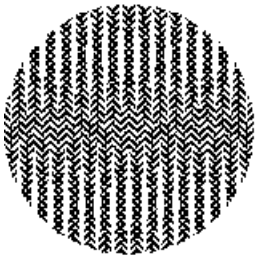

Institut National de la Recherche Agronomique

Département

Projet de plateforme « sol virtuel »

Conception informatique générale

Version 2010

Auteurs : Nicolas Moitrier[1] et Nathalie Moitrier[2]

Relecteurs : les membres du groupe projet « sol virtuel »[3]

Dernière modification : 22 juin 2010

1 Chef de projet informatique de la plateforme « sol virtuel » <nicolas.moitrier@avignon.inra.fr>

2 Ingénieur en développement informatique de la plateforme « sol virtuel » <nathalie.moitrier@avignon.inra.fr>

3 Alain Mollier <mollier@bordeaux.inra.fr>, Anatja Samouelian <Anatja.Samouelian@orleans.inra.fr>, François Lafolie <lafolie@avignon.inra.fr>, Isabelle Cousin <Isabelle.Cousin@orleans.inra.fr>, Valérie Pot <vpot@grignon.inra.fr>

Date	auteur	remarques
01/05/2009	Nicolas Moitrier	Version initiale
18/06/2009	Nathalie Moitrier Nicolas Moitrier	Scission en 2 documents : séparation des concepts généraux, de ceux techniques
30/06/2009	Nicolas Moitrier	Insertion de la licence Pastis
13/07/2009	Nicolas Moitrier	Mise à jour des exigences logicielles
17/07/2009	Nathalie Moitrier	Correction des chapitres 3.2 et 3.3
21/09/2009	Nathalie Moitrier	Revue générale
28/09/2009	Nicolas Moitrier	Inspiration Stics (dans l'introduction) + ajouts en 3.2.5 Réorganisation des sections et ajouts dans le chapitre 3
23/09/2009	Nicolas Moitrier	Ajout du chapitre 4.3
02/11/2009	Nicolas Moitrier	Précisions sur les données nominatives, au chapitre 5.3.1
08/12/2009	Nicolas Moitrier	Ajout de la section 4.1.2.8, « Complexité et qualité du code source »
08/01/2010	Nicolas Moitrier	Prise en compte des remarques de Philippe Clastre
18/01/2010	Nicolas puis Nathalie Moitrier	Refonte et réécriture du 4.2 + 4.2.{1,2,3}
24/02/2010	Nicolas Moitrier	Réorganisation générale des chapitres suite au retours de GRID Ajout de la section 3.5 sur la distribution des logiciels
02/04/2010	Nathalie Moitrier	Relecture finale Regroupement des sections 2.1.3 + 2.1.7 + partie contenue 2.1.2 Ajout de la section 2.1.3 Déplacement des sections 3.2 et 3.3 dans conception de modèles Chapitre 3 : recentrage sur les composants/parties.
12/04/2010	Nicolas Moitrier	Relecture finale Fusion des sections 2.1.2 et 2.1.3
22/04/2010	Nicolas Moitrier	Reconstruction de l'illustration 3
10/05/2010	Nicolas Moitrier	Prise en compte des corrections et remarques du groupe projet
14/06/2010	Nathalie puis Nicolas Moitrier	Relecture finale
22/06/2010	Groupe projet	Relecture finale

Tableau 1: suivi des modifications

Table des matières

Index des illustrations

Index des tables

1 Introduction

Ce document est la suite logique du document d'analyse informatique du cahier des charges fonctionnel, intitulé « Analyse informatique du cahier des charges », du 4 novembre 2009. L'analyse fonctionnelle -qui n'est pas le sujet de ce document- consiste à définir et valider le « quoi ». Le présent document de conception informatique exprime ce qui est de l'ordre du « comment ». A partir de cette phase du projet -représentée par ce document de conception- il est nécessaire de faire nettement la distinction entre l'impératif de recherche scientifique -synonyme de besoin de temps, de remises en question, de perceptions différentes- et l'impératif de production informatique -synonyme de besoin de certitudes, d'échéances, de prévisions et de spécifications stables. C'est donc à l'instar du document d'analyse informatique (section 2.3 « Une démarche souple ») que la conception informatique s'inscrit dans une démarche de développement informatique souple, via une méthode agile. Elle accepte donc la nécessité de progresser petit à petit, par essai/erreur. La conception se veut donc itérative, dans laquelle aucune définition des besoins ou réalisations n'est définitivement acquise. Ce document peut donc devenir obsolète -au bout d'une ou deux années- puisqu'il s'agit de procéder par touches successives, de s'autoriser des corrections et des re-cadrages sur les réalisations. A titre d'exemple, les spécifications du système de couplage pourraient être suranné dans l'avenir. Ce pourrait être le cas notamment si l'adjonction de nouveaux processus-modules impose des formalismes de couplage indisponibles avec le premier système choisi. La conception informatique devrait alors évoluer pour s'adapter à ces changements, autant imprévisibles que « perturbateurs ».

Ce document exprime uniquement l'architecture générale de la plateforme « sol virtuel » et les concepts généraux. Les aspects techniques détaillés seront exprimés dans le document de conception détaillé. L'objectif de séparer ces deux niveaux « d'abstraction » est de permettre la diffusion de chaque document à un public bien ciblé.

Le public visé par ce document est principalement composé d'informaticiens en développement. C'est à dire les rôles « informaticiens de plateforme » et « informaticiens de modules ». Indirectement, les « scientifiques de modules » sont aussi concernés. Enfin, le groupe projet « sol virtuel » se doit valider son contenu.

Comme l'analyse informatique de ce projet, la conception informatique se préoccupe à la fois du niveau plateforme -ou infrastructure- et du niveau module. Le premier niveau plateforme, donc dédié aux informaticiens de plateforme, est la partie qui permettra l'articulation et le couplage des modules. Le niveau module, dédié aux informaticiens de modules, s'occupe de la mise en place des modules et de leur adaptation pour leur intégration dans la plateforme. De par l'importance stratégique -puisque directement en relation avec les utilisateurs du projet- de cette deuxième partie, le focus sera mis sur les étapes de conception détaillée, de codage dans un langage informatique et les tests unitaires pour le développement des modules.

A ce stade du projet, un rapide sondage permet de détecter qu'un grand nombre des codes informatiques des modèles existants pourraient être utiles -et même indispensables- à « sol virtuel ». En effet, ces codes informatiques remplissent les fonctions de nombreux processus déjà identifiés dans « sol virtuel ». Cependant, outre leur adaptation à l'infrastructure informatique de « sol virtuel », il apparait clairement nécessaire de procéder à une étape préalable de modularisation. Cela consiste à rendre le plus indépendants possibles les différentes parties des modèles existants.

2 Contraintes et exigences

Le choix de solutions techniques doit tenir compte des attentes, des contraintes et des exigences, à la fois fonctionnelles et non fonctionnelles C'est le sujet du présent chapitre.

2.1 Retour sur le cahier des charges

Les besoins non fonctionnels exprimés dans le cahier des charges fonctionnels doivent être interprétés comme des contraintes. Ces besoins ont été rediscutés, précisés et arbitrés dans la réunion du 24 mars 2009, entre le groupe projet et l'équipe informatique. Cette section du document de conception informatique revient sur les points abordés lors de cette réunion et expose les choix -techniques ou architecturaux- que cela va imposer.

2.1.1 Accessibilité

Deux types d'accès à la plateforme peuvent être proposés aux utilisateurs :

1. l'accès local
2. l'accès déporté, total ou partiel.

L'accès local ou client lourd est un logiciel qui peut proposer des fonctionnalités complexes avec un traitement autonome, sur le poste de l'utilisateur. Contrairement au client léger (accès déporté), le client lourd ne dépend pas d'un quelconque serveur pour fonctionner. Il peut éventuellement s'y connecter pour l'échange de données, dont il prend généralement en charge l'intégralité du traitement. C'est donc une solution « classique », qui permet d'avoir une interface graphique riche et réactive. Évidemment, elle ne dépend pas de l'existence d'un réseau informatique, sauf pour télécharger des mises à jour[4]. Le client lourd est généralement plus facile à développer et déboguer. Mais il est aussi caractérisé comme étant couteux au niveau de la maintenance et du déploiement/installation.

L'accès déporté ou client léger -au sens logiciel du terme- est une application logicielle laissant quasiment tout le traitement au serveur. L'exemple le plus courant du logiciel client léger est le navigateur Web[5]. Mais ce peut être n'importe quelle application Internet : messagerie, chat, visio-conférence... Son principal intérêt est la facilité de mise en œuvre du déploiement. En effet, par nature il élimine quasiment le besoin d'installer un logiciel client sur les machines des utilisateurs. Éliminant du même coup le problème lié aux différentes configurations matérielles (PC, Mac...) et logicielles (Windows, Linux...) qui sont la bête noire des informaticiens. Les mises à jour peuvent être automatiques et transparentes pour l'utilisateur[6]. Malheureusement pour les développeurs informatiques, la conception d'applications en mode client léger -notamment celles purement Web- est complexe et leur maintenance corrective difficile. De plus les technologies liées aux clients léger sont relativement jeunes, parfois immatures et manquant de documentations ou de communautés d'utilisateurs.

La réunion du 24 mars 2009 entre le groupe projet et l'équipe informatique donne un

4 OpenOffice.org impose de télécharger l'intégralité de l'application à chaque nouvelle version, même mineure.
5 Le logiciel « client Web » sur le poste de travail de l'utilisateur ne fait que de l'affichage. C'est le serveur qui calcule les pages HTML à présenter.
6 Les logiciels de la fondation Mozilla -Firefox et Thunderbird- fonctionnent de cette manière.

arbitrage en faveur de l'approche « client lourd », pour les premières versions de la plateforme. C'est une position prudente qui n'empêche pas l'évolution progressive vers du client léger, si une forte demande se fait sentir de la part des utilisateurs. L'objectif est de privilégier la rapidité de réalisation informatique. L'accessibilité de la plateforme se fera donc localement, la plateforme sera installée complètement sur le poste de travail de l'utilisateur.

Conséquences :

1. L'utilisateur aura à sa charge (en plus de l'installation initiale) de mettre à jour sa version autonome à chaque nouvelle version diffusée de la plateforme.
2. Les composantes logicielles (langages et librairies) de la plateforme doivent être multi-plateforme (au sens du système d'exploitation).

2.1.2 Autonomie et intégration de modules

Il a été exprimé que la plateforme doit être accessible aux acteurs « informaticiens de modules ». Ceux-ci devraient être essentiellement des scientifiques. Ils doivent être autonomes dans l'utilisation de la plateforme. En particulier, il est important que les scientifiques puissent mettre des modules dans la plateforme sans avoir recours à l'aide des informaticiens.

Les acteurs « informaticiens de modules » doivent pouvoir intégrer eux mêmes, dans la plateforme, aussi bien des codes existants que des codes nouveaux. L'implémentation du code source informatique des modules au sein de la plateforme peut être réalisée de deux façons différentes :

- écriture directe du code source dans l'interface graphique dédiée de la plateforme ;
- insertion du code source existant à partir de fichiers.

Dans les deux cas, une grande vigilance s'impose. Le but est de ne pas perturber la fiabilité de l'ensemble. Dans ce but, il faut fournir :

- une structure prédéfinies pour les modules, en fonction du langage de programmation ;
- des normes de programmation ;
- des jeux de tests de non-régression[7].

L'autonomie dans la modification/création de modules ne doit pas isoler les scientifiques et faire disparaitre une finalité forte de la plateforme : le partage et la valorisation. Cette autonomie ne doit cependant pas être un frein aux retours vers la communauté scientifique. C'est à dire que les utilisateurs fournissant des modules doivent « jouer le jeu » en partageant leur production. Une trop grande autonomie peut engendrer un risque de ce point de vue. Une attention toute particulière sera portée pour ne pas rendre le code informatique des modules trop dépendants de la plateforme (voir section 2.1.2.2 , Double vie du code source).

7 Un jeu de test de non-régression consiste à tester qu'une partie d'un code informatique, préalablement testée lors des tests unitaire, fourni toujours le même résultat. Après une modification d'une partie du code, il permet de s'assurer qu'aucun défaut n'a été introduit dans les parties non modifiées.

