INDEX

INTRODUCTION

Ce travail s'insère dans le cadre du projet collectif de *"Gestion Intégrée de Ressources Naturelles des Territoires et des Politiques Publiques"*, conduit par le LIRMM, l'INRA et le CIRAD.

Le travail suivant se divise en 2 parties : une partie très technique de couplage des trois logiciels utilisés, ce qui représente la partie la plus importante. La seconde correspond à la mise en place d'un modèle applicatif afin de tester ce couplage.

1. Le travail technique de couplage rassemble trois logiciels aux fonctionnalités complémentaires :
    - Une plate-forme de simulation (Systèmes Multi-Agents).
    - Un SGBD.
    - Un logiciel d'analyse spatiale.
2. Le travail de modélisation en application à ce couplage se cadre sur les travaux initiaux effectués par l'INRA en Lozère et sur le travail de C. Ivanès. L'étude porte sur la commune de St Georges de Lévejac , sur l'impact de l'embroussaillement et du parcours par les troupeaux sur le paysage.

La forme de ce rapport essaie d'expliciter au mieux l'aspect technique de l'outil, afin de pouvoir le réutiliser le plus facilement possible.

Les sources de programmation dans le simulateur et dans le SIG sont présente afin de mieux appréhender le modèle et les techniques utilisées.

Partie I: Techniques

1. Contexte

Ce travail est la continuité de mon travail effectué au CIRAD durant l'ATP (été 1998), qui concernait le couplage possible entre les Systèmes Multi-Agents (SMA) et les Systèmes d'Information Géographique (SIG). Il vise donc à y ajouter un troisième pôle que sont les Systèmes de Gestion de Bases de Données (SGBD). L'apport de bases de données a un double intérêt : le premier est d'augmenter la quantité d'information présente dans le SIG (ce travail a été réalisé par C. Ivanès lors du Master SILAT). Le second intérêt est de posséder des informations concrètes sur une situation donnée afin d'initialiser et d'enrichir le modèle SMA, et de pouvoir tirer des résultats significatifs de la simulation.

Les résultats de ce type de simulation sont beaucoup plus parlant, lorsqu'il s'agit de les restituer à leurs destinataires: Chambre d'Agriculture, SAFER, ou exploitants agricoles. Le fait de montrer des cartes réelles, avec des représentations du cadastre et d'utiliser dans la simulation des acteurs réels, renforce considérablement l'impact des résultats énoncés. Il faut cependant toujours garder à l'esprit que ces simulations ne peuvent pas être correctement exploitées tant qu'elles ne sont pas validées.

2. Présentation des outils utilisés

2.1 VisualWorks

VisualWorks est un langage de programmation en SmallTalk contenant l'interface de simulation Multi-Agents développée par le CIRAD Green. De nombreux suppléments sont disponibles afin d'augmenter ses capacités de couplage. On peut citer DLL&CConnect, offrant la connexion avec les DLL Windows et des morceaux de code écrits en C. Ce module est indispensable pour utiliser les liens DDE (cf. ci-dessous). La connexion à une base de données est incluse par défaut, mais supporte uniquement les bases Oracle, Sybase et Informix. La connexion ODBC est réalisable avec un supplément appelé ARAGON (cf. ci-dessous).

2.2 MS ACCESS

ACCESS est un SGBD relationnel, facile d'utilisation et dont les connexions avec d'autres applications utilisent ODBC. La base de données utilisée contient les information sur deux sections de la commune étudiée (Parcelle, Exploitation, SAU, occupation du sol,…)

2.3 ArcView

Logiciel SIG contenant le parcellaire cadastral des deux sections communales (polygones), relié à l'ensemble des données de la base ACCESS par l'interface ODBC. ArcView offre aussi la possibilité de liens DDE. Son langage de programmation Avenue permet de créer des scripts puissants pour automatiser des taches, mais permet aussi de commander ArcView depuis une autre application (grâce aux liens DDE).

2.4 l'interface ODBC

Interface développée sous Windows, elle rend accessible n'importe quelle base de données à une application compatible. Elle standardise les entrées/sorties de la base. Chaque base possède son propre pilote ODBC. Une même application peut donc attaquer plusieurs SGBD différents en utilisant la même méthode d'accès. De plus, les pilotes ODBC n'ont pas besoin que le SGBD soit présent sur le système, car il attaque directement le fichier de la base.

2.5 ARAGON

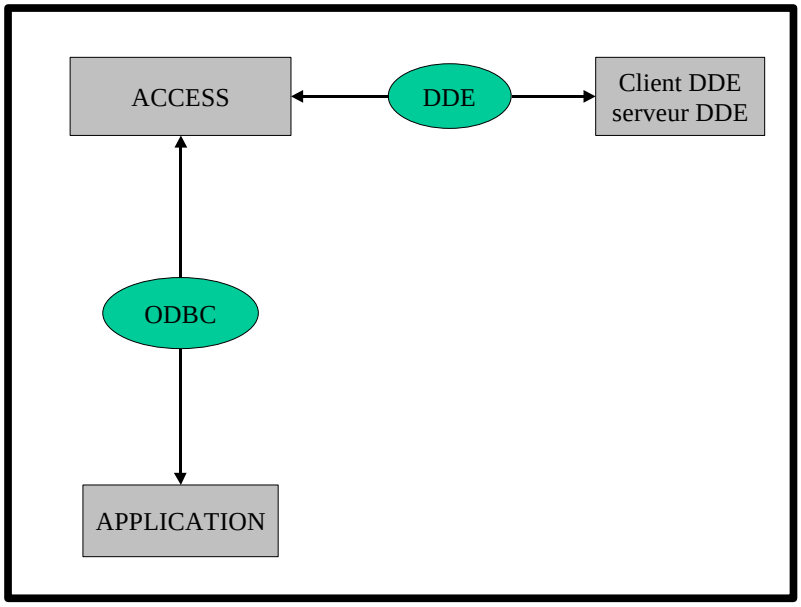
Supplément de VisualWorks développé par l'Institut xxx Allemand, il permet l'accès aux bases de données via ODBC.

2.6 DDE : Dynamic Data Exchange

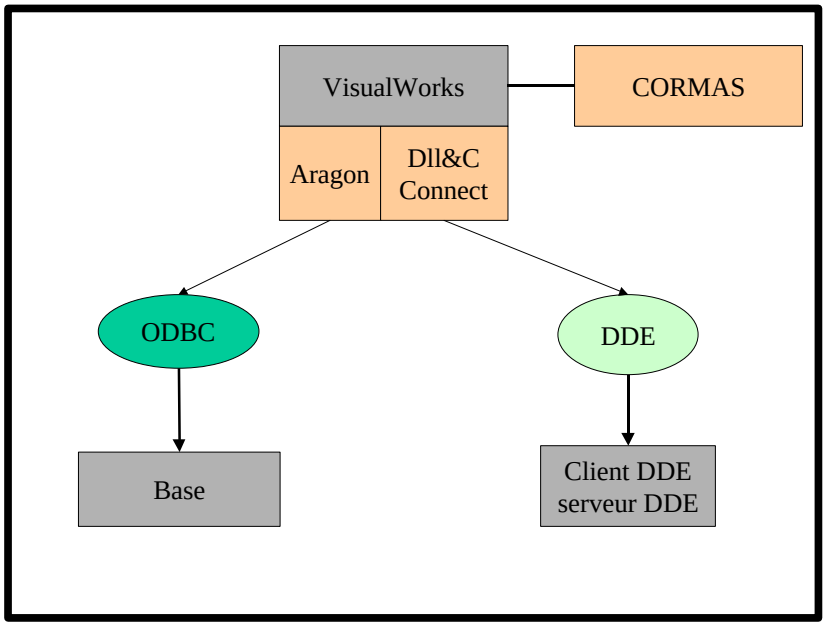
Protocole de communication entre logiciels exclusivement sous Windows, de type client/serveur, il permet d'échanger des données ou de piloter une autre application. Les deux applications doivent être présentes sur le système. Ce protocole appartient à la famille des protocole Windows du type OLE, COM, Automation, il est le précurseur de ces protocoles et devient de plus en plus obsolète.

