Intégration de modèles de cultures sur des plates-formes de modélisation agronomique et environnementale. Application au modèle Stics

Jean-Claude Poupa

▶ **To cite this version:**

Jean-Claude Poupa. Intégration de modèles de cultures sur des plates-formes de modélisation agronomique et environnementale. Application au modèle Stics. [Rapport Technique] auto-saisine. 2008, 256 p. ⟨hal-01462358⟩

HAL Id: hal-01462358

https://hal.science/hal-01462358v1

Submitted on 6 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Intégration de modèles de cultures sur des plates-formes de modélisation agronomique et environnementale

Application au modèle Stics[1]

Jean-Claude Poupa

décembre 2008

[1]STICS (**S**imulateur mul**Ti**disciplinaire pour les **C**ultures **S**tandard) est un modèle de simulation d'une culture sur une parcelle à pas de temps journalier. **Nadine Brisson**, INRA, *Unité Climat, Sol et Environnement*, Avignon.

Le langage de programmation Fortran s'est pendant pendant longtemps imposé pour le calcul scientifique et, quelque cinquante ans après sa création, reste utilisé dans les laboratoires pour traduire des traitements numériques. Avant le développement des logiciels scientifiques dans les différentes disciplines, les sous-programmes Fortran ont largement été échangés dans la communauté scientifique. Des «packages» se sont ainsi constitués puis ont été diffusés dans les laboratoires. Plus récemment, «habillés» dans un environnement WINDOWS, certains sont devenus «logiciels», et continuent à évoluer dans leur langage natif sous le contrôle d'équipes de recherches, qui réalisent elles-mêmes spécifications et traductions informatiques. C'est le cas du modèle de simulation de culture *Stics*.

Faut-il jeter les logiciels anciens ?

L'intégration de ces produits comme composants dans des systèmes plus larges soulève des difficultés majeures. Ils apparaissent alors comme des «boîtes noires» qui ne peuvent communiquer qu'à travers des fichiers. Les solutions empiriques adoptées ont leurs limites dès lors que l'on sort du domaine applicatif initial.

Ainsi, le couplage d'un modèle de culture avec des modèles économiques pour évaluer des politiques environnementales se traduit par un changement d'échelle de la parcelle à la région qui requiert des fonctionnalités nouvelles. S'il est en plus nécessaire de simuler des évènements quotidiens, comme le climat ou des décisions techniques, il faut «ouvrir la boîte noire» pour accéder à la dimension chronologique.

Les apports de l'Informatique théorique

L'utilisation d'automates de traduction en langage C améliore la portabilité mais ne fait que reproduire une «boîte noire» codée autrement. La reprise du projet informatique avec les méthodes de développement classiques mobiliserait des ressources humaines et financières importantes.

Une autre solution est le recours aux instruments mathématiques de l'Informatique Théorique pour, simultanément, reconnaître la plupart des algorithmes effectifs et structurer les données. L'objectif est alors de pouvoir produire une spécification formelle, à partir de laquelle il devient ensuite possible de générer du code dans tout autre langage, et de gérer les paramètres du modèle dans des bases de données.[1]

Théorie des Langages Formels La reconnaissance des algorithmes s'appuie sur la Théorie des Langages Formels mais elle n'est efficace que si la programmation est relativement structurée, les branchements imbriqués se traduisant rapidement par une explosion combinatoire du nombre de blocs.

[1] **Jean-Claude Poupa**, *Cstics : couplage informatique de STICS avec des modèles extrinsèques*,
Séminaire Stics, Arles, 23-24 janvier 2003.

Il s'agit finalement de déterminer dans quel mesure le code source à analyser utilise un sous-ensemble du langage Fortran77 qui peut être généré par une grammaire algébrique, l'instruction de branchement appartenant à ce sous-ensemble en l'absence d'autres moyens pour traduire certaines structures syntaxiques usuelles dans la norme ANSI de ce langage, décrites en section 1.3.

Théorie des Graphes La structuration des données, nécessaire pour classer variables et paramètres d'une simulation, s'appuie sur la Théorie des Graphes, les relations entre les variables informatiques étant représentées par les arcs. Des algorithmes de cheminement sont utilisables pour réaliser des classifications. Pour un modèle dynamique, l'examen des graphes définis pour les sous-programmes peut permettre de séparer variables endogènes et exogènes. Parmi ces dernières, il est encore nécessaire d'isoler les séries temporelles externes, lesquelles gèrent des décisions techniques et des évènements aléatoires.

Définir un environnement de développement

Un environnement de développement a été constitué dans le monde du logiciel libre pour mettre en œuvre ces méthodes, qui relèvent des techniques de compilation usuelles, avec les outils de programmation du projet GNU[2] sous le système LINUX. Définies autour des versions successives de *Stics*, plusieurs commandes de compilation permettent d'analyser du code source Fortran77 utilisé en calcul scientifique, pour reconnaître des structures syntaxiques génériques, puis les traduire dans d'autres langages.

Ces commandes produisent des fonctions C compilables, immédiatement utilisables pour construire des applications mais qui peuvent aussi être reprises dans un environnement orienté objet.

Reconnaître les algorithmes Une première étape a permis d'effectuer la reconnaissance des algorithmes et d'isoler les sections de code non produites par la grammaire algébrique génératrice. Un code C a été généré sur la base d'une gestion dans des tables globales des variables communes Fortran, préalablement réparties en groupes dans des fichiers de déclarations pour les besoins du projet $Hysafe$.[3] Les fonctions C ont été reprises une à une pour réintroduire explicitement les traitements non reconnus et corriger les situations ambiguës. Elles sont appelées depuis un module de supervision qui lit les paramètres du modèle dans les anciens fichiers *Stics*, reprend et complète les contrôles et initialisations, puis enchaîne les séquences d'évaluations quotidiennes sur la période de simulation. La première version $Hysafe$, opérationnelle depuis septembre 2003, est utilisée

[2] **Mike Loukides, Andy Oram**, *Programmer avec les outils GNU*, ÉDITIONS O'REILLY, 1997.

[3] **Christian Dupraz, Isabelle Lecomte**, *Modélisation des interactions arbre-culture par couplage de STICS à un modèle arbre spatialisé 3D : principe et architecture informatique*, Séminaires Stics, Arles 23-24 janvier 2003, Carry-le-Rouet 17-18 mars 2005.

par différentes équipes.[4]

Ce volet algorithmique a pour objectif de produire un code dans un langage de plus haut niveau avec

- le remplacement des branchements par les structures de contrôle usuelles,
- l'amélioration du typage des données,
- le prototypage des fonctions,
- la vectorisation.

Il s'agit finalement de construire, à partir du code Fortran, des composants élémentaires ensuite utilisables dans l'environnement de développement du projet GNU avec le compilateur C/C++.

Structurer les données Une seconde étape analyse, au cours de cette même reconnaissance syntaxique, le rôle des variables informatiques communes dans le contexte des instructions pour construire les classifications. La structuration de cet ensemble de milliers de variables *Stics* banalisées et identifiées par des noms propres répond à plusieurs besoins.

1. Gérer les paramètres du modèle, plante, sol, climat et itinéraire technique, dans des bases de données.
   Séparer les constantes utilisées pour évaluer ces paramètres.
2. Définir complètement les conditions initiales, état des variables exogènes et valeurs initiales des variables endogènes, avant la simulation.
3. Conserver la dimension temporelle des variables endogènes pour pouvoir gérer des évènements quotidiens, et simuler des décisions.

Cette étape doit permettre de préciser les formalismes en regénérant les «équations» de *Stics*, pratiquement des ensembles d'expressions arithmétiques et logiques restituées avec les notations mathématiques usuelles.

L'objectif est cette fois de définir une architecture informatique afin de pouvoir définir des règles de génération du code en fonction des objectifs applicatifs.

Une heuristique décrite dans un document technique

Si les méthodes utilisées relèvent des mathématiques appliquées, la mise en œuvre concrète suit le flot des instructions avec des suites de problèmes pratiques à résoudre. Ce document reprend la chronique de ce processus, avec des détails techniques et de nombreuses notes contextuelles qui signalent des situations ambiguës ou inattendues. Démarré avec la version 4 de *Stics* et géré sous LaTeX, il est mis à jour en permanence avec les versions successives de *Stics*, jusqu'à la version 6 actuelle.

Les instruments informatiques réalisés, décrits dans les chapitres suivants, sont disponibles sous forme d'un enchaînement de commandes qui génèrent finalement des fonctions C pour chaque unité de programme Fortran, et un ensemble de messages qui énumèrent des points à examiner ou traduire explicitement.

[4] **Jean-Claude Poupa, Dominique Ripoche**, *Automatic production of an algorithmic representation of the STICS model in view of its implementation at the regional scale*, European Society for Agronomy, VII ESA congress, Copenhagen, Denmark, 11-15 july 2004.

D'autres fonctions sont générées pour gérer les dictionnaires STICS, et accéder ainsi aux formalismes du modèle de culture.

Des applications C sont ensuite fabriquées à partir de cette bibliothèques de fonctions. Des outils annexes ont été développés pour pouvoir vérifier l'équivalence des évaluations Fortran et C en cours d'exécution.

Ces instruments sont également utilisables pour restructurer le code Fortran des versions en cours.

Bien que développés autour du code de *Stics*, et au départ pour une application ciblée, des solutions génériques ont été adoptées afin que ces instruments puissent être repris et adaptés par des équipes informatiques pour traiter d'autres logiciels Fortran (1.1.8).

Des composants mobilisables dans des environnements de programmation par objets

Les fonctions compilées sont utilisées dans différents projets comme méthodes natives dans des langages objets. Les algorithmes et structures de données étant traduites de façon très classique en langage C, il est facile de générer aussi des codes pour les langages JAVA ou C++, qui reprennent ces formes syntaxiques : le résultat serait la disponibilité d'un environnement de programmation par objets initialisable avec le code Fortran des versions courantes de STICS.

Chapitre 1

Analyse syntaxique globale du code source Fortran

Le code source Fortran du logiciel *Stics* est une traduction dans un langage informatique d'un modèle de culture. L'analyseur syntaxique décrit dans ce chapitre permet simultanément :

- de reconnaître les déclarations et instructions utilisées pour répertorier les formes syntaxiques effectivement adoptées ;
- d'isoler les variables communes du modèle et de préciser leurs types et dimensions ;
- accessoirement de décrire des formats d'édition communs ;

Les algorithmes mis en œuvre sont définis à partir de la description formelle de ce langage proposée par une équipe d'ingénieurs du *Service Informatique et Mathématiques Appliquées de la Direction des Études et Recherches d'Électricité de France.*[1]

Cette étape de précompilation produit, pour une version donnée, un ensemble de fichiers indissociables, utilisés dans les étapes suivantes : elle se déroule en trois phases pratiquement à travers trois commandes successives.

1. **S1** effectue une première analyse syntaxique et un prétraitement du code ;
2. **S2** construit le dictionnaire des variables communes ;
3. **S5** produit les descriptions des structures de contrôle et formats ;

1.1 Reconnaissance des instructions

Apparu en 1954, le langage Fortran est antérieur aux langages formels qui se sont développées au cours des années 60 à l'issue des travaux de **Noam Chomsky**. La version normalisée par l'ANSI, dite Fortran 77, a repris ce cadre

[1] **FORTRAN77 Guide pour l'écriture de programmes portables**, ouvrage réalisé sous la direction de **Françoise Ficheux-Vapné**, ÉDITIONS EYROLLES, 1985.

théorique, avec des constructions syntaxiques générées par des grammaires formelles à côté des formes anciennes.

1.1.1 Notations syntaxiques

Les structures syntaxiques utilisées sont décrites en utilisant les notations classiques pour la description des grammaires formelles, à savoir :

les variables syntaxiques non terminales sont encadrées par les caractères < et > ;
la dérivation est traduite par la flèche $\longrightarrow$;
une séquence optionnelle présente une seule fois est encadrée par les crochets ouvrant [et fermant] ;
une séquence optionnelle répétable est encadrée par les accolades ouvrante { et fermante } ;
un choix est traduit par la barre verticale | ;

1.1.2 Les "paquets de cartes"

Une instruction Fortran était initialement codée sur une carte de 80 colonnes : si le support physique est aujourd'hui disparu, le mode de représentation demeure.

La présence d'un caractère en colonne 1, avant ou après une instruction, signifie que la carte contient un commentaire. Si ce caractère est usuellement la lettre `c`, d'autres sont admis, avec des variantes en fonction des compilateurs.

Si la carte ne contient pas un commentaire, elle est structurée comme suit.

- Les cinq premières colonnes codent une étiquette numérique, ou contiennent des espaces en l'absence d'étiquette.
- La sixième colonne, dite "colonne suite", gère les instructions longues. La présence d'un caractère, initialement un chiffre ou usuellement la lettre `S`, signifie que l'instruction se poursuit sur la carte courante.
- Les instructions et déclarations Fortran sont codées des colonnes 7 à 72.
- Les huit dernières colonnes sont réservées et ignorées par le compilateur.

Des corrections à risques La ligne de longueur variable a ensuite succédé à la carte. Si la longueur des commentaires sur la ligne n'est plus bornée, la reconnaissance des instructions pose de nouveaux problèmes.

- L'utilisation de la tabulation horizontale introduit une ambiguïté, ce caractère étant traduit par un décalage de plusieurs espaces physiques.
- Une correction en dehors d'un environnement de développement Fortran risque de provoquer un décalage à droite au-delà de la position 72, et une troncature effective pour le compilateur mais invisible sous un éditeur classique.

Les solutions adoptées vont finalement de dépendre des compilateurs et produits de développements utilisés et il apparaît comme dangereux de modifier du code Fortran sous un éditeur quelconque.

1.1.3 Les éléments lexicaux

L'alphabet du langage est constitué
- de l'ensemble des 26 lettres sous la forme majuscule, auquel est ajouté le tiret souligné (_), utilisé comme séparateur dans un mot,
- des 10 chiffres décimaux,
- de 14 caractères spéciaux (`espace = + - * / ( ) , . ' " : $`).

Des extensions apparaissent souvent en fonction des compilateurs.

Des tabulations horizontales sont parfois présentes : elles sont remplacées par des espaces pour pouvoir interpréter l'ancienne image de carte.

Les lettres minuscules sont souvent admises mais les représentations majuscules et minuscules sont équivalentes : l'instruction `GOTO` peut s'écrire indifféremment `goto` ou `Goto`, voire `GO TO` ou encore `Go To`, avec en sus un nombre variable d'espaces.

Les lettres, les chiffres et le tiret souligné sont utilisables pour construire des identificateurs, le premier caractère devant être une lettre.

Les mots-clés sont des identificateurs réservés.

Les étiquettes sont des nombres d'au plus cinq chiffres.

Les identificateurs et les nombres sont séparés par au moins un caractère spécial, avec éventuellement un nombre variable d'espaces.

1.1.4 Production d'une forme normalisée

La variabilité syntaxique précédente, avec les variantes propres aux compilateurs, rend difficile le repérage des instructions, d'où la nécessité de produire une forme canonique, dite normalisée.

Normalisation syntaxique

Pour ce faire, les lettres sont converties sous la forme majuscule et les espaces séparateurs inutiles supprimés. On aboutit ainsi à un code compact, sur lequel il est possible de rechercher des motifs pour localiser une instruction ou un élément syntaxique particulier.

Les chaînes de caractères constantes sont normalement encadrées par des quotes. La présence de deux quotes consécutives signifie que le texte contient une apostrophe, ou qu'il est vide. Il en est de même pour le guillemet s'il est utilisé comme délimiteur, son doublement signifiant qu'il est effectivement présent dans le texte. Les textes placés entre quotes ou guillemets dans une instruction sont édités tels quels, sans conversion majuscule ni suppression des espaces, avec les caractères présents.

Instructions et commentaires

Les instructions codées sur plusieurs lignes sont concaténées : la fin de l'instruction est repérée par la marque de fin de ligne. Chaque instruction est repérée par son rang.

Les commentaires sont restitués avec la marque # en début de ligne. Ceux introduits entre les cartes suites sont regroupés après l'instruction reconstituée.

Les noms des fichiers lus sont ajoutés en commentaires : la marque # est doublée, balise ensuite utilisée pour numéroter les fichiers des déclarations et définir les groupes des variables communes de la section 1.2.5.

Contrôles syntaxiques

Les anomalies syntaxiques examinées sont successivement :

les instructions Fortran, après concaténation des lignes *suite*, de plus de LFORTRAN = 1024 caractères ;
les caractères non autorisés dans l'alphabet retenu ;
une chaine de caractères avec quote droite non trouvée et ligne suite absente ;
un fichier source qui ne se termine pas par une marque de fin de ligne ;
l'existence d'un champ supplémentaire entre la fin de ligne et l'instruction END.

Les autres erreurs se traduisent par un déroutement, pratiquement pour des problèmes d'accès aux fichiers ou pour des anomalies de programmation.

1.1.5 Forme d'une instruction

Les tabulations horizontales lues sont préalablement remplacées par une suite de huit espaces dans les tampons de lecture. Si la ligne lue n'est pas un commentaire, l'élément de rang 6 est examiné pour déterminer s'il s'agit d'une instruction ou de la suite de l'instruction précédente.

Une instruction peut être étiquetée : la ligne démarre alors par un nombre $n < 99999$. L'instruction ou sa suite sont codés dans la zone instruction, à partir de l'élément de rang 7. Une instruction est introduite par un identificateur, nom dont le premier caractère est une lettre, mot-clé du langage Fortran ou variable d'une opération d'affectation. Les espaces sont à ce stade conservés et l'identificateur est recherché. Une instruction Fortran est finalement

1. une opération d'affectation dans
   - une variable simple si l'identificateur est immédiatement suivi de l'opérateur d'affectation =,
   - une table si la parenthèse fermante de l'expression parenthèsée est suivi de ce même opérateur = ;
2. un ordre, une déclaration ou une directive identifiée par un mot-clé ;

Le mot-clé IF introduit les instructions conditionnelles : il est immédiatement suivi d'une expression logique entre parenthèses. Deux formes syntaxiques sont utilisées pour une instruction Fortran. Si ↓ représente la marque de fin de ligne,

1. IF (< *expression* >) THEN↓
   dite *if structuré*, correspond à la structure conditionnelle des langages formels et sa présence est de bonne augure ;

2. IF ($< expression >$) $< instruction >\downarrow$
   dite *if logique*, exécute, si l'expression est vraie, soit une affectation, soit une instruction introduite par un mot-clé autre que IF ;
3. IF ($< expression >$) $< nombre1 >, < nombre2 >, < nombre3 >\downarrow$
   dite *if arithmétique*, instruction honnie qui gère des branchements, non utilisée dans *Stics*.

L'analyse se poursuit pour la seconde forme avec la recherche de l'identificateur de l'instruction conditionnelle, mot-clé autre que IF ou variable.

Les mots-clés ainsi reconnus par l'analyseur sont rangés dans un lexique : des statistiques sont éditées en fin d'exécution.[2]

La lecture de l'instruction se poursuit jusqu'à la fin de la ligne, éventuellement au-delà de la position 72, et les cartes suites éventuelles sont concaténées.[3]

Boucles DO...ENDDO La norme du langage *Fortran77* gère les boucles à pas constant dans une structure classique qui utilise les étiquettes pour délimiter le corps de la boucle (1.3.2), avec selon l'usage l'instruction finale étiquetée CONTINUE. Les nouvelles versions de STICS introduisent une nouvelle structure dans laquelle les mots-clés DO et ENDDO remplacent les étiquettes.

Pour éviter de traiter cette forme particulière, qui n'apporte pas d'autres fonctionnalités, et les sections critiques liées (1.3.3), l'ancienne structure est restituée par ajout d'étiquettes fictives[4] derrière le DO, et devant l'instruction CONTINUE qui remplace alors le ENDDO.

1.1.6 Unités de programmes et déclarations

Les fichiers suffixés par ``.for'' contiennent le code Fortran des programmes, sous-programmes ou fonctions, lesquelles définissent les unités de programmes, um même fichier pouvant contenir plusieurs unités.

Les autres fichiers utilisés pour générer le logiciel, suffixés par ``.inc'', sont inclus via la directive INCLUDE dans les unités de programmes avant compilation. Ils contiennent ici les déclarations des variables communes et une liste de formats d'édition communs.

La version *Stics* d'avril 2006 est composée 131 fichiers sources, dont 21 inclus : les 120 fichiers de code regroupent 138 unités de programmes.

Unités de programmes

Les instructions Fortran des unités de programmes sont numérotées et un code normalisé est généré pour l'ensemble de ces unités : le numéro d'identification attribué est ensuite utilisé dans les fichiers compte-rendus de toutes

[2] La présence du mot-clé S dans certaines versions résulte d'un décalage à droite pour coder une carte suite (1.1.2) : il peut alors être nécessaire de remplacer une tabulation pour retrouver la structure initiale et reconnaître la carte suite.

[3] La structure décrite plus haut est gérée en amont, le code source Fortran ne devant normalement pas être modifié.

[4] La première étiquette porte le numéro 6666, les autres étant attribuées par incrémentation.

les étapes et finalement pour localiser les instructions Fortran dans le code C généré.

Reconnaissance d'une unité Le nom d'une unité de programme Fortran est normalement déclaré dans une première instruction :

- PROGRAM `monprogramme` pour le programme principal, cet ordre non obligatoire permettant de fixer le nom du programme indépendemment du compilateur ;
- SUBROUTINE `maprocedure` pour les sous-programmes ;
- [< *type* >] FONCTION `mafonction` pour les fonctions.

Les noms des unités sont normalement uniques : si un nom est répété, il est suffixé par le numéro du module courant pour assurer cette unicité.[5] Le nom ANONYME est utilisé pour les programmes non nommés, variantes de programmes principaux.

Une unité se termine normalement par l'instruction END qui peut néanmoins être absente : elle est alors ajoutée dans le code normalisé pour faciliter les traitements ultérieurs.[6]

La dernière instruction n'est pas toujours suivie de la marque de fin de ligne : cette dernière est alors ajoutée.[7]

Lexique des unités Un lexique des unités de programmes est constitué : il va permettre de piloter les traitements dans les étapes suivantes.

Un unité de programme est décrite par le quadruplet

$$(type, nom, nmodule, debut, fin).$$

- *type* code le type d'unité avec les lettres S pour SUBROUTINE, P pour PROGRAM, F et T respectivement pour FUNCTION non typée et typée, ? pour inconnu.
- *nom* est le nom de l'unité.
- *nmodule* contient le rang du fichier source.
- *debut* et fin indiquent les rangs des instructions de début et fin pour la forme normalisée.

Le lexique des unités est l'ensemble de ces quadruplets : on dénombre 103 unités pour la version 5 et 138 pour la version 6.

Fichiers inclus de déclarations

Les déclarations sont numérotées indépendemment des unités de programmes et les rangs ne sont utilisés que pour les localiser dans les comptes-rendus propres à cette étape.

[5] LECUSM est déclaré dans trois fichiers pour la version 6, d'où les trois unités LECUSM, LECUSM_95 et LECUSM_96.

[6] La dite instruction est toujours présente dans la version 6.

[7] L'absence de cette marque se traduit par l'apparition de nouveaux mots à l'issue des concaténations de fichiers, soit "END...".

Les déclarations de variables sont regroupées dans un fichier propre et les formats mis à part : c'est la présence du mot-clé FORMAT qui pilote l'édition dans le fichier annexe.

Le code déclaratif est normalisé de la même façon pour être ensuite repris dans cette première étape avec les commandes **S2** pour la construction du dictionnaire des variables communes et **S5** pour l'intégration des formats d'édition.

1.1.7 Traduction informatique

Les commandes s'exécutent depuis un répertoire choisi à partir d'un nom générique d'application et d'un ensemble de suffixes prédéfinis.

Les modules sources Fortran sont rangés dans un répertoire fils. Pour l'application *stics*, la liste est éditée par la commande

```
ls *.for *.inc >stics.liste
```

Ce fichier '`stics.liste`', recopié dans le répertoire d'exécution, va permettre de démarrer la précompilation.

Les modules cités sont lus et traités, sauf si :

1. le premier caractère de la ligne est la marque %,
   ceci pour pouvoir choisir le code source d'une unité de programme ou ignorer des modules ;[8]
2. le nom de l'unité de programme est suffixé par la marque '_',
   laquelle repère les sous-programmes ajoutés pour la mise au point, comme '`Suivi_.for`' (3.1.3) ;

Un fichier optionnel, '`stics.banni`', peut être créé pour fournir une liste de mots-clés pour lesquels les instructions ne sont pas traitées, mais restituées en commentaires.

La commande

S1 <*application*> <*repertoire fortran*> [<*marque*|]

produit des fichiers pour les étapes suivantes.

Les erreurs importantes peuvent se solder par un déroutement.[9]

Repérage de commentaires L'argument optionnel `marque` est un code textuel utilisé pour identifier des commentaires dans le code Fortran, par exemple les mises à jour 'JCP', les unités de programmes concernées et les commentaires qui suivent étant alors édités dans le fichier

– `stics.JCP`.

[8] Trois exemplaires du sous-programme LECUSM sont proposés : 'lecusmCASetPURE.for', 'lecusmCAS.for', 'lecusmPURE.for'.
La fonction EFFETTASSSURINFIL était présente dans 'effettasssurinfil.for' et 'tassement.for' dans la version d'avril 2006.
Le fichier 'comfruit.inc' n'est jamais inclus et contient l'unique variable DEVJOURFR déclarée dans 'common.inc'.

[9] Outre les fichiers inaccessibles, il s'agit généralement de débordements pour le traitement d'une nouvelle version : il faut alors revoir les valeurs de certaines constantes puis de recompiler la commande.

Fichiers repris dans les étapes suivantes

Cette précompilation produit un code normalisé utilisé pour la suite des traitements et un premier lexique des unités de programmes.

Code normalisé La forme normalisée est éditée dans trois fichiers :

- `stics.ins`, code normalisé pour toutes ces unités ;
- `stics.dcl`, code normalisé pour les déclarations des variables communes ;
- `stics.fmt`, code normalisé des formats d'édition ;

Les noms des fichiers sont reproduits derrière la double marque ##.

Les instructions sous la forme normalisée sont précédées du rang de l'instruction.

Les commentaires sont introduits par la marque #.

La marque de fin de ligne en fin de fichier est ajoutée en cas d'absence.

La directive Fortran END est aussi introduite si nécessaire pour les unités de programmes.

Localisation des unités de programmes La liste des unités de programmes Fortran, avec les rangs des premières et dernières instructions dans le code normalisé, est copiée dans le fichier

- `stics.upf`.

Restitution de la forme normale par unités de programme Le code normalisé des unités de programmes, certaines étant regroupées dans un même module source, est édité dans le répertoire fils `Fsource`, avec un fichier par unité de programme, suffixé par ``.f''.[10]

Comptes-rendus

Deux fichiers sont produits :

- `stics.cr1`, fichier compte-rendu global ;
- `stics.S1`, compte-rendu détaillé : ensemble des avertissements ;

Le compte-rendu global dénombre les mots-clés du langage Fortran : la version 6 introduit ainsi les nouvelles instructions IMPLICIT, PARAMETER, REWIND et ENDDO. Les inclusions de fichiers sont également répertoriées, et les présences vérifiées.

Le compte-rendu détaillé contient des messages précédés du rang de l'instruction avec entre crochets un code d'erreur,[11] soit pratiquement pour la version 6 :

- l'absence des marques de fin de lignes dans les modules sources,
- la non déclaration des types des fonctions,

[10] Un rang est ajouté pour distinguer les fichiers des programmes sans noms, soit "`anonyme_`n`.f`".

[11] La version 5 utilise des caractères qui n'appartiennent pas à l'alphabet initial, pratiquement les symboles < et > : un code normalisé est néanmoins généré et les problèmes posés sont traités plus loin (note 89, section 3.8.7).

– le renommage de sous-programmes homonymes (LECUSM) déclarés dans plusieurs modules.

Gestion du code Fortran

Plusieurs fichiers sont produits pour refabriquer une version Fortran et gérer les relations avec le code C.

Make Fortran Deux autres fichiers sont édités pour produire le *makefile* qui va permettre de fabriquer la version Fortran avec le compilateur GNU *g*77.

- `stics.make` contient les règles de dépendance qui déclenchent la compilation des sous-propgrammes et fonctions pour produire les modules objet ;
- `stics.macro` fournit les macro-instructions utiles, générées automatiquement vu le grand nombre de fichiers sources ;

Les programmes principaux sont exclus : la règle est à ajouter en fonction de l'application (annexe A).

Correspondance avec le code source Fortran Trois fichiers gèrent le lien avec le code des fichiers sources :

- `stics.module` contient les rangs des modules sources de la liste `stics.liste` ;
- `stics.module.for` et `stics.module.inc` établissent la correspondance entre le rang de l'instruction sous la forme normalisée et le numéro de ligne dans le fichier source Fortran (première si carte suite) pour chacun des suffixes terminaux ;

Comparaison des versions

Un utilitaire, *Sversion*, permet de comparer deux versions successives : il signale les modules modifiés, ajoutés ou supprimés.

1.1.8 Utilisation pour d'autres logiciels

Initialement produits pour restructurer le logiciel STICS et en extraire des composants, ces instruments peuvent être utilisables pour d'autres logiciels Fortran produits de façon analogue dans des laboratoires de recherches. Si la chaîne de traitement est opérationnelle avec l'ensemble des mots-clés de ce logiciel, les séquences de traitement sont à développer pour prendre en compte d'autres modes de programmation. En l'absence de ces séquences, un mécanisme est mis en place pour pouvoir utiliser ces instruments dans d'autres contextes.

Masquage d'instructions Un premier passage va permettre de répertorier les mots-clés utilisés. Il est alors possible de ne traiter qu'un sous-ensemble du code en plaçant les instructions non traitées en commentaires dans cette phase de prétraitement, au moyen du fichier '`S1.banni`' lu dans le répertoire courant s'il existe par la commande **S1**.

Masquage des entrées-sorties Les instructions de lecture et écriture sont traduites pour *Stics* avec les formats rencontrés pour pouvoir effectivement comparer les exécutions des codes Fortran et C mais, vu la complexité de ces traitements dans le langage Fortran, les solutions adoptées ne sont pas génériques comme pour les structures de contrôle, et liées au contexte applicatif. En conséquence, le traitement de ces instructions risque fort de générer des milliers de messages pour d'autres logiciels.

Une solution est alors d'exclure ces instructions d'entrée-sortie, pratiquement READ, WRITE et FORMAT, et de traiter le problème avec d'autres moyens mieux adaptés. Ces instruments restent ainsi utilisables pour restructurer les séquences d'évaluation.

1.2 Dictionnaire des variables communes

Les variables Fortran sont, comme les fonctions, des noms symboliques construits avec les lettres, les chiffres et le tiret souligné : le premier caractère est une lettre et les formes majuscule et minuscule sont équivalentes.[12]

Les variables utilisées pour décrire les formalismes de *Stics* et utilisées dans les unités de programmes Fortran, déclarées dans des fichiers suffixés par ``.inc'', ne sont accessibles que dans les unités dans lesquelles ces fichiers sont explicitement inclus au moyen de la directive INCLUDE.

Le code normalisé issu du traitement de l'ensemble de ces fichiers de déclarations est édité dans le fichier suffixé par ``.dcl''. Le nom du fichier d'origine figure dans un commentaire introduit par la marque ## (1.1.4) et les rangs de ces fichiers sont repris dans le dictionnaire comme numéro de groupe.[13]

1.2.1 Déclarations des types des variables

Les calculs numériques sont définis sur des ensembles de nombres, entiers relatifs ou réels. Les domaines de définition sont habituellement précisés par la notion de *type*, `entier` ou `réel`. D'autres domaines sont proposés par les langages de programmation usuels, tant pour les nombres que pour d'autres objets, comme les chaînes de caractères. A ces domaines sont associés des modes de représentation en mémoire : le compilateur doit donc connaître le type de la variable référencée.

Les variables du logiciel *Stics* sont définies exclusivement sur les quatre types suivants[14] :

INTEGER pour les nombres entiers ;
REAL pour les nombres réels dits simple précision ;
CHARACTER pour les chaînes de caractères ;

[12] *QNplante* et *qnplante* désignent bien la même variable *Stics*, qui devient QNPLANTE sous la forme normalisée.

[13] Certains groupes peuvent être vides, parce qu'ils ne contiennent aucune déclaration comme ``messagesMain.inc'', ou seulement des formats comme ``format.inc''.

[14] Le type REAL*8, dit double précision, n'est utilisé que pour quelques variables locales.

LOGICAL pour les variables logiques qui prennent les valeurs TRUE ou FALSE.

Dans le langage Fortran, la déclaration du type de variable n'est pas obligatoire. La convention implicite suivante est alors appliquée : la variable est entière si la première lettre de son identificateur appartient à l'ensemble {I,J,K,L,M,N}, réelle sinon.[15]

1.2.2 Dimensions des variables

Une variable, déclarée ou non, peut être une variable simple ou une table à une ou plusieurs dimensions, avec des bornes inférieures et supérieures.

Tables numériques Les dimensions des tables numériques sont déclarées dans des expressions parenthèsées qui suivent les noms de variables dans l'une de ces listes :

- déclaration explicite de type ;
- déclaration de commun (1.2.3) ;
- instruction DIMENSION ;

Un élément d'une telle liste est décrit par les règles syntaxiques

$< variable > \longrightarrow < nom > [(< dim > \{, < dim >\})]$
$< dim > \longrightarrow [< nombre >:] < nombre >$

Le composant $< nom >$ est le nom symbolique de la variable alors que $< nombre >$ code les bornes inférieure et supérieure.

Cultures associées Les tables numériques sont jusqu'à la version 5 des vecteurs ou des matrices. La version 6 introduit une pseudo-dimension binaire pour gérer dans les variables communes des couples de valeurs.

Chaînes et tables textuelles La forme précédente est reprise pour les variables textuelles mais une dimension supplémentaire peut aussi être introduit par le caractère '*' :

- dans la déclaration du type, soit CHARACTER$* < nombre >$,[16]
- à droite du nom de variable, soit $< nom > * < nombre >$.[17]

Les valeurs gérées dans ces tables sont des codes textuels ou des mots qui sont à distinguer des textes proprement dits, généralement des messages à éditer : les premières (≤ 32 caractères) sont dites *textes courts* et les secondes (> 32 et ≤ 256 caractères) *textes longs.*

[15] L'instruction IMPLICIT permet de modifier cette règle : la version 6 l'utilise pour la supprimer avec l'option NONE.

[16] CHARACTER*7 RESSUITE0

[17] CHARACTER CAR*11

1.2.3 Communs

Les variables Fortran sont rémanentes dès lors qu'elles sont citées dans le sous-programme ou la fonction dans une liste ordonnée introduite par l'instruction COMMON.[18] Ces listes, dites "communs", sont la plupart du temps désignées par des identificateurs, dits "étiquettes", placés entre deux barres obliques : l'absence d'étiquette correspondant à une liste particulière, dite "commun blanc", identifiée dans le dictionnaire par l'étiquette 'BLANC'.[19] Les communs peuvent être vus comme des tables dans lesquelles sont rangées séquentiellement les valeurs des variables citées, partagées seulement entre les unités de programmes dans lesquelles les déclarations sont incluses.

Variables communes

Ces variables citées dans les communs sont dans la suite appelées communes, en opposition aux autres variables appelées locales qui disparaissent après exécution de l'unité de programme.

Les communs qui gèrent les variables *Stics* sont déclarés dans les fichiers d'inclusion : les variables citées dans les listes sont localement consultables et modifiables dès lors que ces fichiers sont inclus.

Ces fichiers d'inclusion sont repérés par leurs rangs d'archivage, lesquels sont utilisés pour déterminer si les variables communes sont accessibles (2.1.3) et définir une première classification (2.4).

Synonymies

L'équivalence entre des variables rangées dans des communs étiquetés ou le commun blanc est basée exclusivement sur la position dans la liste, le nom de variable n'ayant qu'une valeur symbolique dans l'unité où le commun est déclaré.

Il est ainsi possible de nommer l'élément de rang i d'un tableau T sous laforme $T(i)$ dans une unité, x dans une autre : il suffit, si les types des variables sont identiques, que les rangs dans les listes des communs respectifs soient les mêmes, les deux désignations pointant alors sur le même emplacement mémoire.

Extension des communs par concaténation de listes

Les communs sont aussi utilisés pour regrouper des variables par thèmes. Si un commun est cité plusieurs fois dans une unité de programme, les listes de variables sont concaténées.

[18] Cette instruction obsolète avait pour fonction de regrouper les variables partagées dans une zone mémoire réservée sur les machines des années 60 : l'identification est basée sur le rang en mémoire.

[19] La zone vide qui apparaît dans la version 6 (//) désigne aussi le "commun blanc".

Fichiers d'inclusion Si un commun apparaît dans plusieurs fichiers d'inclusion, les listes de variables peuvent alors être différentes dans des unités de programme qui n'incluraient pas tous les fichiers requis : ces communs homonymes sont recherchés et la présence simultanée des fichiers à inclure est contrôlée.

Variables locales des fichiers inclus

Dans la version 6, les variables sont déclarées avant d'être citées dans les listes des communs : toute variable ainsi déclarée mais non citée dans un commun est locale dans une unité de programme qui inclut le fichier des déclarations, sauf pour les pseudo-constantes de la section suivante.

1.2.4 Pseudo-constantes

La version 6 de *Stics* introduit l'instruction PARAMETER qui permet de déclarer et d'initialiser des pseudo-constantes.[20] Ces symboles sont gérés comme les autres variables et sont décrites dans le dictionnaire comme appartenant à un commun fictif identifié par l'étiquette 'PARAMETER'.

1.2.5 Description des variables communes

Un dictionnaire des variables communes est construit pour les étapes suivantes. Chaque rubrique contient au moins le quintuplet

$$(nom, groupe, type, dim, commun)$$

dans lequel

- *nom* est le nom de la variable informatique ;
- *groupe* correspond au rang du fichier d'inclusion ;
- *type* code le type, positif si déclaré, négatif si implicite, soit entier $(1, -1)$, réel $(2, -2)$), logique(3), texte court (4), texte long (5) ;
- *dim* précise le nombre de dimensions, nul seulement pour une variable numérique simple, toute variable textuelle étant vue comme une table de caractères ;[21]
- *commun* contient le nom du commun ;[22]

Si $(dim > 0)$, les couples $(binf, bsup)$ sont ajoutés pour gérer les bornes inférieure et supérieure propres à chaque dimension, la borne inférieure étant unitaire par défaut dans le langage Fortran.

Ce dictionnaire est classé dans l'ordre alphabétique, le tiret souligné étant l'ultime caractère.

1.2.6 Traduction informatique

La commande

[20] AS = 1 et AO = 2.

[21] Les tailles des tables sont fixées par des constantes pour les types textes court et long.

[22] BLANC pour le commun blanc.

lit le fichier ‘‘`stics.dcl`’’ produit par **S1** et édite
- le dictionnaire dans ‘‘`stics.dic`’’ ;
- la liste des étiquettes des communs dans ‘‘`stics.eti`’’ ;
- les noms des fichiers inclus de déclarations dans ‘‘`stics.inc`’’ ;
- un compte-rendu dans ‘‘`stics.cr2`’’ ;
- des avertissements dans ‘‘`stics.S2`’’ ;
- la localisation des déclarations dans ‘‘`stics.cit`’’, avec
  - les rangs des citations dans la forme normalisée,
  - le nom du fichier inclus entre crochets,
  - le nom du commun entre barres obliques.

Comptes-rendus d'exécution

Le passage à la version 6 s'est traduit par des modifications notoires.

Version 5 Neuf fichiers de déclarations contiennent 208 communs qui regroupent 1997 variables communes dont 1654 sont explicitement déclarées, les 343 autres l'étant implicitement.

Les deux communs NB1003 et CLIMATB2 apparaissent dans deux groupes différents : le contexte est analysé dans l'étape de traitement des noms symboliques (2.1.3). les listes associées doivent être :
- concaténées si les deux groupes sont présents dans le sous-programme ou la fonction ;
- mises en correspondance pour gérer des synonymies si un seul groupe est présent ;
- ignorées si les deux groupes sont absents ;

La longueur maximale des variables textuelles est de 200 caractères.

Version 6 Les 20 fichiers de déclarations retenus (note 8) contiennent 202 communs qui regroupent 1680 variables toutes explicitement déclarées.

La présence des pseudo-constantes initialisées par l'instruction PARAMETER est signalée et ces variables communes, souvent utilisées en indices, sont à initialiser explicitement dans l'étape de génération du code.

La longueur maximale des variables textuelles passe à 400 caractères.

1.3 Structures de contrôle

Les structures de contrôle sont des entités syntaxiques fondamentales des langages informatiques de la classe des **langages algébriques**. Elles regroupent des ensembles ordonnés d'instructions, lesquelles s'exécutent séquentiellement, avant ou après évaluation d'un **prédicat**, traduit dans une **expression logique**.

Les structures fondamentales sont respectivement :

le **schéma conditionnel** qui traite une alternative ;

le **schéma itératif** qui effectue une action tant que la valeur de l'expression logique demeure vraie.

La traduction de ces structures dans le langage Fortran ne s'effectue pas toujours directement au moyen d'instructions appropriées. Si les nouvelles versions ont introduit de telles instructions, les schémas conditionnels et itératifs sont souvent gérés au moyen de branchements, soit parce que le langage n'offre pas l'instruction requise, soit pratiquement parce que la technique du branchement demeure efficace.

Des programmes Fortran peuvent néanmoins sous certaines conditions être décrits par des grammaires génératrices de langages algébriques, lesquelles traduisent en fait le respect de quelques règles simples de programmation : il est alors possible de produire à partir du code Fortran initial un code convenablement structuré dans un autre langage informatique.

La faisabilité dépend effectivement des pratiques de programmation et certaines propriétés sont alors à rechercher. Il faut en particulier vérifier que les blocs d'instructions décrits dans les structures de contrôle sont exécutés en totalité ou pas du tout, ce qui exclut les branchements depuis ou vers l'extérieur : ces séquences de code sont dites **sections critiques**.

Les branchements délimitent des blocs d'instructions notés entre accolades $\{< bloc >\}$.

Toutes ces structures sont décrites sous formes de **relations**, avec des **attributs** définus sur des **domaines** numériques entiers, ou textuels : les n_uplets sont édités dans des fichiers relus dans les étapes suivantes pour reconstituer les algorithmes.

Les appels des unités de programmes, ainsi que les entrées et sorties, sont traités dans cette section.

Cultures associées La version 6 a introduit une dimension supplémentaire binaire à un certain nombre de variables communes pour pouvoir gérer les couples de valeurs en présence de cultures associées : la reconnaissance des structures de contrôle liées à la gestion de ces couples est effectuée dans cette étape.

1.3.1 Schéma conditionnel

Le schéma conditionnel est introduit par une instruction qui débute avec le mot-clé IF.

If structuré

La norme ANSI du langage a introduit une nouvelle instruction conditionnelle qui répond aux exigences de la programmation structurée. La description syntaxique s'effectue cette fois sur un ensemble d'instructions d'un module, avec la syntaxe suivante

IF $(< condition >)$ THEN

{< *bloc*1 >}
[ELSE
{< *bloc*2 >}]
ENDIF

Cette forme est reprise dans les autres langages, avec suppression du ENDIF inutile puisque la marque de fin de bloc est présente, soit

IF (< *condition* >) THEN {< *bloc*1 >} [ELSE {< *bloc*2 >}]

Les ensembles d'instructions précédents peuvent contenir d'autres instructions composées, dont des *if structuré*, ce qui se traduit par plusieurs niveaux d'imbrications : les mots-clés IF et ENDIF ont alors le même rôle que les parenthèses ouvrante et fermante dans une expression.

Retours impératifs dans le schéma conditionnel La version 6 de *Stics* introduit une nouvelle structure avec retour impératif vers l'unité appelante à travers l'instruction RETURN (1.3.3), soit

IF (< *condition* >) THEN
{< *bloc*1 >}
RETURN
ELSE
{< *bloc*2 >}
RETURN
ENDIF

Cette forme est reconnue pour pouvoir être traduite sous la forme d'une instruction conditionnelle suivie d'un unique et ultime retour inconditionnel, soit pratiquement une factorisation qui réduit la complexité, et supprime les ruptures de séquence dans les sections critiques (1.3.4). Elle s'écrit alors

IF (< *condition* >) THEN {< *bloc*1 >} ELSE {< *bloc*2 >} RETURN ;

Ces formes peuvent également être contenues dans les blocs : la factorisation remplace toutes les instructions instructions RETURN finales des blocs imbriqués par une seule.

Représentation formelle Cette structure syntaxique est finalement décrite par le quintuplet

$$(module, alors, sinon, finsi, jesors)$$

dans lequel

module est le nom de sous-programme ou fonction ;
alors contient le numéro de l'instruction conditionnelle ;
sinon désigne l'instruction ELSE si elle existe, vaut 0 sinon ;
$finsi$ correspond à l'ultime instruction ENDIF ;
$jesors > 0$ indique que le retour est impératif, et dénombre les imbrications ;

Forme traduite avec des branchements

La version 6 introduit une nouvelle forme traduite par des branchements (3.7.2), soit

IF $<$ *cond* $>$ GOTO $n1$
$<$ *bloc*1 $>$
GOTO $n2$
$n1$ $<$ *instruction* $>$
$<$ *bloc*2 $>$
$n2$ $<$ *instruction* $>$

structure qui s'écrit formellement

Si $\neg$ $<$ *cond* $>$ **alors** $<$ *bloc*1 $>$ **sinon** $<$ *bloc*2 $>$;

et peut être traduite avec le *if structuré* fortran.

If logique

Le if logique est une instruction simple dont la syntaxe est la suivante :

IF ($<$ *condition* $>$) $<$ *instruction* $>$

L'instruction à droite est une instruction exécutable autre que IF et DO, éventuellement une affectation : le caractère qui suit la parenthèse fermante est en conséquence une lettre. Si le mot qui débute par cette lettre n'est pas le mot-clé THEN, il s'agit bien d'un if logique.

L'instruction lue sur la ligne est exécutée si l'expression logique est vraie, le traitement se poursuivant en séquence sinon. Si cette instruction est un branchement,

IF ($<$ *condition* $>$) GOTO $<$ n $>$
{$<$ *bloc* $>$}
$<$ n $>$ $<$ *instruction* $>$

il peut être supprimé et la séquence traduite sous la forme conditionnelle classique

IF $\neg$($<$ *condition* $>$) {$<$ *bloc* $>$}

Choix multiple

Les choix multiples sont parfois traduits par la répétition des formes

IF ($<$ *condition* $>$) THEN
{$<$ *bloc* $>$}
GOTO $<$ n $>$
ENDIF

ou encore

IF ($<$ *condition* $>$) GOTO $<$ n $>$
$<$ *instruction* $>$

suivie de l'instruction finale

$< n >$ CONTINUE

Si la valeur de l'étiquette finale n ne varie pas, c'est un choix multiple.

Ces formes syntaxiques sont souvent utilisées dans les sous-programmes de *Stics*, avec des centaines de branchements, explicites ou liés aux options des ordres de lecture Fortran (END $=< n >$ ou ERR $=< n >$).[23]

1.3.2 Schéma itératif

Les itérations sont habituellement réparties en trois groupes :
- Boucle *Pour*, à pas constant ;
- Boucle *Tant que*, contrôlée par un prédicat initial ;
- Boucle *Jusqu'à*, contrôlée par un prédicat final ;

Si le langage Fortran 77 gère la première forme avec l'instruction DO, il n'existe pas d'instructions dans la norme ANSI pour traiter les deux autres formes,[24] et il faut recourir aux branchements.
L'utilisation de ces branchements depuis des boucles *Pour* introduit un quatrième groupe :
- Boucle *Pour et Jusqu'à*, à pas constant avec sortie si le prédicat final n'est plus vrai ;

Les formes effectivement rencontrées dans le code de *Stics* et reconnues sont décrites dans cette section.[25]

Négation des conditions logiques La forme $\neg(< cond >)$ est efficace mais peu lisible.

Les expressions logiques ont en conséquence eté reconstruites en appliquant les règles de la logique formelle pour les opérations binaires,[26] traduites dans les couples opposés
- $(=, \neq), (<, \geq), (>, \leq)$ pour les opérateurs de comparaison,
- $(\vee, \wedge)$ pour les opérateurs de relation.

1. Itération à pas constant

Cette structure de contrôle utilise une variable de boucle i, deux bornes inférieure inf et supérieure sup, et un pas p. Elle est classiquement notée

Pour i depuis inf jusqu'à sup, pas p, faire $\{< bloc >\}$ fait.

[23] k répétitions pour k valeurs possibles de α génèrent $k(k+1)/2$ blocs.
$k = 285$ dans LECSORTI, $k = 191$ dans LECPLANT.

[24] Les instructions WHILE, DO WHILE, DO UNTIL sont propres à certains compilateurs.

[25] Les nouvelles formes apparues dans la version 6 ont été ajoutées pour permettre la génération automatique du code C.

[26] L'opérateur $\neg$ n'est pas utilisé pour les opérations binaires dans le code Fortran mais seulement sur les variables logiques, lesquelles sont transformées en variables numériques : l'opération unaire Fortran devient binaire.

Cette forme d'itération est traduite dans le langage Fortran par l'instruction composée dont la syntaxe est la suivante :

DO $< n > i = inf, sup[, pas]$
$\{< bloc >\}$
$< n > \{< instruction\ finale >\}$

Cette instruction composée traduit le schéma itératif à pas constant si le corps de la boucle, section critique, ne contient pas de branchements vers l'extérieur, généralement en avant, ce qui provoque la sortie de boucle.

Dans le cas où $(sup < inf)$, la boucle ne devrait pas être exécutée mais les résultats peuvent varier d'un compilateur Fortran à l'autre.

S'il est d'usage d'utiliser l'instruction vide CONTINUE comme instruction finale, le langage admet toute autre instruction, laquelle appartient au corps de la boucle : la nature de cette instruction finale est codée dans un attribut spécifique, *vide*, unitaire pour l'instruction vide, nul sinon.

Une boucle DO est représentée simplement par le quadruplet

$$(module, debut, fin, vide),$$

debut et fin localisant les première et dernière instructions.

2. Itération controlée par un prédicat initial

Cette itération est usuellement notée

Tant que $(< condition >)$ faire $\{< bloc >\}$ fait.

L'ensemble des instructions est exécuté tant que l'évaluation de l'expression logique rend vrai.

Il n'existe pas de mot-clé WHILE pour piloter cette structure dans la norme ANSI du langage Fortran, bien que certains compilateurs reconnaissent un DO WHILE. Elle peut être traduite sous la forme

$< n >$ IF $(< condition >)$ THEN
$\{< bloc >\}$
GOTO $< n >$
ENDIF

La présence d'un if structuré repéré par une étiquette conduit à rechercher en aval une instruction de branchement inconditionnel à cette étiquette suivie de la ligne ENDIF : avec le mot-clé d'usage WHILE des autres langages, cette itération s'écrit simplement

WHILE $(< condition >)$
$\{< bloc >\}$

3. Itération controlée par un prédicat final

Cette forme relègue l'évaluation de l'expression logique à la fin de la boucle et est notée

RÉPÉTER $\{< bloc >\}$ JUSQU'À $(< condition >)$.

Il n'existe pas de mot-clé UNTIL pour piloter cette structure dans la norme ANSI du langage Fortran, bien que certains compilateurs reconnaissent un DO UNTIL. Elle peut être traduite sous la forme

$< n >$ $< instruction >$
    $\{< bloc >\}$
    IF $< condition >)$ GOTO $< n >$

ou encore

$< n >$ IF $(< condition1 >)$ THEN
    $\{< bloc >\}$
     IF $(< condition2 >)$ GOTO $< n >$
    ENDIF

La présence d'un IF LOGIQUE qui pilote un branchement conditionnel arrière conduit à rechercher l'existence de cette structure de contrôle.

Avec les mots-clés DO et UNTIL,[27] ces itérations s'écrivent simplement

DO
    $\{< instruction >$
    $< bloc >$
    $\}$ UNTIL $(< condition >)$

et dans le dernier cas

DO
    $\{< instruction >$
    $< bloc >$
    $\}$ UNTIL $((< condition1 >) \wedge (< condition2 >))$

Le branchement arrière peut aussi être l'ultime instruction d'un bloc conditionnel final, soit

$< n >$ $< instruction >$
    $\{< bloc1 >\}$
     IF $(< condition >)$ THEN
    $\{< bloc2 >\}$
    GOTO $< n >$
    ENDIF

ce qui peut être traduit

[27] C'est la structure DO-WHILE du langage C.

DO
$\{< instruction >$
$< bloc1 >$
IF $(< condition >)$
$\{< bloc2 >\}$
} UNTIL $(< condition >)$

Ce dernier bloc conditionnel peut lui-même être contenu dans un autre bloc conditionnel (4.5.14), soit

$< n > < instruction >$
$\{< bloc1 >\}$
IF $(< condition1 >)$ THEN
$\{< bloc2 >\}$
IF $(< condition2 >)$ THEN
$\{< bloc3 >\}$
GOTO $< n >$
ENDIF

alors traduite avec $(< condition1 >) \wedge (< condition1 >)$ comme prédicat de contrôle de l'itération.

Des formes plus complexes introduisent un branchement conditionnel, par exemple

$< n > < instruction >$
$\{< bloc1 >\}$
IF $(< condition1 >)$ THEN
$\{< bloc2 >$
IF $(< condition2 >)$GOTO $< n >\}$
ELSE
$\{< bloc3 >\}$
ENDIF

alors traduit

DO
$\{< instruction >$
$< bloc1 >$
IF $(< condition1 >)$
$\{< bloc2 >\}$
ELSE
$\{< bloc3 >\}$
} UNTIL $((< condition1 >) \wedge (< condition2 >))$

4. Itération à pas constant avec sortie par branchement

Les itérations à pas constant avec sorties par branchements, signalées dans les comptes-rendus comme structures problématiques, sont finalement reconnues

et traduites sous les formes classiques, sans branchement. La reconnaissance s'effectue lors de l'entrée dans la boucle *Pour* pour les trois formes suivantes, exclusives.

i) Boucle *Tant que*

L'instruction Fortran d'initialisation d'une boucle à pas constant est quelquefois suivie d'un branchement conditionnel en sortie de boucle, soit la forme

DO $n1$ ...
IF $<$ *cond* $>$ GOTO $n2$
$<$ *bloc*1 $>$
$n1 <$ *instruction* $>$
$<$ *bloc*2 $>$
$n2 <$ *instruction* $>$

alors traduite comme boucle *Tant que*,
- la variable de boucle étant initialisée hors boucle,
- incrémentée à la fin du bloc itératif,
- le prédicat de contrôle construit,
- une conditionnelle finale ajoutée si $<$ *bloc*2 $>$ est présent.[28]

ii) Boucle *Jusqu'à* avec sortie conditionnelle

L'ultime instruction d'une boucle à pas constant est parfois un branchement conditionnel en sortie de boucle, soit la forme

DO $n1$ ...
$<$ *bloc*1 $>$
IF $<$ *cond* $>$ GOTO $n2$
$n1 <$ *instruction* $>$
$<$ *bloc*2 $>$
$n2 <$ *instruction* $>$

alors traduite comme boucle *Jusqu'à*,
- la variable de boucle étant initialisée hors boucle à la valeur de la borne inférieure diminuée du pas,
- incrémentée au début du bloc itératif,
- le prédicat de contrôle construit,
- une conditionnelle finale ajoutée si $<$ *bloc*2 $>$ est présent.

Le branchement conditionnel peut aussi être l'ultime instruction d'un bloc conditionnel, soit la forme

DO $n1$ ...
$<$ *bloc*1 $>$
IF $<$ *cond* $>$ THEN
$<$ *bloc*2 $>$
GOTO $n2$
ENDIF
$<$ *bloc*3 $>$
$n1 <$ *instruction* $>$

[28] Note 195 section 4.5.16.

traduite de la même façon comme boucle *Jusqu'à*, sans omettre l'exécution de $< bloc3 >$ dans le bloc conditionnel *sinon* si $< cond >$ est faux.

iii) Boucles *Jusqu'à* imbriquées

La version 6 introduit des boucles *Pour* imbriquées avec une unique sortie des deux boucles depuis la boucle interne,[29] soit

DO $n1 \ldots$
$< bloc1 >$
DO $n2 \ldots$
$< bloc2 >$
IF $< cond >$ GOTO $n3$
$n2 < instruction >$
$< bloc3 >$
$n1 < instruction >$
$< bloc4 >$
$n3 < instruction >$

alors traduite dans deux boucles *Jusqu'à* interne et externe gérées comme précédemment, avec en sus
– une conditionnelle en sortie de boucle interne si $< bloc3 >$ est présent.[30]

1.3.3 Ruptures de séquences

Une rupture de séquence dans un bloc peut être introduite par plusieurs instructions Fortran :

GOTO n traduit un branchement explicite à l'étiquette n ;

RETURN, retour de sous-programme ou fonction vers l'appelant, est assimilable à un branchement à la fin de ce sous-programme ;

END $= n$, option de l'ordre READ, renvoie vers l'étiquette n si la marque de fin de fichier est rencontrée.

ERR $= n$, option des ordres READ, WRITE et OPEN, provoque également le branchement à l'étiquette n, souvent vers une séquence de déroutement qui conduit à l'arrêt du programme demandé par l'instruction STOP.

La présence simultanée de ces deux dernières options génère deux branchements, soit deux références pour une même instruction.

Une rupture de séquence dans un module est finalement décrite par le quintuplet

$$(module, type, source, cible, n)$$

dans lequel

module contient le nom du module ;
type code le type de branchement ;

[29] Cette forme gère l"accès aux couches des horizons du sol (4.5.10).
[30] Ce bloc gère l'initialisation de la première couche de l'horizon suivant.

source et *cible* reçoivent les rangs des instructions référencées par les étiquettes de branchement ;
n, dénombre les branchements consécutifs vers une même étiquette, pour gestion des choix multiples ;

Les types de branchements sont les suivants :

branchement conditionnel	1
branchement inconditionnel	2
fin de fichier	3
erreur lecture ou écriture	4
retour conditionnel	5
retour inconditionnel	6

1.3.4 Ruptures de séquences dans les sections critiques

La section critique, définie dans un module, est un bloc d'instructions dont l'exécution est contrôlée par une expression logique, ou prédicat :

1. toute rupture de séquence qui se traduit par une sortie du bloc indépendemment de la valeur du prédicat de contrôle est interdite ;
2. toute entrée dans le bloc non contrôlée par ce même prédicat est également interdite ;

Ces ruptures de séquences provoquées par des instructions de branchements vers des étiquettes sont reprises telles quelles par les traducteurs informatiques dès lors que le langage cible dispose des instructions équivalentes, comme le langage C.

Reconnaissance des structures de contrôle

Si toutes les voies d'entrée ou de sortie des sections critiques font effectivement référence à la valeur de l'expression logique associée, la structure de contrôle liée définit un objet syntaxique d'un langage algébrique, lequel peut alors être traduit automatiquement dans un langage informatique structuré. Si tel n'est pas le cas, les adaptations sont à faire au vu du code source.

Représentation des ruptures de séquences dans les sections critiques

Les ruptures de séquences dans les sections critiques sont décrites par le septuplet

$$(module, tscri, premier, dernier, type, source, cible)$$

dans lequel

module contient le nom du module ;
tscri code la structure de contrôle génératrice ;
premier et *dernier* localisent la section critique ;
type, *source* et *cible* décrivent le branchement ;

Les structures de contrôle sont codées dans *tscri* comme suit :

if structuré : bloc THEN	1
if structuré : bloc ELSE	2
boucle DO	3
branchement inconditionnel : référence avant	4
retour inconditionnel	5
branchement conditionnel : référence avant	6
retour conditionnel	7
branchement inconditionnel : référence arrière	8
branchement conditionnel : référence arrière	9

Contrôle d'intégrité

Les branchements internes d'une section critique peuvent être traduits dans le schéma conditionnel. Cet état est traduit par la propriété

$\forall c \in \mathcal{C}$
$premier \leq cible \leq dernier \Longleftrightarrow premier \leq source \leq dernier.$

Les branchements avec sortie du bloc courant ou entrée depuis un autre bloc posent problème et sont à examiner cas par cas.

1.3.5 Appels de sous-programmes

Les unités de programmes Fortran sont appelées dans un arbre dont la racine est un programme principal. Les tâches sont habituellement réparties dans un ensemble de sous-programmes appelées avec l'instruction Fortran CALL dans un bloc de supervision, laquelle est contenue dans le sous-programme PRINCIP pour la version 5, dans le sous-programme principal pour la version 6.[31]

Le séquencement des appels dans ce bloc de supervision va permettre de gérer la dynamique du modèle en séparant l'initialisation, les évaluations quotidiennes et les éditions finales : il est édité dans le compte-rendu pour toutes les unités de programmes.

1.3.6 Entrées et sorties

Les instructions de lecture et d'édition vers les fichiers s'effectuent à travers des unités logiques et utilisent des formats, généralement déclarés dans des instructions identifiées par des étiquettes (3.8).

Unités logiques

L'unité logique est lue derrière la parenthèse ouvrante qui suit READ ou WRITE : c'est usuellement une variable numérique, un nombre, la marque $*$ pour l'entrée ou la sortie standard, ou une expression entière.[32]

[31] Aucune fonction n'est citée dans cette séquence dans le code de STICS.

[32] La version 6 gère les éditions pour les couples de cultures dans des tables de la forme FICRAP(IPL).

Les valeurs numériques des unités logiques désignent normalement des fichiers ouverts dans des instructions antérieures mais l'unité est quelquefois une variable textuelle traitée comme un enregistrement.[33]

Ouverture et fermeture des fichiers

L'ouverture et la fermeture des fichiers s'effectuent à travers les unités logiques : le fichier est accessible dans une séquence d'instructions après ouverture et tant qu'il n'est pas fermé.

Ces séquences sont reconstituées pour pouvoir ensuite ignorer les blocs de recopie dans des fichiers temporaires dans la phase de génération de code, et repérer les fichiers qui paraissent demeurer ouverts.[34]
Elles sont décrites dans le quadruplet

$$(module, ulogic, ouvert, clos)$$

dans lequel :

module est le nom de l'unité de programme ;
ulogic désigne l'unité logique, souvent numérique mais quelquefois contenue dans une variable ;
ouvert localise l'instruction d'ouverture du fichier ;
$clos > 0$ localise l'instruction de fermeture ;

Formats et liste des entrées et sorties

Les formats fixes sont identifiés dans l'unité de programme par un numéro d'étiquette derrière l'unité logique, qui fait référence à une description de l'organisation d'un enregistrement : format fixe. En l'absence d'un tel numéro, il s'agit d'un format libre.

Format fixe Les formats sont des chaînes textuelles qui décrivent la structure d'un enregistrement en positionnant les valeurs dans des champs.[35] Ils suivent l'instruction FORMAT repérée par l'étiquette ($eti \geq 10000$), présente quelque part dans l'unité de programme ou dans un fichier inclus.

Ces formats sont mis en relation avec les éléments de la liste des entrées et sorties. Cette liste contient des noms de variables, des constantes et expressions en édition, ainsi qu'une structure propre au langage Fortran, dite boucle implicite, qui reprend l'itération à pas constant.

[33] Les instructions de lecture du sous-programme LECPARAMV6 utilisent comme unité logique la variable locale textuelle VPARA, dans laquelle la valeur est extraite de la chaîne de caractères antérieurement lue à travers un format.

[34] Dans LECUSM, l'unité 11 apparaît non close car la fermeture s'effectue avec la variable locale FICUSM qui reçoit cette valeur.

[35] Certains formats sont relativement longs : 696 caractères pour le format 1075.

Format libre Le format libre, associé à une valeur non numérique, apparaît sous plusieurs formes.

La marque $*$ apparaît fréquemment dans des sous-programmes de lecture ou d'écriture qui lisent ou éditent les variables consécutivement : ces instructions consécutives sont alors dénombrées.
L'étiquette fictive 0 est attribuée à ce format.

Le texte est parfois fourni directement entre quotes : il est restituer en édition.
Une étiquette fictive ($n \geq 10000$) est générée.

Des variables textuelles sont quelquefois directement citées à la place du format : les valeurs sont éditées.
Une étiquette fictive ($n \geq 10000$) est générée.

Boucles implicites

Une boucle implicite apparaît entre parenthèses dans la liste des entrées et sorties sous la forme

(X(I),I = N1,N2)

pour énumérer les 5 éléments de la table X entre N1 et N2, bornes qui sont souvent des nombres mais parfois des variables ou des expressions à évaluer (3.8.2).

Un format est normalement défini pour chaque élément de la table mais ce nombre d'éléments est inconnu si les bornes ne sont pas des nombres, ce qui pose un problème pour établir la correspondance entre la liste des entrées-sorties et la suite des champs décrits dans le format. Ces boucles sont finalement repérées dans cette étape par recherche du symbole =, absent dans ce contexte pour les variables numériques.[36]

Description des formats

Un format, associé à une instruction de lecture ou d'édition, est représenté par un septuplet

$$(module, source, unit, eti, boucle, format)$$

dans lequel

- *module* contient le nom du module ;
- *source* est le numéro de l'instruction utilisant le format ;
- *unit* est la valeur textuelle qui désigne l'unité logique Fortran ;
- *eti* est l'étiquette du format, ou $n = 0$ pour le format libre, $n > 10000$ pour une valeur textuelle ;
- *boucle* dénombre les boucles implicites de la liste associée des entrées et sorties ;
- $format$ est la valeur textuelle qui décrit le champ à lire ou éditer, ou une variable textuelle, ou le nombre d'instructions consécutives en format libre, ou le message "`format inconnu`" ;

[36] Ce symbole peut être présent dans des textes placés entre quotes pour une édition.

1.3.7 Traduction informatique

La commande **S5** permet de déterminer s'il est possible de retrouver les structures de contrôle des langages algébriques. Développée autour des versions de STICS, elle peut être enrichie pour traiter d'autres formes syntaxiques.

Appelée avec l'argument `stics`, **S5** lit les fichiers

"`stics.upf`", lexique des modules ;
"`stics.fmt`", code normalisé des formats du fichier d'inclusion ;
"`stics.ins`", ensemble du code normalisé ;

et édite les relations produites dans les fichiers

"`stics.sis`", instructions conditionnelles structurées ;
"`stics.do`", boucles Fortran DO ;
"`stics.goto`", ensemble des branchements, explicites et autres ;
"`stics.cri`", sections critiques ;
"`stics.format`", formats de lecture et édition ;[37]
"`stics.unit`", unités logiques utilisées dans les unités de programmes en lecture ou écriture ;
"`stics.ouvert`" localise les séquences dans lesquelles les fichiers sont ouverts et les unités logiques ;

"`stics.cr5`" contient un compte-rendu utilisable dans l'étape suivante pour gérer la dynamique des variables : les unités de programmes appelées et les unités logiques utilisées sont citées en séquence pour chaque unité de programme.

Les erreurs, qui traduisent généralement des non reconnaissances de formes ou des débordements, se soldent par un déroutement qui clôt immédiatement l'exécution.

1.4 Bilan de l'analyse syntaxique globale

Cette étape préliminaire peut toujours être appliquée a des logiciels Fortran pour examiner les possibilités de reprise ou génération du code dans des projets informatiques, et en évaluer la faisabilité.

Reconnaissance des instructions
Le prétraitement fournit une première vue sur le sous-ensemble du langage Fortran effectivement utilisé.

Variables communes
Le mode de gestion des variables est ensuite essentiel. L'utilisation de communs uniques déclarés dans des fichiers d'inclusion permet de vectoriser ces variables, les quelques cas résiduels étant traités à part. Cette stratégie ne s'applique plus si ces communs sont déclarés dans les unités de programmes avec des listes différentes dans les unités de programmes. En revanche, l'absence de communs ne pose pas problème.

[37] Les formats longs sont décompsés en séquence d'au plus 32 caractères.

Structures de contrôle

C'est ensuite le mode de programmation qui est fondamental : il faut pouvoir retrouver les structures de contrôle usuelles. Le diagnostic de non faisabilité est sans appel, une utilisation abusive de branchements se traduisant par une explosion combinatoire de cas, donc de messages et de débordements de piles.

Entrées-sorties

L'acquisition des données et la restitution des résultats peuvent être détachés du logiciel. Une traduction peut toutefois s'avérer utile dans un premier temps dans la phase de validation logicielle, pour reprendre les jeux de données existants et restituer les résultats sous une forme quasi-équivalente.

Chapitre 2

Traitement des noms symboliques

Les instructions Fortran contiennent, en plus des mots réservés qui définissent le langage, des noms symboliques constitués de lettres, les formes minuscules et majuscules étant équivalentes, et de chiffres, le premier caractère étant une lettre. Ces noms désignent des variables ou des sous-programmes[1] et fonctions, la recherche étant faite dans cet ordre. Ils peuvent être suivis d'une liste parenthèsée, laquelle contient soit les indices d'une table, soit les paramètres d'une fonction.[2]

Variables Les variables Fortran sont locales ou communes, les premières masquant les secondes.

1. Les variables locales sont éphèmères et n'existent que dans l'unité de programmation où elles sont citées.
2. Les variables communes, citées dans les fichiers d'inclusion comme décrit en section 1.2 et extraites par la commande *S2*, sont partageables et pérennes.

Les variables Fortran peuvent ne pas être déclarées et une convention implicite est appliquée si elles apparaissent dans une instruction (1.2.1).[3]

Les variables déclarées ne sont définies que si elles reçoivent une valeur, indéterminées sinon : la section suivante examine les modes d'affectation possibles.

Fonctions Les fonctions sont intrinsèques ou externes, la recherche s'effectuant dans cet ordre.

[1] Un sous-programme est une fonction à résultat neutre.

[2] L'analyse syntaxique ne permet pas de lever l'ambiguïté, les indices ou paramètres étant séparés par des virgules et pouvant contenir des expressions arithmétiques.

[3] L'instruction Fortran IMPLICIT NONE rend obligatoire les déclarations des variables communes de la version 6.

1. Les fonctions intrinsèques, mathématiques ou autres, appartiennent aux bibliothèques du langage. Elles sont identifiées par des noms d'usage qui peuvent varier d'un compilateur à l'autre. Une liste est initialisée puis complétée avec les nouvelles fonctions trouvées dans le code analysé.
2. Les fonctions externes sont compilées comme les sous-programmes. Elles sont répertoriées dans le lexique des modules édité par la commande $S1$ du chapitre 1.

Toutes ces fonctions peuvent admettre des paramètres et modifier les variables ainsi transmises, et pour certaines rendre des valeurs.

Relations entre variables Les variables communes définissent un ensemble de variables informatiques banalisées identifiées par des noms propres. Les opérations informatiques d'évaluations, traduites dans des expressions numériques ou des appels de fonctions, définissent des relations sur cet ensemble, restituées dans un graphe ensuite utilisable pour classer les variables et produire des descriptions formelles des traitements effectués.

Classification des variables STICS est un modèle dynamique et il y a lieu de distinguer les paramètres du modèle, exogènes, invariants pendant la simulation, des variables endogènes évaluées. Les variables exogènes sont elles-mêmes regroupées par familles (constantes, plante, sol, techniques culturales, climat).

L'analyse de l'unité de supervision, et l'examen du mode d'acquisition des paramètres du modèle, vont permettre d'initialiser une classification des variables, définie dans le contexte de programmation de ce logiciel. Les classes des autres variables sont évaluées par cheminement dans le graphe des relations.

Cette classification est ensuite utilisable pour générer des composants en fonction d'objectifs applicatifs.

2.1 Description des variables

Il est convenu, en dehors du langage Fortran, de distinguer variables communes et locales à travers une convention d'écriture : les premières débutent par une lettre majuscule, comme les noms *Propres*, les secondes restent en minuscules, comme les noms *communs*. Les crochets [] sont par ailleurs ajoutés à droite pour chaque dimension d'une table. Les variables Fortran sont quant à elles notées en CAPITALES, minuscules et majuscules étant équivalentes.

Les variables *Stics* numériques de la version 5 sont simples, vectorielles ou matricielles. La version 6 introduit des dimensions nouvelles pour gérer deux cultures associées, avec

- deux plantes cultivées,
- une zone ensoleillée et une zone ombragée,

les deux dimensions pouvant être présentes simultanément.

Les variables utilisées reçoivent généralement des valeurs à l'issue d'opérations d'affectations dans lesquelles elles sont citées comme opérandes ou éléments de listes de lecture.

Dans le langage Fortran, elles sont accessibles dans les sous-programmes et fonctions appelés dès lors qu'elles sont transmises comme paramètres : SIN(X) peut modifier X.

2.1.1 Affectation d'une valeur dans une variable

Dans une opération d'affectation directe, l'expression qui figure à droite de l'opérateur = est évaluée et le résultat rendu dans l'opérande gauche. Cette opération peut s'effectuer :

1. dans une instruction simple, par exemple Y = F(X) ou A = 1 ;
2. comme instruction dans un IF LOGIQUE Fortran tel que décrit en 1.3.1 ;
3. lors de l'initialisation d'une itération à pas constant, soit la boucle DO Fortran du paragraphe 1.3.2 ;

Pour les tables, les valeurs des indices sont fournis entre parenthèses à droite des noms des variables.[4]

Les modes d'affectation utilisés dans le code Fortran de *Stics* sont répertoriés ci-après. La reconnaissance de l'un de ces modes ne signifie pas que la variable reçoit effectivement une valeur : les affectations peuvent être conditionnelles et les variables citées dans certains modes, notamment les paramètres des fonctions Fortran, ne sont pas forcément modifiées.

mode = 1 : évaluation d'une expression

Si l'opérande droite n'est pas une valeur (voir le mode suivant), c'est une expression, arithmétique, textuelle ou logique. La variable citée à gauche de l'opérateur d'affectation = va recevoir le résultat de l'évaluation de cette expression, soit de façon générique

$< variable >=< expression >$

Les deux opérandes sont normalement de même type mais l'opération reste définie pour deux types numériques différents, les conversions étant faites implicitement.[5]

Les noms cités dans une expression sont examinés. Si une expression parenthèsée suit un nom, il s'agit d'une table ou d'une fonction.

mode = 2 : affectation d'une constante

Les constantes citées à droite de l'opérateur d'affectation = sont
- des nombres pour les expressions arithmétiques,
- des chaînes de caractères entre quotes ou guillemets pour les expressions textuelles,

[4] Une fonction n'est pas pas admise comme opérande gauche d' affectation.

[5] Un nombre réel affecté dans une variable entière est réduit à sa partie entière.

– les valeurs `.true` ou `.false` pour les expressions logiques,
soit l'instruction générique

$< variable >=< nombre > | < texte > | < logique >$

Un nombre est transcrit sous la forme décimale, avec ou sans partie entière, comme `.65`, ou dans la notation dite avec exposant, comme `1e-2`.