2.1.2.1 Code fourni et code généré

Dans le but d'être autonome dans la conception de modules compatibles « sol virtuel », le groupe projet demande à ce que la plateforme génère automatiquement les prototypes des fonctions informatiques et tout le code nécessaire à la gestion des entrées/sorties. Les entrées/sorties sont des variables des fonctions, passées en paramètre de celles-ci. La génération se fait à partir :

1. du choix du processus auquel se rattache le module ;
2. des variables en entrée/sortie choisies pour ce module.

Pour illustration, prenons le processus « P1 » choisi par l'utilisateur. Ce processus déclare deux entrées ('a' et 'b') et peut fournir deux sorties ('A' et 'B'). L'utilisateur choisit que son module -représentant tout ou partie de ce processus- s'appelle « M1P1 », qu'il utilise une des deux entrées du processus 'a' et qu'il fournit une des deux sorties 'A' déclarée par ce même processus. Nous avons donc un prototype de fonction qui en langage « naturel » pourrait s'écrire comme suit :

```
fonction M1P1 = "a: entrée, type décimal" + "A: sortie, type décimal"
```

En langage informatique C/C++, cela pourrait s'écrire :

```
void M1P1( const float a , float & A );                    // M1 réalise P1
```

L'utilisateur pourra alors insérer son code informatique (C/C++/Fortran) librement dans le corps de la fonction « M1P1 ». Il aura accès à toutes les entrées qu'il aura déclarées pour le module -en lecture seule- et il devra renseigner toutes les sorties qu'il aura déclarées. La plateforme s'occupera de fournir les entrées demandées et pourra en retour vérifier que toutes les sorties déclarées sont bien renseignées.

Dans ce scénario, l'utilisateur est totalement responsable de la gestion mémoire et des entrées/sorties fichiers de son module. Il aura toutefois la possibilité pour son module d'utiliser des variables d'état dont les valeurs seront conservées d'un pas de temps à l'autre, donc entre chaque appel de fonction.

2.1.2.2 Double vie du code source

Créer ou intégrer un code source informatique dans la plateforme, c'est disposer d'un environnement qui aidera à son utilisation : gestion du temps, couplage avec d'autres modules,, interface utilisateur graphique pour permettre de le paramétrer...

Bien que la réalisation informatique de la plateforme soit le principal but de l'équipe projet, nous devons avoir conscience que son succès sera aussi lié à la souplesse d'utilisation des codes produits dans et hors la plateforme. Les codes informatiques des modules devront donc être exploitables hors de la plateforme. Cette forte contrainte pour l'équipe, permettra entre autre de limiter le nombre de versions différentes des codes informatiques des modules créés ou intégrés, et donc de bénéficier plus facilement des divers améliorations et évolutions de ces codes. De part sa relative indépendance la version du code informatique « hors plateforme » peut alors évoluer et s'améliorer plus vite que la version « dans la plateforme ».

Cette « bascule » doit être la plus facile et rapide possible pour les « informaticiens de modules ». La plateforme « sol virtuel » doit donc permettre cette « double vie ». Sous réserve de respecter la structure proposée pour la modularisation d'un code source dans la plateforme,

l'utilisateur doit pouvoir librement toucher son code et le réinsérer quand bon lui semble. Devant la contrainte importante -par rapport aux aspects techniques- que constitue une telle tâche, la première version de la plateforme intègrera uniquement la possibilité d'écriture directe du code source dans une interface graphique dédiée. Mais rapidement, elle doit permettre l'insertion de codes sources existants et leur réinsertion de manière ad-hoc.

2.1.3 Langue de la plateforme

Dans le but de faciliter la diffusion de la plateforme au niveau international, la langue des logiciels informatiques de la plateforme sera l'anglais. Cela concerne tous les développements informatiques -infrastructure comme modules- de la plateforme et leurs commentaires, en résumé tous les codes sources et leurs dérivés possibles.

2.1.4 Maintenance logicielle

Les composants logiciels constituant la plateforme doivent pouvoir évoluer et être maintenus par les acteurs informaticiens de plateforme.

2.1.5 Support logiciel

Le support logiciel consiste à apporter un soutien technique et scientifique dans l'utilisation de la plateforme auprès des utilisateurs. Les tâches que cela engendre dépendent du niveau de support souhaité. A minima et obligatoirement, cela engendre des tâches de documentations et de tests sur l'ensemble des composants de la plateforme (manuels d'utilisations des différents composants, documentations scientifique de tous les modules présents dans la plateforme, documentation des modèles disponibles, etc).

Le premier niveau de support, qui sera mis en place dès la diffusion de la première version de la plateforme, est le support pour l'utilisation des différents composants. L'intervention des membres de l'équipe projet et des concepteurs de modules sur ces tâches de support sera au départ restreinte à la production d'informations accessibles via différents canaux sur le site Web du projet : documentations, questions/réponses types FAQ, forum de discussions...

Par la suite, deux niveaux de supports supplémentaires pourront être mises en place :

- Une offre de fonctions et méthodes logicielles utiles à un usage scientifique : fonctions mathématiques et statistiques, méthodes d'estimation et d'optimisation de paramètres, analyse de sensibilité aux entrées des modules...
- La prise en charge de l'insertion de nouveaux modules et leur maintenance.

En terme de moyens humains et des compétences nécessaires -scientifiques, mathématique autant qu'informatiques- cela ne peut être envisagé que lorsque l'infrastructure de la plateforme sera pleinement opérationnelle. Le niveau de support sera défini ultérieurement lorsque les premières maquettes informatiques seront proposées et que les moyens mis en œuvre -notamment techniques et humains- stabilisés.

2.1.6 Interaction avec d'autres plateformes

Le département EA a initié des plateformes scientifiques -à fortes composantes

informatiques- dans des domaines connexes. Lors de discussions avec le groupe projet, la question du positionnement de « sol virtuel » par rapport aux autres plateformes a été posée.

Le groupe projet indique que l'objet d'étude « sol virtuel » doit être vu comme une brique élémentaire par les autres plateformes. La plateforme « sol virtuel » est naturellement une « cellule » d'une plateforme comme « paysage virtuel ». C'est à dire que « sol virtuel » pourrait fournir des modèles de fonctionnement local servant dans la plateforme « paysage virtuel ».

Il en va de même pour « Record » et une éventuelle « plante virtuelle ». Plus précisément, le couplage entre « sol virtuel » et « Record » peut être de fournir, pour la première, un module/modèle à la deuxième, que cela soit sous forme d'exécutable ou de code source informatique. L'inverse est aussi possible. Record pourrais fournir des modules comme : simulation d'itinéraires techniques ou de croissance de plantes. La plateforme « sol virtuel » adoptera donc un niveau de réutilisation modules/modèles mais avec un niveau d'exigence d'échanges (cf. 2.2.3 Exigence d'échanges).

Une des sorties de la plateforme « sol virtuel » sera le code source des modules et modèles. L'objectif est de pouvoir modifier directement « dans le code », pour réaliser les couplages voulus et ainsi faciliter l'intégration dans d'autres plateformes ayant des choix techniques déjà réalisés. Pour les modèles -qui sont des « agencements de modules »- cela permettra d'accéder à la gestion des boucles temporelles et spatiales. La plateforme d'accueil pourra alors à loisir modifier la gestion de ces boucles ou tout simplement les remplacer par celles natives.

Ce sujet de l'interaction de « sol virtuel » avec d'autres plateformes est le sujet de la section 2.2 Niveaux d'exigences vis-à-vis d'autres plateformes, ci-dessous. Il précise, notamment les contraintes qu'exigeraient un bon niveau d'interopérabilité avec une plateforme donnée.

2.2 Niveaux d'exigences vis-à-vis d'autres plateformes

L'interaction avec d'autres plateformes impose de se donner un certain niveau d'exigence quant aux échanges possibles entre les deux parties, notamment d'un point de vue informatique. De ce niveau découle directement la facilité des échanges et dans le même temps des contraintes/exigences qui s'intensifient.

2.2.1 Exigence d'interopérabilité

L'exigence d'interopérabilité est le niveau le plus fort. Il consiste à demander une capacité -pour un produit donné- à fonctionner avec d'autres produits, sans restriction de mise en œuvre. Pour cela, la finalité du produit (ici logiciel) doit être connue et la mise en œuvre de son interopérabilité spécifiée. Cette dernière exige la connaissance intégrale de l'ensemble des interfaces du produit.

Dans notre cas, il s'agit de pouvoir s'échanger des modules ou des modèles entre deux plateformes, avec un effort d'adaptation nul ou très faible. Un modèle produit par une plateforme pouvant alors devenir un module pour l'autre, les exigences peuvent aller jusqu'à l'utilisation du même système de couplage pour les plateformes concernées, surtout s'il s'agit de modules.

2.2.2 Exigence de compatibilité

L'exigence de compatibilité est moins contraignante que celle d'interopérabilité. Elle impose

une capacité -pour deux produits- à pouvoir fonctionner ensemble, sans présupposer de l'effort d'adaptation ou des restrictions de mise en œuvre. Plus l'effort à fournir et les restrictions seront faibles, meilleur sera le degré de compatibilité entre les plateformes.

Ce niveau peut être suffisant pour l'échange de modèles entre « sol virtuel » et une autre plateforme. Il est probablement moins adapté pour la réutilisation des modules. En effet, un modèle de « sol virtuel » peut-être un modèle « encapsulé » dans une forme informatique compatible pour une autre plateforme. Cela est plus difficile à envisager à l'échelle d'un module. Notamment parce le formalisme de module « sol virtuel » peut-être très éloigné -et même complètement incompatible- avec une plateforme qui aurait fait d'autres choix.

2.2.3 Exigence d'échanges

L'exigence d'échanges est la moins contraignante des exigences. Elle suppose que les deux plateformes entrent en synergie pour favoriser les échanges technico-informatiques :

- bonnes pratiques et démarche qualité ;
- méthodes, librairies et outils de développement ;
- algorithmes ou codes de modules ;
- toutes autres réalisations informatiques ;
- comptabilité de licence (ou droits).

2.3 Développement durable

Dans tout développement informatique se pose la question de la durabilité du code informatique. Le développement de logiciels durables s'oppose au jetable. Il est un objectif à atteindre.

Le développement jetable est souvent utilisé « de fait ». Il répond efficacement à une contrainte souvent principale : obtenir rapidement et simplement un résultat logiciel. La ré-utilisabilité et le partage avec d'autres sont secondaires.

Le modèle durable part du principe qu'une production logiciel peut toujours resservir plus tard. La fabrication doit donc être pérenne. Pour cela, elle doit être la plus tolérante et la plus indépendante possible face à l'évolution des besoins et des progrès techniques. Une attention toute particulière doit être donc placée sur **la documentation et la traçabilité des changements opérés**. La durabilité d'une production logicielle n'est pas exclusivement liée à celle de son code informatique. Il est important de séparer la connaissance scientifique de sa « formalisation » informatique. Les descriptions « mathématiques, numériques, algorithmiques » peuvent très bien assurer cette pérennité et sont -par nature- plus tolérantes aux changements.

La qualité des codes informatiques des modules importés dans la plateforme « sol virtuel » pourrait être très inégale. La plateforme devra donc veiller à ce que l'insertion ou les changements dans un module ne puissent pas remettre en cause la stabilité -au moins informatique- de l'ensemble. Cela est d'autant plus vrai si les codes sources ont une « double vie » (voir section 2.1.2.2 Double vie du code source). L'audit, c'est à dire la détection de nombreuses anomalies ou mauvaises -dans le sens de dangereuses- pratiques de programmation, automatique des modules est donc indispensable. Au minimum, les compilateurs doivent être activés avec les options d'alertes au

maximum de leur possibilités[8]. Le document de conception détaillée proposera des pratiques utiles dans ce sens.

2.4 Aspects de la qualité

Un logiciel est dit « de qualité » s'il répond correctement aux attentes spécifiées. La plateforme se veut un outil favorisant la mutualisation et les échanges, tout en garantissant à tous les profils d'utilisateurs (définis) une qualité en termes de fiabilité et de robustesse des données produites. Cette notion de partage -donc d'ouverture- engendre des choix dans la manière de concevoir les composants logiciels. En effet pour garantir une cohérence de l'ensemble, il est indispensable de définir un cadre -dans le sens de pratiques de programmation informatiques- clair, cohérent dans la phase de réalisation informatique.

Un certain nombre de règles/concepts sont définis dans le cadre de « sol virtuel » au travers du document de conception détaillée. L'objectif est de donner les mêmes règles et usages à tous les acteurs -intervenant sur la conception de l'infrastructure ou de la création de modules.