3 Présentation des couplages possibles.

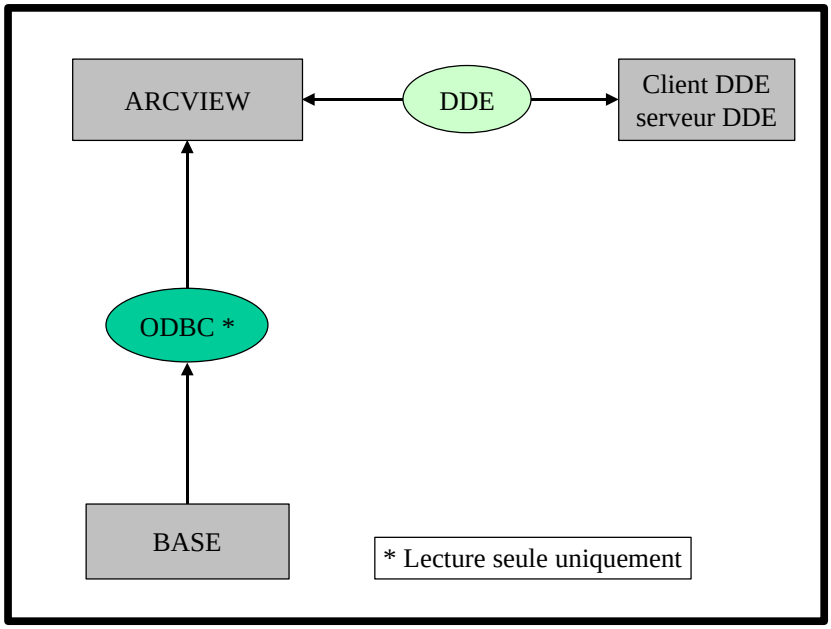
3.1 ACCESS



3.2 VisualWorks

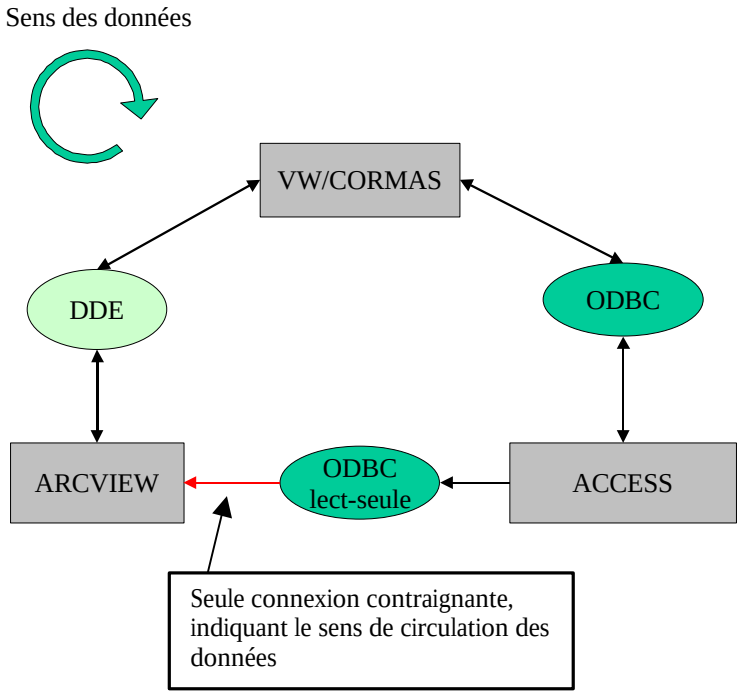


3.3 ArcView



4. Couplage choisi

Les couplages possibles par logiciel nous amènent à choisir un couplage triangulaire, de manière à ce que les données puissent circuler d'une application à l'autre.



Par la suite, nous verrons comment ce couplage est implémenté et quel modèle a été choisi pour le tester.

5. Le Modèle : Premières réflexions

5.1 Les besoins

Voici la liste des besoins pour réaliser une simulation intégrant tous les outils présentés:

- Pouvoir accéder aux données stockées dans la base pour initialiser le système du SMA.
- Pouvoir stocker dans cette base les résultats intermédiaires et finaux de la simulation, pour les utiliser dans le SIG.
- Pouvoir modéliser le processus de décisions des acteurs du modèle dans le SMA et reporter leurs actions sur l'environnement contenu dans le SMA ou le SIG.
- Pouvoir modéliser la dynamique spatiale de l'environnement (SMA et SIG).
- Réaliser les calculs spatiaux dans le SIG et les répercuter dans le simulateur.

5.2 Conséquences

Il faut donc préciser ce qui est connu de chacun des logiciels et assurer la correspondance et l'équivalence des données réparties dans le système. Il faut donc réfléchir à une structure d'accueil des données au sein des trois logiciels.

Il faut aussi harmoniser le temps entre les trois logiciels pour assurer l'équivalence des données. Cette harmonisation peut se faire en sérialisant l'ensemble des opérations de manière à ce que les données circulent toujours dans le même sens, sans conflits.

Les transformations de l'espace s'effectueront dans le SIG, mais il s'agira de définir la forme initiale des données spatiales (Vecteur/Raster), de leur évolution, et des différentes représentations que l'on pourra faire (légendes). On peut déjà penser que l'environnement humain se situera dans le simulateur tandis que l'environnement de l'espace naturel se trouvera dans le SIG. (Il s'agit alors de ne pas utiliser la grille raster contenue dans CORMAS, mais de diffuser les informations importantes pour les décisions des agents, depuis le SIG vers le simulateur.)

6. Aspect technique des connexions

6.1 Connexion VW-ArcView : DDE

DDE est un protocole de communication entre deux applications sous Windows. Ces applications doivent pouvoir utiliser une DLL, ddeml.dll, afin d'engager une discussion. ArcView gère cette connexion par défaut, par contre, VW nécessite un module supplémentaire DLL&Cconnect.

Le langage de communication utilise trois mots-clé :

- « *Execute* » : Demande l'exécution d'un programme ou d'un script à l'application cliente. C'est une forme de pilotage à distance.
- « *Request* » : Formule une requête à l'application cliente et attend une réponse sous la forme d'une chaîne de caractères.
- « *Poke* » : Modifie une donnée de l'application serveur.

Les commandes *Execute* et *Request* se font dans le langage de programmation de l'application distante. Dans le cas d'Excel, par exemple, on utilise le langage macro Excel 4.0. Dans notre cas, l'application cliente est ArcView et les communications s'effectue avec son langage de programmation Avenue.

Les scripts Avenue sont lancés depuis VW en instanciant la classe ArcViewClient. Cette classe contient les méthodes de connexion (*Execute* : et *Request* :). Ces scripts peuvent être directement écrits dans VW dans une chaîne de caractères, ou bien être écrits dans le projet ArcView et ensuite être appelés depuis VW par la commande 'av.run()' d'Avenue.

```
avclt := ArcViewClient new.
chaineAcces := avclt request: 'av.run("RequeteAcces2",nil)'.
avclt terminate.
```

ChaineAcces reçoit le résultat du script appelé RequeteAcces2. L'objet nil est le paramètre de la requête, ici, la requête n'a pas besoin de paramètres d'entrée.

La classe ArcViewClient est définie par héritage de la classe DDEClient en modifiant les deux paramètres d'identification de l'application concernée : nomDDE et son 'topic'. Pour Arcview, son nom d'identification est '*ArcView*' et son topic est '*system*'.

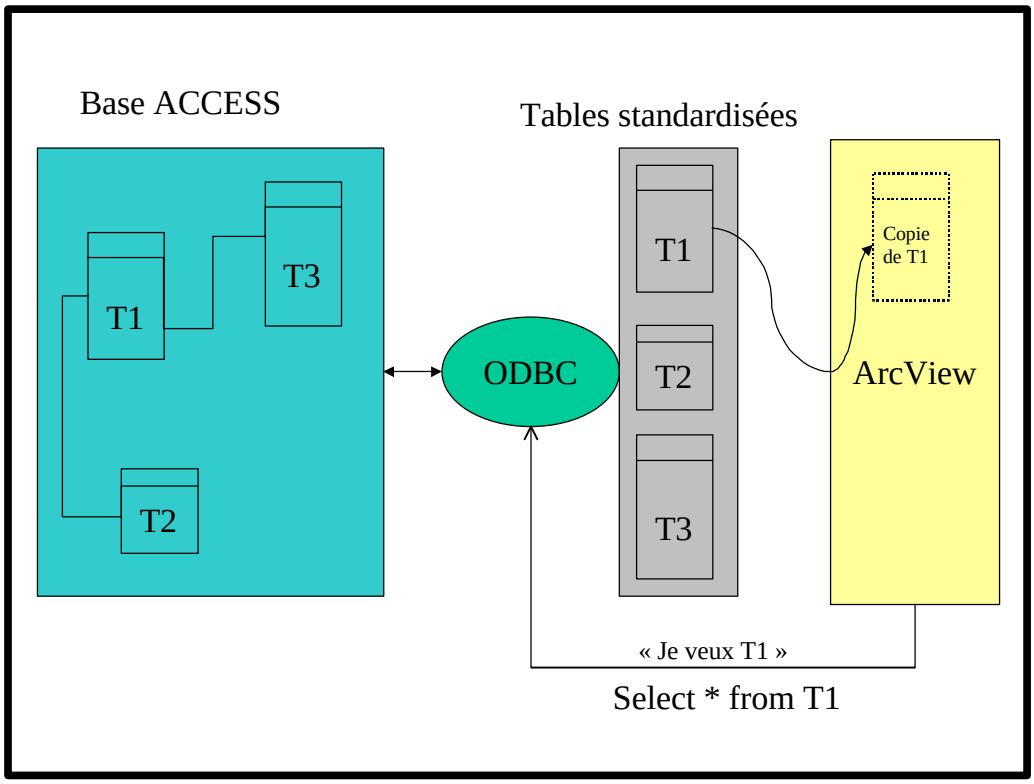
Dans notre cadre de travail, il est nécessaire de modifier les méthodes de connexion pour ajuster le paramètre de temps limite de connexion '*RequestTimeOut*' afin que les scripts éxécutés dans ArcView aient suffisamment de temps (par défaut, ce temps est de 1000ms et n'est pas suffisant).

