



HAL
open science

Conception et réalisation d'une interface unifiée et automatisée pour la plateforme de simulation de l'expérience CONSERT

Philippe Bollard

► **To cite this version:**

Philippe Bollard. Conception et réalisation d'une interface unifiée et automatisée pour la plateforme de simulation de l'expérience CONSERT. Interface homme-machine [cs.HC]. 2011. dumas-00592165

HAL Id: dumas-00592165

<https://dumas.ccsd.cnrs.fr/dumas-00592165v1>

Submitted on 11 May 2011

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

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



CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE RÉGIONAL RHÔNE-ALPES
CENTRE D'ENSEIGNEMENT DE GRENOBLE

MÉMOIRE

présenté par Philippe BOLLARD

en vue d'obtenir

LE DIPLÔME D'INGÉNIEUR C.N.A.M.

en INFORMATIQUE

Conception et réalisation d'une interface unifiée et automatisée pour la plateforme de simulation de l'expérience CONSERT

Soutenu le 8 avril 2011

JURY

Président : M. Eric Gressier-Soudan

Membres : M. Jean-Pierre Giraudin

M. André Plisson

M. Mathias Voisin-Fradin

Tuteurs : M. Stéphane Chaillol

M. Alain Hérique

M. Gérard Zins



CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE RÉGIONAL RHÔNE-ALPES
CENTRE D'ENSEIGNEMENT DE GRENOBLE

MÉMOIRE

présenté par Philippe BOLLARD

en vue d'obtenir

LE DIPLOME D'INGÉNIEUR C.N.A.M.

en INFORMATIQUE

Conception et réalisation d'une interface unifiée et automatisée pour la plateforme de simulation de l'expérience CONSERT

Soutenu le 8 avril 2011

Les travaux relatifs à ce mémoire ont été effectués au Laboratoire de Planétologie de Grenoble sous la direction de M. Stéphane Chaillol, tuteur de stage, et de M. Alain Hérique, référent scientifique.

Remerciements

Je remercie tout d'abord les membres du jury : M. Éric Gressier-Soudan, professeur au CNAM Paris, M. Jean-Pierre Giraudin, professeur à l'université Pierre Mendès France de Grenoble ainsi que M. André Plisson et M. Mathias Voisin-Fradin, respectivement directeur et directeur-adjoint du centre d'enseignement CNAM de Grenoble.

Je remercie particulièrement mon tuteur, M. Stéphane Chaillol ainsi que les membres de l'équipe RADAR, M. Alain Hérique et M. Wlodek Kofman ainsi que Mme Sonia Zine. Ceux-ci m'ont accueilli au sein du projet CONSERT et m'ont ainsi permis de découvrir le contexte d'un laboratoire de recherche. Je remercie aussi M. Gérard Zins et M. Bruno Bzeznik pour leur participation active et leur aide lors du projet.

Je salue également tout le personnel du LPG, permanents et non-permanents, thésards ou stagiaires... C'est une véritable petite famille très chaleureuse. Comme tout nouvel arrivant, j'ai bénéficié des précieux conseils d'Akila, de Béatrice ou encore d'Odile. Merci à ces « petites mamans » ! Je ne peux oublier mes collègues du « Geek center » : Ludovic, Damien, Rémy et par extension Guillaume, Sylvain, Laurent, Richard, Ghislain.

Je remercie mes amis de l'AICNAM-PST, en particulier Éric et mon parrain Dominique, pour leur précieuse aide quant à la relecture et à la préparation de la soutenance. J'ai aussi une pensée pour Christine qui doit maintenant s'éloigner après sa thèse : c'est promis, j'essayerai d'être un aussi bon secrétaire que toi :)

Je remercie Patrick, un fidèle relecteur mais avant tout un ami et mon premier employeur. Il a su me guider dans le monde professionnel tout en me laissant suffisamment de liberté pour expérimenter de nouvelles techniques. Je lui suis tout aussi reconnaissant de m'avoir encouragé et laissé le temps de poursuivre ma formation d'ingénieur au CNAM.

Je salue ma famille et tous mes amis qui m'ont soutenu et motivé pendant ces dernières années. Un grand merci à Laurie, Dominique et Marie, Fabrice et Mireille, les « Mousquetaires du TER », Frédéric, Fabrice et Sabine, Nicolas et bien d'autres... J'associe ma voix à celles de mes compagnons du chœur Entresol que j'ai hâte de retrouver.