Dans notre contexte l'accent sera mis sur :

- l'algorithme ;
- la maitrise des outils utilisés ;
- la gestion des erreurs ;
- la réalisation de tests ;
- la traçabilité des opérations ;
- la réalisation et maintenance de la documentation ;
- la qualité et la complexité du code source.

2.4.1 L'algorithme

Un programme informatique est constitué d'un savoir traduit en un langage machine. La séparation entre le concept et le code est une garantie de robustesse et de facilité dans les opérations de maintenance et d'évolution de la forme informatique.

Il est donc important que le concept -principalement dans le cas de modules- soit formalisé sous forme d'instructions hiérarchisées, dans un langage universel, avant toute implémentation informatique. C'est ce que nous appellerons : algorithme. **Il faut donc bannir le codage direct d'une idée**. Cette formalisation garantit une compréhension entre l'auteur du concept et les informaticiens, aussi bien dans le cas des modules que de l'infrastructure.

Dans la mesure du possible, le code informatique doit être écrit de la manière la plus proche possible du langage naturel. Dans le cas idéal, le code se lirait aussi facilement et clairement qu'une description textuelle. Ce n'est malheureusement jamais le cas, surtout avec les langages de « bas niveau », proches du langage machine. **Les commentaires dans le code source sont donc obligatoires.** Il y a trois type de commentaires :

1. Ceux qui accompagnent et décrivent une astuce de programmation, peu intuitive par nature.

8 Le compilateur GCC offre les options « -Wall » et « -Wextra » pour demander les vérifications les plus usuelles. Elles peuvent être accompagnées de nombreuses autres options qui impose encore plus de rigueur.

2. Ceux qui résument ce que produit un bloc d'instructions de quelques lignes. C'est à dire les grandes étapes d'une fonction/routine informatique.
3. Ceux qui décrivent succinctement l'algorithme d'une fonction/routine informatique.

Ces commentaires dans le code source n'exonèrent évidemment pas de réaliser toute les documentations complémentaire qui pourrait utiles à la compréhension générale du module et des cas particuliers -conditions ou limites- de fonctionnement.

2.4.2 La maitrise des outils utilisés

Au delà de la syntaxe, maitriser un langage informatique, c'est avoir une bonne compréhension des mécanismes qui transforment le code source originel en code binaire et exécutable. Ceci est indispensable pour garantir la bonne « marche » des fonctionnalités attendues.

Or les langages de haut niveau anticipent certaines décisions de conception. C'est à dire qu'il décident d'eux même la manière de coder en instructions de bas niveau les instructions de haut niveau données par l'utilisateur. L'expérience montre que, même lorsqu'on met à profit les fonctions offertes par ces langages, il est très important de savoir de quelle manière elles se rapportent aux problèmes de bas niveau. Sans cette connaissance, il est facile de se trouver confronté à des problèmes de performance et de comportements imprévisibles.

2.4.3 La gestion des erreurs

Si elle permet une plus grande fiabilité du code informatique, la gestion des erreurs engendre des coûts dans la phase de développement. Elle a aussi une influence sur les temps d'exécution. Les erreurs peuvent être classées en deux types :

- Les erreurs de type sémantique sont classées en trois sous-types : les erreurs d'algorithmes, les erreurs de calculs ou les erreurs de données.
- Les erreurs techniques sont classées en deux sous-types : les erreurs liées aux opérations invalides et les erreurs liées aux données invalides.

Pour des questions de fiabilité, robustesse la plateforme sol virtuel gèrera tous ces types d'erreurs. Le détail des erreurs traités sera précisé dans le document de conception détaillée.

2.4.4 Réalisation des tests

Le test est une tentative voulue et systématique de faire tomber en panne un programme censé fonctionner correctement. Il est généralement lancé après chaque modification du programme ou en cas de doute sur son fonctionnement. Le test ne doit pas être confondu avec le débogage. Ce dernier est entrepris dès que l'on sait que le programme est défectueux, donc potentiellement après un test. Tester permet donc de montrer :

- qu'une fonctionnalité est correctement réalisée ;
- la présence de bogues, mais pas leur absence ;
- l'absence/existence de régression (voir note 7, page 7) après une intervention sur le code.

Étant donné que la construction d'un programme informatique -quelque soit sa taille- ne peut être parfaite, il ne peut donc pas être sans défauts. Il est donc indispensable de mettre en place des procédures de tests pour en détecter le plus possible.

L'idéal pour écrire un code « sans » bogues est de le faire générer par un programme. Dans le cas de « sol virtuel », la tâche qui consiste à assembler les modules et les ordonnancer peut facilement être automatisée à partir de spécifications. La plateforme peut aussi automatiser, la gestion des entrées-sorties, au prix de la mise en place d'un système plus intrusif dans le code des modules.

2.4.5 L'importance de la traçabilité

La traçabilité est -avec la documentation- un élément important de la qualité. En informatique de développement, la traçabilité des modifications du code est capitale, quelque soit la taille de l'équipe informatique. La traçabilité des modifications du code comporte deux parties :

1. La traçabilité des modifications proprement dite : qui a modifié quoi, pourquoi et quand. Idéalement géré par un outil de gestion de version. Exemple : Subversion (alias SVN).
2. La traçabilité des demandes de modifications : qui a demandé quoi -éventuellement pourquoi- et quand. Idéalement géré par un outil de gestion de demandes d'anomalies. Exemple : Mantis.

La gestion de versions a de l'intérêt dès qu'un code informatique à une durée de vie significative, ce qui est la cas de la plateforme sol virtuel. Elle devient vite indispensable dès que plusieurs programmeurs-contributeurs travaillent -successivement ou en même temps- sur un même code, au travers d'une connexion par un réseau informatique. Le logiciel détecte alors les conflits et facilite la mise en œuvre de solutions.

2.4.6 La réalisation et la maintenance de la documentation

La documentation dans la phase de développement est indispensable pour garantir une certaine facilité dans la maintenance -corrective et évolutive- de l'ensemble du code informatique produit. Le contenu sera alimenté -pour une grande partie- par les spécifications détaillées des fonctions et des commentaires pertinents dans le corps de ces fonctions. Cela est valable aussi bien pour l'infrastructure que pour les modules.

2.4.7 La qualité et la complexité du code source

La production de code est réalisée pour répondre aux spécifications. Les codes sont produits et maintenus par des développeurs potentiellement différents. Il est donc important que ces codes soient compréhensibles par l'ensemble des développeurs. Un minimum de points/usages seront donc spécifiés dans une charte et devront être respectés. L'ensemble de ces points seront repris plus précisément et techniquement dans le document de conception détaillée.

De manière générale, plus un document est complexe, plus il est difficile à comprendre et à analyser, et donc à corriger. En terme de développement de logiciel, la complexité d’une application a un impact direct sur le pourcentage d'erreurs et la robustesse du code. En effet, la complexité du

logiciel se reflète sur sa difficulté à être testé : plus un logiciel est complexe, plus il est difficile de le tester et de le valider.

Pour garantir un logiciel de qualité tout en assurant des couts de tests et de maintenance faibles, la complexité d'un logiciel devrait être mesurée dès le début du codage. Ainsi lorsque les valeurs recommandées sont dépassées, le développeur peut intervenir rapidement, avant que la situation « empire ». Pour quantifier la complexité d'un logiciel, on se sert de métriques.

Le document de conception détaillée sera l'occasion de préciser ce que sont ces métriques et le processus d'évaluation de la complexité. Toutefois, même si sont intérêt est évident pour l'équipe informatique, sa mise en œuvre demandera des compétences qu'il conviendra d'acquérir lors de la construction et de la mise en place des premières productions logicielles. Les résultats de cette démarche ne seront pas visible avant quelques mois après la diffusion publique de la plateforme. Il est prématuré d'envisager un tel chapitre dès la première version du document de conception détaillé.

Ces notions de complexité et de qualité du code source touchent forcément de près les développeurs informatiques « de métiers »[9] et donc notamment ceux qui ont le rôle « informaticien de plateforme », qui sont fortement impliqués dans le développement de la plateforme. Cela concerne aussi les scientifiques qui endossent le rôle « informaticien de modules ». Une certaine qualité étant aussi requise, que ce soit pour un module préexistant et importé dans la plateforme que pour un module créé à partir de l'interface graphique de la plateforme.

La plateforme doit produire automatiquement le maximum de code informatique qui n'est pas lié au code de calcul des modules. Cette partie « code d'infrastructure » sera générée par des procédures qui respecteront un niveau de qualité élevé. Pour les « informaticiens de modules », cela permet d'alléger le cout d'adaptation aux formalismes informatiques de la plateforme lors de la production du code de calcul des modules.

Dans l'avenir, il est envisageable que la plateforme propose d'assister l'utilisateur dans la saisie de formalismes numériques/mathématiques « classiques ». Par exemple, demander uniquement la saisie de coefficients d'une équation différentielle « simple » pour générer automatiquement le code informatique. Toutefois, si la plateforme doit faciliter la vie des utilisateurs, elle doit lui laisser la liberté de choisir la meilleure façon de faire. Simplifier et masquer les besoins les plus communs ne doit en effet pas empêcher les réalisations complexes ou non prévues.

9 Par opposition aux développeurs « occasionnels », qui programment généralement pour leurs besoins propres et de manière irrégulière.

3 Architecture globale

3.1 Les trois grandes parties de la plateforme

L'étape d'analyse a mis en avant trois parties (ou étapes) concernant l'infrastructure de la plateforme. L'architecture globale se conforme donc à ces trois parties. Cette structure est symbolisée dans l'illustration 1 « les 3 grandes parties de la plateforme », ci-dessous.

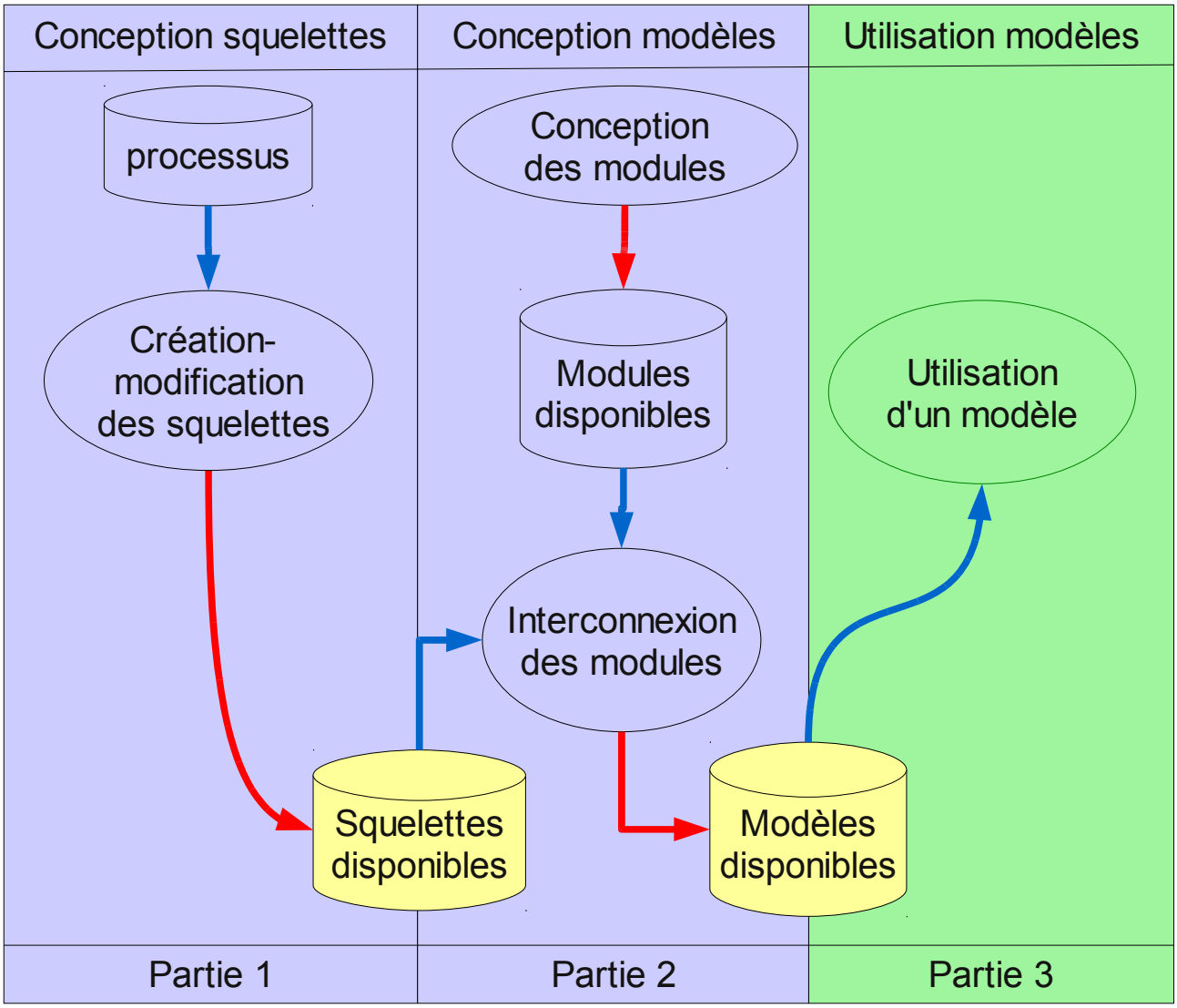

Illustration 1: les 3 grandes parties de la plateforme

L'accès à ces parties est différent selon les acteurs concernés :

- les parties 1 et 2 (couleur violette) sont accessibles uniquement aux acteurs scientifiques de

modules et de modèles

- la partie 3 est accessible aux acteurs scientifiques utilisateurs de modèles (cf.définition des acteurs chapitre 3.3 du document d'analyse).