Pseudo-constantes L'initialisation d'une variable commune avec l'instruction PARAMETER (1.2.4) lors de sa déclaration est assimilée à une affectation.

mode = 3 : initialisation après déclaration

Des variables sont initialisées par une instruction DATA suivie d'un nom et d'une liste de valeurs encadrée par des barres obliques (/), soit

DATA $< variable > / < liste\ de\ valeurs > /$ [6]

Les valeurs sont numériques, textuelles ou logiques.

mode = 4 : lecture dans un fichier

Une instruction de lecture a la structure générique suivante :

READ$(< controle >) < liste\ de\ variables >$

Les valeurs lues sont affectées dans les variables citées dans une liste d'éléments séparés par des virgules. Ces éléments sont :

- des noms de variables, éventuellement suivis d'une expression entière entre parenthèses pour gérer les indices des tables, la structure $< variable >$ étant décrite au chapitre suivant (3.4.1) ;
- une structure syntaxique de la forme suivante, placée entre parenthèses, dite **boucle implicite**,
  $(< variable >, < vboucle >=< n1 >, < n2 >)$[7]
  $< vboucle >$ étant la variable de boucle locale, qui relève du mode suivant, $< n1 >, < n2 >$ les bornes inférieure et supérieure, nombres, variables ou expressions.

mode = 5 : variable de boucle DO

Les variables de boucles, numériques, ne sont pas toujours déclarées dans le langage Fortran. Elles sont citées dans une instruction DO de la forme

DO $< etiquette > < variable >=< debut >, < fin > [, < pas >]$

avec un pas unitaire par défaut.

Les ordres de lecture et d'édition utilisent la boucle implicite précédente.

Les variables de boucles sont parfois communes, donc consultables et modifiables dans toutes les unités de programmes.

[6] La forme DATA $< liste\ de\ variables > / < liste\ de\ valeurs > /$, initialement non utilisée, n'est pas reconnue.

[7] La norme Fortran 77 permet de gérer plusieurs variables de boucles : la reconnaissance est ici faite pour une boucle simple.
La forme reconnue est décrite plus précisément en section 3.8.2.

2.1.2 Transmission d'une variable en paramètre

Une fonction peut appeler une autre fonction et transmettre en paramètres des noms qui désignent indifféremment les variables et leurs valeurs. Deux modes supplémentaires sont introduits pour gérer les échanges de paramètres des fonctions et sous-programmes Fortran.

mode = 6 : paramètre effectif

La variable est citée dans la liste des paramètres dits effectifs, seule et non dans une expression,[8] lors d'un appel d'un sous-programme et d'une fonction, soit

CALL $< sous\text{-}programme > (< liste >)$ ou
$< fonction > (< liste >)$

la seconde forme pouvant être présente dans une expression quelconque ou une liste, de paramètres ou d'édition.[9]

Une variable ainsi transmise en paramètre effectif est consultable, mais peut aussi recevoir en Fortran le résultat d'une évaluation faite dans la fonction appelée (2.2.4).

mode = 7 : paramètre formel

Des variables sont citées lors d'une déclaration de sous-programmes ou fonctions dans la liste des paramètres dits formels, soit

SUBROUTINE $< sous\text{-}programme > (< liste >)$ ou
FUNCTION $< fonction > (< liste >)$ ou
REAL FUNCTION $< fonction > (< liste >)$

Les paramètres formels sont des variables locales, lesquelles peuvent masquer localement les variables communes homonymes. Ils sont modifiables, mais il faut examiner les instructions pour déterminer s'ils sont effectivement modifiés.

Type des fonctions Les fonctions rendent généralement une valeur. Le resultat d'une fonction typée est rendu dans la variable locale homonyme de la fonction,[10] vue comme un paramètre formel. Si le type de la fonction n'est pas explicitement déclaré, il peut l'être ensuite par déclaration de la variable homonyme,

- explicitement avec les déclarations des variables locales,[11]
- implicitement dans une opération d'affectation ou un appel.

[8] La variable X est paramètre effectif dans l'appel F(X), ne l'est pas dans F(X+1).
[9] Les fonctions sont absentes des listes de lecture.
[10] La récursivité est inconnue dans le langage Fortran.
[11] C'est apparemment la solution adoptée pour la version 6.

2.1.3 Portée des variables et synonymies

Les variables citées dans une liste de communs sont rémanentes mais accessibles seulement dans les unités de programmes où sont cités ces communs, généralement par ajout des fichiers d'inclusion (1.2.3), quelquefois dans une déclaration locale du commun.

Toutes les autres variables n'existent que dans les unités où elles sont citées et sont locales : les valeurs sont perdues en fin d'exécution du sous-programme ou de la fonction appelés.

Homonymies

Les homonymies entre variables communes et locales sont signalées par un message : la variable commune est alors inaccessible car masquée par la variable locale.

Hétéronymies

Les communs introduisent la possibilité de désigner une même variable informatique par plusieurs noms dans différents modules, l'équivalence étant gérée à travers le rang dans la liste (1.2.3). Ce cas de figure est recherché et il apparaît que les variables communes *Stics* sont désignées par des noms uniques.

Cette correspondance selon les rangs dans les listes pose problème dans la version 5 dans les deux cas suivants, qui n'apparaissent plus dans la version 6.

1. **Communs homonymes des fichiers inclus**
   Un même nom de commun peut être cité dans plusieurs fichiers inclus avec des listes propres, concaténées si ces fichiers sont inclus dans l'unité de programme.[12] Si ces fichiers ne sont pas présents ou absents simultanément, plusieurs listes peuvent alors être associées à ce commun, ce qui génère des hétéronymies : un avertissement est édité et il faut alors vérifier que les variables communes citées dans l'unité n'appartiennent pas à la liste locale construite par concaténation.[13]
2. **Communs locaux**
   L'instruction COMMON est utilisée 7 fois en dehors des fichiers inclus dans la version 5.
   - EAUPL, TOM2 et DOMI sont des communs cités dans des unités de programmes dans lesquelles les fichiers de déclarations ne sont pas inclus ;[14]
   - UDEVFR gère le partage de la variable DEVJOURFR entre FRUIT et FPF ;
   - RAPPORTINC gère des équivalences entre variables locales dans LECRAP et RAPPORT ;

[12] Dans la version 5, les communs NB1003 et CLIMATB2 apparaissent dans les fichiers ``combym.inc'' et ``comtecplt.inc''.

[13] Sous-programme CALPENMAN : note ?? section 4.4.3.

[14] Note 156 section 4.5.8.

Comme précédemment, des hétéronymies peuvent ainsi être générées et les contrôles sont à effectuer variable par variable au vu des listes.[15]

Variables locales

Les listes des paramètres des sous-programmes et fonctions définissent les premières variables locales, les types n'étant pas déclarés en Fortran dans ces listes.

Les types des variables locales sont recherchés dans les déclarations au début de l'unité de programme sans qu'il soit possible de distinguer variables et fonctions : un nom qui suit un spécificateur de type est présumé désigner une variable locale seulement s'il n'appartient pas aux listes de fonctions externes et intrinsèques.

Pour les autres déclarations, les listes associées aux communs locaux (version 5) sont ignorés; celles qui suivent l'ordre DIMENSION sont traitées : les variables inconnues sont déclarées implicitement.[16]

Les instructions sont ensuite examinées et toute variable citée est recherchée d'abord dans la liste des variables locales, ensuite dans celle des variables communes visibles, déclarée implicitement et ajoutée si elle ne figure pas dans ces listes.[17]

Les variables de boucle communes sont remplacées par des variables locales.[18]

Si une variable locale est homonyme d'une variable commune, le type implicite est remplacé par le type de la variable commune.

Pour toute variable locale susceptible de recevoir une valeur, la modalité d'affectation est notée.

Variables textuelles Des variables textuelles locales sont utilisées, notamment dans les unités d'édition. Elles sont reconnues et enregistrées dans le dictionnaire local avec le nombre de caractères comme première dimension, le nombre de valeurs comme seconde dimension si le cas se présente.

2.1.4 Vecteurs et matrices

Les cultures associées de la version 6 peuvent être vues comme des couples de valeurs : les variables *Stics* gèrent alors, comme pour les versions antérieures, des nombres, vecteurs ou matrices.

[15] Les échanges de variables à travers ces communs sont à proscrire, le mode usuel étant le passage de paramètres.

[16] Une erreur comme REALQH2OF au lieu de REAL QH2OF fait que la déclaration n'est pas reconnue : QH2OF sera néanmoins déclarée dès lors qu'elle est citée dans une instruction (4.3.5).

[17] Une fonction citée dans une expression arithmétique et jusqu'alors non répertoriée parmi les fonctions intrinsèques est confondue avec la table homonyme, et déclarée implicitement.

[18] Les occurrences des cultures associées sont gérées avec les deux variables communes de boucles IPL et ENS.

Dimension des espaces vectoriels

Les vecteurs et matrices sont gérés dans des tables Fortran dont la taille est fournie en clair, le nombre figurant entre parenthèses derrière le nom de la variable dans une instruction de déclaration.[19]

Pour modifier la dimension d'un espace applicatif de référence, il faut en conséquence :

- modifier en clair toutes les déclarations dans toutes les unités qui effectuent des évaluations sur cet espace ;
- rechercher les valeurs citées en clair dans le code comme bornes supérieures de boucles ou opérandes d'expressions logiques ;

Les dimensions des espaces sont répertoriés par analyse des déclarations : des noms symboliques, notés en petites capitales, vont désigner ces constantes, avec les valeurs à droite qui mesurent les nombres d'unités, choisies en fonction des contextes applicatifs. Les constantes pour la version 6 sont les suivantes.

TEMPS	731	jours calendaires d'une campagne bisannuelle
BASEPROFIL	1000	profondeur du sol
NHORIZONS	5	nombre d'horizons du sol
PRESIDUS	60	profondeur de la zone humifère
NRESIDUS	8	types de résidus
NCAILLOUX	10	types de cailloux
NRECOLTE	20	nombre de récoltes, dont coupes
NBOITE	50	nombre de boîtes pour simuler la production de fruits
NACTION1	300	nombre d'opérations techniques quotidiennes fréquentes
NACTION2	10	nombre d'opérations techniques quotidiennes peu fréquentes

Dans la version 5, la profondeur du sol est réduite à 200 et la durée de la campagne à 366.

Ces constantes sont reprises dans la génération du code C pour déclarer les tables et allouer l'espace (3.1.2).

Gestion des indices

Les éléments des tables sont désignés au moyen d'indices gérés dans des variables simples ou tabulaires, normalement entières.[20] Ces variables d'indices sont parfois des variables communes : une copie locale est alors créée et un mécanisme de gestion mis en place pour pouvoir restituer la valeur finale de l'indice.

[19] Cette dimension est déclarée avec le type, ou dans le commun, ou explicitement avec l'instruction DIMENSION.

[20] Des variables locales sont à introduire pour éviter les conversions, explicites ou implicites, ambigües (partie entière ou arrondi).

2.1.5 Variables communes non numériques

L'utilisation de variables communes non numériques dans les évaluations rend complexe l'échange de variables avec d'autres fonctions ou logiciels externes.[21] Afin de supprimer ce problème pour les applications générées, les variables logiques et les variables textuelles citées dans les expressions logiques et affectations sont numérisées.

Variables logiques

Les variables logiques Fortran sont utilisées pour coder des changements d'état. Elles rendent les valeurs *vrai* ou *faux* lorqu'elles sont citées dans des expressions logiques et sont initialisées dans des affectations : la variable figure en opérande gauche avec comme opérande droite

- les valeurs ".TRUE." ou ".FALSE.",
- une variable logique (version 6),
- une expression logique, non traduite.

Ces variables logiques sont remplacées par des variables entières qui prennent les valeurs 1 ou 0 : les opérations unaires deviennent binaires.[22]

Variables textuelles

Les expressions logiques textuelles sont définies dans des opérations logiques binaires avec une variable textuelle comme opérande gauche et en opérande droite

- la valeur textuelle placée entre quotes ou guillemets,[23]
- une variable textuelle (version 6),
- une expression textuelle, non traduite.[24]

Les variables textuelles sont initialisées par affectation d'une chaîne de caractères ou lecture dans un fichier.[25]

Les fonctions textuelles de traitement de chaînes utilisées en édition ne sont pas reprises.

Images numériques Des variables textuelles sont quelquefois utilisées dans des expressions logiques avec une valeur textuelle citée en clair dans le code :[26] une variable image, numérique entière, est alors créée par suffixage de l'identificateur avec un tiret souligné '_'.

[21] Il faut alors pouvoir transmettre des tables de codes textuels en paramètres.

[22] Soit la variable logique Fortran B : la condition logique (B) est traduite en C *(b == 1)*. Pour une culture simple, la condition redondante Fortran (GELEE(IP).EQV..TRUE) devient en C *((gelee == 1) == 1)*.

[23] la marque utilisée est doublée si elle apparaît dans la dite valeur (apostrophe).

[24] Il s'agit d'opérations de concaténations pour l'accès aux fichiers.

[25] Ces variables gèrent une quarantaine de codes textuels dans la version 5. La version 6 introduit des citations de variables STICS avec les indices des cultures associées, soit un ensemble final d'une centaine de valeurs textuelles, comme "`ratioresfauche(2)`".

[26] Le nombre de codes textuels passe de 38 à 173 entre les versions 5 et 6, avec en sus les indices '(1)' ou '(2)' à droite des noms.

La numérisation s'effectue en répertoriant ces valeurs placées entre quotes ou guillemets,[27] dans les expressions logiques Fortran : la valeur numérique de la variable image est le rang dans la liste. L'expression textuelle peut ainsi être remplacée dans la phase de génération de code par une expression entière équivalente (3.5.3).[28]

2.1.6 Constantes décimales

Les constantes numériques citées dans une expression sont réelles si elles contiennent un point décimal ou un exposant (1e-2), entière sinon. En langage C, elles sont par défaut représentées en double précision, alors qu'elles demeurent en simple précision avec la compilateur Fortran utilisé. Cette situation génère deux difficultés.

1. Les types de données diffèrent et il en résulte des écarts qui vont dépendre des compilateurs et qui peuvent se propager au cours des périodes successives (4.1.2).
2. Les prototypes des fonctions (2.2.5) peuvent varier, un paramètre effectif de la forme 1.0 étant par défaut en double précision.

Afin de pouvoir produire un code C qui effectue les mêmes évaluations, des variables locales déclarées en simple précision et initialisées avec ces valeurs décimales sont ajoutées si des constantes réelles sont utilisées dans une expression arithmétique, ou comme paramètres effectifs dans des appels.

Les nombres qui figurent en opérandes droites des affectations sont conservés.

Gestion des constantes citées dans des variables locale Des variables locales sont déclarées pour gérer les constantes décimales, et initialisées. Les valeurs initiales sont conservées dans le dictionnaire pour être reprises lors de la génération du code : les formes 'x.' et '.d' sont remplacées par 'x.0' et '0.d'.

Les variables $zero_$, $un_$, $deux_$, $dix_$, $cent_$, $mil_$, gèrent les nombres réels correspondants, le préfixe $k_$ suivi d'un rang étant utilisé pour les autres nombres, dont les notations E.

2.2 Description des fonctions externes

Le partage de variables entre modules informatiques séparés s'effectue classiquement par échange de paramètres entre fonctions. Les fonctions fournies avec le compilateur dans des bibliothèques sont dites intrinsèques. Les autres, développées, sont usuellement déclarées dans le module appelant à travers une description formelle dite **prototype**. Dans le langage Fortran, cette description n'est pas effectuée directement et doit être générée dans le contexte de l'unité de

[27] Les espaces et tabulations, parfois présents pour gérer des champs de longueur fixes en Fortran, sont supprimés : la valeur textuelle est donc un mot dans la traduction C.

[28] Les mises à jour de variables textuelles en cours de simulation, comme dans le sous-programme d'inversion de dominance (3.2.1) qui permute les noms des fichiers des paramètres de la plante (FPLT) et des indices foliaires (FLAI), ne sont pas numérisées.

programme, pour les fonctions déclarées comme telles et les sous-programmes avec paramètres.

Prototypage Les paramètres sont cités entre parenthèses dans une liste ordonnée et identifiés par leurs rangs. Un paramètre est défini sur un domaine auquel correspond un type de valeur. La liste de paramètres peut être vide, d'où la notation $f()$ pour désigner une fonction.

Une fonction rend généralement une valeur d'un type donné. Une fonction qui ne rend rien est de type neutre, usuellement noté *void*, comme la fonction sans paramètre $void\ imprimer()$.

Les types d'une fonction et de ses paramètres, dits formels, sont normalement déclarés : ils définissent le prototype de cette fonction. Lors des appels, les paramètres transmis, dits effectifs, doivent être du même type, tout comme le type de la valeur rendue.

La définition du prototype permet aussi de déclarer si les paramètres sont des valeurs, donc non modifiables, ou des références à des variables qui peuvent alors être mises à jour.

2.2.1 Fonctions intrinsèques Fortran

La liste des fonctions intrinsèques Fortran, fournies dans les bibliothèques ou créées,[29] est établie dans un fichier annexe, "`stics.f`", avec en regard le nom de la fonction C correspondante, appartenant à la bibliothèque du langage C ou développée sur mesure, suivi du type de la fonction, soit par exemple "NINT Arrondi int".

La marque '*' signifie que les règles usuelles de conversions implicites de types relatives à l'évaluation des expressions numériques sont utilisables (3.3.6).[30]

Pour les fonctions développées, il faudra déclarer explicitement le prototype.

2.2.2 Déclaration des paramètres et fonctions

Les sous-programmes et fonctions Fortran, dits unités de programmes, sont déclarés sous les formes suivantes :

SUBROUTINE $< nom >$ ($<$ *liste de parametres* $>$) ou
FUNCTION $< nom >$ ($<$ *liste de parametres* $>$) ou
$<$ *type* $>$ FUNCTION $< nom >$ ($<$ *liste de parametres* $>$)

Le nom de l'unité de programme ne peut pas être celui d'une fonction intrinsèque.

La liste entre parenthèses énumère les noms de paramètres formels séparés par des virgules : ils définissent des variables locales.

Les paramètres formels sont souvent dans le code Fortran de *Stics* homonymes de variables communes visibles, lesquelles sont alors masquées.

[29] La fonction C *Minval*() remplace la fonction *Stics* Fortran MINVAL.
[30] Les fonctions Fortran INT,IFIX,AINT,FLOAT,REAL appliquent ces conventions.

Déclaration des paramètres formels

Les paramètres sont des variables locales simples ou des tables de types donnés, déclarées explicitement avec les autres variables locales, implicitement sinon.

Si ces paramètres masquent des variables communes, des contrôles de concordance des types, et des dimensions et bornes pour les tables, sont néanmoins effectués, ceci pour pouvoir ensuite valider la suppression de paramètres inutiles (2.2.5).

Type Les types des paramètres formels sont éventuellement déclarés explicitement avec les variables locales dans les premières instructions, implicitement sinon.

Les déclarations Fortran

```
SUBROUTINE RATMO(RG,AANGST,BANGST,LATITUDE,JUL,PM,TM,RATM)
INTEGER JUL
REAL RG,LATITUDE,PM,TM,RATM,NUAGE,FRACINSOL
```

définissent huit variables locales simples, l'une entière, les autres réelles, AANGST et BANGST, non déclarées, étant implicitement réelles.

Les variables communes simples *Angst*, *Bangst*, *Latitude*, *Ratm*, déclarées à l'issue des inclusions faites au début du module source contenant cette fonction, sont masquées.

Dimension Les paramètres sont des variables simples si les dimensions ne sont pas explicitement déclarées avec le type ou avec l'instruction DIMENSION en l'absence de communs.

Les déclarations Fortran

```
SUBROUTINE AJUPRO(NH,EPC,QN,N,Q,QEXC)
DIMENSION EPC(5),QN(5)
DIMENSION Q(1000)
```

définissent implicitement six variables locales, les types des paramètres formels n'étant pas déclarés : les trois variables *Epc*[], *qn*[] et *q*[] sont des tables.

Les déclarations Fortran

```
FUNCTION CALMACRO(CODEFENTE,DA,HCC,HMIN,EPC)
INTEGER CODEFENTE
REAL DA,HCC,HMIN,EPC
```

définissent explicitement cinq variables locales simples, dont quatre masquent les variables communes *Da*[], *Hcc*[], *Hmin*[] et *Epc*[], tables à une dimension.

Type de la fonction

Un sous-programme Fortran, déclaré par le mot-clé SUBROUTINE, est une fonction de type neutre.

Pour une fonction Fortran introduite par le mot-clé FUNCTION, une valeur est rendue si une affectation est faite dans la variable locale homonyme de la fonction. Le type est alors déclaré

- explicitement avec la fonction,
- ou explicitement avec la variable locale homonyme de la fonction,
- ou implicitement si cette variable homonyme est mise à jour dans une opération d'affectation.

En l'absence de déclaration explicite ou implicite, la fonction est de type neutre.

2.2.3 Appels des fonctions

Les fonctions sont appelées avec une liste optionnelle de paramètres effectifs énumérés entre parenthèses et séparés par des virgules. La syntaxe est la même que celle utilisée pour référencer les variables : $u(i)$ désigne l'élément de rang i de la table u ou l'appel de la fonction u avec le paramètre i. L'ambiguïté ne peut être levée qu'après consultation des dictionnaires des symboles.

Les sous-programmmes Fortran sont introduits par le mot-cle CALL ;

Les fonctions Fortran peuvent être citées dans :

- les expressions arithmétiques des opérations d'affectations ;
- les expressions logiques des instructions conditionnelles ;
- les listes de paramètres, directement ou dans des expressions ;[31]
- les indices de tables, directement ou dans des expressions ;
- les listes d'édition ;

Arbre des appels Les liens entre fonctions appelantes et appelées définissent un graphe sans circuit, les références croisées et la récursivité étant inconnues dans le langage Fortran.

Pour chaque unité de programme, un arbre des appels est construit avec comme sommets les noms de fonctions appelées. L'arbre ayant pour racine le sous-programme PRINCIP contient l'ensemble des liens, avec au plus 8 générations.[32]

L'ensemble de ces arbres est édité par parcours du graphe dans un ordre canonique pour pouvoir suivre l'enchaînement des appels. Il est restitué sous une forme textuelle pour visualiser les enchaînements des appels et sous une forme numérique pour gérer les déclarations des prototypes dans l'étape de génération du code.

Liste de paramètres

Un paramètre effectif dans un appel est une valeur, une variable contenant une valeur ou une expression à évaluer : la valeur est gérée en cours d'exécution dans la variable locale déclarée comme paramètre formel de même rang. Si cette

[31] Dans l'instruction TROSEE = TEMPROSEE(HUMIDITE * TVAR(TCULT)), TEMPROSEE et TCULT sont des fonctions.

[32] Ce nombre est lié pour l'essentiel aux fonctions utilisés pour le calcul du rayonnement net quotidien : CALRNET,RATMO,RGEX,TVAR,DECANGLE.

dernière est mise à jour, la variable transmise comme paramètre effectif est modifiée.

Pour tous les appels, il est vérifié que les types et dimensions des paramètres effectifs sont les mêmes que ceux des paramètres formels, et les listes de même longueurs. Si les types sont différents, des conversions implicites peuvent être effectuées, type partie entière d'un nombre réel.

2.2.4 Définition des paramètres par valeur ou référence

La valeur contenue dans la variable locale homonyme x d'un paramètre formel Fortran est restituée au programme appelant si le paramètre effectif correspondant est une variable : toute affectation dans le module appelé de x modifie alors la valeur de la variable y transmise par l'appelant. Pratiquement, cela signifie que l'appel $f(y)$ modifie y.

Les langages informatiques actuels offrent la possibilité de contrôler le passage des paramètres, par valeur ou par référence.[33]

Valeur Un paramètre reçu dans une fonction et non modifiable en cours d'exécution est à définir par valeur : pour une fonction numérique, ce paramètre peut alors être un nombre ou une expression à évaluer, une variable étant vue comme une expression réduite à un seul terme.

Référence La possibilité de recevoir une valeur est introduite dans la déclaration du paramètre formel par une marque explicite, généralement l'opérateur unaire d'indirection $*$. Le paramètre est non pas une valeur, mais une variable recevant cette valeur.

Paramètres des fonctions Fortran

Le langage Fortran ne distinguant pas valeur et référence, un moyen de reconnaissance est de rechercher les modalites d'affectations des variables locales qui gèrent les paramètres formels (mode 7 de la section 2.1.1) :

- les affectations directes (modes 1 et 2) sont utilisées pour rendre plusieurs valeurs ;
- les autres modes d'affectation, par lecture, initialisation DATA ou variable de boucle, sont à proscrire ;
- les mises à jour par transmission de paramètres depuis les fonctions appelées ;[34]

[33] Le paramètre formel N est utilisé dans les déclarations des sous-programmes AJUPRO et HUMHEURE. C'est aussi une variable de boucle, commune, du sous-programme PRINCIP d'enchaînement des simulations quotidiennes : sa transmission par valeur garantirait qu'une erreur dans la fonction appelée ne modifie pas la boucle principale.

[34] Le paramètre formel FORME du sous-progamme RTRANS est ainsi transmis comme paramètre effectif au sous-programme KGEOM qui le met à jour (versions 5 et 6).

Parcours de l'arbre des appels La transmission d'un paramètre formel comme paramètre effectif d'une fonction appelée (mode 6) peut se traduire par une mise à jour dans la descendance : il faut en conséquence parcourir l'arbre des appels pour examiner si les paramètres reçus sont modifiables.[35]

Effets de bords Les transmissions par référence de variables communes sont signalées par des messages afin de prévenir d'éventuels effets de bords.

Appels de fonctions dans des expressions transmises

Un autre problème se pose si le paramètre effectif est une expression contenant un appel de fonction : il est alors vérifié que les paramètres de cette fonction appelée ne sont pas transmis par référence.

2.2.5 Prototypes de fonctions

Des formes génériques sont adoptées pour regrouper les déclarations précédentes. Le sous-programme AJUPRO est représenté sous la forme générique

void *Ajupro*(int *Nh*,float *Epc*[],float *qn*[],int *N*,float **q*[],float **qexc*)

- Le type neutre indique que la fonction ne rend pas directement une valeur.
- Les crochets permettent de localiser des tables, ici à une dimension.
- La marque d'indirection $*$ signale que les paramètres $q[]$[36] et *qexc* sont définis par référence.
- Les noms de variables communes sont comme précédemment transcrits avec la première lettre majuscule, les variables locales en minuscules.

L'utilisation des variables communes comme paramètres, effectifs ou formels, est source d'ambiguïtés : l'ultime recours est parfois de vérifier ce que fait le compilateur Fortran.

Concordance des paramètres formels et effectifs

Il est vérifié que les paramètres formel et effectif de même rang sont du même type et ont le même nombre de dimensions.

Les problèmes posés sont illustrés avec la fonction Fortran BATTANCE (version 6), déclarée comme suit,

```
FUNCTION BATTANCE(N,NPLT,NREC,CODEPERENNE,PLUIESEMIS,TRR,
PLUIEBAT,MULCHBAT,XMLCH1,ELONG,PROFSEM)
INTEGER NPLT,N,CODEPERENNE,NREC
REAL BATTANCE,PLUIESEMIS,TRR,PLUIEBAT,ELONG,PROFSEM,MULCHBAT,XMLCH1
```

et appelée depuis LEVEE dans l'affectation

[35] Le nombre de branches à explorer est fini par construction pour le langage Fortran, et limité dans la version 5 à trois générations avec les fonctions *Decangle()* et *Tetacrit()* comme feuilles terminales.

[36] Cette notation signifie ici que q est un vecteur de réels, et non une table de pointeurs sur des vecteurs comme en langage C.

LEVBAT=BATTANCE(N,NPLT(IPL),NREC,CODEPERENNE(IPL),
PLUIESEMIS,TRR(N),PLUIEBAT,MULCHBAT,XMLCH1,ELONG,PROFSEM(IPL))

Des identificateurs désignent simultanément

- des variables communes, dont la série bisannuelle $Trr[]$ et les couples de cultures associées $Nplt[]$, $Nrec[]$, $Codeperenne[]$, $Profsemis[]$;
- des paramètres formels ensuite déclarés comme variables simples ;
- des paramètres effectifs sous la forme d'éléments de tables, sauf pour $Nrec[]$;

Un effet de bord possible resulterait d'une mise à jour de TRR dans BATTANCE, qui modifierait alors la composante de rang N de la série TRR dans LEVEE.

Pour les variables définies pour les couples de culture, l'élément modifié est celui désigné par IPL.

Cultures associées Les variables communes qui gèrent des couples sont souvent déclarées comme variables simples dans la liste des paramètres formels et citées comme telles dans le corps de la fonction, référencées comme éléments de tables dans les listes de paramètres effectifs.

Ces homonymies rendent le code peu lisible :[37] le code généré dans l'étape suivante doit être repris à vue, d'où l'intérêt de traiter le problème dans le code Fortran.

Paramètres homonymes de variables communes

Pour plusieurs fonctions ou sous-programmes Fortran, des variables communes sont citées dans la liste des paramètres formels. Tant que les fichiers des déclarations ne sont pas inclus, les dites variables ne sont pas visibles et des variables locales homonymes sont créées, avec la dimension du paramètre formel déclaré.[38]

Si la variable commune est accessible, un avertissement signale qu'une variable locale homonyme la masque : pour lever toute ambiguïté, et éviter qu'une inclusion ne modifie la portée des variables, la solution est d'utiliser une variable locale hétéronyme.

Variable commune transmise comme paramètre effectif

Le sous-programme CALRAERO est déclaré dans les versions 5 et 6 avec le paramètre formel RAS homonyme d'une variable commune. Ce paramètre est de fait défini par référence puisqu'il reçoit une valeur.

CALRAERO est appelé depuis SHUTWALL avec le paramètre effectif homonyme RAS :

[37] Sur l'exemple cité, LEVEE transmet NPLT(IPL) à la fonction BATTANCE, laquelle la récupère dans la variable locale simple NPLT : une mise à jour locale affecterait finalement l'élément de rang IPL de la variable commune NPLT.

[38] Ces homonymies rendent complexes les recherches par contexte : LAI est localement une variable simple et non la table commune.

– la variable commune RAS est mise à jour si le fichier contenant sa déclaration est inclus ;
– sinon, c'est la variable locale RAS qui est utilisée ;

Les deux autres appels utilisent les variables locales RASMAX et RASMIN comme paramètres effectifs.

Réduction des listes de paramètres

Les variables communes apparaissent souvent aux mêmes rangs dans la liste des paramètres effectifs.[39]

Les longues listes de paramètres sont difficiles à gérer et la complexité des prototypes rend les fonctions difficilement utilisables : il faut en sus, contrôler l'équivalence des déclarations des variables comme paramètres, formels chez l'appelé, effectifs chez l'appelant.

Pour réduire la longueur de ces listes, dès lors que les homonymies suggèrent des équivalences entre paramètres toujours cités aux mêmes rangs et variables communes, une solution est de s'appuyer sur le mode de partage adopté pour la plupart des unités de programmes, et d'utiliser finalement ces variables communes, en intégrant le groupe si le fichier associé n'est pas inclus.

Cette information est prise en compte dans la description des prototypes de telle sorte que les paramètres formels et effectifs qui désignent toujours les mêmes variables communes puissent être supprimés dans l'étape suivante de génération de code : il faut cependant qu'il y ait concordance entre paramètres formels et effectifs, pour les types et dimensions.

2.3 Relations entre variables

Les deux sections précédentes ont permis de localiser les variables communes partagées modifiables qui peuvent recevoir une valeur, par affectation ou passage de paramètre par référence. Ces relations sont traduites dans un graphe, les variables étant les sommets et les arcs les relations : $x \rightarrow y$, x *produit* y, signifie que x est utilisé pour évaluer y.

L'examen des affectations et des listes de paramètres permet aussi d'isoler des sous-ensembles de variables locales utilisées dans des calculs intermédiaires, représentées dans un graphe local : les chemins qui relient deux variables communes sont ensuite remplacés par un arc dans le graphe des relations entre variables communes.

Le graphe final regroupe dans la dernière version 7255 arcs : il est ensuite utilisé pour structurer l'ensemble des variables et repérer les variables modifiées.

[39] C'est le cas pour 17 paramètres parmi les 23 du sous-programme NAPPE, 10 parmi 19 pour DYNAMICTALLE.

2.3.1 Graphes des relations entre variables

Dans la version Fortran de *Stics*, les évaluations sont normalement effectuées dans des opérations d'affectations ou par échange de paramètres.[40] L'analyse du code permet de construire des graphes pour l'ensemble des variables communes et locales citées dans les fonctions :

- les sommets terminaux, origines d'aucun arc mais admettant des successeurs, peuvent être vus comme le tronc d'un arbre ;
- les variables lues ou initialisées avec une valeur constante, et qui demeurent invariantes, extrémités d'aucun arc mais admettant des prédécesseurs, sont alors les feuilles de ces arbres ;
- les variables non citées dans les expressions arithmétiques, mais utilisées dans les expressions logiques ou comme indices de tables, lues ou directement initialisées avec une valeur numérique, définissent des sommets isolés.

Les variables locales sont utilisées dans les fonctions comme variables intermédiaires, éphèmères, pour évaluer les variables *Stics* communes, partagées et pérennes : les chemins qui relient les variables communes en passant par des variables locales sont remplacés par un arc unique.[41]

Variables communes modifiables L'ensemble des graphes des fonctions va permettre de construire des listes de variables communes modifiables associées à l'exécution de ces fonctions, dont des variables non citées localement dans des expressions, mais néanmoins mises à jour en fonction de valeurs évaluées localement à travers le processus de gestion des paramètres.

2.3.2 Opération d'affectation

La forme $y = u + f(x)$ indique classiquement que la variable y reçoit une valeur qui dépend des valeurs du terme u d'une expression arithmétique et du paramètre x de la fonction f, déclarée localement ou intrinsèque, le symbole $=$ étant ici l'opérateur d'affectation : elle génère deux arcs, $u \rightarrow y$ et $x \rightarrow y$.

L'opération d'affectation est reconnue par une grammaire formelle avec des expressions arithmétiques complexes dans lesquelles sont citées :

1. des variables simples ou des tables indicées, les variables citées en indices ne générant pas d'arcs ;
2. des appels de fonctions qui admettent comme paramètres d'autres expressions contenant éventuellement des appels de fonctions ;
3. des constantes numériques avec des nombres décimaux,ou les formes $1e6$ et $1.e6$ qui ne génèrent pas d'arcs ;
4. des expressions arithmétiques entre parenthèses ;

[40] Le langage Fortran permet aussi d'échanger des valeurs par renommage des variables dans des communs (1.2.3) : DEVJOURFR, mise à jour dans FRUIT, est partagée au moyen du commun UDEVJOUR avec la fonction FPF dans la version 5,

[41] Les sommets visités sont à marquer pour prévenir les boucles.

La syntaxe du langage Fortran ne distinguant pas l'indiçage des tables et le passage des paramètres, un identificateur qui n'appartient pas aux listes de fonctions, déclarées ou intrinsèques répertoriées, désigne une variable, non nécessairement déclarée dans ce langage.

Ces opérations d'affectations génèrent des milliers d'arcs dans la forêt des graphes des unités de programmes.

Sommets non définis Les variables indéterminées, car non initialisées (2.1.1), peuvent néanmoins être référencées : elles apparaissent alors commme des sommets non définis.

Sommets locaux

Les variables locales qui n'apparaissent que comme sommets sur un chemin reliant des variables communes peuvent être supprimées, ce chemin étant alors remplacé par un arc entre ces variables communes. L'application de cette règle pose cependant problème dès qu'une variable locale instrumentale est utilisée dans plusieurs évaluations dans la mesure ou elle produit des liens entre toutes les variables communes citées dans ces évaluations : cela peut se traduire pratiquement par une explosion combinatoire.

Inversion de dominance La version 6 introduit un mécanisme d'inversion de dominance entre cultures principale et associée qui se traduit par des transferts de valeurs pour plusieurs centaines de variables avec recours à des variables locales intermédiaires, numériques réelles ou entières, mais aussi textuelles et logiques. Les milliers d'affectations associées génèrent une multitude de chemins qui passent par ces sommets locaux et relient finalement toutes les variables communes : l'algorithme est alors inopérant puisqu'il faut alors explorer l'ascendance de toutes ces variables communes. La solution pratique adoptée est finalement d'exclure le sous-programme concerné, en l'occurrence DOMINVER pour le code analysé.

2.3.3 Variables transmises en paramètres

La fonction appelante transmet des paramètres vers la fonction appelée.

Côté appelant, les paramètres effectifs peuvent être des variables ou des expressions contenant des variables, voire des appels de fonctions avec leurs propres paramètres. Toutes les variables communes citées sont origines d'arcs vers d'autres variables évaluées par l'appelé. Pour les variables locales sont énumérés tous les chemins en provenance de variables communes, lesquelles sont de la même façon origines d'arcs. Pour tout paramètre de rang k est ainsi défini un ensemble $\mathcal{E}_k$ de sommets origines, éventuellement vide.

Côté appelé, les valeurs des paramètres sont gérées dans les variables locales déclarées comme paramètres formels. Tous les chemins vers des variables communes sont énumérés. Un ensemble de sommets d'arrivée $\mathcal{F}_k$ est construit pour le paramètre de rang k.

A l'issue de chaque appel, les arcs sont ajoutés pour chaque élément des produits cartésiens $\mathcal{E}_k \times \mathcal{F}_k$ dans le graphe partiel de l'appelant : les 168 appels de fonctions avec paramètres génèrent 121 arcs supplémentaires.

2.3.4 Modifications des paramètres transmis

Un autre moyen est modifier la valeur d'une variable x est de transmettre vers la fonction appelée la référence à une variable, souvent notée $\&x$ dans les langages informatiques : l'appelé va retrouver la valeur de x avec l'opérateur inverse dit d'indirection, noté $*x$.

Ces opérations unaires ne sont pas définies en Fortran et le langage confond valeur et référence : toute variable directement transmise en paramètre effectif est alors modifiée au retour si le paramètre formel correspondant l'est chez l'appelé.[42]

Si le paramètre formel de rang k est modifié, il faut alors rechercher chez l'appelé tous les chemins en provenance de variables communes, qui définissent l'ensemble $\mathcal{F}'_k$ de sommets origines d'arcs. Côté appelant, on recherche tous les chemins vers des variables communes en provenance du paramètre effectif de rang k, lesquels définissent l'ensemble $\mathcal{E}'_k$ des extrémités, réduit à un sommet si le paramètre effectif est une variable communee.

A l'issue de chaque appel, les arcs sont ajoutés pour chaque élément des produits cartésiens $\mathcal{F}'_k \times \mathcal{E}'_k$ dans le graphe partiel de l'appelant : 50 arcs supplémentaires sont ainsi ajoutés.

2.3.5 Effet de bord

Des mises à jour de variables communes sont effectuées dans des sous-programmes ou fonctions à partir de valeurs évaluées localement mais sans que les dites variables soient citées.

Paramètres et variables communes

Les appels de sous-programmes et fonctions génèrent les arcs de la section 2.3.3 par fermeture transitive sur des chemins et transmission de paramètres non modifiables : les modifications, non visibles dans le corps de la fonction appelante, portent sur des variables communes.[43] Le compte-rendu global édite la liste des variables communes susceptibles d'être ainsi mises à jour localement alors qu'elles ne sont pas citées.

[42] Les fonctions intrinsèques Fortran ne modifient pas les paramètres avec lesquels elles sont appelées.

[43] La variable CRES apparaît ainsi comme variable modifiée dans les sous-programmes EFFEUILL et SENESCEN du fait de l'appel de APORES, qui met à jour CRES en fonction d'une quantité reçue en paramètre. Quant à la variable CBIO, également mise à jour dans APORES, elle n'apparaît pas comme modifiable car indépendante des paramètres reçus.

Fonctions citées dans des expressions arithmétiques ou logiques Les appels de fonctions apparaissent dans les mêmes contextes que les références aux tables et sous les mêmes formes syntaxiques : une variable transmise en paramètre peut ainsi être modifiée. Il est vérifié qu'il n'existe pas de tels arcs, donc que les fonctions *Stics* appelées ne modifient pas les variables communes.[44]

Modifications masquées

Les mises à jour qui ne dépendent pas de variables communes n'apparaissent pas dans les graphes précédents. Outre les cas marginaux de définition de communs (note 40), la situation se présente pour des mises à jour de variables communes qui ne font pas références à d'autres variables mais seulement à des nombres explicites, en clair dans le code : il suffit alors d'un appel pour modifier une variable commune non citée chez l'appelant.

Ces nombres peuvent apparaître localement, directement dans les expressions ou indirectement par affectation dans des variables locales, ou encore être reçus en paramètres.[45]

2.4 Classification des variables communes

Les variables communes *Stics* sont identifiées par leurs noms propres et pratiquement accessibles dans tous les modules, en lecture comme en écriture. Afin de ne pas devoir redéclarer nommément ces deux milliers de variables, une vectorisation est effectuée à travers un système d'allocation dans des tampons : les noms *Stics* sont néanmoins conservés pour pouvoir être repris en commentaires dans le code généré.

Des classes sont ensuite définies sur l'ensemble des variables en fonctions d'objectifs applicatifs :

sauvegarde des états quotidiens,
gestion de décisions quotidiennes et évaluation des stratégies culturales,
prise en compte d'évènements climatiques aléatoires,
simulation d'une culture sur un ensemble de parcelles,
simulation de rotations de cultures sur une parcelle,
etc.

Cette classification sert de base pour générer le métamodèle proposé dans les chapitres suivants (3.2).

[44] La fonction Fortran TETACRIT utilise une variable N non déclarée qui pourrait modifier par effet de bord la variable de boucle commune qui enchaîne les simulations quotidiennes : les fichiers des déclarations n'étant pas inclus au début de cette fonction dans le code Fortran, N est localement inconnue, donc déclarée implicitement en local.

[45] Le sous-programme APORES est parfois appelé avec une valeur numérique en paramètre utilisée pour évaluer la variable commune AIRG. Le paramètre formel correspondant, EAURESIDU, mesure une quantité d'eau contenue dans un résidu de culture : s'il dépasse le seuil 1, l'apport est assimilé à une opération d'irrigation supplémentaire et le compteur d'opérations NAP est incrémenté.

Indéterminations Les variables déclarées ne sont pas nécessairement utilisées : en l'absence d'initialisation, elles sont indéterminées.

Les quelque sept cents variables déclarées dans la version 5 mais non utilisées sont supprimées dans les versions C correspondantes.

Ce nombre est réduit à une cinquantaine, sans oublier les quelques variables permutées dans le sous-programme d'inversion de dominance (2.3.2) alors qu'elles n'ont reçu aucune valeur, dans la versuion 6.

Fichiers inclus La répartition des variables dans les fichiers inclus de déclarations définit une autre classification : les classes, dites groupes, sont les rangs de ces fichiers.[46]

2.4.1 Dynamique du modèle

Un premier objectif est de pouvoir séparer les variables évaluées quotidiennement par le modèle de celles qui sont invariantes sur la période de simulation.

Dynamique des unités de programmes

La séquence de supervision qui enchaîne les appels de sous-programmes (1.3.5) se déroule en trois étapes :

1. acquisition des paramètres et initialisation,
2. évaluations quotidiennes sur la période de simulation,
3. restitution des résultats.

L'examen de l'arbre des appels (2.2.3) permet ensuite de suivre la descendance des sous-programmes précédents et finalement de classer toutes les unités de programmes relativement à la dynamique d'exécution du modèle, à savoir :

1. les unités d'initialisation, utilisées exclusivement avant simulation ;
2. les unités dynamiques qui s'exécutent en cours de simulation ;
3. les unités de restitution, qui s'exécutent généralement à l'issue de la simulation, mais aussi après récolte ;[47]

Quelques unités atypiques, qui effectuent des formatages ou initialisations de variables textuelles, sont intégrées dans cette dernière catégorie.[48]

Dynamique des variables

Ce séquencement permet seulement d'établir une classification des variables informatiques en trois groupes, avec la variable $classe \geq 0$ et les conventions suivantes.

[46] Un classement a ainsi été effectué en répartissant les variables communes dans huit fichiers d'inclusion pour les besoins du projet *Hysafe* en 2003.

[47] BILCOUPE est appelé dans la boucle quotidienne pour les coupes des cultures fourragères.

[48] JULIEN convertit les numéros des jours en dates calendaires sous la forme textuelle ; NUMDEC, NBJOURD, DECADE, DEBUTDEC sont utilisées pour les données décadaires ; OTEBLANC supprime les espaces d'une chaine.

1. $classe < 100$: **variables initiales**, statiques, mises à jour exclusivement dans des fonctions appelées avant les évaluations quotidiennes ;[49]
2. $classe = 100$: **variables temporelles**, dynamiques, modifiables par affectation ou référence dans au moins une fonction d'évaluation appelée au cours des itérations quotidiennes ;[50]
3. $classe \geq 200$: **variables d'édition** ;
   (a) $classe = 200$: restitution des résultats à l'issue de la simulation ;
   (b) $classe = 300$: édition de messages ;

Variables initiales Elles sont lues ou évaluées avant la simulation. La plupart sont des variables exogènes qui définissent les paramètres du modèle, invariants. Quelques unes sont des variables de contrôle, dites options, lues avec les paramètres et utilisées pour piloter des choix.

Variables temporelles Il s'agit formellement des séries chronologiques, représentables dans des vecteurs définis pour la durée d'une campagne. La plupart sont gérées dans des variables simples, la valeur du jour remplaçant celle de la veille, ou la valeur initiale ; quelques unes le sont dans des tables.[51]

Ces variables temporelles regroupent

1. les variables endogènes du modèle,
2. des variables exogènes initialisées quotidiennement.[52]

Variables d'édition Evaluées à l'issue de la simulation, elles restituent les résultats.

Les sous-programmes LECSORTI et LECPROFI, bien qu'appelés dans la boucle quotidienne depuis SORTIE pour noter des résultats intermédiaires, sont à classer avec les fonctions d'édition : il faudra vérifier que tous les arcs du graphe des relations entre variables répertoriés ont pour extrémités des variables d'édition.

Des noms des variables *Stics* à éditer, lus depuis les sous-programmes PRINCIP et LECPROFI, sont gérés dans des tables textuelles : les valeurs des variables choisies sont affectées dans les tampons d'édition dans les sous-programmes LECSORTI, LECRAP et PROFIL.[53]

Les messages, valeurs textuelles, sont aussi des variables d'édition.

[49] Les pseudo-constantes (1.2.4) ne sont pas classées parce que non initialisées en cours d'exécution.

[50] Les sous-programmes PHOTPD, HUMCOUV et HUMHEURE, appelés directement depuis PRINCIP, reçoivent ainsi des valeurs dans des paramètres transmis. D'autres variables sont mises à jour indirectement par transmission dans une chaîne d'appels.

[51] La période retenue, bisannuelle (731 jours) dans la version 4, est devenue annuelle (366 jours) pour le projet **Hisafe** puis à nouveau bisannuelle dans la version 6.

[52] Les séries climatiques et des décisions techniques sont construites au jour le jour.

[53] Les variables textuelles VALPAR, VALRAP, VALPROF et la variable numérique DATEPROF sont liées à des options d'édition non accessibles dans le code Fortran de la version traduite.

2.4.2 Rôles des unités de programmes

Les classements des variables non temporelles sont liés aux unités de programmes qui les évaluent, et pour la version 5 aux unités logiques de lecture pour lever des ambiguïtés ou indéterminations.[54]

Les variables mises à jour dans une unité de programme appelée dans la boucle quotidienne sont temporelles. Cette boucle est contenue dans l'unité de supervision[55] et des mises à jour sont effectuées en dehors de cette boucle, elle-même incluse dans une boucle de rotation des cultures.[56]

Les actions associées sont traduites à travers une notion de rôle définie pour l'unité, et si nécessaire pour des séquences d'instructions de l'unité de programme : les variables lues ou mises à jour dans une unité ou un un bloc doté du rôle k relèvent a priori de la classe k.

Initialisation des rôles des unités et blocs

L'examen de l'unité de supervision permet de localiser relativement à la dynamique précédente les blocs d'initialisation, de simulation et de restitution des résultats. Les paramètres sont quant à eux lus dans des fichiers par des unités de programmes données.

Le codage des rôles reprend celui des classes. Les rôles des sous-programmes et fonctions sont initialement indéterminés, et celui d'un programme principal nul. Un fichier de directives est ensuite utilisé pour initialiser la distribution des rôles de blocs d'instructions dans les unités à partir des rangs de ces instructions : ces rôles sont alors transmis aux unités appelés depuis ces blocs. Les descripteurs utilisent la syntaxe

$$< nom >< role >< fin > \{< role >< fin >\}$$

pour localiser les séquences d'instructions,

$< role >$ reprenant les valeurs des classes précédentes,
$< fin >$ désignant l'instruction finale, ou une marque de fin.

Dans la description ci-après

```
ANONYME 0 17900 100 18024 200 0
INITNONSOL 0 8123 70 8162 0 8219 70 8220 0 0
LECSOL 20 0
LECTURES 0 17320 4 17323 0 0
LECSTAT 0 17899 30 17924 0 0
INITIALISATIONS 0 0
```

1. le programme principal est divisé en trois blocs avec les rôles
   - 0 du début à l'instruction 17900,
   - 100 ensuite jusqu'à l'instruction 18024,

[54] Paramètres initiaux et constantes climatiques étaient lus avec les unités logiques FICUSM et FICSTA depuis PRINCIP.

[55] Sous-programme PRINCIP pour la version 5, programme principal pour la version 6.

[56] La version 6 introduit en sus des boucles sur les couples de plantes.

- 200 après ;

2. INITNONSOL lit successivement dans la version 6 des données de la culture précédente et les paramètres du sol édités conjointement dans la simulation antérieure, ce qui se traduit avec les rôles suivants :
   - 70 pour le bloc 8124-8162 et une nouvelle lecture de l'entête du fichier des données de la culture précédente avec l'instruction de rang 8220,
   - 0 pour les autres séquences ;
3. le rôle 20 est associé à l'ensemble des instructions de LECSOL avec la lecture des paramètres du sol ;
4. la séquence 17321-17323 de LECTURES correspond au rôle 4 avec la lecture des constantes climatiques ;
5. le rôle 30 est affecté à la séquence 17899-17924, les séquences initiales et finales étant en classe 0.
6. le rôle 0 est associé à l'ensemble des instructions de INITIALISATIONS ;

Redéfinition des rôles des unités de programmes

Les directives précédentes permettent de déterminer les rôles effectifs des unités appelées depuis les sections de code indiquées à l'issue de l'analyse du code source au moyen de l'arbre des appels : ces rôles sont en effet transmis à travers les branches, jusqu'aux feuilles finales.

Si les rôles de certaines unités demeurent indéterminées, parce que les rôles des instructions d'appel ne sont pas définis, une valeur par défaut est proposée et ajoutée dans un nouveau fichier de directives (2.7). Si par ailleurs une unité apparait plusieurs fois,[57] un avertissement est édité.

Pour les rôles d'initialisation, c'est le niveau supérieur qui est choisi, ce qui conduit à réduire le nombre d'unités de classe 0.

Suppression de directives Une unité peut être utilisée pour l'initialisation ou l'édition et un rôle attribué en conséquence. Si cette unité est par ailleurs utilisée pour les évaluations quotidiennes, la directive est à supprimer explicitement, le rôle 100 étant attribué.[58]

Inversion de dominance Le mode de gestion de la relation de dominance entre cultures, avec permutation des valeurs dominant/dominé pour les variables (2.3.2), fait que le sous-programme qui effectue ces permutations, DOMINVER, est doté du rôle 100 : il est finalement exclu dans le parcours de l'arbre des appels afin que les unités de réinitialisations ou contrôle appelées n'héritent pas de ce rôle.

[57] C'est l'ultime rôle qui est retenu.

[58] Le cas se présente pour BFROID et PHOTPD.

2.4.3 Classement des variables lues

Les valeurs acquises dans des fichiers regroupent
- des paramètres du modèle,
- des options,
- des valeurs initiales de variables dynamiques,
- des valeurs de forçage pour remplacer sous conditions les valeurs calculées.

Cette organisation est traduite un plan de classement ensuite utilisé pour la vectorisation des variables.

Fichiers de paramètres

Les paramètres du modèle de culture sont lues dans des fichiers par des unités de programme dans des unités logiques, souvent l'unité 12, ce mode d'acquisition demeurant stable entre les versions : ces traitements reprennent cette organisation pour essayer de structurer ces ensembles de données.

Une premier classement reprend ces "fichiers de paramètres" tels que décrits dans la notice d'utilisation du logiciel, soit

1. des paramètres généraux lus par LECPARAM ;
2. les paramètres de la plante lus par LECPLANT, avec en fin de fichier des données relatives à un ensemble de groupes variétaux lus dans une boucle jusqu'à l'occurrence du code variétal choisi ;
3. les paramètres du sol lus par LECSOL ;
4. les séries climatiques lues par LECSTAT avec l'unité logique LECSTA, le début du fichier contenant des dates, indications et constantes pour lesquelles un autre rôle est à affecter (2.4.2) ;
5. les données techniques lus par LECTECH, sous forme de données simples ou d'actions quotidiennes, alors lues dans des tables avec les dates des interventions ;[59]
6. le fichier dit des "paramètres initiaux" est constitué de données lues dans l'unité de supervision PRINCIP pour la version 5, dans l'unité LECUSM pour la version 6, avec l'unité logique FICUSM. Il regroupe des données hétéroclites, à savoir :
   - des options de simulation ;
   - des valeurs textuelles, dont les noms des fichiers de paramètres de la plante et techniques ;
   - la période de dimulation (IWATER,IFWATER) et le numéro du sol ICHSL ;
   - les valeurs initiales de variables descriptives des horizons du sol ;
   - les valeurs initiales de variables d'état des cultures pérennes ;[60]

[59] Ces actions sont définies pour les couples de cultures associées de la version 6, y compris pour l'irrigation et la fertilisation.

[60] Ces variables sont l'indice foliaire initial LAI0, la profondeur initiale du front racinaire ZRAC0, la biomasse initiale MAGRAIN0, la quantité d'azote initiale QNPLANTE0, le stade cultural STADE0, la densité racinaire initiale DENSINITIAL pour les 5 horizons.
Elles sont définies dans la version 6 pour les couples de cultures associées.

Constantes climatiques Le fichier des données climatiques contient en entête des constantes définies pour une latitude et lues avant les séries, dans PRINCIP pour la version 5, LECTURES pour la version 6, toujours avec l'unité logique FICSTA.

Paramètres non classés La version 6 introduit un nouveau fichier de "*paramètres nouvelle version en attente d'être redirigés dans leurs fichiers respectifs*", lu par LECPARAMV6.

Cultures associées

La version 6 introduit une boucle pour lire les paramètres des plantes pour les cultures associées, les fichiers requis étant ouverts par LECPLANT.

Les données techniques sont quant à elles lues pour une seule culture, les variables définies pour des couples de valeurs étant gérées dans LECTECH à travers la variable commune IPL.

Plusieurs versions de LECUSM sont disponibles pour lire les paramètres initiaux : les variables définies pour des couples de valeurs sont désignées dans les listes de lecture avec les valeurs d'indices

- 1 et 2 pour les modules sources "`lecusmPURE.for`" et "`lecusmCAS.for`",
- IPL pour "`lecusmCASetPURE.for`".

Etat de la culture précédente

Le fichier issu de la simulation de la culture précédente, édité par le sous-programme RECUP, est lu dans la phase d'initialisation

- pour la version 5 dans le sous-programme INITIAL,
- pour la version 6 dans le sous-programme INITNONSOL appelé depuis INITIALISATIONS.

Les données de la culture précédente sont :

la période de simulation,
d'autres variables descriptives des horizons du sol,
les tables qui décrivent l'état des résidus dans les couches de la zone humifère.

Paramètres du sol Les caracteristiques permanentes du sol, pratiquement les paramètres lus, sont dans la version 6 recopiées dans ce fichier par RECUP, et relus par INITNONSOL. Si l'état de la culture précédente est pris en compte, ces nouvelles valeurs remplacent celles lues auparavant par LECSOL.

Classement des variables lues

Ces modes d'acquisition sont reconnus à travers les unités de programme qui effectuent les lectures et accessoirement les unités logiques. Ils sont traduits pour la version 6 dans le classement suivant.

1. **Constantes prédéfinies et paramètres non classés**

- (*classe* = 5) : paramètres dits généraux lus depuis LECPARAM dans `"param.par"` ;
- (*classe* = 9) : nouveaux paramètres de la version 6 lus depuis LECPARAMV6 dans `"paramv6.par"` ;
- (*classe* = 4) : constantes climatiques (4.3.7) lues dans `"stat.dat"` depuis LECTURES ;[61]

2. **Paramètres de la culture**
   (a) Paramètres de la plante lus par LECPLANT ;
      - (*classe* = 10) : espèce végétale ;
      - (*classe* = 11) : groupe variétal ;[62]

   (b) Paramètres du sol lus par LECSOL `"param.sol"` ;
      - (*classe* = 20) ;

   (c) Séries climatiques lues par LECSTAT dans `"stat.dat"` ;
      - (*classe* = 30) ;

   (d) itinéraire technique lu par LECTECH ;
      - (*classe* = 40) : facteurs techniques ;
      - (*classe* = 41) : opérations quotidiennes ;[63]

3. **Valeurs initiales et de forçage**
   (a) Etat initial du sol et de la culture
      - (*classe* = 60) : valeurs lues avec les paramètres initiaux par LECUSM dans `"travail.usm"` ;

   (b) Etat final de la culture précédente
      - (*classe* = 70) : données de la culture précédente lues depuis INITNONSOL dans `"recup.tmp"` ;

   (c) Valeurs de forçage
      - (*classe* = 80) : besoins en eau et en azote lus par LECEPITAF ;
      - (*classe* = 90) : indices foliaires lus par LECLAI ;

Si des variables sont lues dans plusieurs fichiers, c'est la classe de rang supérieur qui est retenue.

Mises à jour de paramètres

Des variables lues, paramètres du modèle pour la plupart, sont souvent réévaluées ou réinitialisées dans différents contextes, après lecture, lors de l'évaluation des conditions initiales, voire en cours de simulation. Si des mises à jour sont effectuées dans les unités de lecture, la classe retenue est celle de l'unité dans laquelle la lecture est effectuée.

[61] La date de la première observation climatique est également lue, soit les variables STICS ANSEMIS et JULZERO.

[62] Les valeurs des 17 paramètres variétaux sont ajoutées en fin de fichier avec des séquences introduites par le code variétal.

[63] Les interventions culturales sont gérées dans des tables indicées par le rang de l'intervention, une première table contenant les dates. Ces variables, exogènes, sont reprises dans la boucle des simulations quotidiennes pour constuire des séries chronologiques.

1. **Calibrage par optimisation de paramètres**
   Le sous-programme LECOPTIM lit dans le fichier "`param.sti`" des nouvelles valeurs de paramètres issues d'un module d'optimisation pour quelques paramètres à choisir parmi plusieurs centaines : les substitutions s'effectuent dans de longues séquences conditionnelles.
2. **Options**
   Les paramètres sont quelquefois mis à jour après lecture ou dans d'autres unités de programmes selon les valeurs d'options, acquises pour la plupart avec les paramètres généraux.
3. **Corrections**
   Des corrections sont effectuées en cas d'incompatibilité, souvent aux limites[64] en surface ou pour choisir une borne, la substitution étant signalée dans le fichier historique.

Ces cas sont signalés dans le compte-rendu global, pour les options et corrections,[65]

- par exploration du graphe des relations entre variables pour les évaluations,
- par examen des modes d'affectation dans le dictionnaire pour les réinitialisations.

Valeurs de forçage

Certaines variables temporelles peuvent être simulées ou observées.

1. Les stades de développement et reproduction sont lus avec les paramètres de la plante ou simulés, la valeur 999 signifiant que ces stades sont à évaluer.
2. Les valeurs de forçage des besoins en eau et en azote et des indices foliaires sont acquises dans un fichier dont le nom est un paramètre initial.

2.4.4 Classification des variables

Au cours de l'analyse des instructions, les variables qui reçoivent une valeur dans des unités de programmes pour lesquelles les rôles sont définis sont affectées dans la classe correspondante ; les autres, supposées dynamiques, appartiennent a priori à la classe 100.

Les classifications des variables s'effectuent ensuite en plusieurs étapes.

1. Les rôles des unités de programmes, éventuellement des séquences dans ces unités, sont utilisés pour :
   - séparer les variables dynamiques, séries temporelles de classe 100, des variables statiques d'initialisation, de classe < 100, ou d'édition, de classe ≥ 200 ;

[64] Une profondeur de semis nulle est remplacée par la valeur 1.

[65] Les mises à jour dans LECOPTIM sont masquées pour ne pas faire apparaître les 703 paramètres optimisables de la version 6 dans ces listes. Il en est de même pour DOMINVER afin de ne pas faire apparaître les permutations de paramètres (2.3.2).

- d'affiner la classification les variables d'initialisation, de classe > 0 pour les variables lues, nulle pour les autres à ce stade avec la convention adoptée ;

l'examen des structures itératives permet d'affiner cette classification.

2. L'exploration du graphe des relations entre variables va permettre :
   - de construire des classes de variables statiques, finalement des classes du paramètres du modèle qui définissent les conditions initiales avant simulation ;
   - de repérer des variables lues mises à jour dans la boucle quotidienne, valeurs initiales de variables dynamiques ou paramètres modifiés, temporelles ;
   - d'isoler les variables d'édition, état final pour la restitution de résultats ;
   - de repérer les variables initialisées dans le code ou gérant des cumuls, sommets isolés et relations exclusivement réflexives ;
3. D'ultimes ajustements sont réalisés pour restituer dans le dictionnaire des variables les classes à partir desquelles va s'effectuer la vectorisation dans l'étape de générarion du code (3.2.4).

Cultures associées Le mode de gestion des cultures associées introduit des perturbations pour lesquelles des solutions contextuelles ont été adoptées.

Variables de gestion des cultures associées Les variables communes IPL, ENS et NBPLANTES sont utilisées pour gérer les couples de valeurs pour les cultures associées (2.6.1). Il s'agit de fait de constantes qui prennent les valeurs 1 ou 2 : elles sont rétablies dans la classe 1.[66]

Inversion de dominance Le mode de gestion des cultures associées fait que les valeurs des paramètres des deux plantes sont permutées si la plante dite principale lors de l'initialisation est dominée en cours de simulation : la mise à jour dans la boucle quotidienne fait tous ces paramètres apparaissent comme variables temporelles.
La solution adoptée consiste à exclure sous-programme DOMINVER qui effectue ces permutations.

Exploration finale du graphe des relations entre variables

Une variable évaluée hérite des classes de ses prédécesseurs.
L'absence de prédécesseurs correspond à l'affectation d'une constante numérique.

Classement des variables initiales Toute variable initiale non lue appartient normalement à la classe 0 avec les conventions de codage des rôles adoptées.

Pour toute variable y, l'ensemble des arcs $\{x_i \to y\}$, le sommet x_i désignant une autre variable initiale, est examiné : le classement final, qui reprend le classement des variables lues, est alors le suivant.

[66] IPL et ENS apparaissent comme temporelles car utilisées comme variables de boucle dans les unités d'évaluation ; NBPLANTES est lue avec les données de la culture précédente ;

1. **Constantes prédéfinies et paramètres non classés**
   $(classe > 0) \wedge (classe < 10)$
   - $(classe = 1)$: sommets sans prédécesseurs ;[67]
   - $(classe = 2)$: sommets avec arcs réflexifs, qui représentent des dénombrements ou cumuls ;
   - $(classe > 2) \wedge (classe < 10)$: tous les prédécesseurs sont des constantes ou paramètres non classés et la variable hérite de la classe de rang maximal ;
2. **Paramètres de la culture**
   $(classe \geq 10) \wedge (classe < 60)$
   La variable hérite de la classe k si tous les antécédents appartiennent aux classes k, ou $k + 1$, ou sont des constantes : on retrouve les classes de paramètres lus, soit
   (a) $(classe = 10) \vee (classe = 11)$: espèce végétale et groupe variétal ;
   (b) $(classe = 20)$: paramètres du sol ;
   (c) $(classe = 30)$: séries climatiques ;
   (d) $(classe = 40) \vee (classe = 41)$: facteurs techniques et opérations quotidiennes ;

   Ces quatre classes de paramètres de la culture définissent des composantes connexes, ce qui traduit la logique de modélisation du système sol-plante et des évènements quotidiens.
3. **Variables évaluées dans l'étape d'initialisation**
   $(classe \geq 50) \wedge (classe < 60)$
   - $(classe = 50)$: un prédécesseur est une variable évaluée dans la boucle quotidienne ;[68]
   - $(classe = 51)$: il existe deux prédécesseurs paramètres de la culture de classes différentes ;[69]
4. **Valeurs initiales et de forçage**
   $(classe \geq 60) \wedge (classe < 100)$
   (a) $(classe = 60)$: initialisation d'une culture pérenne avec des paramètres initiaux (2.7) ;
   (b) $(classe = 70)$: données de la campagne précédente ;
   (c) $(classe = 80) \vee (classe = 90)$: valeurs de forçage ;

Dictionnaire des variables

La classification reprend la structure du modèle. Elle permet de définir

[67] Il s'agit de constantes, initialisées par affectation ou passage de paramètre, conditionnellement ou non.

[68] Les dites variables sont normalement initialisées avant l'évaluation : ces sources d'ambiguïtés sont signalées.

[69] Les variables climatiques TRG et TRGEXT étaient modifiées par le paramètre technique TRANSPLASTIC pour les cultures sous abri dans la version 5.

1. des états numériques avant simulation, après acquisition des paramètres puis initialisation ;
2. des états quotidiens décrits par des séries chronologiques ;

La classification précédente définit, par rapport à la dynamique du modèle, trois ensembles non disjoints :

1. **les variables statiques**, invariantes sur la période de simulation : constantes, paramètres, données initiales ;
2. **les variables dynamiques**, évaluées quotidiennement ;
3. **les variables d'édition** exclusivement utilisées pour restituer des résultats ;

X peut ainsi être lue avec les paramètres et reprise pour les évaluations quotidiennes.[70]

Codage des classes Trois classifications sont ici définies :
- *classe* est relative à l'ensemble des variables ;
- *hclasse* gère les variables *h*ors boucle quotidienne, statiques ;
- *iclasse* regroupe les valeurs *i*nitiales issues de la culture précédente ;

1. Les valeurs des classes des variables sont initialement notées dans *classe*.
2. Une variable peut recevoir une valeur lue puis être mise à jour lors des simulations quotidiennes : la classe de lecture est notée dans *hclasse*, sauf pour les données de la culture précédente.
3. La classe de lecture des données de la culture précédente est notée dans *iclasse*.

Initialisation des variables temporelles Le classement hors boucle quotidienne dans *hclasse* permet de retrouver les variables utilisées pour initialiser les variables temporelles, principalement les classes de paramètres de la culture mais aussi les valeurs initiales et de forçage.

Les variables temporelles appartenant jusqu'alors à la classe 100, le classement est complété pour les variables lues mises à jour dans la boucle quotidienne par application de la règle

Si $(iclasse > 0)$ alors $classe = 100 + iclasse$; sinon $classe = 100 + hclasse$;

ceci pour distinguer dans l'étape suivante valeurs initiales ($classe = 70$) et paramètres.[71]

Si par ailleurs la variable ne gère pas un paramètre lu, *hclasse* reçoit la valeur de *ihclasse*, soit

Si $(hclasse \leq 0)$ alors $hclasse = iclasse$;

[70] PROFHUM est lue comme paramètre du sol, réévaluée pour prendre en compte le tassement, restituée pour la campagne suivante.

[71] PROFHUM appartient alors aux classes 170 et 20, alors que ZESX, paramètre seulement mis à jour, relève des classes 120 et 20.

Relecture de variables statiques Si un paramètre est relu avec les données de la campagne précédente sans être ensuite évalué dans la boucle quotidienne, alors

$classe = iclasse$ et $hclasse$ désigne la classe de paramètres.[72]

Le cas de figure se présente avec la relecture d'un ensemble de paramètres du sol, certains étant corrigés pour prendre en compte tassement et détassement.[73]

2.5 Gestion des variables

Si les variables locales sont reprises telles quelles, les variables communes sont pour la plupart gérées dans des vecteurs définis à partir des classifications qui précèdent, les méthodes d'allocation et de restitution étant décrites dans le chapitre 3.

Un mécanisme est mis en place pour pouvoir copier une variable commune dans une variable locale homonyme au début d'une fonction, puis éventuellement la restituer à l'issue des évaluations.

2.5.1 Variables de simulation

Toutes les citations des variables communes, dans une affectation, une condition logique, une expression numérique ou logique, une liste de paramètres, une zone d'indices, sont examinées. Un dénombrement est effectué pour les variables citées dans les fonctions appelées depuis la boucle quotidienne, dites de simulation. Les listes d'entrée-sortie et les variables utilisées pour la gestion des fichiers ne sont pas prises en compte dans ces dénombrements, ces opérations relevant de l'initialisation ou de l'édition.

Les variables de simulation regroupent les variables temporelles, mises à jour, et des variables exclusivement consultées, dont quelquefois des variables textuelles utilisées dans l'évaluation de conditions logiques (2.1.5).

Après analyse du code, le dictionnaire des variables communes contient le nombre de ces citations, nul pour les variables utilisées exclusivement pour l'initialisation ou d'édition, positif pour celles référencées dans les fonctions de simulation.

2.5.2 Copies locales de variables communes

Les variables communes numériques étant ensuite gérées dans des tables réelles (3.1.1), des conversions de types sont à prévoir pour restituer localement des variables entières, qu'elles le soient ou non dans le code Fortran : la variable locale est à initialiser avec la variable commune et sa valeur à restituer si nécessaire avant de rendre le contrôle à l'appelant. Le rang dans le dictionnaire initial est alors conservé dans le dictionnaire local pour gérer la correspondance.

[72] ARGI, non évaluée quotidiennement, relève des classes 70 et 20.

[73] Ces variables, qu'elles soient modifiables ou non, sont sauvegardées pour la cultutre n et relues pour la culture $n+1$.

Il faut aussi vérifier l'absence de modification avant appel d'un sous-programme et d'une fonction qui utiliserait une telle variable commune, d'où la nécessité d'examiner le graphe des relations entre variables. [74]

D'autres directives relatives à la vectorisation sont traduites dans ce fichier (3.10.1).

Variables d'indices

Pour toute variable commune citée en indice dans une table ou dans une expression entière qui évalue un indice, une copie locale est effectuée : il est vérifié que toutes ces variables sont bien entières.[75]

Ces variables communes sont parfois évaluées localement avant d'être citées comme indices : c'est le cas pour les variables de gestion des dates des stades XXX de développemment, *Ixxx* et *Nxxx*.

Variables de boucle

Une copie locale des variables de boucle, sauf celles qui gèrent les cultures associées,[76] est effectuée : la mise à jour est notée afin que la valeur de sortie de boucle puisse être restituée à l'appelant, comme en Fortran.

Paramètres effectifs entiers

Les variables communes numériques étant ensuite gérées dans des tables réelles (3.1.1), un appel d'une fonction définie avec un paramètre entier ne serait pas conforme à la déclaration initiale du prototype de la dite fonction : une copie locale est en conséquence effectuée pour les variables communes entières transmises comme paramètres effectifs, la conversion de type étant effectuée explicitement lors de l'initialisation.

Il est vérifié qu'un tel paramètre n'est pas modifié dans la fonction appelée, donc n'est pas défini par référence.

Variables désignées

La possibilité de forcer une copie locale a été introduite pour pouvoir désigner la variable commune par son nom dans le code de la fonction C, plutôt que d'accéder à la composante du vecteur : les couples (u, x), cités dans un fichier, désignent les variables communes x pour lesquelles une copie locale est demandée dans l'unité de programme u. Lors de la génération du code de la fonction correspondante dans l'étape suivante, la variable locale est initialisée, et sa valeur restituée en fin d'exécution chaque fois qu'une mise à jour est repérée dans le

[74] Les dates des stades de développement, gérées dans des variables entières, sont évaluées dans le corps des sous-programmes.

[75] Le contrôle est systématique et un avertissement édité même si la conversion est faite explicitement avec une fonction Fortran.

[76] IPL et ENS

graphe des relations entre variables (3.3) : il faut évidemment s'assurer que ces variables communes ne sont pas utilisées dans une unité de programme appelée.

Ce mécanisme a été étendu pour pouvoir générer de nouvelles fonctions C qui utiliseraient les variables communes citées, les variables locales étant initialisées puis les valeurs restituées en cas de mise à jour, ceci pour permettre d'effectuer les évaluations avec les dites variables (3.10.1). L'opérateur d'indirection $*$ est ajouté devant le nom d'une variable qui peut être évaluée ($*x$), pour indiquer qu'il s'agit de traiter la référence et non exclusivement la valeur. Cette opération est également possible pour les fonctions issues des unités de programmes, la restitution étant alors forcée.

2.5.3 Initialisation de variables dynamiques

Alors que des variables dynamiques ne sont pas définies avant l'installation de la culture, comme les variables descriptives de la plante, d'autres existent et contiennent des valeurs initiales, qu'il s'agisse de gérer des ressources ou une culture pérenne.

Ces variables dynamiques initialisées sont répertoriées et l'information est notée dans le dictionnaire des variables communes pour pouvoir ensuite gérer le processus de réinitialisation dans les étapes ultérieures.

2.6 Gestion des couples de cultures

La version 6 introduit une ou deux dimensions binaires sur un grand nombre de variables, numériques, logiques ou textuelles, pour la gestion de couples de cultures associées : une culture simple est gérée dans ces mêmes structures binaires

Les indices utilisés, définis sur $\{1, 2\}$, permettent d'identifier respectivement

1. la plante cultivée *(principale, associée)*,
2. la dominance *(ensoleillée, ombragée)*,

soit

- deux ou quatre occurrences pour un couple de cultures,
- une seule occurrence pour une culture simple, les valeurs pour l'indice 2 n'étant plus définies.

La dominance peut s'inverser sur la période de simulation : la plante dominante devient principale, et la plante dominée associée (2.3.2).

2.6.1 Variables de gestion des couples

Ces dimensions binaires sont gérées avec les variables suivantes :

1. IPL $= 1$ désigne la culture dominante, dite principale, et IPL $= 2$ la culture dominée, dite associée ;
2. NBPLANTES vaut 1 pour une culture simple, 2 pour un couple de cultures ;

3. ENS = AS désigne la zone éclairée et ENS = AO la zone ombragée, AS et AO étant des pseudo-constantes Fortran qui valent respectivement 1 et 2.

