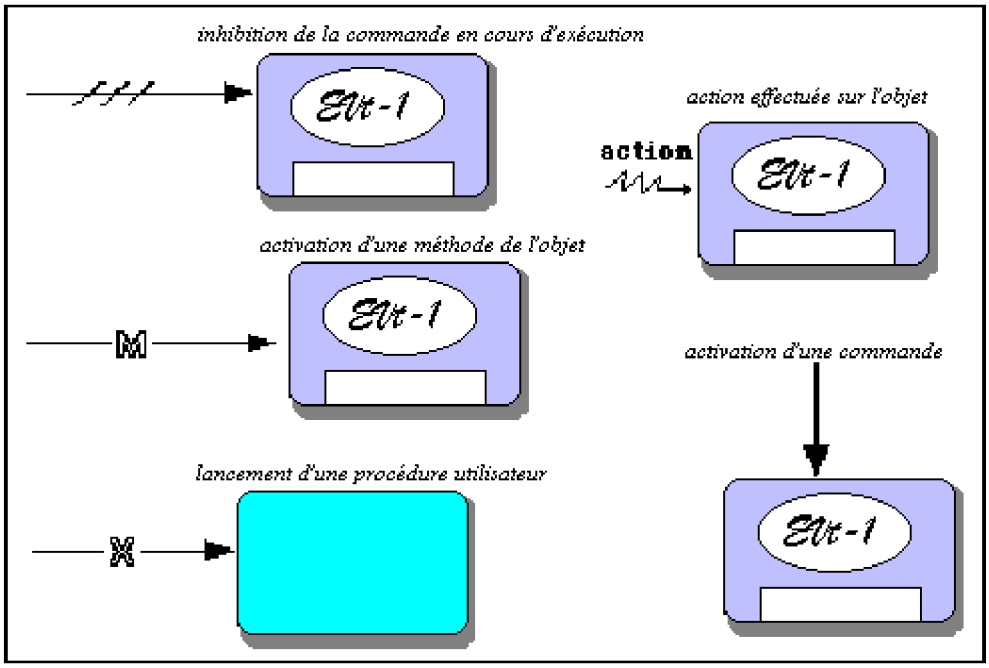


L'événement donné est déclenché par une action extérieure à l'objet.

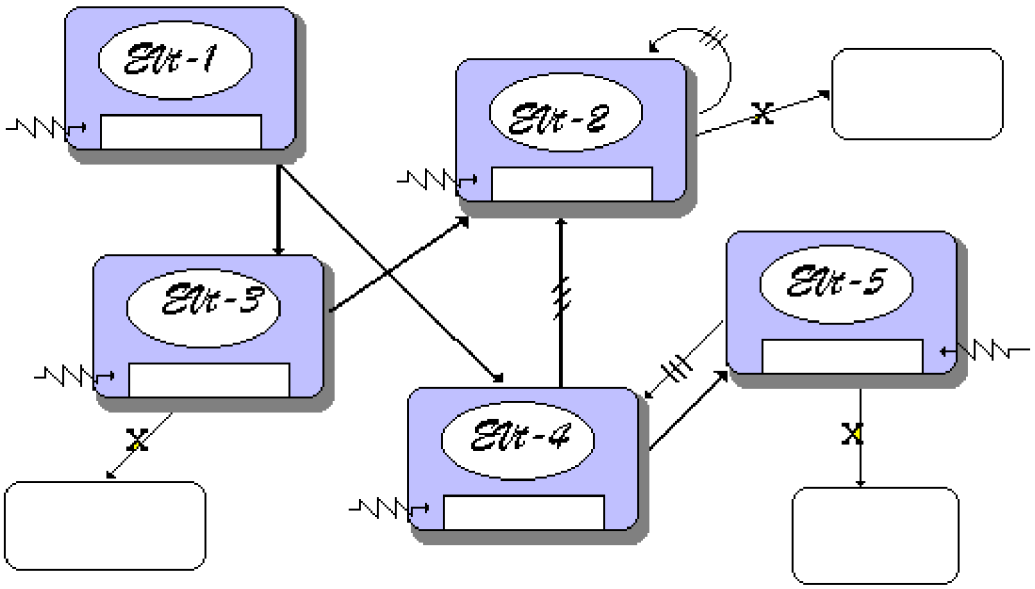


Les arcs du graphe représentent des actions lancées par un sommet.