En fonction du type d'accès, les choix conceptuels peuvent différer, partant du principe que le niveau d'appréhension des concepts est différent.

L'attribution de la couleur jaune indique le partage du contenu par deux composantes.

Les trois parties sont liées et successives. Les données produites dans une partie sont nécessaires pour la réalisation de la partie suivante. La notion de piscine -issue de l'analyse informatique- est ici symbolisée par un cylindre. Les ellipses correspondent aux logiciels de la plateforme, manipulant les piscines.

3.2 Les composants logiciels

Chacune des parties de la plateforme met en œuvre des composants logiciels bien identifiés. Chaque composant a vocation à être utilisé par des acteurs identifiés de la plateforme. Exemple : la création de processus et squelettes concerne le rôle « scientifique de plateforme ». Les besoins logiciels sont propres à chaque composant. Ils sont détaillés dans les sections suivantes.

3.2.1 Partie 1, conception de processus et de squelettes

La partie 1 de la plateforme consiste à définir des processus d'un point de vue scientifique, puis à fabriquer des squelettes à partir de l'assemblage ordonnée et cohérent des processus définis.

3.2.1.1 Conception de processus

La conception de processus nécessite un langage de formalisation des processus. Ce langage doit aussi contenir la définition des entrées et sorties, utiles aux processus. Le fichier ainsi produit représente la piscine des processus.

Ce fichier doit être librement éditable par un éditeur de fichier texte. Mais son usage principal doit être réalisé au travers d'une interface graphique. Celle-ci doit notamment montrer à l'utilisateur les processus en amont et en aval du processus qu'il souhaite créer, modifier ou observer.

3.2.1.2 Conception de squelettes

La section 4.1 du document d'analyse précise qu'un squelette est un graphe orienté dont les nœuds sont les processus. Pour une dimension donnée -1D/2D/3D- les squelettes possibles sont produits à partir de l'assemblage de tous les processus disponibles pour cette dimension. Les processus qui proposent des sorties identiques produisent des variantes de squelettes, sans conflits.

La conception de squelettes peut donc être réduite -dans un premier temps- à de la génération automatique d'assemblages maximal de processus. Cette réalisation logicielle n'est pas visible directement par l'utilisateur. Celui-ci accède aux squelettes au travers d'une représentation graphique qui montre à la fois les squelette possibles, ainsi que les processus impliqués. Cette interface doit donc permettre et surtout faciliter l'appréhension des squelettes possibles à partir des caractéristiques des processus.

Dans un second temps, l'utilisateur pourra manipuler le squelette qu'il a sélectionné pour en

faire une représentation du sol plus spécifique en activant ou non certains processus. Cette deuxième étape nécessite donc un langage de formalisation des squelettes, en précisant notamment les processus et les entrées/sorties sélectionnés (cf. les contrats de processus décrits dans l'analyse).

3.2.2 Partie 2, conception de modules et de modèles

La partie 2 de la plateforme consiste à réaliser une représentation concrète -informatique- de la partie 1. C'est à dire qu'un processus est réalisé par un code informatique, appelé module, et qu'un squelette est représenté par un exécutable informatique, appelé modèle.

3.2.2.1 Conception de modules

Comme pour les processus, la conception de modules nécessite un langage de formalisation des modules qui utilisera la définition des entrées et sorties utiles aux processus, mais en respectant les contrats (cf. analyse). En plus des entrées/sorties, ce langage doit permettre la déclaration des paramètres nécessaires au fonctionnement de chaque module. Ces informations serviront à la génération automatique du code d'infrastructure qui va encapsuler le code source informatique de calcul du module. Au final, ces informations représentent la piscine des modules.

Comme pour les processus, ce fichier doit être librement éditable par un éditeur de fichier texte. Mais son usage principal doit être réalisé au travers d'une interface graphique. L'utilisateur pourra alors :

- faire le lien avec le processus et son module ;
- déclarer les entrées dont il a besoin ;
- déclarer les sorties qu'il peut fournir ;
- déclarer les paramètres spécifiques ;
- saisir/valider le code source informatique.

3.2.2.2 Conception de modèles

3.2.2.2.1 Interconnexion de modules

La conception de modèles est basée sur le choix d'un squelette et de certains processus. Ainsi, pour réaliser un modèle, il faut choisir -pour chaque processus contenu dans le squelette sélectionné- un module qui le représente. L'utilisateur pourra être contraint dans sa sélection de modules par les données d'entrées dont il dispose. Exemple : Le module de transfert hydrique utilisant les équations de Richards nécessite des relations « conductivité hydraulique – teneur en eau » et « potentiel hydrique – teneur en eau » pour chaque couche homogène de sol. Si l'utilisateur ne les possède pas alors ce module ne pourra pas être utilisé pour le modèle en cours de construction.

Le langage de formalisation des modèles doit donc représenter les modules choisis. Les informations utiles pour les modèles déjà élaborés seront rassemblées dans la piscine des modèles.

3.2.2.2.2 Description commune aux modules

Les trois parties implémentent des composants indépendamment du domaine d'application de la plateforme. Cependant, l'interconnexion des modules dans « sol virtuel » passe par la

spécification de paramètres de sol communs/partagés par les modules. Ce « typage » du sol passe par une caractérisation numérique, une description du sol numérico-informatique commune à tous les modules. Cette description sera partagée par tous les modules lors de l'exécution d'un modèle.

Pour simplifier, nous considèrerons que la caractérisation du sol est constante durant toute l'exécution d'un modèle. C'est donc pour l'instant, une description a-temporelle et a-processus. Plus tard, il sera possible de définir des modules qui changent cette caractérisation dynamiquement pendant l'exécution d'un modèle. Cependant, la forme de description du sol restera la même. Cela ne changera donc pas la forme des modules existants.

Le sol est décrit, quelque soit la dimension 1D/2D/3D, par la définition de son profil. Ce profil va de la surface jusqu'à une profondeur souhaitée. Ce profil est découpé en strates -pour simplifier volontairement horizontales- homogènes appelées « couches ». Une couche possède des propriétés identiques (par exemple de transferts hydriques) en tous points. Chaque couche est identifiée par sa profondeur -en valeur absolue-, qui donne sa position de début, selon un axe vertical qui commence à la surface. Lors de l'introduction de modules 2D et 3D, cette représentation devra être complétée par un maillage plus adapté, non homogène. C'est à dire qu'en 2D/3D, la définition de zones de sol -avec des propriétés différentes- pourra être quelconque.

La résolution numérique employée par les modules -de transferts- est basée sur un maillage. Ce maillage -dit « de calcul »- définit un ensemble de points pour lesquels chaque module est capable de fournir ses sorties. Comme pour les couches, la distribution des points du maillage de calcul n'est pas nécessairement régulière. Cependant, il y a au moins un point de maillage de calcul à chaque changement de couche.

En résumé, le sol est défini par deux vecteurs de points, qui représentent des valeurs :

- un vecteur qui définit la limite de chaque couche ;
- un vecteur qui définit les points du maillage.

Tous les modules de transfert -non locaux- utilisent cette représentation commune du sol en couches (appelés « layers”) et maillage (de calcul). A titre d'exemple, voir « Illustration 2: caractérisation numérique du sol ».

L'utilisateur peut vouloir observer des points indépendamment des points du maillage. Or, les modules ne seront pas tenus de fournir les valeurs en ces points. La plateforme offrira donc un système d'interpolation adapté.

Pour définir l'état initial du sol, il est nécessaire de connaitre la valeur de chaque variable d'état de chaque module en tout point du maillage. L'utilisateur ne pouvant pas raisonnablement fournir toutes ces valeurs, une interpolation est donc nécessaire pour calculer les valeurs aux points du maillage, à partir des données disponibles. La plateforme pourra donc proposer plusieurs techniques d'interpolation.

Pour clore cette section, voici une illustration de la caractérisation du sol. L'illustration 2 (caractérisation numérique du sol) donne un exemple avec les valeurs suivantes :

- un vecteur de 4 valeurs de profondeur -Z_0, Z_1, Z_2 et Z_3- qui définissent 3 couches ;
- un vecteur de 10 valeurs de profondeur -M_0, à M_9- qui définissent 10 points de maillage pour le calcul de toutes les variables d'état des modules ;
- un vecteur de 5 valeurs de profondeur -water_flow$_0$ à water_flow$_4$- qui définissent 5 profondeurs comme valeurs de sorties de la variable d'état « water flow » ;
- un vecteur de 2 valeurs de profondeur -soil_temperature$_0$ et soil_temperature$_1$- qui définissent 2 profondeurs comme valeurs de sorties de la variable d'état « soil temperature »

Les vecteurs des valeurs initiales des variables d'états sont volontairement ignorées pour ne pas surcharger le graphique.

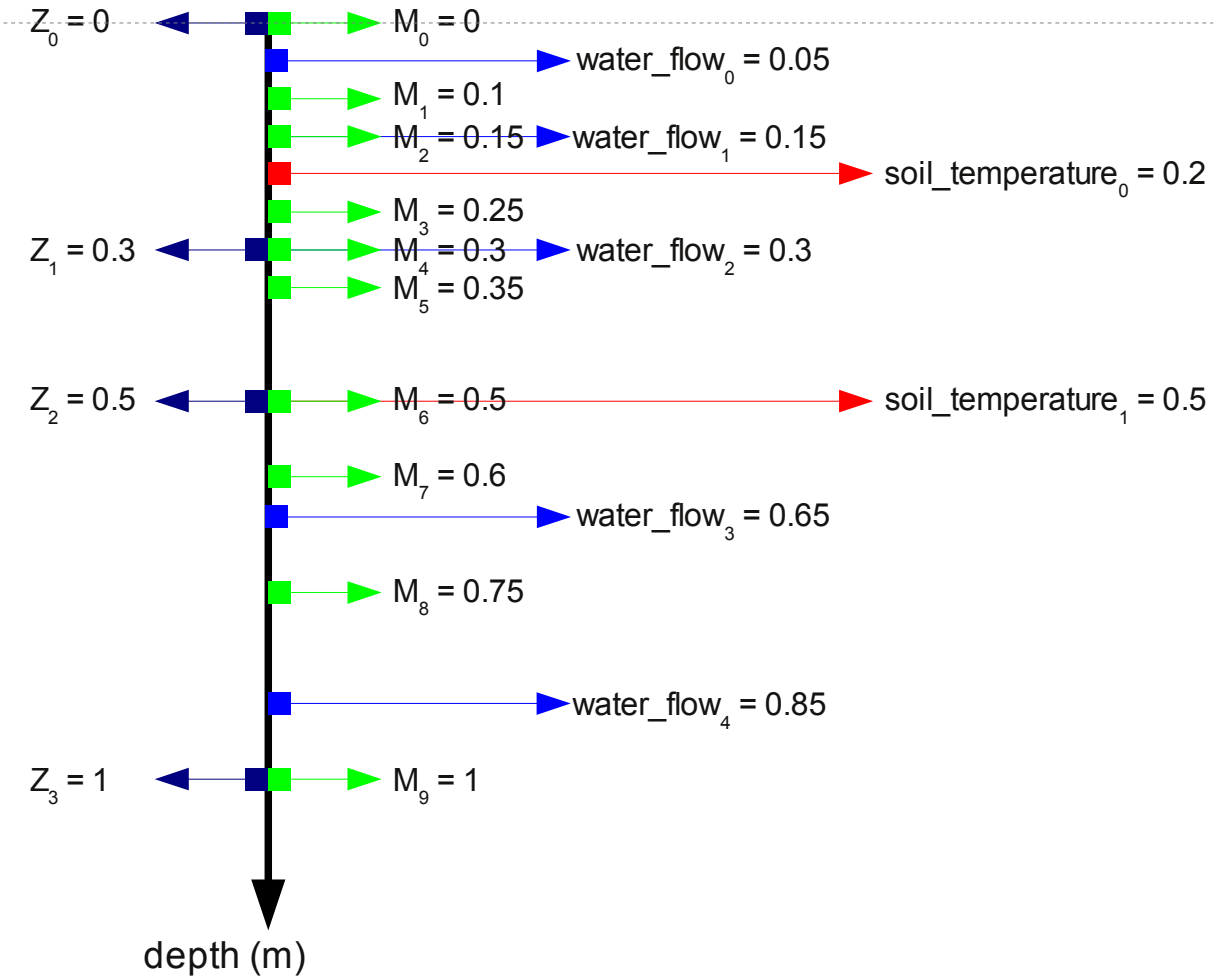

Illustration 2: caractérisation numérique du sol

3.2.2.2.3 Caractérisation d'une simulation

Pour exécuter une simulation avec un modèle de la plateforme « sol virtuel », il est nécessaire de caractériser des paramètres, dont certains sont liés entre-eux :

- aux besoins propres des modules ;
- à la description du sol (voir section précédente) ;
- à la gestion du temps.