Indiçage des couples

Après examen des quelque 15 000 citations des variables concernées, les couples sont indicés comme suit.

1. IPL, variable commune, gère les deux plantes cultivées, avec les valeurs 1 et 2 : elle est parfois utilisée comme variable de boucle, depuis 1 jusqu'à NBPLANTES.[77]
2. ENS, variable commune, gère l'ensoleillement avec généralement les deux pseudo-constantes AS=1 et AO=2 (1.2.4) : elle est aussi utilisée comme variable de boucle, depuis AS jusqu'à AO, y compris en l'absence de culture associée.[78]
3. Les nombres 1 et 2 apparaissent en clair.
4. Les variables de boucle locales II et IZ varient, comme pour les deux premiers items, depuis 1 jusqu'à NBPLANTES ou depuis AS jusqu'à AO.
5. Aucune expression à évaluer n'est utilisée pour ces indices.

2.6.2 Formes syntaxiques

Une reconnaissance de ces formes est effectuée pour pouvoir paramétrer la prise en compte d'une culture associée dans les étapes suivantes, et ignorer ces dimensions binaires pour simuler les cultures simples.

Forme itérative Les évaluations sont faites dans des boucles initialisées par des instructions de la forme

DO $< n >$ IPL $= 1,$ NBPLANTES
DO $< n >$ ENS $=$ AS, AO

La première itération s'exécute une ou deux fois en fonction de la valeur NBPLANTES alors que la seconde s'exécute toujours deux fois. Les deux boucles sont parfois imbriquées.

Forme conditionnelle Certains blocs conditionnels, structurés ou gérés par des branchements, sont contrôlés par des prédicats qui utilisent les variables NBPLANTES, IPL, ENS, y compris dans le corps des boucles précédentes.

[77] La valeur de la variable commune IPLT se trouve ainsi être modifiée après exécution de la boucle. Quant à la borne NBPLANTES, elle prend les valeurs 1 ou 2.

[78] La variable commune ENS vaut finalement 3 après exécution d'une telle boucle.

Forme séquentielle Les traitements s'effectuent parfois en séquence par affectation dans :[79]

- ENS des valeurs des pseudo-constantes AS ou AO (1.2.4) ;
- IPL des constantes 1 ou 2 ou NBPLANTES ;
- NBPLANTES de 1 ou 2 ;

2.6.3 Séparation des valeurs

Dans le code Fortran, un espace est réservé pour la plante dominée et/ou la zone ombragée, que la culture soit pure ou non. Ce mode de gestion de couples complexifie la gestion des variables communes et double l'espace requis :[80] une solution a été adoptée pour pouvoir n'allouer que l'espace utile pour une culture pure (3.2.1).

Les variables déclarées avec la valeur 2 comme borne supérieure, inexistantes avant la version 6, le sont a priori exclusivement pour gérer ces couples de cultures, ce qui est confirmé par l'utilisation des variables de gestion des couples de cultures en indices : un avertissement est édité pour les quelques variables communes ou locales pour lesquelles l'ambiguïté demeure.[81]

Les instructions dans lesquelles le second élément est référencé sont repérées comme suit :

1. l'indice est le nombre 2 ou la pseudo-constante AO ;
2. une seconde itération est effectuée dans une boucle bornée supérieurement par NBPLANTES = 2 ;
3. les variables communes IPL ou ENS utilisées en indices contiennent la valeur 2 ;

2.6.4 Paramètres des unités de programmes

Les variables qui gèrent des couples sont aussi utilisées comme paramètres de sous-programmes et fonctions : cette dimension est supprimée, et le prototype décrit en conséquence.[82]

2.7 Traduction informatique

Cette étape est actuellement exécutée par la commande *S6*.

Production de la commande

Des constantes et listes sont prédéfinies pour faciliter l'intégration des versions successives avec

[79] Une seule affectation conditionnelle est effectuée : si (CODOPTIM = 2) alors IPL reçoit la valeur 2.

[80] Le système sol-plante est défini pour chaque culture sur la durée de la campagne culturale.

[81] La variable locale EPCINI(2) dans TASSEMENT gère effectivement un autre couple (4.5.2).

[82] Le paramètre peut être dans le code Fortran la variable qui gère le couple, ou un élément du couple.

1. des tailles de tables qui peuvent s'avérer insuffisantes à l'exécution,
2. des noms symboliques introduits pour traiter des situations contextuelles, en l'occurrence
   - SUPER pour désigner l'unité de supervision PRINCIP des anciennes versions,
   - BASCULE pour le sous-programme problématique d'inversion de dominance DOMINVER (3.2.1),
   - CALIBRE pour l'unité d'acquisition des paramètres optimisés qui remplacent les paramètres lus (2.4.3),
   - CONST1 et CONST2 pour les constantes textuelles 'AO' et 'AS'...

La solution pratique est de lancer une première exécution. Les échecs liés aux franchissements de limites vont se solder par un déroutement qui clôt l'exécution après édition d'un message explicite. A priori, il va suffire d'ajuster la valeur de la constante C correspondante dans le code source `"S6.c"` puis de générer à nouveau la commande au moyen du *make*.

D'autres constantes gèrent des limites fonctionnelles, comme le nombre de dimensions des tables MAXDIM : si les algorithmes peuvent potentiellement prendre en compte des dimensions supplémentaires, des restrictions sont imposées à travers ces valeurs prédéfinies, le code n'étant pas nécessairement validé au-delà.

Le traitement de nouvelles versions peut rendre nécessaire le développement de fonctions nouvelles liées à l'utilisation d'autres modes de programmation, comme pour la version 6 (2.6).

Fichiers de directives

Deux fichiers sont à fabriquer avant l'exécution de la commande pour gérer les fonctions intrinsèques (2.2.1) et les rôles des unités de programmes (2.4.2).

Un autre fichier peut être fourni pour traiter des variables communes dans des variables locales (2.5.2).

Fonctions intrinsèques Fortran et fonctions C équivalentes

"`stics.f`" contient les noms des fonctions intrinsèques Fortran, avec sur la même ligne les fonctions C équivalentes, ou la marque $*$ pour des conversions de types (3.3.6). Lors d'un premier passage ou pour une version nouvelle, les noms symboliques inconnus sont répertoriés dans le compte-rendu.

Rôles des sections de code dans les unités de programmes

"`stics.role`" est lié au code Fortran traité pour les directives qui utilisent les rangs des instructions dans le code normalisé (1.1.7). Cette dépendance peut être supprimée en intégrant ces séquences dans des sous-programmes de lecture spécifiques.

Pour une nouvelle version, tant que les rôles déduits des directives ne correspondent pas aux rôles effectifs évalués après construction de l'arbre des appels des unités de programmes (2.4.2), ce fichier est à mettre à jour. Les non concordances sont signalées dans le compte-rendu global et des directives sont proposées dans "`stics.role.k`" après les directives

antérieures, k dénombrant ces non concordances : le fichier définitif est pratiquement validé à l'issue d'une ou deux itérations.

Copies locales de variables communes

"`stics.copie`" permet de forcer la création de variables locales : le couple de noms (u, x) indique qu'il faut déclarer dans la fonction $U()$ la variable locale x, copie de la variable commune X, la valeur de x étant étant restituée dans X en fin d'exécution.

Appel et exécution

L'appel

S6 *stics*

lit ces fichiers de directives, soit :

"`stics.f`", table de correspondance entre fonctions intrinsèques Fortran et C ;
"`stics.role`", règles de classification des variables dans le code Fortran normalisé ;
"`stics.copie`", liste de variables communes à déclarer localement ;

et ceux produits antérieurement, soit :

"`stics.inc`", liste des fichiers inclus de déclarations, produit par *S2* ;
"`stics.upf`", liste des unités de programmes ;
"`stics.ins`" pour l'analyse séquentielle des instructions par unités de programmes ;
"`stics.dic`", dictionnaire des variables communes partagées ;
"`stics.do`", pour retrouver les corps des boucles DO ;

Elle produit les fichiers :

"`stics.tag`", liste des codes textuels ;
"`stics.lien`", représentation textuelle de l'arbre des appels ;
"`stics.fils`", représentation matricielle de cet arbre ;
"`stics.pro`", description formelle des prototypes ;
"`stics.para`", liste des paramètres formels homonymes de variables communes ;
"`stics.graphe`", graphe des relations entre variables communes ;
"`stics.arc`", liste des arcs ;
"`stics.sommet`", liste des sommets ;
"`stics.ante`", sommets origines ;
"`stics.succ`", sommets extrémités ;
"`stics.com`", dictionnaire des variables communes ;
"`stics.loc`", dictionnaire des variables locales ;
"`stics.lex`", lexique des fonctions *Stics* ;
"`stics.couple`", instructions de contrôle de la gestion des couples de cultures ;
"`stics.cr6`", compte-rendu global ;

"`stics.S6`", compte-rendu détaillé ;
"`stics.S6ass`", compte-rendu détaillé propre aux cultures associées (2.6) ;
"`stics.indice`", liste des variables communes utilisées en indices ;
"`stics.role.k`", nouveau fichier des directives de codage des rôles, $k > 0$ étant le nombre de directives supplémentaires proposées ;

Les erreurs importantes se soldent par un déroutement qui clôt l'exécution. Outre les absences de fichiers, il s'agit souvent pour des versions nouvelles de débordements de piles : il suffit alors de modifier les constantes prédéfinies qui majorent la taille des tables associées.

2.7.1 Codes textuels

La liste des codes textuels est éditée pour être reprise dans l'étape suivante de génération du code afin d'initialiser les variables numériques associées et de construire les expressions logiques numériques (2.1.5), la correspondance s'effectuant sur la base du rang dans la liste.

Les rangs des instructions qui utilisent ces codes textuels sont édités sur la ligne avec un tiret séparateur.

2.7.2 Arbres des appels

L'arbre des appels est édité en clair sous une forme classique avec décalage pour chaque génération.

Une seconde forme numérique édite la liste des arcs associés à ces appels, avec comme numéros des sommets les rangs des unités de programmes dans le lexique : elle est reprise dans l'étape de génération de code pour déclarer les prototypes des fonctions appelées.

2.7.3 Prototypes des fonctions

Les paramètres d'une fonction sont identifiés par un rang dans une liste, le rang 0 étant affecté au nom de la fonction. Les prototypes des fonctions sont représentables sous la forme d'ensembles de sextuplets

(nom,rang,type,dim,indi,stable),

dans lesquels

nom identifie la fonction ou le paramètre formel ;
rang est nul pour la fonction, donne le rang du paramètres sinon ;
type précise les types ;
dim donne le nombre de paramètres si ($rang = 0$), la dimension des paramètres sinon ;
indi code la définition par valeur ou référence ;
stable vaut 1 si le paramètre est inutile, 0 sinon (2.2.5) ;

Les paramètres homonymes d'une variable commune sont restitués avec la première lettre majuscule

Homonymies paramètres et variables communes

Les problèmes relatifs aux homonymies entre paramètres et variables communes étaient initialement restitués dans le fichier compte-rendu. Des solutions ont été trouvées pour la traduction C :

1. les paramètres effectifs et formels dont les noms sont invariants sont supprimés ;
2. les autres paramètres formels sont des variables locales, ou des pointeurs s'ils sont définis par référence.

Ces problèmes sont néanmoins localisés pour contrôle dans le fichier annexe "`stics.para`".

2.7.4 Utilisation des variables communes par fonctions

Le graphes des relations entre variables communes sont repris pour documenter l'utilisation de ces variables. Les noms des fonctions sont par convention suivis des parenthèses () et les noms des tables des crochets [].

Les noms des variables sont placés entre deux symboles $ pour pouvoir rechercher des variables spécifiques, par exemple x, ou $x[]$.

Graphe des relations entre variables communes

Un arc entre deux variables communes dans une fonction est représenté par le quintuplet

$$(module, depuis, vers, ndepuis, nvers)$$

dans lequel

module identifie la fonction qui contient l'arc,
depuis désigne le sommet origine,
vers désigne le sommet extrémité,
ndepuis et *nvers* sont les rangs des sommets dans le dictionnaire des variables communes.

Le triplet $(module, ndepuis, nvers)$ est édité dans le fichier "`stics.graphe`".

Variables citées : liste des sommets

Toutes les variables citées dans les unités de programmes sont finalement vues comme des sommets, origines ou extrémités d'arcs, ou sommets isolés : ils sont édités dans "`stics.sommet`" pour pouvoir trouver les vecteurs requis dans le paramétrage des fonctions de l'étape suivante.

Variables utilisées dans les évaluations numériques

Pour chaque fonction sont éditées les listes des variables modifiables, sommets extrémités, et des variables exclusivement consultées, sommets origines : les citations sont dénombrées et indiquées entre parenthèses. Ces deux listes

sont utilisables pour suivre les processus d'évaluations : elles sont éditées dans les fichiers

- `"stics.succ"` pour les variables mises à jour,
- `"stics.ante"` pour celles qui sont seulement consultées.

2.7.5 Dictionnaire des variables communes

Les variables communes classées sont reprises dans un dictionnaire, une variable étant décrite par le sextuplet

$$(nom, classe, hclasse, nlu, type, dim)$$

dans lequel

nom désigne la variable,
classe est la classe finale (2.4.4) et
hclasse la classe hors boucle quotidienne,
$nlu > 0$ désigne le rang de l'unité de lecture,
$type > 0$ code les cinq types de variables, $type = 0$ les images numériques (2.1.5),
dim est le nombre de dimensions.

Si $dim > 0$ ($dim > 1$ pour les images numériques des tables de codes textuels), les couples $(binf, bsup)$ sont ajoutés pour gérer les bornes inférieure et supérieure propres à chaque dimension.

2.7.6 Dictionnaire des variables locales

Une variable locale est déclarée, explicitement ou implicitement, dans une fonction : un dictionnaire de ces variables locales est alors constitué. Des variables supplémentaires sont ajoutées, pour gérer les indices et remplacer les variables de boucle communes.

Une ligne du dictionnaire est décrite par l'octuplet

$$(module, formel, ini, icommun, copie, nom, type, dim)$$

dans lequel

module contient le nom de la fonction,
$formel$ vaut 1 si la variables est un paramètre formel, 0 sinon,
ini contient la valeur initiale des constantes décimales (2.1.6) , '*' sinon,
$icommun > 0$ code le rang dans le dictionnaire des variables communes,
$copie > 0$ indique si la variable commune est localement mise à jour (2) ou seulement citée (1),
nom désigne la variable,
$type > 0$ code les cinq types de variables avec en sus le code 6 pour les réels double précision,
dim est le nombre de dimensions.

Si ($dim > 0$), les couples ($binf, bsup$) sont ajoutés pour gérer les bornes inférieure et supérieure propres à chaque dimension.

Toute variable locale homonyme d'une variable commune masque cette variable commune.

2.7.7 Lexiques des fonctions

La description des unités de programmes Fortran (1.1.6) est complétée avec des attributs supplémentaires utilisés dans l'étape suivante pour définir les prototypes des fonctions C :

- *type* est le type de la fonction, int par défaut pour récupérer les codes d'erreurs, float sinon ;
- *nom* est l'identificateur de la fonction ;
- *hors* vaut 0 pour la simulation,
  1 pour l'initialisation,
  2 pour l'édition des résultats,
  3 pour l'initialisation des messages ;
- *debut* et fin sont les rangs des premières et dernières instructions ;
- *ncommun* dénombre les fichiers inclus de déclarations des variables communes ;
- *nlect* est le nombre d'instructions de lecture des fonctions d'acquisition des "paramètres" ;
- *edi* dénombre les appels des unités d'édition ;

2.7.8 Compte-rendu du traitement des noms symboliques

L'analyse des noms symboliques s'effectue par examen des unités de programmes Fortran puis exploration des arbres des appels de fonctions et de relations entre variables. Un compte-rendu global est restitué ainsi qu'un compte-rendu détaillé qui regroupe des messages relatifs aux instructions et aux unités de programmes

Compte-rendu global

Pour pouvoir valider le classement des unités de programmes, la liste des unités appelées est editée en fonction du rang des instructions. Le classement effectué à l'issue du traitement du code est comparé au classement lu et les différences sont signalées.

Fonctions Le processus de construction des prototypes des fonctions est restitué à travers le suivi de la transmission des paramètres pour déterminer s'ils sont à déclarer par valeur ou référence.[83]

[83]Les transmissions s'effectuent finalement pour la plupart par valeur sur une seule génération, au plus sur deux avec les fonctions Fortran TETACRIT et DECANGLE.

La plupart des paramètres des sous-programmes et fonctions Fortran peuvent être transmis par valeur car non modifiés : il reste huit variables communes transmises par référence dans la boucle quotidienne.[84]

Les prototypes sont ensuite édités avec la liste des variables communes qui peuvent être supprimés comme paramètres par application de la règle de la section 2.2.5 : sept des variables communes précédentes transmises par référence sont ainsi supprimés.[85]

Variables Les nombres d'arcs et de variables communes citées sont évalués pour chaque unité de programme Fortran : la ligne

F n (s_1, s_2) $[+k]$

indique que n arcs sont définis dans l'unité F depuis s_1 sommets origines vers s_2 sommets extrémités, avec en sus $k > 0$ sommets isolés, variables communes exclusivement consultées : les couples $(0, 0)$ correspondent à des sous-programmes qui ne contiennent que des appels et des fonctions qui effectuent des évaluations avec des variables locales, dont des paramètres, et rendent des valeurs.

Les listes des tables communes et locales et leurs tailles sont éditées et des dénombrements sont effectués. Les variables locales transmises par référence sont également signalées.

Effets de bord Les variables communes modifiables lors d'une transmission de paramètres Fortran, qu'ils soient dans la traduction C transmis par référence ou gérés dans les tables de variables communes, sont signalées pour permettre de localiser d'éventuels effets de bord (2.3.5).

Variables lues mises à jour ou réinitialisées Les variables lues puis mises à jour (2.4.3) sont répertoriées dans deux listes.

1. La première, produite par examen du graphe, cite le sous-programme de lecture et l'unité de programme dans laquelle un arc localise une mise à jour éventuelle.
2. La seconde, construite à partir du dictionnaire, cite aussi ce sous-programme et localise le rang de l'ultime affectation définie avec un nombre ou une expression locale en opérande droite, opération non représentée dans le graphe : les paramètres remplacés par une valeur optimisée dans LECOPTIM ou permutés dans DOMINVER sont exclus.

La classe finale est indiquée entre parenthèses.

Compte-rendu détaillé

Le compte-rendu détaillé regroupe pour chaque fonction tous les messages d'avertissements ou d'informations qui repèrent des situations ambiguës ou atypiques. Ils sont identifiés par le rang de l'instruction source, par défaut la

[84] HNAPPE, HMAX, HPB, PROFNAPPE pour le sous-programme NAPPE, RATM pour CALRNET, RDIF pour RAYTRANS et RAS pour CALRAERO.

[85] L'appel de CALRAERO depuis SHUTWALL peut mettre à jour le paramètre RAS.

première instruction de l'unité de programme, suivi d'un code d'erreur entre crochets : une classification des erreurs est effectuée pour les resituer dans les séquences de traitement.[86]

Supervision du traitement $[1, 99]$

$[1, 9]$: initialisation ;
$[10, 19]$: lecture du code normalisé ;
$[20, 29]$: portée des variables communes ;
$[30, 39]$: déclaration des variables locales ;
$[40, 49]$: déclaration des paramètres formels ;

Analyse des instructions $[100, 999]$

$[100, 199]$: reconnaissance de l'instruction ;
$[200, 299]$: déclaration de variables locales ;
$[300, 399]$: condition logique ;
$[400, 499]$: affectation ;
$[500, 599]$: appel de fonction ;
$[600, 699]$: liste de lecture ;
$[700, 799]$: liste d'édition ;
$[800, 899]$: construction des prototypes des fonctions ;

Contrôles et classifications

$[1000, 1099]$: cohérence déclarations et appels des fonctions ;
$[1100, 1199]$: graphe des relations entre variables ;
$[1200, 1299]$: classification des variables ;
$[1300, 1399]$: copie locale de variables communes ;
$[1400, 1499]$: rôles des fonctions ;

Ces codes sont retransmis depuis les fonctions où ils sont définis vers les fonctions appelées, ce qui permet d'identifier les traitements en cours.[87]

Cultures associées

Les formes décrites dans "`stics.couple`" permettent de localiser les instructions de gestion des couples *(dominant/dominé), (ensoleillé/ombragé)*. Elles sont éditées avec les noms des unités de programmes et les numéros des instructions pour pouvoir être reprises dans l'étape de génération de code, les variables de gestion étant citées comme suit :

$< unite > < rang >$ IF$(< variable > \{, < variable >\})$
$< unite > < rang >$ DO$(< variable > \{, < variable >\})$
$< unite > < rang > < variable >=$

[86] Une valeur négative code une erreur fatale souvent liée à des problèmes de fichiers mais qui peut aussi nécessiter de modifier une constante, voire compléter un traitement local.

[87] La fonction de reconnaissance des indices est par exemple appelée pour traiter tant les expressions arithmétiques et logiques que les listes de paramètres et d'entrées-sorties.

2.7.9 Révision Fortran

Des fonctionnalités supplémentaires ont été ajoutées pour localiser les déclarations et instructions dans les modules sources Fortran.

L'appel de *S6* avec l'option '-F' produit des fichiers supplémentaires utilisables pour effectuer des corrections du code Fortran : les avertissements associés sont, avec cette option, supprimés du compte-rendu détaillé.

Chapitre 3

Génération des fonctions

Ce chapitre décrit le processus de construction des fonctions C à partir des unités de programmes Fortran.

Les descripteurs, produits dans les étapes antérieures sont lus dans les différents fichiers et rangés dans des tables. Le code source est traité instruction par instruction pour chaque unité de programme. Un code C compilable, restructuré, est généré avec le rang de l'instruction Fortran d'origine en commentaire. Les blocs d' instructions associés aux structures de contrôles sont repérés par des indentations, les rangs des instructions initiales et finales étant indiqués en commentaires lors de la fermeture de ces blocs.

Des avertissements sont édités pour localiser les séquences à examiner, pour contrôle et/ou correction du code C, ou révision du code Fortran.

Les versions Fortran de STICS utilisent quelque deux milliers de variables communes, simples ou tabulaires, numériques ou textuelles, consultables et modifiables dans tous les sous-programmes et dans les fonctions. Les variables numériques gèrent des nombres, vecteurs et matrices. Les variables textuelles sont des chaînes de caractères ou des tables de codes.

Dans les versions C, ces variables sont vectorisées et gérées dans des tables, déclarées dans la fonction de supervision et transmises comme paramètres des fonctions informatiques.[1]

1. Une forme simple a d'abord été définie pour une première restitution de la dynamique du modèle : sa mise en œuvre pour la version 5 a débouché sur une version opérationnelle.
2. La nécessité de traduire les propriétés du modèle et de traiter les changements d'échelles a conduit à définir des formes plus complexes, qui intègrent finalement le modèle de culture dans un modèle générique, dit métamodèle : sa réalisation a débouché sur une autre version opérationnelle pour la version 6.

Les traitements informatiques effectués sont utilisables pour restructurer le code Fortran, et faire en sorte qu'il intègre certaines propriétés du modèle, pour

[1] Les tables de la version C provisoire produite pour le projet HiSAFE sont globales.

pouvoir ensuite améliorer la vectorisation.

3.1 Gestion des variables communes : forme simple

Cette section décrit le mode d'allocation initialement utilisé pour produire immédiatement des versions opérationnelles provisoires.

3.1.1 Paramètres informatiques communs

Un ensemble de variables étant défini, le paramétrage est adaptable en fonction d'objectifs applicatifs.

Celui proposé pour cette traduction, qui reprend la dynamique du modèle (2.4), est relativement général, avec un nombre minimal de paramètres : il est piloté par des primitives d'allocation et de restitution regroupées dans un module spécifique, ``Sallouer.c''.

Variables STICS

Les variables communes STICS, identifiées par des noms, sont numériques, logiques ou textuelles : elles sont gérées dans la traduction dans un nombre restreint de tables, les noms propres étant restitués en commentaire.

- Les variables numériques, entières ou réelles,simples, vectorielles ou matricielles, ainsi que les variables logiques, sont gérées dans des tables réelles.
- Les variables textuelles (1.2.2) sont réparties en deux groupes :
  - les mots, textes courts d'au plus 32 caractères regroupés dans une table ;
  - les messages, textes longs d'au plus 256 caractères,[2] utilisés exclusivement pour des éditions, traités dans une fonction spécifique (3.8.3).

Vectorisation

La répartition dans les tables s'effectue en fonction de la dynamique du modèle (2.4.1).

1. Les variables sont réparties en deux groupes.

   (a) Les variables statiques citées exclusivement dans les fonctions d'initialisation ou d'édition, dites d'**initialisation**, sont gérées dans :
   - la table *Code* pour les textes courts ;
   - le vecteur $X0$ pour les nombres ;

   (b) Les variables statiques consultées dans les fonctions qui effectuent les évaluations quotidiennes, dites de **simulation** (2.5.1), sont regroupées dans le vecteur X ;
   les variables textuelles sont remplacées par leurs images numériques (2.1.5).

[2] Cette borne est dépassée pour la déclaration des variables MES376 et MES430 dans la version 6 mais les valeurs textuelles gérées restent inférieures à cette limite.

2. Les variables évaluées quotidiennement, numériques, sont vectorisées dans Y : les séries temporelles sont reprises intégralement.[3]
3. Les variables numériques d'édition sont regroupées dans le vecteur Z.

Avec ces conventions, une évaluation quotidienne est traduite au moyen d'une fonction Φ sous la forme

$Y_t = \Phi(X, Y_{t-1})$

Y_0 étant l'état initial de la culture et X un vecteur des variables exogènes.[4]

Horloge calendaire

La dimension quotidienne est gérée dans le code Fortran dans la variable globale N, horloge calendaire qui gère le rang des simulations quotidiennes. Elle est remplacée dans la version 5 par le paramètre *hui*, transmis à toutes les fonctions appelées depuis la boucle quotidienne et leurs descendances.

Fichier d'audit

Une référence vers un fichier d'audit est systématiquement transmise pour pouvoir éditer des messages, dont ceux du fichier dit "historique".

3.1.2 Allocation et restitution des variables communes

Les crochets [] sont introduits dans les notations informatiques pour pouvoir désigner les tables et accéder aux éléments. Les variables informatiques communes, variables effectives du modèle de simulation, sont alors gérées dans le quintuplet

$$(Code[], X0[], X[], Y[], Z[]).$$

$Code[]$ est une table de codes textuels (char *), alors que $X0[], X[], Y[]$ et $Z[]$ sont des tables de nombres réels (float), un nombre entier étant représenté sous la forme réelle.

Allocation d'un espace dans une table

Un espace est alloué dans une table pour chaque valeur d'une variable commune pour une culture pure, soit la plante principale et la zone ensoleillées : des contrôles sont à effectuer afin de vérifier que les valeurs de la plante associée ou de la zone ombragée ne sont pas référencées (2.6).[5]

Pour les vecteurs, matrices et tables de codes textuels, les éléments de rang 0 sont toujours alloués afin de conserver les valeurs d'indices.[6]

[3] Le passage d'une période bisannuelle à annuelle et la réduction de la profondeur du sol à 200 centimètres a permis de réduire la dimension de l'espace vectoriel généré $\mathbb{R}^k$ à $k = 166\,990$ pour les besoins du projet HiSAFE.

[4] Y contient les variables endogènes mais aussi les exogènes initialisées quotidiennement.

[5] Pour les cultures associées, il faudra allouer deux, voire quatre espaces, pour les variables concernées.

[6] Cet élément n'est présent en Fortran que si la borne inférieure 0 est déclarée.

Valeurs numériques

1. Une variable simple occupe une position.
2. Pour un vecteur à n composantes, $n+1$ éléments sont réservés.
3. Pour une matrice à n lignes et m colonnes, $(n+1)(m+1)$ éléments sont réservés : le terme m_{ij} est rangé en position $(n(j+1)+i)$.

Un espace supplémentaire dans la table $X[]$ est à réserver pour les variables textuelles numérisées.

Valeurs textuelles Une valeur textuelle référencée lors des évaluations apparaît comme un code utilisable dans des expressions logiques, initialisé dans une instruction d'affectation ou de lecture. Ces codes textuels sont empilés dans une table : un élément occupe une position et n éléments sont réservés pour une table Fortran de n codes.

Gestion des messages

Pour un texte long, pratiquement un message à éditer, un numéro d'ordre est attribué pour être repris comme valeur de la variable de choix de la fonction d'édition des messages (3.8.3).

Primitives informatiques d'allocation et restitution

Le système d'allocation est géré à travers des primitives d'allocation et de restitution des références , qui utilisent les informations contenues dans le dictionnaire des variables communes produit dans l'étape précédente :

- *Allouer()* rend le numéro du tampon et le nombre d'éléments réservé pour coder une valeur ;
- *Restituer()* produit la référence vers la variable citée ;

Ces deux fonctions d'allocation et restitution sont indissociables, l'allocation s'effectuant lors de la lecture du dictionnaire, et la restitution à la demande.

Pour les cultures associées, il faudra en fait réserver autant de valeurs que d'occurrences possibles.

Noms STICS et classes Au cours de la génération du code, les noms des variables communes sont introduits en commentaires de l'instruction. Ceux des variables statiques sont préfixés du chiffre des dizaines de la classe initiale, pour pouvoir distinguer paramètres de la culture et constantes. Les crochets sont ajoutés pour les tables numériques ou regroupant des codes textuels.

Modification dynamique de la taille des tables Afin de pouvoir générer des versions adaptées à différents contextes d'utilisation, un mécanisme est défini pour pouvoir remplacer les bornes supérieures des tables numériques par les constantes prédéfinies de la section 2.1.4.

Les substitutions sont faites pour la version 5 et les bornes candidates au remplacement sont seulement signalées pour la version 6 : un commentaire est

ajouté pour ces bornes dans le fichier qui répertorie les constantes numériques citées en clair dans les instructions (3.9.1).[7]

3.1.3 Validation des versions

Une version logicielle C est validée relativement à la version Fortran traduite au moyen de procédures de comparaison de toutes les évaluations des variables numériques communes du début à la fin de la simulation, à un seuil de précision donné. Ces comparaisons s'effectuent :

1. après acquisition des paramètres et initialisation pour les variables statiques ;
2. avant ou après les évaluations du jour pour les variables temporelles ;
3. accessoirement en fin de simulation pour les variables d'édition ;

Le processus de validation se déroule en deux étapes.

1. Les fonctions et sous-programmes de suivi, générés, sont appelés respectivement en C et Fortran à des stades d'exécution des algorithmes donnés : toute valeur non nulle d'une variable commune est éditée dans un fichier avec l'identifiant de cette variable et les indices pour les vecteurs et matrices.
2. Une commande, fabriquée, est utilisée après exécution des versions C et Fortran pour comparer les contenus des deux fichiers produits au seuil de précision choisi (A.4.3).[8]

Fonctions et sous-programmes de suivi (version 5)

Le code des fonctions C est généré après allocation dans les tampons et les sous-programmes Fortran équivalents reprennent les noms des variables communes. Les fonctions de génération , définies pour une stratégie d'allocation, sont déclarées dans le module `"Sallouer.c"`.

Le code généré est immédiatement compilable, sous réserve toutefois d'accéder pour la version Fortran aux fichiers d'inclusion qui contiennent les déclarations des variables communes. Son exécution se traduit par l'édition dans des fichiers, sur une ligne et avec des espaces séparateurs,

- du nom de la variable,
- du ou des indices des tables,
- de la valeur, entière ou réelle, avec une précision d'au plus six décimales pour les nombres réels.

[7] Les corrections requises sont à effectuer dans le code Fortran, ce qui permet ensuite de retrouver les variables *Stics* équivalentes dans le code C.

[8] Les erreurs liées à la précision ne sont pas nécessairement traitées de la même façon par les compilateurs C et Fortran et dans les fonctions des bibliothèques numériques respectives. Le fait d'adopter la simple en double précision peut modifier l'exécution d'une séquence contrôlée par un prédicat : les erreurs peuvent ensuite se propager, jusqu'à aboutir en fin de simulation à des écarts qui deviennent perceptibles.

Variables statiques Les fonctions $SuiviX0()$ et $SuiviX()$, respectivement les sous-programmes SUIVIX0 et SUIVIX, éditent les valeurs des variables numériques communes statiques initiales et temporelles : ils permettent de restituer les états complets après acquisition des paramètres et initialisation.

Variables dynamiques La fonction $SuiviY()$ et le sous-programme SUIVIY, éditent pour un jour donné transmis en paramètre, les valeurs des variables dynamiques : ils permettent de restituer les états du jour j ;
- avant simulation si $(j < 0)$;[9]
- après simulation si $(j > 0)$;

Variables d'édition La fonction $SuiviZ()$ et le sous-programme SUIVIZ, restituent les seules variables d'édition des résultats en fin de simulation.

Accès aux variables communes

Des fonctions C sont générées au cours de cette allocation pour fabriquer la commande $Vstics$ (A.4.3) de comparaison des fichiers de suivi issus de l'exécution simultanée des versions C et Fortran pour la version 5 :
- $SticsX0()$ et $SticsX()$ pour les variables statiques ;
- $SticsY()$ pour les variables dynamiques ;
- $SticsZ()$ pour les variables d'édition ;

3.2 Gestion des variables communes : métamodèle

Les versions C précédentes ont été produites pour répondre à des besoins immédiats et le paramétrage est insuffisant par rapport aux objectifs assignés, en l'occurrence traduire les propriétés du modèle et intégrer des changements d'échelles : un métamodèle a été défini en conséquence.

La version 6 a par ailleurs
- introduit des structures de données nouvelles pour gérer des couples de cultures associées,
- rétablit la périodicité bisannuelle et l'ancienne profondeur du sol.[10]

Ce dernier point met en évidence la nécessité de gérer au mieux les structures de données pour pouvoir exécuter simultanément de nombreuses simulations dans différents espaces, et accessoirement ajuster la taille de ces espaces aux applications (simuler une culture avec précision ou évaluer globalement des effets environnementaux).

[9] $j = -1$ restitue l'état initial des variables temporelles au premier jour, dont les variables endogènes lues (2.4.3).

[10] L'espace vectoriel $\mathbb{R}^k$ (note 3 section 3.1.1) est alors de dimension $k = 1\,516\,284$.

3.2.1 Cultures associées

La version 6 associe à un grand nombre de variables STICS des couples de valeurs pour pouvoir gérer deux cultures associées, soit

1. la plante principale et la plante associée ;
2. la zone d'ensoleillée et la zone ombragée ;

Ces couples sont définis pour toutes les variables descriptives de la plante et du système sol-plante, ainsi que pour des opérations techniques concernées : le modèle à deux cultures effectue donc deux évaluations, voire quatre pour les parties aériennes du fait de l'ensoleillement.

Inversion de dominance La simulation démarre avec la plante principale dominante et la plante associée dominée mais cette hiérarchie peut être inversée sur la période de simulation : des permutations sont alors effectuées dans les tables.[11] Cette inversion introduit une rupture dans les séries chronologiques, une variable n'étant plus nécessairement définie pour la même plante pendant la simulation (voir note 31 section 4.2.7).

Gestion des couples principal/associé, soleil/ombre

Les valeurs sont gérées dans des tables Fortran par ajout de une ou deux dimensions binaires.[12]

Déclarations Les bornes supérieures déclarées *bsup* des tables informatiques sont examinées pour retrouver les variables, soit,

- $bsup = 2$ pour gérer les couples d'occurrences, ou instances ;
- $bsup > 2$ pour les dimensions des espaces vectoriels et le nombre de codes textuels ;[13]

Les évaluations s'effectuent avec ces variables informatiques qui gèrent finalement deux ou quatre occurrences, que la plante associée soit présente ou non, repérées par un nombre restreint d'indices :

- les valeurs 1 ou 2 ;
- les pseudo-constantes AS $= 1$ ou AO $= 2$;
- les variables communes IPL et ENS, et l'expression IPL $+ 1$;
- une variable de boucle locale avec la borne supérieure NBPLANTES ;

[11] Le sous-programme DOMINVER gère ainsi des transferts pour plus de cinq cents variables, et rend finalement inopérant l'algorithme de construction du graphe des relations entre ces variables (3.2.1).

[12] Les tables Fortran de la version 6 ont finalement au plus trois dimensions.

[13] L'examen des indices effectué par la commande **S6** (2.6) a permis de vérifier l'absence de variables STICS qui seraient des vecteurs à deux composantes ou des matrices à deux lignes ou deux colonnes pour une borne inférieure unitaire.

Restitution du modèle pour culture simple

Un moyen de restituer le modèle pour une culture simple est de supprimer les dimensions supplémentaires introduites pour gérer les couples de valeurs : le modèle de culture simple est ainsi reconstitué avec les variables définies pour la plante seule. Les variables de gestion des couples, IPL, ENS et NBPLANTES, contiennent la valeur 1 et les instructions d'évaluation relatives à la culture associée ou la zone ombragée sont à supprimer. Ces instructions sont répertoriées dans l'étape précédente (2.6.2) et signalées dans le fichier "`stics.ass`" (2.7).

Ces opérations étant faites, il est apparu que quelque quatre cents évaluations de termes d'expressions arithmétiques ou logiques font référence pour une culture simple aux valeurs alors non définies pour la culture associée ou la zone ombragée : ces termes sont ignorés ou neutralisés lors de la génération du code.[14]

Suppression des évaluations relatives à la culture associée ou la zone ombragée Les instructions de contrôle relatives à la gestion des cultures associées sont examinées.[15] Si la valeur du prédicat est déterminée, les instructions et blocs d'instructions qui s'exécutent pour la culture associée sont supprimés : une instruction vide est ajoutée si nécessaire, suivie du code fortran en commentaire, pour ne pas générer d'erreurs syntaxiques dans les formes conditionnelles. Dans le cas contraire, un avertissement est édité.

Neutralisation des références à la culture associée ou la zone ombragée Si une expression contient une référence au second élément des couples, la suppression de l'indice du couple aboutirait à référencer de fait la culture principale ou la zone ensoleillée.[16]

- **Expressions arithmétiques** Le premier terme est annulé et les autres sont supprimés s'ils suivent immédiatement un signe arithmétique.
- **Conditions logiques** Les termes sont annulés et un avertissement est édité.[17]

Des références aux cultures associées ou zones ombragées apparaissent aussi dans les listes de lecture des paramètres "nouvelle version" (4.3.4) : elles sont localisées et traitées à part au moyen de fonctions fabriquées pour traiter ces cas particuliers.[18]

[14] Le problème est à résoudre en amont, ces références, cumuls ou conjonctions, devant être contrôlées ou supprimées : la liste de ces quelque quatre cents termes est éditée dans "`stics.ombre`".

[15] Les opérations logiques élémentaires peuvent être évaluées statiquement au moyen de règles, la second opérande ayant une valeur fixe et les opérations n'étant connectées que par l'opérateur $\wedge$.

[16] Des instructions cumulent les valeurs pour les deux zones AS et AO dans une boucle dpuis 1 jusqu'à NBPLANTES (APPORTS).

[17] Ces séquences ne sont pas supprimées comme précédemment en l'absence du prédicat de contrôle de la présence de la seconde culture : on aboutit à des structures de la forme *Si* $(0 \leq 0)$ *alors* $\{< vide >\}$.

[18] $VoirParaV6()$ recherche dans la liste de lecture le motif "(2)".

3.2.2 Métamodèle

La forme $Y_t = \Phi(X, Y_{t-1})$ ne restitue pas la dynamique

1. quotidiennement dans le modèle de culture puisque

   Y_t contient toutes les variables temporelles, qu'elles soient
   - évaluées par le modèle, *endogènes*,
   - ou liées à des faits ou processus externes, *exogènes* ;

   X regroupe les variables statiques, *exogènes* ;

2. aux échelles spatiale et sur le long terme.

Métamodèle Soit une campagne culturale j sur une parcelle i.

1. **Les variables statiques**, invariantes pendant la simulation d'une culture, sont affectées dans les vecteurs :
   - pour la dimension spatiale :
     S_i descripteur du **S**ol de la parcelle ;
   - pour la campagne culturale :
     V_j descripteur du **V**égétal cultivé ;
     M_j séries climatiques ;
     W_j itinéraire technique prévu ;
2. **Les variables dynamiques**
   - $\mathcal{Q}_{[i,j,t]}$ suite des états de la culture évalués **Q**uotidiennement.

La dynamique du métamodèle peut alors s'exprimer sous la forme

$\mathcal{Q}_{[i,j,t]} = \Phi(S_i, V_j, M_j, W_j, \mathcal{Q}_{[i,j,t-1]})$,
avec $\mathcal{Q}_{[i,j,0]}$ état initial, état final $\mathcal{Q}_{[i,j-1,n]}$ du dernier jour n de la campagne précédente $j-1$.

Les variables endogènes sont notées en majuscules calligraphiques.

La fonction Φ est une composition de fonctions $\varphi_1 \circ \cdots \circ \varphi_i \circ \cdots \circ \varphi_n$.

Variables exogènes Les vecteurs S_i, V_j, M_j, W_j, sont constitués de valeurs de variables simples et construits comme suit.

Sol Les paramètres sont définis globalement pour la **P**arcelle puis pour k **H**orizons, soit $S = (P, H^k)$.

Plante Les paramètres sont définis pour une espèce **C**ultivée et un **G**roupe variétal, soit $V = (C, G)$.

Climat Les données météorologiques sont des séries annuelles, et affectées dans M_j.

Itinéraire technique Les actions techniques sont repérées par des rangs, identifiées par une date t et décrites par un vecteur de variables descriptives, et à affectées dans W_j.

Représentation spatiale Le métamodèle dynamique peut être enrichi pour représenter les processus spatiaux en décomposant l'état quotidien en quatre blocs, soit

$\mathcal{A}_{[i,j,t]}$, parties **A**ériennes ;
$\mathcal{R}_{[i,j,t]}$, profils par couches de la **R**hizosphère ;
$\mathcal{O}_{[i,j,t]}$, profils pour la décomposition de la matière **O**rganique dans les couches de la zone humifère ;
$\mathcal{H}_{[i,j,t]}$, profils des **H**orizons ;

soit finalement le quadruplet

$$\mathcal{Q}_{[i,j,t]} = (\mathcal{A}_{[i,j,t]}, \mathcal{R}_{[i,j,t]}, \mathcal{O}_{[i,j,t]}, \mathcal{H}_{[i,j,t]}).$$

Cultures associées La dynamique du métamodèle pour deux cultures associées peut être traduite au moyen d' une fonction Ψ sous la forme

$$\mathcal{Q}_{[i,j,1,t]} = \Psi(S_i, V_{[j,1]}, M_j, W_j, \mathcal{Q}_{[i,j,1,t-1]}, \mathcal{Q}_{[i,j,2,t-1]})$$
$$\mathcal{Q}_{[i,j,2,t]} = \Psi(S_i, V_{[j,2]}, M_j, W_j, \mathcal{Q}_{[i,j,1,t]}, \mathcal{Q}_{[i,j,2,t-1]})$$

La généralisation à un ordre $k > 2$ est envisageable.

3.2.3 Vectorisation : partition des variables

L'objectif est alors de trouver un algorithme pour gérer les milliers de variables communes STICS[19] et restituer le métamodèle. La solution consiste à répartir ces variables dans des tables informatiques qui représentent les vecteurs du métamodèle, et intègrent la dynamique de la simulation avec

- deux dimensions temporelles, quotidienne et annuelle,
- une dimension spatiale pour gérer un territoire,

auxquelles s'ajoutent une table instrumentale pour gérer les constantes utilisées dans les évaluations.

Gestion de la dimension chronologique quotidienne

Les évaluations du jour sont faites dans

1. quelque 650 variables, simples, vectorielles ou matricielles, la valeur du jour remplaçant celle de la veille ;
2. environ 60 séries chronologiques déclarées dans des tables sur des périodes bisannuelle (731 jours) ;[20]
3. deux matrices pour le système racinaire définies jusqu'à la profondeur 1000 pour une période bisannuelle ;

[19] Le nombre de variables communes est réduit de un quart dans la version 6.
[20] Il existe une seule série annuelle (366 jours), JULOFFN lue dans LECEPITAF (version 6).

Séries quotidiennes Les séries actuelles sont des variables

- **exogènes**, qui gèrent des observations climatiques et des opérations techniques quotidiennes prédéfinies, paramètres de la simulation ;
- **endogènes**, évaluées quotidiennement par le modèle en fonction
  - de la période précédente dans la plupart des cas ;
  - des périodes antérieures quelquefois, avec une recherche des éléments les plus anciens ;

elles sont gérées dans des variables informatiques selon plusieurs modalités.

1. La dimension temporelle est ignorée : les valeurs du jour— nombres, vecteurs ou matrices— remplacent celles de la veille ;
2. Les variables sont déclarées sur une période bisannuelle (731 jours), quelquefois annuelle (366 jours), mais avec deux bases de temps :
   (a) calendaire pour les observations climatiques, la période i étant le rang du jour dans l'année ;
   (b) séquentielle pour les séries évaluées quotidiennement, la période i étant le rang de la simulation ;
3. Les variables sont des tables qui gèrent des suites chronologiques d'opérations culturales— irrigation, fertilisation ou récolte— les éléments de rang i étant associés à une date gérée dans l'une des variables descriptive de l'opération.
4. Les évènements uniques, comme l'opération semis ou plantation, sont gérés dans des variables simples.
   La valeur 999 pour les dates des stades de développement signifie que la date est à évaluer, toute autre valeur étant une date de forçage.

Vectorisation Les données dynamiques sont finalement vectorisées dans la table $Y_[]$, laquelle gère les états $\mathcal{Q}_{[i,j,t]}$. Des composantes fictives sont par ailleurs ajoutées pour traiter d'éventuels références hors période (3.10.3).[21]

Gestion des campagnes culturales

La campagne culturale définie une autre périodicité, annuelle mais avec des dates sur deux années calendaires.

- Le vecteur V_j, qui décrit la plante cultivée, est représenté dans la variable $V_[]$: il regroupe exclusivement des variables simples ;
- Les vecteurs M_j et W_j, définis en début de campagne pour toute la période de simulation de la culture, sont gérés dans les variables $M_[]$ et $W_[]$: il s'agit des opérations techniques réalisées aux dates prévues et les données météorologiques quotidiennes ;
  Les données de forçage, également définies pour une date, sont gérées avec les opérations techniques ;

[21] Dans le code Fortran, les initialisations s'effectuent souvent sur une période bisannuelle et la composante de rang 0 est quelquefois utilisée.

– Les autres données, regroupées dans la variable $X_[]$, décrivent pour l'essentiel un état initial $\mathcal{Q}_{[i,j,0]}$, décrit par des paramètres initiaux ou par lecture des données de la culture précédente.

Rotation des cultures L'état final d'une culture pour les horizons et les couches travaillées du sol est édité dans un fichier et relu pour la culture suivante avec la valeur 2 de l'option CODEINITPREC (4.3.2, 4.3.9). Deux fonctions sont générées à partir des sous-programmes d'édition et lecture pour gérer ces listes de valeurs de variables numériques communes, variables simples, vecteurs et matrices, dans un vecteur transmis en paramètre : elles rendent le nombre de valeurs.

– *CultSuiv()* affecte les valeurs des variables de la liste éditée dans ce vecteur.
– *CultPrec()* restitue ces valeurs dans les variables de la liste lue.

La première ligne du fichier précédent contient des informations de contrôle pour gérer l'enchaînement des simulations et les périodes calendaires, ainsi qu'un nombre d'horizons : elle n'est pas reprise dans la vectorisation.

Gestion de l'espace

L'espace géographique est vu comme une partition en parcelles homogènes sur lesquelles les cultures se succèdent d'une campagne à l'autre, avec éventuellement des cultures intermédiaires ou l'absence de culture.

Le vecteur S_i, qui décrit la parcelle, est représenté dans la variable $S_[]$: il est construit avec

– des variables simples qui décrivent globalement le sol,
– des vecteurs de $\mathbb{R}^6$, la composante 0 étant relative à la surface et les autres aux 5 horizons.

Les modifications de l'état du sol liées aux opérations techniques d'une campagne culturale relèvent de l'état $\mathcal{Q}_{[i,j,0]}$ et sont à gérer dans la variable $X_[]$.[22]

Constantes et données générales

Un certain nombre de variables STICS apparaissent comme indépendantes des simulations. Elles sont lues pour la plupart avec les paramètres généraux, quelques constantes climatiques étant lues avec les données météorologiques (2.4.3). Toutes ces variables sont gérées dans la table $K_[]$, qui regroupe :

– des variables simples, grandeurs physiques ou valeurs choisies pour un lieu et une période, pratiquement une latitude pour la période actuelle ;
– des tables associées à des typologies de composants du sol ou de produits incorporés ;

[22] La version 6 pose problème du fait de la relecture optionnelle des paramètres du sol avec l'état de la culture précédente pour prendre en compte les modifications de structures consécutives au tassement (4.3.9).

Pilotage de la vectorisation

La distinction entre variables exogènes et endogènes ne peut pas être faite dans le code informatique actuel pour les séries chronologiques mises à jour quotidiennement avant utilisation (2.4.1) : toutes ces variables sont donc gérées dans la tablesa $Y_[]$. Il suffit de déplacer ces mises à jour de séries exogènes dans des unités d'initialisation pour qu'elles soient vectorisées dans la classe dont elles relèvent (2.4.4).

3.2.4 Vectorisations multiples

Plusieurs variables lues sont ensuite réévaluées dans l'étape d'initialisation ou lors des évaluations quotidiennes. Un moyen de conserver les valeurs lues est d'effectuer une affectation lors de la lecture (3.8.4) simultanément dans les vecteurs :

- associés au rôle courant de l'unité de lecture ($S_, V_, W_$) ;
- $Y_$ pour les évaluations quotidiennes et/ou $X_$ pour les données initiales ;

Les phénomènes de tassement du sol lors des interventions culturales sont souvent représentés dans des variables informatiques qui contiennent successivement des valeurs différentes :

1. lues avec les paramètres du sol (PROFHUM,Q0...) ;
2. relues avec la culture précédente pour tenir compte du tassement par les engins de récolte ;
3. modifiées dans les évaluations quotidiennes du fait d'opérations techniques ;

Pour retrouver paramètres et variables,[23]

- une triple affectation est effectuée ($Y_, X_, S_$) pour la lecture des paramètres ;
- les valeurs lues de la culture précédente sont affectées dans ($Y_, X_$) ;
- $Y_$ est utilisé lors des évaluations quotidiennes ;

Réinitialisation Une fonction de réinitialisation de ces variables, *IniY()*, est générée pour pouvoir retrouver les valeurs lues des paramètres ou données initiales dans des applications (3.9.1).

3.2.5 Vectorisation : ordonnancement des composantes

Les tables précédentes sont construites par allocation d'espace pour les variables STICS successives : l'espace alloué peut être organisé pour structurer cet ensemble de variables, et définir des références vers les autres vecteurs cités dans le métamodèle.

[23] Il aurait été nécessaire de se donner plusieurs variables informatiques pour distinguer paramètres et variables évaluées à l'issue d'opérations exogènes, et conserver les valeurs intiales des cultures suivantes.

Représentation du sol et des états du système sol-plante

Les tailles des tables déclarées dans le code informatique permettent d'identifier les variables descriptives :

- des horizons $\mathcal{H}_{[i,j,t]}$, vecteurs de $\mathbb{R}^6$;
- de la zone humifère $\mathcal{O}_{[i,j,t]}$ vecteurs de $\mathbb{R}^{100}$ dans la version 6, $\mathbb{R}^{60}$ antérieurement ;[24]
- du système sol-plante $\mathcal{R}_{[i,j,t]}$, vecteurs de $\mathbb{R}^{1000}$;

Le mécanisme d'allocation utilise ces tailles pour ranger dans les vecteurs les variables simples d'abord, puis consécutivement les vecteurs pour chacun des espaces vectoriels.

Représentation du sol Le métamodèle décrivant le sol S comme un couple (P, H^k) avec $k = 5$ horizons, il est possible de définir deux variables :

1. $P[]$, vecteur des variables simples :
2. $H[][6]$, matrice qui regroupe les variables descriptives des horizons et de l'état de surface ;

Il suffit ensuite de définir les références vers les adresses d'allocation correspondantes pour accéder directement à ces deux structures.

Zone humifère L'état du sol dans les couches d'incorporation des résidus est représenté dans les états $\mathcal{Q}_{[i,j,t]}$. L'état initial ($t = 0$) est géré dans $X_[]$ et les états quotidiens dans $Y_[]$, la dimension chronologique étant perdue. Les matrices correspondantes peuvent alors être référencées comme précédemment.

Système sol-plante Les vecteurs descriptifs relèvent de l'état quotidien $\mathcal{Q}_{[i,j,t]}$ et les matrices qui gèrent la dimension chronologique sont gérés dans $Y_[]$.

Accès au groupe variétal

Les paramètre de la plante sont définis pour l'espèce cultivée C et le groupe variétal G qui regroupe un nombre restreint de variables descriptives : la séparation de ces deux ensembles va permettre de définir deux variables informatiques :

1. $C[]$, vecteur descriptif d'une culture ;
2. $G[]$, vecteur associé au choix variétal, qui peut varier pour une même culture ;

Données initiales

Les données initiales lues avec les paramètres initiaux sont regroupées, avec d'abord les valeurs simples puis celles fournies pour les horizons, ceci pour pouvoir construire des matrices descriptives de ces horizons.

[24] La profondeur de la zone humifère est néanmoins limitée à la valeur 60, un franchissement après lecture du paramètre du sol PROFHUM se soldant par un déroutement avec arrêt d'exécution (4.3.8).

Les autres données vectorisées dans $X_[]$ sont rangées au début du vecteur, ceci pour conserver la structure du vecteur du sol $S_[]$.[25]

3.2.6 Traduction informatique

Les vecteurs ainsi alloués reprennent à l'identique les variables STICS, et les valeurs de la période t se substituent à celles de la période $t-1$.

Les variables sont rangées à partir de la position 1, la composante de rang 0 restant réservée, par exemple pour récupérer des erreurs.

Gestion de la dimension chronologique

Plusieurs périodes sont définies

1. **Campagne culturale** La constante CAMPAGNE définit la durée maximale d'une campagne culturale, pratiquement 366 jours, qui peut s'étendre sur deux années calendaires.
2. **Délai de mémorisation** Le métamodèle prévoit de conserver la dimension chronologique pour empiler les états et pouvoir défaire et refaire les simulations, avec la constante DELAI.
3. **Longévité**

Horloge calendaire Le jour simulé est toujours repéré par la variable commune N mais pour la version 6 dans une boucle Fortran DO WHILE.[26] Elle est remplacée dans le code C par la variable $t_$ transmise en paramètre des fonctions d'évaluation.

Reprise de l'existant

Les variables non utilisées dans les évaluations sont regroupées dans des tables particulières.

1. **Codes non numériques**
   Bien que les variables textuelles soient immédiatement numérisées (2.1.5), il faut néanmoins les lire, d'ou la présence de la table de codes textuels $Code[][]$, transmise aux fonctions de lecture et d'édition.[27]
2. **Variables d'édition**
   Il existe aussi un ensemble de variables numériques d'édition (2.4.4) non utilisées au cours des évaluations : elles sont regroupées dans la table $Z_[]$, transmise aux fonctions de lecture, initialisation et édition.

[25] Paramètres du sol et variables descriptives de l'état du sol à l'issue d'interventions techniques sont, dans la version 6, gérées dans les mêmes variables informatiques Fortran.

[26] Cette structure est reconnue par certains compilateurs Fortran mais ne fait pas partie de la norme FORTRAN77.

[27] Ce paramètre $Code[][]$ n'est introduit que s'il existe des variables textuelles dans la version Fortran.

Variables indéterminées

Les variables, communes ou locales, qui ne reçoivent aucune valeur, indéterminées, ne sont pas reprises dans les dictionnaires : les références sont remplacées par des appels de la fonction entière $Zombi()$, avec les noms de variables en paramètres.

Représentation des variables communes

Toutes les variables STICS communes sont finalement gérées dans le 10_uplet

$$(Code[][], K[], S[], V[], M[], W[], X[], Y[][], U[], Z[]),$$

chaque table gérant les valeurs suivantes :

0. $Code[][]$, codes textuels, les images numériques (2.1.5) étant gérées dans les vecteurs ;
1. $K[]$, constantes ;
2. $S[]$, variables descriptives du sol ;
3. $V[]$, variables descriptives de la plante cultivée ;
4. $M[]$, séries climatiques ;
5. $W[]$,itinéraire technique et forçage ;
6. $X[]$, état initial de la culture ;
7. $Y[][\text{CAMPAGNE}]$, table des séries déclarées évaluées quotidiennement ;[28]
8. $U[]$, autres variables évaluées quotidiennement ;
9. $Z[]$, variables d'édition ($classe \geq 200$) ;
10. $Q[\text{DELAI}][]$ récapitule les valeurs pour la durée k, la période t se substituant à $t - k$;

Traduction informatique Les traitements relatifs à la vectorisation sont regroupés dans le module source `"Sallouer.c"` et deux fichiers inclus sont utilisés, `"Allocation.h"` et `"Tampon.h"`, ceci pour pouvoir adapter la vectorisation aux classifications choisies, et éventuellement renommer les vecteurs.

L'ajout ou la suppression de vecteurs sont aussi possibles mais il vont nécessiter de revoir les déclarations des prototypes dans les séquences de supervision des applications (4.2).

Références vers les variables Le mécanisme d'allocation affecte aux variables le rang de la première composante dans le vecteur qui la gère, un vecteur de $\mathbb{R}^k$ occupant k positions : les langages informatiques actuels offrent un opérateur unaire,& en langage C, qui rend la référence d'une variable.

Outre les autres vecteurs du métamodèle, il est ainsi possible de récupérer ces références en utilisant les noms des variables communes STICS. Si par exemple la variable EXCES, qui gère les surplus d'eau en surface et dans les horizons, est

[28] La vectorisation des deux matrices descriptives du système racinaire génère a priori 1 465 464 composantes (3.10.2).

vectorisée en position p, l'affectation '$Exces = \&S[p]$;' va permettre d'accéder aux flux latéraux en surface $Exces[0]$ et aux bases des horizons $Exces[i]$.

Les valeurs des références de toutes les variables sont éditées dans des fichiers avec comme noms les vecteurs du métamodèle, dans l'ordre d'allocation, et les bornes des tables en commentaires, ceci pour pouvoir ensuite déclarer localement des variables locales homonymes et les initialiser avec ces valeurs (3.9.1).

3.2.7 Gestion des dictionnaires des variables et fonctions

Un protocole est mis en place pour gérer les noms symboliques du logiciel source, lesquels désignent les variables informatiques et unités de programmes,

Accès au dictionnaire des variables communes dans les versions C

La mise en correspondance des noms des variables numériques communes et des composantes des vecteurs dans les applications générées s'effectue à partir d'une copie du dictionnaire. Deux méthodes sont appliquées et traduites par deux fonctions.[29]

1. *DicStics*() est générée et effectue des appels de fonction définies dans le module de supervision ;
2. *DicoStics*() initialise des tables de structures et effectue les affectations ;

Edition des variables des vecteurs dans les versions Fortran

Pour éditer ces mêmes états du jour dans l'application Fortran avant ou après exécution de l'unité de programme correspondante, des sous-programmes sont générés pour chacun des vecteurs du métamodèle : $Y_$ est édité par SUIVIY_ dans le fichier "`Y_`".

Seules les valeurs non nulles sont éditées derrière le nom de variable. Pour les vecteurs et matrices, ces valeurs sont précédées des numéros de composante ou ligne et colonne, la valeur -1 signalant la fin de la liste.

Ces sous-programmes sont appelés par SUIVI, lui-même appelé dans un programme principal (4.4.5).

3.2.8 Forme vectorielle pour les applications objets

La présence de la table $Y[][]$ à deux dimensions rend complexe l'utilisation des fonctions C comme méthodes natives dans le langage JAVA, une telle table étant alors un tableau d'objets.

Une autre version, créée sous le nom *Jstics*, regroupe dans un même vecteur les variables antérieurement vectorisées dans $U[]$ et $Y[][]$, $Y[]$ désignant alors ce nouveau vecteur : les variables numériques communes sont alors gérées dans l'octuplet

[29] La première méthode génère un grand nombre d'appels de fonctions qui rend la production du code objet relativement coûteuse. La seconde méthode initialise les tables à la compilation.

$$(K[], S[], V[], M[], W[], X[], Y[], Z[]).$$

Les méthodes natives utilisent alors comme paramètres des tables déclarées avec les types simples, en l'occurrence des vecteurs.

3.3 Traitement des fonctions

Le lexique des unités de programmes Fortran est utilisé pour piloter la lecture des instructions. Pour toute nouvelle unité un fichier est ouvert. La fonction est déclarée à partir de son prototype (2.2.5) puis les variables locales sont déclarées et initialisées. Le corps de la fonction est ensuite traduit. En fin de module, les restitutions requises sont effectuées et le fichier est fermé : la fonction C est alors compilable.

Le traitement effectué transforme finalement les unités de programmes Fortran en fonctions de langages algébriques.

3.3.1 Déclaration des fonctions

Un sous-programme Fortran est introduit par l'instruction SUBROUTINE et une fonction par FUNCTION. Le nom réservé ANONYME indique qu'il s'agit d'un programme principal, l'instruction PROGRAM ayant été omise.

Le nom de sous-programme ou fonction est repris, première lettre majuscule et autres minuscules, pour signifier qu'il s'agit d'un objet global : il est suivi des déclarations des paramètres formels dans une liste parenthèsée, éventuellement vide. Si un même nom est utilisé pour désigner plusieurs unités de programmes, le rang de l'unité dans la liste est ajouté en suffixe pour après la première occurrence.[30]

Les unités de programmes anonymes (1.1.6) sont supposées être des programmes principaux : la fonction *Stics_n()* désigne une telle unité de rang n.

Objets externes

Les déclarations des prototypes des fonctions utilisées sont faites en entête du fichier, par inclusion de fichiers ou désignation explicite des fonctions *Stics* appelées.

Les fonctions intrisèques et de lecture sont déclarées dans deux fichiers d'inclusion :

- `"Commun.h"` (ou `"Super.h"`) pour les fonctions décrites dans le module de supervision (4.2), auxquelles sont ajoutées les directives d'inclusion des bibliothèques standards du langage C ;
- `"Lecture.h"` pour les fonctions de lecture (3.8.4), dans les fonctions *Stics* qui contiennent au moins une telle instruction ;

[30] Les trois exemplaires du sous-programmes LECUSM génèrent les fonctions C *Lecusm(),Lecusm_95()* et *Lecusm_96()* (version 6).

Nouveaux paramètres

Les fonctions générées accèdent aux variables *Stics* communes du logiciel Fortran à travers des paramètres, suffixés par le tiret souligné (_), ceci pour ne pas être homonymes de variables Fortran communes ou locales, ou d'autres symboles.

- Le pointeur sur le fichier compte-rendu est désigné par le paramètre $f_$ de type FILE.[31]
- Le rang du jour simulé, variable commune N dans le code Fortran, est géré avec le paramètre formel $t_$ et les paramètres effectifs
  - $t_$ en cours de simulation,
  - n si la variable locale est présente, 0 sinon, en cours d'initialisation (4.4.3).
- La table des codes textuels est transmise pour les unités de lecture et d'édition, ainsi que pour les unités qui appellent des fonctions de lecture sauf les fonctions d'édition (4.2.3).
- Les vecteurs du métamodèle sont transmis si :
  1. les fichiers de déclarations des variables communes sont inclus, ou des paramètres supprimés sont des variables communes (2.2.5) ;
  2. ils gèrent au moins une variable commune,[32] y compris une image numérique :
     - citée ou mise à jour directement dans la fonction ;
     - citée ou mise à jour dans une fonction appelée ;[33]

Version 5 Les tables de gestion sont transmises en séparant variables statiques ($Code_[][], X0_[], X_[]$), temporelles ($Y_[]$), d'édition ($Z_[]$), accessoirement si des communs sont explicitement cités (2.1.3),[34] sauf pour les utilitaires d'édition avec paramètres.[35]

Le rang du jour calendaire est le paramètre *hui*.

Anciens paramètres des unités de programmes

Pour les sous-programmes et fonctions Fortran déclarés avec des paramètres, le prototype est recherché : les paramètres formels conservés (2.2.5) sont définis par valeur ou référence (2.2.4).

Les variables numériques simples non modifiables sont transmises par valeur : les noms des variables locales sont ajoutés dans la liste des paramètres formels.

[31] Les éditions dans le fichier dit "historique" de *Stics* sont faites dans ce fichier.

[32] L'utilisation d'un nom de variable commune pour désigner un paramètre formel peut ainsi introduire un vecteur en paramètre d'une unité de programme ($Y_[]$ dand la fonction POTSOL).

[33] Un avertissement est édité si le vecteur n'est pas cité chez l'appelant pour signaler l'effet de bord : des variables qui n'apparaissent pas dans le code pourraient ainsi être mises à jour.

[34] Fonction FPF, section 4.5.8.

[35] Les fonctions de mises en forme de champs d'édition (note 48 section 2.4.1) sont exclues.

Les variables numériques simples modifiables sont remplacées par un pointeur formé du préfixe "$p_$" suffixé par le rang du paramètre initial. C'est ce pointeur qui est déclaré dans la liste des paramètres formels, la variable définie par référence étant ensuite déclarée localement.[36]

Toutes les tables sont transmises par référence : le nom du paramètre formel est préfixé par l'opérateur C d'indirection $*$.

3.3.2 Déclaration et initialisation des variables locales

Après lecture des déclarations Fortran, et traitement éventuel de l'instruction DATA, les variables du dictionnaire local autres que celles citées

- dans une liste d'initialisation DATA,
- comme paramètres formels définis par valeur,

sont déclarées et initialisées : des modalités particulières sont appliquées pour les copies des variables communes numériques de la section 2.5.2, pour lesquelles certaines valeurs locales sont à restituer.

Gestion des couples principal/associé, soleil/ombre Certaines variables locales gèrent des couples de la même façon que pour les variables communes. Les mêmes règles sont appliquées pour la génération de code avec suppression de la dimension binaire qui gère le couple et suppression ou neutralisation des évaluations et références à la culture associée ou zone ombragée (3.2.1).

Dans le cas ou il n'est pas établi que la variable gère un couple de cultures associées (2.6.3), l'ambiguité est levée en ajoutant un élément fictif, ce qui fait que la variable locale est déclarée et initialisée comme table à 3 éléments et traitée comme telle dans le code généré.

Déclarations

Les déclarations locales sont générées par les règles syntaxiques décrites en section 1.2.2, avec toutes les variantes relatives aux variables textuelles.[37] La recopie locale des variables numériques communes modifiées localement a pour objectif d'améliorer la lisibilité du code source.
Deux modes de gestion de ces variables sont adoptés :

1. les variables simples, réelles ou entières, et les vecteurs d'entiers, sont repris tels quels ;
2. les tables numériques réelles sont déclarées comme pointeurs pour les vecteurs et tables de pointeurs sur les vecteurs lignes pour les matrices ;

Les tailles de ces tables sont celles des variables communes.

Les autres variables locales, déclarées explicitement ou implicitement, sont reprises, avec suppression éventuelle de la dimension relative à la gestion des cultures associées.

[36] La variable *cnresidu*, paramètre formel de rang 3 modifié dans le sous-programme APORES , est remplacée par le pointeur $*p_3$ dans la liste des paramètres formels.

[37] CHARACTER*20 NOMPARAM(0 :10) dans ECRIPARAM, version 6.

Les variables textuelles Fortran sont redéfinies en C comme chaînes de caractères, pour gérer :

- des codes, textes courts d'au plus 32 caractères ;[38]
- des lignes, textes longs d'au plus 256 caractères ;

Initialisations

Les variables locales sont initialisées par défaut avec la valeur numérique nulle, ou la chaîne vide pour les données textuelles : les boucles d'initialisation sont générées immédiatement après les déclarations.

Les initialisations Fortran DATA sont traduites avec la directive, par affectation ou dans une boucle pour la forme '$< n > * < valeur >$' qui répète n fois la valeur fournie dans la liste.

Les variables simples associées à des paramètres formels transmis par référence sont initialisées avec le pointeur déclaré pour ce paramètre , soit pour l'exemple de la note 36 : $cnresidu = *p_3$;

Les copies des variables communes numériques sont initialisées en fonction du mode de déclaration précédent :

1. les variables simples, réelles ou entières, et les vecteurs d'entiers, sont initialisées par affectation des valeurs courantes ;
2. les tables réelles sont initialisées avec les pointeurs sur les variables communes, regroupés dans une table pour les matrices ;

3.3.3 Retour vers la fonction appelante

Le code normalisé d'une fonction se termine par l'instruction Fortran END, réintroduite par la commande $S1$ si absente du code source : le contrôle est rendu à la fonction appelante. Cette instruction finale est atteinte en séquence, ou après exécution de l'instruction Fortran RETURN, équivalente à un branchement, décrite avec les ruptures de séquences (3.7.3).

Epilogue d'une fonction Le code généré est complété par deux séquences finales d'affectations :

1. la restitution dans les variables numériques communes des valeurs gérées dans les copies locales ;[39]
2. le transfert des valeurs gérées dans les variables simples associées à des paramètres formels définis par référence à travers le pointeur reçu lors de l'appel, soit pour l'exemple de la note 36 : $*p_3 = cnresidu$;

3.3.4 Déroutement

L'instruction Fortran STOP entraîne l'arrêt immédiat de l'exécution. Elle est traduite en C par l'appel de la fonction de traitement des erreurs

[38] La table Fortran d'un seul caractère CODPAS est vue comme un code.

[39] La valeur de sortie de boucle est ainsi rendue pour les variables de boucle communes.

int *Erreur*(int n, char $*message$, FILE $*f$) ,

qui clôt l'exécution en éditant *message* dans le fichier référencé par $*f$, $-n$ étant le rang de l'instruction STOP(4.2.1).

Récupération des erreurs Pour pouvoir traiter ces erreurs fatales dans d'autres contextes applicatifs, avec une autre fonction *Erreur*() qui rend le contrôle à l'appelant, l'appel s'effectue dans une instruction return : le numéro de la ligne source Fortran est noté dans *Y_*[0].

3.3.5 Appel

Un sous-programme FORTRAN est appelé par l'instruction CALL, suivie de son nom. Une fonction FORTRAN est appelée directement par son nom, en début de ligne ou derrière un if logique si elle ne rend pas de valeur, dans une expression arithmétique ou logique si elle rend un résultat.

Les fonctions C générées sont définies avec les paramètres formels auxquels correspondent les paramètres effectifs de même nom, à l'exception de la variable commune Fortran N (3.3.1).

Liste des paramètres

Un nom de sous-programme ou fonction FORTRAN est optionnellement suivi d'une liste de paramètres effectifs, alors placée entre parenthèses. Cette syntaxe introduit une ambiguïté, une fonction pouvant être confondue avec une table. Le nom est d'abord recherché dans les dictionnaires des variables locales et communes : la dimension d'une table devant être explicitement déclarée, l'absence du nom dans la liste des variables signifie qu'il s'agit bien d'une fonction.

Les noms de fonctions C sont édités dans le code généré avec la liste des paramètres effectifs entre parenthèses, éventuellement vide. La structure syntaxique de est décrite par les règles

$< fonction > \longrightarrow < nom > (< liste\ parametres >)$
$< liste\ parametres > \longrightarrow < effectif > \{, < effectif >\}$
$< effectif > \longrightarrow < nombre > \mid < variable > \mid < expression >$

Cette grammaire signifie qu'un paramètre effectif est une constante numérique, une variable ou une expression arithmétique, les dérivations finales étant décrites dans les sections suivantes.

La syntaxe du langage Fortran ne permet pas de déterminer si les paramètres sont définis par valeur ou référence (2.2.4) :[40] la recherche des paramètres modifiables dans les anciennes unités de programme ayant été faite pour construire les prototypes des fonctions, ces informations sont reprises pour gérer le passage des paramètres par valeur ou référence dans la traduction C.

[40] Dans l'appel F(X), X désigne à la fois la variable et sa valeur : la variable peut être mise à jour dans le corps de la fonction F.

Transmission par valeur Les nombres et expressions sont nécessairement transmis par valeur. Ils sont reconnus comme précédement, les fonctions pouvant être appelées depuis des expressions et les expressions transmises en paramètres contenir des appels de fonctions.

Dans ce contexte, toutes les variables, paramètres définis par valeur ou variables référencées dans les expressions transmises, sont citées comme précédemment dans les expressions arithmétiques.

Transmission par référence Les références sont les adresses des variables, en langage C les noms des tables ou les valeurs rendues au moyen de l'opérateur unaire & placé devant une citation de variables. Si la variable globale X est allouée en position p dans le tampon T, le paramètre effectif $\&T[p+i]$ est l'adresse de l'élément de rang i dans la table X.

Une référence transmise en paramètre est gérée dans la fonction appelée dans un pointeur spécifique, comme indiqué en section 3.3.1, avec affectation de la valeur initiale dans une variable locale homonyme du paramètre formel correspondant, et restitution finale.

Dans le cas particulier où une variable locale transmise en paramètre est une table, l'opérateur & est à proscrire puisque le nom de la table désigne en langage C son adresse.

3.3.6 Fonctions intrinsèques

Les références externes inconnues sont souvent des fonctions Fortran dites intrinsèques, fournies dans des bibliothèques sous des noms d'usage spécifiques. Une liste de ces fonctions est constituée : elle est mise en correspondance une liste de fonctions C équivalentes (2.2.1), pour la plupart disponibles dans la bibliothèque standard du langage C, quelquefois créées.[41]

Certaines fonctions intrinsèques propres à des compilateurs peuvent apparaître comme des unités de programmes car inconnues pour d'autres compilateurs : la fonction C correspondante est alors à développer.[42]

D'autres fonctions utiles peuvent être ajoutées en fonction des besoins, comme *Age*() (3.2.6), qui peut apparaître dans le code généré pour vérifier la non négativité des indices des séries temporelles.

Conversions de types Les fonctions de conversion de type sont remplacées par l'opération unaire usuelle dite *cast*.[43] La présence de la marque $*$ comme nom de fonction C dans la table de correspondance indique que le nom de l'opérateur est le type de la fonction : NRAC = INT(ZRAC) devient $nrac = (int)\ (Zrac)$.

[41] À titre d'exemple, les fonctions Fortran MIN*(n1,n2)* et AMIN1*(n1,n2)*, qui rendent le minimum de deux nombres, entiers pour la première forme et réels pour la seconde, sont remplacées par la fonction C *Min(n1,n2)* qui gère les conversions de types.