Lorsqu'une réponse est attendue après un « *Request* », une chaîne de caractères est renvoyée. Un traitement de cette chaîne devra être effectué pour incorporer les données dans le simulateur.

6.2 Connexion ACCESS-ArcView : ODBC

Cette partie du travail a été réalisée par C. Ivanès lors de son stage pour le Master SILAT.

Elle utilise la connexion SQL d'ArcView, permettant d'obtenir l'accès à des tables d'une base reliée par ODBC. Cette connexion est en **lecture seule** car ArcView crée une copie temporaire de la table. Cette copie est alors jointe avec les données contenant les entités spatiales décrivant le parcellaire cadastral des deux sections communales et peuvent ainsi être étudiées dans ArcView.

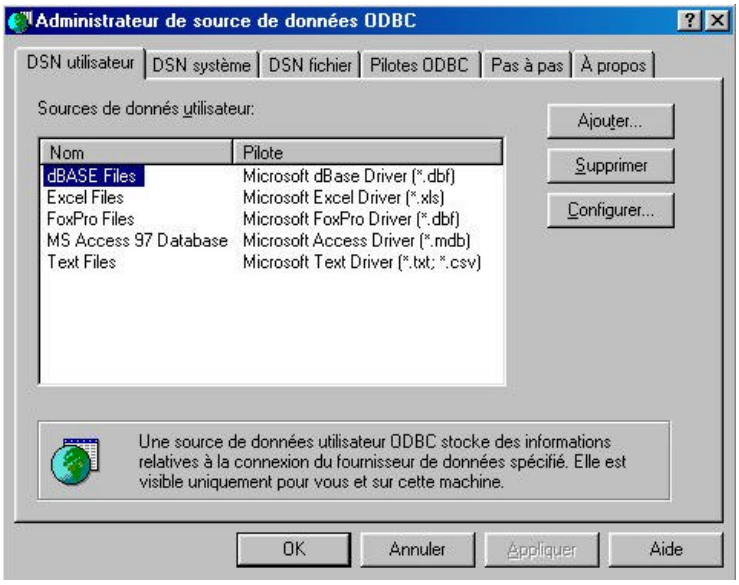


Cette copie de table est toutefois dynamique ; si une modification est faite dans la base, elle est répercutée dans la copie. Ce rafraîchissement peut être forcé dans un script, pour être sûr que la mise à jour a été effectuée en temps voulu.

6.3 Connexion VW-ACCESS : ODBC

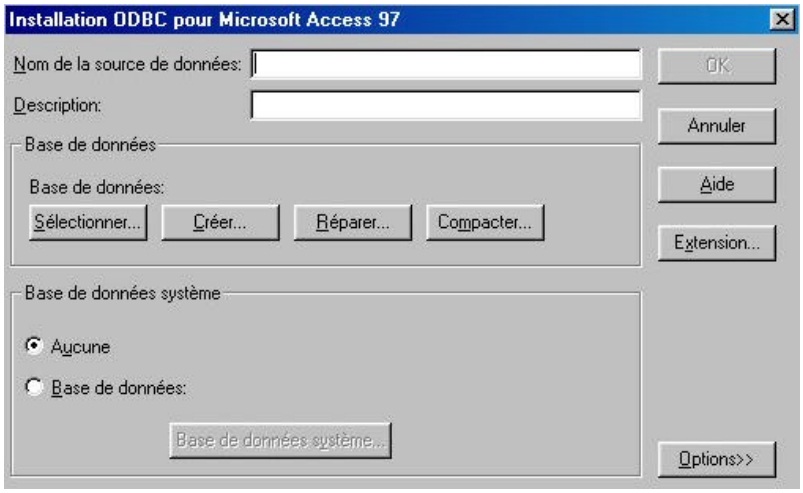
Comme il a été montré dans la présentation des logiciels, VW ne possède par défaut la connexion à une base via ODBC. Le module ARAGON pallie à ce problème.

Pour accéder à une base de donnée via ODBC, il faut en premier lieu créer une nouvelle connexion dans l'interface d'ODBC (DNS utilisateur) qui se trouve dans le panneau de configuration de Windows.



Pour déclarer notre base de données, Cliquer sur « *Ajouter* ».

Il faut choisir le pilote correspondant au type de base de données que l'on doit utiliser (ici il s'agit du driver MS ACCESS, puis cliquer sur « *Terminer* ».



Une nouvelle fenêtre apparaît, qui va nous permettre de définir l'ensemble des paramètres de la connexion :

- Nom de la source de données : dans notre cas nous choisirons « Georges ».
- Description : un petit laïus décrivant la connexion.
- Cliquer sur « Sélectionner » et rechercher le fichier correspondant.

- En cliquant sur « Extension », on peut accéder à des options supplémentaires ; en particulier, il est possible de définir un nom d'utilisateur et un mot de passe. Dans ce cadre, nous ne les utiliserons pas.
- Cliquer sur OK et quitter le Panneau de Configuration, l'interface de connexion est prête.

Les connexions dans VW sont gérées par la Classe DBConnect, qui hérite de la Classe ODBCConnection d'ARAGON. Cette classe a été ajoutée manuellement dans la catégorie de classe du modèle StGeorges (créé par CORMAS). Elle gère les processus de :

- Connexion et déconnexion
- Récupération des données
- Mises à jour des tables
- Ou tout autre requête SQL

L'établissement d'une connexion est défini au sein de la méthode *connect*. Ses paramètres d'entrée sont *user* (nom d'utilisateur, ici ''), *password* (ici ''), et *l'Environnement*, qui est un paramètre général qui cible dans notre cas le nom de la source de données ODBC ('Georges').

Une fois la connexion établie, les processus de dialogue sont appelés des **sessions**. Une session se compose ainsi :

```
uneSession := uneConnection getSession.
uneSession prepare: 'Requete SQL'.
uneSession execute.
uneSession answer.
uneSession disconnect.
```

Une session peut avoir besoin de paramètres de sorties (bindOutput), en particulier lorsque l'on veut récupérer les enregistrements de la table, il faut connaître la classe qui peut recevoir les données. Un enregistrement correspond à une instance de la classe correspondante.

Important : La classe instanciée par les enregistrements d'une table doit obligatoirement avoir **le même nombre d'attributs** qu'il y a dans la table associée.

Exemple :

```
uneConnexion := DbConnect new ; connect.
uneConnexion loadData : 'Parcelle' in : lesParcelles.
uneConnexion disconnect.
```

La méthode *loadData: in:* permet de récupérer tous les enregistrements de la table Parcelle dans une liste (*lesParcelles*). Chaque enregistrement est une instance de la Classe Parcelle (définie dans CORMAS). Voici le code de cette méthode :

```
      loadData: aTable from: aClassName in: aList
              "Load the customer data from the database."

              | conn sess ans aClass|
              (conn := self connection) isNil
                      ifTrue: [^Dialog warn: 'You are not connected to a database
system.'].

              self executeWithProtection:
                      [sess := conn prepare: 'select * from ', aTable.
                      aClass := (Smalltalk at: (aClassName asSymbol)) new.
                      sess bindOutput: aClass.
                      sess execute.

                      self executeWithProtection:
                              [[(ans := sess answer) == #noMoreAnswers]
                                      whileFalse: [ans == #noAnswerStream ifFalse:
[aList addAll: ans upToEnd]]]].

              "Whether or not there was an error, disconnect the session."
              sess notNil ifTrue: [sess disconnect]
```

On voit ici que le paramètre bindOutput reçoit ‘Parcelle new’, qui indique la création d’un nouvel objet Parcelle. Celui-ci contiendra tous les attributs de l’enregistrement sélectionné.

Une session peut avoir besoin de paramètres d’entrée (bindInput), pour effectuer les mises à jour dans la table. Ce paramètre reçoit alors une instance de la classe concernée. Voici le code pour les mises à jour des parcelles.

```
      updateOcsol: uneParcelle
        "Met à jour l'occupation du sol dans l'enregistrement correspondant à l'objet
parc dans la table ParcelleDyn."

        | conn sess |
        (conn := self connection) isNil ifTrue: [^Dialog warn: 'You are not connected
to a database system.'].
        self
                executeWithProtection:
                        [sess := conn prepare: 'update ParcelleDyn set ocsol = :ocsol
where parcelle_id = :id '.
                        sess bindInput: uneParcelle.
                        sess execute.
                        sess answer].
        sess notNil ifTrue: [sess disconnect]
```

:ocsol et :id font référence aux attributs de l’objet uneParcelle.