Au niveau temporel, une simulation peut se caractériser de deux manières :

- des dates de début et de fin de simulation ;
- une durée.

La tableau ci-dessous décrit tous les paramètres communs utiles à une simulation :

Paramètre	Libellé	Type	Unité	Bornes min → max	Défaut
Nombre de couches de sol	Nb of layers	Entier	NA	1 → 10	1
Couches du sol	Layers	Vecteur de réels	m	0 → profondeur layer max	0
Nb de points de maillage	Nb grid	Entier	NA	2 → 1000	2
Maillage	Grid	Vecteur de réels	m	0 → profondeur layer max	
Distance max points maillage	Max grid distance	Réel	m	Min grid distance → 5.10^{-2}	5.10^{-2}
Distance min points maillage	Min grid distance	Réel	m	10^{-4} → max grid distance	10^{-4}
Date de début de simulation	Start time	Date calendaire (UTC ?)	AAAA-MM-JJ HH:MM:SS	1900-01-01 00:00:00 → end time	NA
Date de fin de simulation	End time	Date calendaire (UTC ?)	AAAA-MM-JJ HH:MM:SS	Start time. → 2199-12-31 23:59:59	NA
Durée de simulation	Time to run	Réel	s	0 → NA	NA
Pas de temps max	Max time bound	Réel	s	Min time bound → 3600	3600
Pas de temps min	Min time bound	Réel	s	10^{-2} → max time bound	10^{-2}

Tableau 2: exemple de paramètres caractérisant une simulation 1D

3.2.3 Partie 3, utilisation de modèles

La partie 2 de la plateforme permet de produire des modèles sous différentes formes logicielles, avec un niveau d'appréhension bien différent selon le type d'utilisateur :

- exécutable en lignes de commande, pratique pour faire de l'exécution par lots (batch) ;
- exécutable avec interface graphique, probablement l'usage le plus fréquent ;

- code source informatique du modèle, à compiler par les soins de l'utilisateur.

La saisie des valeurs d'entrées -sous forme de paramètres- est une étape nécessaire au fonctionnement des modèles. Le nombre d'entrées peut être important. De plus pour garantir un fonctionnement cohérent du modèle, la validité de ses entrées -domaine de valeurs possibles- doit être assurée. D'où la quasi-obligation de fournir aux utilisateurs une interface graphique adaptée pour chacun des modèles possibles.

Cette interface graphique est une couche logicielle qui facilite -pour l'utilisateur- l'accès au modèle sous-jacent de son choix. L'utilisation d'un modèle nécessite l'entrée de valeurs de paramètres qui permettent l'initialisation des modules et du système couplage (cf. section précédente). Les valeurs utiles au déroulement des modules, les entrées qui changent -par nature- à chaque pas de temps, ne sont pas nécessaires dans cette interface. En effet, elles sont fournies par des modules dédiés à cette tâche. Ces modules n'ont pas de caractère scientifique. Ils servent à fournir les entrées globales au modèle. Ils simplifient grandement la conception des modules qui dépendent de leurs données puisqu'ils s'interfacent pour « cacher » les différentes sources de données possibles : simple fichier ACSI, fichier XML, tables de bases de données...

L'interface graphique qui permet l'utilisation d'un modèle donné doit donc contenir au minimum les objets graphiques pour la saisie des données nécessaires. L'apparence graphique de ces objets doit spécifiquement être adaptée au type d'information de la donnée : valeur scalaire entière ou réelle -signée ou non-, vecteur, nom de fichier...

L'interface graphique contiendra aussi des objets graphiques utiles à la visualisation des données en sortie du modèle choisi. Cela comprend les données calculées à chaque pas de temps ou des données interpolées. L'interface doit aussi permettre l'accès à des grandeurs agrégées dans l'espace et pour diverses périodes de temps, c'est à dire des sommes ou des moyennes des variables d'état ou des grandeurs dérivées de celles-ci.[10]

La grande difficulté de cette partie « utilisation de modèles » provient du grand nombre de modules et modèles possibles. Il y a autant d'interfaces graphiques -qui doivent être adaptées- que de modèles. Il est donc nécessaire de mettre en œuvre un système d'interface graphique automatique. Ce système est complexe à mettre en œuvre mais absolument nécessaire, puisqu'il n'est pas envisageable de produire « à la main » toutes les interfaces graphiques possibles.

Dans le cadre de la plateforme « sol virtuel », la génération automatique d'une interface graphique est basée sur la définition des paramètres, entrées et sorties des modules concernés par le modèle choisi. A chaque donnée est rattachée un objet graphique qui permet la saisie de sa valeur. Suivant le type de la donnée -scalaire, réel...- le mode opératoire de saisie peut être différent : champs texte, boutons, ascenseurs... La définition de l'interface graphique du modèle est produite dans un fichier par un logiciel qui déduit les besoins en fonction de la description du modèle choisi et de ses modules. Ce fichier est ensuite lu par une interface graphique -minimaliste et indépendante de tout modèle- dont l'aspect s'adapte en fonction des définitions du fichier. Cette interface possède aussi un générateur -adapté au modèle choisi- qui met les données saisies au format attendu par le

10 La plateforme « openfluid » utilise le terme de « shopping list » qui définit la liste des variables souhaitées ainsi que les pas de temps pour lesquels les récupérer.

modèle.

Le cadre de ce travail peut dépasser celui de « sol virtuel ». Il pourrait être utile à d'autres plateformes du département EA, notamment « Record ».

3.3 Distribution logicielle de la plateforme

Lorsqu'un nouvel utilisateur de la plateforme veut utiliser un des composants logiciels de la plateforme, il est capital qu'il puisse à la fois :

1. facilement trouver quels composants logiciels sont disponibles par rapport à son besoin (création de processus, insertion du code d'un module...) ;
2. facilement récupérer et installer le composant souhaité uniquement.

Des difficultés lors de l'une de ces deux étapes pourraient avoir un effet désastreux sur la bonne acceptation du projet de la part des nouveaux utilisateurs. C'est pour cela que la plateforme doit se doter d'un mécanisme simple qui réponde à la fois à ces deux points. Le scénario actuellement retenu devra fonctionner aussi bien sur les plateformes Linux que Windows. La plateforme MacOS est aussi envisageable, via un émulateur capable de faire fonctionner les applications natives Linux.

3.3.1 Scénario Windows

1. L'utilisateur se rend sur le site Web de la plateforme et va dans la rubrique dédiée à l'aide au choix d'un logiciel.
2. En fonction de son objectif, le site Web lui indique le logiciel à utiliser et lui propose un lien de téléchargement spécifique à sa version de Windows (XP ou Vista/7, 32 ou 64 bits).
3. L'utilisateur exécute le logiciel téléchargé pour procéder à son installation.
4. Le logiciel est disponible à partir de raccourcis présents dans les rubriques habituelles. Exemple : une icône dans le menu « Démarrer » et une sur le « bureau » de l'utilisateur.

3.3.2 Scénario Linux

1. L'utilisateur se rend sur le site Web de la plateforme et va dans la rubrique dédiée à l'aide au choix d'un logiciel.
2. En fonction de son objectif, le site Web lui indique le logiciel à utiliser et lui propose un lien de téléchargement/installation spécifique à sa distribution Linux (Debian/Ubuntu ou Redhat/Mandriva, 32 ou 64 bits)[11].
3. Le logiciel est disponible à partir de raccourcis présents dans les rubriques habituelles de son environnement graphique. Exemple pour KDE : le menu « K » (équivalent du menu « Démarrer » sous Windows).

11 Pour les distributions de la famille Debian/Ubuntu, il est prévu de mettre en œuvre un dépôt officiel qui simplifierait énormément cette étape 2.

4 Les solutions techniques

Les solutions techniques à mettre en place, dans le cadre de « sol virtuel » sont de 4 types :

- interfaces utilisateurs graphiques ;
- connexion de modules ;
- représentation et traitement des données ;
- langages informatiques.

4.1 Les interfaces utilisateurs graphiques

4.1.1 Attente « interface graphique riche »

Les attentes en terme d'interfaces graphiques sont de permettre un accès facile et convivial aux différentes parties fonctionnelles de la plateforme pour chaque type d'acteurs identifiés (voir document d'analyse, section 3.2, page 18).

L'infrastructure de la plateforme est composée d'un ensemble d'ateliers logiciels (section 4.4 du document d'analyse). Il est important d'analyser toutes les spécificités techniques de ces ateliers pour réaliser un choix pertinent. Notamment en terme de richesse d'objets graphiques et d'interactions possibles. Cela peut exclure les interfaces utilisant des technologies de type client-serveur, plus précisément celles utilisant les navigateurs Web, dont la programmation peut devenir complexe selon les capacités graphiques à mettre en œuvre.

4.1.2 Attentes dans les trois parties de la plateforme

Les interfaces graphiques concernant la partie 1 de la plateforme visent à rendre disponible la description des processus et l'organisation des squelettes. Ce sont les ateliers des processus et des squelettes. Ces interfaces sont le point d'entrée pour les scientifiques qui souhaitent participer à la construction de nouvelles représentations du sol, ou plus simplement y adjoindre leurs connaissances sur des phénomènes précis. Un focus tout particulier doit être mis sur la simplicité et la clarté des processus disponibles -leurs caractéristiques et les interactions avec les autres processus- et leurs assemblages -générés automatiquement- dans les différents squelettes possibles. Par défaut, ces interfaces doivent mettre l'utilisateur dans un scénario d'observateur uniquement, n'autorisant pas de modifications sans un avertissement explicite sur les conséquences des ses futurs actes.

Les interfaces graphiques concernant la partie 2 permettent l'intégration de modules -donc de codes informatiques modulaires- et la réalisation de modèles -à travers l'aide au choix de processus, des squelettes, puis de modules. Ce sont les ateliers des modules et des modèles (point 1 uniquement, voir document d'analyse section 4.4.4). Ces interfaces concernent des acteurs plus « techniques » et donc potentiellement plus habitués à la nécessaire complexité d'interfaces graphiques offrants de nombreuses possibilités. Il est cependant souhaitable de réaliser des interfaces à deux niveaux de complexités/fonctionnalités. L'utilisateur pourra alors choisir entre un mode « simple » -qui offre les possibilités les plus standards et guide l'utilisateur pas à pas- et un mode « expert » -offrant toutes les fonctionnalités possibles.

Les interfaces graphiques concernant la partie 3 permettent de configurer et d'exécuter les

modèles. C'est l'atelier des modèles (à partir du point 2, voir document d'analyse section 4.4.4). Plus précisément, l'atelier des modèles aura une interface dynamique capable de s'adapter en fonction du modèle choisi et donc des modules à paramétrer. Le coté dynamique de l'interface ne permettant pas de contrôler précisément le contenu des objets graphiques affichés, une attention particulière sera portée sur les différentes situations graphiques possibles.