Les actions sont de 4 types

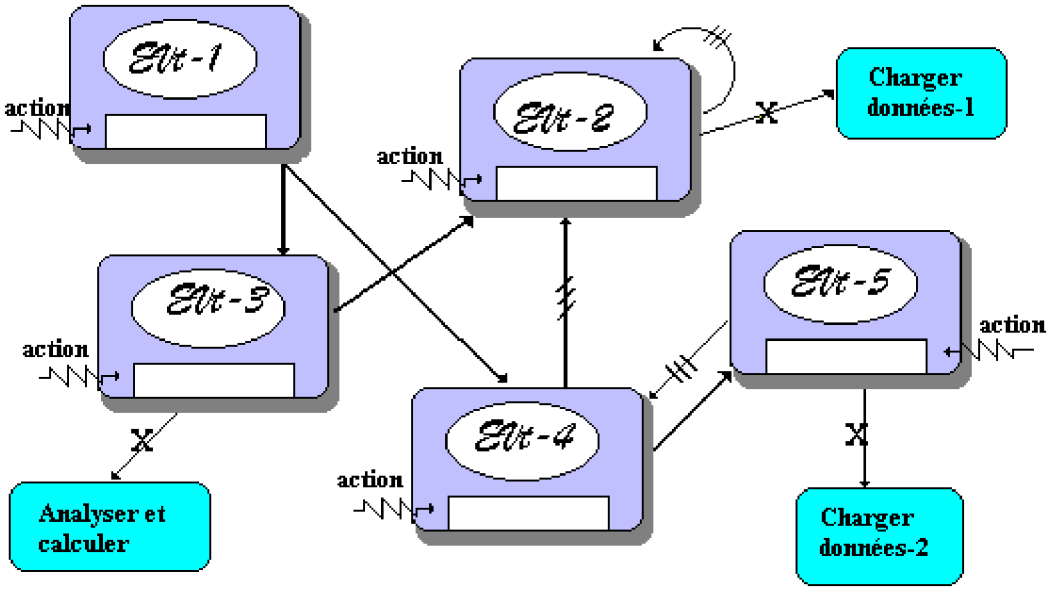


Soit le graphe événementiel suivant composé de 5 objets sensibles chacun à un événement particulier dénoté Evt-1,..., Evt-5; ce graphe comporte des réactions de chaque objet à l'événement auquel il est sensible :



Imaginons que ce graphe corresponde à une analyse de chargements de deux types différents de données à des fins de calcul sur leurs valeurs.

La figure suivante propose un tel graphe événementiel à partir du graphe vide précédent.



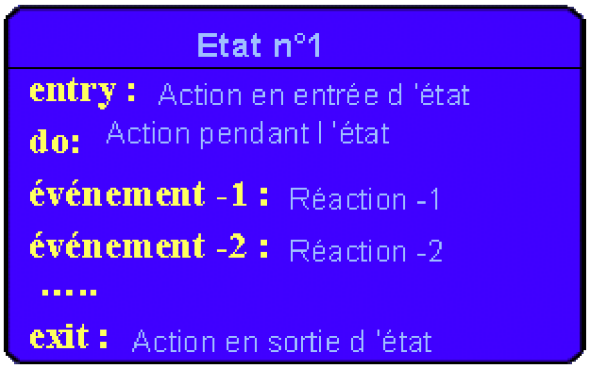
Cette notation de graphe événementiel est destinée à s'initier à la pratique de la description d'au maximum 4 types de réactions d'un objet sur la sollicitation d'un seul événement.

Remarques :

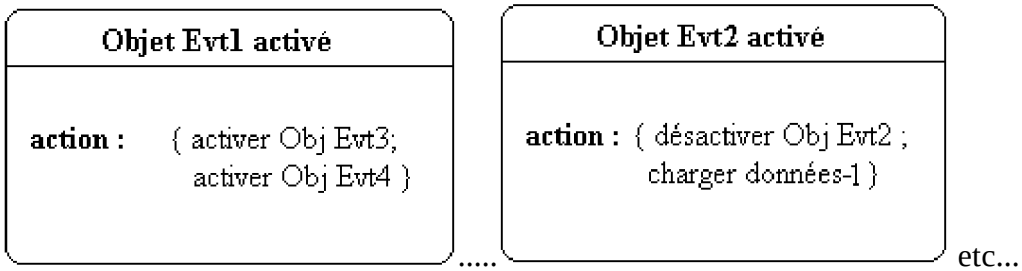
- L'arc nommé, représentant l'activation d'une méthode correspond très exactement à la notation UML de l'envoi d'un message.
- Lorsque nous voudrons représenter d'une manière plus complète d'autres réactions d'**un seul** objet **à plusieurs événements différents**, nous pourrons utiliser la notation UML réduite de diagramme d'état pour un objet (réduite parce qu'un objet **visuel** ne prendra pour nous, que 2 états: activé ou désactivé).

3.2 les diagrammes d'états UML réduits

Nous livrons ci-dessous la notation générale de diagramme d'état en UML, les cas particuliers et les détails complets sont décrits dans le document de spécification d'UML.