Partie II: Application au modèle

1. Le modèle conceptuel

Nous poserons que le pas de temps de la simulation sera l'année.

Un ensemble d'agents « Exploitation » réalise un parcours avec le troupeau sur les parcelles pâturables, limitant ainsi l'embroussaillement naturel de ses terres. Mais les parcelles voisines, abandonnées par leur propriétaire ont un effet néfaste de diffusion de l'embroussaillement. C'est pourquoi l'exploitant doit être obligé de défricher de temps en temps certaines de ses terres, afin d'augmenter la quantité de nourriture pour son troupeau, mais aussi de rendre accessibles ses terres carun trop grand embroussaillement empêche l'accès.

Ces agents sont représentés par les exploitants vivant sur les deux sections communales G et H de Saint Georges de Lévejac. Ils ont été identifiés : 17 exploitations possèdent des terres sur ces deux sections, mais 8 seulement habitent dans cette zone (dans le hameau de Soulages). Un de ces exploitants ne possèdent pas de troupeau, nous nous intéresserons donc aux 7 exploitants restants. Ces données sont dans la base ACCESS (stgeorges.mdb, table Parcelle et Exploitant).

1.1 le comportement des exploitants

Les exploitants de la zone étudiée font pâturer leur troupeau sur les parcelles en parcours. Pour accéder à une parcelle, l'exploitant a le droit de passer sur les chemins de la commune, et sur les parcelles dont il est le propriétaire ou qu'il exploite. Le passage sur une parcelle est cependant soumis à une condition : la parcelle ne doit pas être trop embroussaillée, sinon le troupeau ne peut pas circuler (On détermine cette accessibilité avec l'indice d'embroussaillement de la parcelle, il ne doit pas dépasser 30).

Les données nous montrent que les exploitants vivants dans cette zone habitent le hameau de Soulages. En utilisant les chemins communaux, on peut déterminer si une parcelle est accessible ou non depuis un chemin et de proche en proche, si l'indice d'embroussaillement le permet, connaître l'ensemble des parcelles accessibles par un exploitant.

Pour chaque parcelle, sa surface, son occupation du sol (ocsol), son indice d'embroussaillement (IE), le nombre d'unités fourragères par hectare (UF_selon_ocsol/ha) nous permettent de calculer son nombre d'UF ainsi :

$$UF_{parc}=surface*UF\_selon\_ocsol*(1-IE)^2$$

Plus la parcelle est embroussaillée, moins elle produit d'UF.

Pour chaque exploitant, on calcule son bilan fourrager : on somme le nombre d'UF (Unités Fourragères) qu'apportent les parcelles en parcours accessibles et les autres parcelles de cultures (orge et prairies temporaires). On calcule ensuite le besoin du troupeau, égal au nombre de têtes du troupeau fois le besoin d'une brebis (600UF/an).

Si le bilan est positif, l'exploitant réussit à nourrir correctement son troupeau, on lui augmente alors sa « cagnotte ». Si le bilan est négatif, la cagnotte diminue et l'exploitant essaie de défricher des parcelles en Parcours (diminuer IE pour faire augmenter UF_{parc}).

Même si la bilan est positif, les UF des parcours doivent correspondre au moins au tiers des apports totaux. Si ce n'est pas le cas, l'exploitant essaie de défricher, mais ne diminue pas sa cagnotte.

Le passage du troupeau sur un parcours provoque une diminution de l'embroussaillement (IE diminué de 1).

Les parcelles exploitées, mais dont les exploitants n'habitent pas la zone étudiée, seront considérées comme des parcelles « neutres ». Les exploitants entretiennent leurs terres (IE constant).

Les parcelles non exploitées seront considérées comme abandonnées, il n'y a aucun entretien, la nature est seule maître (processus de croissance de l'embroussaillement).

1.2 Le phénomène de Croissance/Diffusion

La croissance correspond à un embroussaillement naturel et intrinsèque d'une parcelle. Celle-ci se traduit par une augmentation régulière de 1 de l'IE. Au delà de 30, l'IE est tel que le troupeau ne peut plus la traverser, mais aussi cette parcelle devient semencière: on parle alors de diffusion.

La diffusion traduit la dispersion des semences d'une parcelle sur les parcelles voisines, qui englobe tous les processus de dispersion de semences par le vent, l'eau ou les animaux. Ces semences sont susceptibles de germer si l'occupation du sol de la parcelle cible est propice (les semences germeront facilement sur toutes terres travaillées puis abandonnées par la suite, ou bien sur des terres de type parcours, mais dont l'IE ne serait toutefois pas trop élevé, à cause de la compétition à l'accès aux ressources). La diffusion entraîne donc une augmentation de l'IE d'un maximum de 1 sur les parcelles cibles (le maximum de 1 évite un embroussaillement trop rapide d'une parcelle qui serait entourée par plusieurs parcelles semencières. On considère qu'on ne peut pas sommer la diffusion.).

Type Parcelle		Croissance	Diffusion	Exploitation
Exploitée	Parcours	+1	Au max +1 si IE<30	-1 si parcourue
	Terre Cultivée	+0	+0 la terre est trop travaillée	(-)
	Prairies	+0	+0	(-)
	Abandonnée	+1	+1	(-)
	Neutre	0	0	(-)

2. Préparation des données

2.1 Dans la base de données

1. Reclassification de l'occupation du sol: afin de simplifier les différents états des parcelles, un nouvel attribut a été rajouté. Celui-ci prend les valeurs suivantes: **P** pour les parcours, **TL** pour les terres en culture, **STH** pour les prairies et **X** pour les parcelles de bâtis, sols et non-déterminées.

2. De nouveaux attributs ont été ajoutés. Certains sont utiles uniquement afin d'obtenir l'équivalence avec les classes correpondantes de CORMAS (**UF**, **UF_selon_ocsol**, **acces**, **cagnotte**, **listeParcelle, voisinage**), d'autres sont nécessaires pour que ces données puissent attreindre ArcView (**IE**).

3. Les données initiales d'UF ont été calculées dans la base et ajoutées dans les enregistrements, suivant les chiffres des travaux INRA de M. Etienne (Unité Ecodéveloppement, Avignon).

4. Une copie de la table Parcelle appelée ParcelleDyn permettra de faire les mises à jour des données lors de la simulation, tout en gardant les données initiales intactes. Ainsi, la table Parcelle sera utilisée uniquement pour l'initialisation du modèle, tandis que ParcelleDyn sera utilisée pour toutes les autres opérations.

2.2 Dans ArcView

1. Création de la couche des chemins, correspondant aux interstices de la couche du parcellaire cadastral. Celle-ci a été réalisée en transformant la couche du parcellaire en une grille très fine (pas de 2m) avec le module Spatial Analyst de Arcview, et en reclassifiant les cellules des parcelles avec l'attribut "0" et toutes les cellules sans données avec la valeur 1. Cette couche a ensuite été transformée en fichier de forme (Shape polygone), donnant ainsi le "négatif" du parcellaire cadastral.
2. A cause d'erreurs dans le parcellaire cadastrales, les parcelles communales sont toutes utilisables par les exploitants, ce qui n'est pas aberrant dans la modélisation.

3. Les scripts utilisés dans ArcView.

4 scripts ont été créés pour le besoin du modèle. Ces scripts réalisent les analyses spatiales nécéssaires pour le fonctionnement du modèle, qui ne peuvent pas être réalisés dans CORMAS.

3.1 Calcul du voisinage des parcelles.

Ce script est une fonction qui possède un paramètre d'entrée qui est la parcelle dont on veut connaître le voisinage et donne en retour une chaîne de caractères. On effectue une sélection des parcelles à 10m de celle-ci et les identifiants de chacune de ces parcelles sont stockés dans une chaîne de caractères qui sera retournée.

```
'Script Voisine
projet = av.getProject
vue = projet.finddoc("Evolution")
letheme = vue.findtheme("EvolutionGH")
latable = letheme.getFtab

id=latable.findfield("identifian")
exploitant=latable.findfield("Exploitation_id")
parcelle=self
chaine=""

aBitmap=latable.getSelection
latable.Query("[identifian] = "+ parcelle.Quote, aBitmap, #VTAB_SELTYPE_NEW
)
latable.updateselection
letheme.selectbytheme(letheme,#FTAB_RELTYPE_ISWITHINDISTANCEOF,100,#VTAB_SELTYPE
_NEW)
for each rec in latable
 if (latable.getselection.get(rec)=true) then
        v=latable.ReturnValue(id,rec)
        chaine=chaine+" "+v.asString
 end
end
return chaine
```