[42] La fonction Fortran MINVAL(TAB,DIM), qui rend la valeur minimale dans une table, est inconnue des compilateurs sous LINUX.

[43] Fortran utilise plusieurs fonctions de conversion entière : INT,AINT, IFIX.

La fonction Fortran NINT n'est pas vue comme une opération de conversion de type : elle rend la valeur arrondie et est traduite par la fonction C *Arrondi()*.

3.4 Désignation des variables et constantes

Les composants syntaxiques < *nombre* > et < *variable* > cités précédemment dans la liste des paramètres effectifs sont présents dans d'autres structures syntaxiques : expressions arithmétiques et logiques, listes de lecture et d'édition, indices des tables.

3.4.1 Désignation des variables

Les variables Fortran sont identifiées par des noms (1.1.3), les formes minuscules et majuscules étant équivalentes. Elles sont simples ou tabulaires. Les indices des tables sont ajoutés à droite entre parenthèses et séparés par des virgules.

L'élément syntaxique < *variable* > peut être décrit par les dérivations :

$< variable > \longrightarrow < nom > (< liste\ indices >)$
$< liste\ indices > \longrightarrow < indice > \{, < indice >\}$
$< indice > \longrightarrow < entier > \mid < variable\ entiere > \mid < expression\ entiere >$

Le composant < *entier* > désigne normalement un nombre entier positif ou nul, et les autres composants produisent un nombre entier.[44]

Restitution des variables

Les variables locales Fortran sont désignées directement par leurs noms repris sous la forme minuscule : X(I) devient $x[i]$.

Pour les variables communes Fortran, c'est l'élément du tampon (3.1.2) qui est cité : X(I) devient $T[p+i]$ si la position p a été allouée à la variable commune X dans la table T reçue en paramètre. Le nom de la variable commune est ajouté en commentaire avec la première lettre majuscule.[45]

Couples de valeurs Les indices sont extraits en séparant ceux relatifs aux variables du modèle de la désignation des couples d'occurrences des cultures associées (2.6). Deux modes de gestion sont appliqués pour les versions générées.

1. Pour une culture simple, il n'existe qu'une occurrence possible, la plante principale étant dominante : les indices utilisés pour choisir ces occurrences sont supprimés.

[44] Les compilateurs admettent généralement des variables ou expressions non entières : c'est la partie entière du nombre évalué qui est alors retenue comme valeur d'indice.

[45] MASVOLCX(TYPECAILLOUX(ICOU)) devient $T[p + typecailloux[icou]]$, avec le commentaire /* 0 : $Masvolcx$[] */.

2. Pour les cultures associées, les indices sont généralement des variables simples, communes ou locales, qu'il faut évaluer en cours d'exécution pour restituer la valeur désignée. Si une valeur de la variable X précédente occupe k positions, X(I,2) devient $T[p+k+i]$ puisque c'est la deuxième valeur qui est citée.

3.4.2 Représentation des nombres

Les modes de représentation utilisés sont restreints à la notation décimale usuelle et à la notation avec exposant. Les syntaxes sont les suivantes :

$< nombre > \longrightarrow [< signe >] < decimal > [. < decimal >]$
$< nombre > \longrightarrow [< signe >] < decimal > [.]$
$< nombre > \longrightarrow < nombre > E[< signe >] < decimal >$
$< signe > \longrightarrow -|+$

Le point est aussi utilisé pour coder les opérateurs FORTRAN de relation dans les expressions logiques, ce qui peut nécessiter d'examiner les caractères suivants pour reconnaître le point décimal, comme dans la condition logique Fortran (PRECIPSOL.GT.0..OR.NSTOC.GT.0). En conséquence, la fonction de reconnaissance des nombres rend un nombre ou un opérateur Fortran de relation.

3.4.3 Chaînes de caractères

Les valeurs textuelles sont notées entre quotes ou guillemets soit

$< chaine > \longrightarrow$ '$< texte >$' | "$< texte >$"

Tous les caractères sont admis dans le composant $< texte >$, le délimiteur utilisé étant doublé pour être restitué dans le texte.

3.5 Expressions arithmétiques et logiques

Les expressions arithmétiques sont présentes dans les opérations d'affectation mais peuvent aussi être évaluées dans des conditions logiques, comme paramètres effectifs de fonctions ou encore dans des listes d'édition.

Les expressions entières forment un sous-ensemble et sont utilisées pour évaluer les indices des tables et les bornes des boucles.

Les expressions logiques contrôlent les instructions conditionnelles, parfois pour des variantes du langage Fortran des boucles **tant que**.[46]

Dans toutes ces expressions sont citées les variables et il y a lieu de traiter pour les tables Fortran les couples de valeurs des cultures associées.

[46] La forme "DO $< n >$ WHILE $<$ *expression logique* $>$ ..." des versions 5 et 6 est reconnue mais non traduite : le corps de cette unique boucle contient comme seule instruction un branchement inconditionnel en arrière.
La boucle **pour** qui enchaîne les simulations quotidiennes est remplacée dans la version 6 par un "DO WHILE..."

3.5.1 Expressions arithmétiques

Avec les notations syntaxiques du paragraphe 1.1.1, les crochets étant des métasymboles, les expressions sont produites par les règles suivantes :

$< expression > \longrightarrow < facteur > \{< opa >< facteur >\}$
$< expression > \longrightarrow < facteur > \{** < facteur >\}$
$< expression > \longrightarrow (< expression >)$
$< opa > \longrightarrow + | - | * | /$
$< facteur > \longrightarrow < nombre > | < variable > | < fonction >$

Dans la première règle, la répétition des opérations nécessite de fixer des règles de priorités entre opérateurs.

En sus des opérateurs arithmétiques usuels, ** traduit une exponentiation ou une puissance entière.

La troisième règle, récursive, introduit le parenthèsage et permet la gestion explicite des priorités.

La syntaxe Fortran ne permet pas de distinguer variables et fonctions. Les noms citées sont recherchés tant qu'ils ne sont pas trouvées dans les dictionnaires de variables puis dans les listes de fonctions, soit la séquence :

1) variables locales ;
2) variables communes ;
3) fonctions intrinsèques ;
4) fonctions *Stics* ;

La syntaxe des appels de fonctions a été décrite en 3.3.5.

Exponentiation et puissance entière

Le langage Fortran utilise l'opérateur ** pour évaluer x^y, prioritaire, défini pour x et y réels. Cet opérateur est aussi utilisé pour évaluer des puissances entières x^n, bien que l'opération ne soit pas définie dans le langage, ce qui aboutit à des variations liées aux compilateurs (4.1.2).

En langage C, cette opération est effectuée par un appel de la fonction mathématique $pow(x, y)$ de la bibliothèque standard mais il faut récupérer l'erreur de domaine si $x \leq 0$ et y n'est pas entier.

La solution adoptée pour cette traduction utilise deux fonctions : puissance et exponentiation.

1. Si l'exposant lu est le nombre 2, la puissance est évaluée simplement par appel de la fonction $Pn(x, 2)$.
   Les autres puissances entières sont traduites par l'exponentiation, du fait de problèmes de précision révélés à l'exécution (4.1.2).
2. Sinon, la fonction $Px(n, x, y)$ est appelée et l'évaluation est faite par la fonction d'exponentiation $pow(x, y)$ seulement si $x > 0$: si $x \leq 0$, un

déroutement qui clôt l'exécution est généré, avec comme numéro d'erreur le rang n de l'instruction Fortran traduite.[47]

Les appels de fonction étant prioritaires devant les opérateurs arithmétiques, l'ordre des évaluations est conservé.[48]

3.5.2 Expressions entières

Les expressions entières sont normalement définies avec des nombres entiers et des variables ou fonctions entières. Le langage Fortran accepte toutefois des nombres ou fonctions réelles alors converties implicitement.

La reconnaissance des expressions entières (opérande droite d'affectation entière, bornes pour les boucles, paramètre effectif entier, indice de table...) est restreinte aux formes simples : des avertissements sont édités si nécessaire afin de pouvoir vérifier le code généré .[49]

Des fonctions intrinsèques FORTRAN sont quelquefois utilisées pour effectuer les conversions entières :

INT et IFIX convertissent par troncature une valeur réelle en valeur entière : elles sont remplacées en C par la conversion implicite qui effectue cette même troncature.

NINT, qui effectue la meilleure approximation entière, est remplacée par la primitive *Arrondi*().

Les variables entières citées peuvent être des tables entières, lesquelles contiennent alors en indices d'autres expressions entières.[50]

Conversions entières Les conversions entières sont effectuées explicitement dans le code C généré dans les situations suivantes :

- des variables ou fonctions réelles sont citées ;
- des variables communes sont utilisées, donc gérées dans un vecteur réel, sauf si l'opérande gauche d'affectation est aussi une variable commune ;

3.5.3 Expressions logiques

Les expressions logiques sont construites, d'un point de vue formel, avec :

- **les opérateurs logiques** binaires $\wedge$ de conjonction et $\vee$ de disjonction inclusive, et l'opérateur unaire $\neg$ de négation ;
- **les opérateurs de relation** binaires usuels $=, \neq, >, \geq, <, \leq$, définis sur les ensembles de nombres ;

[47] Il est alors nécessaire de revoir le code généré : l'absence d'initialisation de variables déclarées peut produire ce type d'erreurs avec certains compilateurs.

[48] (KARM**2*U) est traduit *(Pn(karm,2)*u)*.

[49] Des bornes de la forme MAX(INT(ZRAC),ISEMHAUT) (sous-programme LEVEE) sont sources d'ambiguîtés.

[50] MASVOLCX(TYPECAILLOUX(ICOU)) désigne la densité des cailloux dant la valeur est gérée dans le paramètre général MASVOLCX, table définie pour une typologie de cailloux, TYPECAILLOUX étant un paramètre du sol défini pour les horizons.

La table suivante indique les équivalences avec les opérateurs Fortran et C.

	FORTRAN	C
$=$	.EQ.	$==$
$\neq$	.NEQ.	$!=$
$<$	.LT.	$<$
$\leq$	.LE.	$<=$
$>$	.GT.	$>$
$\geq$	.GE.	$>=$
$\wedge$	.AND.	$\|\|$
$\vee$	.OR.	&&
$\neg$	.NOT.	!
$=$	.EQV.	$==$
$\neq$	.NEQV.	$!=$

Grammaire générique Les expressions logiques Fortran sont produites par les règles générales

$< formule > \longrightarrow < operande >$ `.and.` $< operande >$
$< formule > \longrightarrow < operande >$ `.or.` $< operande >$
$< formule > \longrightarrow$ [`.not.`] $< operande >$

Le composant $< operande >$ se dérive sous des formes numériques, logiques ou textuelles.

Valeurs numériques

La plupart des conditions logiques reprennent le composant $< expression >$ des expressions arithmétiques pour produire des relations binaires :

$< operande > \longrightarrow < terme > < opr > < terme >$
$< terme > \longrightarrow (< formule >) | < formule >$
$< terme > \longrightarrow (< expression >) | < expression >$
$< opr > \longrightarrow$ `.eq.`|`.neq.`|`.gt.`|`.ge.`|`.lt.`|`.leq.`

Priorité des opérateurs Si le composant $< expression >$ n'est pas placé entre parenthèses, ces dernières sont ajoutées dans le code C généré en fonction des priorités adoptées dans le langage Fortran.[51]

Valeurs logiques

Les variables logiques Fortran produisent des relations unaires avec deux valeurs constantes mais aussi une relation binaire avec deux nouveaux opérateurs logiques :

$< operande > \longrightarrow < logique >$
$< operande > \longrightarrow < logique >$ `.eqv.` $< logique > | < logique >$ `.neqv.` $< logique >$

[51] (XPREC.LE.L/2.AND.X.GE.L/2) est traduit $((xprec <= (l/2)) \&\& (x >= (l/2)))$.

$< logique > \longrightarrow < variable >$ |`.true.`|`.false.`

Les variables logiques étant numérisées dans la traduction C, les valeurs logiques ".FALSE." et ".TRUE." sont remplacées respectivement par 0 et 1.[52]

Valeurs textuelles

Les formes textuelles utilisent le composant $< chaine >$ et produisent des relations binaires avec deux opérateurs :

$< operande > \longrightarrow < textuel >$
$< operande > \longrightarrow < textuel >$ `.eq.` $< textuel >$ | $< textuel >$ `.neq.` $< textuel >$
$< textuel > \longrightarrow < variable | < chaine >$

Deux modes de traduction sont adoptés, selon que les variables textuelles sont référencées exclusivement comme variables d'initialisation ou d'édition, ou utilisées dans la boucle quotidienne comme variables de simulation (2.5.1).

1. La comparaison des valeurs textuelles s'effectue au moyen de la fonction *strcmp*() de la bibliothèque standard du langage C.
2. Pour les variables numérisées (2.1.5), la valeur numérique associée est le rang dans la liste des codes en clair répertoriés antérieurement, les espaces et tabulations étant supprimés : la valeur textuelle est ajoutée entre guillemets en commentaire derrière le nom de la variable référencée.[53]

Tables de codes textuels La version 6 introduit la table de codes CODEVAR. Si les codes lus sont effectivement numérisés, les éléments ne sont pas reconnus dans une expression logique.

3.5.4 Expressions textuelles

Les expressions textuelles sont remplacées par des expressions numériques si les images ont été créées (2.1.5).

Les variables et valeurs textuelles peuvent être citées dans les listes d'édition, les expressions logiques ou encore comme paramètres lors d'un appel : elles sont vues comme des expressions simples.

Remplacement par des expressions numériques

La substitution s'effectue pour toutes les variables temporelles numérisées : il est vérifié que l'image numérique est présente à la ligne suivante du dictionnaire des variables communes.

[52] L'utilisation des opérateurs binaires dans la version 6 introduit une redondance : la condition logique Fortran IF (GELEE .EQV. .TRUE.) est équivalente à IF (GELEE), ce que révèle la traduction C : if *((Gelee == 1) == 1)*.

[53] L'expression logique (N.LT.NPLT.AND.CODEPLANTE.EQ.`'snu'`) apparaît dans le sous-programme d'évaluation de la croissance racinaire CROIRA, CODEPLANTE étant lue avec les paramètres de la plante : elle est traduite sous une forme exclusivement numérique en utilisant la variable image *Codeplante_*, la valeur "`snu`" figurant en commentaire.

Accès à un champ textuel

Les opérations textuelles ne sont traduites que très partiellement.

Indiçage Un mode d'accès particulier est utilisé en Fortran pour gérer des champs dans des tables de caractères.[54] Il peut être décrit par ajout d'une règle de dérivation du composant syntaxique $< indice >$ (3.4.1), soit

$< indice > \longrightarrow < borne > : < borne >$
$< borne > \longrightarrow < entier > \mid < variable\ entiere > \mid < expression\ entiere >$

Ce mode n'est actuellement pas traité : un message est édité pour la première occurrence dans une unité de programme et le commentaire " `!traitement champ textuel`" est ajouté en regard de l'instruction C générée.

Fonction La fonction intrinsèque INDEX est utilisée pour extraire le nom du fichier dans le répertoire courant.[55] Elle est remplacée dans la traduction C par la fonction $Droite(x, y)$ qui recherche la chaine y à droite de x.

Concaténation

Des expressions textuelles sont construites en Fortran au moyen de l'opérateur de concaténation //.
Cette opération n'est pas traduite mais sa présence est signalée par un message édité pour la première occurrence dans l'unité de programme : le commentaire " `!concaténation non traduite`" est ajouté derrière l'instruction C générée.[56]

3.6 Evaluation des variables

Les calculs numériques sont pour la plupart traduits dans des affectations, plus rarement par passage de paramètres par référence (2.2.4) : les valeurs sont conservées dans les variables informatiques.

3.6.1 Affectations directes

En Fortran comme en C, c'est le symbole = qui est utilisé utilisé comme opérateur d'affectation :

- l'opérande droite est la variable qui reçoit le résultat ;
- l'opérande gauche contient un nombre ou une expression arithmétique, ou en Fortran des valeurs ou expressions logiques ou textuelles ;

Une affectation s'effectue dans une instruction Fortran simple ou conditionnelle (if logique).

[54] Exemple : après l'instruction MOUI='RAPPORT.STI', MOUI(1 : 7) désigne 'RAPPORT'.
[55] Par exemple INDEX(FPLT(IPL),'solnu.plt').GT.0) dans le sous-programme LECTECH.
[56] Ces expressions sont utilisées pour construire les chemins d'accès aux fichiers en fonction des systèmes d'exploitation hôtes.

Les expressions arithmétiques sont directement traduites.

Pour les affectations logiques, seules sont reconnues comme opérandes droites
- les constantes ".FALSE." et ".TRUE.", remplacées par 0 et 1,
- les variables logiques, locales ou communes.[57]

Les textes, entre quotes ou guillemets sont repris :
- pour les textes courts avec la fonction de recopie de chaînes *strcpy()* de la bibliothèque standard du langage C ;
- pour les textes longs directement dans un ordre d'édition d'une instruction de choix multiple,[58] avec comme variable de choix le rang de la variable (3.8.3).

Variables de simulation textuelles

Pour les variables de simulation textuelles initialisées par affectation directe d'une valeur textuelle, les variables numériques images sont simultanément initialisées dans une nouvelle instruction avec le rang dans la liste des codes (2.1.5) : le code textuel est ajouté entre guillemets en commentaire derrière le nom de la variable. Si ce code est inconnu, parce que non cité dans une expression logique, la variable numérique demeure indéterminée.[59]

Ces variables numériques sont gérées comme les autres dans les vecteurs.

3.6.2 Transmissions en paramètres

Des valeurs évaluées dans une fonction peuvent être rendues à l'appelant dans une variable transmise en paramètre. Dans ce cas, décrit en 2.3.4, si la variable citée comme paramètre effectif chez l'appelant n'est pas elle-même une référence, elle est préfixée par l'opérateur unaire de référence & en langage C.[60]

3.6.3 Gestion locale des variables communes

Les paramètres formels définis par référence, les variables communes modifiées et les indices des tables sont recopiés dans des variables locales, et restitués en fin d'exécution comme indiqué en 3.3 si nécessaire. En conséquence, les noms de variables apparaissent en clair sous la forme minuscule dans le code C généré et les indices des tables sont bien des entiers.[61]

Ce choix permet de se prémunir contre des modifications imprévues consécutives à des erreurs qui pourraient être retournées à la fonction appelante, la table $X[]$ étant nécessairement en langage C un paramètre défini par référence.

[57] Ce type d'affectation apparaît dans la version 6 et avec les couples de cultures associées.
[58] L'affectation conditionnelle n'est pas traduite et sa présence génère un avertissement.
[59] La numérisation n'est pas faite pour une affectation dans des tables de codes textuels.
[60] Le nom d'une table $x[]$ désigne l'adresse de l'élément de rang 0, autrement dit $\&x[0]$.
[61] Si des compilateurs C effectuent les conversions entières d'indices réels, ils retiennent la partie entière et éditent généralement des avertissements : cette traduction automatique n'a pas été effectuée dans la version provisoire **Hysafe**.

3.7 Contrôle de l'exécution

Dans les langages algébriques, des séquences d'instructions sont contrôlées intégralement et exclusivement par des prédicats logiques, qui permettent d'effectuer des choix ou d'exécuter des itérations. Ces séquences sont exécutées en totalité ou pas du tout, donc ne contiennent aucun branchement depuis ou vers l'extérieur : ils définissent une transaction.

Le langage Fortran n'est pas initialement un langage algébrique et les structures de contrôle itératives et conditionnelles sont souvent traduites par des branchements, parfois en l'absence d'autres solutions avec la norme Fortran77 pour les structures itératives. Il se peut néanmoins que les conditions logiques Fortran contrôlent effectivement à travers ces branchements des transactions.

L'analyse syntaxique de la section 1.3 a permis de répertorier des séquences d'instructions contrôlées par des prédicats, conditionnelles, itératives ou introduites par des branchements explicites, et décrites dans des relations. Pour pouvoir les restituer dans un langage algébrique, il faut examiner si le bloc ainsi constitué contient des branchements vers l'extérieur, ou depuis l'extérieur vers une instruction du bloc alors repérée par une étiquette logique Fortran. Ces branchements ont également été répertoriés dans la phases d'analyse syntaxique et décrits dans une relation : les blocs qui les contiennent sont appelés sections critiques.

Les descripteurs de blocs et sections critiques sont repris pour traduire le code et quelques règles simples ont pu être appliquées pour construire des structures syntaxiques algébriques en supprimant les branchements.[62] Les cas résiduels font l'objet de messages d'avertissements,[63] et sont à reprendre en fonction du contexte pour les restituer dans un langage algébrique.

Le résultat final est un code décrit formellement par une grammaire algébrique, généré à partir d'un sous-ensemble du langage C et utilisable dans le monde du logiciel libre.

Cultures associées La commande $S5$ (1.3.7) a produit un fichier de repérage des structures de contrôle liées à la gestion des couples. Il est utilisé pour retrouver les séquences exécutées selon le nombre de cultures, et produire ainsi une version culture pure : les blocs d'instructions associés à la seconde culture sont supprimés, ainsi que les affectations dans les variables de gestion de ces couples, initialisées à la valeur unitaire et invariantes.

3.7.1 Itération à pas constant

La norme du langage Fortran77 ne reconnaît que l'itération à pas constant de la section 1.3.2. Elle est introduite par le mot-clé DO, suivi de

une étiquette qui localise la fin de la boucle,

[62] Les étiquettes de ces branchements sont citées en commentaires dans le code généré.

[63] Le volume de ces messages pour un logiciel quelconque permet d'évaluer la faisabilité pratique d'une traduction avec ces instruments.

le nom de la variable de boucle,[64]
les bornes inférieure et supérieure de l'intervalle de variation,
un pas optionnel, positif ou négatif, unitaire par défaut.

Les bornes et le pas sont des valeurs, des variables ou des expressions entières, normalement invariantes pendant l'exécution de la boucle.

Contrôle de la boucle

Une variable de boucle est normalement locale et ne doit pas être modifiée dans le corps de la boucle : un avertissement est édité si la variable Fortran est globale, ou s'il s'agit d'un paramètre formel.

Corps de la boucle

Le corps de la boucle, délimité par une étiquette de branchement finale en Fortran, définit un bloc d'instructions, placé entre accolades en langage C. Ce bloc est géré dans une relation produite lors de la reconnaissance syntaxique. La transcription s'effectue pour toutes les instructions en séquence, tant que l'instruction finale n'est pas trouvée. Les accolades peuvent être supprimées si le bloc est réduit à une instruction, éventuellement une autre boucle en langage C, la structure étant récursive.

L'instruction finale étiquetée contient usuellement le mot-clé CONTINUE, qui marque la fin de la boucle. Dans le cas où il s'agit d'une autre instruction, cette dernière est intégrée dans le corps de la boucle : un message est néanmoins édité, cette situation pouvant être source d'ambiguïté dans la compilation Fortran. Plusieurs boucles Fortran peuvent se terminer sur une même étiquette finale. L'accolade fermante est ajoutée si nécessaire avec en commentaire le rang de l'instruction Fortran.

Conversion en boucle Tant que La première instruction du corps de la boucle est quelquefois un branchement conditionnel derrière l'instruction finale de cette boucle : cette structure est convertie en boucle *Tant que*, l'ancienne variable de boucle étant alors initialisée puis incrémentée, et le prédicat de contrôle reconstruit.

Contrôle d'intégrité

Le corps de la boucle définit une section critique et s'exécute normalement tant que la valeur de la variable de boucle reste dans l'intervalle de variation.

[64] Dans la version 5 de *Stics* apparaît le mot-clé WHILE à la place du nom de la variable de boucle. Il est suivi d'une expression logique alors utilisée pour générer en langage C une boucle Tant que : un avertissement est néanmoins édité car cette forme n'appartient pas à la norme Fortran77.

Sortie de boucle par un branchement Il apparaît quelquefois des branchements en avant vers une étiquette située en dehors du corps de la boucle : ces formes peuvent en fait traduire les les structures itératives classiques *Tant que* et *Jusqu'à* (1.3.2). Elles sont signalées pour une reprise à vue du code C généré.

1. Le prédicat de contrôle est la conjonction de la condition logique de branchement et de l'appartenance de la variable de boucle à l'intervalle de variation.
2. L'incrémentation de la variable de boucle est faite par une instruction explicite.
3. Les sections de code dont l'exécution dépend des conditions de sortie de la nouvelle boucle sont examinées :
   - si l'instruction de branchement n'est pas la dernière du corps de la boucle, il faut ajouter une structure conditionnelle en sortie de boucle pour traiter le bloc qui précède le branchement ;
   - si les étiquettes de fin de boucle et de ne sont pas consécutives, une séquence conditionnelle est à ajouter pour prendre en compte le bloc situé entre ces étiquettes ;

Cette forme est aussi utilisée dans des boucles imbriquées qui gèrent les couches des différents horizons du sol, avec branchement depuis la boucle interne directement en sortie de la boucle externe (4.5.10).

La restructuration serait à effectuer dans le code Fortran (annexe **??**).

Entrée dans une boucle par un branchement Les branchements depuis l'extérieur de la boucle sont normalement inexistants.

Gestion des couples de cultures

Les boucles s'exécutant au plus une seule fois pour une culture simple, elles sont alors supprimées. Les corps des boucles sont conservés, sauf si la borne inférieure de la boucle vaut 2 (4.3.5).[65]

Les occurrences sont les suivantes.

Plantes dominante/dominée Les évaluations relatives aux deux plantes sont faites dans des boucles qui varient depuis 1 jusqu'à NBPLANTES, paramètre initial lu qui vaut 1 ou 2 : la variable de boucle est généralement la variable commune IPL, quelquefois une variable locale.

Zones ensoleillée/ombragée Ces évaluations sont faites dans une boucle qui utilise la variable commune ENS et les deux constantes AS $= 1$ et AO $= 2$.

Ecoulement de l'eau sur les feuilles La quantité d'eau est cumulée dans une boucle avec un pas unitaire décroissant depuis NBPLANTES $- 1$ jusqu'à 1. Les valeurs des indices ne pouvant être que positives, cette boucle s'exécute une fois seulement si (NBPLANTES $= 2$), et jamais pour une culture pure.[66]

[65] Il n'existe pas dans le code actuel de boucles à pas négatif sur les couples de cultures.

[66] Sous-progamme CALPRECIP : DO 3 IPL=NBPLANTES-1,1,-1
Pour des cultures associées, il suffirait d'exécuter le bloc conditionnel Si (NBPLANTES $= 2$).

3.7.2 Instruction conditionnelle

L'instruction conditionnelle Fortran est introduite par le mot-clé IF (1.1.5).

Le if arithmétique, qui gère un choix ternaire, n'est pas utilisé.

Dans le if logique, c'est l'instruction située à droite sur la ligne qui est exécutée, laquelle peut être une affectation ou un appel, ou un branchement en avant ou en arrière, traité avec les ruptures de séquences.

Le if structuré a été introduit dans la norme Fortran77. Le mot-clé THEN introduit la séquence exécutée si l'expression logique est vraie, suivie du mot-clé optionnel ELSE pour introduire la séquence alternative jusqu'au mot-clé final ENDIF. Le if logique suivi d'une instruction d'affectation ou d'un appel correspond de fait à cette même structure avec un bloc THEN réduit à une instruction.

Suppression de blocs conditionnels Afin de pouvoir supprimer des sections de code obsolètes ou sans objet dans ce contexte, *SautAlors*[] contient une liste de conditions logiques pour lesquelles le bloc *Alors* peut être supprimé, soit actuellement les conditions simples :

- "$Pas = 2$", option données météorologiques décadaire ;
- "$Numcult > 1$" et "$Nbans > 1$", la boucle sur les cultures n'étant pas reprise sous cette forme ;[67]

et toute conjonction dans laquelle l'une de ces conditions est vérifiée.

La suppression est effectuée seulement si une structure itérative pilotée par un branchement final (1.3.2) n'est pas en cours de reconnaissance (4.5.7).

Rang des cultures L'exécution de certaines séquences est contrôlée par le rang de la culture, en particulier pour les cultures pérennes pour lesquelles l'année d'installation est à distinguer des autres. Une autre liste, *RangCulture*[], regroupe les variables de gestion des rangs des cultures, pratiquement *Numcult* et *Nbans*, ceci pour localiser les blocs conditionnels contrôlés par des prédicats dans lesquels ces variables sont citées.

Structure conditionnelle

Cette structure est traduite en langage C sous la forme générique

IF $< formule >$ $\{< bloc1 >\}$ ELSE $\{< bloc2 >\}$,

Les blocs sont gérés comme précédemment dans une relation produite au cours de l'analyse syntaxique. Les accolades ouvrante et fermante encadrent un bloc de plusieurs instructions : les débuts et fins de blocs sont alors repérés en commentaire avec indication des rangs des instructions Fortran correspondantes. L'alternative introduite par ELSE est optionnelle.

Les accolades peuvent être supprimées si les blocs contiennent une seule instruction, éventuellement conditionnelle ou itérative en langage C, la structure étant récursive. Elles sont conservées en présence de l'alternative ELSE, ceci pour éviter toute ambiguïté dans la reconnaissance des structures imbriquées.

[67] "$Codeinitprec = 2$" serait à ajouter, sous réserve de revoir dans le code Fortran la gestion des rotations de culture (4.3.9).

Contrôle d'intégrité

Les blocs conditionnels définissent des sections critiques et le choix du bloc à exécuter dépend exclusivement de l'expression logique associée.

Sortie de bloc conditionnel La structure suivante est reconnue :

IF $<$ *formule* $>$ THEN
$<$ *bloc*1 $>$
GOTO $<$ *etiquette* $>$
ENDIF
$<$ *bloc*2 $>$
$<$ *etiquette* $>$ CONTINUE

L'absence de l'alternative Fortran ELSE autorise, si cette stucture n'est pas elle-même imbriquée dans une autre conditionnelle, la traduction C

IF $<$ *formule* $>$ $\{<$ *bloc*1 $>\}$ ELSE $\{<$ *bloc*2 $>\}$,

Dans tous les autres cas, les blocs sont restucturés au vu du code avec reconstruction des prédicats logiques à partir des conditions logiques des branchements conditionnels Fortran, qui sont finalement supprimés.

Entrée dans un bloc conditionnel Les branchements vers une instruction des blocs conditionnels sont normalement inexistants.

Gestion des couples de cultures

L'analyse des conditions logiques utilisées pour gérer les couples permet d'associer les blocs conditionnels aux cultures. Il s'agit généralement d'opérations logiques simples de la forme

$<$ *variable* $><$ *op* $><$ *valeur* $>$ avec

- les variables IPL,ENS,NBPLANTE en opérande gauche,
- les opérateurs $=, <, >$,
- les valeurs 1, 2, AS,AO, ou $<$ *variable* $>$ en opérande droite.[68]

L'opérateur $\wedge$ (et logique) est parfois utilisé pour connecter plusieurs opérations logiques.

If structuré La séquence qui traite la culture associée, bloc *sinon* si la condition logique désigne la culture dominante, bloc *alors* dans le cas contraire, est supprimée : les instructions conditionnelles sont éditées en commentaires.

If logique L'instruction est également éditée en commentaire : le branchement en avant est ainsi supprimé. La condition logique est supposée désigner la culture associée : un avertissement est édité dans le cas contraire.[69]

[68] La condition (IPL $<$ NBPLANTES) n'est vraie que si (NBPLANTES $>$ 1).

[69] Le saut ne s'effectue pour la culture principale que dans le sous-programme CUMAOETAS, avec en sus un branchement critique à l'extérieur d'un bloc conditionnel.

3.7.3 Ruptures de séquences

Les ruptures de séquences sont introduites principalement par l'instruction GOTO, laquelle entraîne un branchement, conditionnel ou inconditionnel, vers une instruction finale repérée par une étiquette. Les origines et arrivées des branchements délimitent des blocs d'instruction mais il suffit de quelques branchements imbriqués pour entraîner une explosion combinatoire du nombre de blocs, et de la complexité : un message d'erreur est alors édité, et un point d'interrogation ajouté dans le code C produit afin de générer une erreur à la compilation.

L'instruction non finale RETURN, retour de sous-programme ou fonction, est équivalente à un branchement terminal : elle est cependant traduite telle quelle avec retour vers l'appelant et édition d'un avertissement.[70]

Contrôle d'intégrité Les blocs associés définissent des sections critiques contrôlées par les conditions logiques des branchements ou retours vers l'appelant.

Branchements en avant

Si un bloc ne contient ni instruction de branchement, ni étiquette utilisée par une instruction GOTO externe, son exécution est de fait controlée par l'expression logique associée à l'instruction conditionnelle.

Le branchement conditionnel

IF $<$ *formule* $>$ GOTO $<$ *etiquette* $>$
$<$ *bloc* $>$
$<$ *etiquette* $>$ CONTINUE

est remplacé par l'une des structures suivantes

if $\neg <$ *formule* $><$ *bloc* $>$
ou
if $<$ *formule* $><$ *vide* $>$ else $<$ *bloc* $>$,

le composant $<$ *vide* $>$ désignant l'instruction neutre qui n'exécute rien.

Le branchement inconditionnel

GOTO $<$ *etiquette* $>$
$<$ *bloc* $>$
$<$ *etiquette* $>$ CONTINUE

définit des sections de code inaccessibles : elles sont néanmoins traduites pour d'éventuelles reprises et contrôlées par un prédicat toujours faux, soit

if (0) $<$ *bloc* $>$

Un avertissement est édité.

[70] Elle était avant la version 6 traitée comme un branchement. Son utilisation fréquente dans les nouvelles versions dans des blocs conditionnels générait alors de multiples imbrications, d'où la génération d'un retour vers l'appelant. Le code généré reproduit en conséquence ces sorties multiples, dont certaines dans des blocs internes : elles sont localisées par des avertissements.

Négation des conditions logiques La méthode la plus immédiate est d'utiliser l'opérateur unaire $\neg$ devant l'expression logique parenthèsée mais les formules sont peu lisibles.

Une autre méthode restructure la condition logique par application des règles suivantes aux opérations logiques binaires :[71]

1. l'opérateur de comparaison est remplacé par son symétrique, soit les couples $\{(==, !=), (\leq, >), (\geq, <)\}$;
2. les opérateurs de relation $\vee$ et $\wedge$ sont inversés, mais seulement si un seul de ces opérateurs est présent dans la condition.

Branchements en arrière

Les branchements en arrière traduisent une structure itérative contrôlée par un prédicat (1.3.2).

La boucle *Tantque* n'est pas présente dans le code fortran des versions actuelles mais est néanmoins reconnue pour les versions futures.

Les prédicats des boucles *Jusqua* sont construits à partir des prédicats de contrôle des blocs conditionnels,[72] et éventuellement ceux des branchements conditionnels. Il est vérifié que le branchement est l'ultime instruction des blocs *Alors* ou *Sinon*. Dans la négative, un avertissement est édité : le code est généré en séquence mais il devra être déplacé derrière le bloc itératif en cours.[73]

Imbroglio de branchements

L'utilisation simultanée des branchements avant et arrière pour gérer des blocs conditionnels et itératifs peut aboutir à une explosion combinatoire de séquences avec des prédicats de contrôle complexes, de multiples imbrications, et un code non structuré illisible.

Un mécanisme d'échappement a été mis en place pour traiter autrement ce type de code, les unités concernées étant repertoriées dans la liste *NonGoto*[]. Les étiquettes de branchement, qui traduisent en fait un état, sont rangées dans la variable locale *cible_* : le code généré est à reprendre au vu des branchements effectifs et de l'état courant, ce formalisme rendant plus lisible l'algorithme effectif (4.5.14).

Saut inconditionnel Si un saut inconditionnel est effectué en dehors de tout bloc conditionnel, il traduit en fait la non exécution d'une séquence associée à un branchement antérieur,[74] : la variable *cible_* est alors utilisée pour générer cette séquence conditionnelle sans branchement.

[71] L'opérateur $\neg$, '.NOT.' en Fortran, n'est utilisé que comme opérateur unaire devant les variables logiques.

[72] Une instruction vide est générée pour conserver les blocs, la condition et le branchement étant restitués en commentaires.

[73] Ce cas de figure se présente une seule fois dans la version 6 avec la mise à jour de SOMSENRESTE dans LIXIV.

[74] Cf sous-programme TESTTRANSRAD (4.3.7).

3.7.4 Choix multiple

Le choix multiple (1.3.1) apparaît dans les sous-programmes LECSORTI et LECRAP avec quelques centaines de tests et un branchement vers l'étiquette finale 10 : il est traduit en langage C par l'instruction ELSE IF.

Listes d'édition Les noms des variables des listes d'édition sont lus dans des fichiers puis recherchés dans des tables initialisées par des instructions Fortran DATA. Les valeurs des variables choisies sont affectées dans les tampons d'édition dans une boucle à l'issue d'un choix multiple, tel que décrit en 1.3.1.

3.7.5 Affectations statiques

Plusieurs sous-programmes permettent de choisir des variables *Stics*, variables simples, éléments de tables, voire expressions[75], pour les affecter dans des tables à des fins d'édition pour LECRAP et LECSORTI, d'optimisation pour une liste de paramètres de la simulation dans LECOPTIM. Les variables choisies ont préalablement été recopiées dans des fichiers prédéfinis gérés au moyen de l'interface graphique WINSTICS.

Ces affectations sont effectuées de façon statique dans des séquences conditionnelles avec énumération de centaines de choix et désignation en clair des variables *Stics* dans le code source, soit plusieurs milliers de lignes de code Fortran : la prise en compte de nouvelles variables nécessite alors d'ajouter une instruction conditionelle explicite dans le sous-programme concerné puis de recompiler le code Fortran.

L'existence des dictionnaires dans la traduction C permet de paramétrer ces affectations à partir des références des variables évaluées dynamiquement à travers le mécanisme d'allocation.

3.8 Les entrées et sorties

Les entrées et sorties de données depuis et vers des fichiers s'effectuent à travers des unités logiques et utilisent des formats Fortran, lesquels décrivent des champs d'enregistrements, mis en correspondance avec des listes de variables (1.3.6). Les instructions se présent sous la forme générique suivante :

READ($< unite >< format >$) $< liste\ lecture >$
WRITE($< unite >< format >$) $< liste\ edition >$

Le composant $< unite >$ se dérive en $< nombre >$ ou $< variable >$ pour identifier le numéro de l'unité logique : le caractère $*$ est quelquefois utilisé et paraît désigner l'entrée ou la sortie standard.

[75] par exemple $str1/nst1$.

Lecture Les lectures des paramètres et conditions initiales s'effectuent souvent dans des fichiers Fortran temporaires, créés à partir des fichiers gérés par l'interface WINSTICS : le processus de lecture est reconstitué au moyen de fonctions C qui traitent le flot binaire. Les unités logiques ne sont pas reprises. [76]

Edition Les éditions utilisent des formats C équivalents ou proches, ce qui introduit quelques variations dans la mise en page des bilans. Les unités logiques Fortran sont remplacées par des pointeurs C vers les fichiers.

3.8.1 Formats des enregistrements

Le format décrit colonne par colonne l'information élémentaire codée sur un caractère dans un enregistrement de longueur fixe, paramètre dit DCB (data controle bloc) des langages de commandes des anciens systèmes informatiques centraux. Des caractères non imprimables, comme la tabulation horizontale ou le retour chariot, sont également présents, d'où la nécessité de traiter les fichiers comme des flots d'octets pour réappliquer les conventions effectivement adoptées.

Les formats utilisés dans le logiciel *Stics* décrivent des nombres et des tables de caractères. Ils contiennent aussi des messages en clair et des directives de mise en page pour l'édition.

Les formats sont édités par la commande **S5**, séquentiellement pour toutes les instructions qui les utilise, respectivement en lecture et édition (1.3.6).

Syntaxe générique d'un format

Les formats *Stics* sont finalement produits par les seules règles de dérivation suivantes :

$< format > \longrightarrow < groupe > \{, < groupe >\} \mid * \mid / \mid \$$
$< groupe > \longrightarrow [< fois > X,][< fois >] < droite >$
$< groupe > \longrightarrow [< fois >](< groupe >)$
$< droite > \longrightarrow < description > \mid (< groupe >)$
$< description > \longrightarrow < type >< taille > [. < precision >]$
$< description > \longrightarrow ' < texte > '$
$< type > \longrightarrow$ A | I | F | E

Le composant $< groupe >$ introduit une règle récursive pour la répétition de séquences notées entre parenthèses.

$< fois >$ génère le nombre de répétition de l'élément situé à droite,
$< taille >$ fournit le nombre total de colonnes affectées pour l'élément,
$< precision >$ précise le nombre de chiffres fractionnaires.

[76] Les accès aux fichiers s'effectuent à travers les unités logiques 12 pour les paramètres techniques, généraux et de la plante, les valeurs étant recopiées vers l'unité logique 36 puis relues, 3 pour les paramètres du sol, FICSTA pour les données climatiques, FICUSM pour les paramètres initiaux, 12 pour les données de la culture précédente.

Le caractère $*$ correspond au format libre, dans lequel les valeurs numériques ou textuelles sont séparées par des espaces, la ligne définissant l'enregistrement.

La barre oblique (/) traduit le passage à la ligne.

Le dollar ($) en fin de ligne indique qu'il ne faut pas passer à l'enregistrement suivant.

Le composant $< texte >$ est construit avec tous les caractères : l'apostrophe est doublée pour ne pas être confondue avec le délimiteur.

Interprétation des formats

Les formats fixes Fortran matérialisent les colonnes des anciennes cartes perforées en lecture et des listes d'impression. Les formats libres ont ensuite été introduits pour reconnaître les valeurs séparées par des espaces.

Tout nouvel ordre de lecture ou d'édition provoque le passage à l'enregistrement suivant : la liste de variable peut être vide ou incomplète eu égard au nombre de champs décrits dans le format.

Formats fixes Les formats possibles sont relativement nombreux mais seuls les formats qui suivent sont actuellement utilisés.

- k`X` signifie qu'il faut ignorer n colonnes.
- `I`n code un nombre entier sur n positions, soit $n-1$ chiffres et le signe.
- `F`$n.m$ représente un nombre réel sur n colonnes. Si le point décimal est présent, il délimite les parties entière et décimale. Dans le cas contraire, les m positions de droite forment la partie décimale : il reste $n-m$ chiffres pour la partie entière et le signe.
- `E`$n.m$ reprend cette forme mais avec exposant (`e+xx` ou `e-xx`) : elle n'est pas reprise en C et traduite comme le format `F` précédent.[77]
- `A`n correspond à un tableau de n caractères.

L'espace est confondu avec le zéro à gauche ou à droite du nombre. Le signe est en première position et le blanc peut remplacer le signe $+$.

Les formats d'édition contiennent souvent des chaînes de caractères entre quotes, avec doublement de la quote pour introduire l'apostrophe.

Format libre Le format libre, introduit par le caractère *, permet de reconnaître des nombres ou des mots séparés par des espaces, tabulations ou sauts de lignes.

En plus des représentations associées aux formats fixes précédents, la notation avec exposant est aussi utilisée, soit par exemple *2.3e-5* pour 2.3^{-5}.

3.8.2 Listes des entrées-sorties

La liste des entrées-sorties suit les instructions READ et WRITE et permet d'énumérer les valeurs décrites par le format et transcrites dans les champs des

[77] Ce format n'apparaît finalement que pour transmettre des données de la culture précédente vers la suivante avec des ordres de grandeur qui ne justifient pas ce mode de représentation.

enregistrements. Elle énumére des variables simples, des éléments de tables, des constantes numériques ou textuelles ou des expressions pour les listes d'édition.

Une liste vide peut correspondre à l'édition d'un texte défini avec un format fixe ou un saut d'enregistrement.

Génération des listes

Les listes de lecture désignent les variables et éléments de tables qui reçoivent les valeurs. Les listes d'édition peuvent contenir en sus des valeurs et des expressions. Les grammaires génératrices de ces listes reprennent les composants des sections 3.4 et 3.5.

Lecture Les listes de lecture sont générées par les dérivations syntaxiques :

$< liste\ lecture > \longrightarrow < element > \{, < element >\}$
$< element > \longrightarrow < variable > \mid < boucle\ implicite >$
$< boucle\ implicite > \longrightarrow (< variable >, < nom > = < borne >, < borne >)$
$< borne > \longrightarrow < nombre > \mid < variable > \mid < expression >$

La boucle implicite est un autre mode d'écriture des boucles utilisé en lecture, comme dans les exemples suivants :

READ(12,620)(HINIT(I),I=1,5)
READ(12,635)(CRES(I,IR),IR=1,8),(CBIO(I,IR),IR=1,8),(NBRO(I,IR),IR=1,8)[78]

$< nom >$ génère la variable locale de la boucle implicite (2.1.1).

Si $< borne >$ produit un nombre, les bornes de la table sont connues et les formats correspondants peuvent être contrôlés.[79] Dans le cas contraire, les bornes sont évaluées en cours d'exécution et les correspondances entre types et formats ne sont plus vérifiables.

Dans cette syntaxe, une seule table apparaît dans la boucle implicite alors que le langage Fortran permet de citer des listes de tables et d'autres boucles : ces structures syntaxiques n'ont pas été traduites et leur présence se solde par un arrêt du traitement de la liste d'entrées-sorties, comme pour les erreurs.

Edition Les listes d'édition reprennent les règles de production des listes de lecture mais d'autres dérivations du composant $< element >$ décrivent l'édition de valeurs et expressions numériques ou textuelles, soit

$< element > \longrightarrow < nombre > \mid < chaine > \mid < expression > \mid (< expression >)$

Cette dernière syntaxe introduit une ambiguïté pour un élément de liste qui débute par une parenthèse ouvrante, boucle implicite ou expression.[80]

[78] IR est la variable de la boucle implicite et I la variable d'une boucle DO externe.

[79] Pour la table DATE du sous-programme LECLAI, un élément est lu en I4 et les deux suivants le sont en I3.

[80] Les expressions "`(1.-GEL1(IPL))*100.,(1.-GEL2(IPL))*100.`", utilisées pour l'édition des bilans, ne sont pas reconnues.

Correspondance entre valeurs et variables

Les formats sont définis respectivement pour les types entiers, réels et textuels : il y a lieu de vérifier que ces formats correspondent aux types Fortran reconnus, déclarés explicitement ou implicitement.

Boucles implicites Les traitemets effectués diffèrent selon le type de format.

Format fixe Le nombre de champs décrits doit être au moins égal au nombre de pas : les bornes de l'intervalle sont en conséquence des nombres, et non des variables. Si cette boucle implicite termine la liste, les champs non utilisés sont ignorés.

Format libre Il n'est plus nécessaire de connaître le nombre de pas : les bornes peuvent alors être des variables et la lecture s'effectue tant qu'il reste des éléments dans la liste et des valeurs dans le flot d'entrée.

3.8.3 Edition des messages d'erreurs

Des ordres d'édition sont présents dans n'importe quelle unité pratiquement pour signaler des erreurs ou anomalies, vers la sortie standard ou dans le fichier dit historique : les messsages à éditer sont affectés dans des variables textuelles de la forme *MESxx* dans un sous-programme d'initialisation.

Dans la traduction C, ces variables ne sont pas utilisées :[81] les textes sont repris dans une fonction C issue de la traduction du sous-programme d'initialisation MESSAGE, qui sélectionne la chaîne à éditer.[82]

Editions complémentaires Les ordres d'édition sont quelquefois complétés par des informations contextuelles : ces instructions ne sont pas traduites en dehors des sous-programmes d'édition ou de lecture, la table des codes textuels, et parfois les vecteurs requis, n'étant pas nécessairement accessibles dans la fonction C.
Le message est néanmoins édité si la variable textuelle est citée en début de liste.

Génération de la fonction d'édition des messages

La présence d'une affectation d'un texte long dans une unité de programme initialise une instruction de choix multiple, avec comme valeur de la variable de choix le rang de la variable textuelle : la valeur textuelle est utilisée pour générer l'ordre d'édition associé.[83] Ces ordres sont ensuite générés pour toutes

[81] Les déclarations sont faites dans les fichiers `"message.inc"` et `"message`*xxx*`.inc"`, *xxx* étant un nom de module, inclus au début des unités de programmes : il s'agit de tables Fortran de tailles variables, d'au moins 40 caractères, de type *texte long* (3.1.1).

[82] Les quotes et guillemets qui figurent dans ces textes, utilisés comme caractères de contrôle par les compilateurs, sont supprimés.

[83] Ces affectations n'apparaissent finalement que dans les unités d'initialisation MESSAGEUNIX (version 5) et MESSAGE (version 6).

les affectations de textes longs avec les rangs des variables citées pour désigner les cas, l'instruction de choix multiple étant close à la fin du traitement de l'unité de programme.

Cette fonction est déclarée avec les paramètres formels :
- $*f_$, pointeur vers le fichier d'édition ;[84]
- $nmessage_$, rang du message ;
- $module$, nom de la fonction appelante ;

L'édition du message est précédée de l'identification de la fonction appelante.

Appels de la fonction d'édition des messages

La présence d'une variable de type *texte long* dans une liste d'édition, quelquefois à la place du format (1.3.6), est traduite par un appel de la fonction $Message(f, n)$, avec les paramètres effectifs :
- f, référence vers le fichier d'édition associée à l'unité logique Fortran (FICHIST ou $*$) ;
- n, numéro d'ordre du message, rang attribué lors de la lecture du dictionnaire des variables communes ;
- le nom entre guillemets de la fonction appelante, valeur textuelle ;

3.8.4 Fonctions de lecture des fichiers STICS

Les instructions C générées à partir des ordres Fortran de lecture contiennent des appels de fonctions développées sur mesure, intégrées dans le module de supervision de la section 4.2.

Les messages d'erreurs sont édités dans le fichier compte-rendu reçu en paramètre : la marque # au début du message indique la fin de fichier.

Contrôle du flot de caractères

La fonction

char *Suivant()*

est une primitive de niveau système qui lit le caractère suivant dans un fichier vu comme un flot binaire, utilisée par les autres fonctions de lecture. Elle rend le caractère suivant autre que le `retour chariot` des fichiers DOS, qui est ignoré, ou le caractère nul pour la fin de fichier : c'est la marque de fin de ligne qui signale la fin d'un enregistrement. Cette primitive gère des variables globales et pilote les appels de niveau système.

Lecture des valeurs numériques et textuelles

Les trois fonctions

int *Entier(*int *nsource*, FILE *f, int *k*, int *n*, char **nom*, int *fin)*,
float *Reel(*int *nsource*, FILE *f, int *k*, int *n*, int *m*, char **nom*, int *fin)*,

[84] C'est généralement le fichier STICS dit historique, repris comme fichier-compte-rendu.

int *Texte(*int *nsource*, FILE *F, char **cible*, int *k*, int *n*, char **nom*,
int *fin)*,

décodent les champs des enregistrements localisés à travers les formats et affectent les valeurs dans les variables des listes de lecture. Elles éditent dans le fichier compte-rendu le nom de la variable de la liste de lecture, reçu en paramètre, suivi de la valeur lue.

Listes de paramètres Trois paramètres sont utilisés pour l'édition des compte-rendus :

- *nsource*, rang de l'instruction Fortran utilisé comme code d'erreur, négatif pour un déroutement ;
- f, pointeur sur un fichier compte-rendu ;
- *nom* désigne la variable courante dans la liste de lecture ;

Les longueurs des champs des formats fixes sont transmis en paramètres effectifs des fonctions appelées, soit

- le nombre k de positions entre deux champs (format k`X`) ,
- la largeur n d'un mot ou de la partie entière d'un nombre (formats `A`n, `I`n ou `F`$n.m$),
- la largeur m de la partie décimale (format `F`$n.m$).

La valeur nulle du paramètre n correspond au format libre.

Les valeurs textuelles sont récupérées dans la variable *cible*.

L'ultime paramètre, fin, indique si une valeur à lire est la dernière d'une liste de variables, éventuellement vide, ce qui permet de simuler la fin de l'enregistrement Fortran par lecture du reste de la ligne.[85]

Lecture des valeurs numériques Les variables entières ou réelles simple précision sont lues dans une instruction d'affectation, avec en opérande gauche la variable lue et en opérande droite les appels des fonctions, soit par exemple

- $k = Entier(nsource, f, 1, 4, "k", 0)$;
  affectation dans la variable entière k du nombre lu dans le champ décrit par le format "1X,I4".
- $x = Reel(nsource, f, 1, 7, 3, "x", 1)$;
  affectation dans la variable réelle x du nombre lu dans le champ décrit par le format "1X,F7.3",
  et passage à la ligne suivante ($fin = 1$).

Lecture des codes textuels Les variables textuelles sont recopiées dans le paramètre *cible* à l'issue de l'appel de la fonction, soit par exemple

- $Texte(nsource, f, codan, 1, 3, "codan", 0)$;
  lecture d'un mot dans le champ décrit par le format "1X,A3" avec recopie dans la table *codan*[].

[85] Cette dernière valeur peut n'être connue que dynamiquement, par exemple si les bornes d'une boucle de lecture sont évaluées : un avertissement est alors édité.

– $l = Texte(nsource, f, codan, 0, 0, "codan", 0)$;
lecture en format libre avec affectation de la longueur du code lu dans l, ou $l < 0$ si fin de fichier.

Positionnement dans les fichiers WINSTICS

La fonction

int *Wstics(*int *nsource*, FILE *f, int *saut_)*

gère les problèmes des positionnements dans les enregistrements gérés par l'interface WINSTICS.

Il apparaît que les valeurs, numériques ou textuelles, sont précédées des noms des paramètres, codés à partir de la colonne 70 dans un champ de 20 caractères. Ces couples (`''nom''`, `''valeur''`) sont ensuite recopiés dans un fichier temporaire seulement si le nom de paramètre est effectivement présent.[86]

La fonction $Wstics()$ reprend empiriquement ces éléments pour accéder à la première position du champ dans lequel est codé la valeur, en simulant l'ordre de lecture Fortran en format libre sans liste de variable. Elle édite entre crochets dans le fichier compte-rendu le nom du paramètre ainsi lu.[87]

Le paramètre $saut_$ permet de gérer les sauts d'enregistrements liés au mode d'acquisition des paramètres pour des opérations techniques répétitives, la valeur 1 signifiant que l'enregistrement est à ignorer.

Une fin de fichier inattendue génère un déroutement et clôt l'exécution.

Opérations techniques répétitives Dans le sous-programme LECTECH, si la valeur `"nbinterventions"` est lue comme nom de paramètre, l'enregistrement qui suit n'est pas un couple $(nom, valeur)$ et un branchement supprime la séquence de recopie dans le fichier temporaire : la fonction $Wstics()$ rend alors la valeur 1, 0 sinon. Cette valeur est directement affectée dans la variable $saut_$, ensuite transmise comme paramètre dans l'appel suivant de $Wstics()$.

Traduction Cette fonction est appelée dans une instruction d'affectation avec en opérande droite la variable locale $saut_$ comme paramètre effectif, et en opérande gauche cette même variable, ainsi mise à jour pour l'appel suivant, soit par exemple

$saut_ = Wstics(nsource, f, saut_)$;

[86] Un nom est codé dans un champ de 19 positions contenant au moins un caractère autre qu'un espace : des ambiguïtés demeurent quant aux définitions de l'équivalence de deux chaînes de caractères dans le langage Fortran, ce qui peut se traduire par des traductions qui vont dépendre du compilateur utilisé.

[87] Ce nom n'est pas nécessairement celui de la variable de la liste de lecture : des variantes orthographiques existent et la valeur textuelle `"nbinterventions"` remplace les variables communes ($Naps, Nap, Napn$) et locales ($nbcoupe2, nbcoupe3$) qui dénombrent des opérations culturales.

Couples de valeurs pour les cultures associées et comptage champs

Les paramètres de la plante et certains paramètres techniques sont gérés sous forme de couples dans des variables *Stics*, la lecture s'effectuant dans une boucle avec une ou deux itérations, et la variable de boucle en indice. Un mode particulier de gestion de quelques couples est introduit pour des paramètres dits "nouvelle version", chaque élément du couple étant cité en clair avec les indices 1 et 2 dans les fichiers WINSTICS, la seconde valeur étant toujours lue, que la culture associée soit présente ou non.

Par ailleurs, le mode de lecture des fichiers WINSTICS est modifié, le nom du paramètre et sa valeur d'abord lus dans des variables textuelles locales puis décodées dans une longue séquence conditionnelle : un code spécifique a du être fabriqué pour traiter ces nouvelles formes dans LECPARAMV6.

Saut d'enregistrement

Une autre fonction de lecture est utilisée pour pouvoir ignorer le champ valeur s'il est néanmoins présent alors qu'il est sans objet :

int *SautChamps(*FILE **f, int fin)*

permet d'ignorer une suite de champs séparés par des espaces (4.3.4).

Le paramètre *fin* permet de demander le passage à la ligne suivante dans le fichier compte-rendu.

SautChamps() est utilisable pour un saut d'enregistrement lié à un ordre de lecture avec une liste vide, *Wstics*() étant réservé pour les fichiers WINSTICS.

Séries climatiques

Les fichiers climatiques contiennent un nombre variable de séries définies pour un nombre variable de périodes.

Nombre de séries Les instructions de lecture des séries climatiques dans LECSTAT citent une liste de 7 séries alors que des fichiers ne contiennent les valeurs que pour 5.

Ces instructions sont reprises sous forme conditionnelle, la valeur 1 étant alors affectée pour l'indice 0 de la série dans la fonction de supervision de la lecture (4.2.5) : *SautChamps*() est appelé pour l'ultime série avec $fin = 1$ pour terminer la lecture en format fixe et éditer la marque de fin de ligne.

Séries annuelles incomplètes La dernière période lue est notée dans la variable JULFIN, laquelle est ensuite reprise dans le sous-programme INICLIM pour réduire la période de simulation à cette date ultime dans la variable MAXWTH.

3.8.5 Numérisation des variables textuelles

Des variables textuelles sont utilisées pour évaluer des conditions logiques en cours de simulation (2.1.5). Elles sont remplacées dans le code généré par des

variables numériques, le tiret souligné (_) étant ajouté à droite du nom de la variable textuelle. Les valeurs citées dans le code, dans des expressions logiques ou des affectations, sont reprises dans une liste, l'espace étant remplacé par le tiret souligné pour reconstituer des mots.

Deux fonctions sont générées :

1. int *NumCode(* char **code*) consulte la liste des codes cités en clair dans les expressions logiques et rend le rang de la valeur textuelle ;
2. char **ValCode(* int n) rend le code de rang n ;

Les valeurs textuelles lues dans les fichiers de paramètres sont ainsi numérisées.

3.8.6 Déroutements en lecture

Les ordres de lecture contiennent comme directives de contrôle, optionnelles, des étiquettes de branchements pour les fins de fichiers et les erreurs (1.3.3). Ces étiquettes gèrent des branchements après la séquence de lecture, avec généralement une édition dans le fichier historique suivie d'un STOP fortran qui clôt l'exécution. En l'absence de branchement lié à une directive, un saut inconditionnel après l'ultime ordre de lecture permet de poursuivre les traitements.

Ce mode de contrôle génére des sorties des boucles de lecture en fin de fichier, alors que les itérations sont en cours. La solution adoptée pour pouvoir néanmoins contrôler les opérations de lecture utilise le code de retour de la fonction $Texte()$, avec un choix conditionnel si la longueur l n'est pas négative.

La solution utilise le fait que ces boucles finales lisent en pratique un code textuel parmi les variables numériques : les impossibilités de lecture sont néanmoins signalées mais elles sont limitées aux variables qui suivent dans l'itération en cours.

3.8.7 Traduction des instructions d'édition

Des sous-programmes spécialisés éditent des résultats et regroupent de nombreuses instructions d'écriture avec des formats de mise en page.

Quelques ordres d'édition autres que les messages dans le fichier historique sont présents dans des sous-programmes appelés dans la boucle quotidienne : ils sont finalement ignorés et placés en commentaires derrière une instruction vide (4.2.3).[88] Les écritures dans les fichiers non ouverts sont également ignorées.

Gestion des fichiers

Les éditions s'effectuent à travers des références de fichiers C qui reprennent les unités logiques Fortran (1.3.6), avec les conventions de nommages suivantes :

– un nombre est préfixé par *F_ (F_12)* ;

[88] Le problème se pose pour le sous-programme SORTIE : ces instructions sont à déplacer dans un sous-programme BILSORTIE. D'autres autres ordres utilisent l'unité logique FICDBG.

– un nom est transcrit avec la première lettre majuscule, les autres minuscules *(Ficsort)* ;

Ces références sont ajoutées dans la liste des paramètres formels de la fonction.

La gestion des fichiers n'est pas traduite : les ordres d'ouverture et fermeture figurent toutefois en commentaires d'instructions vides.

Listes d'édition et formats

Une instruction C est générée pour chaque élément d'une liste d'édition. Les champs descriptifs des formats d'édition sont recherchés pour les types de données de la liste, et traduits : l'ultime valeur est éditée avec le reste du format et la marque finale de fin de ligne.

Les formats d'édition décrivent des champs définis pour les types de variables et gèrent les espaces et sauts de lignes. Les instructions d'édition Fortran sont traduites telles quelles en langage C mais la conversion des formats d'un langage à l'autre peut introduire quelques décalages dans la mise en page finale.

Les formats Fortran I, F, E, A, pour les types entiers, réels et textuels sont traduits avec les caractères de conversion correspondants du langage C qui suivent la marque %. Les largeurs des champs ne sont pas reprises : les valeurs par défaut de la bibliothèque standard sont retenues. [89]

Des espaces séparateurs sont introduits entre les champs consécutifs s'ils n'ont pas été explicitement spécifiés.

Les répétitions de champs, simples ou placés entre parenthèses, sont directement appliquées dans le format C, ce qui peut allonger certains formats.

Le saut de ligne, introduit par la barre oblique /, est généré.

Les caractères interprétés dans un format C sont précédés du caractère d'échappement.

La fin de ligne finale est ajoutée pour simuler la fin d'enregistrement, sauf si la marque $ est présente en fin de format Fortran.

Variables textuelles Les textes long sont interprétés comme des messages et édités comme tels (3.8.3).

Les variables contenant des codes textuels ne sont édités que dans l'étape d'acquisition des données externes, les instructions d'édition étant placées en commentaires pendant l'initialisation et les évaluations quotidiennes, effectuées par des fonctions qui n'admettent que des vecteurs en paramètres.

Fichiers d'édition

Un compte-rendu est édité dans un fichier dit historique géré par l'unité logique FICHIST.

Les autres unités logiques Fortran sont les suivantes :

Version 5

[89] La forme A<LONG1> est reconnue et traduite comme une chaîne C de longueur variable.

- FICSORT : édition quotidienne d'une liste prédéfinie de variables, pilotée par SORTIE ;
- FICSORT2 : édition quotidienne de la liste VALPAR d'au plus 20 variables lues dans le fichier `"var2.mod"`, pilotée par SORTIE, à partir du tampon VALSORTIE initialisé par appel de LECSORTI ;
- FICRAP : édition quotidienne et périodique de la liste VALRAP d'au plus 40 variables lues dans le fichier `"rap.mod"`,[90] pilotée par RAPPORT, à partir du tampon VALSORTIERAP initialisé par appel de LECRAP ;
- FICBIL : édition depuis BILAN et BILCOUPE les jours de récolte de résultats avec libellés et valeurs ;
- 12 : récupération des résidus de récolte ;
- 10 : profil du sol contrôlé dans LECPROFI, sauvegardé dans TABPROF et édité par PROFIL ;
- FICMICROCLI ou 16 : données quotidiennes horaires éditées sous le contrôle de HUMHEURE ;
- 36 : copies dans des fichiers temporaires non reprises.

Version 6

- FICBIL(IPL), FICSORT(IPL), FICSORT2(IPL) et FICRAP(IPL) sont des tables à deux éléments ;
- FICBIL(IPL) est affecté dans FB pour l'édition des bilans ;[91]
- FICDBG et 55 sont utilisés dans SENESCEN et SORTIE

Les noms des unités logiques sont édités en commentaire derrière l'ordre d'écriture pour pouvoir rechercher ces instructions dans le code généré, soit par exemple pour le sous-programme SORTIE appelé dans la boucle quotidienne "WRITE(FICSORT(IPL))" (4.5.17).

Numérisation des éditions

Pour rester dans un univers numérique et supprimer les tables de codes transmises en paramètres, une solution est de ne pas attribuer le rôle d'édition 200 (2.4.1) aux unités retenues, pratiquement BILAN et BILCOUPE pour la restitution des bilans après récolte.

La table *Code_*[] nétant plus transmise, les éditions utilisent la fonction $ValCode()$ qui restitue les valeurs textuelles à partir des images numériques (3.8.5) : si l'image n'est pas définie, la valeur par défaut `'xxx'` est restituée.

3.9 Production du code

Le traitement de nouvelles versions peut nécessiter de modifier certaines constantes comme pour la commande $S6$ (2.7), à la suite d'un message explicite ou d'une interruption du système d'exploitation.

[90] Une lecture initiale est effectuée le premier jour de la simulation depuis LECSORTI pour générer l' entête de ce fichier.

[91] Le sous-programme DOMINVER effectue aussi l'échange des unités logiques lors de l'inversion de dominance.

Si le code source de la commande $S8$ est contenu dans le fichier `"S8.c"`, les protocoles d'allocation des variables communes et les fonctions de suivi associées sont gérés dans le module `"Sallouer.c"`, les déclarations des constantes et prototypes des fonctions étant quant à elles regroupées dans le fichier `"Allocation.h"`

Il est possible de repérer des constantes qui peuvent varier entre versions ou applications, telles les durées ou profondeurs, bornes supérieures de tables : il suffit de modifier dans la primitive $Ajuster()$ l'initialisation de la table entière des équivalences $*equi[2]$, vide par défaut (3.1.2).[92]

Un commentaire est ajouté en sus dans la liste des constantes pour toute instruction qui contient ces valeurs modifiées. Un avertissement est également édité si l'ancienne borne supérieure d'une table dont la taille a été ainsi modifiée est citée en clair.[93]

3.9.1 Génération du code C et Fortran

La commande

S8 *stics* [−*option*]

édite le code C dans les répertoires fils C et D,
et le code Fortran dans le répertoire fils F.

Appel de la commande

La commande $S8$ lit un ensemble de fichiers.

Directives Trois fichiers optionnels contrôlent la génération du code :

- les deux premiers gèrent des listes d'unités de programmes :
  "*stics.non*" : le code C n'est pas généré ;
  "*stics.nongoto*" : les instructions GOTO ne sont pas interprétées (3.7.3) ;
- le troisième concerne le mode de vectorisation des variables communes :
  "*stics.table*" : modification des bornes supérieures (3.10.2) ;
  gestion de séries sur une durée donnée (mode retro) ;

Fichiers descriptifs Les autres fichiers lus sont produits dans les étapes antérieures, soit

"*stics.lex*", lexique des fonctions *Stics*,
"*stics.fils*", matrice de l'arbre des appels de ces fonctions,
"*stics.f*", correspondance des fonctions intrinsèques Fortran et C,
"*stics.com*", dictionnaire des variables communes retenues,

[92] La modification de la profondeur du sol, réduite de 1000 à 200 centimètres dans la version *Hysafe*, est contrôlée avec l'introduction du couple {1000, 200}, les oublis étant signalés. L'application de ce mécanisme pour la gestion dynamique du nombre de types de résidus est plus difficile, les tables correspondantes étant définies pour 8 ou 10 types, tailles retenues pour d'autres typologies.

[93] $Tsol[]$ et $Tsolveille[]$ demeurent définis jusqu'à la profondeur 1000 et sont initialisés comme tels dans le sous-programme INICLIM.

"*stics.loc*", dictionnaire des variables locales,
"*stics.pro*", description formelle des prototypes des fonctions,
"*stics.do*", description des boucles DO,
"*stics.goto*", description des ruptures de séquence,
"*stics.sis*", description des structures conditionnelles,
"*stics.cri*", localisation des sections critiques,
"*stics.graphe*", graphe des relations entre variables communes,
"*stics.tag*", liste des codes utilisés en cours de simulation,
"*stics.inc*", liste des fichiers inclus,
"*stics.format*", liste des formats de lecture et édition,
"*stics.unit*", table de correspondance entre unités logiques Fortran et pointeurs de fichiers C (3.8.7),
"*stics.ouvert*", localisation des séquences de lecture,
"*stics.couple*", rangs des instructions de contrôle relatifs aux couples de cultures (1.3.7),
"*stics.module*", "*stics.module.for*, "*stics.module.inc* : utilisés pour établir la correspondance avec les modules sources Fortran (1.1.7), localiser les avertissements et effectuer les validations (4.6).

Les instructions à décoder sont lues dans "*stics.ins*" pour les unités retenues, les homonymes de fonctions intrinsèques et les unités vides étant exclus.[94]

Comme pour la commande *S6*, les erreurs importantes se soldent par un déroutement qui clôt l'exécution. Les formes non reconnues sont signalées par un avertissement et le reste de l'instruction est ignoré, avec souvent édition d'une marque pour générer une erreur à la compilation du code C produit.

Options Quelques options sont accessibles au moyen d'un ultime argument de la ligne de commande.

- `-A` génère un code qui utilise la fonction *Age*() pour gérer les périodes antérieures (3.2.6).
- `-F` permet de marquer les avertissements pour en contrôler l'édition.

Fichiers produits

L'exécution génère le code dans des répertoires fils et produit des fichiers dans le répertoire courant.

Code C des unités de programmes Le code C généré à partir du source Fortran édité dans des fichiers sous le répertoire fils C, dans le fichier `"Unite.c"` pour une unité de programme Fortran UNITE,[95] ou dans des fonctions `"Stics_`*n*`.c"` pour les unités anonymes.

[94] Les sous-programmes non repris dans la version C sont remplacés par des unités de programmes qui exécutent immédiatement un RETURN, ceci pour pouvoir effectuer l'édition de liens Fortran sans devoir mettre les appels en commentaires dans le code source.

[95] Les fonctions C refabriquées, comme *Julien*(), sont générées mais avec le suffixe `".n"`.

Code C annexe Le code C supplémentaire est édité sous le répertoire fils D.

1. `"DicStics.c"` initialise le dictionnaire des variables *Stics* pour établir la correspondance avec les composantes des vecteurs (3.2.7) ;[96]
2. `"DicStics.adr"` fournit les adresses de toutes les variables au moyen de pointeurs C, types **float* pour les vecteurs et **char* pour les codes textuels, avec en commentaires les tailles des tables et, pour les variables Fortran entières et logiques le type pour signaler l'opération de conversion.
   Les déclarations peuvent être reprises dans des fonctions C pour pouvoir utiliser les variables STICS.
3. `"AdrStics.h"` fournit les mêmes adresses mais en introduisant le nom du tampon comme préfixe du nom de variable.[97]
4. Les déclarations des vecteurs sont faites dans `"Vectorisation.h"`.
   `"Tailles.h"` associe des noms symboliques aux tailles des tables.
   `"IniVecteurs.h"` contient le code d'initialisation.
   `"Rang.h"` regroupe des constantes de localisation des niveaux d'allocation dans les vecteurs (3.2.5).
5. `"NumCode.c"` et `"ValCode.c"` contiennent les fonctions $NumCode()$ et $ValCode()$ utilisées pour traiter les variables textuelles (3.8.5) ;
6. `"IniY.c"` gère les réinitialisations si les valeurs des paramètres du modèle, ou données initiales,[98] sont perdues lors des évaluations quotidiennes (3.2.4) ;
7. `"CultSuiv.c"` et `"CultPrec.c"` gèrent la transmission du vecteur pour la rotation des cultures (3.2.3) ;
8. `"SynchroRetro.c"` effectue les décalages avant les évaluations du jour pour les séries gérées en mode retro (3.10.3) ;
9. `"Proto.h"` contient la liste des déclarations des prototypes des fonctions ;
10. `"Acquerir.c"` est un double de toutes les instructions de lecture, les noms des unités de programmes étant ajoutés en commentaires ;[99]

Code Fortran annexe Les sous-programmes de suivi générés pour validation (3.2.7) sont édités sous le répertoire fils F.

Comptes-rendus

`"stics.S8"` contient le compte-rendu de l'analyse et de la génération du code ;
`"stics.cr8"` énumère les vectorisations multiples pour les variables lues et évaluées ;

[96] Cette méthode n'est plus utilisée et c'est la fonction fabriquée $DicoStics()$ qui génère le dictionnaire.

[97] On distingue ainsi la variable du modèle $Y_Profhum$, la valeur initiale $X_Profhum$ et le paramètre du sol $S_Profhum$.

[98] *Resperenne0*, lue avec les paramètres initiaux, est réévaluée.

[99] Ce code source n'est pas utilisé : il permet néanmoins de localiser dans quelles fonctions sont lues les variables.

Listes annexes Deux listes sont éditées, soit

"*stics.dec*", constantes numériques autres que $\{0, 1, 2\}$ citées en clair dans les instructions,[100],

"*stics.ombre*", termes qui font référence à la culture associée ou la zone ombragée (3.2.1).

3.9.2 Code des fonctions C

Pour chaque unité de programme Fortran, une fonction C est créée et éditée dans un fichier dont le nom est celui déclaré dans le code Fortran, transcrit avec la première lettre majuscule et les autres minuscules. Si plusieurs exemplaires sont présents, le rang est introduit à droite derrière un tiret souligné (_).

Les fonctions sont immédiatement compilables :[101] les éventuelles erreurs de compilation peuvent traduire des problèmes résiduels localisés mais non traités dans la phase de génération du code.

- Un point d'interrogation est ajouté en lieu et place d'une instruction (? ;) pour localiser des situations critiques, souvent liées à des branchements.
- Les variables non initialisées mais référencées apparaissent comme non déclarées car non retenues dans le dictionnaire des variables communes.
- Les fonctions C étant entières par défaut, une incompatibilité de type apparaît là où des valeurs textuelles sont attendues.
- Une incompatibilité des dimensions des opérandes d'affectation génère une instruction erronée.
- ...

Prototypes des fonctions Les prototypes des fonctions sont établis lors de la génération du code C en fonction des vecteurs requis pour gérer les variables *Stics* utilisées : déclarations et appels sont gérés en conséquence.

Si ce mode de gestion permet la traduction automatique de l'ensemble des unités d'évaluations quotidiennes, il nécessite des corrections pour les unités d'acquisition des données, d'initialisation et d'édition, lesquelles sont reprises dans le module de supervision de la version C (A.4).[102] Afin d'éviter ces corrections, les fonctions générées pour ces unités sont déclarées avec l'ensemble des vecteurs accessibles au cours des évaluations quotidiennes.

3.9.3 Compte-rendu

Le fichier `"stics.S8"` contient, pour chaque fonction analysée, les avertissements édités lors de la génération du code, identifiés par le rang de l'instruction suivi du code d'erreur entre crochets.