4.1.3 Attente « multi-plateforme »

La librairie graphique utilisée doit être utilisable sur les plateformes systèmes identifiées et doit aussi tenir compte des différentes configurations possibles chez les utilisateurs. Par ordre de priorité, cela se situe en termes de :

1. systèmes d'exploitations
   - les différents Microsoft Windows (XP et Vista/Seven)
   - les différentes distributions Linux (la famille Debian/Ubuntu)
2. tailles d'écrans
   - en résolution, c'est à dire nombre de pixels (minimum 1280 horizontal et 900 vertical)
   - en taille physique, c'est à dire la diagonale
3. choix d'apparence graphique
   - la taille des fontes
   - l'apparence des objets graphique (ex : boutons, ascenseurs), notamment leur taille

4.1.4 Résumé des contraintes

En résumé, les contraintes techniques -pour l'ensemble des ateliers- liées au choix de la librairie graphique sont :

- représentation de graphes ;
- composant pour l'édition avancée de codes sources ;
- composant de visualisation de courbes ;
- rapidité de génération des composants graphiques ;
- multi-plateforme.

4.2 Connexion de modules

4.2.1 Attente « système de couplage »

Pour la réalisation de modèles, il faut agencer les modules correspondant aux processus souhaités en faisant intervenir les notions de temps et d'espace. C'est le rôle du système de connexion de modules -ou système de couplage- que de permettre d'intégrer et d'assembler des codes informatiques. S'il peut-être considéré comme le cœur d'une plateforme de couplage scientifique, sa mise en œuvre dans « sol virtuel » ne concerne que la partie 2 de la plateforme. Ce n'est donc pas uniquement le choix d'un système de couplage qui va répondre aux attentes scientifiques exprimées. Il ne doit pas être un sujet de blocage, si les attentes en terme de couplage

ne sont pas permises -et facilement réalisables- avec le système mis en œuvre.

Il est utile de mettre en perspective ce qu'offre les systèmes de couplage existant -notamment ceux utilisés dans des plateformes scientifiques- avec les besoins de « sol virtuel ». En effet, l'utilisation d'un système de couplage performant -offrant de nombreuses possibilités de couplage- entraine inévitablement une complexité supplémentaire, non négligeable, dans son apprentissage et sa mise en œuvre. Or, dans le cadre actuel de « sol virtuel », l'idée est de valider rapidement la cohérence et la pertinence des concepts issus du cahier des charges et de l'analyse, notamment ceux liés aux processus et squelettes. Choisir un système de couplage est donc vu comme l'ajout d'une complexité extérieure supplémentaire.

La complexité mathématique exigée pour le système de couplage de « sol virtuel » est relativement faible. Les modules utilisent en effet des formalismes simples et identiques : équations différentielles uniquement du 1er et 2nd ordre. A court terme, c'est donc une solution de couplage direct qui est mise en œuvre : chaque module est invoqué directement au travers de sa fonction principale de calcul. Outre sa simplicité, ce choix est intéressant puisqu'il permet une adaptation vers un système autre (celui de la plateforme Record par exemple) de manière la plus simple possible.

4.2.2 Attente « accès au code source »

Outre les attentes sur le système de couplage, la plateforme doit permettre un accès aux codes sources informatiques. Cela concerne à la fois celui des modules et celui des modèles. C'est une sortie de la partie 2 qui répond à un besoin dans la partie 3. Les codes sources des modules et des modèles disponibles permettent une importation dans des codes externes. L'objectif principal est dans ce cas de favoriser l'intégration des produits de « sol virtuel » dans les autres plateformes, notamment Record (systèmes de culture) et « paysage virtuel » qui sont demandeur de représentations plus ou moins complexes du sol.

La possibilité d'exporter les codes sources des modules de la plateforme « sol virtuel » à des fins de couplages externes ne doit pas modifier le code original des modules. En effet, le groupe projet souhaite que la méthode ne soit pas intrusive. Notamment pour éviter que cela ne rende le code informatique dépendant de la technique de couplage utilisée.

4.3 Représentation et traitement des données

Afin de représenter les quatre grands types d'informations -processus, squelettes, modules et modèles- la plateforme doit se doter d'un formalisme de représentation et traitement -interrogation et manipulation- de données.

Le langage qui semble le plus adapté à la situation est XML[12]. En effet, ce langage générique sert essentiellement à stocker/transférer des données de type texte (unicode) structurées en champs arborescents. Ce langage est qualifié d'extensible car il permet de définir librement des balises (des marqueurs nommés) et des éléments (attributs typés) à ces balises. Il est fortement recommandé par le *World Wide Web Consortium* (W3C), promoteur de ce standard pour favoriser l'échange d'informations.

XML permet la représentation des données utiles à la plateforme. Avec les langages associés à XML que sont XPath et Xquery, il est possible de faire des requêtes permettant non seulement d'extraire des informations d'un document XML, ou d'une collection de documents XML, mais

12 « eXtensible Markup Language », http://fr.wikipedia.org/wiki/Xml

également d'effectuer des calculs complexes à partir des informations extraites et de construire de nouveaux documents ou fragments XML. XQuery se base sur Xpath. Il est une spécification du W3C et joue -par rapport aux données XML- un rôle similaire à celui du langage SQL vis-à-vis des données relationnelles. On peut d'ailleurs trouver de fortes analogies entre ces deux langages.

La représentation exacte des données utiles à la plateforme, et notamment les piscines, ne sont pas le sujet de ce document de conception générale. Les détails d'implantations seront développés dans le document de conception détaillée.

4.4 Langages informatiques

4.4.1 Attente « infrastructure de la plateforme »

Le choix des langages pour l'infrastructure de la plateforme -hors système de couplage- est relativement libre. Mais il convient qu'il réponde convenablement aux critères suivants :

1. tolérance aux évolutions ;
2. largement utilisé ;
3. standard ;
4. coutumier ;
5. multi-plateforme, dans le cadre des systèmes d'exploitations sélectionnés.

4.4.1.1 Tolérance aux évolutions

Le mode de gestion de projet retenu pour « sol virtuel » (section 2.4 du document d'analyse) va engendrer de nombreuses évolutions. Celles-ci peuvent même être de nature à radicalement revoir l'architecture logicielle. Les langages à objets ont prouvé leurs aptitudes sur ce critère de la tolérance aux évolutions. En effet, grâce au paradigme de l'encapsulation -qui est un des quatre grands principes des langages à objets- les codes informatiques ainsi produits sont fortement modulaires et fortement cloisonnés. Ceci permet de ré-agencer le code -même lourdement- en limitant les effets de bords.

4.4.1.2 Largement utilisé

De nos jours l'usage d'Internet pour aider aux développements informatiques a radicalement changé la manière dont les développeurs travaillent. Les critères de sélection de langages, librairies et outils informatiques ont évolués dans l'esprit des développeurs. En particulier, l'aspect payant des produits commerciaux n'est plus perçu comme une garantie de qualité. A l'heure actuelle, le critère jugé important, dans le choix d'un outil informatique, c'est d'abord la présence sur la « toile » : de forums de discussions -entre utilisateurs ou avec les concepteurs-, de FAQ[13], de tutoriels, d'exemples d'utilisations et de toutes les documentations utiles qu'un développeur est en droit d'attendre pour démarrer et être efficace sur un outil.

4.4.1.3 Standard

Depuis la naissance de l'informatique, si le nombre de types de langages de programmation[14]

13 “Frequently Asked Questions” (« Foire Aux Questions » en français)

14 Exemples : impératif, fonctionnel, orientée objet, déclaratif, concurrent, générique, logique, par contraintes...

n'a presque pas évolué, le nombre de langages informatiques ne cesse d'augmenter. Depuis l'avènement de l'Internet, il y a même eu une forte accélération[15]. Si tous ces langages sont utiles -sinon ils ne seraient pas populaires- beaucoup sont jeunes et encore en pleine évolution. Les changements de versions majeurs sont souvent source d'incompatibilités. Et même si ces évolutions se font souvent sous la pression d'un grand nombre d'utilisateurs actifs, les choix des évolutions sont le résultat de la décision de leur(s) concepteur(s). Une telle instabilité pourrait nuire à « sol virtuel ».

Pour ne peut pas être gênée par des incompatibilités dues à des changements de versions majeurs, la plateforme ne doit pas seulement utiliser un langage informatique standard[16] -même « standard de fait »-, elle doit privilégier un langage normalisé -par une entité internationale- et dont les versions ont prouvé leur stabilité sur le long terme.

4.4.1.4 Coutumier

Aux critères précédents s'ajoute celui d'utiliser de préférence un langage connu -ou proche- des informaticiens qui vont intervenir dans l'infrastructure de la plateforme. Rien n'interdit l'utilisation d'un nouveau langage, s'il est plus adapté et performant pour répondre aux besoins de la plateforme. Mais si les informaticiens n'y sont pas coutumiers, ils devront inévitablement passer par une phase d'apprentissage, dont le temps est difficilement quantifiable.

Pour toutes ces raisons, les deux langages actuellement les plus adaptés qui répondent le mieux possible à tous ces critères sont les langages orientés objets Java et C++.

4.4.1.5 Multi-plateforme

Les différents parties logicielles de « sol virtuel » doivent utiliser des librairies multi-plateforme. C'est surtout une contrainte forte à propos des ateliers qui se présentent sous forme d'interfaces graphiques. Utiliser de telles librairies limite le choix des possibles. C'est aussi une source de difficultés supplémentaires en terme de développements informatiques. La phase de tests est plus longue, car dupliquée sur chaque système d'exploitation.

4.4.2 Attente « implémentation des modules »

Dans le cahier des charges fonctionnel il est précisé que le langage d'implémentation des modules doit correspondre à un des trois langages « C, C++, Fortran ». Cette même contrainte est imposée dans le cadre de la plateforme « Record ». L'intégration dans « Record » de modules venant de « sol virtuel » est donc plus facile de ce point de vue.

4.4.3 Attente « système de couplage »

Le langage du système de couplage de « sol virtuel » doit nécessairement être compatible avec les trois langages « C, C++, Fortran ». Idéalement, le langage du système doit être un de ces trois. Celui offrant le plus de possibilités en terme de capacités fonctionnelles[17] et de concepts informatiques[18] est C++. Là aussi, le rapprochement potentiel avec Record est facilité.

15 Parmi les plus connus : PHP, ASP, JSP, Python, Ruby, JavaScript, ECMAScript/JScript, VBScript, ActionScript...
16 Attention à faire la distinction entre standard et norme. Standard -signifiant aussi « norme » en anglais- est fréquemment utilisé dans le sens de norme en informatique.
17 comme la gestion des flux pour les entrées/sorties.
18 comme les paradigmes de la programmation objet.

4.5 Les algorithmes de modules et de modèles

4.5.1 L'algorithme de module

Le « cœur » d'un module dans « sol virtuel » est un code de calcul. Pour un pas de temps écoulé donné, il reçoit des entrées -parmi lesquelles ce pas de temps- et calcule des sorties. Le code de calcul peut refuser le pas de temps qui lui est proposé. Notamment lorsqu'il n'y a pas de convergence pour ce pas de temps choisi. Dans ce cas, l'appelant doit lui en proposer un autre, plus court, jusqu'à ce qu'il soit accepté par le module. Les sorties qu'il donne sont toujours les dernières calculées. Mais à tout moment, le module doit pouvoir reprendre son état précédent. C'est notamment le cas, si un autre module refuse de fonctionner avec le pas de temps courant. De plus, avant de pouvoir exécuter son code de calcul, le module doit être initialisé. C'est à dire qu'il doit calculer son état initial, donc mettre des valeurs initiales à ses sorties.

Pour reprendre un paradigme de la programmation objet, un module a un état et des comportements :

- L'état d'un module est représenté par les données qu'il contient. Ce sont au minimum les variables de ses sorties, appelée « sorties calculées ». Pour permettre le calcul de ses sorties sur un pas de temps, il doit aussi contenir les variables de son état précédent. Il contient donc en plus -au minimum- les sorties précédentes, appelées « sorties validées ».
- Les comportements d'un module sont représentés par trois fonctions :
    1. une fonction d'initialisation
    2. une fonction de calcul
    3. une fonction de validation

En termes algorithmiques, l'exécution d'un module passe par les trois étapes suivantes, contrôlées par le coupleur de la plateforme :

1. **L'initialisation** consiste à recevoir des paramètres et les entrées initiales. Ces données servent à calculer les valeurs initiales des sorties, « calculées » comme « validées ». A la fin de cette étape, le module est en mesure de fournir ses sorties pour les modules en aval.
2. **Le code de calcul** est exécuté pour un pas de temps donné. Il change l'état du module -les « sorties calculées »- en fonction des données précédemment validées (les « sorties validées »). A la fin de cette étape, le module est donc en mesure de fournir les nouvelles « sorties calculées » pour les modules en aval. Mais ses « sorties validées » restent inchangées. Si le code de calcul refuse le pas de temps donné, cette étape doit être re-exécutée avec un pas de temps plus court. De nouvelles « sorties calculées » sont alors produites à partir des « sorties validées ».
3. **La validation** consiste à faire définitivement passer le module dans son nouvel état valide. Les « sortie validées » prennent donc la valeur des « sortie calculées ».