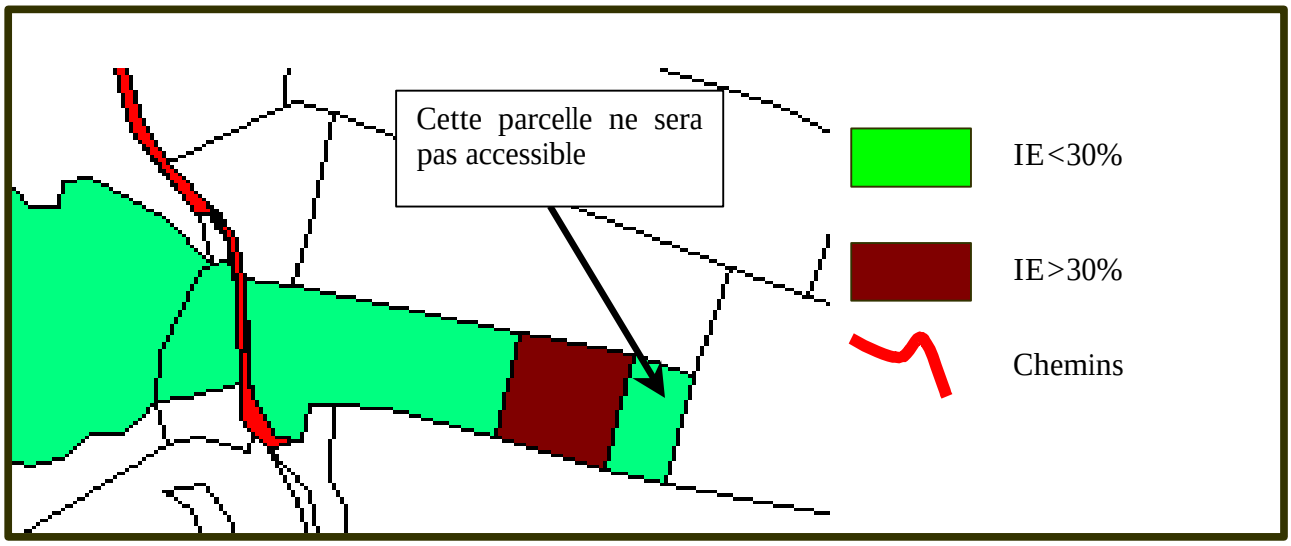
3.2 Calcul de l'accessibilité des parcelles.

Ce script est une procédure qui ne renvoie rien. Elle met à jour l'attribut *Accessible* qui appartient au fichier de forme du parcellaire cadastral. Etant donné que les exploitants concernés habitent Soulages et que tous les chemins sont connexes au hameau, une parcelle accessible depuis un chemin est accessible par un exploitant.

Le script détermine un arbre de connexité entre les parcelles et le chemin de la manière suivante: pour chaque exploitant, on réalise une première sélection des parcelles de cet exploitant se trouvant à 10m du chemin et dont l'IE ne dépasse pas 30. On réitère ensuite cet algorithme en ajoutant à chaque fois, la nouvelle sélection de parcelle correspondant à ces critères, jusqu'à ce que la sélection ne change plus. On obtient alors, pour un exploitant donné, un arbre de parcelles accessibles. Par cet algorithme, si une parcelle potentiellement accessible se trouve au delà d'une parcelle trop embroussaillée, elle ne peut pas être accessible (cf. schéma ci-dessous).

L'ensemble des parcelles appartenant à un exploitant regroupe les parcelles qu'il exploite et les parcelles dont il est le propriétaire, mais qui ne sont exploitées ni par lui, ni par un autre exploitant. C'est pour cela que l'on est obligé de faire une correspondance entre Exploitant et Propriétaire lors du calcul de l'accessibilité et lors de l'initialisation du modèle dans CORMAS.

E1	E2	E3	E4	E5	E6	E7
P5	P45	P154	P111	P114	P139	P170



Voici l'algorithme correspondant:

```
        'Script Accessibilite
        projet = av.getProject
        vue = projet.finddoc("Evolution")
        letheme = vue.findtheme("EvolutionGH")
        latable = letheme.getFTab
        leschemins = vue.findTheme("Chemin.shp")
        covgh=latable.findfield("covgh_id")
        acces=latable.findfield("accessible")
        exploitant=latable.findfield("Exploitation_id")
        latable.refresh
        listeExploitant= { "E1", "E2", "E3", "E4", "E5", "E6", "E7" }
        listeProprietaire= { "P5", "P45", "P154", "P111", "P114", "P139", "P170" }
        aBitmapIE=latable.getselection  'selection des parcelles IE<30%
        latable.Query(" [IE]<30 ", aBitmapIE, #VTAB_SELTYPE_NEW)
        aBitmapAcces=latable.getselection.clone
        aBitmapAcces.clearall
        aBitmap4=latable.getselection.clone
        aBitmap4.clearall
        aBitmapExploitant=latable.getselection.clone
        for each exploitation in listeExploitant

             aBitmapExploitant.clearall
             range=listeExploitant.find(exploitation)
             latable.Query(" [Exploitation_id] = """+exploitation+""""", aBitmapExploitant ,#VTAB_SELTYPE_NEW)
             latable.Query(" ([Exploitation_id]=""""") AND ([Proprietaire_id]=
"""+listeProprietaire.get(range)+""")", aBitmapExploitant,#VTAB_SELTYPE_OR)
             'latable.Query(" [ocsol] = ""P""", aBitmapExploitant ,#VTAB_SELTYPE_AND)
             latable.Query(" [proprietaire_id] = ""P29""", aBitmapExploitant ,#VTAB_SELTYPE_OR)

             latable.setselection(aBitmapExploitant)
             latable.updateselection
             'aBitmapExploitant est la selection des parcelles de l exploitant
             letheme.selectbytheme(leschemins,#FTAB_RELTYPE_ISWITHINDISTANCEOF,10,#VTAB_SELTYPE_AND)
             aBitmap2=latable.getSelection.clone
             aBitmap2.And(aBitmapIE)
             latable.setselection(aBitmap2)
             latable.updateselection
             while(true)
               letheme.selectbytheme(letheme,#FTAB_RELTYPE_ISWITHINDISTANCEOF,10,#VTAB_SELTYPE_NEW)
               aBitmap3=latable.getselection.clone
               aBitmap3.And(aBitmapExploitant)
               aBitmap3.And(aBitmapIE)
               latable.setselection(aBitmap3)
               latable.updateselection
               if (aBitmap3=aBitmap4) then
                   break
               else aBitmap4.copy(aBitmap3)
               end
             end
        end
          if (latable.StartEditingWithRecovery) then
          for each rec in latable
                if (aBitmap3.get(rec)=true) then
                   latable.beginTransaction
                   latable.SetValue(acces,rec,TRUE)
                   latable.EndTransaction
                else
                   latable.beginTransaction
                   latable.SetValue(acces,rec,FALSE)
                   latable.EndTransaction
                end
               end
          end
        latable.StopEditingWithRecovery(TRUE)
```

3.3 Requête d'accessibilité

Ce script retourne la liste des parcelles et leur accessibilité(0 ou 1) dans une chaîne de caractères de la forme: " P1 0 P2 0 P3 1 P4 0 …". Cette chaîne sera traitée par CORMAS dès sa réception pour y extraire les accessibilités de chaque parcelle et les mettre à jour.

```
'Script RequeteAcces2
projet = av.getProject
vue = projet.finddoc("Evolution")
letheme = vue.findtheme("EvolutionGH")

latable = letheme.getFtab

acces=latable.findfield("accessible")
parcelles=latable.findfield("identifian")

listeAcces=List.make
chaine=""
for each rec in latable
  if (latable.returnValue(parcelles,rec)<>"") then
    listbis={}
    chaine=chaine+latable.returnValueString(parcelles,rec)+" "
    if (latable.returnValueString(acces,rec)="true") then
         chaine=chaine+"1"
    else chaine=chaine+"0"
    end
    chaine=chaine+NL
  end
end

return chaine
```

4. Fonctionnement du modèle

On considérera les bases de la programmation Smalltalk dans CORMAS comme acquises. Pour plus de détails il vous est vivement recommandé de vous inscrire aux formations proposées par le CIRAD GREEN sur la présentation des systèmes multi-agents et de leur plate-forme de simulation CORMAS.

Le nom du modèle est StGeorges. Ceci s'interprète comme une catégorie de classe où l'on trouvera l'ensemble des classes nécessaires au fonctionnement de ce modèle. Les différentes classes de cette catégorie sont:

- *StGeorges* contenant les méthodes de la simulation (initialisation évolution et mise à jour de la base) et la liste des agents utilisés contenus dans diverses collections d'objets (lesParcelles, lesExploitants, parcelles abandonnées, parcelles exploitées, parcelles neutres). Cette classe contient aussi les méthodes de mise à jour de la base (**MAJDatabase**), de récupération des voisins d'une parcelle (**recupVoisin**) et du calcul de diffusion d'une parcelle (**diffusion:**).
- *Parcelle* représentant les agents passifs de la simulation, elle possède les mêmes attributs que les éléments de la table parcelle de la base de données. Ses méthodes se limitent aux méthodes d'accès aux attributs.
- *Exploitant* représentant les agents "communicants" de la simulation même s'ils ne communiquent pas vraiment. Ils agissent toutefois sur leur environnement qu'est l'ensemble des parcelles. Ses attributs sont aussi constitués des mêmes attributs contenus dans la table Exploitation de la base. En plus des méthodes d'accès aux attributs la classe exploitant possède une méthode **chercheDefriche** utilisée comme son nom l'indique pour rechercher les parcelles trop embroussaillées et les remettre en état.
- *DbConnect* contenant l'ensemble des méthodes de connexion et de manipulation de la base de données.