La classification des erreurs permet globalement de les resituer dans les instructions et composants syntaxiques où elles surviennent, et plus précisément

[100] Cette liste est utilisable pour définir en entête des modules les constantes mathématiques ou physiques, comme le nombre π initialisé différemment dans différentes unités de programme.

[101] La commande `"gcc -c F.c"` compile la fonction $f()$ et produit le code objet `"F.o"`.

[102] Les unités Fortran qui génèrent ces fonctions sont repérées par les préfixes LEC, INI et BIL.

dans la chaîne de traitement de ces composants, les fonctions appelées héritant des codes d'erreurs.

Une valeur négative de ces codes se solde par un déroutement qui clôt immédiatement l'exécution après édition de l'ultime message d'erreur.

Reconnaissance des déclarations et instructions Le plan de classement adopté pour coder les erreurs de l'analyse syntaxique des déclarations et instructions Fortran est actuellement le suivant :

[100, 199] : reconnaissance et prétraitement ;
- [100, 109] : reconnaissance de blocs d'instructions ;
- [110, 119] : instruction conditionnelle ;
- [120, 129] : rupture de séquence conditionnelle ;
- [130, 139] : branchement ;
- [140, 149] : retour de fonction ;
- [150, 159] : appel de sous-programme ;
- [160, 169] : appel de fonction ou opération d'affectation ;
- [170, 179] : initialisation DATA ;
- [180, 189] : boucle DO ;
- [190, 199] : fin de sous-programme ou fonction ;

[200, 299] : déclaration des variables locales ;

[300, 399] : reconnaissance des instructions conditionnelles ;

[400, 499] : traitement des opérations d'affectations ;

[500, 599] : traitement des appels de fonctions ;

[600, 699] : traduction des ordres de lecture ;

[700, 799] : traduction des ordres d'édition ;

[800, 899] : déclaration des fonctions ;

[900, 999] : reconnaissance des boucles à pas constant ;

[1000, 1099] : traitement des branchements explicites ;

Expressions entières Des expressions numériques entières sont évaluées comme indices des tables et à ce titre sont évaluées dans différentes instructions. Elles sont également présentes pour définir les bornes des boucles à pas constants.[103]

Les erreurs sont repérées par ajout de la valeur 20 au premier code associé à l'instruction qui contient l'expression, d'où les codes d'erreurs :

- [320, 329] : condition logique ;
- [420, 429] : affectation ;
- [520, 529] : liste de paramètres d'une fonction appelée ;
- [720, 729] : liste d'édition ;
- [920, 929] : boucle à pas constant ;

[103] La reconnaissance syntaxique n'est pas effectuée pour les listes de lecture, dans lesquelles les indices sont des nombres ou des variables simples, comme indiqué en 3.8.2.

Expressions arithmétiques Les expressions arithmétiques sont présentes dans les opérations d'affectation, les listes de paramètres des fonctions appelées et les listes d'édition.[104]

Elles sont repérées par ajout de la valeur 40 au premier code associé à l'instruction qui contient l'expression initiale, éventuellement retransmis lors des appels récursifs, d'où les codes d'erreurs :

[440, 449] : affectation ;
[540, 549] : liste de paramètres d'une fonction appelée ;
[740, 749] : liste d'édition ;

Instructions de lecture La classification a été affinée pour la traduction des ordres de lecture avec l'ajout de codes d'erreurs pour retrouver les spécificités Fortran, soit :

[650, 659] : extraction d'une variable d'une liste ;
[660, 669] : reconnaissance d'une boucle implicite ;
[670, 679] : extraction d'un format fixe ;

3.10 Dictionnaire des variables STICS

Le modèle *Stics* est décrit avec un formalisme qui utilise un grand ensemble de variables identifiées par des noms propres, reprises comme variables informatiques. La vectorisation fait que ces variables sont gérées dans des tables et, par défaut, n'apparaissent plus directement dans les expressions mais seulement dans les commentaires des instructions. Il est cependant possible de générer un code avec des expressions qui reprennent les noms de variables choisies, pratiquement pour rendre plus visibles certains traitements dans des applications.

Ces variables *Stics* sont gérées dans le code Fortran avec des tailles prédéfinies pour les tables, souvent maximales pour prévenir tout problème. L'espace total alloué devient conséquent et le nombre de simulations simultanées dans une application est vite limité par les ressources disponibles. Plusieurs solutions pour réduire cet espace sont adoptées, les unes inhérentes aux propriétés du modèle, les autres pilotées par des directives.

3.10.1 Désignation locale de variables communes

L'étape de traitement des noms symboliques introduit un mécanisme de gestion des variables avec copies locales de variables communes citées dans le fichier "*stics.copie*" (2.5.2).

L'utilisation de variables locales homonymes permet de générer un code lisible, pour éventuellement adapter ce code pour une application en citant directement les variables *Stics*.

[104] Elles n'apparaissent pas dans la grammaire de génération des expressions logiques de la section 3.5.3, une expression parenthèsée étant traitée comme une expression logique.

Les variables simples sont déclarées comme telles et les valeurs recopiées juste avant le retour vers l'appelant : il faut donc s'assurer que l'unité de programme n'appelle pas des sous-programmes, lesquels pourraient consulter ou mettre à jour ces variables communes, qui contiendraient alors les valeurs initiales chez l'appelant.[105]

Les tables étant déclarées comme pointeurs, ce problème ne se pose pas.

Gestion locale des séries évaluées quotidiennement

Des copies locales peuvent être faites pour les séries gérées dans des tables pour la durée de la campagne. Il en résulte un code beaucoup plus lisible avec les séries désignées en clair, par exemple $lai[t]$.

Construction de fonctions privées

L'ajout dans cette table de nouvelles fonctions permet d'en générer une trame avec les variables choisies déclarées et initialisées, restituées en fin d'exécution pour les variables simples.

3.10.2 Ajustement des espaces d'évaluation

La prise en compte des dimensions spatiales et temporelles pour des applications sur une région en longue période conduit à maîtriser l'espace alloué lors de la vectorisation, d'où la nécessite de pouvoir ajuster cet espace.

1. La taille des séries est d'abord réduite à la durée maximale des simulations, soit la campagne culturale.
2. Des directives sont lues dans le fichier "*stics.table*" pour pouvoir ajuster les tailles des tables.

Ces adaptations sont risquées au sens ou des débordements peuvent survenir à l'exécution : des solutions ont été adoptées pour prévenir ces situations, dont des appels de fonctions définies pour gérer ces situations (4.2.1).

Contrôle des périodes

La simulation d'une culture s'effectue pour une campagne au plus annuelle (≤ 366) mais qui peut s'étendre sur deux années calendaires : toutes les tables, dont les matrices, sont définies dans la version 6 sur deux années, soit au plus 731 jours.[106]

Ces périodes sont en fait gérées dans plusieurs espaces :

1. les paramètres du modèle sont définis dans le temps calendaire, avec :
   - des séries météorologiques annuelles ;
   - des dates d'interventions techniques, avec des rangs d'intervention pour les opérations répétitives ;

[105] Les fonctions Fortran n'accèdent pas aux variables communes dans les versions actuelles.
[106] Le nombre 366 demeure comme reliquat de la version 5 adaptée pour le projet *Hysafe*.

Les numéros des jours sont les rangs à compter du début de la première année, soit 1 à au plus 731 pour une campagne sur deux années calendaires.

2. les indices des variables du modèle gérées dans des tables sont les rangs, depuis 1 jusqu'à 366 au plus, des simulations quotidiennes.

Si cette représentation est conservée pour les paramètres, une réduction est effectuée pour les variables : les tables sont alors définies pour une durée annuelle, définie par la constante CAMPAGNE = 366.

Cette réduction ne pose pas problème lors des évaluations quotidiennes, contrôlées par la variable de boucle N dans le code Fortran, horloge calendaire $t_$ dans le code C, qui varient depuis 1 jusqu'en fin de campagne. En revanche, les dites variables sont quelquefois initialisées avant simulation, ou restituées en phase d'édition, avec des indices qui peuvent varier sur la période bisannuelle, et générer des débordements.[107]

Initialisation des séries L'indice est remplacé dans le code généré par l'appel de la fonction $Lim()$ avec comme paramètres cet indice et la durée de la campagne. Si la valeur de l'indice n'est pas supérieure à cette durée, $Lim()$ rend cette valeur, CAMPAGNE + 1, sinon, l'indice 367 étant alloué dans le code C pour pouvoir gérer cette situation.[108]

Ajustement des tailles des tables

Les bornes supérieures des tables sont des nombres qui désignent des espaces de définition : profondeurs du sol, nombre d'horizons, durées, calendriers d'opérations. Les directives d'ajustement permettent de choisir d'autres valeurs pour ajuster ces espaces et vectoriser les variables communes.

Le système sol-plante est traité par itération sur des couches élémentaires du sol de un centimètre et des profondeurs maximales : les variables utilisées sont déclarées avec la valeur retenue, actuellement 1000 pour la profondeur totale, souvent 60 pour la zone humifère.

Les durées sont normalement bisannuelles, soit 731, rarement annuelle, soit 366 : ces nombres sont utilisés pour reconnaître les séries gérées dans des tables lors des évaluations quotidiennes.

Les calendriers gèrent des dates d'intervention dans des tables de tailles variables, jusqu'à 300 pour les opérations d'irrigation et fertilisation.

Des conventions peuvent être adoptées pour choisir des nombres qui identifient les différents espaces.

Profondeurs du sol La profondeur maximale du sol n'est atteinte que si les horizons sont décrits jusqu'à dix mètres, ce qui est rare : la profondeur a d'ailleurs été réduite à deux mètres dans la version 5 pour *Hysafe*.

Une telle réduction peut désormais être demandée à travers la directive '1000 200', qui remplace la borne supérieure 1000 par 200 pour toutes les

[107] Il s'agit généralement d'initialisations à 0.

[108] L'indice 0 est utilisé dans les réinitialisations des cultures fourragères.

variables, communes ou locales, déclarées avec cette taille. Le remplacement est aussi effectué pour les bornes supérieures des boucles DO Fortran (1.3.2), lesquelles contrôlent alors une initialisation.[109]

Exploration des couches du sol L'indice est remplacé dans le code généré par l'appel de la fonction *Sol*(), avec comme paramètres le rang de la fonction C, l'indice et la borne. La valeur de l'indice est rendue si elle ne dépasse pas la borne, une erreur à l'exécution est générée sinon, et c'est la valeur de la borne qui est rendue.

Reconnaissance des espaces Les différents espaces sont actuellement repérés par les nombres :

- 731 et 366 pour les séries (on choisirait 367 s'il fallait gérer un calendrier opérations quotidiennes) ;
- 1000 et 200 pour la profondeur du sol ;
- 60 pour la zone humifère ;
- 5 pour les horizons ;

Qelques séries sont déclarées avec la borne 720 : la directive '720 731' permet de les reconnaître.

3.10.3 Gestion des périodes antérieures et durées de vie

Les évaluations dans les séries déclarées s'effectuent le jour de la simulation à la période t, en fonction généralement des valeurs à la période $t-1$. Quelques traitements qui traduisent des processus de vieillissement recherchent les valeurs antérieures à partir de la période $t-k$, le délai k traduisant une longévité (3.2.6).

En l'absence d'une demande de copie locale, ces séries peuvent être remplacées par des tables de tailles réduites à deux ou $k+1$ éléments. La modification est demandée par une directive de la forme 'x k' : la série x est alors représentée dans une table de taille $k+1$, le dernier élément contenant l'évaluation du jour.[110] Les séries concernées sont repérées en inversant le signe de la borne supérieure pour construire le dictionnaire

Ce transfert des séries dans des tables est à effectuer dans l'application pour initialiser les évaluations du jour, par décalage d'une période :
la fonction *Synchro*(), appelée par la fonction générée *SynchroRetro*() (3.9.1), réalise ce décalage pour les séries gérées en mode retro. .

Contrôle des périodes antérieures

Dans ce mode de gestion, dit **retro**, l'indice est remplacé dans le code généré par l'appel de la fonction *Retro*(), avec comme paramètre le rang de la fonction C, la période courante et le délai k qui vaut

[109] Les autres itérations qui utiliseraient des branchements conditionnels, ou les réaffectations du type 'SI $x > 1000$ ALORS $x = 1000$', ne sont actuellement pas prises en compte.

[110] '*Rl* 2' va transformer la table *Rl*[731][1000] en *Rl*[2][200] après l'ajustement de la profondeur '1000 200'. Si une copie locale a été demandée, l'ancienne forme est conservée.

- 0 pour la période t,
- 1 ou k si la période t est suivie du signe $-$,
- $t- < indice >$ si une variable d'indice est utilisée,

$Retro()$ évalue la valeur de l'indice i dans la nouvelle table,
- rend cette valeur si $i > 0$,
- ≤ 0 sinon, et une alerte est générée.

Cas des séries ambiguës Certaines séries sont gérées dans des tables Fortran sous la forme (`0 :731`). Si l'élément de rang 0 est effectivement référencé pour gérer une valeur non temporelle,[111] la fonction $Retro()$ déclenche l'alerte : cette forme n'est plus utilisable dans ce cas de figure.[112]

Suppression de la dimension temporelle Ce mode de gestion permet de réduire de façon très importante l'espace nécessaire pour des applications en fonction des options choisies, comme par exemple pour l'évaluation de la croissance racinaire contrôlée par le paramètre de la plante CODERACINE, avec la méthode de la densité vraie qui requiert deux matrices définies sur la profondeur du sol et la durée de la simulation.

La taille est minimale avec $k = 1$, mais il faut alors restreindre les choix d'options et gérer les éventuelles erreurs résiduelles à l'exécution.

Initialisation des séries

Certaines séries sont initialisées dans des boucles sur une période bisannuelle, dont celles relatives au système racinaire regroupées dans des matrices.

L'indice est alors remplacé dans les fonctions d'initialisation par l'appel de la fonction $IniSerie()$, laquelle rend l'indice s'il est défini pour la campagne ou la durée de vie, un indice fictif sinon.[113]

3.10.4 Non vectorisation

La directive '*x 0*' est interprétée comme une demande de non vectorisation pour des variables non utilisées avec certaines options.[114] Si cette variable x apparaît dans une instruction, la citation est remplacée par l'appel de la fonction numérique $Nenni()$, qui reçoit en paramètres le rang de la fonction et le nom de la variable, et génère une erreur à l'exécution. L'instruction C est :
- conservée pour les évaluations quotidiennes ;
- supprimée pour les initialisations, et remplacée par une instruction vide ;

[111] Une variable Fortran serait alors à ajouter pour gérer cette valeur.

[112] Plusieurs tables, dont les matrices de séries RL et DRL qui gèrent le système racinaire, sont déclarées avec une ligne 0, en fait non utilisée : le nombre de lignes peut donc être réduit.

[113] Une composante ou une ligne fictive sont allouées dans le vecteur $Y_[]$.

[114] La méthode dite de la densité vraie contrôlée par le paramètre de la plante CODERACINE requiert deux matrices de séries : $Rl[731][1000]$ et $Dl[731][1000]$ occupaient en l'absence d'ajustement près de 95% de l'espace alloué pour les évaluations quotidiennes.

Chapitre 4

Restructuration et assemblage des fonctions C

Les étapes précédentes produisent, à partir de la liste des fichiers contenant le code source Fortran, un ensemble de fonctions C structurées compilables, avec pour chacune d'elles les résultats des analyses syntaxiques décrites dans les chapitres 2 et 3. Ces fonctions peuvent être mises en correspondance avec les unités de programmes éditées dans des fichiers séparés (1.1.7).

Le processus de vectorisation est paramétré au cours de ces étapes dans un fichier dans lequel sont définis les rôles des unités de programmes, et accessoirement de blocs dans ces unités : toute correction du code source Fortran nécessite de refaire l'ensemble des étapes et de réexaminer ces rôles.[1]

Une version C structurée qui exécute à l'identique les algorithmes effectifs de la version Fortran peut alors être construite. Ce chapitre décrit le processus d'assemblage, restructuration et validation de ces fonctions, pour produire une version de référence qui effectue rigoureusement les mêmes évaluations.[2] Les fonctions sont réparties dans des modules puis corrigées si nécessaire, toute modification du code généré étant signalée en commentaire. Le logiciel est progressivement construit au moyen d'un *Makefile*. Les descriptions des fonctionnalités s'appuient sur les notices et les commentaires du code source Fortran.

Cette analyse est aussi utilisable pour :

1. restructurer le code Fortran,[3] avec les limites inhérentes à ce langage ;
2. classer les données, tant pour pouvoir paramétrer les applications que construire des bases de données ;

Une procédure de validation est proposée pour pouvoir comparer simultanément les évaluations en cours d'exécution des versions C et Fortran.

[1] Il en est de même pour toute correction de la chaîne de traitement.

[2] Une version provisoire, avec des variables qui demeurent globales et des variables textuelles non numérisées, a été développée pour le projet *Hysafe* et est opérationnelle depuis septembre 2003 : elle est actuellement utilisée par plusieurs équipes.

[3] De nombreuses notes techniques sont ajoutées dans cette perspective.

Cette méthode de construction peut être appliquée avec d'autres stratégies d'allocation adaptées à des contextes applicatifs (3.1.1) :[4] la procédure d'assemblage mobilise toutefois des ressources programmeurs de quelques journées-hommes pour aboutir à une version opérationnelle validée.

Documentation Cette description du processus de simulation est complétée par de nombreuses notes contextuelles, utiles pour apurer le code Fortran.

4.1 Généralités

Le processus de construction s'appuie sur les fichiers produits dans les étapes antérieures dans le répertoire courant et repérés par des suffixes. Les anomalies, ambiguïtés ou singularités localisées au cours des analyses syntaxiques, effectuées par les commandes $S6$ pour le traitement des noms symboliques[5] et $S8$ pour les instructions, sont signalées par des avertissements édités pour chaque unité de programme (suffixes `".S6"` et `".S8"`). Des renseignements complémentaires relatifs aux liens entre unités, édités par $S6$, sont regroupés dans un compte-rendu global (suffixe `".cr6"`). Les stuctures de contrôle peuvent aussi être examinées en consultant les fichiers descriptifs édités par $S5$ (suffixes `".sis"`, `".do"`, `".goto"` et `".cri"`).

Code C Les rangs absolus des instructions Fortran (1.1.4) sont repris en commentaires entre parenthèses, au niveau de l'instruction traduite (n) ou pour localiser les blocs associés aux structures de contrôle effectives (n_1-n_2), souvent issues de la traduction de branchements, l'étiquette figurant aussi en commentaire.
Les rangs des instructions Fortran dans les fichiers sources sont aussi ajoutés.

Les noms des variables communes figurent en commentaires, précédés du premier chiffre de la classe pour les variables statiques, avec la première lettre majuscule, les accolades à droite pour les vecteurs et matrices.

Construction du logiciel C La construction du logiciel s'effectue dans un répertoire fils de génération logicielle, par exemple $\sim /G$.

Un fichier `"makefile"` permet de fabriquer le logiciel à partir d'une fonction de supervision. Les fonctions STICS générées sont intégrées une à une, éventuellement revues en effectuant les compilations requises au pas à pas, puis l'édition de liens finale, sans omettre la bibliothèque mathématique *-lm*, jusqu'à aboutir à un produit exécutable. Un environnement de mise au point est

[4] Pour une application qui exécute simultanément un grand nombre de simulations, les dimensions des tables peuvent être adaptées. Pour des processus de décision, une matrice de séries temporelles peut être ajoutée en paramètre.

[5] Ces noms sont notés en capitales dans le contexte Fortran (sous-programme PRINCIP), en italique sinon, avec ajout des parenthèses pour les fonctions (*Princip()*) et des crochets pour les tables (*Cailloux[]*), la lettre majuscule indiquant qu'il s'agit d'un objet non local.

aussi développé pour suivre les exécutions C et Fortran et valider les versions générées.

Refabrication d'une version Fortran Les unités de programmes Fortran sont simultanément rapatriés dans le répertoire fils $\sim /Fmake$. Les fichiers "`stics.make`" et "`stics.macro`" produits par la commande $S1$ (1.1.7) sont repris pour fabriquer le $makefile$ qui va regénérer la version Fortran.

Validation D'autres répertoires sont créées pour chaque culture : les fichiers STICS utiles sont recopiés et les simulations exécutées dans ces répertoires.

Ces mêmes simulations peuvent être exécutées en Fortran dans une autre arborescence, après ajout de la gestion des sous-programmes de suivi.

Les comparaisons sont faites en fonction de la dynamique du modèle, d'abord pour l'état initial puis pour les évaluations quotidiennes, jusqu'à vérifier l'équivalence de toutes les évaluations jusqu'à la fin de la simulation.

4.1.1 Supervision

La simulation se déroule globalement en quatre étapes.

1. Acquisition des paramètres de la simulation d'une culture sur une parcelle par lecture dans des fichiers.
2. Evaluation des conditions initiales, avec en option la lecture de données produites par la simulation précédente, et acquisition des séries climatiques pour une campagne.[6]
3. Evaluations quotidiennes des états de la culture entre les dates de début et fin de simulation.
4. Restitution finale des résultats sous la forme de bilans.[7]

Ce séquencement et les relations avec le système d'exploitation sont gérés dans le module C de supervision.

Arbre des appels des fonctions

La commande $S6$ a produit les arbres des appels des sous-programmes et fonctions Fortran.[8]

La version 5 de STICS s'exécute par appel du seul sous-programme PRINCIP depuis un programme principal. Le programme principal de la version 6 reprend ce sous-programme dans le fichier "`main.for`".

[6] Cette étape s'exécute dans une première boucle qui permet d'enchaîner sur des campagnes consécutives une culture avec les séries climatiques correspondantes, à itinéraire technique constant.

[7] Pour les cultures fourragères , les bilans sont édités à l'issue des coupes dans l'étape précédente (4.5.18).

[8] En l'absence d'appels récursifs ou croisés, le graphe des appels est un arbre (fichier `"source.lien"`).

Vectorisation

Le graphe des relations entre variables communes (2.3) est utilisable pour déterminer quels vecteurs doivent être déclarés comme paramètres des fonctions C qui traduisent les unités de programmes Fortran.

1. L'ensemble des arcs, édité dans le fichier `"source.graphe"`, permet de retrouver, pour toute variable citée dans une unité de programme, l'ensemble des variables utilisées pour son évaluation, localement dans des affectations, ou par transmission de paramètres dans les fonctions appelées.
2. Les sommets isolés, qui correspondent à des variables consultées pour évaluer des conditions logiques ou des indices, sont repérés et édités dans le fichier `"source.sommet"`.

4.1.2 Précision des calculs et propagation d'erreurs

L'exécution d'un même algorithme traduit dans différents langages aboutit normalement à des résultats jugés équivalents dans le contexte applicatif, mais des écarts infinitésimaux peuvent néanmoins apparaître en fonction des compilateurs et les erreurs se propager en cours de simulation.

Le débogage de ces situations permet de retrouver les mode de traduction utilisés par les compilateurs, pour aboutir à des évaluations rigoureusement équivalentes en arithmétique binaire.

Conversions entières

L'évaluation d'une expression entière à un stade de l'algorithme aboutit quelquefois à deux résultats différents, k pour l'une et $k+1$ pour l'autre.

Les variables numériques réelles, représentées en virgule flottante, sont souvent converties en variables entières dans le code Fortran, par extraction de la partie entière.

- explicitement par appel de fonctions intrinsèques,[9]
- implicitement dans le contexte d'évaluation d'une expression réelle.

Les appels des fonctions de conversion sont très nombreux dans les séquences itératives d'exploration du système sol-plante : elles gèrent le passage d'une variable continue au rang d'une couche du sol, variable discrète.

Arithmétique en virgule flottante

Avec les formalismes retenus pour simuler des cultures, la précision est largement suffisante si toutes les grandeurs sont définis sur les domaines numériques usuels. En revanche, pour les grands nombres et au voisinage de zéro, il faut recourir aux règles de l'arithmétique binaire pour expliquer les écarts qui apparaissent au-delà de la sixième décimale lors des opérations de débogage.

[9] Une analyse des performances indique que ces fonctions de conversion consomment 20% du temps de calcul.

Choix de la précision Des écarts sur les derniers chiffres de la partie décimale apparaissent selon que les évaluations sont faites en simple ou double précision. Soit

ε_1 le plus petit nombre en simple précision tel que $1 - \varepsilon_1 \neq 1$,
$\varepsilon_2 < \varepsilon_1$ le plus petit nombre en double précision tel que $1 - \varepsilon_2 \neq 1$,
$x \in [1 - \varepsilon_1, 1 - \varepsilon_2]$:

la partie entière de x est alors 0 en double précision, 1 en simple précision.

Les résultats vont dépendre du mode d'évaluation des expressions, laquelle peut se faire en double précision si un terme est évalué selon ce mode. Les écarts ne sont généralement pas visibles dans les résultats de la simulation d'une culture mais ils peuvent néanmoins se propager dans certains contextes et devenir numériquement visibles, bien qu'encore infinitésimaux relativement aux grandeurs évaluées.

1. Les valeurs des expressions évaluées dans les conditions logiques peuvent basculer, ce qui se traduit par l'exécution des blocs conditionnels *alors* ou *sinon*, et des évaluations différentes.[10]
2. La présence d'un facteur multiplicatif fait que les écarts peuvent s'amplifier selon un mode exponentiel.[11]

Constantes réelles Les constantes réelles citées dans les expressions ou comme paramètres,[12] alors gérées en double précision en langage C, sont remplacées par des variables locales déclarées en simple précision, ce qui permet d'effectuer les mêmes évaluations sans devoir convertir les types numériques (2.1.6).

Décomposition des expressions Les fonctions mathématiques effectuent les évaluations en double précision. L'utilisation de variables locales pour gérer les valeurs rendues est un moyen de contrôler les changements de types, et les modes d'évaluation : '$x = a + e^y$;' est ambiguë, alors que '$u = e^y$; $x = a + u$;' ne l'est plus. L'introduction de variables en simple précision dans le code Fortran pour ces fonctions, et l'opérateur d'exponentiation, aboutit à un code C qui effectue des évaluations identiques.[13]

[10] Une telle divergence apparaît dans une simulation d'une culture de maïs dans le sous-programme TRANSPI (4.5.14) du fait d'un écart de l'ordre du picomètre de transpiration réelle pour des couches 8 et 9 du sol le jour 59, ce qui entraîne l'application d'un algorithme de répartition sur l'ensemble de la zone racinaire, et des écarts dès la seconde décimale sur des variables descriptives du système sol-plante. Le résultat n'est cependant pas perturbé, avec une variation de la récolte finale d'une cinquantaine de grammes sur la douzaine de tonnes récoltée. Le remplacement de l'opérateur $<$ par $\leq$ supprime ces divergences.

[11] Pour cette même simulation, une divergence démarre dans le sous-programme DENSIRAC dès que le front racinaire franchit la couche du semis, avec les valeurs 1 et 2 du facteur multiplicatif MULTLVFRONT. Les écarts se propagent ensuite lors des évaluations des variables descriptives du système racinaire, pour aboutir à une variation finale de quelques kilogrammes de grains à la récolte. Le passage en double précision supprime ces divergences, avec une précision de l'ordre de $10^{-15} cm$ pour atteindre la couche suivante.

[12] Ces valeurs sont répertoriées dans le fichier des constantes `"source.dec"` (3.9.1).

[13] La décomposition de l'expression $6.1070((1 + \sqrt{2})\sin(0.017453293t/3))^{8.827}$ évaluée par $Tvar(t)$ permet de maîtriser le processus numérique tout en améliorant la lisibilité.

Afin de générer une version C équivalente, les fonctions C de la bibliothèque mathématique sont appelées depuis une fonction dans laquelle le changement de type est effectué et la valeur en simple précision rendue à l'appelant, la première lettre du nom de fonction étant convertie en majuscule.[14]

Puissance entière et exponentiation

L'opérateur Fortran d'exponentiation '$**$' est utilisé pour l'évaluation des puissances entières, ce qui est en fait incorrect vu que l'opération est définie sur des nombres réels. Des écarts infinitésimaux peuvent apparaître à l'exécution lors de l'évaluation du terme T^4, T étant une température absolue, si l'exposant est le nombre entier 4 dans le code Fortran : après substitution du nombre réel '4.0', les évaluations sont équivalentes.[15]

Cet opérateur Fortran est finalement traduit en C avec un exposant réel, sauf pour la puissance 2, ce qui ne paraît ne pas générer de dérives visibles (3.5.1).

4.1.3 Déclaration des fonctions

Toutes les unités de programmes Fortran sont converties en fonctions, formellement décrites par des prototypes, lesquels fournissent les types et dimensions de paramètres, et indiquent si ces derniers sont des valeurs, ou des références vers des variables informatiques.

Les prototypes sont inconnus dans le langage Fortran : ils sont construits par la commande $S6$ (2.7.8). La représentation formelle de ces prototypes est reprise par la commande $S8$ pour déclarer les fonctions. La forme textuelle est éditée dans le compte-rendu global.

Les variables locales sont normalement déclarées dans la version 6, les déclarations implicites étant interdites (IMPLICIT NONE).

Tailles des tables

La présence dans une fonction de tables dont les bornes supérieures sont susceptibles de varier d'une version à l'autre est signalée dans le fichier compte-rendu. Le fichier `"source.dec"` des constantes produit par $S8$ contient, pour les instructions pour lesquelles ces valeurs sont citées en clair, le commentaire `"borne dynamique?"`.[16]

---

[14] $Log(x)$ exécute *return (float) log(x);*

[15] L'évaluation fortran T**4 est différente de T**4.0, cette dernière étant égale aux évaluations C *pow(T,4)* et *pow(T,4.0)*.

[16] La modification dynamique est demandée dans la version 5 pour la borne 1000, profondeur du sol dans les anciennes versions, à remplacer par la valeur 200 (note 92 section 3.9) ; pour la version 6, on revient à 1000. Il en est de même avec la période, bisannelle (731 jours) puis annuelle (366 jours), et à nouveau bisannuelle pour la version 6. Il faut alors vérifier que les structures de contrôle sont en conformité avec les déclarations, ce qui n'est pas toujours le cas, tant pour la durée que pour la profondeur du sol.

Ajout de variables locales

Les variables locales qui gèrent une copie d'une variable commune, homonymes repérées par une première lettre minuscule (2.5.2), sont déclarées à la suite des autres variables locales Fortran.

Des variables supplémentaires sont ajoutées si nécessaire pour initialiser les tables : elles sont suffixées par un tiret souligné ($i_$, $j_$).

Les copies locales sont faites en langage C par affectations de valeurs pour les variables simples, de pointeurs pour les tables.

Initialisations des variables locales

Toutes les variables locales sont initialisées :
- les variables locales issues du code Fortran sont explicitement initialisées avec la valeur nulle par défaut ;
- les valeurs des variables communes simples sont transférées dans les variables locales correspondantes dans une séquence d'initialisation ;
- pour les tables numériques, les variables locales sont des références vers les variables communes ;
- les valeurs des paramètres formels définis par référence (2.2.2) sont transférées dans des variables locales ;

Restitutions des valeurs des variables locales

Les valeurs des variables locales sont restituées si nécessaire avant le retour vers la fonction appelante :
- dans la variable commune simple, y compris pour les indices ;
- à travers la référence du paramètre formel ;

Les valeurs des variables de boucle communes utilisées localement en Fortran, remplacées par une variable locale en C, ne sont pas nécessairement restituées.[17]

4.1.4 Optimisation

Les outils GNU de mesure de performances sont utilisés sur les versions générées, pratiquement les commandes *time* et *gprof* : des corrections à vue peuvent être faites pour des optimisations immédiates mais les problèmes mis en évidence sont à traiter en amont, avec :
- le choix d'autres stratégies de génération du code C mieux adaptées au contexte Fortran ;[18]
- une révision prioritaire des algorithmes consommateurs de ressources, pratiquement les fonctions qui traitent du système sol-plante ;
- la suppression des évaluations redondantes ;

[17] La variable Fortran contient alors la valeur de sortie de la dernière boucle.

[18] L'appel de la fonction de conversion entière INT est à remplacer par la citation du paramètre de la fonction (4.1.2).

4.2 Réalisation du module de supervision

Un module de supervision est à définir pour piloter une version de base qui effectue finalement les mêmes évaluations que la version Fortran, donc simule de la même façon une culture sur une parcelle pour une campagne. D'abord édité dans le fichier `"Stics0.c"`, ce module gère les interfaces avec le système d'exploitation et distribue les tâches. Il contient les fonctions communes, non disponibles dans les bibliothèques et qui peuvent être appelées depuis les autres modules, des fonctions de mises au point et les structures de données associées, ainsi que la fonction principale, dite de supervision.

4.2.1 Fonctions communes

Les fonctions communes peuvent être appelées par toutes les autres fonctions, d'où le regroupement des déclarations de prototypes dans `"Commun.h"` pour gérer les inclusions requises.

Le code de ces fonctions, qui utilise souvent les fonctions mathématiques de la bibliothèque standard du langage C, est édité dans `"Intrin.c"`. La compilation produit le module objet `"Intrin.o"`. L'édition de liens finale va requérir l'argument `-lm` pour satisfaire les références vers la librairie mathématique.

Fonctions intrinsèques

Un ensemble de fonctions dites intrinsèques est défini pour remplacer des fonctions et opérations Fortran pour lesquelles les fonctions équivalentes n'existent pas dans la bibliothèque standard du langage C (3.3.6).

Fonctions intrinsèques numériques Fortran Les fonctions suivantes reprennent des fonctions intrinsèques Fortran (2.2.1).

- $Max()$ et $Min()$ pour les comparaisons de deux nombres ;
- $Mod()$ pour l'opérateur modulo ;
- $Sign()$ pour le transfert de signe Fortran ;
- $Tronc()$ pour extraire la partie entière ;[19]
- $Arrondi()$ pour la meilleure approximation entière ;
- $Rflot()$ pour convertir un entier en flottant ;

Exponentiation Deux fonctions remplacent l'opérateur d'exponentiation Fortran ** :

- $Pn(a, n)$ pour la puissance entière a^n,
- $Px(k, a, x)$ pour évaluer a^x, avec k rang de l'instruction Fortran,
  - si $a > 0$, appel de la fonction mathématique $pow(a, x)$,
  - sinon déroutement par appel de la fonction $Erreur()$ du paragraphe 4.2.1 qui clôt l'exécution sur l'erreur de domaine numéro k.[20]

[19] Au vu de la section 4.1.2, cette fonction devient inutile.

[20] Les indicateurs d'états de la bibliothèque standard déclarés dans $< errno.h >$ ne sont pas utilisés.

Conversions de types des fonctions mathémartiques Des doubles des fonctions mathématiques de la bibliotèque C `"math.h"` sont éventuellement créés pour gérer les conversions de double en simple précision : *exp*() est remplacée par *Exp*(). Cette redéclaration permet de faire en sorte que les évaluations C et Fortran soient effectuées avec la même arithmétique (4.1.2).

Fonction calendaire

La fonction calendaire *Gregoire*() rend le nombre de jours d'une année : elle est utilisée pour gérer les correspondances entre les séries calendaires des paramètres et les rangs des simulations quotidiennes, mais aussi pour convertir les numéros des jours en dates dans la fonction d'édition *Julien*().

Autres fonctions intrinsèques

Quelques fonctions ont été traduites mais ne sont pratiquement plus utilisées.

Recherche d'une valeur minimale dans une table $Minval(k, tab, i_1, i_2)$ rend la valeur minimale trouvée entre deux éléments d'une table table.[21] Elle est redéfinie avec quatre paramètres :

- k, rang de l'instruction Fortran,
- $tab[]$, table examinée,
- i_1 premier élément, habituellement 1,
- i_2 dernier élément, nul pour générer l'erreur numéro k à l'exécution, qui édite seulement un message d'avertissement : c'est alors le premier élément qui est rendu.

Recherche d'un motif dans une chaîne $Droite(x, y)$ est appelée en remplacement de la fonction intrinsèque Fortran INDEX pour rechercher la chaîne y à la fin de x, pratiquement pour retrouver un nom réservé de fichier comme composant terminal dans un chemin.

Gestion des variables et fonctions

Des fonctions de contrôle sont appelées à l'issue des neutralisations de variables et modifications des tailles des tables (3.10.2) :

- *Nenni*() traite la citation d'une variable neutralisée ;
- *Lim*() contrôle le temps dans l'espace de la simulation, la campagne ;
- *Sol*() contrôle l'exploration des couches du sol ;
- *Retro*() gère les indices des périodes antérieures après transcription des séries dans des tables ;

Exclu() gère les appels de sous-programmes non traduits (3.9.2).

[21] Cette fonction apparaît dans la version 5 déclarée avec comme paramètres un nom de table et une borne supérieure, appelée avec le seul nom de table : elle rend alors l'élément 1. Cet appel est à reprendre avec le paramétrage requis.

Gestion des erreurs et déroutements

Les erreurs sont gérées dans deux fonctions, codées dans le fichier `"Erreur.c"`.

1. *Erreur*() est définie avec trois paramètres :
   - `int` *nsource*, avec $|nsource|$ rang de l'instruction Fortran source, ou 0 si erreur système ;
   - CHAR $*z$, texte du message ;
   - FILE $*f$, pointeur sur un fichier compte-rendu ;

   En l'absence de fichier, les messages sont dirigés sur la sortie standard. Un déroutement avec retour au système est généré si ($nsource \leq 0$).[22]
2. *Alerte*() gère les erreurs des fonctions de contrôle précédentes, normalement inexistantes, et génère un déroutement qui clôt l'exécution.

Synchronisation

Les décalages avant évaluation quotidienne pour les séries gérées en mode retro (3.10.3) sont effectuées par la fonction *Retro*().

4.2.2 Lecture des paramètres et valeurs initiales

L'acquisition des données nécessite de décrypter les formats utilisés par l'interface WINSTICS, opération effectuée au moyen primitives qui lisent les fichiers en mode binaire pour gérer le flot de caractères.

Lecture des fichiers WINSTICS

Les instructions Fortran de lecture READ sont traduites par des appels de fonctions spécifiques, construites sur mesure en fonction des formats utilisés pour reconnaître les champs dans les enregistrements (3.8.4) : elles sont regroupées dans le module `"LecStics.c"`.

Les valeurs textuelles présentes dans le code Fortran sont simultanément numérisées par appel d'une fonction *Numcode*() générée dans le module `"NumCode.c"` (3.8.5), liée au code de la version Fortran courante, qu'il faut recopier depuis le répertoire frère $\sim /D$.

Les prototypes de ces fonctions appelées, et des primitives utilisées, sont déclarés dans `"LecStics.h"`.

Accès système Les fonctions précédentes utilisent des primitives qui acquièrent les octets avec des appels système *read*. Cette lecture binaire requiert aussi les appels système *open* et *close* pour gérer les descripteurs des fichiers. Ces accès systèmes sont centralisés dans ce module de supervision, avec trois fonctions :

- *Suivant*(), primitive qui acquiert l'octet suivant ;
- *Ouvrir*() qui ouvre le fichier à lire en binaire et un fichier compte-rendu en écriture ("w") ou ajout ("a") ;

[22] Le STOP Fortran est traduit de cette façon.

– *Fermer*() qui ferme ces fichiers et réinitialise les variables globales de lecture binaire ;

Elles utilisent des variables globales à déclarer.

Gestion des erreurs Une fonction spécifique,*ErrLec*(), est ajoutée directement dans le module le supervision pour traiter les erreurs de lecture. Les rangs des instructions Fortran sources sont restitués avec le nom de l'unité de programme et le nombre d'alertes au moyen des fonctions *Uprogramme*() et *Avertir*(), cette dernière étant alors appelée à la fin de la fonction principale.

Un déroutement est généré si le nombre d'erreurs dépasse le seuil MAXERR.

Contrôle des formats WINSTICS

Des erreurs résiduelles font que les applications Fortran sont interrompues en lecture. Les formats sont alors examinés : un message est édité pour signaler l'anomalie et les corrections sont à effectuer dans le fichier.

La commande

Sformat <fichier WINSTICS >

édite ainsi les lignes pour lesquelles un espace manquant génère une erreur à l'exécution sous LINUX.[23]

4.2.3 Editions en cours de simulation

Des messages sont édités en cours d'exécution dans le fichier historique. Des bilans et des états sont édités en cours et en fin de simulation,

Messages

Le sous-programme d'initialisation MESSAGE, qui range dans des variables textuelles longues les contenus des messages d'avertissements, est finalement transformé en une fonction d'édition, l'ordre d'écriture du code Fortran étant remplacé par un appel de fonction : $Message(f, n)$ édite dans le fichier f la variable textuelle longue de rang n (3.8.3).

Le code de la fonction, contenu dans "`Message.c`" du répertoire $\sim /C$, est recopié : la déclaration du prototype est présente dans "`Commun.h`", $Message()$ pouvant être appelée depuis toute autre fonction.

Bilans et états

Certains sous-programmes de la boucle quotidienne éditent des données en cours de simulation (3.8.7), dont des variables textuelles, numérisées en C. Les fonctions d'évaluation n'admettant comme paramètres que des vecteurs,

1. les instructions d'écriture WRITE sont placées en commentaires ;

[23] Le format '`69X,A20`' requiert la présence de la colonne de rang 89.

2. les valeurs des variables textuelles sont restituées par appel de la fonction *ValCode()* (3.8.5) ;
3. les éditions s'effectuent dans le fichier compte-rendu, fichier historique de l'application fortran ;

Gestion des fichiers Les ouvertures et fermetures des fichiers en écriture sont pilotées depuis le module de supervision.

4.2.4 Fonctions d'accès aux variables communes vectorisées

Le lexique des unités Fortran est géré dans $Lex[]$, le dictionnaire des variables communes dans $Dic[]$. Deux fonctions sont utilisables pour restituer ces tables :

1. *DicStics*() (3.2.7),généré, utilise des fonctions prédéfinies regroupées dans "`VecStics.c`".
   - $AjoutU()$ pour l'ajout d'une unité de programme dans le lexique ;
   - $AjoutV()$ pour l'ajout d'une variable dans le dictionnaire, avec classement alphabétique dans $Index[]$;
   - $Bivec()$ et $Trivec()$ pour gérer les vectorisations multiples ;
   - $EvalDyn()$ pour noter les unités de programmes qui modifient les variables évaluées quotidiennement ;
   - $EvalIni()$ pour les évaluations dans les unités d'initialisation ;

   DicStics() restitue la liste $lmeta[]$ des vecteurs du métamodèle et rend le nombre de variables communes $nstics$.
   Le fichier source "`DicStics.c`" est édités dans $\sim /D$, pour intégration dans le module de supervision.
2. *DicoStics*(), fabriqué, effectue les mêmes tâches mais procède par initialisation de tables de structures avec trois fichiers à inclure édités dans $\sim /D$, "`DicStics.h`", "`LecStics.h`" et "`VmultiStics.h`".[24]
   Le fichier source "`DicoStics.c`" relève du module de supervision.

Recherche ou affectations de valeurs

Le code C est généré avec les rangs des composantes des vecteurs dans lesquels les variables sont vectorisées : toute correction Fortran, aussi marginale soit-elle, peut modifier ces rangs. Afin de pouvoir néanmoins accéder à ces variables indépendemment du mode de vectorisation, en les nommant, des fonctions sont définies dans le module de supervision. Si $< E >$ désigne une composante de ces vecteurs, variable commune simple X ou élément de table $T[i]$,

1. $Valeur(< E >)$ rend la valeur désignée ;
2. $Affecter(< E >, \mathtt{u})$ affecte la valeur `u` dans la composante désignée ;

[24] Ce mode de traitement réduit de façon importante le temps de compilation, la production du code objet s'avérant coûteuse avec les nombreux appels de fonctions de *DicStics*().

3. $Adresse(< E >)$ rend un pointeur sur l'élément ;
   $Adresse(< T >)$ rend le pointeur sur la table T ;

Elles sont directement utilisables pour piloter des structures de contrôle ou neutraliser des options, ceci pour disposer d'un code de supervision indépendant des mises à jour Fortran.

Correspondance entre variables et vecteurs

Un ensemble de fonctions gère la correspondance entre le formalisme du modèle de culture et la forme vectorielle : elles sont regroupées dans les modules sources "`VecStics.c`" et "`Comparer.c`", et les prototypes déclarés dans "`VecStics.h`".

Edition des valeurs en cours d'exécution Des fonctions d'édition sauvegardent les vecteurs en cours d'exécution, après initialisation ou au cours des évaluations quotidiennes (4.2.6), lesquels peuvent être repris dans une autre application : les appels sont effectués dans la fonction $EdiMeta()$ des modules de supervision.

Comparaison des évaluations C et Fortran Des fonctions de suivi permettent de comparer toutes les valeurs, avant la première simulation et au jour j, à un seuil de précision donné.

La fonction $VoirFortran()$, contenue dans le module de supervision, permet de comparer les composantes des vecteurs de la version C avec les valeurs des variables communes correspondantes de la version Fortran. L'équivalence est contrôlée par appel de $Comparer()$ pour les variables numériques, les variables textuelles et leurs images numériques étant exclues.

La mise en œuvre des fonctions de suivi est décrite à l'issue de l'exécution de l'étape d'initialisation (4.4.5). L'équivalence étant vérifiée, les comparaisons se poursuivent jusqu'à la fin de la simulation, éventuellement en localisant par dichotomie la première période pour laquelle des divergences apparaissent.

Gestion des durées de vie Ce mode de gestion nécessite des mises à jour des tables ains vectorisées à chaque période (3.10.3) :[25]

1. $IniRetro0()$ appelle $Toujours()$ pour initialiser les tables, la valeur initiale, pas toujours nulle, étant gérée dans l'élément de rang 0 ;
2. $IniRetro()$ appelle $Aujourdhui()$ qui effectue le décalage d'une période, le dernier élément contenant la valeur du jour ;

4.2.5 Fonction de supervision

L'objectif est de pouvoir reproduire simplement la simulation élémentaire d'une culture sur une parcelle. La reconnaissance de algorithme effectif de su-

[25] Ces mises à jour étaient pilotées par $IniJour()$ dans des versions antérieures.

pervision s'effectue pratiquement à vue par examen du code Fortran, des fichiers annexes comme l'arbre des appels pouvant toutefois être consultés.

Sous-programme des versions 4 et 5 PRINCIP est appelé depuis deux programmes principaux qui initialisent quatre variables communes.[26]

1. Une première séquence effectue globalement des lectures de fichiers de paramètres et d'édition, sauf les séries climatiques.
2. Une première boucle enchaîne une même culture pour des campagnes successives.
   (a) La simulation est initialisée à l'issue de trois appels de sous-programmes :
   - INITIAL effectue des initialisations à partir des paramètres lus à l'extérieur de cette boucle, et optionnellement la lecture des données issues de la culture précédente ;
   - LECSTAT lit les séries climatiques pour la campagne ;
   - INICLIM poursuit les initialisations ;

   (b) Une seconde boucle effectue les évaluations quotidiennes à travers une séquence d'appels de sous-programmes du premier au dernier jour de la campagne.
   Des éditions d'états intermédiaires sont possibles pour les listes de variables lues dans des fichiers.[27]
   Des bilans sont par ailleurs édités pour les coupes des cultures fourragères.

   (c) La simulation se termine par deux appels de sous-programmes d'édition :
   - BILAN est un fichier formaté qui contient les résultats de la simulation ;
   - RECUP édite des données pour la culture suivante ;
3. Une séquence terminale clôt l'application.

Programme principal de la version 6 L'algorithme reste rigoureusement le même mais il est piloté dans le programme principal contenu dans le fichier `"main.for"` et des regroupements sont effectués dans des sous-programmes : le code C est édité dans le fichier `"Stics_nn"`, *nn* étant le rang de l'unité de programme (3.3.1).

Le traitement des couples pour les cultures associées introduit des boucles depuis 1 jusqu'à 1 ou 2.

Restructuration de l'algorithme de supervision

Ce séquencement est repris dans la fonction principale C, en supprimant la boucle annuelle sur les cultures, les rotations pouvant être traitées autrement et en faisant varier d'autres paramètres. La gestion des cultures associées

[26] CODESIMUL, CODEVERSION et CODECOMPIL sont textuelles et CULTUREPURE numérique. CODESIMUL, référencée dans la boucle quotidienne, est numérisée (2.1.5).

[27] Ces options, qui font appel au sous-programme RAPPORT, ne sont pas reprises : d'autres solutions sont adoptées pour pouvoir restituer des variables en cours d'exécution.

n'étant pas reprise sous cette forme, les boucles inconditionnelles sur les couples deviennent sans objet.

L'algorithme de supervision se déroule finalement en quatre étapes :

1. lecture binaire des fichiers externes ;
2. évaluation des conditions initiales ;
3. simulations quotidiennes de la culture ;
4. restitution finale des résultats ;

Des sauvegardes des valeurs peuvent être effectuées à l'issue des étapes 1 et 2

Les fonctions sont appelées avec les tables de gestion des variables communes utiles en paramètres et peuvent rendre une valeur entière, code utilisable pour faire remonter les erreurs.

Suivi Les avertissements et erreurs en cours d'exécution du code Fortran sont édités dans le fichier historique '`history.sti`'. Ces éditions sont réparties pour les étapes dans la traduction C, soit les fichiers :

- comptes-rendus en lecture ;
- suffixés par :
  - '`_ini`' dans l'étape d'initialisation ;
  - '`_hist`' en cours d'évaluation ;

Modules de supervision

Les modules de supervision ne contiennent pas de références à des instructions Fortran, ceci pour rester indépendant des variations du code dans les versions de même niveau : les instructions C équivalentes, repérées par leurs rangs (1.1.4), ne sont présentes que dans les fonctions appelées.

La fonction principale $main$(`int` $argc$,`char` $*argv$[]) est définie classiquement avec les paramètres qui gérent les arguments optionnels de la ligne de commande.

Plusieurs commandes sont définies pour sortir progressivement du contexte STICS actuel.

1. Initialisation et simulation avec lecture des fichiers STICS

$$Stics0\ [< appli >[< jour >[< ecart >]]]$$

simule la culture à l'identique de la version Fortran.

Les arguments ont la signification suivante :

1. $< appli >$ est un identificateur de la culture ;
2. l'option $< jour >$ demande la comparaison avec les évaluations Fortran :
   - après initialisation si $jour = 0$;
   - à l'issue de la simulation quotidienne $jour > 0$ sinon ;

   Les évaluations Fortran sont lues dans le répertoire $Fmake$.
3. $< ecart >$ permet de choisir un seuil d'équivalence autre que la constante EPSI $\geq 10^{-6}$;

En l'absence d'argument, le nom de la culture est demandé.

Une campagne étant au plus annuelle, les simulations sont réduites à la durée maximale, soit 366 jours.

Les tables issues de la vectorisation sont éditées après acquisition des paramètres et données initiales (4.2.6), pour pouvoir ensuite poursuivre les simulations en mode vectoriel.

2. Simulations par lecture des vecteurs et mise au point

$$Vstics\ [< appli >[< jour >[< ecart >]]]$$

effectue les simulations à partir des vecteurs lus.

L'option $< jour >$ demande, en l'absence de $< ecart >$, l'édition des vecteurs à la période indiquée, après initialisation ($jour = 0$), ou à l'issue d'une évaluation quotidienne ($jour > 0$).

L'option $< ecart >$ demande la comparaison avec les évaluations Fortran, comme précédemment mais avec un seuil d'équivalence qui peut être réduit, la notation décimale pour les vecteurs lus étant source de dérives du fait de la précision initiale.

3. Simulation libre

$$Rstics\ [< appli >[< jour >]]$$

effectue la simulation par lecture des vecteurs comme $Vstics$ mais sans les variables globales utilisées pour la lecture des paramètres et la mise au point, les vecteurs et le dictionnaire étant des variables locales de la fonction principale.

L'option $< jour >$ pilote l'édition des vecteurs à la date choisie.

Des références vers les séries qui gèrent des événements externes quotidiens sont définies sous forme de pointeurs C qui reprennent les noms des variables STICS, pratiquement pour les données météorologiques, l'irrigation et la fertilisation. Ces variables informatiques sont utilisables comme paramètres effectifs de fonctions qui peuvent être appelées dans la boucle quotidienne.

Déclaration et initialisation des vecteurs du métamodèle

Les fichiers à inclure édités dans le répertoire frère $\sim /D$ (3.9.1) sont recopiés, et les dépendances notées dans le $makefile$.

Le fichier des déclarations des vecteurs du métamodèle `"Vectorisation.h"` est inclus avec les déclarations des variables globales. Les tailles des tables sont reprises sous la forme de constantes symboliques incluses avec le fichier `"Tailles.h"`.

Les initialisations sont faites depuis la fonction $IniVec()$, avec la fonction d'accès aux variables communes $Affecter()$ pour les variables numériques, dont celles relatives à la gestion des cultures associées, invariantes pour cette version.

- Le fichier d'initialisation des vecteurs `"IniVecteurs.h"` est d'abord inclus : il utilise les variables de boucles i et j, à déclarer.

- Le code du sous-programme INITPROGRAMME est finalement réduit à l'initialisation de la variable textuelle CODEVERSION. En effet,
  - le sous-programme MESSAGE d'initialisation est transformé en une fonction d'édition (3.8.3) ;
  - l'ouverture du fichier historique à travers l'unité logique FICHIST est remplacée par la création d'un pointeur sur un fichier compte-rendu.
- D'autres initialisations faites antérieurement dans PRINCIP le sont désormais dans LECTURES et dans une boucle sur les deux cultures. Les initialisations des unités logiques pour la gestion des fichiers n'étant pas reprises, il reste l'option de gestion de la périodicité des séries, avec la variable *Pas* finalement forcée à 1 pour ne conserver que la périodicité quotidienne et supprimer l'option décadaire.[28]
- La variable commune de boucle sur les cultures NUMCULT, et la borne NBANS, reçoivent la valeur 1.

Distribution des tâches

Les quatre étapes de l'algorithme de supervision sont pilotées dans des modules externes par des fonctions qui reçoivent en paramètres les vecteurs et variables requises par toutes les fonctions appelées.

1. L'acquisition des paramètres et données initiales, décrite en section 4.3, est pilotée par la fonction *Lecture*(), progressivement construite en recherchant les appels des sous-programmes de lecture, et en effectuant les opérations requises sur les fichiers lus. Quelques séquences de lecture isolées ou atypiques sont reprises dans des fonctions nouvelles. Les séries climatiques sont dénombrées et la valeur 1 est affectée en période 0 pour signifier que la série est présente dans le fichier climatique (3.8.4).
2. L'évaluation des conditions initiales, décrite en section 4.4, est reprise dans la fonction *EtatInitial*(), construite en reprenant les appels des sous-programmes d'initialisation, éventuellement apurés des séquences de lecture, ains que des sections de code de l'unité de supervision Fortran.
3. Le corps de la boucle des simulations quotidienne est repris dans la fonction *Simuler*(), appelée avec le rang de la simulation et les vecteurs en paramètres.
4. Les éditions finales sont pilotées en sortie de boucle.

Contrôle d'exécution La fonction *Valeur*() est utilisée pour retrouver les valeurs de variables communes utilisées comme variables de contrôle dans la distribution des tâches, ce qui permet de rester indépendant du processus de vectorisation.[29]

[28] L'évaluation se fait dans le code Fortran par traitement du paramètre initial textuel WDATA1 dans la boucle sur les cultures.

[29] `Valeur("Maxwth")` rend le rang MAXWTH de la dernière simulation.

Réinitialisation des paramètres du modèle

Une double vectorisation est effectuée chaque fois qu'une même variable informatique est utilisée pour gérer un paramètre et une variable du modèle (3.2.4) : la fonction $IniVar()$ permet de réinitialiser le vecteur des variables du modèle pour une autre simulation.

4.2.6 Sauvegarde et acquisition des vecteurs

L'algorithmique de la version générée après lecture des enregistrements Fortran s'appuie intégralement sur une représentation vectorielle. Un protocole a été mis au point pour copier et lire ces vecteurs, et transmettre le dictionnaire des variables *Stics*.

Les fonctions utilisées sont regroupées dans le module source "`VecStics.c`".

Représentation des vecteurs

Les vecteurs contiennent au départ beaucoup de composantes nulles : une syntaxe est adoptée pour désigner explicitement les composantes des vecteurs, celles qui ne sont pas citées étant nulles.

Les valeurs non nulles des variables numériques sont restituées sur une ligne sous la forme décimale, avec par défaut 6 chiffres à droite du point décimal, ou entière :

1. Variable simple
   - $< Nom >< valeur >$
2. Vecteur
   - $< Nom[< indice >] >< valeur >$
     pour une composante ;
   - $< Nom[< indice >-< indice] >< valeur >$
     pour une suite de composantes de même valeur ;
3. Matrice
   - $< Nom[< ligne >][< colonne >] >< valeur >$
     pour un terme ;
   - $< Nom[< ligne >][< colonne >-< colonne] >< valeur >$
     pour des suites de termes de même valeur sur la ligne ;

Variables numérisées Les images numériques sont restituées dans les vecteurs : les codes textuels non vides sont néanmoins édités pour documentation sous la forme

- $< Nom >< code >$ pour une variable textuelle simple ;
- $< Nom[< indice >] >< code >$ pour un élément d'une table de codes ;

Durées de vie Pour les variables gérées dans des tables définies pour une période antérieure (3.10.3), c'est la période, et non le délai, qui est utilisé comme indice, sauf l'indice 0 qui gère les valeurs initiales avant simulation.

Copie des vecteurs

Les noms des fichiers de sauvegarde des vecteurs après lecture des paramètres par *Stics0* reprennent le nom de l'application, ici l'argument < *appli* >, complétés à droite par un tiret souligné suivi du nom de vecteur, soit pour $V_$ "`appli_V_`".

Si la copie est demandée par *Vtics* au moyen de l'argument < *jour* >, ce nombre est introduit par un point derrière le nom de l'application, soit par exemple "`appli.20_V_`" pour le jour 20, ou "`appli.0_V_`" pour l'état inital avant simulation représenté par la période 0 : pour relire cet état, il suffit de fournir le nom de l'application suffixé par la période, ici "`appli.0`" ou "`appli.20`".

La fonction *EdiMeta*(), déclarée et appelée dans le module de supervision, édite toutes les composantes non nulles d'un vecteur avec la syntaxe précédente, y compris pour les vectorisations multiples (3.2.4).

Copie du dictionnaire

Le contenu du dictionnaire est édité sous une forme textuelle par la fonction *CopieDico*() dans le fichier "`Stics.dic`", pour pouvoir être reconstitué dans des applications afin de gérer les variables *Stics*.

La première ligne est une entête qui contient les tailles des vecteurs.

Redémarrage d'une simulation avec *Vstics*

Les vecteurs précédents sont utilisables pour redémarrer une simulation, avant initialisation ou après la simulation de rang j. Il est alors inutile de dérouler la séquence complexe d'acquisition des données, d'où le second module de supervision déduit du premier, appelé par la commande *Vstics* qui lit l'état d'une période antérieurement édité par *Stics*0 ou *Vstics*, puis poursuit la simulation.

La fonction *LireMeta*() lit la représentation textuelle précédente pour restituer un vecteur, en gérant les vectorisations multiples, et *Indices*() interprète les formes syntaxiques adoptées.

Précision Les valeurs sont éditées dans des fichiers avec des formats par défaut qui tronquent les parties décimales au-dela d'un certain rang : il en résulte des écarts, et des dérives par rapport aux évaluations initiales, avec des durées de vie qui peuvent varier d'une journée, et des valeurs du jour alors différentes si l'algorithme fait mourir la production la plus ancienne.

Ces différences, négligeables et non visibles dans les bilans, génèrent rapidement des écarts entre les évaluations, amplifiés pour une simulation démarrée par lecture des vecteurs du fait de la notation décimale : la validation est donc à faire avec *Stics*0

Enchaînement de simulations

La forme vectorielle permet aussi de simuler des rotations à partir de vecteurs produits pour la parcelle par d'autres simulations : la fonction de supervision

est à construire.

4.2.7 Génération logicielle

A ce stade, les premières versions logicielles peuvent être fabriquées par ajout progressif des fonctions C générées, au moyen d'un ancien *makefile*, les macroinstructions non utilisées étant placées en commentaires.

Racine logicielle

Le logiciel est à ce stade constitué :

1. de fichiers de code pratiquement indépendants des versions Fortran en cours avec :
   - les modules de supervision "`Stics0.c`" et "`Vstics.c`" ;
   - les fonctions communes regroupées dans le module "`Intrin.c`" ;
   - les déclarations "`Commun.h`" des prototypes des fonctions partagées des modules précédents ;
   - les fonctions de reconnaissance des enregistrements Fortran "`LecStics.c`" et les déclaration des prototypes "`LecStics.h`" ;
   - les fonctions de traitement des vecteurs et de comparaison "`VecStics.c`" et les déclaration des prototypes "`VecStics.h`" ;
   - les trames des fonctions de distribution des tâches, avec les déclarations "`Pilote.h`" des prototypes ;
2. de fichiers propres aux versions Fortran recopiés depuis le répertoire $\sim /D$:
   - les fichiers d'inclusions pour
     - les déclarations des tailles des tables "`Tailles.h`",
     - la déclaration des vecteurs "`Vectorisation.h`",
     - les initialisations "`IniVecteurs.h`" ;
   - le module de gestion du dictionnaire associé "`DicStics.c`" ;
   - la fonction de numérisation des codes textuels "`NumCode.c`" ;
   - la fonction "`IniVar.c`" d'initialisation des variables à partir des paramètres du modèle (3.2.4) ;
   - les fonctions "`CultPrec.c`" et "`CultSuiv.c`" de gestion des rotations de culture (3.2.3) ;
3. la fonction d'édition des avertissements "`Message.c`", recopiée depuis $\sim /C$;

Sa fabrication produit un code exécutable, qui ne fait rien tant que le corps des fonctions de distribution des tâches est vide, le rang de l'ultime simulation demeurant nul par défaut.

Intégration des fonctions générées

Les fonctions STICS générées issues des unités de programmes Fortran sont reprises une à une et les appels effectués dans les fonctions de distribution des tâches : le mode opératoire est le suivant.

1. Le code source est recopié depuis le répertoire frère $\sim /C$.
2. Les messages édités pour la fonction générée sont consultés.
   (a) Analyse des noms symboliques (suffixe `"S6"`) : la liste des groupes de variables communes effectivement incluses (1.2.3) est éditée. En l'absence d'inclusion, il est indiqué si ces variables communes sont accessibles en tant qu'anciens paramètres Fortran (2.2.5).
   (b) Reconnaissance des instructions (suffixe `"S8"`) : un récapitulatif dénombre les fonctions C substituées aux fonctions intrinsèques Fortran (3.3.6),[30] et signale la présence des tables pour lesquelles les tailles ont été modifiées (3.1.2).
   (c) Examen des références à la culture associée et/ou la zone ombragée (suffixe `"ombre"`), pour vérifier que la neutralisation des termes des expressions arithmétiques et logiques est effective (3.2.1)

   – Les algorithmes effectifs sont reconstitués, en utilisant accessoirement le metteur au point.
   Le code source est éventuellement revu, voire restructuré : les instructions modifiées sont repérées par la marque $ en commentaire.
   – Si un identificateur inconnu désigne une fonction intrinsèque Fortran, la table de correspondance avec les fonctions C (suffixe `".f"`) est complétée , le code C étant produit si nécessaire.
   – La table de correspondance entre unités de programme et classes (suffixe `".role"`) peut aussi être mise à jour : il faut alors exécuter à nouveau les commandes $S6$ et $S8$ et repartir à zéro du fait de la modification de la stratégie d'allocation (3.1.1).[31]
3. Toutes les fonctions dont les prototypes sont déclarés en entête de la fonction générée courante, l'arbre des appels étant consultable (suffixe `"lien"`), sont à reprendre : les règles de dépendance sont à intégrer dans le $Makefile$.
4. La déclaration du prototype et les appels sont ajoutés.
5. Un ultime make génère le code exécutable.

Les fonctions de distribution des tâches sont relativement stables mais l'ajout ou la suppression d'une variable dans une unité appelée peut modifier le prototype de la fonction C équivalente, par ajout ou suppression du vecteur reçu en paramètre pour gérer cette variable.[32]

[30] Des contrôles peuvent s'avérer utiles, notamment pour les fonctions Fortran spécifiques de conversions entières, par troncature ou arrondi : INT,IFIX,AINT,NINT...

[31] L'inversion de dominance (3.2.1) au cours des évaluations quotidiennes se traduit par des permutations vues comme des mises à jour : les paramètres de la plante apparaissent ainsi comme des variables temporelles. Ce sous-programme est finalement classé hors boucle pour résoudre ce problème.

[32] Une commande, *Sproto*, est disponible pour comparer nouveaux et anciens prototypes à la suite d'une mise à jour Fortran.

Documentation du code C Le code C généré contient de nombreux commentaires pour identifier les variables et localiser les instructions et les blocs à travers les rangs des instructions Fortran. Les blocs des structures de contrôle reconnues sont mis en évidence au moyen des indentations mais les branchements intempestifs peuvent perturber cette mise en forme : le code C permet néanmoins de visualiser les algorithmes.

Les corrections ensuite faites dans le code C généré sont pratiquement repérées par la marque $ en commentaire : la commande *Smarque* fournit la liste des fichiers sources qui contiennent cette marque.

4.3 Lecture des données externes

La première étape effectue, après les quelques initialisations rapatriées de la séquence de supervision (4.2.5), la lecture des paramètres du modèle, de valeurs initiales de variables et d'options diverses. Les données sont contenues dans des fichiers, souvent produits en amont par des interfaces graphiques WINDOWS, avec des formats d'enregistrements à décrypter. L'option retenue a finalement été d'effectuer l'acquisition des valeurs, numériques ou textuelles, octet après octet, d'où des lectures en mode binaire pour pouvoir interprérer les formats effectifs sans restriction liée au langage C et regénérer ces valeurs. Un audit valeur par valeur est édité dans un fichier compte-rendu, lequel reprend aussi les messages des anciens sous-programmes.

Les fonctions C de lecture sont regroupées dans le module "`LecSstics.c`", et les déclarations des prototypes dans le fichier à inclure "`LecSstics.h`" : elles utilisent des primitives gérées dans le module de supervision pour traiter le flot d'octets en lecture (4.2.5).

Le logiciel démarre avec la lecture de paramètres dits initiaux suivie d'un ensemble d'opérations de gestion de noms de fichiers, puis des appels de sous-programmes de lecture pour les différents fichiers de paramètres, sauf les séries climatiques. Cette séquence est exécutée dans les sous-programmes

- PRINCIP pour la version 5,
- LECTURES, appelé depuis le programme principal, pour la version 6, sauf pour les paramètres du sol : le code C généré est directement repris dans le module de supervision.

Les appels des fonctions C correspondantes sont regroupés dans le module "`Lecture.c`". La macro-définition LECTURE permet de gérer la liste des modules objets dans le *Makefile*.

Paramètres et variables du modèle Le traitement des séquences de lecture des paramètres permet de repérer toutes les mises à jour ensuite effectuées, lesquelles génèrent un nombre relativement important d'avertissements. L'objectif est ici d'identifier les problèmes, les solutions se situant en amont :

1. dans une étape de révision du Fortran, par duplications de variables, déplacements d'instructions, restructurations des unités de programmes ;

2. dans des applications qui prendraient en charge les contrôles d'intégrité dans des bases de données dotées d'interfaces ;

Classes de paramètres et vectorisation Les contrôles et évaluations faites requièrent souvent d'autres vecteurs que celui qui contient les paramètres lus. Ainsi, les éditions des valeurs lues sont pilotées par le paramètre général CODESIG vectorisé dans $K_$, ce qui fait que ce vecteur est toujours présent comme paramètre dans la fonction C qui traduit le sous-programme de lecture.

Classification des variables lues Un rôle a été affecté aux unités de lecture pour pouvoir effectuer une classification des variables (2.4.2), en l'occurrence des familles de paramètres du modèle, lus directement ou quelquefois évalués après lectures d'autres variables, communes ou locales. Ces lectures et évaluations s'effectuent quelquefois dans plusieurs unités de programmes, ce qui nécessite d'effectuer un choix : la lecture est prioritaire et c'est la valeur maximale du rôle qui est retenue si plusieurs lectures, ou aucune, sont effectuées.

Citations de variables d'autres classes Certaines lectures ou affectations locales peuvent exceptionnellement dépendre des valeurs de quelques paramètres lus ailleurs : le vecteur correspondant est alors déclaré en paramètre pour le métamodèle (3.3.1).

Contrôles et corrections de variables après lecture Des contrôles sont parfois réalisés immédiatement après lecture. Si certains génèrent un déroutement avec édition d'un ultime message dans le fichier historique, d'autres effectuent des corrections, alors signalées dans le compte-rendu.

Cette version C étant supposée s'exécutée parallèlement à la version Fortran pour valider l'ensemble des évaluations, les séquences de contrôle qui ne modifient pas les valeurs des variables ne sont pas nécessairement reprises ici (4.3.7).

Mises à jour des variables lues en dehors des unités de lecture Les valeurs lues sont remplacées dans différents contextes par d'autres valeurs les premières étant alors perdues. Ces mises à jour sont faites dans les phases d'initialisation des cultures mais aussi au cours des évaluations quotidiennes, si bien que des variables définies comme paramètres du modèle sont évaluées en cours d'exécution, et vectorisées avec les variables dynamiques. Elles sont localisées par examen du graphe des relations entre variables pour les évaluations, par consultation du dictionnaire pour les réinitialisations par affectations de valeurs constantes, et signalées dans le compte-rendu, dans l'unité de lecture ou dans celle qui effectue la mise à jour selon leur ordre relatif.[33]

[33]Pour l'affectation d'une constante, l'avertissement cite la dernière unité de programme analysée qui a effectué une telle affectation. Les autres affectations sont traitées dans le graphe et répertoriées.

Il est aussi vérifié que les unités de lecture n'évaluent pas des variables lues ailleurs.

Branchements inconditionnels avant Les séquences de traitement des erreurs sont localisées par des étiquettes logiques, utilisée dans les instructions de lecture ou des branchements explicites, et se soldent souvent par l'exécution d'un STOP fortran qui clôt immédiatement l'exécution : le saut de cette séquence en fin de lecture s'effectue par un branchement inconditionnel. Le code C associé est placé dans un bloc conditionnel contrôlé par le prédicat (0), toujours faux : il ne s'exécute jamais et peut donc être supprimé.

4.3.1 Pilotage de la lecture

Les instructions Fortran de lecture READ sont traduites par des appels des fonctions du module `"LecStics.c"`, générés à partir des sous-programmes de lecture, lesquelles reproduisent à l'identique l'interprétation des formats dans les champs des enregistrements (3.8.4).

Lecture des fichiers WINSTICS

Plusieurs sous-programmes lisent des fichiers produits par l'interface WINSTICS dans lesquels les variables sont classées par rubriques. La lecture démarre par une recopie dans un fichier temporaire avec une boucle par un branchement arrière inconditionnel, la sortie étant générée par le branchement associé à la fin de fichier (3.7.3) : ces séquences, repérables par les instructions Fortran d'ouverture et fermeture de fichiers, sont supprimées, pour démarrer directement sur la lecture du premier paramètre.[34] Le positionnement est assuré par un saut d'enregistrement traduit par un appel de la fonction $Vstics()$ (3.8.4).[35]

La lecture est effectuée dans une longue séquence variable par variable.[36] Les anomalies sont signalées dans le fichier compte-rendu et les rangs des instructions sont notés : elles sont dénombrées en fin d'exécution.[37]

Elle est souvent suivie d'opérations de contrôles et initialisations variées.

Appels des fonctions de lecture

Les fonctions issues de la traduction des sous-programmes de lecture, lesquelles regroupent des instructions de lecture mais aussi des contrôles et initia-

[34] Les fichiers WINSTICS sont lus avec l'unité logique 12 et le fichier temporaire recopié puis relu avec l'unité 36. Les opérations se suivent en séquence pour LECPARAM et LECPLANT, sont gérées dans une conditionnelle après lecture du premier enregistrement dans LECTECH. Ce système n'est pas repris pour LECPARAMV6.

[35] Un autre mode de lecture est utilisé dans la version 6 sans fichier temporaire au moyen d'un choix multiple (4.3.4).

[36] La variable est désignée par son nom, cité entre quotes dans les évaluations conditionnelles de LECPARAMV6.

[37] Une même instruction va générer plusieurs alertes dans une boucle.

lisations,[38] sont appelées dans la fonction *Lecture*() construite au pas à pas par ajout de :

1. d'abord la déclaration du prototype, présente chez l'appelant ;
2. l'ouverture du fichier de paramètres en lecture binaire et d'un fichier compte-rendu associé, en mode écriture (`"w"`) ou ajout (`"a"`) ;
3. l'appel de la fonction de lecture telle que générée ;
4. la fermeture des fichiers ;

La dernière version gère un seul appel conditionnel en fonction de l'option CODESUITE : l'utilisation de la fonction *Valeur*() (4.2.4) rend la traduction de cet appel indépendante de la vectorisation effectuée.[39]

Pour toute unité de programme Fortran contenant une instruction de lecture, le fichier `"LectStics.h"` a été inclus en entête de la fonction C générée : la règle de dépendance est à ajouter dans le *makefile* pour produire le module objet correspondant, en complétant la macro-définition LECTURE.

L'ordre des appels est à conserver, certaines séquences étant contrôlées par des paramètres lus auparavant.[40]

Désignation des fichiers de paramètres et comptes-rendus associés

Les noms de fichiers sont construits par ajout de suffixes réservés, le point étant utilisé pour le fichier de paramètres , et le tiret souligné pour le compte-rendu.

Le compte-rendu associé à la lecture d'un fichier contient :
- toutes les valeurs lues éditées en séquence avec l'image des variables textuelles numérisées ;
- les éditions du fichier historique (3.8.7), messages et copies contrôlées par l'option CODESIG ;
- les messages d'erreurs ;

La traduction est à valider par exécution du code C pour les fichiers successifs et examen des comptes-rendus détaillés, les éventuelles anomalies liées aux formats Fortran étant à traiter au cas par cas.

Arrêt d'exécution La lecture des paramètres est parfois suivie de contrôles : les erreurs se soldent dans la version Fortran par l'édition d'un message dans le fichier `"history.sti"`, suivie par l'instruction Fortran STOP qui clôt immédiatement l'exécution. Ces séquences sont traduites par un appel de la fonction *Erreur*() du module de supervision, laquelle renvoie au système comme code d'erreur le rang de l'instruction Fortran.

[38] Dans la version *Hysafe*, ces opérations sont réparties dans deux modules de tailles équivalentes : `"Lire.c"` pour la lecture et `"CtlLire.c"` pour les contrôles et initialisations.

[39] *Lecture*() n'est pas indépendante du code Fortran : des mises à jour peuvent modifier les prototypes des fonctions générées.