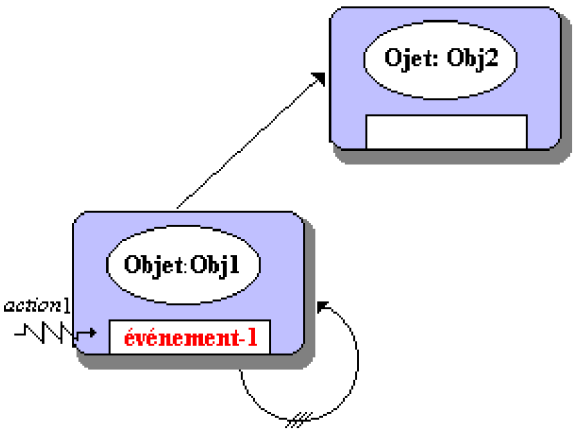


Voici les diagrammes d'états réduits extraits du graphe événementiel précédent, pour les objets **Evt-1** et **Evt-2** :

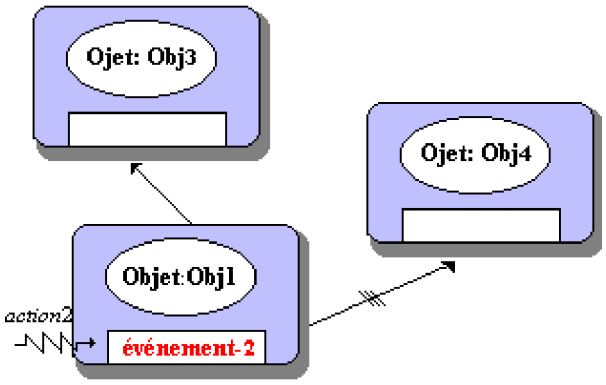


Nous remarquerons que cette écriture, pour l'instant, ne produit pas plus de sens que le graphe précédent qui comporte en sus la vision globale des interrelations entre les objets.
Ces diagrammes d'états réduits deviennent plus intéressants lorsque nous voulons exprimer le fait par exemple, qu'un seul objet **Obj1** réagit à 3 événements (événement-1, événement-2, événement-3). Dans ce cas décrivons les portions de graphe événementiel associés à chacun des événements :

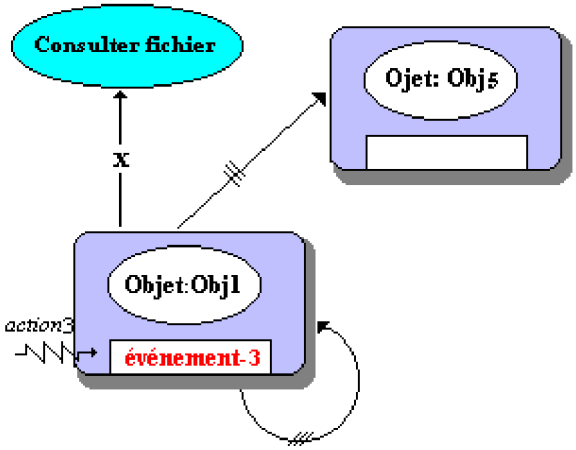
Réaction de obj1 à l'événement-1 :



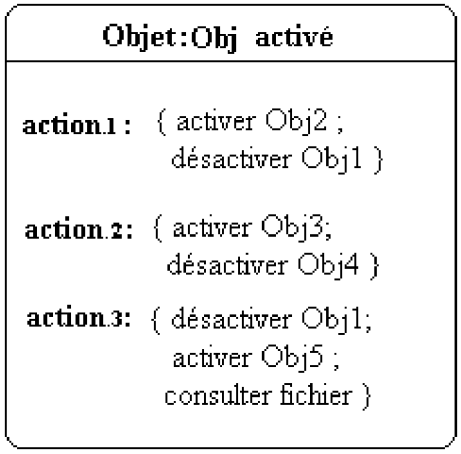
Réaction de obj1 à l'événement-2 :



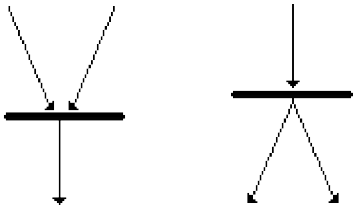
Réaction de obj1 à l'événement-3 :



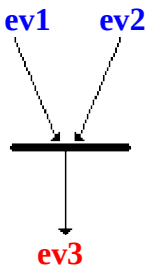
Synthétisons dans un diagramme d'état réduit les réactions à ces 3 événements :



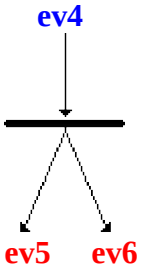
Lorsque nous jugerons nécessaire à la compréhension de relations événementielles dans un logiciel visuel, nous pourrons donc utiliser ce genre de diagramme pour renforcer la sémantique de conception des objets visuels. La notation UML sur les diagrammes d'états comprend les notions d'état de départ, de sortie, imbriqué, historisé, concurrents...
Lorsque cela sera nécessaire nous utiliserons la notation UML de synchronisation d'événements :



Dans le premier cas la notation représente la conjonction des deux événements **ev1** et **ev2** qui déclenche l'événement **ev3**.