4.1 La méthode d'initialisation

Cette méthode est placée dans la classe StGeorges. Elle s'occupe de préparer la simulation et de mettre en place toutes les données. On peut la décomposer ainsi:

(a) Récupération du parcellaire cadastral en instanciant la classe *Parcelle* avec chacun des enregistrements de la table via une connexion par *dbConnect*. Ces instances de classe seront stockées dans la collection **lesParcelles**.

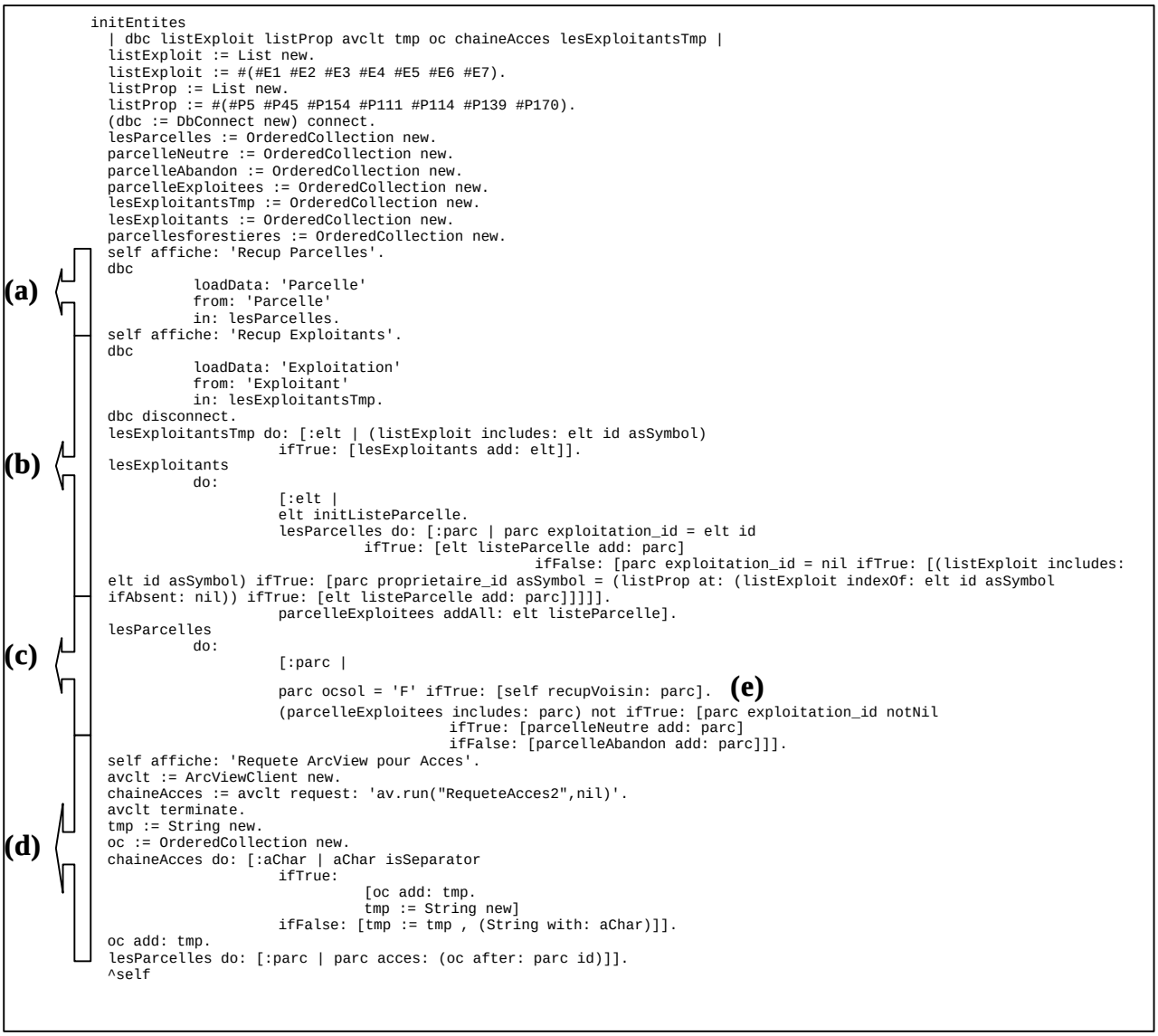
(b) Récupération des données de la table exploitation de la même manière, en instanciant la classe *Exploitant* et en les stockant dans la liste **lesExploitants**. L'attribut **listeParcelle** de chaque exploitant est mis à jour avec la collection de parcelles qu'il exploite ou dont il est le propriétaire.

(c) On réalise un tri sur les parcelles afin de créer les listes **parcellesExploitees** (exploitées par l'un de nos 7 exploitants), **parcelleNeutre** (exploitées par d'autres exploitants), **parcelleAbandon** (appartenant à un propriétaire, mais non exploitées).

(d) On lance la requête d'Accessibilité d'Arcview et l'on exécute la requête d'accessibilité. Les données reçues sont alors traitées et chaque parcelle est mise à jour (attribut **accès**).

(e) On lance la requête de voisinage pour toutes les parcelles forestières et on met à jour la liste **voisinage** des parcelles avec la méthode **recupVoisin**. Le voisinage sert uniquement aux parcelles susceptibles de diffuser; cette requête prenant du temps et afin d'accélérer l'initialisation, seules les parcelles concernées sont traitées. Cependant, pendant la simulation et le processus de diffusion, si la parcelle ne connaît pas encore son voisinage, la procédure recupVoisin sera lancée. Ce temps de calcul s'étale pendant la simulation.

Le code de la méthode d'initialisation est donné ci-dessous, les repères (de **a** à **f**) sont placés aux portions de code correspondantes:



```
initEntites
  | dbc listExploit listProp avclt tmp oc chaineAcces lesExploitantsTmp |
  listExploit := List new.
  listExploit := #(#E1 #E2 #E3 #E4 #E5 #E6 #E7).
  listProp := List new.
  listProp := #(#P5 #P45 #P154 #P111 #P114 #P139 #P170).
  (dbc := DbConnect new) connect.
  lesParcelles := OrderedCollection new.
  parcelleNeutre := OrderedCollection new.
  parcelleAbandon := OrderedCollection new.
  parcelleExploitees := OrderedCollection new.
  lesExploitantsTmp := OrderedCollection new.
  lesExploitants := OrderedCollection new.
  parcellesforestieres := OrderedCollection new.
  self affiche: 'Recup Parcelles'.
  dbc
(a)           loadData: 'Parcelle'
           from: 'Parcelle'
           in: lesParcelles.
  self affiche: 'Recup Exploitants'.
  dbc
           loadData: 'Exploitation'
           from: 'Exploitant'
           in: lesExploitantsTmp.
  dbc disconnect.
  lesExploitantsTmp do: [:elt | (listExploit includes: elt id asSymbol)
                     ifTrue: [lesExploitants add: elt]].
(b)
  lesExploitants
           do:
                     [:elt |
                     elt initListeParcelle.
                     lesParcelles do: [:parc | parc exploitation_id = elt id
                               ifTrue: [elt listeParcelle add: parc]
                                                   ifFalse: [parc exploitation_id = nil ifTrue: [(listExploit includes:
  elt id asSymbol) ifTrue: [parc proprietaire_id asSymbol = (listProp at: (listExploit indexOf: elt id asSymbol
  ifAbsent: nil)) ifTrue: [elt listeParcelle add: parc]]]]].
                     parcelleExploitees addAll: elt listeParcelle].
  lesParcelles
(c)        do:
                     [:parc |
                     parc ocsol = 'F' ifTrue: [self recupVoisin: parc].  (e)
                     (parcelleExploitees includes: parc) not ifTrue: [parc exploitation_id notNil
                                          ifTrue: [parcelleNeutre add: parc]
                                          ifFalse: [parcelleAbandon add: parc]]].
  self affiche: 'Requete ArcView pour Acces'.
  avclt := ArcViewClient new.
  chaineAcces := avclt request: 'av.run("RequeteAcces2",nil)'.
  avclt terminate.
  tmp := String new.
(d)
  oc := OrderedCollection new.
  chaineAcces do: [:aChar | aChar isSeparator
                     ifTrue:
                               [oc add: tmp.
                               tmp := String new]
                     ifFalse: [tmp := tmp , (String with: aChar)]].
  oc add: tmp.
  lesParcelles do: [:parc | parc acces: (oc after: parc id)]].
  ^self
```