[40] Le graphe de la section 2.3 fournit la liste des sommets cités dans des évaluations faites dans les sous-programmes de lecture, liste éditée dans le fichier `"source.ref"`.

Etiquettes Fortran de branchements pour les erreurs de lecture Les séquences de déroutement avec branchements associées aux traitements des erreurs de lecture en Fortran (3.7.3),[41] sont traduites en C par des structures conditionnelless contrôlées par la condition logique (0), toujours fausse. Ces blocs conditionnels qui apparaissent à l'issue de la lecture en fin de sous-programmes sont générés pour accessoirement récupérer le code, mais sont à supprimer, vu qu'ils ne s'exécutent jamais en C. Le branchement Fortran au-dela de cette séquence en fin de lecture en l'absence d'erreurs est signalé lors de l'analyse syntaxique.

Couples de cultures

La duplication des variables pour les cultures associées introduit un indice qui prend les valeurs 1 ou 2. Ces duplications concernent les paramètres de la plante, des paramètres techniques et un certain nombre de paramètres dits "nouvelle version".

1. Les lectures des paramètres de la plante et techniques sont pilotées dans des boucles avec une ou deux itérations, les indices des variables dupliquées étant dans la liste de lecture la variable de boucle IPL : le code généré pour une culture pure supprime simplement cette boucle, les indices étant par ailleurs absents dans les fichiers WINSTICS.
2. La lecture des paramètres "nouvelle version" est adaptée au contexte, les indices apparaissant en clair dans les listes de lecture et les fichiers WINSTICS (4.3.4).

4.3.2 Paramètres initiaux

Des données descriptives de l'unité de simulation et de son état initial sont d'abord lues en format fixe dans le fichier texte `"travail.usm"` par l'unité de programme LECUSM, à laquelle est affectée le rôle 60 : aucune correction n'est requise et la fonction générée *Lecusm*() est directement compilable.

Ces données regroupent pêle-mêle :

1. des caractéristiques diverses de l'unité de simulation ;[42]
   - des options, textuelles (CODESIMUL), ou numériques (CODOPTIM,CODESUITE,CULTUREAN).[43]
   - des noms de fichiers pour les deux cultures (FPLT, FTEC, FLAI) et les deux variables (WDATA1,WDATA2) utilisées exclusivement pour gérer la périodicité annuelle ou décadaire (note 28 section 4.2.5) ;[44]

[41] La séquence est quelquefois inaccessible, l'option de contrôle requise dans l'ordre READ, par exemple ERR=36, étant absente.

[42] CODESIMUL, variable textuelle, était initialisée par affectation (`"culture"` ou `"feuille"`) dans les anciennes versions.

[43] CODESUITE est réinitialisée pour les cultures suivantes au début de la boucle d'enchaînement des cultures (4.4).

[44] Les opérations Fortran sur ces variables textuelles, qui génèrent une dizaine d'avertissements, ne sont pas reprises.

- la période de simulation (IWATER,IFWATER) ;[45]
- un numéro d'identification du sol (ICHSL) ;

2. des données initiales ;[46]
   - quatre variables descriptives de d'état initial du sol pour cinq horizons : humidité HINITF, quantités d'azote NH4INITF et NO3INITF, densité racinaire DENSINITIAL ;
   - un état initial d'un culture pérenne, avec six variables numériques, LAI0,MASEC0,ZRAC0,MAGRAIN0,NPLANTE0,RESPERENNE0, et une textuelle, STADE0 ;

Le fichier `"travail.usm"` est renommé par ajout du suffixe `".usm"` au nom de l'application.

Cultures associées Les fichiers sont définis pour les deux cultures et les variables associées gèrent des couples : les indices (1) ou (2) figurent dans la liste de lecture.

L'état initial d'une culture pérenne est défini pour les deux plantes, ainsi que la variable descriptive des horizons du sol DENSINITIAL, d'où la présence nécessaire des indices.

Trois exemplaires de LECUSM sont fournis : c'est finalement le fichier source `"lecusmPURE.for"` qui est retenu, les variables précédentes étant citées avec l'indice (1) et la variable NBPLANTES initialisée à 1.

Variables lues mises à jour dans d'autres unités de lecture de paramètres

- L'option CODESUITE est forcée à 1 dans le programme principal dans la boucle sur les cultures depuis 1 jusqu'à NBANS, cette borne supérieure lue en entête du fichier des paramètres climatiques (4.3.7) pouvant par ailleurs être corrigée.[47]
- Le nom du fichier des paramètres de la plante, copié dans FPLT, est éventuellement remplacé par le nom prédéfini "`solnu.plt`" dans l'unité de lecture des paramètres techniques LECTECH si la date de semis est indéterminée et si, par ailleurs le paramètre initial STADE0 indique que la culture est en place.
- La période de simulation IWATER-IFWATER peut être corrigée dans :
  - INICLIM si les dates sont incompatibles avec d'autres paramètres ;
  - INITNONSOL selon la date de restitution de la simulation précédente lue dans le fichier "`recup.tmp`" ;
  - le programme principal dans la boucle d'enchaînement des cultures ;
- Les variables descriptives de l'état initial d'une culture pérenne :

[45] Les valeurs lues sont quelquefois remises à jour en cours d'initialisation (4.4.3).

[46] Les variables lues sont explicitement citées dans une liste pour le codage des rôles par $S6$ (2.4.3).

[47] NBANS est décrémentée d'une unité si le le paramètre initial CULTUREAN ne vaut pas 1, ou forcée à 1 pour la valeur '`feuille`' de l'option CODESIMUL affectée au début du programme principal.

- sont remises à zéro dans INITNONSOL pour la valeur 'snu' de STADE0 ;
- reçoivent en fin de simulation les valeurs ensuite copiées dans le fichier "recup.tmp" pour réinitialiser la simulation de la culture pérenne la campagne suivante ;

Version 5 (pour mémoire) Des instructions de lecture de deux variables communes, $Norg$ et $Profhum$, ainsi que des paramètres d'édition, sont présentes mais les enregistrements sont absents des fichiers : ces instructions sont supprimées.[48]

4.3.3 Paramètres généraux

Des paramètres dits généraux, contenus dans le fichier "param.par", renommé pour la version C par ajout du suffixe ".par" au nom de l'application, sont ensuite lus par LECPARAM, auquel est affecté le rôle 5 : la fonction générée $Lecparam()$ est directement compilable.

Ces paramètres généraux regroupent :

- une vingtaine d'options souvent binaires, gérées dans des variables de la forme "$Cod...$" ;
- une centaine de variables qui décrivent des grandeurs physiques indépendantes des cultures et communes à toutes les simulations ;

Aucune variable n'est définie pour une culture associée.

La plupart des variables, qui gèrent des valeurs indépendantes des cultures, sont vectorisées dans $K_$. D'autres vecteurs sont cependant requis, avec :

- $P_$ pour $Kbio$ lue aussi avec les données de la culture précédente ;
- $W_$ pour $Albedomulch$ lue aussi avec les paramètres techniques ;
- $Y_$ pour Ra, $Codecaltemp$ et $Z0solnu$ mises à jour dans la boucle quotidienne ;

Le vecteur $S_$ apparaît aussi car il contient les variables $Codecailloux$ et $Typecailloux[1]$ consultées pour l'édition optionnelle des valeurs des variables $Masvolcx[1]$ et $Hcccx[1]$ vectorisées dans $K_$.[49]

Une dizaine d'options sont lues en début de fichier puis les variables sont regroupées par catégories, certaines contenant une option : croissance aérienne, croissance racinaire, absorption eau et azote, minéralisation et pertes engrais, climat, climat sous abri, sol. Les derniers paramètres sont définis pour des typologies et gérés dans des tables de tailles fixes lues dans une boucle adaptée, soit

- 10 types de cailloux décrits par 2 paramètres ;
- 8 types d'engrais azotés décrits par 4 paramètres ;

[48] Ces deux variables sont par ailleurs relues avec les paramètres du sol, ce qui introduit une ambiguïté de classement (2.4.3) signalée par la commande $S6$. Les séquences qui visent à restaurer ces valeurs initiales après lecture des paramètre du sol deviennent sans objet (recopie de $Norg$ dans $Xorg$ et de $Profhum$ dans $xpro$ et restitution si $(Xorg > 0)$).

[49] L'édition est erronée vu que les paramètres du sol ne sont pas encore lus.

– 8 types de résidus,[50] décrits par 11 paramètres ;[51]
– 5 types de paillage végétal décrits par 5 paramètres gérés dans 5 tables ;[52]
Les valeurs de ces types sont gérées, comme les options, dans des variables de la forme "*Cod...*".

Huit paramètres relatifs à la racine, lus dans la version 5 avec ces paramètres généraux sous contrôle de la variable CODRAC, le sont désormais depuis le sous-programme LECPARAMV6 de la section suivante.

Erreurs de lecture Des tables sont introduites dans la version 6 et les paramètres descriptifs du paillage végétal (note 52) sont lus dans une boucle de cinq itérations, alors que deux types de paillage seulement sont décrits dans le fichier de paramètre fourni.[53]

Variables affectées dans d'autres classes

Plusieurs variables lues avec les paramètres généraux ($classe = 5$) sont modifiés ailleurs, voire relues, dans des classes de rangs supérieurs :
- deux sont classées avec les variables temporelles ($classe \geq 100$) car mises à jour sous conditions depuis des sous-programmes appelés dans la boucle quotidienne ;
  - CODECALTEMP est forcée à 1 dans CALRNET ;
  - Z0SOLNU reçoit les valeurs de paramètres "nouvelle version" dans TASSEMENT ;[54]
- KBIO est utilisée dans INITNONSOL pour lire les résidus de la culture précédente ($classe = 70$) ;
- ALBEDOMULCH est relue sous condition avec les paramètres techniques ($classe = 40$) ;

Corrections en cours d'initialisation

Quelques paramètres sont modifiés dans la boucle d'initialisation des cultures :
- COEFB est divisé par 100 dans INITNONSOL ;
- ZESX prend la valeur de PROFSOL dans INITSOL2 s'il dépasse la profondeur du sol ;
- CODEINITPREC est forcé à 2 si (CODESUITE = 1) et (NBANS = 1) (voir notes 43 et 47) ;

[50] Culture (1), culture intermédiaire (2), fumier (3), compost (4), boue (5), vinasse (6), corne (7), autre (8).

[51] La variable de boucle IRES est commune.

[52] ALBEDOMULCH, est relu avec les paramètres techniques sous contrôle de CODEPAILLAGE pour le rang lu TYPEMULCH (4.3.5).
Dans la version 5, ces paramètres sont gérés dans des variables simples lues dans une boucle contrôlée par les paramètres techniques lus auparavant, soit l'option CODEPAILLAGE et le rang TYPEMULCH : la variable de boucle varie avec le pas 5 jusqu'au rang recherché et les valeurs de l'itération i remplacent celles de l'itération $i - 1$.

[53] La lecture de DECOMPOSMULCH(3) déclenche une sortie de boucle sur fin de fichier : la valeur est indéterminée.

[54] RUGOCHISEL ou RUGOLABOUR.

– CODECALTEMP est forcé à 1 dans INITNONSOL si (CODELAITR = 2) ; CODEMSFINAL est mis à jour avant cette boucle dans INITSIMUL.

4.3.4 Paramètres "nouvelle version"

Un autre fichier de paramètres, `"paramv6.par"`,[55] renommé pour la traduction C par ajout du suffixe `".pv6"` au nom de l'application, apparaît dans la version 6 : il est lu par LECPARAMV6, auquel est affecté le rôle 9, appelé depuis LECTURES. Les variables sont pour la plupart vectorisées dans $K_$ du fait de la classification adoptée pour les variables lues (2.4.3).

Organisé comme les autres fichiers WINSTICS, les modes de lecture et traitement sont spécifiques et ont nécessité le développement de séquences nouvelles adaptées au contexte de programmation (3.8.4 et note 33 section 1.3.6) : le code généré est ainsi directement compilable.

Les instructions conditionnelles sont répétées pour tous les noms en clair dans une boucle de lecture pilotée par un branchement arrière inconditionnel, soit une centaine de tests effectués une centaine de fois la sortie s'effectuant en fin de fichier (3.7.3). Ce branchement est supprimé et la lecture s'effectue finalement en séquence, comme pour les autres fichiers WINSTICS.

Une douzaine de paramètres sont définis pour un couple et les éléments sont nommés dans l'enregistrement avec l'indice associé Fortran en clair, "(1)" ou "(2)" : la variable correspondante est ensuite citée avec ce même indice. Ce second enregistrement est ignoré pour une culture simple, la valeur étant sans objet.

Les instructions de dénombrements des paramètres lus et les deux séquences conditionnelles finales, qui effectuent des contrôles mais n'évaluent pas de variables, ne sont pas reprises, les blocs étant supprimés lors de la génération du code.[56]

Double lecture HAUTMAXTEC et LARGTEC sont vectorisées dans $W_$ car lues ensuite sous condition avec les paramètres techniques en fin de fichier.

Erreurs de lecture L'ajout des ordres de lecture de nouveaux paramètres relatifs à une autre culture génère des erreurs finales si les fichiers de paramètres fournis ne contiennent pas ces nouvelles données.

[55] Il contient le commentaitre `"en attente d'être redirigés dans leurs fichiers respectifs"`.

[56] La première est une option de mise au point activée par l'affectation dans le code CODECRITMOUCH = 0.
La seconde vérifie que le nombre de paramètres lus est celui fourni par l'affectation NBPARAMSPOT = 105.

4.3.5 Paramètres techniques

Les noms des fichiers des paramètres techniques sont lus avec les paramètres initiaux (4.3.2). Le sous-programme LECTECH, auquel est affecté le rôle 40,[57] est appelé depuis LECTURES dans une boucle sur les cultures associées avec la variable IPL qui varie de 1 à NBPLANTES, ouvre le fichier dont le nom est contenu dans FTEC(IPL), renommé dans la traduction C avec le nom de l'application suffixé par `".tec"`.
La fonction générée *Lectech*() est directement compilable.

La plupart des variables sont vectorisées dans $W_$: toutefois

- un certain nombre sont corrigées ou réinitialisées dans la boucle quotidienne, notamment après coupe des cultures fourragères, d'où la nécessité d'accéder au vecteur $Y_$;
- des paramètres généraux ou "nouvelle version" sont utilisés pour des affectations conditionnelles et des éditions, d'où la présence du vecteur $K_$;
- le vecteur $P_$ est transmis car une évaluation conditionnelle utilise le paramètre initial IWATER ;

Les paramètres techniques regroupent les variables descriptives classées par fonctions, auxquelles sont associées des dates, numéros de jours calendaires, l'ordre qui suit n'étant pas celui de la lecture.

Interventions culturales périodiques Le nombre d'interventions pour la campagne est acquis, puis les variables descriptives, dont la date, sont lues dans une boucle et empilées dans des tables.[58]
Ces opérations sont :
- des travaux du sol, avec enfouissement de résidus repérés par les huit types précédents (note 50) ;
- l'irrigation ;
- la fertilisation azotée avec les huit types d'engrais précédents ;

Paillage CODEPAILLAGE code l'absence de paillage (1), la présence d'un paillage végétal (2), ou plastique (3) : une seule intervention est prévue et les variables lues décrivent un apport de mulch végétal.[59]

Culture sous abri Des surfaces d'ouverture et des dates sont lues.

Stades observés Des dates des stades de développement de la plante et une date ultime de récolte peuvent être fournies, la valeur 999 indiquant que ces dates ne sont pas définies.

Semis Date, caractéristiques et code du groupe variétal.

Interventions culturales spécifiques Une seule intervention est prévue à une date lue pour des opérations de rognage, éclaircissage, effeuillage, taille.

Récolte des grains et fruits Seuils physiologiques et périodicité des cueillettes ;

[57] Les variables descriptives des opérations quotidiennes sont regroupées dans la classe 41 (2.4.3).
[58] La variable locale *saut_* (3.3.2) est transmise d'un appel à l'autre pour gérer la suite de ces interventions (note 87 section 3.8.4).
[59] Une occurrence du paramètre général ALBEDOMULCH est relue ici sous condition (note 52).

Fauches des cultures fourragères L'option *Codemodfauche* pilote plusieurs modes de gestion, le nombre de coupes étant lu ou calculé ; les caractéristiques techniques et les valeurs résiduelles sont lues pour les coupes successives.

Autres HAUTMAXTEC et LARGTEC sont relus (4.3.4) si le paramètre "nouvelle version" CODELECTEC vaut 1.[60]

Ces paramètres étant lues, de nombreuses corrections sont ensuite effectuées, dans ce sous-programme et ensuite dans les séquences d'initialisation.

Corrections dans l'unité de lecture

Des contrôles sont effectués après lecture dans LECTECH et des corrections sont effectuées : elles sont généralement signalées dans le fichier historique.

- IPLT, date d'installation de la culture, est forcée à 999 si la chaîne `"solnu.plt"` est contenue dans le paramètre initial FPLT ;
- PROFSEM, profondeur du semis, est forcée à 1 si la valeur lue est nulle ;
- si PROFTRAV,RATIOL ou RATIOLN sont nulles ou négatives, elles sont forcées à la valeur 1 ;
- PROFRES, profondeur des réserves du sol, est définie sur l'intervalle [1 − PROFTRAV] ;
- TEMPFAUCHE cumule les degrés-jours pour les coupes fourragères successives si (CODEMODEFAUCHE = 3) ;
- QRES, quantité de résidus, est remise à zéro pour les jours sans travail du sol ;
- LOCIRRIG est bornée supérieurement à la valeur 50 ;
- RESSUITE, textuelle, est réinitialisée avec `"aucun"` si ($Iplt = 999$) ;
  des opérations textuelles sont aussi effectuées pour harmoniser les orthographes des codes textuels ;[61]
- l'option CODECALFERTI est forcée si une culture associée est présente ;
- l'option CODCUEILLE est modifiée en fonction de la valeur de CODEFAUCHE ;
- TYPEMULCH est remise à zéro en fonction des options CODEPAILLAGE et CODEACTIPAILLIS ;

Initialisation des cultures

La plupart des mises à jour sont effectuées depuis INITNONSOL appelé depuis INITIALISATIONS au début de la boucle des cultures.

- IPLT est remplacée par IFINDORM si (CODEFROID = 3) et (CODEDORMANCE = 1) ;
- le traitement des coupes de cultures fourragères dépend de la valeur du paramètre technique HAUTCOUPE lu pour chaque coupe, la valeur 999 codant l'indétermination ;

[60] Commentaire : *ajout d'un code pour Inaki pour la lecture de hautmaxtec et largtec dans le fichier technique.*

[61] Il s'agit par exemple de substituer "`pailles`" à "`paille`". Cette variable est numérisée dans la traduction C (2.1.5).

  - LAIRESIDUEL, MSRESIDUEL, MSCOUPEMINI sont réévaluées selon deux modes selon que HAUTCOUPE est après lecture connu (< 999) ou indéterminé ;
  - si HAUTCOUPE est indéterminé, il reçoit ensuite la valeur par défaut HAUTCOUPEDEFAUT ;
- l'option CODEFAUCHE est forcée à 2 en fonction de CODELAITR et CODEBESO ;
- PROFRES, profondeur des réserves du sol, est bornée par PROFSOL ;
- ORIENTRANG est convertie de degrés en radians ;
- COUVERMULCH est remise à zéro en fonction des options CODEPAILLAGE et CODEACTIPAILLIS ;
- l'option CODESTADE est remise à zéro en fonction de CODEINITPREC et CODEPERENNE ;
- NBCUEILLE est remise à 1 en fonction de l'option CODCUEILLE ;
- NAP, nombre d'apports d'eau, est remise à zéro en fonction de l'option CODECALIRRIG ;[62]
- NAPN, nombre d'apports d'engrais azotés, est remise à zéro en fonction de l'option CODECALFERTI ;
- DENSITE est corrigée pour la culture associée :
  - directement depuis INITIALISATIONS ;
  - ensuite depuis INITNONSOL ;
  - également à l'issue de la simulation de la culture dans le programme principal ;

Dénombrement des apports d'eau d'irrigation Le vecteur des apports d'eau quotidiens est construit dans le sous-programme INICLIM, appelé dans une boucle pour le couple de cultures.

Le nombre d'apports NAP est réduit d'une unité dans INICLIM si la quantité d'eau d'irrigation du jour n'est pas nulle (AIRG(N) $\neq 0$), ceci pour ne dénombrer qu'un seul apport, même si la quantité apportée pour chaque culture, paramètre technique DOSEI, n'est pas nulle.

Forçage des paramètres techniques de travail du sol pour les cultures associées Les paramètres des cultures autres que principale (IPL > 1) reçoivent les valeurs lues pour la culture principale (IPL $= 1$), ceci dans une boucle qui varie depuis 2 jusqu'à NBPLANTES, soit une itération pour la culture associée.

Ce traitement est effectué dans TESTVARTECH appelé depuis INITIALISATIONS. Le corps de la boucle est en fait supprimé lors de la génération de code (3.7.1), si bien que la fonction C est vide : les avertissements sont finalement sans objet mais signalent néanmoins des problèmes à résoudre en amont.

Mises à jour au cours des évaluations quotidiennes

Plusieurs mises à jour sont liées au mode de gestion des coupes des cultures fourragères, la rénitialisation après recolte s'effectuant par appel de INICOUPE

[62] NAP étant aussi remise à zéro dans IRRIG traité après INITNOSOL, l'avertissement n'est pas édité (note 33).

dans la boucle quotidienne, avec des reports pour la coupe suivante gérés dans les variables MSCOUPEMINI, LAIRESIDUEL, MSRESIDUEL,TEMPFAUCHE (4.5.18).

D'autres corrections sont liées à la gestion des apports d'eau ou d'azote :

- ENGRAIS, type d'engrais, est forcé à 8 dans FERTIL en fonction de l'option CODECALFERTI ;
- NAPN, nombre d'apports d'engrais azotés, est mis à jour dans CALAPNENUPVT et FERTIL ;
- JULAPN, date d'un apport azoté, est forcée à la date de semis sous condition ;[63]
- NAP, nombre de jours d'irrigation, est réévalué dans IRRIG, et incrémenté dans APORES si (QEAURES > 1) ;

D'autres variables sont mises à jour :

- COUVERMULCH est évaluée dans ETATSURF si (CODEPAILLAGE $\neq 2$) ;
- CSURNRES apparaît comme modifiable dans FERTIL car transmise par référence à APORES ;
- DENSITE est réévaluée dans LEVEE et mise à jour dans SENESCEN ;

Autres traitements après lecture

Un certain nombre de contrôles se soldent par l'édition de messages et un STOP Fortran qui clôt l'exécution.

Les éditions dans le fichier historique sont contrôlées par l'option d'édition CODESIG, indéterminée dans la version 5 car lue plus loin avec les paramètres généraux.[64]

L'exécution signale sur un exemple une anomalie de lecture liée à une erreur de format (version 5).[65]

4.3.6 Paramètres de la plante

Le sous-programme LECPLANT lit quelque deux cents variables dans un fichier dont le nom est acquis avec les paramètres initiaux (4.3.2) :

- par lecture directe pour une culture désignée ;
- `"solnu.plt"` en l'absence de culture ;

Il est appelé pour la version 6 depuis LECTURES dans une boucle sur les cultures associées avec la variable IPL qui varie de 1 à NBPLANTES : le fichier est renommé avec le nom de l'application suffixé par `"plt"`.
La fonction générée *Lecplant*() est directement compilable.

Bien que ces paramètres soient normalement vectorisées dans $V_$, d'autres vecteurs sont requis :

[63] Commentaire : *pour Anne-isabelle on impose une ferti au semis si la somme de temp est* ≤ 0.

[64] Les variables étant initialisées par défaut avec une valeur nulle dans la version C, le prédicat de contrôle de l'exécution du bloc, traduction d'un non retour du sous-programme Fortran, est finalement toujours vrai.

[65] Le champ valeur du fichier STICS lu contient la valeur `".00000"` pour la variable entière *Biorognem* : la fonction de lecture ne conserve que la partie entière et édite un avertissement, en interactif et dans le fichier compte-rendu.

- les forçages des stades et les réinitialisations consécutives à des opérations techniques ou des évènements se traduisent par des mises à jour qui font qu'une vingtaine de variables sont vectorisées dans $Y_$;
- le groupe variétal GPRECO est un paramètre technique d'où la nécessité d'accéder au vecteur $W_$, également requis avec l'option CODETRADTEC dans une évaluation logique; le paramètre technique IPLT est par ailleurs forcé à la valeur du paramètre initial IWATER pour la valeur `"snu"` du paramètre CODEPLANTE.
- les citations de IWATER et du paramètre initial RESPERENNE0 expliquent la présence de $P_$;
- l'option d'édition CODESIG est contenue dans $K_$;

Ces variables, dont une vingtaine d'options, sont regroupées dans des rubriques : développement, début de végétation, feuillage, interception du rayonnement, croissance en biomasse, répartition entre organes, croissance et rendement, racines,[66] gel, eau, azote. La dernière rubrique contient le groupe variétal identifié par un nom.

Quelques contrôles sont réalisés, dont certains closent l'exécution avec un message sommaire.

Groupe variétal Le groupe variétal est repéré par un rang, paramètre technique géré dans *Gpreco*. Tous les groupes variétaux d'une plante sont énumérés en séquence en fin de fichier.

1. La lecture s'effectuait jusqu'à la dernière version 6 dans une boucle de pas unitaire jusqu'au rang recherché, les valeurs étant gérées dans une suite de 17 variables simples.
2. La nouvelle version 6 introduit un nombre de variétés, lu avec les paramètres techniques, pour pouvoir optimiser le choix variétal. La lecture s'effectue désormais dans une boucle depuis 1 jusqu'à 12.

Erreurs de lecture Les paramètres descriptifs du groupe variétal sont lus dans une boucle de 12 itérations, alors que le fichier de paramètres fourni contient un nombre inférieur de descriptifs variétaux : la fin de fichier est repérée avec le code de retour de la fonction *Texte()* (3.8.4).

Corrections dans l'unité de lecture

Deux paramètres sont éventuellement forcés immédiatement :
- CODEBESO, option de calcul des besoins en eau, est forcée à 2 si une culture associée est présente;
- SENSRSEC, qui code la sensibilité des racines à la sécheresse, est ramenée à 1 pour une valeur supérieure;

[66] STOPRAC, variable textuelle qui contient le stade d'arrêt de croissance racinaire, est à remplacer par une option numérique.

Mises à jour avant acquisition des données climatiques

Plusieurs paramètres sont réévalués dans INITNONSOL avant lecture des séries climatiques, dont les stades de développement , gérés dans des variables de la forme STXXXYYY pour le passage de XXX à YYY, ensuite pondérés par des paramètres COEFXXXYYY. A ces listes s'ajoutent les paramètres DUREEFRUIT, SRDORDEBOUR, NBINFLO, ENVFRUIT, ADILMAX, TEMIN, TEMAX.

D'autres paramètres sont forcés dans INITNONSOL :

- CODINTERCEPT, CODLAINET sont réinitialisés en fonction de CODELAITR ;
- IDEBDORM, IFINDORM sont corrigés en fonction de CODEFROID et CODEDORMANCE ;
- STLEVAMF et STLEVDR sont remis à zéro et NBJGRAIN est remis à 1 pour la valeur "`drp`" de STADE0 ;
- si RAPSENTURG est nulle ou négative, elle reçoit la valeur 0.001 ;

CODETRANSRAD est forcé à 2 dans TESTTRANSRAD si la culture associée est présente.

Mises à jour après acquisition des données climatiques

Des évaluations conditionnelles des stades de développement, ainsi que DUREEFRUIT, sont faites à partir des séries climatiques depuis INICLIM.

Mises à jour au cours des évaluations quotidiennes

Les stades de développement sont évalués ou forcés, les dates étant alors lues avec les paramètres techniques. Les mises à jour sont effectuées lors des évaluations quotidiennes depuis CALAI, DEVELOP, SENESCEN, OFFRNODU, INICOUPE.

D'autres variables lues sont mises à jour :

- INNGRAIN2 est augmenté de $1/100$ dans BNPL si (FLTAGS > 0) et (INNGRAIN2 $=$ INNGRAIN2) ;
- DLAIMAX reçoit la valeur de DLAIMAXBRUT si (CODELAINET $= 2$) ;
- HAUTMAX est corrigée dans FORMPLANTE, sous-programme qui apparaît comme feuille terminale dans des arbres d'appels depuis BIOMAER, CROISSANCE et PHYSIO ;
- EXTIN est réévaluée dans RAYTRANS pour la valeur 2 de l'option CODETRANSRAD ;
- FORME est forcée à 1 et HAUTBASE réévaluée dans EFFEUIL ;
- NBINFLO est évaluée dans FRUIT ;

4.3.7 Paramètres climatiques

Le fichier des paramètres climatiques contient une entête, lue immédiatement après acquisition des paramètres de la plante,[67] suivie des données climatiques,

[67] Depuis LECTURES pour la version 6, PRINCIP antérieurement.

lues plus loin depuis LECSTAT appelé dans une boucle d'initialisation des campagnes culturales (4.4.3),[68] avec un mécanisme de retour sur les enregistrements déjà lus pour reconstituer les séries calendaires.[69]

Constantes et contexte climatiques

La lecture des séries climatiques est précédée de traitements préliminaires.

L'entête du fichier climatique contient cinq variables directement lues depuis LECTURES pour la version 6, antérieurement depuis PRINCIP, soit cinq paramètres :
- $Nometp$, textuelle, option de calcul de l'évapotranspiration ;
- $Alphapt$ et $Parsurrg$ pour le calcul du rayonnement ;
- $Nbans$, nombre de campagnes culturales ;
- $Latitude$, latitude du lieu ;

Les lignes suivantes contiennent normalement 12 champs lus depuis LECSTAT :
- le premier gère une valeur textuelle copiée dans la variable STATION ;
- les trois champs suivants fournissent la date sous la forme $< an >< mois >< jour >$ mais seule l'année est lue et affectée dans ANSEMIS ;
- le numéro du jour calendaire est affecté dans JULZERO ;
- suivent les valeurs du jour pour 7 séries climatiques ;

  L'enregistrement du premier jour est lu une première fois pour initialiser les bornes de la boucle de lecture des séries climatiques, puis une seconde fois comme première itération dans la dite boucle.

Ces opérations de lecture qui précèdent l'acquisition des séries climatiques sont finalement regroupées dans la fonction $Lecconscli()$,[70] laquelle reçoit en paramètres la table des codes et les vecteurs $K_$ et $M_$ respectivement pour les constantes et les données contextuelles de la campagne.

- Des nombres de jours d'années sont ensuite évalués afin de gérer les années bissextiles[71] pour la lecture des séries climatiques, à savoir :[72]
  - NBJSEMIS, nombre de jours de l'année ANSEMIS ;
  - NBJRECOL, nombre de jours de l'année ANSEMIS + 1 ;
  - NBJANREC = NBJSEMIS si (CULTUREAN = 1), NBJRECOL sinon ;
- Le traitement des séries décadaires si (PAS $\neq$ 1), option introduite par le paramètre initial CODPAS (4.3.2), sans objet, est supprimé lors de la génération du code.[73]
- Un ultime appel de la fonction $SautChamps()$ permet de déterminer le nombre de champs (3.8.4), qui peut être rendu à l'appelant.

[68] Depuis INITIALISATIONS pour la version 6, INITIAL antérieurement.

[69] L'instruction Fortan BACKSPACE est utilisée pour repositionner l'unité logique de lecture au début de l'enregistrement, ou celui qui précède si l'instruction est exécutée dans une boucle.

[70] Fonction $Lecstat1()$ dans la version $Hysafe$, puis $Lecparacli()$.

[71] Cette séquence peut être reprise en utilisant la fonction $Gregoire()$ (4.2.1) pour une évaluation exacte (2100 n'est pas bissextile).

[72] Ces opérations sont refaites dans le sous-programmes INICLIM.

[73] Une liste de conditions logiques est reconnue par la commande $S8$ pour générer les sauts des blocs *alors*.
Cette suppression n'est pas effectuée pour le bloc *sinon* si (PAS = 1).

Ces lectures préliminaires sont pilotées depuis le module de supervision, qui ouvre et ferme le fichier des paramètres climatiques.

Lecture des séries climatiques

La traduction du sous-programme LECSTAT est reprise pour la lecture des séries, après suppression de la séquence intégrée dans *Lecparacli*() : la fonction *Lecstat*() est alors compilable.

Cette lecture est effectuée plus loin après lecture des paramètres du sol (4.3.8) pour reprendre le séquencement Fortran. Le fichier des paramètres climatiques étant à nouveau ouvert, avec le mode "a" pour l'ajout au compte-rendu, l'appel préliminaire de la fonction *SautChamps*() permet d'ignorer l'entête pour démarrer la lecture (3.8.4) en appelant *Lecstat*().

Les lectures sont contrôlées par les bornes inférieure JULZERO et supérieures NBJSEMIS et NBJRECOLTE, nombres de jours des années. La boucle de lecture peut être interrompue avec une fin de fichier si l'année est incomplète :[74] le rang de l'ultime observation est affecté dans la variable $Julfin$, utilisée pour gérer la continuité des périodes entre deux années.

Le champ $< an >$ est relu et la valeur affectée dans la série ANNEE, revue dans INICLIM et utilisée dans les séquences d'édition des bilans après récolte.

Nombre de séries Le code généré pour la version 6 suppose la présence de 7 séries climatiques mais les deux dernières, TTVENT et TTPM, apparaissent comme optionnelles :

1. les champs sont parfois absents bien que les variables soient citées dans la liste de lecture ;[75]
2. la valeur -999.9 figure quelquefois dans ces deux champs ;

Non nullité de l'évapotranspiration Toute valeur lue de l'évapotranspiration TTETP(J) du jour j nulle est remplacée par la valeur `"0.05"`,[76], elle-même remplacée par `"0.1"` dans l'initialisation de la série (INICLIM).

Contrôles pour l'utilisation du module de transfert radiatif

Le sous-programmes TESTTRANSRAD (version 6) est appelé comme ultime instruction dans LECTURES pour vérifier la présence des paramètres utiles pour les deux cultures depuis 1 jusqu'à NBPLANTES :[77] c'est un bloc conditionnel qui

[74] C'est le code de retour de la fonction *texte()* qui signale cette fin de fichier dans la traduction C, laquelle n'est finalement prise en compte dans la boucle que pour la période suivante, les lectures s'effectuant dans un bloc conditionnel.

[75] Dans les anciennes versions, les deux derniers ordres de lecture étaient alors supprimés et la valeur 1 transmise comme ultime paramètre fin de la primitive de lecture *Reel*() de la dernière variable TTRR, ceci pour passer à l'enregistrement suivant (3.8.4) : la fonction *Nchamps*() peut aujourd'hui être appelée depuis LECCONSCLI afin de rendre le nombre de séries si nécessaire..

[76] La lecture s'effectue avec une décimale, donc si $(x < 0.1)$ alors $x = 0$.

[77] Cette fonction peut ne pas être appelée vu qu'elle n'évalue pas de variables.

s'exécute pour la valeur 2 de l'option CODETRANSRAD et arrête le logiciel en exécutant un STOP fortran dès qu'une valeur n'est pas conforme. On dénombre 7 branchements de sortie de ce bloc, qui génèrent des conditionnelles imbriquées et moult avertissements : la structure syntaxique n'est finalement pas reconnue.

La solution adoptée pour gérer les imbroglios de branchements permet toutefois de générer une fonction qui traduit l'algorithme effectif, la séquence de déroutement étant reprise dans un bloc conditionnel (3.7.3).

4.3.8 Paramètres du sol

Les paramètres du sol sont lus avant l'entrée dans la boucle sur les cultures, pour lesquelles sont acquises les séries climatiques de la campagne, par le sous-programme LECSOL dans le fichier `"param.sol"`,[78] renommé par ajout du suffixe `"sol"` au nom de l'application.

L'analyse syntaxique signale une sortie du corps d'une boucle Pour. Cette boucle démarre depuis 1 et s'exécute jusqu'au rang ICHSL, paramètre initial qui identifie le sol (4.3.2).[79] Une condition logique supplémentaire force la sortie si le numéro du sol lu dans la dernière instruction de la boucle est celui contenu dans ICHSL, d'où les avertissements et une perturbation de l'indentation des blocs dans le code généré.[80]

LECSOL est appelé dans la version 6 depuis le programme principal après LECTURES, avant depuis PRINCIP.

Les sols sont décrits :

1. globalement dans trois enregistrements avec des paramètres du sol classiques, mais aussi des variables discrètes de la forme "CODE..." qui codent des propriétés, et des variables relatives au drainage.[81]
2. pour cinq horizons dans cinq enregistrements avec 8 variables, dont l'épaisseur de l'horizon EPC, et le code TYPECAILLOUX qui renvoie vers la typologie lue avec les paramètres généraux.

Les fichiers des paramètres du sol contiennent des suites de descriptions sous la forme de séquences de huit lignes consécutives avec un numéro de sol NUMSOL au début, lues en format libre.[82]

Corrections après lecture

Quelques parmètres sont forcés dans LECSOL après lecture :

[78] Dans les versions antérieures, la variable textuelle CODAS était initialisée avec la valeur `"PR"`, pour signifier qu'il s'agissait d'une culture principale : une image numérique était créée, cette variable étant consultée lors des évaluations quotidiennes (2.1.5).

[79] La variable de boucle, inutilisée dans le bloc itératif, varie avec le pas 7 jusqu'au rang recherché : elle peut être remplacée par un pas unitaire. Les valeurs de l'itération de rang k remplacent celles de l'itération $k-1$.

[80] Ce test redondant est à supprimer puisqu'il reprend le prédicat de contrôle de la boucle.

[81] PROFDRAIN et ECARTDRAIN codent la profondeur de drainage et la distance entre drains, KSOL mesure la conductibilite.

[82] Dans la version 5, les premières variables étaient lues en format fixe : un espace manquant dans le dernier champ de l'enregistrement suffisait pour générer une erreur fatale lors de la lecture de OBSTARAC.

- CAILLOUX est remise à zéro et TYPECAILLOUX à 1 pour les 5 horizons si le paramètre du sol courant, CODECAILLOUX, est nul ;
- OBSTARAC est forcé à 999 si (OBSTARAC = 1000), profondeur maximale du sol ;
- CAPILJOUR est forcée à 0 si (CODEREMONTCAP = 2)

Une évaluation de HUMCAPIL dans INITNONSOL est signalée mais elle est sans objet car faite avant lecture.

Corrections dans la boucle d'initialisation des cultures

Des mises à jour sont faites lors de l'initialisation d'une culture,

1. dans INITSOL1, appelé avant INITIALISATIONS :
   - PROFHUM, profondeur de la zone humifère, prend la valeur de PROFSOL si (PROFHUM > PROFSOL) ;
   - EPD(I), épaisseur de dispersion dans un horizon, est forcée à 1 si (EPD(I) $\leq$ 0) ;
   - CODEFENTE est forcée à 0 si (CODEMACROPOR = 0) ;
   - PROFDRAIN est forcée à 0 si (CODEDRAINAGE = 0) ;
2. dans INITSOL2, appelé depuis INITIALISATIONS :
   - OBSTARAC prend la valeur de PROFSOL si (OBSTARAC > PROFSOL) ;

Mises à jour au cours des évaluations quotidiennes

Des paramètres liés à la densité apparente du sol sont modifiés en cours de simulation dans TASSEMENT :

- EPC(1), épaisseur du premier horizon, est pondérée par le facteur DA(1)/DATRAVAIL, DATRAVAIL recevant les valeurs des "paramètres nouvelle version" DALABOUR ou DACHISEL ;
- PROFHUM est réévalué en tenant compte de cette pondération ;
- INFIL est pondéré par une valeur rendue par la fonction EFFETTASSSURINFIL(DATRAVAIL) ;
- EPD(I) est, comme dans INITSOL1, forcée à 1 si (EPD(I) $\leq$ 0) ;

Autres traitements après lecture

Les contrôles sont relatifs à des limites codées en clair dans le code et se soldent par l'édition d'un bref message suivi de l'exécution d'un STOP fortran, par exemple si (PROFHUM $\leq$ 0) ou (PROFHUM > 60) ;

Les éditions dans le fichier historique sont pilotées par l'option CODESIG.

4.3.9 Culture précédente

La valeur 1 du paramètre initial CODESUITE requiert la présence du fichier `"recup.tmp"`, produit par une autre simulation (4.7), ici renommé avec le nom de l'application suffixé par `"pre"`.

Les lectures effectuées dépendent de la valeur du paramètre général CODEINITPREC.

1. Les données relatives à la culture précédente, classe 70, sont lues si CODEINITPREC = 2.
   Les paramètres du sol, restitués avec les données de la culture précédente, sont relus.[83]
2. Si CODEINITPREC $\neq$ 2, une simple réinitialisation est faite avec la lecture de quatre variables de contrôle :
   NBJRECOL0,IFWATER0, NHE,DERNIER_N.[84]

Ces séquences sont gérées dans une structure conditionnelle contenue dans INITNONSOL, appelé dans la boucle sur les cultures depuis INITIALISATIONS, INITIAL dans les versions antérieures appelé directement depuis PRINCIP. Le code de lecture est extrait du fichier source "`Initnonsol.c`" pour construire la nouvelle fonction *Lecprec*(),[85] les blocs étant localisés dans le fichier descriptif des conditionneles "`stics.sis`" (1.3.7).[86] En l'absence d'avertissements pour ces blocs conditionnels, la fonction *Lecprec*() est alors compilable.

Variables de contrôle

L'entête du fichier contient les informations nécessaires pour gérer la continuité pour les deux cultures.

- NBJRECOL0 et IFWATER0 lues dans INITNONSOL, NBJANREC et IFWATERAVANT éditées depuis RECUP, gèrent la période de simulation précédente ;
- DERNIER_N contient le rang de l'ultime simulation de la culture précédente, N = MAXWTH ;

Si la continuité calendaire n'est pas assurée, le programme s'arrête sur un STOP fortran qui clôt l'exécution : cette séquence peut être supprimée, la boucle sur les cultures n'étant pas reprise sous cette forme (4.4).

Le nombre d'horizons NHE figure aussi dans cette entête.

Etat du sol

Les variables descriptives du sol non globales sont définies successivement pour :

- au plus cinq horizons,[87] avec la variable de contrôle NHE lue, et modifiée dans INITNONSOL (4.4.2) ;
- la profondeur de la zone humifère 100 ;
- la profondeur du sol 1000 ;

Il s'agit généralement de variables temporelles évaluées en cours de simulation.

[83] Les valeurs lues antérieurement (4.3.8) par LECSOL, et DA évaluée à partir de DAF, sont perdues : DA apparaît comme temporelle car mise à jour dans TASSEMENT.

[84] DERNIER_N est mise à jour dans SENESCEN et NHE réévaluée dans SORTIE et TASSEMENT.

[85] *Liravant*() extrait de "`Initial.c`" dans la version *Hysafe*.

[86] L'utilisation de variables locales et de boucles fait que le code édité dans "`D/Lirepara.c`" (3.9.1) n'est pas suffisant.
Le bloc **alors** qui suit la condition logique (CODEINITPREC = 2) $\wedge$ (CODESUITE $\neq$ 2) est repris intégralement ; dans le bloc **sinon**, la conditionnelle pilotée par (CODESUITE = 1) est également reprise pour lire quatre variables.

[87] Le format de lecture décrit 7 champs numériques.

Décomposition des résidus organiques dans le sol

Les résidus de la culture précédente sont gérés dans les matrices CRES,CBIO,NBIO définies pour les huit types prédéfinis de résidus (4.3.5) et le vecteur CHUM. S'ils sont lus, la profondeur est limitée à 40 centimètres, borne supérieure des boucles de lecture dans INITNONSOL et de restitution pour la simulation suivante dans RECUP. Si la culture précédente n'est pas prise en compte, ces variables sont remises à zéro jusqu'à une profondeur de 60 centimètres :[88] CHUM est alors initialisé en fonction de valeurs de paramètres du sol.

Ces variables sont temporelles car mises à jour dans la boucle quotidienne par appel de APORES depuis :[89]

- FERTIL pour la fertilisation ;
- RESIDUS pour la minéralisation cumulée des résidus de la culture après récolte ;
- SENESCEN pour la sénescence de la matière sèche ;
- EFFEUIL pour les opérations d'effeuillage ou rognage ;

et directement dans MINERAL.

CRES est par ailleurs mis à jour directement dans ces sous-programmes ainsi que CHUM dans FERTIL.

Dynamique des résidus Cinq variables descriptives du processus de décomposition des résidus[90] sont lues si la culture précédente est prise en compte.[91] Des mises à jour sont effectuées dans la boucle quotidienne depuis :

- APORES pour WB,KRES,HRES ;
- FERTIL,RESIDUS,SENESCEN,EFFEUIL pour WR ;

Profils du sol

Les quantités d'eau et d'azote et la température du sol sont évaluées dans des couches de un centimètre et gérées dans les vecteurs HUR, NIT, dits profils. Si la culture précédente n'est pas prise en compte, l'initialisation s'effectue à partir de paramètres initiaux définis pour les horizons (4.3.2). Les vecteurs sont ensuite transmis pour ajustement au sous-programme AJUPRO.[92]

La température du sol est gérée dans TSOLVEILLE, profil lu avec les données de la culture précédente, ou initialisé dans INICLIM.

Ces profils sont ensuite mis à jour quotidiennement :

- NIT dans FERTIL, MINERAL et DENIT appelé depuis MINERAL, LIXIV, MAJNSOL, TASSEMENT ;
- HUR dans LIXIV, TASSEMENT ;

[88] Les tables sont déclarées pour une profondeur de 100 centimètres et la décomposition des résidus traitée jusqu'à une profondeur maximale de travail de 60 centimètres.

[89] L'arbre des appels est consultable dans le fichier "`stics.lien`".

[90] WR,KRES, KBIO,HRES,WB.

[91] KBIO est aussi lu avec les paramètres généraux (4.3.3).

[92] AJUPRO n'accède pas aux variables communes, les déclarations n'étant pas incluses. Les échanges s'effectuent au moyen des paramètres, le profil ajusté étant évalué localement dans le paramètre formel Q, et récupéré dans les paramètres effectifs HUR et NIT.

– TSOLVEILLE dans TEMPSOL ;

Horizons du sol

Les tables HINIT et NO3INIT, initialisées dans INITSOL1, sont lues avec la culture précédente : HINIT est réinitialisée en l'absence d'une telle culture.

TS est mise à jour quotidiennement dans SORTIE.

DA, initialisée avec les paramètres du sol dans INITSOL1, est relue avec les paramètres du sol alors restitués avec les données de la culture précédente.

Stocks azote et carbone du sol

Huit variables descriptives (NHUM,CHUMT,NHUMI,CR,NR,CB,NB) sont lues pour la culture précédente, ou mises à jour : elles sont ensuite évaluées dans MINERAL (sauf NHUMI) et NHUM dans FERTIL.

4.3.10 Autres lectures

La lecture des paramètres optimisés, les listes d'édition et la lecture des valeurs de forçage ne sont pas reprises dans cette étape.

Optimisation de paramètres

Des paramètres lus antérieurement peuvent être remplacées par des valeurs optimisées produites par une autre application et contenues dans le fichier "`param.sti`" : l'option CODOPTIM permet d'effectuer cette optimisation pour les cultures principale ou associée dans la version 6 en affectant les valeurs 1 ou 2 dans la variable commune IPL. Les variables sont repérées par un numéro et les nouvelles valeurs sont affectées dans la variable commune correspondante dans une longue suite d'instructions conditionnelles dans le sous-programme LECOPTIM, toute variable candidate à l'optimisation étant citée.

Le code C généré pour cette fonction, qui traduit la séquence Fortran avec une instruction conditionnelle par variable optimisable, n'est pas repris : l'acquisition de ces valeurs pour une liste de paramètres pourrait s'effectuer dynamiquement dans une boucle par recherche des variables dans le dictionnaire puis affectation dans la zone allouée (3.1.2).

Listes d'édition

Les listes des noms de variables retenues pour les éditions finales, contenues dans les fichiers "`var2.mod`" et "`rap.mod`", sont lues pour la version 6 par les sous-programmes LECTURESST2 et LECTUREVARIABLESRAPPORT. Une autre liste d'édition est lue en cours de simulation depuis LECPROFI dans "`prof.mod`".

Les valeurs des variables choisies sont affectées dans les séquences d'édition par appel des sous-programmes LECSORTI, LECRAP et LECPROFI.

Assimilation de données

Certaines options conduisent à substituer en cours d'évaluation des mesures ou états observés aux valeurs simulées : les dates des stades lues avec les paramètres techniques (4.3.5) relèvent de cette catégorie.

La valeur "`feuille`" de l'option CODESIMUL offre la possibilité de prendre en compte des mesures des indices foliaires, alors lus par le sous-programme LECLAI au début de la boucle des cultures, appelé deux fois depuis le sous-programme INITIALISATIONS pour les cultures associées.[93]

La version 6 introduit un autre groupe de variables dites de forçage des besoins en eau et en azote deux séries annuelles.[94] La prise en compte de ces variables, contrôlée par l'option CODEPITAF, est indépendante du nombre de cultures : la lecture s'effectue après acquisition des paramètres du sol depuis LECEPITAF et des initialisations sont ensuite faites au début de la boucle d'enchaînement des cultures dans INITEPITAF. Ces variables sont toutes temporelles car ensuite mises à jour lors des évaluations quotidiennes (2.4.1).[95]

4.4 Evaluation des conditions initiales

Les initialisations des variables temporelles et les évaluations des conditions initiales étaient jusqu'à la version 5 faites dans deux longs sous-programmes

- INITIAL pour les variables indépendantes des données climatiques ;
- INICLIM après lecture des séries climatiques de la campagne ;

La version 6 a repris à l'identique ce séquencement avec d'autres sous-programmes, dont certains regroupés dans INITIALISATIONS, qui reprennent le code antérieur et ajoutent des séquences relatives aux cultures associées.

Les conditions initiales sont évaluées dans une boucle du programme principal, antérieurement du sous-programme PRINCIP, qui répète une même culture avec des séries climatiques lues pour plusieurs campagnes : cette boucle n'est finalement pas reprise dans la traduction C.

Les fonctions d'initialisation sont regroupées dans la fonction C $EtatInitial()$, à l'exception de quelques options reprises dans le module de supervision.

Le compte-rendu de cette étape d'initialisation reprend le fichier historique avec le suffixe "`_hist`".

4.4.1 Initialisations préliminaires

Des variables de contrôle de la boucle sur les cultures sont quelquefois utilisées dans des séquences conditionnelles. Du fait de la suppression de cette boucle :

[93] La variable commune ENS reçoit successivement les valeurs des pseudo-constantes AS puis AO (1.2.4) : la séquence est contenue dans une boucle définie avec la variable IPL qui varie de 1 à NBPLANTES, ce qui génère finalement quatre appels si (NBPLANTES = 2).

[94] EPIQNMAX, EPIBN, EPITN, NOFFN, CODEHUMOFFN, DOSINMIN,DOSIWMIN, PONDEALPHA, PONDEBETA, NORMEALPHA, NORMEBETA, JULOFFN, TEMPOFFN.

[95] Cette option est neutralisée dans les applications actuelles depuis *Stics*0.

- la variable de boucle commune NUMCULT reçoit la valeur 1 ;
- la borne supérieure NBANS, lue dans l'entête du fichier climatique (4.3.2), ensuite adaptée au contexte, est forcée à 1 ;

Cultures associées Le sous-programme INITSIMUL apparaît dans la version 6 pour initialiser avant l'entrée dans la boucle sur les cultures un certain nombre de variables qui gèrent des couples de valeurs, voire des quadruplets avec les pseudo-constantes AS et AO.[96]
La fonction correspondante, *Initsimul*(), est directement compilable.

4.4.2 Initialisation de la culture

La séquence d'initialisation est refaite pour tout nouvelle campagne avec les paramètres du sol pour la même plante cultivée et le même itinéraire technique.

Initialisation du sol

Les variables descriptives du sol sont évaluées dans deux sous-programmes.

1. INITSOL1 est appelé depuis le programme principal dans la boucle sur les cultures.
   L'analyse syntaxique signale une boucle DO avec branchement externe : elle est remplacée par une structure itérative, la condition de sortie étant l'ultime horizon de rang 5, ou un horizon d'épaisseur nulle.
2. INITSOL2 est appelé ensuite depuis INITIALISATIONS, immédiatement après mise à jour de trois variables[97] si la culture associée est présente. Le paramètre général ZESX et 5 paramètres du sol (4.3.8) sont corrigés.

Les fonctions C générées sont directement compilables.

Trois variables descriptives des horizons du sol, DA, HINIT et NO3INIT, sont initialisées ici : les valeurs peuvent être remplacées par celles ensuite lues depuis INITNONSOL si la culture précédente est prise en compte (4.3.9).

Initialisation de la plante, de l'itinéraire technique et des profils

Appelé depuis INITIALISATIONS, INITNONSOL, long sous-programme de plus d'un millier de lignes qui reprend les séquences de INITIAL des versions antérieures, effectue des évaluations après lecture des paramètres pour :

1. les techniques culturales et les dates, dont la construction des séries des apports quotidiens à partir des descriptifs des opérations techniques ;
2. la plante cultivée ;[98]

[96] Le paramètre général CODEMSFINAL est forcé à 2 si (CODEPERENNE = 2) ∧ (CODEINITPREC = 2) ∧ (NBANS > 1).
[97] DASSOINIT, DENSITE, DASSOINITEQV.
[98] L'appartenance aux classes 10 à 40 des paramètres de la culture (2.4.4), est définie dans cette fonction pour un ensemble de variables communes exogènes.

3. les profils du sol, soit à partir de la culture précédente,[99] soit au moyen de paramètres initiaux définis pour les horizons (4.3.2), les profils étant alors transmis au sous-progamme AJUPRO pour ajustement.

Après suppression des séquences de lecture reprises dans *Lecprec*(), *Initnonsol*() est compilable.[100]

La variable lue NHE (4.3.9) est réutilisée successivement comme compteur et variable de boucle

L'analyse syntaxique ne signale que deux anomalies liées à la remise à zéro du terme de rang N pour les couples de séries IRAZO, successivement pour les éléments AS et AO : N est localement une variable de boucle initialisée avec la valeur de la variable commune N, horloge calendaire (3.2.6) indéterminée ou nulle dans la phase d'initialisation.

Ajustement des profils Une révision du sous-programme AJUPRO d'ajustement, appelé depuis INITNONSOL, paraît s'imposer pour reconstituer sans ambiguité les algorithmes.[101]

- Une première boucle se termine sur une instruction d'affectation qui appartient au corps de la boucle : le code C est généré en conséquence.
- Une second boucle en contient une troisième interne réduite à une instruction de sortie qui gère finalement le passage à l'horizon suivant : elle est remplacée par une structure itérative contrôlée par un prédicat.

AJUPRO appelle SPLINE, qui n'accède pas aux variables communes mais utilise des variables homonymes.

Cultures associées et enchaînement des cultures

Appelé pour chaque plante depuis INITIALISATIONS, INITCYCLE initialise des variables qui gèrent des couples de valeurs. Une séquence initiale s'exécute pour les cultures de rang > 1, avec une initialisation de séries[102] pour les cultures pérennes, et un appel de INITSIMUL autrement,[103] séquence supprimée lors de la génération de code pour une culture pure, qu'il faudrait revoir en présence d'une culture associée.

[99] La séquence de lecture est reprise dans *Lecprec*() (4.3.9).

[100] Une solution immédiate efficace est de remplacer le premier bloc *alors* par une instruction vide. Pour une meilleure lisibilité, la conditionnelle peut être pilotée par la négation de la condition logique et le *sinon* supprimé.

[101] Le démarrage de la culture dans les couches superficielles du sol est simulé par des processus dichotomiques avec des discontinuités entre couches qui font que des variations marginales dans l'évaluation des conditions initiales peuvent introduire des écarts infinitésimaux qui se propagent jusqu'à la récolte, quelques grammes/hectare, et se confondent avec les erreurs de précision.

[102] Pour dix séries (cinq variables), les termes de rang 0 reçoivent les valeurs des termes de rang N, variable commune de contrôle de la boucle quotidienne, apparemment non initialisée dans ce contexte.

[103] Cet appel génère des effets de bord : INITSIMUL met à jour des variables communes non citées dans INITCYCLE.

Ajustement des paramètres techniques pour la culture associée

TESTVARTECH est ensuite appelé pour forcer certains paramètres techniques (4.3.5). Après suppression des blocs qui traitent de la culture associée, la fonction C est vide : elle n'est pas reprise.

4.4.3 Initialisation de la campagne culturale

La campagne culturale est initialisée pour la période de simulation.

1. Les données climatiques sont alors lues par LECSTAT et affectées dans des séries calendaires. L'option décadaire n'est pas reprise.[104]
2. Les opérations techniques sont repérées par une date calendaire et décrites par un ensemble de variables, éventuellement empilées pour des opérations répétitives alors désignées par leurs rangs.[105]
   Les données observées, valeurs de forçage, sont aussi repérées par des dates.[106]
3. Les simulations sont gérées dans des séries chronologiques définies à partir du premier jour de la simulation. Une campagne culturale peut s'étendre sur deux années calendaires : les simulations sont faites sur une campagne d'une année mais les tables demeurent définies pour deux années dans la version 6.[107]

La restitution des données dans l'espace de la simulation s'effectue en partie dans cette phase d'initialisation pour la période simulée, mais d'autres évaluations et des mises à jour sont quotidiennes (4.5.2).

Le sous-programme INICLIM gère :
- les aspects chronologiques avec les décalages pour les données climatiques ;
- la construction des séries à partir des dates des opérations techniques ;
- les autres initialisations de la boucle des simulations quotidiennes ;

Dans la version 6, il est appelé dans une boucle depuis 1 jusqu'à NBPLANTES, avec la variable de boucle commune IPL utilisée comme indice pour désigner les éléments des couples pour les variables citées.
La fonction *Iniclim*() et toutes les fonctions appelées sont directement compilables.

[104] Les séquences de traitement sont supprimées dans le code généré (4.3.7) : les sous-programmes DEBUTDEC et NUMDEC et la fonction NBJOURD ne sont plus appelés, et les déclarations des prototypes sans objet.

[105] Les séries des apports d'eau d'irrigation AIRG et d'azote ANIT (4.5.11 et 4.5.12) sont initialisées ici.

[106] Toute valeur IXXX $\neq$ 999 est une date qui signifie que le stade de développement XXX est observé : la mise à jour à la date indiquée dans la boucle quotidienne fait que ces paramètres apparaissent comme modifiés en cours de simulation (4.3.6).

[107] La version 5 produite pour *Hysafe* utilise des séries annuelles : le controle dynamique des bornes (3.1.2) permet de localiser les valeurs citées en clair dans les instructions par examen du fichier "`source.dec`", ici 366, et 200 pour la profondeur du sol.

Boucles d'initialisation de séries chronologiques

La variable commune N, variable de la boucle des évaluations quotidiennes qui contient le rang du jour simulé, est utilisée localement dans INICLIM pour initialiser des séries chronologiques :

1. après évaluation dans une boucle sur les jours calendaires pour les données climatiques, avec appels :
   - des fonctions et sous-programme qui n'accèdent pas aux variables communes ;[108]
     - RGEX évalue le rayonnement extraterrestre et appelle la fonction de déclinaison DECANGLE ;
     - TVAR rend la tension de vapeur saturante ;
     - CALCEEQ évalue l'évaporation à l'équilibre et appelle TVAR ;[109]
     - PHOTPD est un sous-programme qui rend la durée et la photopériode du jour dans des paramètres (note 202 section 4.5.17) ;
   - du sous-programme CALPENMAN qui accède aux variables communes et utilise N comme valeur d'indice ;
2. comme variable de boucle sur les rangs des simulations plus loin avec appel de BFROID qui accède aux variables communes et effectue des mises à jour des termes des séries ainsi désignés ;[110]

La traduction C utilise la variable de boucle locale n, copie de la variable commune N, transmise en paramètre (3.3.1) : la valeur de sortie de boucle est finalement restituée à l'appelant.

Séries chronologiques exogènes

Les séries sont souvent évaluées en fonction de valeurs de variables temporelles évaluées en cours de simulation, qui reçoivent une valeur initiale avant utilisation.

L'examen des arcs définis dans INICLIM (fichier "`stics.arc`") et dans les unités appelées CALPENMAN et BFROID permettrait de repérer les séries exogènes, accessoirement réinitialisées dans la boucle quotidienne et classées comme temporelles de ce fait.

Période de simulation

Les jours de début et fin de simulation sont initialement les rangs lus avec les paramètres initiaux IWATER et IFWATER, définis sur une période de deux ans à partir du premier jour de l'année d'installation de la culture, soit l'intervalle

[108] Les méthodes numériques sont à reprendre, l'utilisation des fonctions trigonométriques directes et réciproques, les facteurs d'échelles et les valeurs citées (note 13) étant sources de variabilité selon les compilateurs et de dérives quant à la précision (4.1.2).

[109] Dans ce contexte, RNET est une variable locale et non la variable commune homonyme.

[110] Ce sous-programme utilise la fonction intrinsèque Fortran MINVAL qui recherche un minimum dans une table (note 21) mais l'intervalle est indéterminé : une valeur nulle de la borne supérieure est transmise pour localiser l'erreur à l'exécution dans la traduction C.

[1, 731] : les données climatiques sont en conséquence lues sur deux années calendaires si nécessaire et affectées dans des tables définies avec la taille requise. La durée d'une campagne culturale est en revanche dans cette version restreinte à l'année, soit l'intervalle [1, 366].

Pour gérer ces deux bases temporelles, le sous-programme INICLIM modifie les paramètres initiaux IWATER et IFWATER en fonction des valeurs des variables JULZERO et JULFIN précédentes, ce qui fait que la période de simulation n'est plus celle spécifiée, ou que la simulation est interrompue.

- La culture étant installée à la date IPLT, paramètre technique, si IWATER > IPLT, alors IWATER reçoit la valeur de IPLT.
- La première observation climatique étant disponible à la date JULZERO, si IWATER < JULZERO, alors un STOP Fortran clôt l'exécution.
- La dernière observation climatique étant disponible à la date JULFIN, si IFWATER > JULFIN , alors IFWATER reçoit la valeur de JULFIN.[111]

L'amplitude de la période de simulation est la différence

$$\text{MAXWTH} = \text{IFWATER} - \text{IWATER} + 1$$

ramenée à la valeur 366 si MAXWTH $\geq$ 366, la variable de boucle N variant ensuite depuis 1 jusqu'à MAXWTH.

Construction des séries

Les séries sont ensuite définies sur cette période, ce qui nécessite de retrouver des dates et de les restituer dans l'espace de la simulation :

1. pour les opérations techniques, uniques ou repérées par un rang (travail du sol, fertilisation, irrigation) ;
2. pour des valeurs observées susceptibles de remplacer des valeurs simulées ;
3. pour les données climatiques, avec reconnaissance d'une date initiale puis lecture séquentielle ; [112]

4.4.4 Autres initialisations

D'ultimes initialisations sont ajoutées dans le programme principal avant les évaluations quotidiennes.[113]

Changements climatiques

Des mises à jour du paramètre général CO2 sont faites dans les dernières mises à jour de la version 6 pour les décennies 2070, 2080, 2090.

[111] Cette opération est effectuée NBANS vaut 1.

[112] Les numéros des jours sont ensuite lus dans une variable locale, mais inutilisés.

[113] Deux variables de gestion des coupes fourragères, SIONCOUPE et NBREPOUSSEFAUCHE, sont mises à zéro.

Forçage des apports en eau et azote

Des initialisations sont effectuées pour la valeur 1 de l'option CODEPITAF, paramètre général, par appel du sous-programme INITEPITAF : cette option neutralisée dans *Stics*0 (4.3.10).

La variable NUMEROAPPN reçoit la valeur 1 si (CODEDATEAPPN = 1), option lue "nouvelle version".

4.4.5 Validation de l'état initial

L'appel

Stics0 <*appli*> 0

entraine la restitution de l'état après initialisation (4.2.6).

Edition des états de la version Fortran

Un répertoire *Fmake* est créé pour accueillir la version Fortran. Le code source Fortran est recopié, ainsi que le *makefile* généré et les sous-programmes de suivi qui éditent les vecteurs du métamodèle dans des fichiers (3.9.1). Les appels sont ajoutés dans le programme principal, avant le démarrage de la boucle quotidienne pour l'état initial, à la fin de cette boucle pour les états quotidiens.

L'application Fortran est ensuite exécutée dans le répertoire fils *Fmake/appli*. L'exécution démarre avec la question

```
suivi jour?
```

L'état est édité pour le jour transmis, 0 correspondant à l'état initial, une valeur négative étant ignorée.

Pour tout vecteur X du métamodèle, SUIVIX_ édite dans le fichier "X_" les noms et valeurs des variables numériques vectorisées. Les valeurs nulles ne sont pas éditées pour les termes des matrices et les composantes des vecteurs, ceci pour réduire la taille de fichiers textes qui peuvent être consultés.[114]

1. **Variable simple**
   Le nom est suivi de la valeur, même nulle.
2. **Vecteur**
   Le nom est édité sur la ligne : les composantes non nulles sont éditées sur les lignes suivantes avec l'indice et la valeur, jusqu'à l'occurrence d'un indice négatif.
3. **Matrice**
   Le nom est édité sur la ligne : les termes non nuls sont édités sur les lignes suivantes avec les indices et la valeur, jusqu'à l'occurrence d'un indice ligne négatif.

[114] Les deux matrices descriptives du système racinaire, RL et DRL contiennent 731×1000 termes.

Comparaison des évaluations C et Fortran

Pour pouvoir effectuer les comparaisons, une version Fortran est d'abord exécutée pour le jour 0, dans un répertoire fils de *Fmake* dont le nom est celui de l'application, afin d'éditer l'état initial. L'appel dans le répertoire d'exécution de la version C

$$Stics0 <appli> 0\ Fmake$$

entraîne la recherche des fichiers "`X_`" produits par l'exécution Fortran.

Les vecteurs C et Fortran sont comparés : les divergences au seuil de précision choisi (4.1.2) sont éditées dans le fichier "`appli_trace`", avec le nom de la variable STICS, puis la liste des unités de programmes qui effectuent des mises à jour de ces variables dans des expressions.[115]

Certaines divergences sont sans conséquence :

- variables communes utilisées comme variables de boucle et ensuite inutilisées ;
- valeurs liées au mode contrôle de la lecture dans une boucle finale interrompue sur une fin de fichier (3.8.6).

Mise au point

Les versions C et Fortran sont exécutées simultanément et les instructions suivies au pas à pas si nécessaire sous le metteur au point *gdb*.

La mise au point est pilotée depuis la version C, laquelle permet de bien visualiser le déroulement des algorithmes à travers le langage et l'indentation des blocs. Elle fournit en commentaire les rangs absolus des instructions Fortran (1.1.6), les rangs des instructions d'ouvertures et fermetures en début et fin de blocs, ainsi que les numéros de lignes correspondantes dans le code source, ce qui permet de placer immédiatement les points d'arrêts dans la version Fortran.

Le graphe des relations entre variables est utilisable pour suivre le processus d'évaluation à travers les dépendances entre les variables citées.[116]

Variables textuelles Dans l'application *Vstics*, les valeurs textuelles sont remplacées par leurs images numériques, vectorisées : toutes les fonctions échangent les vecteurs du métamodèle.

4.5 Simulations quotidiennes

Le modèle STICS simule pour une campagne culturale un processus chronologique déterministe qui s'exécute avec un pas de temps journalier pour une période prédéfinie d'au plus une année : cette dynamique est décrite à travers un autre formalisme.

Soit

[115] Les affectations de constantes ne sont pas prises en compte.

[116] Le fichier "`stics.succ`" localise toutes les variables mises à jour depuis toutes les unités de programmes.

X vecteur des variables statiques se simulation,
Y_t le vecteur des valeurs des variables temporelles du jour t,
Y_0 l'état initial de ces variables temporelles avant simulation.

Une simulation quotidienne est vue globalement comme l'évaluation d'une fonction $\mathcal{F}$ telle que

$$Y_t = \mathcal{F}(X, Y_{t-1}).$$

Les évaluations démarrent le jour 1 avec le couple (X, Y_0), et se poursuivent jusqu'au jour n :
l'état final Y_n est un état initial pour la simulation suivante.[117]

Ordonnancement des évaluations La fonction $\mathcal{F}$ est de fait une composition de fonctions, définies avec ces mêmes paramètres (X, Y_{t-1}). En reprenant les huit modules de STICS de la notice *Concepts et formalismes*, cette composition, non commutative,[118] utilise les fonctions

f_1 : stades de développement,
f_2 : croissance aérienne,
f_3 : production des graines ou fruits et répartition de la biomasse,
f_4 : apports et interactions sol-culture,
f_5 : croissance racinaire,
f_6 : bilan hydrique,
f_7 : bilan azoté,
f_8 : transferts dans le sol.

soit

$$\mathcal{F} = f_1 \circ f_2 \circ f_3 \circ f_4 \circ f_5 \circ f_6 \circ f_7 \circ f_8$$

Ces modules sont parfois divisés en sous-modules qui s'exécutent successivement, comme la croissance aérienne avec les fonctions

f_{21} : absorption de l'énergie solaire,
f_{22} : mobilisation des réserves de la plante,
f_{23} : sénescence de la matière sèche ;

Pour certains modules sont définies plusieurs fonctions équivalentes qui correspondent à des méthodes choisies à travers des options.

Synchronisation quotidienne La non commutativité de la loi de composition des fonctions met en évidence des problèmes de synchronisation : les variables temporelles du modèle étant pour la plupart gérées dans des variables informatiques simples, les valeurs de la période t remplacent celles de la période $t-1$, sauf pour les quelques séries gérées dans des vecteurs ou des

[117] **Ce modèle est perturbé dans la version 6 par des références aux données observées des périodes $t+1$, voire $t+k$, $k>1$** : elles sont localisées dans la suite et un récapitulatif effectué en annexe **??** pour la révision du code Fortran.
[118] Cette structuration, initialement reproduite dans le séquencement des appels, est relativement perturbée dans la version 6.

matrices.[119] Dans ces conditions, l'état Y_t n'est défini qu'à l'issue de l'évaluation complète de la fonction $\mathcal{F}$, laquelle définit une transaction : le paramètre Y_t transmis aux fonctions issues de la traduction des sous-programmes Fortran est en conséquence dans un état incohérent. Un autre mécanisme d'allocation sera à mettre en place pour contrôler cette synchronisation.[120]

Boucle des évaluations quotidiennes Les instructions contenues dans le corps de la boucle, définie pour tous les jours d'une période calendaire depuis une date initiale jusqu'à une date finale, sont pour certaines contrôlées par des structures conditionnelles. La traduction reprend au pas à pas ces séquences dans la fonction $Simuler()$ appelée depuis le module de supervision, avec les appels de sous-programmes d'évaluation et quelques affectations : les variables locales utiles sont reprises.

Les messages produits au cours des évaluations quotidiennes, édités pour la version Fortran dans le fichier historique, ou liés à des erreurs ou ambiguïtés, sont repris dans le fichier compte-rendu.

Contrôle des appels Les unités sont parfois appelées alors qu'aucune évaluation n'est à faire : les premières instructions sont alors des retours conditionnels immédiats, parfois avec des séquences brèves de réinitialisation. Ces séquences initiales sont à regrouper dans des fonctions de contrôle, ceci pour pouvoir effectuer directement des appels conditionnels et éviter les sorties multiples initiales.

Décalages dans le temps Quelques évaluations s'effectuent à la période t par consultation des données des périodes antérieures, alors gérées dans des séries chronologiques : il convient d'examiner dans quelle mesure elles pourraient être faites exclusivement en fonction de la période $t-1$.

4.5.1 Intégration des cultures associées dans la version 6

Les fonctions précédentes étaient traduites dans la version 5 dans un ou plusieurs sous-programmes appelés en séquence depuis PRINCIP. Ces sous-programmes demeurent dans la version 6 mais les appels sont effectués dans d'autres unités de programmes pour gérer les itérations sur les couples pour les cultures associées.[121] Deux nouveaux sous-programmes sont introduits :

1. PHYSIO pour la croissance aérienne et racinaire de la plante ;
2. APPORTS pour les apports d'eau, la fertilisation et le traitement des résidus de cultures ;

[119] Parmi les 458 variables temporelles, 34 seulement sont des séries annuelles (27) ou bisannuelles (7) gérées dans des tables, dont quelques séries climatiques ou relatives aux opérations techniques non endogènes.

[120] La dimension temporelle doit en fait être conservée pour gérer les valeurs de la veille et du jour pour un ensemble de séries : les transactions sont à définir au sein des fonctions.

[121] L'ordre d'évaluation est quelquefois modifié : la croissance racinaire f_5 est désormais simulée avant les apports f_4.

La traduction C pour une culture pure supprime ces sous-programmes intermédiaires et reprend les modules et sous-modules des versions antérieures, avec des appels dans le corps de la fonction *Simuler*() : les séquences de contrôle sont à rechercher dans le code de ces nouveaux sous-programmes.

Croissance et développement de la plante

PHYSIO gère les fonctions précédentes selon deux modes différents dans deux sous-programmes :

1. LAIDEV effectue les appels des fonctions d'évaluation dans une boucle sur les deux cultures[122]
   - des stades de développement f_1,
   - de l'absorption foliaire f_{21},
2. CROISSANCE effectue ses appels dans deux boucles imbriquées, respectivement pour
   - les cultures principale et associée contrôlées par la variable commune IPL,
   - les zones ensoleillée et ombragée contrôlées par la variable commune ENS.[123]

   Un branchement conditionnel exclut la séquence *principal / ombragé*.
   Cette structure contrôle trois modes de gestion des évaluations.
   (a) Un bloc conditionnel s'exécute pour toute valeur autre que 'snu' de la variable CODEPLANTE,[124] pour la culture principale ou associée. Sont évaluées
   - la croissance aérienne f_{22},
   - la sénescence foliaire f_{23} pour l'option CODELAITR = 1,
   - le remplissage des grains ou des fruits f_3.

   (b) La croissance racinaire f_5 est simulée pour la zone ensoleillée pour chaque culture, dont 'snu'.
   (c) La répartition de la biomasse f_{24} est évaluée pour l'option CODELAITR = 1 pour chaque culture,
   et pour les zones ensoleillée et ombragée pour la culture associée.

Inversion de dominance PHYSIO appelle le sous-programme DOMINVER à la date où la culture associée dépasse la culture principale, lequel permute les couples (3.2.1).

Eau, fertilisation et résidus

APPORTS gère les apports d'eau et de produits fertilisants pour les couverts végétaux avec des structures de contrôle spécifiques.