Dans le second cas la notation représente l'événement **ev4** déclenchant conjointement les deux événements **ev5** et **ev6**.



4. Tableau des actions événementielles

L'exemple de graphe événementiel précédent correspond à une application qui serait sensible à 5 événements notés EVT-1 à EVT-5, et qui exécuterait 3 procédures utilisateur. Nous

notons dans un tableau (nommé " ***tableau des actions événementielles*** ")les résultats obtenus par analyse du graphe précédent, événement par événement..

EVT-1	EVT-3 activable EVT-4 activable
EVT-2	Appel de procédure utilisateur "chargement-1" désactivation de l' événement EVT-2
EVT-3	Appel de procédure utilisateur "Analyser" EVT-2 activable
EVT-4	EVT-2 désactivé EVT-5 activable
EVT-5	EVT-4 désactivé immédiatement Appel de procédure utilisateur "chargement-2"

Nous adjoignons à ce tableau une table des états des événements dès le lancement du logiciel (elle correspond à l'état initial des objets). Par exemple ici :

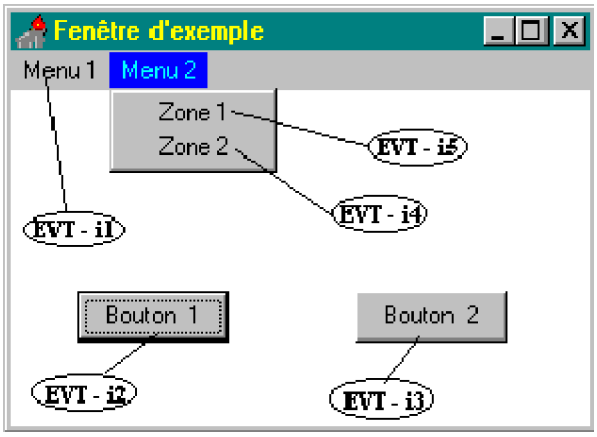
Evt1	activé
Evt2	désactivé
Evt3	activé
Evt4	activé
Evt5	désactivé

etc...

5. Interfaces liées à un graphe événementiel

construction d'interfaces liées au graphe précédent

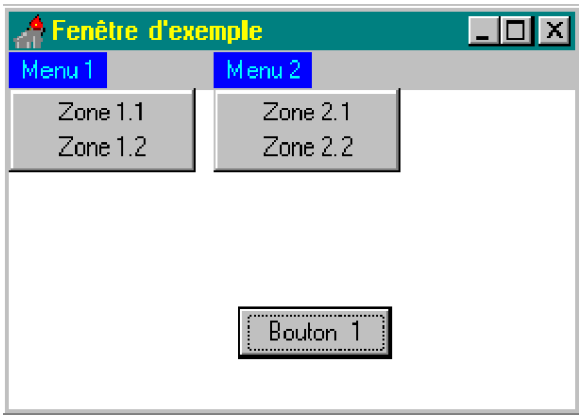
Interface n°1



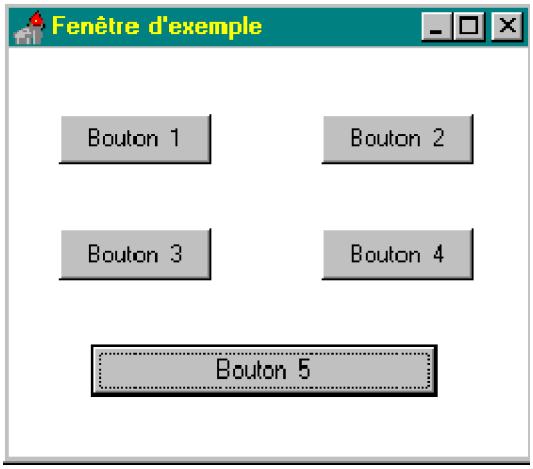
Dans l'exemple de droite, (i1,i2,i3,i4,i5)est une permutation sur (1,2,3,4,5) .

Ce qui nous donne déjà 5!=120 interfaces possibles avec ces objets et uniquement avec cette topologie.

La figure précédente montre une IHM à partir du graphe événementiel étudié plus haut. Ci-dessous deux autres structures d'interfaces possibles avec les mêmes objets combinés différemment et associés au même graphe événementiel :



Interface n°2



Interface n°3

Pour les choix n°2 et n°3, il y a aussi 120 interfaces possibles…

6. Avantages du modèle de développement RAD visuel

L'approche uniquement structurée (privilégiant les fonctions du logiciel) impose d'écrire du code long et compliqué en risquant de ne pas aboutir, car il faut tout tester afin d'assurer un bon fonctionnement de tous les éléments du logiciel.