4.2 la méthode d'évolution

Cette méthode est le cœur de la simulation. Elle sera appelée à chaque pas de temps. Trois parties se distinguent:

Une première phase où les exploitants parcourent les parcelles Parcours (P) accessibles et où le calcul de nombre d'UF rapportées est réalisé. On y ajoute aussi les UF récoltées par les terres cultivées et les prairies (TL et STH). On calcule alors le bilan fourrager en retranchant les besoins du troupeau. Si ce bilan est positif et si le nombre d'UF rapportées par les Parcours correspond au moins au tiers des apports globaux, alors la cagnotte de l'exploitant augmente (+1000). Si une des deux conditions n'est pas remplie, alors l'exploitant cherchera les parcelles susceptibles d'être défrichées ou remises en état (méthode *chercheDefriche*). L'exploitant sera pénalisé seulement si le bilan fourrager est négatif (-500 dans la cagnotte), pour exprimer les besoins d'achat de fourrage supplémentaire ou une perte de bénéfice sur la quantité de lait produit par son troupeau. Dans cette première phase, on fait aussi croître l'IE des parcelles exploitées (constituées uniquement des parcelles en Parcours et non accessibles), et des parcelles abandonnées. Les parcelles dites "neutres" n'évoluent pas.

La seconde phase décrit le processus de diffusion des parcelles. Toutes les parcelles dont l'indice d'embroussaillement est supérieur à 30 déclenchent cette fonction.

La troisième phase est optionnelle: il s'agit de la phase de mise à jour de la base et de l'accessibilité dans ArcView. Elle est déclenchée par un changement d'embroussaillement au point critique (IE=30).

```
evoluerExploitants: compteur
| bilanFourrage changementAcces avclt chaineAcces tmp oc UFParcours UFAutre UFTotal |
changementAcces := false.
lesExploitants
do:
          [:expl |
                    bilanFourrage := 0.
                    UFParcours := 0.
                    UFAutre := 0.
                    UFTotal := 0.
                    expl listeParcelle
                              do:
                                        [:parc |
                                        parc ocsol = 'P' ifTrue: [parc acces = 1
                                                  ifTrue:
                                                            [UFParcours := UFParcours + parc UF.
                                                            parc IE: parc IE - 1.
                                                            parc calculUF.
                                                            parc IE = 29 ifTrue: [changementAcces := true]]
                                                  ifFalse:
                                                            [parc IE: parc IE + 1.
                                                            parc IE = 30 ifTrue: [changementAcces := true]]].
                                        (parc ocsol = 'TL' or: [parc ocsol = 'STH'])
                                                  ifTrue:
                                                            [UFAutre := UFAutre + parc UF.
                                                            parc IE: 0.
                                                            parc calculUF]].
                    UFTotal := UFParcours + UFAutre.

                    bilanFourrage := UFTotal - (expl ovins * 700).
                    self affiche: expl id , ' Bilan Fourrager: ' , bilanFourrage printString.
                    (bilanFourrage >= 0 and: [UFParcours > (UFTotal / 3)])
                              ifTrue: [expl gain: 1000]
                              ifFalse:
                                        [bilanFourrage >= 0 ifTrue: [expl perte: 500].
                                        changementAcces := changementAcces or: [expl chercheDefriche]]].

parcelleAbandon do: [:parc | (parc IE > 0 and: [parc IE < 100])
          ifTrue:
                              [parc IE: parc IE + 1.
                              parc IE = 100 ifTrue: [parc ocsol: 'F']]].
self affiche: 'Diffusion'.
lesParcelles do: [:parc | parc IE >= 30 ifTrue: [changementAcces := changementAcces or: [self diffusion: parc]]].
lesParcelles do: [:parc | parc maj: 0].
changementAcces ifTrue: [self affiche: 'MAJ indispensable'].
changementAcces
          ifTrue:
                    [self affiche: 'MAJ Base'.
                    self MAJDatabase.
                    self affiche: 'Requete ArcView pour MAJ Accessibilite'.
                    avclt := ArcViewClient new.
                    avclt execute: 'av.run("Accessibilite",nil)' with: 80000.
                    chaineAcces := avclt request: 'av.run("RequeteAcces2",nil)'.
                    avclt terminate.
                    tmp := String new.
                    oc := OrderedCollection new.
                    chaineAcces do: [:aChar | aChar isSeparator
                                        ifTrue:
                                        [oc add: tmp.
                                        tmp := String new]
                                        ifFalse: [tmp := tmp , (String with: aChar)]].
                    oc add: tmp.
                    lesParcelles do: [:parc |parc acces: (oc after: parc id)]].
^self
```

4.3 Les méthodes annexes

D'autres méthodes sont utilisées afin d'alléger la présentation des deux méthodes principales. Mises à part les méthodes de dbConnect, les méthodes suivantes se rattachent soit au modèle (Classe StGeorges), soit à la classe Exploitant.

1. ***chercheDéfriche*** (Exploitant): Cette méthode a pour but de faire prendre une décision à l'exploitant qui doit faire quelque chose pour augmenter le nombre d'UF apporté par les Parcours. On autorise l'exploitant à remettre en état jusqu'à 5 parcelles dans l'année. Il procède par niveau d'embroussaillement des plus embroussaillées au moins embroussaillées et des accessibles au non accessibles. Dans le cas extrême où aucune parcelle ne serait améliorable, l'exploitant défriche une parcelle forestière. Dans le pire des cas, il diminue son troupeau de 10 têtes.

```
chercheDefriche
          | pastrouve changement nbParcelleMax nbParcelle |
          pastrouve := true.
          changement := false.
          nbParcelleMax := 5.
          nbParcelle := 0.
          self listeParcelle do: [:parc | parc ocsol = 'P' ifTrue: [parc IE > 29 & (nbParcelle <= nbParcelleMax) &
(parc acces = 1)
                                        ifTrue:
                                                  [self affiche: self id , ' remet en etat serieux' , parc id.
                                                  parc IE: 5.
                                                  parc calculUF.
                                                  nbParcelle := nbParcelle + 1.
                                                  pastrouve := false.
                                                  changement := true]]].
          nbParcelle <= nbParcelleMax ifTrue: [self listeParcelle do: [:parc | parc ocsol = 'P' ifTrue: [parc IE > 29 &
(nbParcelle <= nbParcelleMax) & (parc acces = 0)
                                                  ifTrue:
                                                            [self affiche: self id , ' remet en etat serieux' , parc id.
                                                            parc IE: 5.
                                                            parc calculUF.
                                                            pastrouve := false.
                                                            nbParcelle := nbParcelle + 1.
                                                            changement := true]]].
          nbParcelle <= nbParcelleMax ifTrue: [self listeParcelle do: [:parc | parc ocsol = 'P' ifTrue: [(parc IE > 15
and: [nbParcelle <= nbParcelleMax and: [parc acces = 1 and: [parc IE < 30]]])
                                                  ifTrue:
                                                            [self affiche: self id , ' remet en etat ' , parc id.
                                                            parc IE: 5.
                                                            parc calculUF.
                                                            nbParcelle := nbParcelle + 1.
                                                            pastrouve := false]]].
          nbParcelle <= nbParcelleMax ifTrue: [self listeParcelle do: [:parc | parc ocsol = 'P' ifTrue: [(parc IE > 15
and: [nbParcelle <= nbParcelleMax and: [parc acces = 0 and: [parc IE < 30]]])
                                                  ifTrue:
                                                            [self affiche: self id , ' remet en etat ' , parc id.
                                                            parc IE: 5.
                                                            parc calculUF.
                                                            nbParcelle := nbParcelle + 1.
                                                            pastrouve := false]]].
          pastrouve = true ifTrue: [self listeParcelle do: [:parc | parc ocsol = 'F' ifTrue: [parc IE > 50 & pastrouve
                                                  ifTrue:
                                                            [self affiche: self id , ' defriche la foret ' , parc id.
                                                            self gain: 200.
                                                            parc IE: 0.
                                                            parc ocsol: 'TL'.
                                                            parc UF_selon_ocsol: 4000.
                                                            parc calculUF.
                                                            nbParcelle := nbParcelle + 1.
                                                            pastrouve := false.
                                                            changement := true]]].
          pastrouve = true
                    ifTrue:
                              [self affiche: self id , ' Toujours negatif Diminution du troupeau'.
                              self ovins: self ovins - 10.
                              self gain: 500].
          ^changement
```

2. ***RecupVoisin: uneParcelle*** (StGeorges): Cette méthode utilise DDE pour récupérer la liste des voisins d'une parcelle donnée et la stocke dans l'attribut *voisin*.

```
recupVoisin: parc
          | voisine avclt chaineAcces tmp |
          voisine :=OrderedCollection new.
          avclt := ArcViewClient new.
          chaineAcces := avclt request: 'av.run("Voisine", "' , parc id , '")'.
          avclt terminate.
          tmp := String new.
          chaineAcces do: [:aChar | aChar isSeparator
                                  ifTrue:
                                            [voisine add: tmp.
                                            tmp := String new]
                                  ifFalse: [tmp := tmp , (String with: aChar)]].
          voisine add: tmp.
          parc voisinage: voisine.
```

3. ***MAJDatabase*** (StGeorges): Cette méthode met à jour les enregistrements de la base, en particulier les attributs IE, Ocsol et Acces. Elle utilise les mêmes

```
MAJDatabase
| aDbc  session |
        aDbc := DbConnect new connect.
        session := aDbc connection prepare:
           'UPDATE ParcelleDyn SET IE=:IE,
ocsol=:ocsol, acces=:acces WHERE parcelle_id = :id'.
        lesParcelles do: [:parc | session bindInput:
parc; execute; answer].
        session disconnect.
        aDbc connection commit.
        aDbc disconnect.
```

méthodes que la classe dbConnect, mais dans ce cas, la mise à jour a été optimisée afin d'accélérer la connexion.