Un module passe donc successivement par les étapes 1 puis 2, en boucle, jusqu'à la fin de la simulation. L'étape 1 peut être répétée autant de fois que le pas de temps est refusé. Ceci est résumé dans l'illustration 3. Les rectangles représentent les données -donc l'état- du module et les ovales ses fonctions. Le numéro devant le nom des fonction indique l'ordre des opérations. L'étape d'initialisation est omise afin de simplifier l'illustration.

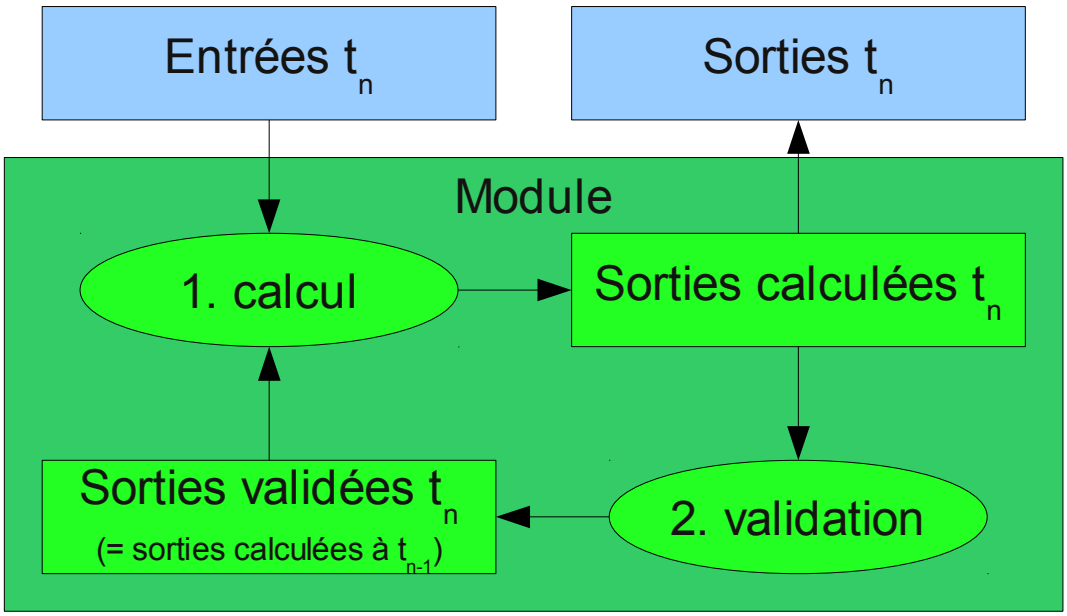

Illustration 3: algorithme de module

L'illustration 4 est une représentation sous la forme de graphe « états-transitions » (issu du langage UML) de l'algorithme de module.

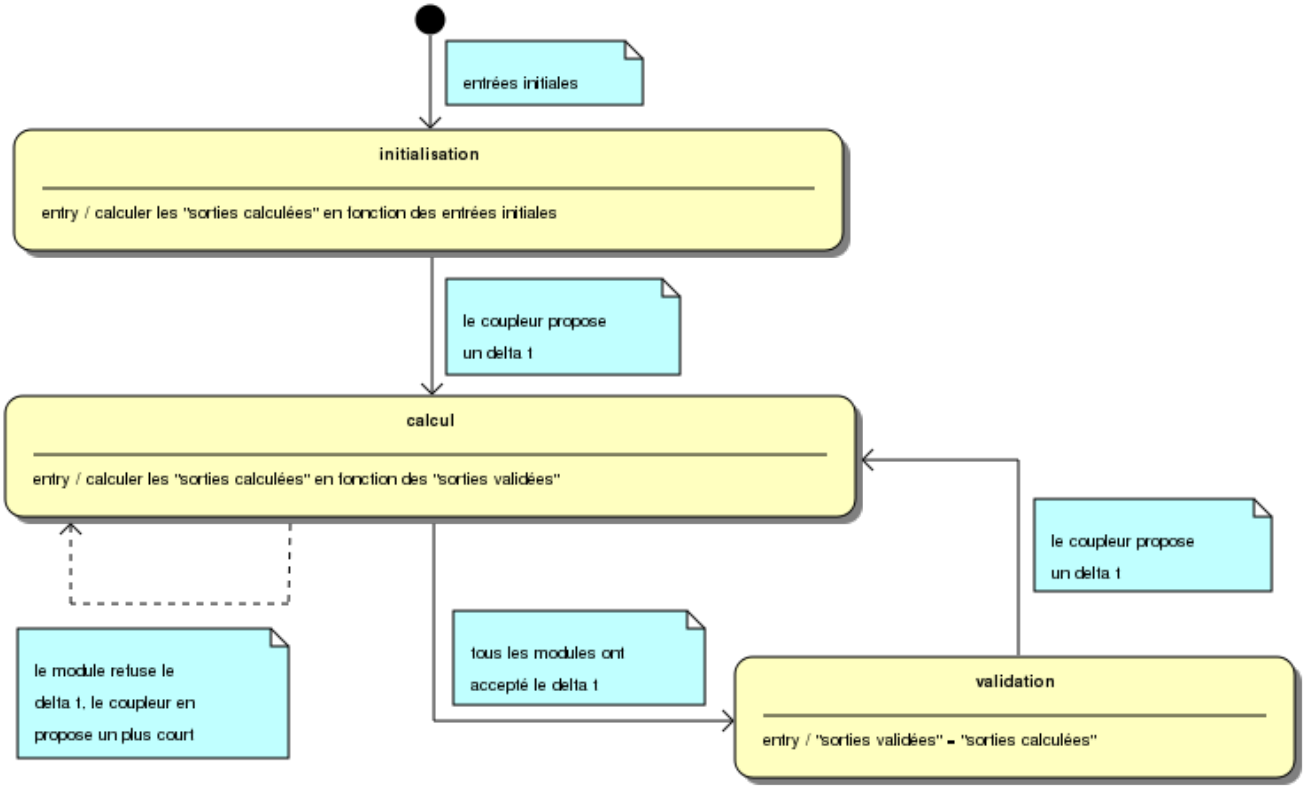

Illustration 4: diagramme UML états-transitions de l'algorithme de module

4.5.2 L'algorithme de modèle

Un modèle est un assemblage -ordonné- de modules. Il doit donc exécuter l'algorithme de chacun de ces modules (voir section précédente). En termes algorithmiques, cela passe par les trois étapes suivantes :

1. Initialisation -ordonnée- de chacun des modules. Les sorties des modules en amont sont récupérées pour être données aux modules en aval.
2. Choix d'un pas de temps et exécution -ordonnée- du code de calcul de chacun des modules. Les sorties des modules en amont sont récupérées pour être données aux modules en aval. Si un module refuse le pas de temps, l'exécution reprend au début avec un pas de temps plus court.
3. Si tous les modules acceptent le pas de temps, valider chacun des modules, passer au temps suivant et continuer à l'étape 2.

5 Licences logicielles

5.1 Introduction

Définir les droits d'utilisation des composants de la plateforme avant tout choix conceptuel permet d'éviter de se voir imposer -via les licences des outils utilisés- un type de licence non souhaité et potentiellement incompatible avec les attentes. Une analyse des attentes -exprimées par le groupe projet- au niveau licences est détaillée dans ce chapitre. Ces attentes ont été discutées avec une juriste de l'INRA et la personne chargée de la valorisation dans le département EA[19].

5.2 Décomposition par logiciels et utilisateurs

5.2.1 Les différentes parties logicielles

Afin de simplifier ce sujet complexe des licences logicielles, focalisons nous sur les parties publiques -c'est à dire largement diffusées- de la plateforme « sol virtuel ». De ce point de vue, la plateforme est un logiciel à installer sur le poste de travail de l'utilisateur. Le logiciel comprend un atelier d'assemblage -sous forme de binaire exécutable- accompagné de briques logicielles -sous forme de codes sources- qui correspondent aux modules -processus du sol- disponibles.

Les principales sorties de ce logiciel sont les modèles, assemblages « carrossés » -c'est à dire agrémentés d'une interface graphique homme-machine dédiée- de modules choisis par l'utilisateur. Les modèles seront diffusés de deux façons :

- Dans la grande majorité des cas, sous forme de binaires exécutables.
- Dans des cas précis -par exemple pour faciliter le couplage avec un logiciel tiers-, sous forme de « code source ».

L'atelier logiciel permet à l'utilisateur de modifier le code d'un module, pour corriger des défauts, changer un formalisme ou l'améliorer. L'atelier d'assemblage permet aussi d'exporter le « code source » de chaque module.

La plateforme est constituée de cinq types de parties logicielles possibles, binaires exécutables ou codes sources. A cela se rajoute, les sorties que peuvent générer les modèles issus de la plateforme. Cela donne donc :

a) le binaire exécutable de l'atelier d'assemblage
b) les codes sources des modules dans l'atelier d'assemblage
c) les codes sources des modules exportés, à partir de l'atelier d'assemblage
d) les binaires exécutables des modèles, avec des interfaces graphiques dédiées
e) les codes sources des modèles, sans interfaces graphiques homme-machine
f) les sorties provenant des modèles (cas d ou e)

19 Sophie Perin et Chantal Bailly respectivement.

5.2.2 Les différents types d'utilisateurs

La plateforme fait la distinction entre différentes cibles -types d'utilisateurs- potentielles des logiciels et données. A chacune de ces cibles est affectée une priorité/importance (1 pour la plus forte) ainsi que les usages principaux et secondaires pressentis.

La première famille de cibles comprend les scientifiques. C'est à dire les utilisateurs dont l'objectif est la recherche scientifique. Il n'y a donc pas d'objectif commercial. La seconde famille comprend toutes les autres cibles. Par opposition à la première cible, l'usage est potentiellement à but lucratif.

Priorité	Cible (type d'utilisateur)	Usage principal	Usage secondaire
1.	Scientifiques du département EA	a, b et d	c et e
2.	Scientifiques INRA	d	a, c et e
3.	Scientifiques autres EPST et universitaires	d	c et e
4.	Scientifiques au niveau international	d	e
5.	Instituts techniques	f	
6.	Partenaires privés dans le cadre de projets de collaborations (ex : projets Européens...)	f	
7.	Entreprises privées	f	

L'objectif principal de la plateforme « sol virtuel » est de favoriser la mutualisation des modules, donc l'apport de modules dans l'atelier d'assemblage. La plateforme souhaitera aussi servir un grand nombre d'utilisateurs scientifiques.

5.3 Les droits concédés

Pour chaque partie logicielle, il convient d'établir les scénarios envisagés quant aux droits concédés, cela servira directement au choix et à l'élaboration des licences.

5.3.1 Atelier d'assemblage

L'atelier d'assemblage concerne exclusivement les scientifiques. A ce titre, ils sont les seuls à pouvoir l'utiliser. Le droit d'utilisation est concédé uniquement à la personne déclarée. Il peut installer l'atelier sur différents ordinateurs, pour son seul usage.

La diffusion du logiciel est possible uniquement par le site Web de la plateforme « sol virtuel ». Avant de pouvoir télécharger l'atelier, les utilisateurs devront s'enregistrer. Seules des informations d'ordre général -nom, email, etc- seront demandées. Ces informations rentreront dans le cadre de la convention que l'INRA a passée avec la Cnil[20]. Les accords sur les différentes licences -des différentes parties logicielles de la plateforme- se feront à l'occasion de cet enregistrement.

L'usage des données qui ont un caractère nominatif sera conforme à la règlementation en vigueur et aux recommandations de la Commission Nationale de l'Informatique et des Libertés (Cnil). En particulier, ces informations ne seront utilisées qu'à des fins strictement professionnelles. La loi française interdisant toute utilisation à des fins commerciales ou publicitaires.