Enfin, je dédie ce mémoire à mon grand-père, le Gadz' Arts « Zabols » qui est parti malheureusement trop tôt pour me voir reprendre le flambeau familial des Arts et Métiers.

Avant propos	v
Sommaire	xi
Liste des acronymes	xiii
Table des figures	xvii
Introduction et contexte du stage	1
1 Données de simulation	7
1.1 Présentation des données	8
1.1.1 Modèle d'orbite	8
1.1.2 Modèle de forme de noyau	8
1.1.3 Modèle de structure de noyau	9
1.1.4 Résultats de simulation	9
1.2 Problématiques et objectifs	10
1.3 Solutions proposées	12
1.3.1 Langage XML	12
1.3.1.1 Structure d'un fichier XML	12
1.3.1.2 Lecture des données et chargement mémoire	13
1.3.1.2.1 DOM	14
1.3.1.2.2 SAX	14
1.3.1.3 Validation des données	15
1.3.1.3.1 DTD	15
1.3.1.3.2 XSD	15
1.3.2 Base de données embarquée SQLite	17
1.4 Évaluation des solutions	18
1.5 Bilan	22
2 Étude de la plateforme de simulation existante	25
2.1 Architecture de la plateforme	25
2.1.1 Éléments d'infrastructure	25
2.1.1.1 Poste utilisateur	25
2.1.1.2 Serveur stockage	26

2.1.1.3	Environnement de calcul intensif	26
2.1.1.3.1	Grappes de calcul	26
2.1.1.3.2	Grilles de calcul	26
2.1.1.3.3	Grille locale du projet CIMENT	27
2.1.2	Logiciels	28
2.1.2.1	Outil de simulation	28
2.1.2.2	Outil de visualisation	29
2.2	Chaîne de simulation	30
2.2.1	Étape n°1 : déploiement des données sources sur la grappe	30
2.2.2	Étape n°2 : lancement du calcul sur la grappe	30
2.2.2.1	Initialisation de l'environnement de calcul	30
2.2.2.2	Paramétrage global du logiciel de simulation	32
2.2.2.3	Paramétrage spécifique du logiciel de simulation	32
2.2.2.4	Lancement du logiciel de simulation	34
2.2.2.5	Contrôle de l'avancement du calcul	35
2.2.2.6	Rapatriement des résultats	35
2.2.3	Étape n°3 : interprétation des résultats	35
2.3	Problématiques et objectifs	36
3	Grilles de calcul : état de l'art	37
3.1	Quelques grilles de calcul	37
3.1.1	Grid'5000	37
3.1.2	EGI	37
3.1.3	Globus	38
3.1.4	NorduGrid	38
3.2	Quelques intergiciels de grilles de calcul	38
3.2.1	OAR	38
3.2.1.1	Lancement de tâche	39
3.2.1.2	Contrôle de l'exécution	39
3.2.1.3	Arrêt de tâche	40
3.2.2	CiGri	40
3.2.2.1	Lancement de campagne	41
3.2.2.2	Contrôle de l'exécution	41
3.2.2.3	Arrêt de campagne	42
3.2.2.4	Interface web	42
3.2.3	gLite	44
3.2.4	Globus Toolkit	44
3.2.5	ARC	44
3.3	Quelques outils complémentaires de gestion de grilles	46
3.3.1	OARgrid	46
3.3.2	GRUDU	46
3.3.3	g-Eclipse	46
3.4	Bilan	48
4	Réalisation de la nouvelle plateforme de simulation	49
4.1	Méthodologie	49
4.1.1	Déroulement du projet	49
4.1.2	Outils et techniques utilisés	49
4.2	Conception de la plateforme	50
4.2.1	Conception de l'architecture globale	50