4. ***Diffusion: uneParcelle*** (StGeorges): Cette méthode déclenche le processus de diffusion de uneParcelle sur ses parcelles voisines. On autorise une diffusion maximale de 1 par parcelle et par pas de temps, afin d'éviter une embroussaillement excessif en une seule fois. La diffusion n'est pas effectuée sur les parcelles dont l'occupation du sol est "X", car cela n'a pas beaucoup de sens (Rappel: "X" correspond aux battis, sols et parcelles non déterminées).

```
diffusion: parc
	"Calcule la diffusion des semences depuis une
parcelle semenciere sur ses parcelles voisines
	(uniquement sur les parcelles dont l'occupation de
sol n'est pas X)"

	| parcVoisine changement |
	changement := false.
	parc voisinage isNil ifTrue: [self recupVoisin:
parc].
	parcVoisine := lesParcelles select: [:parcelle |
parc voisinage includes: parcelle id].
	parcVoisine do: [:vois | (vois ocsol ~= 'X' ) &
(vois maj = 0) & (vois IE <100)
			ifTrue:
				[vois IE: vois IE + 1.
				vois calculUF.
				vois maj: 1.
				vois IE = 30 ifTrue:
[changement := true. self affiche: vois id , 'a un IE de
',vois IE printString.].
				]].
	^changement
```

5. Méthodes de traitement des chaînes de caractères en provenance d'un "request" DDE. Il permet de convertir une chaîne de caractères contenant des éléments séparés par des espaces ou des retours chariot, en une collection d'objets (OrderedCollection). Ainsi à partir de "P4 0 P1 1 P10 0", on obtient { "P4", "0", "P1", 1, "P10", "0"}.

```
	tmp := String new.
	 oc := OrderedCollection new.
	 laChaineATraiter do: [:aChar | aChar isSeparator
			ifTrue:
				[oc add: tmp.
				tmp := String new]
			ifFalse: [tmp := tmp , (String with:
aChar)]].

	'oc contient la collection
```

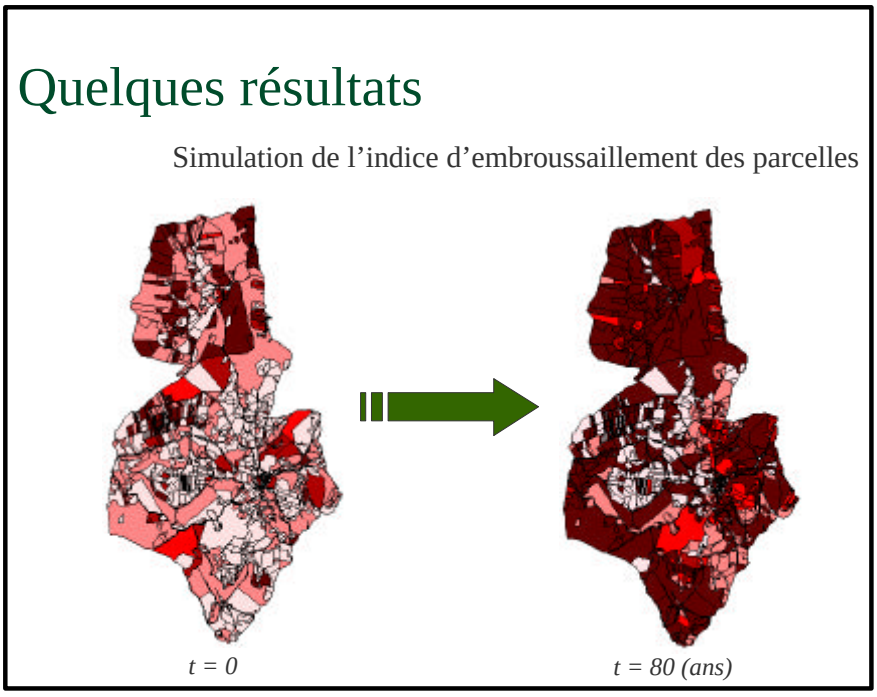
III. Résultats et Discussion

1. Résultats

Les résultats techniques sont satisfaisants: Le système fonctionne correctement, les délais d'attente sont acceptables (100 pas de temps /heure). La visualisation du résultat peut se faire pendant la simulation sans aucun problème.

En ce qui concerne les résultats de la simulation en elle-même, il en est autrement: le modèle n'est pas calé et a des comportements anarchiques. Les données fournies par la base ne sont pas complètes (en particulier les exploitants peuvent avoir d'autres terres ailleurs que sur les 2 sections communales étudiées). De plus les données ne sont pas sûres à 100%, quelques erreurs ont été remarquées. Le système est donc très instable au début de la simulation, puis se stabilise, jusqu'à ce que un grand nombre de parcelles devienne semencière et que les exploitants ne sachent plus où donner de la tête! Ces détails de modélisation seront corrigés par la suite.

Dans tous les cas, la visualisation est intéractive, moyennant un grand espace de bureau, il est possible de visualiser les cartes dans ArcView pendant la simulation. A la fin de celle-ci, on peut aussi voir les graphiques de l'évolution de la cagnotte des exploitants.



2. Discussion

On peut dire ce système à trois faces peut rendre beaucoup de services dans un certain type de simulation.

La base de données apporte la connaissance préalablement préparée, d'habitude si difficile à instancier dans un modèle. Les données de la base peuvent donc aisément remplacer les "Randoms-à-tout-va" qui sont très utilisés lorsque l'on veut représenter une population.

Le SIG apporte évidemment l'aspect spatial avec ses outils d'une part, et son esthétisme de représentation cartographique de l'autre. Il reste toujours le problème des limitations d'ArcView quant à ses possibilités de communication (DDE, et ODBC en lecture seule). Ces limitations nous oblige à jongler avec les données et à leur préciser d'un sens de circulation (ce qui n'aurait

pas été utile par exemple, si ODBC n'avait pas été en lecture seule), mais le résultat est toutefois là.

Les diverses perspectives de cet outil sont envisageables:

- ✓ Utiliser les outils de FORPAST, développés dans CORMAS
- ✓ Reconsidérer le comportement d'agents situés, difficilement utilisable dans ArcView et reconsidérer la grille spatiale de CORMAS
- ✓ Augmenter la palette d'outils de ArcView en ajoutant les fonctionnalités du module GeoProcessing.
- ✓ Revoir l'apport Raster de Spatial Analyst jusqu'ici délaissé.

D'un point de vue du modèle, on pourra envisager plusieurs modifications:

- ✓ Améliorer la prise décision des agents.
- ✓ Prendre en compte l'âge des exploitants.
- ✓ Améliorer le processus de diffusion qui ne cible pas les parcelles voisines, mais uniquement un *buffer* autour de cette parcelle (ce qui évitera qu'une petite parcelle puisse embroussailler une très grosse). Cette option avait été prise en compte au début de la réflexion du modèle, mais elle a dû être abandonnée par manque d'outils spatiaux (Spatial Analyst n'était pas en mesure de pouvoir mettre à jour les attributs de cellules sélectionnées). Il faudra encore une fois voir avec GeoProcessing s'il est possible de travailler avec des vecteurs.

3. Bilan

L'outil est acquis:

- ❖ Le lien CORMAS-SGBD est générique et pourra donc s'appliquer à d'autres situations (c.f. TP sur les connexions aux bases de données).
- ❖ Le lien CORMAS-SIG est générique, à condition de faire transiter les données importantes de l'un à l'autre. L'adéquation manque toutefois de facilité: la séparation Acteurs/Environnement n'est pas toujours une bonne solution.
- ❖ D'un point de vue des outils "annexes", ceux seront réutilisables dans d'autres simulations (Calcul de distance, recherche de voisinage, accessibilité(?).