L'approche événementielle préfère bâtir un logiciel fondé sur une construction graduelle en fonction des besoins de communication entre l'humain et la machine. Dans cette optique, le programmeur élabore les fonctions associées à une action de communication en privilégiant le dialogue. Ainsi les actions internes du logiciel sont subordonnées au flux du dialogue.

Avantages liés à la programmation par RAD visuel

- Il est possible de construire très rapidement un prototype.
- Les fonctionnalités de communication sont les guides principaux du développement (approche plus vivante et attrayante).
- L'étudiant est impliqué immédiatement dans le processus de conception - construction.

L'étudiant acquiert très vite comme naturelle l'attitude de réutilisation en se servant de " logiciels en kit " (soit au début des composants visuels ou non, puis par la suite ses propres composants).

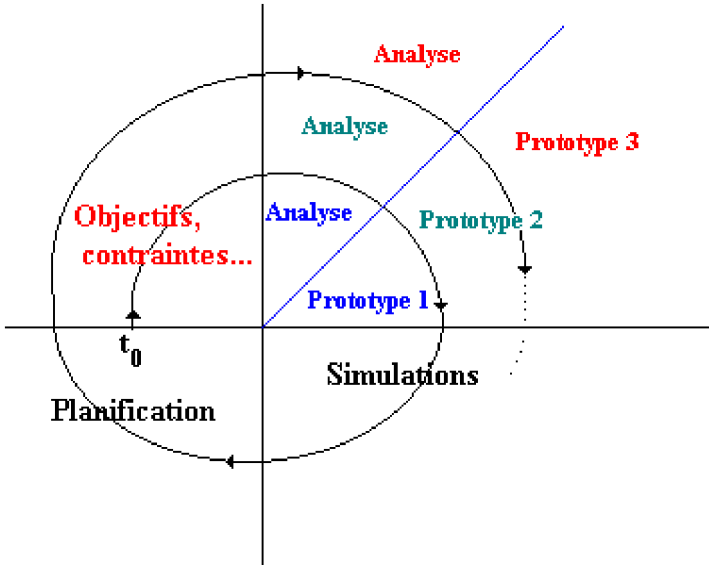
Il n'y a pas de conflit ni d'incohérence avec la démarche structurée : les algorithmes étant conçus comme des boîtes noires permettant d'implanter certaines actions, seront réutilisés immédiatement.
La méthodologie objet de COO et de POO reste un facteur d'intégration général des activités du programmeur.

Les actions de communications classiques sont assurées immédiatement par des objets standards (composants visuels ou non) réagissant à des événements extérieurs.
Le RAD fournira des classes d'objets standards non visuels (extensibles si l'étudiant augmente sa compétence) gérant les structures de données classiques (Liste, arbre, etc..).
L'extensibilité permet à l'enseignant de rajouter ses kits personnels d'objets et de les mettre à la disposition des étudiants comme des outils standards.

Modèles de développement avec un RAD visuel objet

Le développement avec ce genre de produit autorise une logique générale articulée sur la combinaison de deux modèles de développement :