4.2.1.1	Poste client	50
4.2.1.2	Serveur web	51
4.2.1.3	Serveur de stockage	51
4.2.1.4	Grille de calcul	51
4.2.2	Modélisation de l'architecture de l'application	51
4.2.2.1	Tâche	51
4.2.2.1.1	Commande	53
4.2.2.1.2	Transfert	53
4.2.2.1.3	Envoi	54
4.2.2.1.4	Réception	54
4.2.2.1.5	Job CiGri	54
4.2.2.1.6	Attente	54
4.2.2.1.7	Fin	54
4.2.2.2	Séquence	54
4.2.2.3	Processus	55
4.2.2.4	Variable	55
4.2.2.5	Hôte	56
4.2.2.5.1	Grille	56
4.2.2.5.2	Grappe	56
4.2.2.5.3	Nœud	56
4.2.3	Base de données	56
4.3	Implémentation de l'architecture logicielle	58
4.3.1	Choix techniques	58
4.3.1.1	Langage PHP	58
4.3.1.2	Cadriciel Zend Framework	58
4.3.2	Initialisation du projet	59
4.3.2.1	Mise en place du framework	59
4.3.2.2	Création du projet	59
4.3.2.3	Description de l'arborescence	60
4.3.2.4	Configuration du framework	61
4.3.3	Modèle de données	62
4.3.3.1	Modèle de bas niveau	62
4.3.3.2	Modèle de haut niveau	62
4.3.3.3	Couche d'adaptation	62
4.3.4	Contrôleurs	63
4.3.5	Vues	63
4.3.5.1	Gestion des droits et des utilisateurs	63
4.3.5.2	Navigation	63
4.3.5.3	Internationalisation	64
4.3.6	Automatisation	64
4.4	Implémentation des couches de communication	64
4.4.1	Communication entre le serveur applicatif et les hôtes	64
4.4.1.1	Ouverture de connexion	65
4.4.1.2	Exécution de commandes	66
4.4.1.3	Gestion de fichiers	66
4.4.1.4	Fermeture de connexion	66
4.4.2	La synchronisation entre les parties serveur et client	66
4.4.3	Gestion des ressources de calcul	67
4.5	Bilan	67

5	Utilisation de la nouvelle plateforme de simulation	69
5.1	Principaux cas d'utilisation	69
5.2	Utilisation de l'interface	73
5.2.1	Connexion	73
5.2.2	Hôte	74
5.2.2.1	Création d'un hôte	74
5.2.2.2	Liste des hôtes	74
5.2.2.3	Détails d'un hôte	75
5.2.3	Projet	77
5.2.3.1	Création d'un projet	77
5.2.3.2	Liste des projets	77
5.2.3.3	Détails d'un projet	77
5.2.4	Fichier	80
5.2.4.1	Création d'un fichier	80
5.2.4.2	Liste des fichiers	80
5.2.4.3	Détails d'un fichier	80
5.2.5	Séquence	82
5.2.5.1	Création d'une séquence	82
5.2.5.2	Liste des séquences	82
5.2.5.3	Détails d'une séquence	83
5.2.6	Tâche	85
5.2.6.1	Création d'une tâche	85
5.2.6.2	Liste des tâches	85
5.2.6.3	Détails d'une tâche	86
5.2.7	Variable	88
5.2.7.1	Création d'une variable	88
5.2.7.2	Liste des variables	89
5.2.7.3	Détails d'une variable	89
5.2.7.4	Surcharge d'une variable	90
5.2.8	Processus	90
5.2.8.1	Création d'un processus	90
5.2.8.2	Liste des processus	90
5.2.8.3	Détails d'un processus	91
5.2.9	Utilisateur	93
5.2.9.1	Création d'un utilisateur	93
5.2.9.2	Liste des utilisateurs	93
5.2.9.3	Détails d'un utilisateur	94
5.3	Mise en œuvre de la simulation CONSERT	95
5.3.1	Amélioration des scripts Cigri	95
5.3.1.1	Génération de la liste de grappes	95
5.3.1.2	Génération du fichier JDL	96
5.3.1.3	Déploiement sur les grappes	96
5.3.1.4	Nettoyage des fichiers	96
5.3.2	Amélioration des scripts Consert	97
5.3.2.1	Génération du fichier params.prg	97
5.3.2.2	Génération du fichier params.txt	97
5.3.2.3	Compilation et déploiement de l'outil de simulation sur les grappes	97
5.3.2.4	Lancement d'une tâche sur une grappe	98
5.3.2.5	Récupération et concaténation des résultats	98
5.3.3	Paramétrage de l'interface	98