20 Commission Nationale Informatique et Libertés (http://www.cnil.fr/)

5.3.2 Codes sources des modules dans l'atelier

Les codes sources des modules sont diffusés avec l'atelier d'assemblage. Ils peuvent aussi être téléchargés à part, par exemple dans le cadre d'une mise à jour qui les concernent seuls. Dans les deux cas, ils sont soumis aux mêmes droits -d'utilisation et de diffusion- que l'atelier d'assemblage.

Les codes sources des modules sont modifiables uniquement par les utilisateurs de l'atelier. Un scientifique peut donc améliorer un module, mais exclusivement pour ses besoins propres. Ses modifications doivent respecter les auteurs originaux. C'est à dire que la nouvelle version du module doit afficher clairement le nom des auteurs précédents et leurs organismes. Le module contient donc la liste de tous les contributeurs scientifiques successifs, par ordre chronologique.

Les auteurs de modifications sont encouragés à proposer leurs améliorations à la plateforme « sol virtuel », à des fins de mutualisation par la plateforme. Si la direction de la plateforme juge les améliorations intéressantes, ils pourront ainsi en faire profiter les autres scientifiques, dès la diffusion de la nouvelle version du module.

Chaque participant travaillant sur un module concède un accord de licence qui donne explicitement aux représentants du projet le droit d'utiliser le code du participant. Cet accord de licence ne vaut pas transfert de droit d'auteur.

5.3.3 Codes sources des modules exportés

Les codes sources des modules exportés peuvent être utilisés dans n'importe quel contexte de recherche. Ils concernent donc exclusivement les acteurs scientifiques. Il n'y a donc pas de profits possibles, que se soit avec le code source ou un code tiers l'utilisant.

A l'instar des codes sources des modules de l'atelier, les codes sources des modules exportés peuvent être modifiées, en respectant les auteurs originaux.

La diffusion des codes sources des modules améliorés est laissée libre, du moment qu'elle respecte les auteurs originaux et reste dans un cercle scientifique et à but non lucratif.

A l'instar des codes sources des modules dans l'atelier, les auteurs de modifications sont encouragés à proposer leurs améliorations à la plateforme « sol virtuel ».

Les sortie des modules peuvent être utilisée par des tiers pour un usage à but potentiellement lucratif.

5.3.4 Modèles exécutables

A l'instar des codes sources des modules exportés, les modèles exécutables peuvent être utilisés dans n'importe quel contexte de recherche. Ils concernent donc exclusivement les acteurs scientifiques. Il n'y a donc pas de profits possibles. Par contre, les données issues de simulations avec ces modèles peuvent être utilisées par des tiers pour un usage à but potentiellement lucratif.

5.3.5 Codes sources des modèles

Les codes sources des modèles peuvent être utilisés, modifiés et diffusés de la même manière que les codes sources des modules dans l'atelier (4.2).

5.4 La licence retenue

En conclusion des discussions avec les acteurs concernés, la diffusion logicielle de la plateforme sous une licence libre -de type GPL, LGPL ou BSD- n'est pas possible. L'étude d'une licence spécifique INRA est en cours d'élaboration. Elle devrait s'inspirer très largement de la licence prévue pour la diffusion du code source du modèle Pastis, actuellement développé dans l'UMR EMMAH (INRA Avignon). Dès à présent, l'usage d'outils logiciels (librairies) avec des licences libres mais non contaminantes -type LGPL ou BSD- est recommandé.

Ci-dessous la licence Pastis, établie en 2009. En jaune apparaissent les parties spécifiques de la licence « sol virtuel » :

Contrat de licence d'utilisateur final académique

avec accès au code source

Définitions :

- *Logiciel* signifie le contenu du fichier téléchargé ou de tout autre support (CD, disquette, …) fourni avec ce contrat et incluant :
  - le logiciel (programme d'ordinateur) ;
  - les données de l'INRA et / ou de tiers ;
  - la présentation de ces données ;
  - la documentation associée (fichiers explicatifs ou tout autre document écrit) ;
  - toutes mises à jours, versions modifiées, ajouts, copies.
- *Utilisation* signifie l'accès au logiciel, son téléchargement, son installation, son exécution, sa copie ou tout autre avantage résultant de l'utilisation de ses fonctions.
- *INRA* signifie l'Institut National de la Recherche Agronomique, Paris, France ainsi que toute(s) filiale(s) de l'INRA et/ou tout copropriétaire ou coauteur de droits sur le logiciel.
- *Utilisateur final académique* signifie le bénéficiaire de la présente licence qui est employé par un organisme sans but lucratif et ayant pour finalité de conduire des travaux de recherche ou d'enseignement et désigné ci-après par « Vous ».
- *Domaine de la licence* signifie des activités de recherche et développement en vue de la production de connaissances scientifiques ou l'enseignement. Est exclue du domaine toute utilisation à des fins commerciales, que ce soit pour fournir des prestations à des tiers ou vendre n'importe quel produit commercial, sauf accord spécifique et écrit de l'INRA.

1) Droits de Propriété Intellectuelle

a) Le logiciel ainsi que toute copie de celui-ci sont la propriété intellectuelle de l'INRA. La structure, l'organisation et le code source constituent des secrets commerciaux et des informations confidentielles que Vous n'êtes pas autorisés à utiliser ou à divulguer, sauf aux fins mentionnées dans le présent contrat.

b) Le logiciel est protégé par le code de la propriété intellectuelle en France et à l'étranger par les conventions internationales sur le droit d'auteur. La violation d'un des droits de l'auteur de l'œuvre constitue un délit de contrefaçon sanctionné en France par l'article L335-2 du code de la propriété intellectuelle et punie de 2 ans d'emprisonnement et 150 000 €

d'amende.

c) L'auteur –INRA- se réserve tous les droits qui ne sont pas expressément cités dans le présent contrat de licence.

2) Licence du logiciel

a) INRA Vous concède par le présent contrat une licence non-exclusive, mondiale et gratuite d'utilisation du logiciel aux fins décrites ci-dessous, dans le domaine de la licence, tant que Vous respecterez les dispositions du présent contrat.

b) Vous êtes autorisés à télécharger, installer, exécuter le logiciel sur votre ordinateur à concurrence d'une licence pour un ordinateur (un disque dur).

c) Sauf accord spécifique et écrit de l'INRA, Vous n'êtes pas autorisés à télécharger, installer ou exécuter le logiciel sur un serveur.

d) Vous n'êtes pas autorisés à louer, vendre, sous-louer, distribuer, céder, transférer, licencier, sous-licencier ou autrement partager le logiciel ou l'un des droits de l'INRA sur le logiciel.

e) Vous êtes autorisés à faire une copie de sauvegarde du logiciel à condition de ne pas l'utiliser sur aucun autre ordinateur. Vous êtes responsable de la sécurité physique de cette copie et du logiciel.

f) Vous pouvez personnaliser l'installation du logiciel ou étendre ses fonctions. De même, Vous pouvez traduire, adapter, arranger, modifier le logiciel lorsque ces actes sont nécessaires pour permettre l'utilisation du logiciel conformément au présent contrat.

g) INRA se réserve le droit de corriger les erreurs et de déterminer les modalités particulières auxquelles seront soumis les actes suivants :

La reproduction du code source du logiciel ou la traduction de ce code sont autorisées lorsqu'elles sont indispensables pour obtenir l'interopérabilité du logiciel avec d'autres logiciels et lorsqu'elles sont indispensables pour utiliser le logiciel dans le domaine de la licence.

Par ailleurs, Vous n'êtes pas autorisés à faire de l'ingénierie inverse, décompiler ou désassembler le logiciel.

Les informations nécessaires à l'interopérabilité sont disponibles auprès de l'INRA sans que cela implique pour l'INRA un engagement de fournir une assistance ou des services associés au logiciel.

h) Vous n'êtes pas autorisés à extraire (c'est-à-dire transférer de manière permanente ou temporaire) tout ou partie du contenu du logiciel (données) sur un autre support, par tout moyen et sous toute forme. De même, vous n'êtes pas non plus autorisés à réutiliser (c'est-à-dire mettre à disposition du public) tout ou partie du contenu du logiciel (données) quelle qu'en soit la forme.

i) Vous êtes autorisés à produire des modules complémentaires au logiciel, améliorer ou adapter le logiciel à condition toutefois que l'utilisation qui sera faite de ces modules, version complémentaires, améliorations, ajouts ou mises à jour soient utilisés conformément à la présente licence et dans le strict domaine de la licence.

En particulier, vous vous engagez à communiquer ces améliorations et modules à INRA et à l'autoriser à les utiliser pour ses besoins propres de recherche.

Si les améliorations ou modules sont dépendants du logiciel (conformément au code de la propriété intellectuelle), Vous ne pouvez pas les divulguer ou transférer à des tiers, sans accord préalable et écrit de l'INRA sur les modalités de cette distribution.

Si les améliorations ou modules ne sont pas dépendants du logiciel (conformément au code de la propriété intellectuelle), Vous pouvez les divulguer et les transférer à des tiers, à condition de garantir et de couvrir INRA contre toute responsabilité en cas de procès ou d'action intentée par un tiers.

3) Garantie limitée

a) Vous reconnaissez que l'état actuel des connaissances scientifiques et techniques ne permettent pas de tester et de vérifier toutes les utilisations du Logiciel, ni de détecter l'existence d'éventuels défauts. Vous reconnaissez que le changement, l'utilisation, la modification, le développement et la reproduction du Logiciel sont réservés à des utilisateurs avertis et comportent des risques. Vous êtes responsables du contrôle par tous moyens de l'adéquation du Logiciel à vos besoins, du contrôle de son fonctionnement, de son utilisation dans des conditions qui ne causent pas de dommages aux personnes et aux biens.

b) Le Logiciel est fourni « en l'état », sans aucune autre garantie que celle de l'existence du Logiciel, tant expresse que tacite, ceci incluant toute exclusion de garantie quant à un titre (de propriété , d'exploitation), l'absence de contrefaçon, la qualité marchande, le caractère sécurisé, innovant ou pertinent du Logiciel, l'absence d'erreur, la compatibilité avec Votre équipement et/ou configuration logicielle.

4) Exclusions de responsabilité

a) INRA ne peut être tenu pour responsable envers quiconque :

i. de tout dommage dû à l'inexécution totale ou partielle de Vos obligations ;

ii. de tout dommage direct ou indirect découlant de l'utilisation ou des performances du Logiciel subis par l'Utilisateur final lorsqu'il s'agit d'un professionnel utilisant le Logiciel à des fins professionnelles ;

iii. de tout dommage indirect découlant de l'Utilisation ou des performances du Logiciel.

b) Les parties conviennent expressément que tout préjudice financier ou commercial (par exemple : pertes de données, perte de clientèle ou de commandes, perte de bénéfices, perte d'exploitation, manque à gagner, trouble commercial) ou toute action dirigée contre Vous par un tiers constitue un dommage indirect et ne saurait ouvrir droit à réparation par l'INRA.

5) Fin du contrat

En cas de manquement de l'utilisateur final académique à l'une des dispositions du présent contrat, la licence sera résiliée automatiquement. En cas de résiliation, Vous devez immédiatement détruire le logiciel ainsi que toute copie, adaptation, amélioration ou toute partie du logiciel qui serait incluse dans un autre logiciel.

6) Loi applicable

Le présent contrat ainsi que tout litige naissant de l'exécution ou de l'interprétation de la licence seront soumis à a loi française.

7) Contestations

Les parties s'engagent à résoudre leurs différends à l'amiable. En cas de désaccord persistant, les tribunaux de Paris seront compétents.

6 Début de la phase opérationnelle

Répondre aux objectifs affectés à « sol virtuel » demande de relever des défis tout autant informatiques que scientifiques. La tâche qui attend à la fois le groupe et l'équipe projet est immense.

Coté informatique, avant de rentrer dans une phase opérationnelle, il apparait important pour cette année 2010 -qui est la deuxième année d'existence de l'équipe informatique de « sol virtuel »- de faire des choix qui privilégient de valider les besoins et les concepts rapidement. Une fois les démonstrations validées, la plateforme pourra rentrer dans une phase qui la rendra plus opérationnelle. Cela concerne chaque grande partie de la plateforme :

- A la partie 1, c'est vérifier un mode de gestion des processus et des squelettes.
- A la partie 2, c'est tester si la mise en œuvre d'un système de couplage simple est suffisant.
- A la partie 3, c'est estimer la pertinence de concevoir des interfaces graphiques automatiques de modèles.

Rapidement par la suite, il conviendra de mettre en œuvre un système de création simple de modules, rendant le scientifique autonome sur ces aspects.