[122] La boucle s'exécute depuis 1 jusqu'à NBPLANTES, avec la variable commune de boucle IPL, qui vaut 2 ou 3 en sortie.
[123] La boucle s'exécute depuis AS = 1 jusqu'à AO = 2, avec la variable commune de boucle ENS qui vaut 3 en sortie.
[124] La version C utilise l' image numérique *Codeplante_* (2.1.5) pour une culture simple.

1. CALCPRECIP exécute deux boucles imbriquées, pour
   - les cultures principale et associée contrôlées par la variable commune IPL,
   - les zones ensoleillée et ombragée contrôlées par la variable commune ENS,

   et appelle depuis des blocs conditionnels le sous-programme IRRIG des versions antérieures, lequel évalue les apports d'eau f_4.
2. Après réinitialisation à la valeur 1 des variables IPL et ENS,APPORTS évalue, dans une boucle depuis 1 jusqua $Min(5, \text{N})$,[125] soit les 5 jours qui précèdent, le vecteur ABSOTMP(5) ;
   la variable simple LAITMP est quant à elle évaluée pour la valeur 1 de la variable de boucle.[126]
3. Les apports d'azote et de matières organiques f_{42} sont évalués successivement par :
   - CALAPNENUPVT pour des apports si l'option CODEDATEAPPN, paramètre "nouvelle version", est active ;
   - FERTIL, sous-programme des des versions antérieures ;
   - RESIDUS pour la décomposition et minéralisation des résidus de récolte ;

Le code généré à partir de APPORTS est directement intégré dans la fonction *Simuler*().

4.5.2 Initialisations dans la boucle quotidienne

Certaines évaluations faites lors de l'initialisation de la campagne culturale sont reprises dans la boucle quotidienne, ce qui génère des suites d'avertissements dans l'étape de classement des variables (2.7.8) : l'examen du graphe (fichier "`stics.arc`") permet de retrouver les variables modifiées pour effectuer ces initialisations avant l'entrée dans la boucle.

La version 6 introduit de nouveaux modules qui modifient en cours de simulation des paramètres lus et des variables évaluées dans l'étape d'initialisation, statiques dans les versions antérieures. Ces unités sont appelées
- directement dans la boucle quotidienne pour le tassement du sol et le climat sous abri,
- depuis le sous-programme PHYSIO.

Un moyen pour traduire ces ajouts avec le formalisme précédent est finalement d'ajouter à la fonction $\mathcal{F}$ une opérande gauche f_0 de réinitialisation, ou forçage, pour isoler ces opérations.

[125] L'évaluation fait référence aux périodes antérieures de la série ABSO jusqu'au rang N-ND = 0.

[126] Ces initialisations, à faire en début de boucle, ne sont pas déplacées pour pouvoir vérifier l'équivalence des versions C et Fortran.

Date de dormance

Une évaluation de la date de dormance est introduite dans la dernière version 6 sous conditions juste au début de la boucle quotidienne.[127]

Modifications de la structure du sol et décalage des dates de semis et récolte (ancienne version)

La version 6 prend en compte le tassement du sol lié aux interventions culturales. Ce phénomène est d'abord traité au début de la boucle quotidienne par appel du sous-programme TASSEMENT,[128] puis dans les dernières versions entre les appels de PHYSIO et APPORTS avec deux autres sous-programmes, TASSESEMISRECOLTE et DETASSEMENT (4.5.10).

Cultures sous abri

Les températures minimales, moyennes et maximales sont réévaluées quotidiennement dans le sous-programme CLIMABRI pour la valeur 2 de l'option CODABRI, paramètre technique.

CLIMABRI, introduit dès la version 5, peut être appelé dans une boucle dans l'étape d'initialisation, aucune autre variable commune n'étant mise à jour.

Recalcul de la levée en fonction de l'indice foliaire observé

Les évaluations quotidiennes des stades de développement et de l'absorption de l'énergie solaire sont dans la version 6 gérées dans le sous-programme LAIDV, appelé depuis PHYSIO : elles sont décrites dans les sections suivantes mais LAIDV contient des séquences initiale et finale de forçage.

Prologue RECALCULLEVEE est appelé au prealable pour l'option "`feuille`" de CODESIMUL : le graphe associé met en évidence la mise à jour de trois variables, les stades NLEVOBS et NLAXOBS et le taux de couverture TAUXCOUV. L'analyse syntaxique (3.9.1) signale des anomalies à contrôler.

- Des indéterminations sont consécutives aux références systématiques à la culture associée ou la zone ombragée dans des conditions logiques : elles sont levées dans le code C par annulation des opérandes.[129]
- Les évaluations du jour utilisent la valeur du lendemain pour l'indice foliaire LAI, les valeurs simulées et observées étant gérées dans la même variable.

Epilogue Une ultime séquence s'exécute si CODESIMUL = "`feuille`" :

- si la plante est levée (NLEV > 0), les valeurs nulles des indices foliaires sont remplacées par 10^{-3} ;

[127] Cette évaluation est faite pour la première culture, soit NUMCULT = 1.

[128] La variable locale EPCINI est une table à deux éléments qui ne gère pas un couple de cultures (3.2.1) : elle est déclarée avec un élément supplémentaire pour ne pas être traitée comme gérant des couples de cultures lors de la génération du code.

[129] Ces cas sont répertoriés dans le fichier "`stics.ombre`".

– sinon la date de levée NLEV est la date du jour si les valeurs des indices foliaires sont positives ;

4.5.3 Stades de développement

Le sous-programme DEVELOP est appelé depuis LAIDEV pour toute culture. Il utilise plusieurs unités de programmes : les modules objets sont regroupés dans la macrodéfinition $F1$ du $makefile$, traduction de la fonction f_1 précédente.

Les premières instructions sont des retours immédiats dans deux situations :
– culture annuelle avant le semis ;
– culture annuelle moissonnée après la récolte ;

Neuf évaluations erronées se soldent par l'édition d'un message dans le fichier historique suivi d'un STOP fortran qui clôt immédiatement l'exécution.

DEVELOP simule successivement

1. la levée par appel du sous-programme LEVEE ;
2. les stades de développement ;
3. la date de débourrement pour la vigne par appel du sous-programme spécifique DEBOUR ;
4. la date de récolte avec le sous-programme RECOLTE ;

L'analyse syntaxique signale la sortie d'un bloc *Alors*, lequel ne contient finalement qu'un retour conditionnnel :[130] la fonction est néanmoins directement compilable.

L'analyse des noms symboliques énumère les paramètres de la plante remplacés par les valeurs observées si l'option de forçage est activée (4.3.6).[131]

Desynchronisation Une séquence d'évaluation des températures horaires fait référence à la température minimale du lendemain.

Levée

Le sous-programme LEVEE utilise les fonctions BATTANCE et HUMIRAC déclarées avec des paramètres formels homonymes de variables communes, non accessibles du fait de la non inclusion des fichiers de déclarations (2.2.5), dont trois qui gèrent des couples de cultures.[132]

Le paramètre technique DENSITE est pondéré par un coefficient évalué dans ce sous-programme.

La constante 1000 est utilisée comme majorant pour le contrôle de bornes pour l'accès aux couches du sol.[133]

[130] Le contrôle est rendu à l'appelant si (N $\geq$ NREC) $\wedge$ (NLAN $>$ 0) $\wedge$ (NREC $>$ 0) : un bloc conditionnel suffit.

[131] STLEVAMF,STAMFLAX,STLAXSEN,STSENLAN,STFLODRP,STDRPMAT,STLEVDRP,STDRPDES.

[132] Ces paramètres sont localement des variables simples, la dimension n'étant pas déclarée : les paramètres effectifs sont en conséquence les éléments des couples, sauf pour un appel qui transmet NREC et non NREC(1) ou NREC(2).

[133] HUCC,HUMIN,HUR,TSOL : il s'agit ici de délimiter la zone du semis.

Débourrement de la vigne DEBOUR, appelé depuis LEVEE, fait référence à la température du lendemain.

Evaluation des stades

Les stades sont définis à la date du jour dès qu'une somme de températures atteint un seuil donné, paramètre de la plante lu, parfois pondéré dans la phase d'initialisation par un coefficient, autre paramètre de la plante.[134]

Des effets externes sont pris en compte en fonction d'options lues avec les paramètres de la plante, par appel de sous-programmes également utilisés dans l'étape d'initialisation.

Vernalisation Les besoins en froid sont évalués pour les cultures vernalisantes en fonction des valeurs de l'option CODBFROID, paramètre de la plante, par appel du sous-programme BFROID.

Effet photopériodique La photopériode est évaluée en fonction des valeurs de l'option CODEPHOT, paramètre de la plante, pour le jour calendaire et à la latitude du lieu par appel du sous-programme PHOTPD.

Récolte

Le sous-programme RECOLTE évalue la date de récolte selon plusieurs critères en fonction de l'option CODRECOLTE, paramètre technique. Systématiquement appelé par DEVELOP, il n'effectue les traitements qu'à partir du stade de remplissage des organes de récolte. Les erreurs relatives à l'évolution des teneurs en eau des organes de récolte se soldent par un déroutement qui clôt l'exécution.[135]

Les règles de choix prennent en compte :

- le type de récolte, moisson ou cueillette, codé avec trois options lues avec les paramètres techniques ;[136]
- la date NMAT du stade de maturité physiologique pour les cultures moissonnées ;[137]
- une rupture d'état stationnaire pour les cultures moissonnées, avec une décroissance de la teneur en eau des grains à partir du stade de début de dynamique hydrique ;[138]

[134] Les noms de ces paramètres sont construits avec des codes qui identifient les stades, soit pour le passage de XXX vers YYY le seuil STXXXYYY et le coefficient COEFXXXYYY.

[135] Les messages "*il faut deshybase* > 0" ou "*il faut deshybase* < 0" sont édités dans le fichier historique.

[136] CODRECOLTE, CODCEUILLE, NBCEUILLE, ce dernier étant forcé à 1 dans la phase d'initialisation pour les cultures moissonnées.

[137] Cette date est évaluée tant que la somme des températures SOMCOURDRP, cumulée depuis la date NDRP de début du remplissage des grains, n'a pas franchi le seuil STDRPMAT, paramètre du groupe variétal.

[138] La date de ce stade est évaluée dans le sous-programme EAUQUAL tant que la somme des températures SOMCOURDRP n'atteint pas le seuil STDRPDES, paramètre de la plante. La teneur en eau des grains H2OREC, jusqu'alors invariante, initialisée avec le paramètre de la plante H2OFRVERT, va commencer à décroître d'une quantité quotidienne évaluée par la fonction DESHYDRATE.

- des cumuls de températures tant que des seuils, paramètres de la plante, ne sont pas franchis, soit
  - H2OGRAINMAX, teneur minimale en eau pour les fruits ;
  - H2OGRAINMIN, teneur maximale en eau pour les grains ;
  - CNGRAINREC, concentration minimale en azote des grains de récolte ;
  - HUILEREC, teneur minimale en huile ;
  - SUCREREC, teneur minimale en sucre ;

Si les différents seuils de référence ne sont pas atteints, la récolte est provoquée à la date ultime IRECBUTOIR, paramètre technique.[139]

4.5.4 Absorption de l'énergie solaire

La simulation des stades de développement est suivie du choix de la méthode d'évaluation de l'absorption de l'énergie solaire selon la valeur du paramètre de la plante CODELAITR, soit les sous-programmes appelés depuis LAIDEV dans la boucle sur les deux cultures :

1. CALAI, croissance de l'indice foliaire, suivi de l'appel du sous-programme EFFEUIL pour prendre en compte la réduction de la surface foliaire consécutive aux opérations culturales de rognage ou d'effeuillage ;
2. TAUXRECOUV, taux de recouvrement ;

CALAI et TAUXRECOUV utilise le sous-programme FEUILLE.
EFFEUIL appelle APORES qui traite de l'incorporation des résidus de culture dans le sol.
Les modules objets sont regroupés dans la macrodéfinition $F21$ du $makefile$, traduction de la fonction f_{21}.

Séquences communes Les séquences de démarrage de ces deux sous-programmes sont identiques, avec

1. l'initialisation d'un indicateur de stress en fonction des paramètres généraux CODEINNAC et CODEH2OAC ;
2. une séquence de réinitialisation de l'indice foliaire si le stade levée n'est pas atteint (NLEV = 0), qui reçoit la valeur nulle ou celle du paramètre de la plante LAIPLANTULE, avec sortie des blocs conditionnels et retour vers l'appelant dans tous les cas ;[140]

Deux autres avertissements communs sont liés à des références autres que le jour ou la veille :

- si la date du stade SEN est définie, alors la valeur de l'indice foliaire à cette date est utilisée pour le calcul de l'indice foliaire du jour ;[141]

[139] Ces seuils sont définis pour l'espèce cultivée ou le groupe variétal, ce qui peut poser des problèmes de cohérence : STDRPDES étant relatif à l'espèce et STDRPMAT à la variété, si (STDRPDES > STDRPMAT) le stade de début de dynamique hydrique (note 138) n'est jamais atteint pour une récolte effectuée au stade de maturité physiologique (note 137).

[140] Ces sorties, signalées lors de l'analyse syntaxique, sont traduites à l'identique avec des retours inconditionnels.
L'instruction SOMFEUILLE = 0 est par ailleurs inaccessible.

[141] Cette valeur peut être gérée dans une variable mise à jour à la dite date.

– l'indice foliaire le jour du stade LAN est mis à zéro avec la date de ce stade en indice ;[142]

Les domaines de définition et règles de priorité sont à vérifier pour les expressions de la forme $e^{a \log(d/b)}$,

– a et b désignant respectivement les paramètres de la plante ADENS et BDENS,
– d le paramètre technique DENSITE, accessoirement modifié dans LEVEE.

Forçage de paramètres L'analyse des noms symboliques signale des modifications des valeurs lues de paramètres de la plante depuis CALAI :

– l'option CODLAINET, paramètre de la plante, gère le choix entre DLAIMAX ou DLAIMAXBRUT ;
– STLAXSEN et STSENLAN sont mis à jour dans un bloc conditionnel en fin de sous-programme ;

4.5.5 Réduction de la surface foliaire par effeuillage

Les opérations techniques de rognage ou d'effeuillage, dont les dates sont lues avec les paramètres techniques ou évaluées, sont traitées dans EFFEUILL, appelé immédiatement après CALAI. Elles sont en fait exogènes, avec toutefois des variantes traduites par des options, paramètres techniques.

Ces opérations modifient :

– les paramètres de la plante FORME et HAUTBASE, et RAPFORME dans la version 5 ;
– les valeurs initiales de variables lues par INITNONSOL avec IRES = 2, par effet de bord chez l'appelé APORES ;
– le nombre d'opération d'irrigation toujours par effet de bord comme décrit ci-après ;

L'évaluation peut se solder par un STOP qui clôt immédiatement d'exécution si l'indice foliaire devient nul.[143]

Traitement des résidus foliaires APORES, utilisé pour les opérations de fertilisation (4.5.12), et après récolte pour l'incorporation des résidus de culture dans le sol, est appelé ici pour traiter les résidus foliaires.

L'eau apportée par ces résidus est prise en compte à travers le paramètre formel EAURESIDU à partir de $1mm$ d'eau, mais EFFEUILL appelle APORES avec la valeur 0 pour ce paramètre : l'apport d'eau pour le jour suivant n'est donc pas pris en compte (4.5.11).

4.5.6 Mobilisation des réserves de la plante

Le sous-programme BIOMAER est appelé quotidiennement depuis CROISSANCE pour toute culture : les évaluations sont faites après la levée et jusqu'au

[142] Si (N = NLAN) LAI(NLAN) = 0 est équivalent à LAI(N) = 0.

[143] Le message *"effeuillage un peu drastique : il ne reste plus de feuille !"* est édité dans le fichier historique.

jour de récolte pour les cultures moissonnées.

Une instruction conditionnelle préliminaire intialise des variables si la plante n'est pas levée (NLEV = 0) :

- les termes du jour de deux séries sont initialisés avec la valeur du paramètre "nouvelle version" MASECPLANTULE ou remis à zéro ;
- trois indices reçoivent la valeur 1 ;

des retours inconditionnels rendent le contrôle à l'appelant dans tous les cas.

Le sous-programme RAYTRANS évalue les transferts radiatifs au moyen de deux méthodes choisies en fonction de l'option CODETRANSRAD, paramètre de la plante, traduites dans les sous-programmes

1. BEER pour les cultures uniformes ;
2. TRANSRAD pour prendre en compte la géométrie du couvert ;

Les modules objets sont regroupés dans la macrodéfinition $F22$ du $makefile$, traduction de la fonction f_{22}.

Forçage d'option La seconde méthode apparaissant comme incompatible avec celle du taux de recouvrement pour évaluer l'absorption de l'énergie solaire (4.5.4), l'option CODETRANSRAD est forcée à 1 dans BIOMAER pour la valeur 2 de CODELAITR.

Transfert radiatif

Le sous-programme TRANSRAD pilote un arbre d'appels : les modules objets associés sont regroupés dans la macrodéfinition RAD du $makefile$, incluse dans $F22$.

1. RTRANS évalue le rayonnement sous la culture dominante.[144]
   Il utilise trois tables locales[145] et appelle deux sous-programmes :
   - KDIF reçoit en paramètre les tables précédentes et évalue le rayonnement diffus ;
   - KGEOM évalue le rayonnement direct, en appelant la fonction Fortran TETACRIT,[146], laquelle utilise la fonction de déclinaison DECANGLE citée plus haut pour les initialisations climatiques.
2. RAPRDIF calcule le rayonnement diffus.
3. FORMPLANTE, qui traite de la forme et densité foliaire,
   - modifie le paramètre de la plante lu RAPFORME,
   - référence une variable indéterminée.[147]

L'exécution se traduit par de multiples appels de fonctions trigonométriques pour évaluer des séries quotidiennes finalement liées à la seule latitude du lieu :

[144] RTRANS reçoit le paramètre FORME mis à jour dans la descendance (note 34 section 2.2.4).

[145] HTAB, SOCTAB et AZTAB sont partiellement initialisées à travers des listes Fortran DATA de valeurs décimales et expressions.

[146] La variable N est locale, la variable commune homonyme n'étant pas accessible.

[147] ORIGINEHAUT est une variable commune déclarée mais non retenue car ne recevant aucune valeur : une variable C est ajoutée localement et initialisée à zéro.

ces séquences seraient à réexaminer pour des questions de précision et d'optimisation.[148]

Effet température

Un effet température est évalué dans la version 6 par le sous-programme CALCULERFTEMP.[149]

4.5.7 Sénescence de la matière sèche

Le sous-programme SENESCEN est appelé si la croissance aérienne est évaluée au moyen de l'indice foliaire, soit le code 1 de l'option CODELAITR. Le contrôle est toutefois rendu immédiatement à l'appelant si :

1. la plante n'est pas levée ;
2. l'indice foliaire est nul et il n'existe plus de feuilles jaunes ;[150]

L'effet du gel est évalué au moyen de la fonction GEL,[151] le stress photoperiodique avec le sous-programme STRESSPHOT. Les feuilles sénescentes sont traitées comme résidus de culture par appel de APORES.

Les modules objets jusqu'alors non déclarés sont regroupés dans la macrodéfinition $F23$ du $makefile$, traduction de la fonction f_{23}.

Ruptures de séquence SENESCEN contient une itération gérée par un branchement arrière avec des ruptures de séquences qui isolent des instructions. Le mécanisme d'échappement adopté pour ces contextes est utilisé (3.7.3), et le code repris à vue dans trois situations :

- un bloc conditionnel en situation de gel ;
- un branchement arrière pour une culture de rang $Numcult > 1$;[152]
- une itération complexe décrite ci-dessous qui gère la sénescence ;

Evaluation de la sénescence

La simulation du processus de vieillissement s'effectue par examen des valeurs de variables pour les périodes antérieures dans une structure itérative contrôlée par un prédicat final géré par un branchement arrière : la fonction $Age()$ est en conséquence utilisée dans le code généré pour contrôler les domaines de définition et évaluer la longévité maximale (3.2.6).

[148] Le nombre π est souvent réévalué en local sous la forme $\pi = 4\ \arctan(1)$.

[149] CALCULERFTEMP, déclaré avec une liste de paramètres vide, évalue la variable FTEMP, non négative puisque forcée à la valeur 10^{-2} si (FTEMP > 0).

[150] Si (LAI ≤ 0) $\wedge$ MAFEUILJAUNE ≤ 0) alors DLTAISEN et DLTAMSEN sont remises à zéro avant retour.

[151] La fonction évalue des conditions logiques successives, donc hiérarchisées : si la condition de rang k est vérifiée, l'évaluation correspondante est faite et la valeur rendue à l'appelant. Le résultat est indéterminé si aucune des conditions n'est vérifiée.

[152] Ce branchement contrôle une structure itérative : la forme est conservée (3.7.2) mais une seule itération est effectuée.

Une condition de sortie supplémentaire de ce bloc itératif est traduite par un branchement en avant avec sorties de deux blocs conditionnels : la séquence est reprise par interprétation de la variable générée *cible_* qui, à travers les étiquettes des branchements, restitue les états du système évalué.

Apports de résidus

Les feuilles sénescentes sont traitées comme résidus de culture par appel du sous-programme APORES (4.5.12), avec une valeur nulle pour le paramètre qui gère la quantité d'eau contenue dans les feuilles, ceci afin que ces apports quotidiens ne soient pas pris en compte comme opération d'irrigation (4.5.11).

Forçage de paramètres et mises à jour de valeurs lues

L'analyse des noms symboliques signale les modifications suivantes :
- les paramètres de la plante STAMFLAX,STLAXSEN,STSENLAN,STLEVAMF sont remis à zéro ou modifiés dans des blocs conditionnels ;
- le paramètre technique DENSITE est réduit en cas de gel ;
- DERNIER_N, jour lu pour la culture précédente, est corrigé;
- WR et CRES sont mis à jour dans APORES ;

Rang de la campagne pour les cultures pérennes

Plusieurs séquences sont contrôlées simultanément par le rang de la culture (3.7.2), géré dans la variable de boucle commune NUMCULT, en conjonction avec les valeurs 2 des options CODEPERENNE, paramètre de la plante, et CODEINITPREC, paramètre général.

4.5.8 Production des graines ou fruits

La production de graines ou fruits est simulée pour les cultures en place au moyen de plusieurs unités de programmes appelés depuis CROISSANCE.
- FRUIT ou GRAIN sont choisis en fonction de l'option CODINDETERMIN, lue avec les paramètres de la plante.
- EAUQUAL évalue la teneur en eau,, sucre et huile des organes ;

Les appels sont précédés d'une évaluation de la sensibilité au gel par appel de la fonction GEL.[153]

Les modules objets sont regroupés dans la macrodéfinition $F3$ du $makefile$, traduction de la fonction f_3.

[153]Les paramètres de la plante CODEGELFLO, TGELFLO10 et TGELFLO90 peuvent être transmis directement comme paramètres effectifs, ce qui rend inutile les variables locales utilisées dans CROISSANCE.

Production de fruits

Le sous-programme FRUIT, évalue au préalable le nombre d'inflorescence au stade AMF,[154] puis effectue certains contrôles qui se soldent par un retour ou l'édition d'alertes.[155] Un ultime retour apparaît plus loin pour la valeur nulle du paramètres du groupe variétal DUREEFRUIT (4.3.6).

FRUIT utilise la fonction Fortran réelle FPF,[156] et PFMAXN pour la version 6.

Séquence itérative d'évaluation du remplissage des fruits Le processus de remplissage des fruits est simulé dans une boucle pilotée par un branchement arrière, ultime instruction d'un bloc conditionnel final : elle s'exécute au moins une fois puis tant la valeur de la variable ALLOCFRUIT reste au-dessus du seuil ALLOCFRMAX, paramètre de la plante. Cette structure de contrôle est reconnue et traduite pour la version 6.

Production de grains

Des contrôles préliminaires sont effectués dans le sous-programme GRAIN avant la simulation du processus de remplissage.

- Si le stade DRP n'est pas atteint ou si la récolte est effectuée, le contrôle est rendu à l'appelant après remise à zéro du terme du jour de la série IRCARB.
- Une incohérence qui se traduit par l'apparition du stade DRP alors que la plante n'est pas levée génère un déroutement qui clôt l'exécution.
- Si la date du stade DRP est l'ultime date de récolte NRECBUTOIR, une remise à zéro est effectuée et le contrôle rendu à l'appelant.

Référence aux périodes antérieures Le calcul de la vitesse de croissance moyenne à la date NDRP nécessite un retour sur les périodes antérieures pour la série DLTAMS dans une boucle locale, le délai étant fixé par le paramètre de la plante NBJGRAIN, d'où le recours à la fonction $Age()$.

Teneur en eau, sucre et huile des organes

Les évaluations sont faites dans le sous-programme EAUQUAL si le stade DRP est atteint et si la récolte n'est pas effectuée, la structure étant contrôlée non pas par un retour vers l'appelant mais par un branchement conditionnel vers une séquence finale. Trois évaluations erronées se soldent par ailleurs par une fin d'exécution.

La teneur en eau est évaluée dans EAUQUAL,[157] les teneurs en sucre ou huile

[154] Le paramètre de la plante NBINFLO est forcé pour la valeur 2 de l'option CODCALINFLO.

[155] Le terme ANNEE(IFWATER) génère un appel à la fonction $Age()$, comme pour la gestion des processus de vieillissement.

[156] La version 5 utilisait des communs locaux pour partager des variables avec l'appelant : PFMAX, AFPF et BFPF dans les listes des communs TOM2 et TOM2DOMI, DEVJOURFR dans UDEVFR. Une autre ambiguïté résultait de la déclaration locale des variables NFRUITP et PDSFRUITP homonymes de variables communes mais non incluses.

[157] La version 5 utilisait la fonction DESHYDRATE avec le commun local EAUPL.

par la fonction BIOCHFR.

La compilation signale une erreur consécutive à l'utilisation de la variable locale fortran DEFAULT, mot réservé des langages informatiques à remplacer.

Répartition de la biomasse entre les organes

Le sous-programme REPARTIR est appelé si la croissance aérienne est évaluée au moyen de l'indice foliaire (CODELAITR = 1).[158] Il effectue des évaluations après la levée et des réinitialisations après récolte, avec :

- un retour vers l'appelant avant la levée ;[159]
- un branchement GOTO 10 vers une séquence finale depuis deux blocs conditionnels imbriqués après récolte ;

Le mécanisme d'échappement utilisé pour ignorer les branchements (3.7.3) est activé : il suffit alors d'inclure la séquence d'évaluation dans un bloc conditionnel contrôlé par la variable générée $cible_ \neq 10$.

Rang de la campagne La variable RESPERENNE est initialisée si la culture est installée et pour la première campagne (NUMCULT = 1) avec

- RESPERENNE0, paramètre initial, pour la valeur 2 de l'option CODEPERENNE, paramètre de la plante,
- la valeur nulle sinon.

Cette instruction conditionnelle serait à déplacer dans la séquence d'appel pour ne pas dépendre du rang.

4.5.9 Croissance racinaire

Le sous-programme CROIRA évalue la croissance racinaire mais le contrôle d'exécution s'avère complexe :

- le retour vers l'appelant peut s'effectuer six fois en cours d'exécution dans des blocs conditionnels ;[160]
- une occurrence se solde par un message dans le fichier historique suivi d'un STOP qui clôt l'exécution ;

Dans ce contexte sont appelées plusieurs unités de programmes.

Les fonctions HUMIRAC (4.5.3) et ESCALIN,[161] qui n'acèdent pas aux variables communes, rendent des valeurs.

Le sous-programme DENSIRAC est appelé dans la phase terminale, pour la valeur non unitaire de l'option CODERACINE lue avec les paramètres de la plante.

[158] L'appel est effectué en fin de boucle quotidienne avant la version 6, après l'évaluation de la croissance racinaire ensuite.

[159] Le test est effectué après celui de la date de récolte.

[160] Ce mode de contrôle plus fréquemment utilisé dans la version 6 a conduit à renoncer à une traduction sous forme de blocs conditionnels jusqu'à la fin de la fonction, comme effectué dans la version 5.

[161] La fonction est une suite de séquences conditionnelles terminées par un RETURN, à remplacer par un choix multiple.

CALPSIBASE est appelé juste avant le retour final : il évalue la seule variable commune PSIBASE.[162]

Les modules objets sont regroupés dans la macrodéfinition $F5$ du $makefile$, traduction de la fonction f_5, antérieurement évaluée après les apports et interventions f_4.

Profondeur maximale du sol Les simulations s'effectuent dans des couches élémentaires de un centimètre jusqu'à une profondeur d'au plus dix mètres, la valeur 1000 étant citée en clair (pour contrôler les structures itératives et conditionnelles : cette constante, signalée dans le fichier "`stics.dec`" (3.9.1), serait à gérer dans une variable pour pouvoir choisir les profondeurs pour les applications logicielles, afin de se prémunir contre les débordements,[163] et éviter des instructions de forçage dans des intervalles.[164]

Profondeur effective du sol La profondeur utile du sol est gérée dans la variable commune NHE, initialement lue dans INITNONSOL puis réévaluée, ici et ailleurs (SORTIE,TASSEMENT), dans une boucle sur les horizons.[165]

Séquence itérative jusqu'à la profondeur des racines Un profil de densité racinaire est évaluée dans une boucle à pas unitaire depuis 1 jusqu'à 1000, avec une condition de sortie si la variable de boucle dépasse la longueur des racines. Le mécanisme d'échappement utilisé pour ignorer les branchements est activé et la structure itérative reprise.

Pseudo-référence aux périodes antérieures Les références à la période NGER $-$ 1 s'effectuent dans un bloc qui s'exécute dans une boucle si NGER $=$ N : c'est donc une référence classique à la valeur de la veille.[166]

Densité racinaire

La densité racinaire est évaluée à partir de la date de germination, un retour appelant étant effectué pour les périodes antérieures.[167] Un déroutement est possible en cours d'exécution en l'absence de "racines efficaces", avec un ultime message dans le fichier historique avant arrêt de l'application.

[162] L'évaluation s'effectue par cumul sur les couches du sol jusqu'à la profondeur racinaire avec une remise à zéro si la valeur PSISOL(IZ) de la couche IZ dépasse le seuil défini par le paramètre "nouvelle version" PSIHUMIN. Si ces valeurs demeurent stationnaires en profondeur, les problèmes de précision peuvent être source d'écarts conséquents, le cumul se poursuivant tant que l'expression logique associée demeure constante (23 février 2009).

[163] Dans la version 5, FLRAC et LRACZ, définies à la profondeur 200, sont référencées dans une boucle depuis 1 jusqu'à 1000.

[164] INTZRACBAS et INTZRACHAUT sont ramenées dans l'intervalle $[1 \ldots 1000]$.

[165] Une boucle interne effectue un traitement sur les couches de l'horizon i : l'épaisseur EPC(I), paramètre du sol, est cumulée à la sortie de cette boucle.

[166] Il suffit de substituer dans le code Fortran N à NGER pour faire disparaître ces fausses références.

[167] La première instruction teste en fait le jour de récolte .

Séquence itérative de cumul de degrés-jours La recherche de la date de production des racines sénescentes s'effectue dans une boucle à pas unitaire depuis 1 jusqu'à N avec cumul de DTJ(I-N), et une condition de sortie dès que la valeur du paramètre de la plante DEBSENRAC est atteinte : la date NSENCOURAC est alors N-I. Le mécanisme d'échappement utilisé pour ignorer les branchements est activé et la structure itérative reprise.

Référence aux périodes antérieures La séquence précédente requiert la consultation des périodes antérieures jusqu'à la date NSENCOURAC, laquelle a été copiée dans les évaluations de la veille dans la variable commune NSENCOURPRERAC : la matrice DRL est alors mise à jour pour les couches explorées par les racines sur la période [NSENCOURPRERAC − NSENCOURAC].

Discontinuité de l'évolution du front d'enracinement L'évaluation de la densité racinaire introduit le facteur multiplicatif entier MULTLVFRONT appliqué au paramètre de la plante LVFRONT, variable discrète qui prend en compte les variations relatives de la longueur racinaire totale, ZRAC, et de la vitesse de croissance racinaire, DELTAZ. Les valeurs peuvent varier selon que les calculs sont effectués en simple ou double précision (4.1.2), ce qui peut se traduire par des écarts infinitésimaux en fin de simulation pour des compilations différentes.[168]

4.5.10 Tassement du sol

Initialement traités au début de la boucle quotidienne (4.5.2), le tassement du sol est désormais pris en compte avec les interventions techniques, et finalement intégré à la fonction f_4 : il modifie la structure du sol, et du système sol-plante, et peut conduire à différer les dates d'interventions culturales évaluées dans f_1.

Sortie de boucles imbriquées Les sous-programmes utilisés gèrent une structure de contrôle particulière, à reprendre. Il s'agit de boucles imbriquées avec sortie depuis la boucle interne par un branchement hors des deux boucles si une couche du sol est atteinte, la borne inférieure de la boucle externe étant mise à jour sinon.

Structure du sol et système sols-plante

Les sous-programmes DETASSEMENT et TASSESEMISRECOLTE, TASSEMENT antérieurement (4.5.2), modifient, sous l'inflence des techniques culturales aux dates lues avec les paramètres techniques :

- des variables descriptives de l'état du sol lues dans INITNONSOL (4.4.2) : HUR,EPC,PROFHUM,INFIL,DA,NHE,NIT,EPD ;
  INFIL est pondérée par la fonction EFFETTASSSURINFIL(DA).

[168] Soit $Int(x)$ la fonction qui rend le plus petit entier $\leq x$ et $k = Int(\text{DELTAZ})$. MULTLVFRONT vaut k si
$((Int(\text{ZRAC}) - Int(\text{ZRAC - DELTAZ})) \leq k$, $k+1$ sinon. Le facteur multiplicatif 10^4 qui gère les unités amplifie la dérive.

– le paramètre général z0SOLNU ;

TASSESEMISRECOLTE contient une boucle *Pour* avec une ultime instruction de sortie, à reprendre.

Ces sous-programmes utilisent la fonction CALMACRO, définie pour évaluer la macroporosité et simuler les transferts d'eau dans le sol (4.5.16).

La recherche des sommets extrémités dans le graphe associé met en évidence des corrections effectuées des valeurs simulées de variables descriptives du système racinaire, dont la densité dans les couches du sol RL.

Dates de semis et récolte

Les dates de semis et récolte NPLT et NREC, initialisées dans la boucle quotidienne à partir des dates calendaires IPLT et IREC lues avec les paramètres techniques, sont réévaluées pour la valeur 1 des options CODEDECISEMIS et CODEDECIRECOLTE, paramètres "nouvelle version", par appels des sous-programmes DECISIONSEMIS et DECISIONRECOLTE, depuis le programme principal dans les dernières versions.

Désynchronisation La décision de semis prend en compte les températures des NBJSEUILTEMPREF jours qui suivent la date courante, paramètre "nouvelle version", pour différer le semis en cas de gel.

4.5.11 Apports d'eau

La disponibilité quotidienne de l'eau pour la culture à partir des précipitations ou irrigations et après interception par le couvert végétal est simulée dans le sous-programme IRRIG. Les valeurs sont gérées dans des variables simples, sauf l'eau d'irrigation avec la série AIRG, définie à compter du premier jour de la simulation.

Dans la version 6, IRRIG est appelé depuis CALCPRECIP (4.5.1), lequel traite des relations entre les couverts végétaux pour les deux cultures et met à jour quelques variables. La dernière version introduit en outre un traitement particulier des pluies à la levee pour une culture pure de quinoa en poquet avec appel du sous-programme AFFECTRRQUINOAPOQUET. Afin de conserver l'ordre des appels et initialisations, c'est finalement la fonction *Calcprecip*() qui est appelée depuis *Simuler*().

Les modules objets sont regroupés dans la macrodéfinition $F4$, traduction de la fonction f_4.

Eau d'irrigation

Les apports du jour sont :

1. évalués avant simulation dans le sous-programme d'initialisation INICLIM avec affectations des doses d'eau DOSEI apportées à des dates fixes JULAPI, lues avec les paramètres techniques ;

2. mis à jour en cours de simulation dans IRRIG pour la valeur 1 de l'option CODECALIRRIG, avec évaluation des doses requises sur la base d'une quantité minimale disponible pour la germination puis d'un seuil de stress à ne pas franchir ;[169]

Résidus de culture

Les quantités d'eau contenues dans les résidus organiques incorporés dans le sol au cours d'opérations techniques, ou déposés en surface pour les feuilles, sont traités dans le sous-programme APORES comme des opérations quotidiennes d'irrigation pour l'équivalent de un millimètre d'eau et plus :

1. la variable commune NAP, paramètre technique qui dénombre ces opérations, est incrémentée d'une unité ;
2. l'apport d'eau est ajouté dans la série AIRG, non pas à la date de l'apport, mais le lendemain ;

4.5.12 Apports d'azote et de matières organiques

La fertilisation est globalement simulée par le sous-programme FERTIL, lequel traite :

- des apports d'azote par la fertilisation minérale et organique, la pluie et l'irrigation ;
- des effets de l'incorporation dans le sol à des dates données de résidus organiques ;

FERTIL appelle PERTENG qui simule le devenir des engrais minéraux dans le sol, en tenant compte des pertes par volatilisation, dénitrification et lessivage.

De nouvelles séquences gérées dans APPORTS sont ajoutées dans la version 6, avec la réinitialisation des cinq premiers jours et les appels de CALAPNENUPVT et RESIDUS (4.5.1) : elles sont intégrées directement dans la fonction *Simuler*().

Les modules objets sont ajoutés dans la macrodéfinition $F4$, traduction de la fonction f_4.

Apports d'azote

Les apports d'azote du jour sont gérés dans la série quotidienne ANIT :

1. les doses DOSEN, associées à des rangs repérés par des dates JULAPN, lues avec les paramètres techniques, sont intégrées avant simulation dans le sous-programme d'initialisation INICLIM ;
2. les apports consécutifs aux coupes des cultures fourragères, doses ANITCOUPE apportées à des dates fixes JULFAUCHE, sont ajoutées dans le sous-programme INICOUPE, appelé depuis la boucle quotidienne pour réinitialiser le redémarrage des prairies (4.5.18).

[169] Le nombre d'opérations NAP, lu avec les paramètres techniques, est mis à jour.

3. d'autres apports sont effectués en cours de simulation dans FERTIL pour la valeur 1 de l'option CODECALFERTI, les quantités étant évaluéés à partir du paramètre technique DOSIMXN : le paramètre technique ENGRAIS est alors forcé à la valeur 8 et le nombre d'opérations NAPN est incrémenté.

Fertilisation contrôlée L'option CODEDATEAPPN introduite dans la version 6 gère d'autres doses DOSEN, associées à des rangs pour lesquels sont fournies des seuils UPVTTAPN, lus avec les paramètres techniques à la place des dates JULAPN. Ces apports sont traités par CALAPNENUPVT avant l'appel de FERTIL : ils sont effectués le jour du semis, puis si une somme de températures dépasse la valeur du paramètre technique UPVTTAPN > 0 pour le rang courant :

- les valeurs sont ajoutées au terme du jour dans la série ANIT ;[170]
- la date courante est affectée dans JULAPN pour le rang de l'opération ;
- le nombre d'opérations NAPN incrémenté ;

Apports de matières organiques

Le sous-programme APORES traite de la décomposition des matières organiques pour huit types prédéfinis de résidus, décrits par des paramètres généraux (4.3.3). Il est d'abord appelé comme instruction finale dans une boucle finale depuis FERTIL pour les interventions techniques successives afin de traiter l'incorporation des résidus aux dates données.

APORES agit alors par effet de bord en modifiant des variables communes.

1. Le paramètre formel CNRESIDU, défini par référence, est mis à jour en fonction des valeurs des paramètres généraux associés CNRESMIN et CNRESMAX pour le type de résidu de rang IRES, variable commune qui reçoit sa valeur chez l'appelant.
2. Une opération d'irrigation supplémentaire est dénombrée pour une quantité d'eau supérieure à un 1 millimètre, valeur transmise dans le paramètre formel EAURESIDU : la série AIRG qui gère l'eau d'irrigation est mise à jour et le nombre d'apports NAP alors incrémenté.

L'appel depuis FERTIL peut ainsi, pour le type de résidu IRES :

- modifier le paramètre technique CSURNRES ;
- générer une opération d'irrigation pour un apport d'eau supérieure au millimètre, valeur gérée dans le paramètre technique EAURES ;

Décomposition et minéralisation des résidus de récolte

Appelé quotidiennement, le sous-programme RESIDUS effectue ses évaluations dans un bloc conditionnel le jour de la récolte, à l'exception d'une instruction dde cumul dans QMINRCULT qui se prolonge après récolte.

[170] Il serait nécessaire de distinguer les apports externes liés à l'itinéraire technique, exogènes, des quantités finalement disponibles pour la plante, liées à d'autres apports déclenchés par des processus externes ou associés à la dynamique de la culture.

L'appel final de APORES peut mettre à jour CSURNRESSUITE pour les types de résidus de culture principale (IRES = 1) ou culture intermédiaire (IRES = 2), mais il ne génère pas d'opération d'irrigation vu que c'est la valeur nulle qui est transmise comme paramètre effectif associé à EAURESIDU, et non le paramètre technique EAURES défini pour ces deux types de résidus.

4.5.13 Etat de surface du sol

Le sous-programme ETATSURF, appelé après APPORTS, évalue d'abord la quantité de pluie restante PRECIP si un seuil de ruissellement fixé par le paramètre général PMINRES est dépassé.[171]

Il traite ensuite de l'état de surface consécutif à un paillage végétal en l'absence de travail du sol au-delà de la couche superficielle : des variables descriptives du mulch sont évaluées,[172] et les effets sur le ruissellement éventuel sont pris en compte, le contrôle étant rendu à l'appelant sinon.

4.5.14 Bilan hydrique

Les besoins en eau de la culture sont évalués selon deux méthodes par appels de KETP ou SHUTWALL dans l'unique instruction conditionnelle de BEAUPL en fonction de l'option CODEBESO, paramètre de la plante.

KETP ou SHUTWALL appellent le sous-programme d'évaluation de l'évaporation du sol SOLNU, lequel requiert les fonctions FPER et FPER,[173] et utilisent la fonction de calcul de la tension de vapeur saturante TVAR (4.4.3).

SHUTWALL appelle :

1. CALRNET, pour l'évaluation du rayonnement net, appelle
   - RGEX pour le calcul du rayonnement extraterrestre, qui requiert DECANGLE (4.4.3) ;
   - RATMO, sous-programme de calcul du rayonnement atmosphérique, qui utilise RGEX ;
   - TVAR pour la tension de vapeur saturante ;
2. CALTRAERO, sous-programme d'évaluation de la résistance aérodynamique du couvert ;
3. CALTCULT,[174] pour le calcul des températures des cultures, appelle :
   - CALRNET, racine de l' arbre précédent ;
   - RATMO, branche de l'arbre précédent ;
   - GSOL, fonction d'évaluation de l'effet du vent sous le couvert ;

[171] Cette modification n'est pas prise en compte dans l'évaluation de l'eau disponible pour la plante : la quantité de pluie est en effet transmise à IRRIG depuis CALCPRECIP via le paramètre PRECIPTMP et c'est ensuite que PRECIP est mis à jour dans ETATSURF.

[172] Le paramètre technique lu COUVERMULCH est réévalué.

[173] $Fper(x,a) = \sqrt{a^2 + 2ax} - a$ et $Finv(x,a) = ((x+a)^2 - a^2)/2a$: le paramètre des fonctions a, géré dans la variable AEVAP initialisée dans INITSOL2, dépend de l'humidité du sol et du paramètre général ACLIM.

[174] Les algorithmes numériques utilisés sont sources de dérives quant à la précision (4.1.2).

– PHOTPD, sous-programme d'évaluation des durée et photopériode du jour ;

La transpiration est évaluée par le sous-programme TRANSPI après simulation des transferts d'eau.
Les stress hydriques sont évalués par appel de STRESSEAU et EXCESDEAU évalue un indice d'excès d'eau.

Les nouveaux modules objets définissent la macroinstruction $F6$, traduction de la fonction f_6.

Evaporation du sol Le sous-programme SOLNU initialise la variable EOS en fonction de la valeur de l'option CODEBESO puis exécute un bloc conditionnel d'une centaine d'instructions.[175] Cette séquence simule les phases d'évaporation du sol à travers un imbroglio de branchements qui fait que, les formes syntaxiques n'étant pas reconnues, un autre mode de traduction est adopté avec édition de l'étiquette de branchement dans une variable d'état (3.7.3).

La restructuration à vue s'avérant complexe, une copie locale des variables communes utilisées dans cette séquence d'une soixantaine d'instructions est demandée lors de l'exécution de la commande $S6$ (2.7) : le code C validé peut ainsi être repris tel quel dans les versions successives dès lors que le code Fortran ne change pas, les valeurs étant restituées en fin de sous-programme.[176]

SOLNU effectue ensuite des mises à jour pour les couches du sol.[177]

Méthode K x ETP

Le sous-programme KETP appelle immédiatement SOLNU. Les besoins en eau sont ensuite simulés, sauf pour la valeur 1 du paramètre "nouvelle version" CODEPITAF, option qui entraîne l'exécution d'une séquence de forçage qui se termine par un retour chez l'appelant.

La méthode traite le couvert végétal pour la seule culture principale.

Méthode résistive

Le sous-programme SHUTWALL prend en compte les cultures principale et associée, selon le mode classique mais aussi avec la variable logique locale DEUXCULT : plusieurs conditions logiques font référence à la culture associée, qu'elle soit présente ou non.[178]

[175] CODESOLNU est forcée à 1 : le bloc *sinon* lié à la condition logique (CODESOLNU = 1) est exécuté, le bloc *alors* étant vide.

[176] Il est vérifié que SOLNU n'appelle pas d'unités de programmes qui accèdent aux variables communes.

[177] Une changement d'échelle est géré par la conversion entière ESOL = AINT(ESOL*1000)/1000 : les parenthèses sont réintroduites en C pour gérer les priorités des opérateurs de conversion et arithmétiques (3.3.6).
Les bornes supérieures, qui gèrent les profondeurs, sont variées et non contrôlées, dont la valeur 1000 (3.1.2).

[178] La condition Fortran est éditée en commentaire derrière une instruction vide dans le code C, pour contrôle.

Deux erreurs closent l'exécution après édition d'un message dans le fichier historique.

Boucle d'évaluation Les évaluations sont faites dans un bloc itératif avec un branchement inconditionnel arrière comme ultime instruction d'un bloc conditionnel inclus dans un autre bloc conditionnel : la forme est reconnue (1.3.2) et l'itération contrôlée par la conjonction des conditions logiques associées.

Les variables de contrôle sont des variables communes, la différence de température DIFFTCULT et un nombre d'itérations NITETCULT,[179] évaluées dans le sous-programme CALTCULT appelé depuis ce bloc itératif : l'itération se poursuit tant que

$$(\text{DIFFTCULT} > 0.5) \wedge (\text{NITETCULT} \leq 5),$$

ce prédicat étant aussi utilisé dans CALTCULT.

Température de la culture

CALTCULT évalue la température de la culture, sauf pour l'option technique CODABRI = 2. : TCULT, TCULTMIN et TCULTMAX reçoivent alors les valeurs TMOY, TMIN et TMAX et le contrôle rendu à l'appelant.[180]

Pseudo-boucle d'évaluation L'évaluation s'effectue dans une séquence avec un branchement arrière conditionnel comme ultime instruction d'un bloc conditionnel final. Le prédicat de contrôle résultant reprend le prédicat de la boucle d'évaluation précédente en conjonction avec la valeur 1 de l'option CODEBESO, soit

$$(\text{DIFFTCULT} > 0.5) \wedge (\text{CODEBESO} = 1) \wedge (\text{NITETCULT} \leq 5).$$

CALTCULT étant appelé depuis SHUTWALL, qui effectue les évaluations pour la valeur 2 de l'option CODEBESO, ce bloc s'exécute une seule fois.[181]

Résistance aérodynamique

Le sous-programme CALRAERO, appelé exclusivement depuis SHUTWALL, gère les variables dans une liste de paramètres et n'accède pas aux variables communes. Les paramètres formels RAA et RAS sont définis par référence : les variables communes RAAMAX,RASMAX,RAAMIN,RASMIN,RAS transmises comme paramètres effectifs sont ainsi mises à jour.

Deux paramètres formels sont homonymes de variables communes, le paramètre général ZR et la variable RAS, le premier étant supprimé dans la traduction C car toujours transmis comme paramètre effectif (2.2.5).[182]

[179] SOLNU, appelé dans cette boucle, utilise la condition logique NITETCULT = 0 pour une initialisation conditionnelle.

[180] DIFFTCULT et NITETCULT n'étant pas évalués, la boucle précédente est alors infinie.

[181] NITETCULT devrait être une variable de contrôle locale : l'initialisation depuis SOLNU serait alors à faire dans SHUTWALL.

[182] Le choix d'autres noms pour ces paramètres formels lèverait les ambiguïtés et les vecteurs $K_$ et $Y_$ ne seraient plus requis.

Transpiration des plantes

Le sous-programme TRANSPI traite de la transpiration de la plante, répartie dans la variable commune EPZ sur l'ensemble de la zone racinaire dans une boucle sur la profondeur du sol.

L'algorithme de redistribution sur l'ensemble de la zone racinaire en cas de sol sec est sensible à la précision en arithmétique binaire (note 10 section 4.1.2) : son application aux limites serait à examiner.[183]

Stress hydrique

Les indices de stress, évalués avant la version 6 dans TRANSPI, le sont désormais dans les sous-programmes STRESSEAU et EXCESDEAU.

4.5.15 Bilan azoté

Décrit dans un module de la notice *Concepts et formalismes*, le bilan azoté est effectué dans plusieurs sous-programmes qui ne sont pas appelés en séquence, selon un ordre qui varie entre les versions 5 et 6.

Le sous-programme MINERAL, appelé avant BEAUPL dans la version 5, effectue les évaluations dans une boucle pour les couches de la zone humifère jusqu'à la profondeur PROFHUM, paramètre du sol :[184] cette boucle contient une succession de boucles internes définies pour les huit types de résidus organiques (4.3.3).[185]

MINERAL appelle deux sous-programmes,[186] DENIT et VOLATORG, ce dernier appelant EQAMMO qui requiert les fonctions KABR et KHBR. La version 6 introduit le sous-programme MINPITAF appelé en sortie de la boucle principale pour la valeur 1 de l'option de forçage CODEPITAF.

Les évaluations quotidiennes se poursuivent avec l'appel conditionnel de OFFRNODU pour la prise en compte de la fixation symbiotique pour les légumineuses, effectué plus loin dans la version 5.

OFFREN, ABSON, MAJNSOL, STRESSNN et NGRAIN apparaissent dans la version 6 pour isoler ou compléter des évaluations faites antérieurement dans d'autres sous-programmes.

Les nouveaux modules objets définissent la macroinstruction $F7$, traduction de la fonction f_7.

Volatilisation d'ammoniac

VOLATORG, appelé si la variable NVOLATORG franchit le seuil 0.1, évalue heure par heure la volatilisation d'ammoniac issue des amendements organiques

[183] La condition logique de contrôle utilise la variable EP qui reçoit la valeur 10^{-10} (cf "`stics.dec`") si CUMLR est nul.

[184] La valeur est lue dans LECSOL avec une décimale et arrondie localement pour désigner l'ultime couche IHUM.

[185] Les évaluations sont faites pour un résidu si la quantité dépasse le seuil 10^{-10}.

[186] Dans la version 5, MINERAL se termine par un appel de APORES pour traiter de l'incorporation des résidus.

liquides.[187]

Equilibres entre les formes d'azote ammoniacal EQAMMO est appelé dans la boucle sur les heures du jour. Il utilise les fonctions KABR et KHBR pour évaluer des constantes d'équilibre pour la température absolue de la culture, puis exécute une boucle depuis 1 jusqu'à 1000, mais avec comme ultime instruction un branchement extérieur si une condition logique est vérifiée. Cette forme est reprise dans une itération avec un prédicat de contrôle unique.

Dénitrification

DENIT simule la dénitrification dans les couches superficielles jusqu'à la profondeur PROFDENIT, paramètre général : il est appelé pour la valeur 1 de l'option CODEDENIT, autre paramètre général.

Forçage des besoins en azote

MINPITAF exécute la même boucle que MINERAL les couches de la zone humifère et une boucle interne sur les huit types de résidus organiques.

Deux autres boucles consultent les périodes antérieures dans la variable commune HOFFRN.[188]

Fixation symbiotique

OFFRNODU est appelé depuis le programme principal pour la valeur 2 des paramètres de la plante CODELEGUME et CODESYMBIOSE, une première fois après BEAUPL puis à nouveau dans une séquence conditionnelle si (MASECDIL $>$ 0). Il utilise la fonction FXTFONC qui rend une valeur réelle.

Les évaluations sont effectuées après la levée et si les racines sont en place, le contrôle étant rendu à l'appelant pour les autres périodes, après remise à zéro de FIXREEL en l'absence de racines.[189]

Demande en azote de la plante

BNPL, qui reprend des séquences de l'ancien sous-programme ABSORBN, est ensuite appelé, le contrôle étant rendu à l'appelant si (MASECDIL ≤ 0).

---

[187] NVOLATORG est évalué dans FERTIL en fonction de la quantité d'eau du résidu, paramètre technique : la valeur dépend de la quantité $(0.37 + 0.029(100 - \text{EAURES}) + 0.117)$. Cette évaluation serait à déplacer ici, la variable n'étant pas utilisée ailleurs.

[188] Déclarée avec 720 éléments, cette table Fortran n'est pas reconnue comme série temporelle (3.2.3) : la déclaration est à corriger pour restituer cette série, et éviter de devoir sauvegarder de telles tables à l'issue des simulations quotidiennes.
Les corrections sont à faire pour les quatre autres tables à 720 éléments : EPIQN, EPIKC, EPIQNPLMAX, TEMPOFFN.

[189] Trois paramètres de la plante, STLEVDNO, STDNOFNO et STFNOFVINO, sont modifiés.

Absorption de l'azote

La version 6 introduit deux nouveaux sous-programmes, ABSON et MAJNSOL, appelés dans un bloc conditionnel si (MASECDIL > 0). Le controle est rendu à l'appelant depuis ABSON pour la valeur 1 de l'option de forçage CODEPITAF pour toute période antérieure à NLEVOBS ou postérieure à NLANOBS.

L'absorption de l'azote est simulée plus loin par appel du sous-programme STRESSN, qui reprend en partie l'ancien sous-programme ABSORBN : le contrôle est immédiatement rendu à l'appelant si (MASECDIL ≤ 0).

Azote des organes de récolte

Antérieurement évalué dans GRAIN ou FRUIT, la quantité d'azote des organes de récolte est désormais évaluée dans le sous-programme NGRAIN appelé après STRESSN.

4.5.16 Transferts d'eau et de nitrates dans le sol

Appelé depuis le programme principal, LIXIV simule les processus de transferts de l'eau dans le sol. Il utilise les sous-programmes TRANSF et NAPPE et la fonction CALMACRO, unités de programmes pour lesquelles, les variables communes n'étant pas accessibles du fait de l'absence d'inclusion des fichiers de déclarations, les variables consultées ou modifiées sont transmises dans des listes de paramètres.[190]

Classé dans le même module dans la notice *Concepts et formalismes*, le sous-programme d'évaluation des températures de la culture et du sol est

Les modules objets définissent la macroinstruction $F8$, traduction de la fonction f_8.

Simulation du transfert

Les transferts sont simulées depuis la surface dans NCEL cellules définies pour les horizons, dont les épaisseurs EPD sont des paramètres du sol.[191] La simulation s'effectue jusqu'à la base du dernier horizon, soit la somme des épaisseurs EPC d'au plus cinq horizons.[192]

Les horizons sont gérés dans des tables définies depuis 1 jusqu'à 5, avec souvent un élément supplémentaire de rang 0 introduit pour la prise en compte des phénomènes de surface.

[190] Ces formes sont source d'ambiguïtés, des variables communes apparaissant comme modifiées parce que citées, alors que la mise à jour est effectuée sur le paramètre formel homonyme, variable locale, et transmise via le paramètre effectif de l'appel, parfois homonyme de cette même variable commune. La traduction C supprime les paramètres formels et effectifs dont les noms sont invariants (2.2.5) : il suffit de renommer les autres paramètres dans le code Fortran pour lever ces ambiguïtés.

[191] Les valeurs inférieures au centimètre sont ramenées à 1 en cours d'initialisation dans INITSOL1, ou lors des évaluations quotidiennes dans TASSEMENT.

[192] Les tables locales demeurent définies jusqu'à 1000 centimètres dans la version 5 bien que la profondeur soit limitée à 200.
ETN demeure initialisée avec l'antique instruction DATA dans la version 6.

Séquence itérative de remplissage de la macroporosité de l'horizon profond sur sol drainé Les transferts dans les sols drainés, option 1 du paramètre du sol CODRAINAGE, sont simulés dans un bloc conditionnel avec une boucle sur les horizons qui s'exécute tant que :[193]

- il existe un excès d'eau dans l'avant-dernier horizon ;
- l'exces d'eau du dernier horizon ne remplit pas sa macroporosité ;

Transferts entre cellules

Les transfert de l'eau et de l'azote calcul du transfert de l'eau et de l'azote sont traités par appels du sous-programme TRANSF, avec des échanges de paramètres complexes.[194]

Gestion de couches de transition TRANSF exécute une boucle sur les couches du sol. Le paramètre formel IZC gère des rangs de couches de transition pour les 5 horizons, évalués dans la phase d'initialisation depuis INITSOL1, affectés dans les variables communes IZC et IZCEL ensuite transmises depuis LIXIV comme paramètres effectifs. La recherche de ces couches s'effectue dans une boucle interne avec une unique instruction de sortie conditionnelle si la couche courante est la couche de transition de l'horizon.[195]

Transfert vers la couche suivante Le transfert en fin d'itération sur les couches du sol est géré par un branchement conditionnel suivi d'une séquence qui se termine par un branchement inconditionnel, forme qui représente finalement une conditionnelle structurée.[196]

Macroporosité

La fonction Fortran CALMACRO est appelée depuis LIXIV mais aussi par NAPPE, et dans un autre contexte par TASSEMENT :[197] elle choisit la formule d'évaluation, affecte le résultat dans la variable locale réelle CALMACRO, et rend la valeur à l'appelant.[198]

---

[193] la structure de contrôle est ambigüe et complexe, avec un branchement inconditionnel arrière, unique instruction d'une boucle Fortran WHILE, ce qui traduit finalement un simple branchement conditionnel.

[194] Parmi les 18 paramètres formels, 16 sont homonymes de variables communes, et 6 sont aussi des paramètres effectifs homonymes.

[195] La variable de boucle ICOU varie depuis 1 jusqu'à 5 et est remise à zéro en sortie pour l'occurrence de la couche de transition. Elle est ensuite utilisée pour déterminer si la couche courante est une couche de transition (ICOU > 0).

[196] Ces branchements sont à supprimer et la structure est à reprendre avec un **if structuré** dans le code Fortran.

[197] Les 5 paramètres formels sont homonymes de variables communes, de dimensions différentes pour 4 d'entre eux d'où les avertissements générés : ils sont à renommer, les paramètres effectifs n'étant pas toujours homonymes.

[198] Les constantes citées en clair dans les formules sont est répertoriées dans "`stics.dec`".

Description de la nappe

Le sous-programme NAPPE évalue plusieurs variables descriptives de la nappe avec des modalités de restitution des valeurs complexes.[199]

Une boucle à pas constant contient comme première instruction un branchement conditionnel de sortie : la reprise de la condition logique permet de restituer une structure de contrôle itérative sans branchement.

4.5.17 Etat final quotidien de la culture

TEMPSOL évalue les températures des couches du sol et de la culture. Il est suivi de l'appel de HUMCOUV pour l'évaluation de l'humidité du couvert végétal, lequel requiert la fonction réelle TEMPROSEE. Ces deux sous-programmes utilisent la fonction d'évaluation de la tension de vapeur saturante TVAR.

Deux sous-programmes évaluent de nouvelles variables :

- CUMAOETAS effectue des cumuls pour les couples de cultures et des transferts de valeurs entre variables ;
- SORTIE calcule des sommes et des moyennes et édite éventuellement des données quotidiennes ;

Les modules objets sont regroupés dans la macroinstruction $F9$.

Températures de la culture et du sol

L'appel de TEMPSOL depuis le programme principal dans la version 6 est précédé d'une réévaluation de EPTCULT et de l'appel de CALTCULT, puis suivi de la mise à jour de TCULTVEILLE et TAIRVEILLE.[200]

Humidité moyenne

HUMCOUV évalue les deux variables communes HUMIDITE et HUMAIR.[201]

Microclimat horaire

Le sous-programme HUMHEURE évalue un microclimat horaire pour l'option CODEMICHEUR = 1, paramètre général : l'appel depuis le programme principal

[199] NAPPE est défini avec 23 paramètres formels, dont 19 homonymes de variables communes et des paramètres effectifs correspondants : HMAX, HNAPPE,HPB, HPH et PROFNAPPE sont ainsi mis à jour. Les 4 autres paramètres sont homonymes de variables locales de LIXIV et des parmètres effectifs correspondants : HNAPPERCH, COTEDRAIN et H sont définis par référence et DE par valeur.

[200] Ces instructions sont à inclure dans TEMPSOL pour ne pas effectuer les évaluations dans la boucle quotidienne, fonction de supervision qui distribuer les tâches (CALTCULT est appelé depuis TEMPSOL dans les versions antérieures).

[201] Défini avec 11 paramètres, 7 peuvent être supprimés si les variables communes sont incluses (2.2.5) : TMOY est à renommer, le paramètre effectif correspondant étant non pas la table, mais la valeur du jour TMOY(N), d'où une erreur de prototypage.
Les constantes citées en clair dans le code sont répertoriées dans "`stics.dec`" : $8.64^4, 2500840, 2358.6, \ldots$

est précédé de l'évaluation de la variable commune NUMDATE, date transmise en paramètre à PHOTPD.[202]

Séquence itérative contrôlée HUMHEURE exécute une boucle depuis 1 jusqu'à 24, incluse dans une structure itérative pilotée par un branchement arrière tant que ($Humimoy - Humidite > 0.05$), avec un nombre maximal de 10 itérations.

Desynchronisation Une séquence d'évaluation des températures horaires fait référence à la température minimale du lendemain.

Evaluations quotidiennes finales et éditions

Les nombres d'apports NAP d'eau et d'azote NAPN de la fonction f_4 sont cumulés dans le programme principal pour les deux cultures dans NAPTOT et NAPNTOT.[203]

Le sous-programme CUMAOETAS[204] traite des couples de cultures mais effectue aussi des mises à jour de variables pour les cultures pures : ces deux rôles sont à séparer dans deux unités de programmes.[205]

Le sous-programme SORTIE

1. effectue d'ultimes évaluations,
2. édite des données dans trois fichiers.

Ces deux actions sont séparées dans la traduction C.

Evaluations Toutes les évaluations sont conservées dans *Sortie*().[206]

Les valeurs du jour de la récolte sont affectées les jours suivants pour plusieurs séries : MAGRAINC, MASECC, QNPLANTEC, CNGRAINC.

L'analyse syntaxique signale un bloc conditionnel vide si (CODEPLANTE = '`fou`').

Edition de données *BilSortie*() reprend toutes les opérations d'édition dans les fichiers associés aux unités logiques Fortran FICSORT, FICSORT2 et 55 (3.8.7).

- Une première séquence édite, si CODESIG = 3, la date calendaire évaluée plus haut, et deux variables.

[202] NUMDATE est aussi évaluée dans DEVELOP et CALTCULT.

[203] L'élément de rang 2 est explicitement référencé, même en l'absence de culture associée.

[204] Des références à la zone ombragée en l'absence d'un couple de cultures sont signalées pour les variables LAI et MASEC.
Une séquence de remise à zéro pour la culture associée est gérée au moyen d'un branchement hors d'un bloc conditionnel.
Un bloc conditionnel vide est associé à la valeur '`vig`' du paramètre CODEPLANTE.

[205] Les cumuls sont représentés par des arcs réflexifs dans le graphe des relations entre variables.

[206] La fonction TEMPROSEE évalue les variables locales TROSEECULT et TROSEEAIR, inutilisées.

– L'essentiel des éditions est effectuée en l'absence de retour chez l'appelant si (CODESIG ≤ 0). La section de code contient d'autres évaluations, dont une séquence finale introduite par le commentaire "`nouvelles variables pour expertise secheresse`", non reprise ici. En revanche, la séquence d'évaluation de la variable commune COMBITHOMAS est dupliquée car contenant des valeurs éditées.[207]

4.5.18 Gestion des coupes des cultures fourragères

Le sous-programme INICOUPE, appelé à la fin de la boucle quotidienne pour la valeur 1 du paramètre technique CODEFAUCHE, effectue en fait trois traitements consécutifs :

1. gestion de la décision de coupe au jour courant ;
2. réinitialisation à l'issue d'une coupe ;
3. edition d'un bilan et d'un état les jours de récoltes ;

Gestion des coupes

Les dates de fauches sont lues avec les paramètres techniques, ou simulées.

La séquence initiale de INICOUPE positionne la variable logique commune SIONCOUPE pour déterminer si la coupe s'effectue le jour courant, l'affectation étant suivi d'un branchement inconditionnel. Ce mode de contrôle, qui génère des branchemets complexes avec sorties simultanées de blocs conditionnels, est à reprendre.

Evaluation de l'état après coupe

A l'issue d'une récolte, des variables sont réinitialisées dans INICOUPE avec :

1. des valeurs lues de paramètres de la plante : STLEVAMF, STAMFLAX, STLAXSEN, STSENLAN, STLEVDRP ;
2. des valeurs initiales lues avec les paramètres techniques pour les coupes successives, repérées par leurs rangs : MSCOUPEMINI, LAIRESIDUEL, MSRESIDUEL, TEMPFAUCHE ;

Edition de bilans

Le sous-programme INICOUPE ne contient pas les ordres d'édition mais appelle des sous-programmes spécialisés, lesquels effectuent des évaluations de variables communes.

[207] Il s'agit de pouvoir valider une version C qui n'éditerait pas ces bilans, ce qui nécessite de vérifier l'équivalence de toutes les évaluations quotidiennes des variables (4.6). La double évaluation est ici possible en l'absence de boucle dans le graphe des relations entre variables, et du fait que COMBITHOMAS n'est pas utilisée ailleurs.

1. BILCOUPE, appelé exclusivement par INICOUPE, met à jour 15 variables.[208] Une erreur de syntaxe dans une déclaration Fortran de variable locale génère deux avertissements : les variables des listes sont néanmoins déclarées implicitement (note 16 section 2.1.3). Les appels s'effectuent :
   - comme première instruction le premier jour pour exécuter une séquence d'initialisation ;
   - comme ultime instruction si la date NRECBUTOIR est atteinte pour exécuter une séquence finale ;
   - à l'issue d'une coupe pour exécuter la séquence alternative de la séquence précédente ;
2. RECUP, appelé pour les coupes fourragères,[209] et en sortie de boucle quotidienne, évalue 5 variables de réinitialisation.[210] Il édite les bilans successifs dans le fichier de nom réservé `"recup.tmp"`.[211]

4.6 Validation des états quotidiens

La boucle quotidienne étant construite, la validation pour la simulation quotidienne de rang n s'effectue comme pour l'état initial (4.4.5).

1. La version Fortran est exécutée dans son répertoire *Fmake/appli*, avec la réponse n à la question `suivi jour ?`
   Les vecteurs sont édités dans les fichiers "`X_`".
2. *Stics*0 est appelé avec la valeur n pour l'argument $< jour >$.
   Pour chaque vecteur sont effectués :
   - une édition dans un fichier de la forme "`appli_X_`" ;
   - une comparaison composante à composante avec le vecteur Fortran lu, les écarts étant édités dans le fichier "`appli_trace`" ;

Si les évaluations de variables dynamiques divergent, il est précisé où sont mises à jour ces variables, vectorisées dans $Y_[]$: les algorithmes peuvent ensuite être déroulés en parallèle en positionnant des points d'arrêt respectivement dans les fonctions C et sous-programme Fortran, le rang de l'instruction Fortran figurant en commentaire du code C, jusqu'à localisation de l'écart.

4.7 Restitution finale des résultats

Des bilans sont actuellement édités dans des fichiers avec des noms prédéfinis et des suffixes réservés, depuis des sous-programmes appelés en cours de simulation comme décrit précédemment, et après simulation. Les sous-programmes ont été traduits tels quels pour visualiser les résultats.

[208] CNPLANTEC,IAMFS,IDRPS, ILANS,ILAXS,ILEVS,IMATS,RECS,ISENS,MAGRAINC,MASECTOT,QH2OI,QNH4I,QNO3I,QNPLANTECTOT.
[209] L'appel était effectué depuis BILCOUPE dans les versions antérieures.
[210] LAI0,MASEC0,QNPLANTE0,RESPERENNE0,ZRAC0.
[211] Ce fichier est ouvert dans un mode ambigu, UNKNOWN, en réécriture ou mise à jour selon le compilateur Fortran. Ces appels de RECUP paraissent par ailleurs inutiles puisqu'un appel inconditionnel est effectué en fin de simulation.

Les éditions demandées dans la boucle quotidienne s'effectuent dans le fichier historique, les éditions finales s'effectuant dans les fichiers associés (4.2.3).

Dates calendaires La mise en forme des dates est assurée par le sous-programme JULIEN : le code généré n'est pas repris et la fonction *Julien*() qui restitue les dates grégoriennes est fabriquée dans le fichier source "`Date.c`".[212]

Bilans périodiques

Les vecteurs du métamodèle pouvant être restitués pour toute période (4.2.5), les longues séquences d'affectation dans des tampons d'édition des valeurs à des dates données de variables énumérées dans des listes lues (3.8.7) puis sélectionnées par un choix multiple (1.3.1), ne sont pas reprises.[213]

Bilans globaux

Les autres éditions sont des bilans après récolte ou en fin de simulation, sous la forme de rapports mis en forme au moyen de formats :

- à l'issue des coupes fourragères avec BILCOUPE ;
- en sortie de boucle quotidienne avec BILAN ;

Les fonctions C générées pour ces deux sous-programmes sont reprises pour pouvoir disposer de vues d'ensemble comparables des résultats, avec toutefois quelques décalages liés à la conversion des formats.[214]

Etat final d'une campagne culturale

L'enchaînement des cultures, annuelles ou pérennes, s'effectue à partir de quelques variables descriptives d'un état final en surface et pour la zone humifère, évaluées et éditées dans un fichier par RECUP, puis relues pour la culture suivante (4.3.9).

[212] L'appel est conservé pour identifier les fonctions de restitution de bilans de campagnes.

[213] Les sous-programmes concernés peuvent a posteriori être exclus de la liste "`stics.liste`" (1.1.7) : LECRAP,RAPPORT,LECPROFI,PROFI,LECTUREVARIABLESRAPPORT.

[214] Quelques différences peuvent apparaître pour des expressions complexes de listes d'édition, non complètement reconnues et partiellement traduites.

Annexe A

Fabrication des applications

Le résultat informatique est une chaîne de traitement, désignée sous le nom *Cstics*, qui traite le code source Fortran77 des versions logicielles actuelles et rend une bibliothèque de fonctions en langage C.

La machine informatique est un ensemble de quelque quinze mille lignes de code C et deux cent cinquante pages de documentation, avec un second niveau de lecture technique dans quelque cinq cents notes de bas de pages qui répertorient les problèmes informatiques et renvoient aux contextes dans le code source. *Cstics* produit aussi une description détaillée des formes utilisées en Fortran dans différents fichiers, et un nombre important de messages dans des comptes-rendus pour localiser les problèmes et valider les solutions.

Les développements sont effectués dans le monde du logiciel libre, avec les outils GNU sous LINUX.

Des applications C et Fortran sont produites : elles sont utilisées pour vérifier l'équivalence de toutes les évaluations quotidiennes en cours de simulation des campagnes culturales, et valider les fonctions C sur un ensemble de cultures.

Deux applications Fortran sont fabriquées.

1. *fstics* contient les corrections minimales du code source pour produire une version exécutable sous LINUX.
2. *gstics* restructure le code pour automatiser la fabrication des fonctions C.

Des applications C génériques fournissent un environnement de programmation structuré qui intègre les formalismes du modèle de culture.

1. *Stics0*, permet d'initialiser les cultures sur un territoire pour construire des ensembles de vecteurs à partir des fichiers de paramètres actuels, décryptés au moyen de primitives de lecture qui interprètent les formats des enregistrements Fortran.
2. *Vtics* reconnaît les états initiaux par lecture des vecteurs et pilote les simulations.

L'accès aux formalismes du modèle s'effectue pour *Stics0* et *Vtics* au moyen de fonctions générées à partir du code source Fortran, à travers un diction-

naire des variables communes qui contient toutes les informations relatives à la vectorisation

Les fonctions C sont utilisables pour accéder aux évaluations quotidiennes :

1. par appels avec les vecteurs précédents en paramètres dans un environnement de programmation classique, depuis un langage ou un logiciel ;
2. par application de méthodes de classes en programmation par objets ;

D'autres fonctions, où méthodes de classes, peuvent être développées pour d'autres évaluations, ou pour effectuer des couplages : les décisions ou évènements externes quotidiens peuvent être traités de cette façon.

Cet environnement de développement permet aussi de traiter les dimensions temporelles et spatiales des cultures sur un territoire.

1. Le mode de traitement actuel des enchaînements de culture est repris, non pas dans le fichier dit '`recup`', mais dans un vecteur, acquis pour la simulation courante et transmis pour la simulation suivante
2. Les simulations sont à effectuer pour l'ensemble des parcelles, par itération de façon classique, ou par exécution en parallèle de processus qui peuvent aussi être synchronisés.

La dimension spatiale pouvant s'avérer coûteuse pour un grand nombre de processus, des directives permettent d'ajuster les tailles des vecteurs aux contextes applicatifs dans l'étape de génération du code cible.

A.1 Environnement de développement

Ce projet informatique a été conçu pour pouvoir être maintenu et repris dans différents contextes par des équipes de développeurs, pour produire des applications dans des langages ou depuis des logiciels, mais aussi pour restructurer et modulariser l'actuel code Fortran.

A.1.1 Hiérarchie des répertoires

Les traitements sont organisés dans un ensemble de répertoires dans lesquels il convient de se repérer pour maîtriser l'algorithmique mise en œuvre et utiliser la documentation.

Le répertoire père, racine d'une arborescence, contient :

1. une suite de répertoires, énumérés ci-dessous ;
2. un ensemble de fichiers utilisés pour produire la bibliothèque de fonctions à partir d'une version Fortran, décrits dans les chapitres 1, 2 et 3 pour les différentes étapes de la compilation ;

Les répertoires fils sont les suivants.