le modèle de la spirale (B.Boehm) en version simplifiée

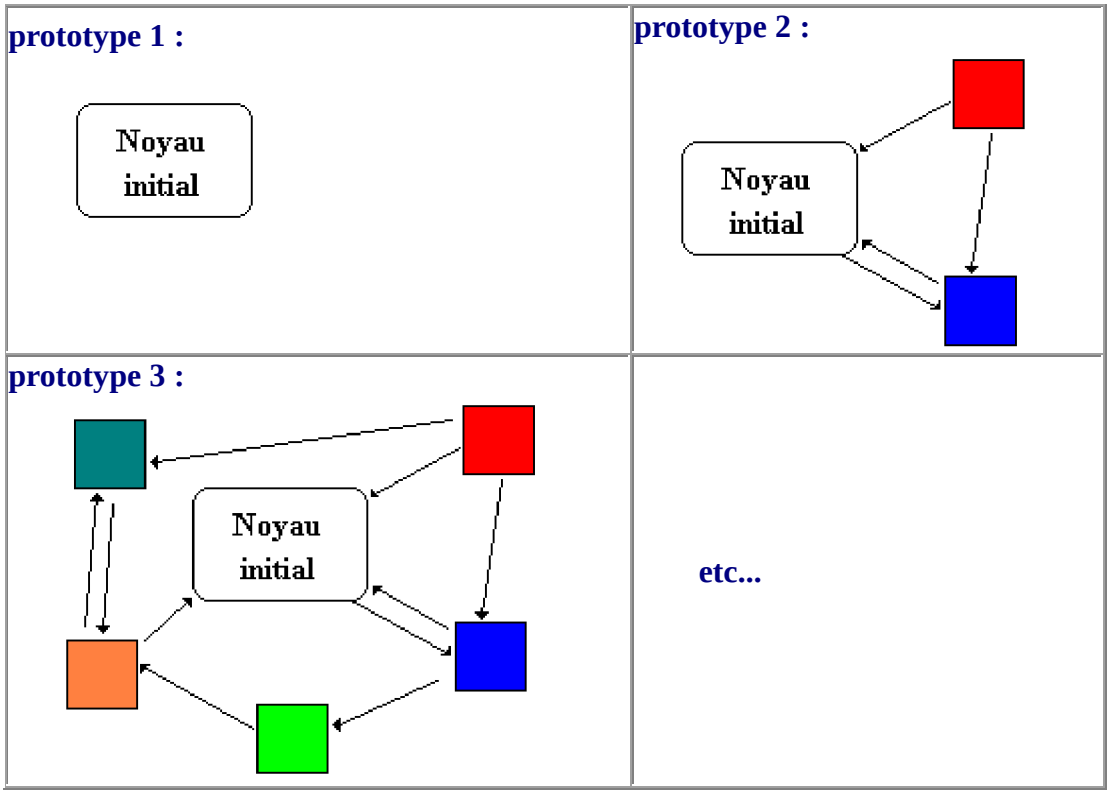


Dans le modèle de la spirale la programmation exploratoire est utilisée sous forme de prototypes simplifiés cycle après cycle. L'analyse s'améliore au cours de chaque cycle et fixe le type de développement pour ce tour de spirale.

le modèle incrémental

Il permet de réaliser chaque prototype avec un bloc central au départ, s'enrichissant à chaque phase de nouveaux composants et ainsi que de leurs interactions.

Associé à un cycle de prototypage dans la spirale, une seule famille de composants est développée pour un cycle fixé.



Ce modèle de développement à l'aide d'objets visuels ou non, fournira en fin de parcours un prototype opérationnel qui pourra s'intégrer dans un projet plus général.

Nous verrons sur des exemples comment ce type d'outil peut procurer aussi des avantages au niveau de la programmation défensive.

7. principes d'élaboration d'une IHM

Nous énumérons quelques principes utiles à l'élaboration d'une interface associée étroitement à la programmation événementielle

Notre point de vue reste celui du pédagogue et non pas du spécialiste en ergonomie ou en psychologie cognitive qui sont deux éléments essentiels dans la conception d'une interface homme-machine (**IHM**). Nous nous efforcerons d'utiliser les principes généraux des **IHM** en les mettant à la portée d'un débutant avec un double objectif :

- Faire écrire des programmes interactifs. Ce qui signifie que le programme doit communiquer avec l'utilisateur qui reste l'acteur privilégié de la communication. Les programmes sont Windows-like et nous nous servons du RAD visuel C# pour les développer. Une partie de la spécification des programmes s'effectue avec des objets graphiques représentant des classes (programmation objet visuelle).

- Le programmeur peut découpler pendant la conception la programmation de son interface de la programmation des tâches internes de son logiciel (pour nous généralement ce sont des algorithmes ou des scénarios objets).

Nous nous efforçons aussi de ne proposer que des outils pratiques qui sont à notre portée et utilisables rapidement avec un système de développement RAD visuel comme C#.

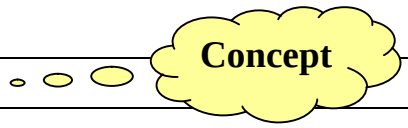
Interface homme-machine, les concepts

Les spécialistes en ergonomie conceptualisent une IHM en six concepts :

- les objets d'entrée-sortie,
- les temps d'attente (temps de réponse aux sollicitations),
- le pilotage de l'utilisateur dans l'interface,
- les types d'interaction (langage,etc..) ,
- l'enchaînement des opérations,
- la résistance aux erreurs (ou robustesse qui est la qualité qu'un logiciel à fonctionner même dans des conditions anormales).

Un principe général provenant des psycho-linguistes guide notre programmeur dans la complexité informationnelle : la mémoire rapide d'un humain ne peut être sollicitée que par un nombre limité de concepts différents en même temps (nombre compris entre sept et neuf). Développons un peu plus chacun des six concepts composants une interface.

Les objets d'entrée-sortie



Une IHM présente à l'utilisateur un éventail d'informations qui sont de deux ordres : des commandes entraînant des actions internes sur l'IHM et des données présentées totalement ou partiellement selon l'état de l'IHM.