Code de la chaîne de compilation du code source Fortran

- *S* contient les commandes associées aux étapes de la compilation, des scripts d'exécution et de recopie, et quelques utilitaires.

Code source Fortran

Trois répertoires sont utilisés.

- *F77* contient le code source immédiatement après désarchivage.
- *Fmake* contient la version révisée des fichiers sources à l'issue des corrections minimales effectuées pour automatiser la chaîne de traitement. D'autres répertoires peuvent être fournis pour d'autres versions.
- *Fsource* reçoit le code normalisé produit au début des traitements, avec un fichier par unité de programme.

Les simulations sont effectuées dans des répertoires fils de *Fmake* ou *F77* pour les cultures.

Code cible

La chaîne de traitement produit du code C et Fortran.

- *C* contient le code des fonctions C générées à partir des unités de programmes, un module Fortran pouvant regrouper plusieurs unités.
- *D* contient d'autres fonctions ou fichiers d'inclusion ensuite utilisés pour construire les applications.
- *F* reçoit le code des sous-programmes Fortran de mise au point qui éditent pour chaque vecteur les variables qui le composent.

Code des applications C

Le code des applications est transféré au moyen de scripts appelés dans le répertoire père, les répertoires utilisés étant fournis en arguments. Pour ces applications :

- *Super* contient le code des fonctions développées pour remplacer des fonctions ou instructions Fortran, et effectuer la supervision, ainsi que le makefile ;
- *G, H...* reçoivent les applications,
  - *copieC* appelé avec l'argument *G* recopie le code C généré ;
  - *copieS*, appelé avec les arguments *Super* et *G*, recopie le code de supervision ;

  Les simulations sont effectuées dans des répertoires fils pour les cultures.
- Le code des sous-programmes de suivi est recopié au moyen de *copieF* dans *Fmake*.

Documentation

- *Doc* contient les fichiers LaTeXde cette documentation.

Code et méthodes natives JAVA

- *Java* contient le code et les classes JAVA ;
- *JavaC* correspond au *Super* des applications C, avec les fonctions développées pour les méthodes natives, et les fonctions communes ;
- *Native* contient tous les fichiers sources utilisés pour construire les bibliothèques partagées associées aux méthodes natives, ces bibliothèques, ainsi que le makefile qui les fabrique ;
- *Lib* regroupe les bibliothèques ;

A.1.2 Génération des fonctions C à partir du code Fortran

La chaîne de traitement du code source reconnaît les formes existantes, et applique des solutions pour résoudre les problèmes informatiques liés au langage Fortran et/ou au mode de programmation.

Révision minimale du code Fortran

Un certain nombre de problèmes critiques sont toutefois à résoudre en amont dans le code Fortran, soit pratiquement :

1. une modularisation minimale nécessaire pour identifier les étapes de la simulation, de l'acquisition des paramètres à la restitution finale des résultats ;
2. une restructuration minimale pour supprimer des formes ambiguës dans quelques sous-programmes ;
3. des adaptations relatives à la gestion des fichiers et quelques singularites propres au langage Fortran ;

Ces corrections, décrites dans la section suivante (A.3), sont indépendantes des algorithmes d'évaluation : leur intégration dans les futures versions permettrait alors d'automatiser complètement la fabrication des bibliothèques et applications dans les langages cibles.

Chaine de traitement du code Fortran

La production du code cible à partir du code source est réalisée par une suite de commandes *(S1,S2,S5,S6,S8)* qui utilisent des suffixes prédéfinis pour gérer des fichiers de directives.

Soit *monstics* un nom générique.
Trois fichiers de directives doivent être fournis au départ :

1. *monstics.liste* contient la liste des modules sources Fortran ;
2. *monstics.f* gère la liste des fonctions intrinsèques Fortran utilisées ;
3. *monstics.role* attribue les rôles aux unités de programmes Fortran pour pouvoir effectuer la vectorisation ;

Trois autres fichiers, optionnels, sont pris en compte s'ils sont présents.

- *monstics.banni* fournit une liste de noms d'instructions à ignorer (1.1.7) ;
- *monstics.copie* pilote les copies locales de variables communes dans les fonctions citées, les noms étant alors utilisés en clair dans le code C (2.7) ;
- *monstics.table* gère des directives de génération de code (3.9.1) ;

Les commandes peuvent être appelées une à une pour suivre les étapes de la compilation et en contrôler le déroulement, ce qui s'avère nécessaire pour traiter des versions qui utilisent de nouvelles formes ou instructions.

Automatisation de la génération de code C

Pour une fabrication immédiate à partir d'une version courante, le script *gCstics* enchaîne ces commandes : il est appelé avec deux arguments, le nom générique et le répertoire contenant le code Fortran, soit

```
gCstics monstics Fmake
```

gCstics permet de refaire une compilation pour :

1. Fournir des directives de génération de code (3.10), afin de :
   (a) déclarer localement une ancienne variable commune, pour utiliser le nom dans le code de la fonction ;
   (b) ajuster la taille des anciennes tables, par exemple réduire la profondeur du sol ;
   (c) gérer les périodes antérieures en ne conservant que celles qui sont nécessaires, généralement la veille et parfois une durée de vie ;
2. Intégrer des corrections du code Fortran, dont les mises à jour de versions dans le langage source actuel, décrit par sa grammaire dans le chapitre 1 ;

A.1.3 Fabrication et mise au point des applications

Les applications sont fabriquées dans leurs répertoires avec les *makefile*, repris ou générés (1.1.7). Les simulations sont ensuite effectuées pour des cultures et l'équivalence des bilans édités après récolte est vérifiée.

Répertoires des cultures

Toutes les applications s'exécutent dans des répertoires fils du répertoire père qui contient le code exécutable, y compris le code source après désarchivage dans *F77*, pour lequel les chemins d'accès aux fichiers sont revus.

Les noms de ces répertoires fils sont ceux des cultures, convention utilisée lors de la mise au point : l'application C qui simule *maculture* recherche les vecteurs édités par l'application Fortran dans le répertoire homonyme.

Applications Fortran

Deux applications Fortran sont fabriquées.

1. ***fstics*** correspond au code source initial tel que compilé dans les applications WINDOWS. Des corrections minimales sont d'abord effectuées pour :
   - supprimer les quelques erreurs ou avertissements du compilateur *g77* sous LINUX ;
   - résoudre des problèmes de formats WINSTICS pour pouvoir exécuter complètement la simulation d'une culture avec l'édition d'un bilan final ;
   - supprimer quelques erreurs trouvées dans le code source ;
2. ***gstics*** est une version révisée dans laquelle des restructurations minimales sont effectuées pour pouvoir automatiser la fabrication des versions C ;
   Elle intègre également les sous-programmes de suivi recopiés par le script *copieF* pour éditer les vecteurs à la période choisie.

Les fichiers de paramètres sont recopiés tels quels dans les répertoire d'exécution, *F77* pour *fstics* et *Fmake* pour *gstics*, soit

- cinq fichiers avec des noms prédéfinis :
  `travail.usm,param.par,paramv6.par,param.sol,stat.dat` ;
- deux autres fichiers pour les paramètres de la plante et technique dont les noms sont fournis dans `'travail.usm'` ;

Applications C

Le script *copieC*, appelé dans le répertoire père, recopie dans le répertoire de réception de l'application fourni en argument, ici *G*, l'ensemble du code généré dans les répertoires *C* et *D*.

Un autre script, *copieS*, permet le transfert des fonctions de supervision, conservées dans *Super*, les noms des répertoires figurant en arguments.

Le fichier *makefile* est à mettre à jour, une fonction C étant associée à chaque unité de programme Fortran.

Les fichiers de paramètres sont aussi à recopier dans le répertoire fils pour la culture mais ils sont renommés, avec comme radical le nom de la culture, et des suffixes prédéfinis pour les sept fichiers précédents : *usm,par,pv6,sol,,cli,plt,tec.*

L'application C d'initialisation des cultures *Stics0*, est fabriquée. Son exécution simule la culture à l'identique de l'application Fortran source, mais avec quelques informations en cours d'exécution, une gestion des déroutements, et des noms de fichiers explicites pour les éditions finales.

Restitution d'une forme vectorielle En l'absence d'arguments, la commande édite les vecteurs après lecture dans des fichiers textes, ce qui va permettre de s'affranchir ensuite des fichiers dits de paramètres. Une autre période, après initialisation, en cours ou en fin de simulation, peut aussi être choisie.

Validation des applications

L'examen des bilans permet une première vérification de l'équivalence des simulations.

Afin de vérifier l'équivalence de toutes les valeurs à toutes les périodes pour les applications Fortran et C avec une précision suffisante, des fonctions ont été développées autour du dictionnaire des anciennes variables communes, pour gérer la correspondance avec les composantes des vecteurs (3.2.7).

Les contrôles sont pilotés par l'application C, qui lit l'état à la même période édité dans le répertoire de la culture par l'application Fortran.

Comparaison des évaluations à une période donnée Un argument des applications C permet de fournir la période : les vecteurs édités par l'application Fortran sont alors lus et les écarts au seuil de précision choisi localisés.

Pour pouvoir interpréter ces écarts, les déroulements des algorithmes dans les applications C et Fortran sont suivis en parallèle avec le metteur au point *gdb* depuis le code C : les numéros des lignes sources Fortran figurent en commentaires et les structures de contrôle sont visualisées par les indentations. Les

arbres d'évaluation produits par *Cstics* sont éventuellement utilisés pour localiser la branche qui introduit l'écart.

Examen des dérives liées à la gestion de la précision Il arrive pratiquement que des écarts apparaissent en fin de simulation à partir du seuil de précision 10^{-5} : ces situations sont examinées, car révélatrices de problèmes numériques aux limites, qu'il s'avère utile de résoudre pour pouvoir effectuer les différentiations dans le contexte des analyses de sensibilité.

A.2 Application Fortran source

Les fichiers *makefile* sont à refaire d'une version à l'autre seulement si la liste des modules sources évolue, par ajout ou suppression d'une unité de programme : des fichiers sont disponibles pour gérer les dépendances et macro-instructions.[1] Les applications sont alors générées.

L'application Fortran source est fabriquée dans le répertoire *F77*, sous le nom *fstics*, et la commande appelée dans le répertoire fils. Son exécution simule normalement la culture, en mode "`batch`" en silence, mais des erreurs peuvent avoit interrompue l'exécution pendant la lecture : en l'absence d'édition du bilan final, l'erreur est à rechercher en consultant le fichier historique.

A.2.1 Choix des unités de programmes

Une liste *fstics.liste* est constituée en recherchant les fichiers suffixés par '`.for`' et '`.inc`' après désarchivage : elle est complétée avec les modules sources '`message.for`' et '`format.inc`' rangés dans des répertoires définis pour les versions françaises, anglaises et allemandes.

Les modules '`lecusmPURE.for`' et '`lecparamv6.for` sont choisis pour les sous-programmes LECUSM et LECPARAMV6 : '`lecusmCASetPURE.for`', '`lecusmCAS.for`' et '`lecparamv6dom.for`' sont placés en commentaires, par ajout de la marque % en début de ligne.

La commande de précompilation *S*1 (1.1.7) génère à partir de cette liste le fichier *makefile* pour fabriquer l'application ***fstics.***

A.2.2 Répertoire de simulation de la culture

Le choix d'un répertoire d'exécution règle le problème des chemins d'accès en fonction des systèmes hôtes :[2] les corrections sont à effectuer dans LECPARAM, LECPARAMV6, LECPLANT, ECRIPARAM, LECSORTI,LECLAI, LECTURES.[3]

[1] L'utilitaire *Sversion* signale les modules modifiés d'une version à l'autre.

[2] Certains fichiers sont recherchés dans le répertoire père.

[3] Les chemins sont parfois gérés dans des variables textuelles locales avec des fonctions ou opérateurs Fortran (LECTURES).

A.2.3 Accès aux fichiers et formats

La lecture des paramètres pose différents problèmes résolus à vue.

Désignation des fichiers

Une première difficulté est relative aux noms de fichiers lus avec les paramètres dits initiaux : majuscules et minuscules n'étant pas équivalents, le fichier attendu n'est pas trouvé. Il suffit alors de renommer correctement le fichier WINSTICS (paramètres techniques ou de la plante) dans `'travail.usm'`.

Formats des fichiers de paramètres

Un problème difficile à localiser est l'erreur de lecture, qui se solde par un déroutement signalé comme ultime message dans le fichier historique, type '`erreur dans le fichier plante`', l'exécution paraissant normale. Ces erreurs concernent résultent d'un ou plusieurs espaces manquants dans un champ d'un enregistrement géré par WINSTICS.[4] Elles apparaissent fréquemment pour les paramètres

- **techniques**, sur les champs '`0 : taille`' et '`0 : culture sous abri froid`' en fin de fichier ;
- **de la plante**, avec un espace manquant en fin de ligne dans le champ '`F : croissance et rendement`' ;

Contrôle des formats La commande *Sformat* localise ces décalages : il suffit ensuite de réintroduire les espaces manquants (4.2.2).

Séries météorologiques partielles

La période de simulation lue dans `'travail.usm'` est ajustée à la longueur des séries climatiques, ce qui explique une récolte à la date du 14 juillet d'une orge de printemps avec 27% d'eau : la prolongation de la série déplace cette récolte le 22 juillet avec 16% d'eau.

Réordonnancement de la lecture des paramètres nouvelle version

L'ordre des variables du fichier WINSTICS '`paramv6.par`' diffère de l'ordre de lecture depuis LECPARAMV6 dans la boucle avec branchement arrière pour la variable SWFACMIN. En rétablissant l'ordre de lecture dans le code Fortran, les paramètres seraient lus en une itération, et le branchement sans objet.

Valeurs indéterminées

La valeur 999 code l'indétermination pour la profondeur et la densité du semis pour un sol nu, ce qui génère une exploration des couches du sol dans

[4] D'autres erreurs demeurent possibles : elles sont à rechercher en localisant l'instruction de lecture Fortran qui a généré le déroutement, branchement vers une instruction finale.

STRESSEAU avec cette valeur comme borne : une solution est de remplacer cette valeur par 0 dans le fichier des paramètres techniques.[5]

A.2.4 Variables locales non initialisées

La non initialisation des variables locales font qu'elles peuvent contenir des valeurs aléatoires. Dans la traduction C, les initialisations sont effectuées avec une valeur nulle, d'où des différences à l'issue des évaluations.

Ces non initialisations peuvent aboutir à des erreurs conséquentes.

- La table EXCEL(0 :5 dans LIXIV, transmise en paramètre de TRANSF, simule des transferts entre réservoirs d'excès d'eau fictifs, valeurs des variables locales d'un sous-programme appelé antérieurement, l'espace étant réalloué pour les variables locales courantes : cela se traduit pour l'exemple de l'orge avec le compilateur *g*77 par un écart de 200 kilogrammes à la récolte.
- Un écart d'une vingtaine de kilogrammes par hectare de grains à la récolte sur l'exemple du tournesol s'explique par la non initialisation RNETA dans SHUTWALL.
- La variable logique PERTALLE dans BIOMAER contrôle des affectations.
- ...

Elles expliquent aussi des dérives quant à la précision.

A.2.5 Formes syntaxiques particulières

Le compilateur GNU *g*77 signale plusieurs erreurs :

- la table CUMDLTAREMOBILN gère un couple de valeurs mais l'indice n'est pas indiqué lors d'une affectation dans BIOMAER ;
- l'opérateur '.EQ.', propre aux variables numériques, est parfois utilisé à la place de '.EQV.' pour les variables logiques ou textuelles dans les versions antérieures ;

D'autres choix qui peuvent poser problème ne sont pas localisés :

- la déclaration REALQH2OF,... dans BILCOUPE n'est pas signalée ;
- le nom de variable DEFAULT, dans EAUQUAL, est à remplacer par DEFAUT pour éviter la confusion avec le mot-clé usuel des langages informatiques ;

A.3 Application Fortran révisée

L'application révisée[6] *gstics*, fabriquée dans *Fmake*, intègre les sous-programmes de suivi générés pour éditer les vecteurs aux périodes données. Une valeur est demandée avec la question '`suivi jour ?`' et l'exécution s'effectue ensuite comme *fstics*, en mode "`batch`" en silence, avec en sus des éditions les copies des vecteurs sous forme textuelle à la période choisie (0 avant la simulation).

[5] Il en résulte un avertissement dans le fichier historique pour forcer à 1 la profondeur.
[6] La commande *S*6 offre une option pour repérer des contextes problématiques (2.7.9).

Les corrections dans les fichiers sources sont repérées par la marque 'JCP' en commentaires.

Liste de contrôle Une nouvelle liste, '`gstics.liste`', reprend la précédente, intégre les nouveaux sous-programmes, et place en commentaires les modules sources inutilisés :

- la permutation des éléments des couples dans le sous-programme d'inversion de dominance DOMINVER rend inopérante la méthode de vectorisation (2.3.2) ;[7]
- les unités d'édition autres que les bilans (3.8.7) sont ignorées ;
- les fichiers d'inclusion inutiles sont supprimés ;[8]

Indentations Les blocs délimités par les structures de contrôle ne sont pas visibles en Fortran : l'éditeur GNU *Emacs* propose une option d'indentation, mais l'opération peut se traduire par un décalage en dehors de la zone de codage des instructions, soit après la colonne 72 des anciennes cartes, situation fréquente pour les instructions longues qui utilisent des cartes suites (1.1.2) : la zone de débordement est toutefois visualisée.

A.3.1 Modularisation minimale

La modularisation minimale vise à isoler les étapes successives dans des sous-programmes, auxquels sont associés des rôles, traduits par des directives de codage pour la vectorisation. L'objectif est de de faire que les rôles soient définis pour l'intégralité des unites de programmes, et non liés à des sections de code dont la localisation est affectée par les mises à jour Fortran (2.4.2).

Le classement des paramètres et données initiales est aussi associé aux rôles des unités de lecture : quelques réorganisations du code sont suffisantes pour éviter les ambiguïtés.

Architecture logicielle

Une réorganisation minimale du code Fortran est nécessaire pour pouvoir séparer les étapes de la simulation :

1. acquisition des paramètres et données initiales ;
2. initialisation de la simulation pour la campagne ;
3. évaluations quotidiennes ;
4. édition des résultats pour la campagne ;

Les opérations de lecture relèvent de la première étape et l'édition de la dernière, ce qui nécessite dans les versions actuelles le déplacement de quelques séquences de code, éventuellement dans de nouveaux sous-programmes.

[7] Une autre application C est à développer pour les cultures associées, le code généré supprimant les couples (3.2.1).

[8] L'unique fichier rémanent est '`comfruit.inc`', lequel redéclare la variable DEVJOURFR déclarée aussi dans '`commun.inc`'.

Le regroupement des séquences associées à ces étapes dans des sous-programmes va permettre une génération directe des fonctions de supervision qui enchaînent ces étapes, avec les prototypes des fonctions appelées.

Constantes et séries climatiques

Le programme principal appelle LECTURES, lequel exécute une séquence d'acquisition de paramètres qui se termine par la lecture de cinq constantes dans l'entête du fichier des paramètres climatiques "`stat.dat`".[9]

L'acquisition des séries climatiques de la campagne s'effectue dans LECSTAT, appelé depuis INITIALISATIONS, lui-même appelé dans la boucle sur les cultures du programme principal à l'issue des initialisations. Elle démarre par une première lecture de l'enregistrement courant pour extraire la date de début JULZERO, suivie d'une instruction BACKSPACE pour pouvoir ensuite relire l'enregistrement dans une boucle temporelle.

Ce mode d'acquisition est repris dans deux nouveaux sous-programmes, LECCONSCLI et LECMETEO.

1. **Constantes et dates**
   LECCONSCLI lit les variables d'entête puis la date de début des séries JULZERO. Appelé depuis LECTURES, il ouvre et ferme le fichier.
2. **Séries météorologiques**
   LECMETEO ouvre le fichier et se positionne sur le premier enregistrement quotidien : les séries sont lues et l'ultime période est rendue pour mettre à jour la date de fin des séries JULFIN. L'appel est effectué depuis LECSTAT, qui ferme le fichier.

Cette restructuration se traduit par des déplacements de code depuis
- LECTURES et LECSTAT vers LECCONSCLI,
- LECSTAT vers LECMETETEO.

Gestion des indices des tables bisannuelles Les valeurs quotidiennes sont lues dans des boucles : l'indice dans les tables est géré de façon ambiguë, d'abord par affectation de la valeur de la variable de boucle puis par incrémentation, la dernière période bisannelle, de 1 à 731, étant notée dans JULFIN. Ces séquences sont à reprendre afin que l'indice de ces tables désigne bien la période.

Séries manquantes Le compilateur Fortran *g77* génère une erreur si le nombre de champs dans l'enregistrement est inférieur au nombre de séries, dans la liste et au vu du format, pratiquement 5 au lieu de 7.

Continuité calendaire et années bissextiles LECSTAT initialise les nombres de jours des années de semis et récolte avec une formule qui utilise des divisions entières et n'est pas fidèle au calendrier grégorien. Ces initialisations sont reprises avec l'opération entière *modulo*, fonction MOD() en Fortran.[10]

[9] Il s'agit des variables NOMETP,ALPHAPT,PARSURRG,NBANS,LATITUDE.

[10] La programmation actuelle génère un bug en 2100.

Initialisation des sols et données de la culture précédente

Les données issues de la simulation de la culture précédente sont lues par le sous-programme d'initialisation de la culture INITNONSOL, appelé depuis INITIALISATIONS, dans le fichier '`recup.tmp`' (4.4.2). L'entête identifie la période précédente : un message clôt l'exécution si les dates des simulations successives sont incompatibles.

Ces lectures, contrôlées par deux options (4.3.9), sont déplacées dans un nouveau sous-programme, LECPREC, appelé si CODESUITE $= 1$, qui lit cette entête, et le reste du fichier si CODEINITPREC $= 2$. L'appel est finalement effectué dans INITIALISATIONS, avant la séquence d'initialisation des sols, ce qui permet de gérer les rôles des unités de programmes

La suppression de ces lectures dans INITNONSOL nécessite une reprise des structures conditionnelles de contrôle. Les séquences de lecture étant déplacées, la séquence résiduelle d'initialisation associée au bloc ELSE est reprise dans un bloc THEN, qui s'exécute alors si (CODESUITE $\neq 1$) $\vee$ (CODEINITPREC $\neq 2$).

Cette restructuration se traduit par

- un déplacement de code depuis INITNONSOL vers LECPREC, avec reprise des contrôles conditionnels,,
- l'ajout de l'appel conditionnel de LECPREC dans INITIALISATIONS.

Relecture de paramètres du sol Un ensemble de valeurs lues avec les paramètres du sol, dont des codes textuels, sont relues avec les données de la culture précédente : les variables non réévaluées pourraient être supprimées de ces listes, simultanément dans l'édition de '`recup.tmp`' et pour la relecture.

Initialisation de la campagne

Les corrections des séries climatiques relèvent de l'étape d'initialisation.

1. L'affectation de la valeur minimale `"0.02"` dans TVENT est à effectuer dans la boucle de INICLIM plutôt que quotidiennement dans SHUTWALL.
2. La correction de TPM par appel de la fonction TVAR est aussi à effectuer dans cette boucle plutôt que quotidiennement dans CALRNET ou KETP.
3. Les mises à jour des températures sont à faire par appel de CLIMABRI dans cette même boucle.

Conversion des précipitations en irrigations L'appel de AFFECTRRQUINOAPOQUET dans CALCPRECIP est occulté pour éviter la remise à zéro de TRR.

Paramétrage de fonctions d'initialisation Le sous-programme INICLIM utilise la variable commune N comme variable de boucle, les sous-programmes appelés l'utilisant pour accéder aux séries. CALPENMAN n'étant appelé que dans cette étape d'initialisation, il peut être redéfini avec la période en paramètre.

Relecture de paramètres de la plante en cours de simulation

Dans la version 2008, si la culture ne peut pas démarrer parce que le sol est trop sec, le sous-programme DENSIRAC lit de nouveaux paramètres de la plante dans le fichier '`solnu.plt`' : il paraît opportun de rétablir la situation antérieure, soit l'édition d'un message suivi d'un STOP qui clôt l'exécution.

Forçage et initialisation des simulations

Les séquences de forçage introduites dans le programme principal sont transférées dans INITIALISATIONS.

Mise à jour de paramètres généraux

Deux séquences du programme principal modifient des paramètres généraux :

1. CO2 reçoit de nouvelles valeurs en fonction de l'année d'installation de la culture ANSEMIS : la fonction fortran TAUXCO2(ANSEMIS) rend cette valeur ;
2. ALPHACO2 est modifié en fonction du paramètre de la plante CODEPLANTE, textuel ;

Lecture de dates et valeurs de forçage

L'appel de LECEPITAF pour le forçage des besoins en azote est déplacé dans INITIALISATIONS.[11]

Evaluations quotidiennes

Le transfert du corps de la boucle des simulations quotidiennes dans un sous-programme propre, SIMULER, avec reprise des inclusions, permet d'attribuer un rôle dynamique à cette unité, sans devoir localiser les sections de code dans un long programme principal pour coder ces rôles pour chaque version (2.4.2).

L'appel de SIMULER est effectué depuis le programme principal dans une boucle DO classique pour la durée de la période de simulation, en remplacement de la structure DO WHILE ... ENDDO introduite dans la version 6.

Afin de séparer les tâches, et ne pas devoir gérer dans SIMULER les paramètres d'édition, la séquence finale de la boucle quotidienne est déplacée dans le sous-programme JOURRAPPORT.

Visibilité des modules Certaines séquences sont regroupées dans de nouveaux sous-programmes pour isoler des modules, en relation avec la fabrication de méthodes natives pour des applications objet (A.5.2).

- PHOTOSYNTHESE, issu de LAIDEV, pilote l'évaluation de l'indice foliaire ;
- AZOTEEAU traite des bilans en eau et en azote ;
- MICROCLIMAT reprend les séquences d'évaluations des températures et humidités du couvert végétal ;
- ETATPARCELLE traite de l'état du sol au semis ou à la récolte ;
- FINAL regroupe les ultimes évaluations de la boucle quotidienne ;

[11] L'appel de LECLAI pour les indices foliaires s'effectue dans ce sous-programme.

Retour conditionnel Plutôt que de fabriquer un sous-programme avec une seule instruction conditionnelle pour isoler un module, un retour conditionnel immédiat peut être utilisé dans les sous-programmes appelés depuis LAIDEV si CODEPLANTE est différent de "`snu`", soit DEVELOP et PHOTOSYNTHESE.

Boucle des évaluations quotidiennes La durée de la simulation est normalement fournie par les paramètres initiaux IWATER et IFWATER pour la durée MAXWTH. Cette durée de campagne peut ensuite être mise à jour :

1. par réduction après lecture des séries climatiques si elles sont incomplètes, MAXWTH étant réévaluée en fonction de JULFIN ;
2. par incrémentation lors des évaluations quotidiennes dans BFROID si la vernalisation est en cours, et accessoirement dans SENESCEN ;

Il peut aussi arriver que cette durée dépasse la longueur d'une campagne, soit au plus 366 jours.

Ces situations font l'objet d'adaptations contextuelles dans le code Fortran qui sont sources d'ambiguïtés.

Boucle sur les cultures La constante NBANS lue en entête du fichier climatique est décrémentée si la culture n'est pas annuelle. Des contrôles sont ensuite effectuer pour pouvoir enchaîner les cultures dans INICLIM mais ils ne s'exécutent que si NBANS > 1.[12]

Editions

Les éditions en cours d'évaluation sont effectuées par appels de sous-programmes, sauf dans SORTIE. Deux nouveaux sous-programmes sont fabriquées par transferts des séquences d'instructions.

1. BILSORTIE reprend les inclusions et les variables locales de SORTIE. Il démarre avec l'appel de JULIEN pour transcrire les dates. Les éditions sont ensuite contrôlées par l'option CODESIG, avec une seule instruction de copie pour la valeur 3, puis des séquences d'édition dans plusieurs fichiers ouverts pour la valeur 0, laquelle est associée à la non exécution de l'instruction RETURN dans SORTIE.[13]
   L'affectation dans la variable locale MATUBER est aussi à reprendre.
2. JOURRAPPORT est appelé depuis SIMULER pour contrôler les sorties quotidiennes.

Données pour la culture suivante Le sous-programme RECUP est appelé pour toutes les cultures : l'appel pour les coupes des cultures fourragères dans INICOUPE est sans objet.[14]

[12] Le rétablissement de la valeur lue de NBANS se solde par un message qui clôt l'exécution.

[13] C'est pratiquement la séquence entre les appels de LECSORTI et LECPROFI, des mises à jour étant effectuées ensuite.

[14] Le fichier `'recup.tmp'` est ouvert puis fermé pour chaque exécution.

A.3.2 Restructuration minimale

Le compilateur *Cstics* reconnaît finalement la quasi-totalité des structures de contrôle pilotées par des branchements, et génère un code qui les supprime. Il restait toutefois quelques structures complexes ou ambiguës pour lesquelles une restructuration s'est avérée nécessaire dans le code Fortran, pour lever toutes ambiguïtés et rendre les algorithmes lisibles.

Evaporation du sol nu

Le sous-programme SOLNU contient une séquence complexe de branchements en avant et en arrière qui traduisent finalement des choix et une itération (4.5.14). Ce code est repris dans un nouveau sous-programme, EVAPOSOL, appelé depuis SOLNU. L'itération est traduite par une boucle *tantque* classique traduite en Fortran avec un branchement conditionnel final.[15]

Sénescence foliaire

La séquence de traitement des productions les plus anciennes pour évaluer la sénescence dans SENESCEN utilise deux conditions logiques pour gérer une itération dans un bloc conditionnel au moyen de branchements : le code est repris et l'itération, contrôlée par une variable d'état, est traduite sous la forme usuelle en fortran par un unique branchement conditionnel final.

Transferts dans les couches du sol

1. Une itération dans LIXIV est contrôlée par un branchement arrière, unique instruction dans une boucle finale DO WHILE (< *condition* >), à remplacer par un branchement conditionnel pour piloter l'itération.
2. La recherche d'une couche de transition dans TRANSF, appelé par LIXIV, s'effectue dans une boucle DO avec comme variable de boucle le rang ICOU de cette couche, ensuite utilisé, et remis à zéro à l'issue d'une sortie conditionnelle contrôlée par un branchement : cette structure complexe est à remplacer par une boucle classique.

Répartition de la biomasse entre les organes

Un branchement dans REPARTIR depuis un bloc conditionnel imbriqué renvoie vers une séquence finale après récolte : l'utilisation d'une variable d'état permet de traduire cette structure conditionnelle de façon classique.

Initialisation après coupe des cultures fourragères

Le sous-programme INICOUPE n'effectue pas que des initialisations mais plusieurs actions (4.3.5).

[15] L'algorithme effectif, devenu lisible, traduit des changements d'états.

1. Il évalue d'abord la variable logique SIONCOUPE pour déterminer si la coupe s'effectue à la date courante, les dates des coupes pouvant être des paramètres ou évaluées, et gère des branchements. Si SIONCOUPE est vrai, la décision de coupe peut être remise en cause dans une séquence conditionnelle mais seulement avant l'ultime jour de simulation, une initialisation étant effectuée le dernier jour.
2. Si la coupe est effective, il réinitialise des variables évaluées quotidiennement à partir de valeurs fournies avec les paramètres de la culture.
3. Des évaluations finales sont ajoutées en fin de sous-programme, qu'il y ait coupe ou non.

Ce sous-programme est à restructurer. L'algorithme est ici repris dans un nouveau sous-programme, COUPE, pour supprimer les branchements depuis des blocs conditionnels, séparer les tâches, et lever l'ambiguïté introduite par le nom du sous-programme.

L'évaluation de la décision de coupe s'effectue à différents niveaux : elle est contrôlée depuis des blocs conditionnels, par des branchements vers deux cibles, l'une pour poursuivre l'évaluation, l'autre pour effectuer les évaluations finales. Ces branchements sont supprimés et c'est SIONCOUPE qui est utilisée pour contrôler les choix, dont l'exécution de séquences résiduelles en l'absence de branchements depuis des blocs conditionnels.

INICOUPE prépare les éditions, appelle BILCOUPE pour restituer les résultats, puis effectue des réinitialisations : les séquences de code avant et après édition sont reprises respectivement dans AVANTCOUPE et APRESCOUPE, les appels étant effectués depuis COUPE.

COUPE se termine avec les évaluations finales.

A.3.3 Ambiguïtés et anomalies de programmation

Les erreurs dans le code source, type non initialisation de variables locales, qui peuvent se traduire par des variations quelques dizaines de kilogrammes de grains par hectare à la récolte, sont normalement corrigées. D'autres écarts relativement très faibles, de l'ordre du milligramme, peuvent apparaître, et révéler des problèmes aux limites.

Continuité des évaluations quotidiennes dans le système sol-plante

Dans TRANSPI, la répartition dans les couches du sol sur l'ensemble de la zone racinaire des quantités gérées dans les variables locales DISPO et RESTEZ s'effectue dans une boucle seulement si (DISPO > 0), ce qui se traduit par un décalage sur les périodes suivantes de la quantité RESTEZ > 0.

L'expression logique (DISPO > 0) utilise la variable locale de cumul EPRECAL : le résultat est affecté par le mode de gestion de la précision, les répartitions pouvant alors s'effectuer à la période suivante pour les codes Fortran et C. Les

différences apparaissent en cours d'évaluation aux périodes concernées puis disparaissent à la période suivante.[16]

Pour supprimer ces différences, et rétablir la continuité quotidienne, les quantités DISPO et RESTEZ sont réparties à chaque période dès lors qu'elles sont positives : les évaluations sont alors équivalentes.

Désignation des paramètres formels

Certaines fonctions sont définies avec des paramètres formels homonymes de variables communes, ce qui introduit des ambiguïtés et nuit à la lisibilité.[17]

Contrôle de la profondeur racinaire

Si le front ZRAC dépasse l'obstacle à l'enracinement OBSTARAC, la couche de profondeur OBSTARAC + 1 est explorée dans DENSIRAC : une solution est de limiter ZRAC à la profondeur OBSTARAC − 1 après évaluation dans CROIRA.[18]

Remise à zéro de profils racinaires

La table LRACSENZ définie sur la profondeur du sol est indicée par la date N de la simulation dans CROIRA : c'est une erreur et les remises à zéro sont à faire dans une boucle.

Taille des tables

EPCINI(2) dans TASSESEMISRECOLTE est une table définie pour les horizons mais restreinte aux deux premiers : EPCINI(5) restituerait sa fonctionnalité et permettrait de la distinguer des tables qui gèrent les couples de cultures.

A.3.4 Variantes du langage Fortran

Les dernières versions introduisent des formes antérieurement traduites autrement et de nouvelles instructions sans intérêt ;

Initialisation de constantes

Le module '`dynamictalle.for`' introduit d'autres modes :

1. GSER et GCD utilisent l'instruction PARAMETER d'initialisation de variables locales déclarées ;
2. GAMMLN utilise localement le type DOUBLE PRECISION avec la notation D (`x+5.5d0`) dans une liste d'initialisation DATA ;

[16] Sur l'exemple du maïs, un suivi des valeurs révèle une différence se quelques cenntaines de grammes à la récolte, non visible dans les bilans.

[17] HUCC et HUMIN, variables communes définies sur les couches du sol, sont aussi paramètres formels de la fonction POTSOL, appelée avec les paramètres effectifs HUCC(IZ) et HUMIN(IZ).

[18] Ces problèmes sont révélés dans les versions C par la fonction *Sol*() qui contrôle ce type de débordement si la profondeur maximale est ajustée (3.10.2).

La version révisée revient aux initialisations classiques en simple précision, sans réintroduire l'ancienne liste DATA avec des variables simples.

Initialisation DATA L'initialisation à zéro dans une instruction DATA de la variable locale ETN(1000) dans LIXIV est à remplacer par une initialisation classique dans une boucle.[19]

Boucles à pas constant sans étiquette

La traditionnelle boucle DO Fortran $< n > i = inf, sup[, pas]$ est parfois remplacée dans les dernières versions par une forme qui supprime l'étiquette finale $< n >$, l'itération se terminant sur l'instruction ENDDO.[20]

Si ces formes sont finalement transformées dans l'étape initiale de normalisation du code (1.1.5) au moyen d'étiquettes fictives (avril 2008), le retour à la forme d'usage et à la norme du langage *Fortran77* ne ferait que rendre le code source plus clair, en particulier pour les boucles imbriquées.

Nouvelles instructions

De nouvelles instructions s'avèrent en fait inutiles.

1. SAVE, utilisée dans GAMMLN pour des variables locales rémanentes, est sans objet vu qu'elles sont initialisées dans un DATA.
2. PAUSE interrompt l'exécution et affiche un message : il faut frapper un caractère pour poursuivre. Ces cas étaient antérieurement traités par un message dans le fichier historique.

A.3.5 Déclaration des variables communes

Quelques corrections sont à faire dans les fichiers inclus.

Unicité des déclarations dans les communs

Les variables sont déclarées dans un seul commun, les communs étant déclarés dans des fichiers inclus dans les modules sources : cette règle fait que l'instruction COMMON n'introduit pas de synonymies (1.2.3). Un seule exception est relative au fichier jamais inclus '`comfruit.inc`' qui redéclare l'unique variable DEVJOURFR : ce fichier est à supprimer pour prévenir des erreurs.

Regroupement des tables et reconnaissance des séries

Les tables sont déclarées dans ces fichier avec des tailles types, reprises pour contrôler la vectorisation, dont le regroupement des séries, tables repérées par la valeur 731 de la borne supérieure. Deux corrections sont à effectuer pour construire ces séries par substitution de cette valeur 731 aux bornes :

[19] L'ajustement de la profondeur dans les applications C est ainsi pris en compte.

[20] Sous-programmes CALCPRECIP, COUPE issu de INICOUPE, INITIALISATIONS, INITNONSOL, LECSOL et programme principal.

- 720, utilisées pour quelques variables dans '`comtransit.inc`';
- 1000 pour RACNOY déclarée dans '`comvarplt.inc`' et utilisée comme série ;

Profondeur du sol et zone humifère

La zone de travail du sol, initialement de 60 centimètres, passe à 1000 pour les trois matrices NBIO,CBIO,CRES qui gèrent l'incorporation des résidus. Le rétablissement de la valeur 60 pour identifier cette zone dans le fichier '`common.inc`' permettrait d'ajuster les tailles et d'économiser l'espace pour les spatialisations, pour un territoire ou par maillage *(Hysafe)*.

A.3.6 Cultures associées

Le mode de traitement des cultures associées demeure source d'ambiguïtés, et de difficultés pour reconnaître les formes.

Références à des données manquantes Des indéterminations demeurent si la culture associée ou la zone ombragée sont référencées (3.2.1).La traduction C suppose que les valeurs par défaut sont nulles et, en conséquence, annule les termes indéterminés : le résultat est erroné pour les valeurs par défaut unitaires.

Les fichiers produits lors de la génération du code permettent de lever ces indéterminations en ajoutant les choix conditionnels (3.9), comme l'évaluation de INNLAIC dans INITNONSOL pour les cultures fourragères.

Forçage paramètre culture principale Le paramètre général lu ALPHACO2 est forcé '1.1' pour les valeurs ("mai", "sor", "can") du paramètre de la plante CODEPLANTE, à '1.2' sinon, Cette variable textuelle est définie pour un couple de culture et l'instruction conditionnelle s'exécute dans une boucle de 1 à 2 : ALPHACO2 reçoit donc la valeur pour la culture associée, en l'occurrence '1.2' pour toute culture simple.

Contrôles des séquences conditionnelles SHUTWALL définit la variable logique DEUXCULT mais utilise d'autres variables pour contrôler certaines séquences, dont la négation de la condition logique utilisée pour évaluer DEUXCULT.

A.3.7 Autres

De menues corrections dans le source Fortran sont suffisantes pour réduire le nombre de paramètres des fonctions C générées.

- Les variables météorologiques lues ne sont pas utilisées dans la boucle quotidienne, sauf la série ANNEE pour un message dans le fichier historique depuis FRUIT qui édite l'année en cours : la suppression de cette référence supprime le passage du paramètre $W_[]$ dans les fonctions C générées pour les évaluations quotidiennes.
- L'initialisation de QNPLANTECTOT, variable d'édition, est à supprimer dans INITNONSOL pour supprimer le passage du paramètre $Z_[]$.

– Les variables d'édition CU0 et SOMELONG0 sont à supprimer dans INICLIM, car alors indéterminées ou initialisées à zéro par le compilateur.

A.3.8 Fabrication de l'application Fortran révisée

D'ultimes interventions sont nécessaires pour produire l'application.

Edition de liens

Les unités de programmes des modules supprimés sont regroupés dans le module source '`fantome.for`' sous la forme d'une suite de sous-programmes vides, pour éviter la génération de références non satisfaites lors de l'édition de liens, les appels conditionnels étant conservés.

Fabrication

Le script *copieF* recopie les sous-programmes de suivi depuis F vers $Fmake$. Après mise à jour du fichier $makefile$, sans omettre FANTOME, La commande 'make gstics' fabrique l'application *gstics*.

A.4 Applications C

La version C équivalente est produite par compilation du code Fortran révisé puis intégration des fonctions générées dans les modules de supervision.

A.4.1 Génération automatisée des fonctions

Les étapes de la compilation sont pilotées par des commandes qui lisent des directives dans des fichiers, créés explicitement ou produits dans les étapes précédentes.

Directives de compilation

Plusieurs fichiers sont utilisés dans les différentes étapes de la compilation. Soit l'application *gstics*.

1. **Reconnaissance du code source et classification des variables**
   Trois fichiers sont nécessaires pour traiter le code Fortran et vectoriser les variables informatiques :
   – "`gstics.liste`" fournit la liste des modules sources retenus ;
   – "`gstics.f`" contient la liste des fonctions intrinsèques Fortran et les fonctions C équivalentes ;
   – "`gstics.role`" code les rôles des unités de programmes (2.4.2), dont héritent les unités appelées.
   A l'issue de la modularisation minimale (A.3.1), il suffit d'attribuer les rôles aux unités de lecture, d'initialisation et d'édition, et le rôle dynamique à SIMULER, les unités d'évaluation héritant de ce rôle.

Le fichier optionnel "`gstics.banni`" permet d'ignorer provisoirement certaines instructions(1.1.7).

2. **Gestion des variables**
   "`gstics.copie`" contient des noms de variables communes à déclarer localement dans les fonctions, une trame étant générée pour toute fonction inconnue (2.5.2) ;
3. **Vectorisation**
   "`gstics.table`" permet d'ajuster les tailles des variables du modèle gérées dans des tables, pour les durées et les couches du sol (3.10.2) ;

D'autres directives peuvent être fournies pour traiter un nouveau code : "`gstics.nongoto`" énumère des unités de programmes dans lesquelles les branchements ne sont pas traduits, les étiquettes logiques étant alors gérées dans une variable locale d'état (3.7.3) ;

Génération automatisée

Le script *gCstics*, appelé avec les noms de l'application et du répertoire Fortran en arguments, enchaîne les commandes *Cstics* et édite l'ensemble des messages associés. En cas de difficultés, les commandes peuvent être exécutées une à une et les comptes-rendus analysés.

Le résultat est un ensemble de fonctions C prototypées issues des unités de programmes Fortran, et quelques fonctions et fichiers à inclure utilisés par les modules de supervision, dont la gestion du dictionnaire des variables.

Appels des unités de programmes non traduites Les unités non appelées des modules sources non retenus dans la liste sont remplacés par des sous-programmes vides, regroupés dans le module source '`fantome.for`'.

A.4.2 Fabrication logicielle

Des modules de supervision sont définis pour piloter les simulations par appels de fonctions vectorielles(4.2.7).

1. **Stics0** reproduit à l'identique l'application Fortran, avec la lecture des fichiers WINSTICS, puis édite les vecteurs dans des fichiers.[21] L'application offre les fonctionnalités de mise au point pour comparaison avec les évaluations Fortran.
2. **Vstics** lit les jeux de vecteurs et effectue les simulations.

Fonctions développées

A côté du programme principal sont développées des fonctions indépendantes des versions Fortran, regroupées dans le répertoire *Super*.

Le script *copieS* recopie ces fichiers depuis un ancien répertoire vers le répertoire de la nouvelle version, ces répertoires étant fournis en arguments.

[21] La copie s'effectue après la mise à jour des paramètres généraux CO2 et ALPHACO2(A.3.1).

Fonctions générées

Le script *copiC* recopie le code source depuis C et D vers le répertoire dont le nom est fourni en argument, par exemple G.

Génération des applications

Un *make*, mis à jour au vu des ajouts et suppressions signalés pas $Sversion$ pour les nouvelles versions, compile l'ensemble des fonctions, développées ou générées, et produit les applications C exécutables, $Stics0$ et $Vstics$.

A.4.3 Validation

La validation est effectuée par comparaison des vecteurs contenant les variables communes à des périodes données pour les applications Fortran et C.

Soit une simulation de culture *maculture* : deux répertoires fils *maculture* sont créés sous *Fmake* et G.

Les fichiers de paramètres sont recopiés dans ces répertoires d'exécution : les simulations des deux applications peuvent être lancées et une comparaison demandée pour une période.

Le compte-rendu de la comparaison est édité dans '`maculture_trace`'. Si des écarts existent, le fichier compte-rendu indique quelles fonctions évaluent les variables concernées.

Ces mises à jour peuvent être localisées par examen du graphe des relations entre variables, et le processus d'évaluation reconstitué au vu de l'arbre des appels (2.7). Les structures de contrôle de ces évaluations peuvent aussi être restituées en consultant les tables descriptives éditées dans des fichiers (1.3.7).

Si nécessaire, les algorithmes sont déroulés avec le metteur au point, *gdb*, simultanément pour les applications Fortran et C, les rangs des instructions Fortran dans les modules sources figurant en commentaires des instructions C, et le nom du module au début.

Acquisition des paramètres et initialisations

Il faut d'abord vérifier l'équivalence avant les évaluations quotidiennes.

1. **Application Fortran**
   Depuis le répertoire *maculture* sous *Fmake*, la commande
   ../gstics
   affiche immédiatement
   `suivi jour ?`
   et il faut alors répondre 0 pour éditer dans des fichiers les vecteurs avant les évaluations quotidiennes.
2. **Application C**
   Depuis le répertoire *maculture* sous G, la commande
   ../Stics0 mais 0
   va lire les vecteurs Fortran dans *maculture* sous *Fmake*,
   et comparer avec les vecteurs C avec la précision par défaut, 10^{-6}.

Si des écarts sont signalés, il faut pour les variables lues examiner les compte-rendus édités pour chaque fichier (4.3.1), puis rechercher les dites variables dans les fichiers WINSTICS.[22]

Evaluations quotidiennes

Les équivalences sont vérifiées à l'issue des évaluations quotidiennes de la période de rang n, nombre transmis aux deux applications, ou le dernier jour de la simulation si n est postérieur à la période finale. Une option permet de choisir la précision des comparaisons, les dérives consécutives à la gestion de la précision dans les deux langages pouvant se traduire par l'édition d'écarts infinitésimaux en fin de simulation pour un seuil trop petit, pratiquement à partir de 10^{-6}.

Ainsi, pour une comparaison immédiate en fin de simulation pour une nouvelle version, la commande

../Stics0 mais 400 0.0001

effectue les comparaisons à l'issue de la dernière période, une campagne étant au plus annuelle, et édite les valeurs qui diffèrent en-deça de la quatrième décimale.

Version Fortran

gstics est appelé dans un répertoire fils de *Fmake* : la réponse 200 déclenche l'édition des vecteurs qui gèrent les variables du modèle à l'issue de la simulation de rang 200 ;

Version C

*Stics*0 `mais 200 0.0001`
effectue la comparaison des vecteurs à l'issue de cette même simulation ;

Si des divergences apparaissent, les algorithmes d'évaluation sont alors suivis au pas à pas pour les deux versions, le graphe des relations entre variables étant utilisable pour localiser ces évaluations (2.7.4).

Edition des vecteurs

Afin de pouvoir visualiser immédiatement les valeurs des paramètres du modèles évalués avant la simulation, ou les valeurs des variables en cours de simulation, l'option "`v`" à la place de la précision génère l'édition des vecteurs.[23]

[22] Les erreurs de lecture Fortran se soldent par un branchement final, avec parfois un bref message dans le fichier '`history.sti`'.

[23] La lecture de ces vecteurs pour redémarrer des simulations pose des problèmes de précision, les valeurs étant tronquées dans la représentation décimale.

A.5 Applications objets

Le traitement du code source Fortran produit un ensemble de quelque cent cinquante fonctions vectorielles, les paramètres et variables du modèle de culture étant gérés dans les vecteurs paramètres de ces fonctions, traduites en langage C, et compilées pour produire un "code objet" en langage machine : la compilation des sources "`*.c`" produit les cibles "`*.o`", qui peuvent alors être archivées dans la bibliothèque partagée *bibStics*.

Ces fonctions étant validées, elles sont mobilisables depuis cette bibliothèque dans des applications objets à travers des méthodes natives, le compilateur du langage objet récupérant le "code objet" lors de l'éditions de liens.

Les bibliothèques sont regroupées dans le répertoire *Lib*, le chemin d'accès étant ajouté dans la variable d'environnement LD_LIBRARY_PATH.

Cette approche a été mise en œuvre et traduite dans le langage JAVA.

A.5.1 Fonctions vectorielles Stics et méthodes natives

L'application objet définit une classe *Culture* avec comme champs les vecteurs et des méthodes qui utilisent des fonctions vectorielles.

L'application *Stics*0 :

- lit les fichiers WINSTICS, suites d'enregistrements Fortran ;
- numérise les variables textuelles utilisées dans les séquences d'évaluation ;
- gère les messages d'erreurs dans une fonction spécifique ;
- copie dans des fichiers les valeurs décimales des vecteurs (K, S, V, M, W, X, Y) avant initialisation de la culture ;

L'acquisition de ces vecteurs dans une application objet va permettre d'initialiser une campagne culturale sur une parcelle puis de simuler la campagne, avec le seul vecteur Y évalué quotidiennement, et une restitution de X en fin de campagne pour la culture suivante.[24]

L'utilisation dans des méthodes des fonctions vectorielles compilées nécessite de mettre en correspondance les vecteurs tels qu'ils sont représentés dans les langages C et JAVA, et transmis en paramètres : l'interface JNI (Java Native Interface) fournit les outils pour ces deux langages.

Construction des méthodes natives

Une classe *Stics* est créée pour regrouper les méthodes qui gèrent les appels des fonctions C vectorielles générées à partir du source Fortran. La commande *javah* génère la trame des fonctions associées dans des fichiers sources C pour ces méthodes, et le fichier d'inclusion "`Stics.h`" qui contient les déclarations des prototypes de ces fonctions. Les fichiers sources sont repris dans le répertoire des fonctions communes *JavaC* : le code source est complété et la fonction compilée.

Le logiciel *Stics* est structuré en modules et sous-modules, qui sont repris pour construire ces méthodes. La fabrication des fonctions C associées, à partir de la trame générée, requiert nécessairement trois étapes.

[24] Cette restitution est faite dans le code Fortran par RECUP dans le fichier "`recup.tmp`".

1. Mise en relation des paramètres des deux langages : un pointeur est fourni pour accéder à des méthodes qui reconnaissent les types de base des langages informatiques (entier, réel, caractère) et les tables qui regroupent ces types, soit une instruction de la forme
   *jfloat *V = (*env) →GetFloatArrayElements(env,plante,0)* ;
   avec *plante* vecteur reçu en paramètre de l'application objet ;
2. Ecriture du corps de la fonction qui appelle des fonctions vectorielles ; Déclaration des prototypes en entête ;
3. Libération des ressources avec une instruction de la forme
   *(*env) →ReleaseFloatArrayElements(env,plante,V,0)* ;

Avec le code généré actuel, il faut aussi déclarer une variable locale C de type FILE transmise en paramètre, et lui affecter une valeur : sortie standard *stdout*, fichier NULL ou fichier à ouvrir et fermer pour des fonctions d'édition.

Construction des vecteurs

Différentes stratégies de vectorisation peuvent être mises en œuvre pour piloter l'allocation dynamique des vecteurs, et adapter l'espace requis à des contextes applicatifs.[25]

Les tailles des vecteurs sont finalement lues en entête du dictionnaire et ensuite utilisées dans l'application objet pour construire les vecteurs. Les méthodes natives étant paramétrées avec des références vers les vecteurs *float *v*, il est ainsi possible de définir plusieurs classes qui utilisent les mêmes méthodes mais définies sur différentes bibliothèques.

Génération de méthodes natives indépendantes

Les choix conditionnels utilisent des variables *Stics* vectorisées avec les paramètres, et referencées dans le code généré par un rang de composante qui dépend des versions Fortran et du mode de vectorisation choisi pour les applications : la reprise de ces appels conditionnels nécessiterait de mettre à jour ces composantes pour toute nouvelle génération du code C.

Des solutions sont mises en œuvre dans le code source Fortran pour générer des méthodes indépendantes.

Regroupement des appels dans des sous-programmes
: Les appels conditionnels pour une méthode sont regroupés dans une fonction appelante : par exemple, *evalEtatParcelle()*, associée à la méthode qui reprend les évaluations de l'état du sol, va appeller *Etatparcelle()*, généré à partir du nouveau sous-programme ETATPARCELLE.

Retour conditionnel
: Pour un appel conditionnel unique, l'instruction conditionnelle est déplacée chez l'appelé, avec un retour immédiat si la condition est fausse, par exemple dans DEVELOP pour la valeur "`snu`" de CODEPLANTE.

[25] La non utilisation de la méthode d'évaluation de la densité racinaire vraie rend inutile l'allocation de la matrice de séries $Drl[366][200]$, définie pour l'ensemble des couches du sol.

Les modifications effectuées pour ne plus avoir à effectuer ces mises à jour liées à la vectorisation sont décrites avec les méthodes de la classe *Stics* (A.5.2), et signalées dans la section relative à la modularisation Fortran (A.3). Les nouveaux sous-programmes sont à ajouter dans la liste pour générer le code C (1.1.7).

Exécution des méthodes natives

L'application JAVA va chercher les méthodes natives par appel de la méthode

System.loadLibrary("bibStics") ;

qui recherche la bibliothèque partagée *bibStics* produite par le compilateur C, nommée "`libbibStics.so`" avec les conventions usuelles, le préfixe *lib* étant utilisé sous LINUX pour l'acccès aux bibliothèques.

Copie du code source C Le code source des fonctions C est recopié dans le répertoire *Native*, avec :

1. le fichier d'inclusion généré pour la classe des méthodes natives par la commande *Javah*, ici "`Stics.h`" avec la classe *Stics* ;
2. le code des fonctions C associée aux méthodes natives du répertoire *JavaC* ;
3. les fonctions générées à partir du code source Fortran :
   - le code C issu des unités Fortran, depuis le répertoire *C* ;
   - des fichiers sources et entête depuis le répertoire *D* :
     "`Tailles.h`" pour les tailles des vecteurs ;
     "`IniY.c`" pour gérer les vectorisations multiples ;
     "`SynchroRetro.c`", pour la gestion du mode retro pour les séries ;
     "`CultPrec.c`" pour l'acquisition du vecteur de la culture précédente ;
     "`CultSuiv.c`" pour la restitution du vecteur de la culture suivante ;
     "`ValCode.c`", restitution des codes textuels pour l'édition des bilans ;
4. des fonctions communes fabriquées, depuis le répertoire *Super* :
   - "`Commun.h`", déclarations des prototypes des fonctions communes ;
   - "`Intrin.c`", conversion des fonctions intrinsèques Fortran ;
   - "`Erreur.c`", gestion les déroutements Fortran ;
   - "`Synchro.c`", décalage des séries en mode retro ;
   - "`Date.c`", édition des dates pour les bilans ;

Déroutements Les fonctions C générées utilisent une fonction qui gère les déroutements, dont les fins d'exécution consécutives à des STOP Fortran, sous la forme

return Erreur(...) ;

après avoir noté un code d'erreur dans la composante de rang 0 du vecteur des évaluations quotidiennes (3.3.4).

Une solution pour éviter ce déroutement est de réviser le source "`Erreur.c`", en supprimant l'instruction *exit()*, le contrôle étant ensuite rendu à l'appelant.

Fabrication de la bibliothèque Le fichier "`makefile`" est mis à jour.

La bibliothèque est fabriquée par un "*make bibStics*" avec l'appel final

gcc -fPIC -shared -o libbibStics.so ...

l'option *PIC* du compilateur permettant de garantir la génération d'un code avec un adressage relatif (Position independant Code).

"`libbibStics.so`" est recopié dans le répertoire des bibliothèques *Lib*.

A.5.2 Architecture d'une application spatio-temporelle

Une classe *Culture* avec les vecteurs (K, S, V, M, W, X, Y) comme types de données et les méthodes natives issues du logiciel *Stics* permet finalement de piloter des simulations sur un territoire pour un ensemble de campagnes :

Les vecteurs (K, S, V, M, W, X, Y) sont suffisants pour restituer les états quotidiens d'un territoire cultivé sur longue période :

1. K est invariant pour l'ensemble des campagnes et parcelles mais peut varier dans l'espace géographique et le temps ;
2. S_i est le vecteur descriptif de la parcelle i d'un territoire, ensemble de parcelles avec un climat homogène ;
3. les rotations des cultures requièrent l'acquisition pour une nouvelle campagne n des vecteurs :
   - V_n pour les paramètres de la plante cultivée ;
   - W_n pour un itinéraire technique applicable ;
   - M_n pour le climat de la campagne culturale ;

Pour des besoins de pilotage et de couplage, ou encore pour utiliser d'autres méthodes, natives ou codées en JAVA, il est nécessaire d'accéder aux formalismes du modèle *Stics* : une classe est définie avec le dictionnaire des variables *Stics*.

Classe Stics

Les méthodes de la classe *Stics* reprennent l'architecture du logiciel *Stics*, en reprenant les modules appelés par des sous-programmes, et en regroupant des séquences relevant d'autres modules (comme indiqué dans les commentaires Fortran) dans de nouveaux sous-programmes.

Les lectures des fichiers WINSTICS n'étant pas reprises, les méthodes sont actuellement les suivantes.[26]

Initialisation

- *iniVariables()* initialise les variables du modèle lues avec les paramètres et données initiales, par appel de la fonction *IniY()* (3.2.4) ;
- *iniParcelle()* initialise la parcelle ; un fichier compte-rendu est ouvert pour les fonctions appelées :
  Initsimul(), Initsol1(), Initsol2(), Initnonsol(), Initcycle() ;

[26] La première lettre d'une méthode est notée en minuscule, la majuscule étant réservée pour les classes. La première lettre d'une fonction *Stics* appelée reste majuscule.

- *iniCampagne()* appelle *Iniclim()*, ancien sous-programme qui :
  - établit la relation entre jours calendaires et rangs des simulations ;
  - initialise un calendrier de campagne dans l'espace de la simulation ;

  le fichier compte-rendu est réouvert en mode ajout ;

Evaluations quotidiennes

1. *evalDeveloppement()* reprend l'appel de *Develop()* depuis *Laidev()*, avec retour immédiat de *Develop()* si (*Codeplante* = 'snu') ;
2. *evalPhotosynthese()* appelle *Photosynthese()*, issue du sous-programme homonyme qui reprend une séquence de *Laidev()*, avec retour immédiat si (*Codeplante* = 'snu'), et les appels conditionnels
   - *Calai()* puis *Effeuill()* si (*Codelaitr* = 1),
   - *Tauxrecouv()* sinon ;
3. *evalCroissance()* appelle *Croissance()*, issue du sous-programme d'évaluation des croissances aérienne et racinaire ;
4. *evalEtatParcelle()* appelle *Etatparcelle()*, issue du sous-programme homonyme qui reprend une séquence entre les appels de *Physio()* et *Apports()*, laquelle traite de l'état du sol, au semis ou à la récolte ;
5. *evalApports* appelle *Apports()*, lequel :
   - évalue les apports en eau avec *Calcprecip()* qui appelle *Irrig* pour ajouter l'irrigation du jour ;
   - traite des apports d'engrais mineral par appel de *Fertil()* ;
   - gère lea appors d'engrais organique par appel de *Residus()* ;
6. *evalOrganique()* appelle *Etatsurf()* pour traiter le mulch en surface, *Mineral()* pour calculer la minéralisation de la matière organique ;
7. *evalBesoinEau()* appelle *Beaupl()*, avec les appels conditionnels de *Shutwall()* ou *Ketp()* en fonction de l'option *Codebeso* ;
8. *evalAzoteEau()* traite
   - des offres et demandes d'azote : *Abson(), Bnpl(), Majnsol(), Ngrain(), Offren(), Offrnodu(),Stressn()* ;
   - de l'eau dans le sol :*Lixiv(), Excesdeau(), Transpi(), Stresseau()* ;
9. *evalChaleur()* reprend le module microclimat, avec appels :
   - *Caltcult()* pour la température de la culture ;
   - *Tempsol()* pour la température du sol ;
   - *Humcouv()* pour l'humidité du couvert végétal ;
10. *evalCoupe()* appelle *Coupe()* pour les cultures fauchées ;
11. *evalFinal()* reprend les appels de fin de boucle quotidienne ;

Synchronisation quotidienne

- *decalerRetro()* appelle la fonction générée *SynchroRetro()* pour décaler d'une journée les périodes pour les séries gérées en mode retro ;

Bilans des campagnes

- *ediBilan()* gère le fichier de restitution des résultats et appelle *Bilan()* ;

Rotation des cultures

- *finCampagne()* met à jour les vecteurs X et Y pour la campagne suivante par appels des fonctions générées :
  - *CultSuiv()* qui copie dans un vecteur *rota* les valeurs éditées par le sous-programme RECUP dans le fichier "`recup.tmp`" ;
  - *IniY()* qui recopie dans Y, après remise à zéro de Y, les valeurs lues avec les paramètres et données initiales ;
  - *CultPrec()* qui réaffecte dans X et Y les valeurs du vecteur *rota* ;

La construction de ces méthodes est à effectuer en amont au moyen d'un code Fortran modularisé, avec des appels de sous-programmes qui isolent les fonctions (et gèrent le *mulTIdisciplinaire* de l'acronyme $sTIcs$).

Classe Dictionnaire Stics

La classe *DicoStics* offre la possibilité d'intervenir sur la simulation du jour avec les les formalismes du modèle *Stics*, en nommant les variables *Stics*.

L'application *Stics*0, comme elle édite les vecteurs, copie dans le fichier "`Stics.dic`" une description du dictionnaire utilisé, ensuite lue par l'application JAVA pour reconstruire ce dictionnaire.

Des méthodes sont définies pour gérer ce dictionnaire accéder aux variables *Stics* en les désignant par leurs noms, et identifier les composantes du vecteur qui contient leus valeurs. Elles permettent de consulter les valeurs de ces variables et, éventuellement, de les modifier.

Classe Territoire

La classe $Territoire$ contient la classe $Culture$ et les classes constructrices des vecteues, soit :

1. *contexte* pour les constantes du territoire et des campagnes, vecteur K ;
2. *Parcelle* pour le sol, vecteur S ;
3. *Sole* pour la plante, vecteur V ;
4. *Technique* pour l'itinéraire technique de campagne, vecteur W ;
5. *Climat* pour la série climatique de campagne , vecteur M ;
6. *EtatParcelle* pour l'état de la parcelle en début de campagne, vecteur X ;

Classe Vecteur

La classe $Vecteur$ permet de gérer des vecteurs de $\mathcal{R}^n$, avec la grammaire
$< nom > [< indice1 > - < indice2 >]$
pour la lecture et la copie dans des fichiers textes.

Classe Xappli

La classe *Xappli* gère les interfaces graphiques.

Classe JavaStics

La classe *JavaStics* pilote l'application.

A.5.3 Exécution d'une application objet

La simulation pour un territoire nécessite au préalable d'éditer le dictionnaire et les vecteurs pour la période initiale, pour les parcelles et les campagnes, ce qui peut être fait au moyen de l'application *Stics*0 pour une version C donnée.

Exécution classique

Vecteurs et dictionnaire étant accessibles, *JavaStics* peut s'exécuter de façon classique par appel de l'interpréteur JAVA, soit

java JavaStics

L'exécution apparaît alors comme relativement longue.

Compilation du source JAVA

Une autre solution est d'exécuter une version JAVA compilée, fabriquée par le compilateur *gcj*.

1. **Compilation**
   Il faut d'abord produire le "code objet" issu du source JAVA, soit

   gcj -fPIC -fjni JavaStics.java

   l'option *-fjni* étant nécessaire pour indiquer au compilateur qu'il faut utiliser l'interface native JNI (et non l'interface par défaut CNI).
2. **Edition de liens**
   Pour produire le code exécutable, il est nécessaire de préciser dans l'option *main* la classe qui contient la fonction *main*(), soit

   gcj JavaStics.o libbibStics.so -o javastics –main=JavaStics

Il suffit alors d'appeler la commande *javastics* depuis un répertoire contenant vecteurs et dictionnaire : l'exécution est alors immédiate.

Table des matières

1 Analyse syntaxique globale du code source Fortran **5**

1.1 Reconnaissance des instructions 5

1.1.1 Notations syntaxiques 6

1.1.2 Les "paquets de cartes" 6

1.1.3 Les éléments lexicaux 7

1.1.4 Production d'une forme normalisée 7

1.1.5 Forme d'une instruction 8

1.1.6 Unités de programmes et déclarations 9

1.1.7 Traduction informatique 11

1.1.8 Utilisation pour d'autres logiciels 13

1.2 Dictionnaire des variables communes 14

1.2.1 Déclarations des types des variables 14

1.2.2 Dimensions des variables 15

1.2.3 Communs 16

1.2.4 Pseudo-constantes 17

1.2.5 Description des variables communes 17

1.2.6 Traduction informatique 17

1.3 Structures de contrôle 18

1.3.1 Schéma conditionnel 19

1.3.2 Schéma itératif 22

1.3.3 Ruptures de séquences 27

1.3.4 Ruptures de séquences dans les sections critiques 28

1.3.5 Appels de sous-programmes 29

1.3.6 Entrées et sorties 29

1.3.7 Traduction informatique 32

1.4 Bilan de l'analyse syntaxique globale 32

2 Traitement des noms symboliques **34**

2.1 Description des variables 35

2.1.1 Affectation d'une valeur dans une variable 36

2.1.2 Transmission d'une variable en paramètre 38

2.1.3 Portée des variables et synonymies 39

2.1.4 Vecteurs et matrices 40

2.1.5 Variables communes non numériques 42

2.1.6 Constantes décimales 43
2.2 Description des fonctions externes 43
2.2.1 Fonctions intrinsèques Fortran 44
2.2.2 Déclaration des paramètres et fonctions 44
2.2.3 Appels des fonctions 46
2.2.4 Définition des paramètres par valeur ou référence 47
2.2.5 Prototypes de fonctions 48
2.3 Relations entre variables 50
2.3.1 Graphes des relations entre variables 51
2.3.2 Opération d'affectation 51
2.3.3 Variables transmises en paramètres 52
2.3.4 Modifications des paramètres transmis 53
2.3.5 Effet de bord 53
2.4 Classification des variables communes 54
2.4.1 Dynamique du modèle 55
2.4.2 Rôles des unités de programmes 57
2.4.3 Classement des variables lues 59
2.4.4 Classification des variables 62
2.5 Gestion des variables 66
2.5.1 Variables de simulation 66
2.5.2 Copies locales de variables communes 66
2.5.3 Initialisation de variables dynamiques 68
2.6 Gestion des couples de cultures 68
2.6.1 Variables de gestion des couples 68
2.6.2 Formes syntaxiques 69
2.6.3 Séparation des valeurs 70
2.6.4 Paramètres des unités de programmes 70
2.7 Traduction informatique 70
2.7.1 Codes textuels 73
2.7.2 Arbres des appels 73
2.7.3 Prototypes des fonctions 73
2.7.4 Utilisation des variables communes par fonctions 74
2.7.5 Dictionnaire des variables communes 75
2.7.6 Dictionnaire des variables locales 75
2.7.7 Lexiques des fonctions 76
2.7.8 Compte-rendu du traitement des noms symboliques . . . 76
2.7.9 Révision Fortran 79

3 Génération des fonctions **80**
3.1 Gestion des variables communes : forme simple 81
3.1.1 Paramètres informatiques communs 81
3.1.2 Allocation et restitution des variables communes 82
3.1.3 Validation des versions 84
3.2 Gestion des variables communes : métamodèle 85
3.2.1 Cultures associées 86
3.2.2 Métamodèle 88

3.2.3 Vectorisation : partition des variables 89
3.2.4 Vectorisations multiples 92
3.2.5 Vectorisation : ordonnancement des composantes 92
3.2.6 Traduction informatique 94
3.2.7 Gestion des dictionnaires des variables et fonctions 96
3.2.8 Forme vectorielle pour les applications objets 96
3.3 Traitement des fonctions 97
3.3.1 Déclaration des fonctions 97
3.3.2 Déclaration et initialisation des variables locales 99
3.3.3 Retour vers la fonction appelante 100
3.3.4 Déroutement 100
3.3.5 Appel 101
3.3.6 Fonctions intrinsèques 102
3.4 Désignation des variables et constantes 103
3.4.1 Désignation des variables 103
3.4.2 Représentation des nombres 104
3.4.3 Chaînes de caractères 104
3.5 Expressions arithmétiques et logiques 104
3.5.1 Expressions arithmétiques 105
3.5.2 Expressions entières 106
3.5.3 Expressions logiques 106
3.5.4 Expressions textuelles 108
3.6 Evaluation des variables 109
3.6.1 Affectations directes 109
3.6.2 Transmissions en paramètres 110
3.6.3 Gestion locale des variables communes 110
3.7 Contrôle de l'exécution 111
3.7.1 Itération à pas constant 111
3.7.2 Instruction conditionnelle 114
3.7.3 Ruptures de séquences 116
3.7.4 Choix multiple 118
3.7.5 Affectations statiques 118
3.8 Les entrées et sorties 118
3.8.1 Formats des enregistrements 119
3.8.2 Listes des entrées-sorties 120
3.8.3 Edition des messages d'erreurs 122
3.8.4 Fonctions de lecture des fichiers STICS 123
3.8.5 Numérisation des variables textuelles 126
3.8.6 Déroutements en lecture 127
3.8.7 Traduction des instructions d'édition 127
3.9 Production du code 129
3.9.1 Génération du code C et Fortran 130
3.9.2 Code des fonctions C 133
3.9.3 Compte-rendu 133
3.10 Dictionnaire des variables STICS 135
3.10.1 Désignation locale de variables communes 135

3.10.2 Ajustement des espaces d'évaluation 136
3.10.3 Gestion des périodes antérieures et durées de vie 138
3.10.4 Non vectorisation 139

4 Restructuration et assemblage des fonctions C 140
4.1 Généralités 141
4.1.1 Supervision 142
4.1.2 Précision des calculs et propagation d'erreurs 143
4.1.3 Déclaration des fonctions 145
4.1.4 Optimisation 146
4.2 Réalisation du module de supervision 147
4.2.1 Fonctions communes 147
4.2.2 Lecture des paramètres et valeurs initiales 149
4.2.3 Editions en cours de simulation 150
4.2.4 Fonctions d'accès aux variables communes vectorisées . . 151
4.2.5 Fonction de supervision 152
4.2.6 Sauvegarde et acquisition des vecteurs 157
4.2.7 Génération logicielle 159
4.3 Lecture des données externes 161
4.3.1 Pilotage de la lecture 163
4.3.2 Paramètres initiaux 165
4.3.3 Paramètres généraux 167
4.3.4 Paramètres "nouvelle version" 169
4.3.5 Paramètres techniques 170
4.3.6 Paramètres de la plante 173
4.3.7 Paramètres climatiques 175
4.3.8 Paramètres du sol 178
4.3.9 Culture précédente 179
4.3.10 Autres lectures 182
4.4 Evaluation des conditions initiales 183
4.4.1 Initialisations préliminaires 183
4.4.2 Initialisation de la culture 184
4.4.3 Initialisation de la campagne culturale 186
4.4.4 Autres initialisations 188
4.4.5 Validation de l'état initial 189
4.5 Simulations quotidiennes 190
4.5.1 Intégration des cultures associées dans la version 6 192
4.5.2 Initialisations dans la boucle quotidienne 194
4.5.3 Stades de développement 196
4.5.4 Absorption de l'énergie solaire 198
4.5.5 Réduction de la surface foliaire par effeuillage 199
4.5.6 Mobilisation des réserves de la plante 199
4.5.7 Sénescence de la matière sèche 201
4.5.8 Production des graines ou fruits 202
4.5.9 Croissance racinaire 204
4.5.10 Tassement du sol 206

4.5.11 Apports d'eau . . . 207
4.5.12 Apports d'azote et de matières organiques . . . 208
4.5.13 Etat de surface du sol . . . 210
4.5.14 Bilan hydrique . . . 210
4.5.15 Bilan azoté . . . 213
4.5.16 Transferts d'eau et de nitrates dans le sol . . . 215
4.5.17 Etat final quotidien de la culture . . . 217
4.5.18 Gestion des coupes des cultures fourragères . . . 219
4.6 Validation des états quotidiens . . . 220
4.7 Restitution finale des résultats . . . 220

A Fabrication des applications 222
A.1 Environnement de développement . . . 223
A.1.1 Hiérarchie des répertoires . . . 223
A.1.2 Génération des fonctions C à partir du code Fortran . . . 225
A.1.3 Fabrication et mise au point des applications . . . 226
A.2 Application Fortran source . . . 228
A.2.1 Choix des unités de programmes . . . 228
A.2.2 Répertoire de simulation de la culture . . . 228
A.2.3 Accès aux fichiers et formats . . . 229
A.2.4 Variables locales non initialisées . . . 230
A.2.5 Formes syntaxiques particulières . . . 230
A.3 Application Fortran révisée . . . 230
A.3.1 Modularisation minimale . . . 231
A.3.2 Restructuration minimale . . . 236
A.3.3 Ambiguïtés et anomalies de programmation . . . 237
A.3.4 Variantes du langage Fortran . . . 238
A.3.5 Déclaration des variables communes . . . 239
A.3.6 Cultures associées . . . 240
A.3.7 Autres . . . 240
A.3.8 Fabrication de l'application Fortran révisée . . . 241
A.4 Applications C . . . 241
A.4.1 Génération automatisée des fonctions . . . 241
A.4.2 Fabrication logicielle . . . 242
A.4.3 Validation . . . 243
A.5 Applications objets . . . 245
A.5.1 Fonctions vectorielles Stics et méthodes natives . . . 245
A.5.2 Architecture d'une application spatio-temporelle . . . 248
A.5.3 Exécution d'une application objet . . . 251