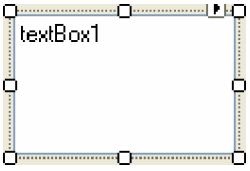
Les commandes participent à la " saisie de l'intention d'action" de l'utilisateur, elles sont matérialisées dans le dialogue par des ***objets d'entrée*** de l'information (boîtes de saisie, boutons, menus etc...).

Voici avec un RAD visuel comme C#, des objets visuels associés aux objets d'entrée de l'information, ils sont très proches visuellement des objets que l'on trouve dans d'autres RAD visuels, car en fait ce sont des surcouches logiciels de contrôles de base du système d'exploitation (qui est lui-même fenêtré et se présente sous forme d'une IHM dénommée bureau électronique).

un bouton

une boîte de saisie mono-ligne

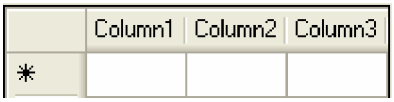
Un menu



une boîte de saisie multi-ligne

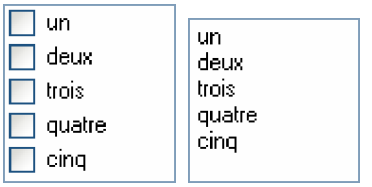
Les données sont présentées à un instant précis du dialogue à travers des ***objets de sortie*** de l'information (boîte d'édition monoligne, multiligne, tableaux, graphiques, images, sons etc...).

Ci-dessous quelques objets visuels associés à des objets de sortie de l'information :

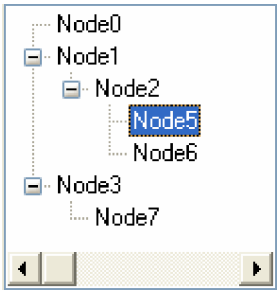


un dataGridView (table)

un pictureBoxe(image jpg, png, bm ,ico,...)



un checkedListBox et un listBox (listes)



un treeView (arbre)

Les temps d'attente

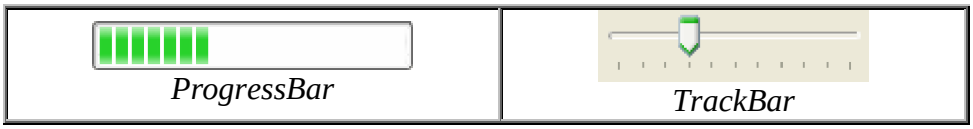
Concept

Sur cette question, une approche psychologique est la seule réponse possible, car l'impression d'attente ne dépend que de celui qui attend selon son degré de patience. Toutefois, puisque nous avons parlé de la mémoire à court terme (mémoire rapide), nous pouvons nous baser sur les temps de persistance généralement admis (environ 5 secondes).

Nous considérerons qu'en première approximation, si le délai d'attente est :

- inférieur à environ une seconde la réponse est quasi-instantanée,
- compris entre une seconde et cinq secondes il y a attente, toutefois la mémoire rapide de l'utilisateur contient encore la finalité de l'opération en cours.
- lorsque l'on dépasse la capacité de mémorisation rapide de l'utilisateur alors il faut soutenir l'attention de l'utilisateur en lui envoyant des informations sur le déroulement de l'opération en cours (on peut utiliser pour cela par exemple des barres de défilement, des jauges, des boîtes de dialogue, etc...)

Exemples de quelques classes d'objets visuels de gestion du délai d'attente :



statusStrip (avec deux éléments ici)

Le pilotage de l'utilisateur

Concept

Nous ne cherchons pas à explorer les différentes méthodes utilisables pour piloter un utilisateur dans sa navigation dans une interface. Nous adoptons plutôt la position du concepteur de logiciel qui admet que le futur utilisateur ne se servira de son logiciel que d'une façon épisodique. Il n'est donc pas question de demander à l'utilisateur de connaître en permanence toutes les fonctionnalités du logiciel.

En outre, il ne faut pas non plus submerger l'utilisateur de conseils de guides et d'aides à profusion, car ils risqueraient de le détourner de la finalité du logiciel. Nous préférons adopter une ligne moyenne qui consiste à fournir de petites aides rapides contextuelles (au moment où l'utilisateur en a besoin) et une aide en ligne générale qu'il pourra consulter s'il le souhaite. Ce qui revient à dire que l'on accepte deux niveaux de navigation dans un logiciel :

- le niveau de surface permettant de réagir aux principales situations,
- le niveau approfondi qui permet l'utilisation plus complète du logiciel.

Il faut admettre que le niveau de surface est celui qui restera le plus employé (l'exemple d'un logiciel de traitement de texte courant du commerce montre qu'au maximum 30% des fonctionnalités du produit sont utilisées par plus de 90% des utilisateurs).

Pour permettre un pilotage plus efficace on peut établir à l'avance un graphe d'actions possibles du futur utilisateur (nous nous servirons du graphe événementiel) et ensuite diriger l'utilisateur dans ce graphe en matérialisant (masquage ou affichage) les actions qui sont réalisables. En application des notions acquises dans les chapitres précédents, nous utiliserons un pilotage dirigé par la syntaxe comme exemple.

Les types d'interaction

Concept

Le tout premier genre d'interaction entre l'utilisateur et un logiciel est apparu sur les premiers systèmes d'exploitation sous la forme d'un langage de commande. L'utilisateur dispose d'une famille de commandes qu'il est censé connaître, le logiciel étant doté d'une interface interne (l'interpréteur de cette famille de commandes). Dès que l'utilisateur tape textuellement une commande (exemple MS-DOS " **dir c: /w** "), le système l'interprète (dans l'exemple : lister en prenant toutes les colonnes d'écran, les bibliothèques et les fichiers du disque C).

Nous adoptons comme mode d'interaction entre un utilisateur et un logiciel, une extension plus moderne de ce genre de dialogue, en y ajoutant, en privilégiant, la notion d'objets visuels permettant d'effectuer des commandes par actions et non plus seulement par syntaxe textuelle pure.
Nous construisons donc une interface tout d'abord ***essentiellement*** à partir des interactions événementielles, puis lorsque cela est utile ou nécessaire, nous pouvons ajouter un interpréteur de langage (nous pouvons par exemple utiliser des automates d'états finis pour la reconnaissance).

L'enchaînement des opérations

Concept

Nous savons que nous travaillons sur des machines de Von Neumann, donc séquentielles, les opérations internes s'effectuant selon un ordre unique sur lequel l'utilisateur n'a aucune prise. L'utilisateur est censé pouvoir agir d'une manière " aléatoire ". Afin de simuler une certaine liberté d'action de l'utilisateur nous lui ferons parcourir un graphe événementiel prévu par le programmeur. Il y a donc contradiction entre la rigidité séquentielle imposée par la machine et la liberté d'action que l'on souhaite accorder à l'utilisateur. Ce problème est déjà présent dans un système d'exploitation et il relève de la notion de gestion des interruptions.

Nous pouvons trouver un compromis raisonnable dans le fait de découper les tâches internes en tâches séquentielles minimales ininterruptibles et en tâches interruptibles.

Les interruptions consisteront en actions potentielles de l'utilisateur sur la tâche en cours afin de :

- interrompre le travail en cours,
- quitter définitivement le logiciel,
- interroger un objet de sortie,
- lancer une commande exploratoire ...

Il faut donc qu'existe dans le système de développement du logiciel, un mécanisme qui permette de " *demander la main au système* " sans arrêter ni bloquer le reste de l'interface, ceci pendant le déroulement d'une action répétitive et longue. Lorsque l'interface a la main, l'utilisateur peut alors interrompre, quitter, interroger...

Ce mécanisme est disponible dans les RAD visuels pédagogiques (Delphi,Visual Basic, Visual C#), nous verrons comment l'implanter. Terminons ce tour d'horizon, par le dernier concept de base d'une interface : sa capacité à absorber certains dysfonctionnements.

La résistance aux erreurs

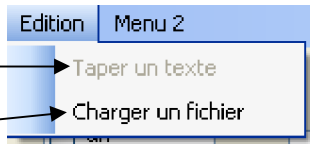
Concept

Il faut en effet employer une méthode de programmation défensive afin de protéger le logiciel contre des erreurs comme par exemple des erreurs de manipulation de la part de l'utilisateur. Nous utilisons plusieurs outils qui concourent à la robustesse de notre logiciel. La protection est donc située à plusieurs niveaux.

1°) Une protection est apportée par le graphe événementiel qui n'autorise que certaines actions (activation-désactivation), matérialisé par un objet tel qu'un menu :

l'item " taper un fichier " n'est pas activable

l'item " Charger un fichier " est activable



2°) Une protection est apportée par le filtrage des données (on peut utiliser par exemple des logiciels d'automates de reconnaissance de la syntaxe des données).

3°) Un autre niveau de protection est apporté par les composants visuels utilisés qui sont sécurisés dans le RAD à l'origine. Par exemple la méthode LoadFile d'un richTextBox de C# qui permet le chargement d'un fichier dans un composant réagit d'une manière sécuritaire (c'est à dire rien ne se produit) lorsqu'on lui fournit un chemin erroné ou que le fichier n'existe pas.

4°) Un niveau de robustesse est apporté en C# par une utilisation des exceptions (semblable à C++ ou à Delphi) autorisant le détournement du code afin de traiter une situation interne anormale (dépassement de capacité d'un calcul, transtypage de données non conforme etc...). Le programmeur peut donc prévoir les incidents possibles et construire des gestionnaires d'exceptions.