5.3.3.1	Initialisation de l'interface	98
5.3.3.2	Sous-chaîne de déploiement de l'environnement de calcul	99
5.3.3.3	Sous-chaîne de calcul de la simulation	99
5.4	Bilan	100
Conclusion		101
Annexes		105
A	Diagramme de Gantt	106
B	Scripts Bash	107
B.1	Scripts Cigri	107
B.1.1	Génération de la liste des clusters	107
B.1.2	Génération du JDL	108
B.1.3	Déploiement sur les clusters	109
B.1.4	Nettoyage	110
B.2	Scripts Consert	111
B.2.1	Génération du fichier params.prg	111
B.2.2	Génération du fichier params.txt	112
B.2.3	Compilation et déploiement de l'outil de simulation sur les clusters	113
B.2.4	Lancement d'une tâche sur un cluster	115
B.2.5	Récupération et concaténation des résultats	116
C	Extraits de code source	117
C.1	Fichier de configuration	117
C.2	Fichier de bootstrap	117
C.3	Classe processus	118
C.4	Classe fabrique d'objets processus	125
C.5	Classe d'adaptation entre le modèle de données et le modèle objet pour les processus	129
C.6	Contrôleur de processus	131
C.7	Ressources, rôles et droits des utilisateurs	139
C.8	Structure XML pour la navigation	141
Glossaire		143
Bibliographie		147

Liste des acronymes

API	Application Programming Interface
CERN	Conseil Européen pour la Recherche Nucléaire
CiGri	CIMENT Grid
CIMENT	Calcul Intensif, Modélisation, Expérimentation Numérique et Technologique
CIRA	Calcul Intensif en Rhône Alpes
CNES	Centre National d'Études Spatiales
CNRS	Centre National de la Recherche Scientifique
CONSERT	COmet Nucleus Sounding ExpeRimenT
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DIET	Distributed Interactive Engineering Toolbox
DOM	Document Object Model
DTD	Document Type Definition
EGEE	Enabling Grids for E-science
EGI	European Grid Infrastructure
ESA	European Space Agency
GRUDU	Grid'5000 Reservation Utility for Deployment Usage
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
INPG	Institut National Polytechnique de Grenoble
INRIA	Institut National de Recherche en Informatique et en Automatique
IPAG	Institut de Planétologie et d'Astrophysique de Grenoble
ISO	International Organization for Standardization
JDL	Job Description Language

JS	JavaScript
LAOG	Laboratoire d'AstrOphysique de Grenoble
LCG	LHC Computing Grid
LHC	Large Hadron Collider
LPG	Laboratoire de Planétologie de Grenoble
MENRT	Ministère de l'Éducation Nationale, de la Recherche, et de la Technologie
MVC	Modèle-Vue-Contrôleur
NASA	National Aeronautics and Space Administration
OSUG	Observatoire des Sciences de l'Univers de Grenoble
PHP	PHP : Hypertext Preprocessor
RENATER	RÉseau NATional de télécommunications pour la Technologie, l'Enseignement et la Recherche
REST	Representational State Transfer
RPC	Remote Procedure Call
SAX	Simple API for XML
SGBD	Système de Gestion de Base de Données
SGBDR	Système de Gestion de Base de Données Relationnel
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SSH	Secure Shell
SSHFS	Secure Shell File System
SVG	Scalable Vector Graphics
UJF	Université Joseph Fourier
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UTF	Unicode Transformation Format
W3C	World Wide Web Consortium
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSLT	eXtensible Stylesheet Language Transformations

Table des figures

1	Situation des deux modules de l'instrument CONSERT	4
2	Vitesse de propagation des ondes en présence d'une comète	4
3	Différents parcours de rayons au sein d'une comète	5
4	Étapes et éléments intervenant dans la simulation CONSERT	7
5	Exemple de représentation graphique d'une orbite	8
6	Modélisation d'un noyau de comète en harmoniques sphériques	9
7	Structure cométésimale d'un noyau	10
8	Comparaison d'architectures de SGBD	18
9	Modèle de classes UML utilisé par les différents prototypes	19
10	Principe de fonctionnement du prototype n°1	19
11	Maillage cubique de l'espace englobant l'orbite	20
12	Principe de fonctionnement du prototype n°2	20
13	Principe de fonctionnement du prototype n°3	21
14	Modèle de base de données utilisé dans le prototype n°4	22
15	Vue simplifiée de l'architecture actuelle de la plateforme de simulation de l'expérience CONSERT	25
16	L'architecture d'une grappe	27
17	L'architecture d'une grille	27
18	Les grappes de CiGri/RaGrid	28
19	Exemple de visualisation d'une simulation avec ondes propagées	29
20	Architecture de la plateforme actuelle de simulation	30
21	Architecture de l'intergiciel OAR	39
22	Évolution de la charge d'une grappe	40
23	Liste des campagnes d'un utilisateur de CiGri	43
24	Informations sur une campagne, paramètres d'exécution	43
25	Informations sur une campagne, liste des tâches	44
26	Architecture de Globus Toolkit	45
27	Architecture de l'intergiciel ARC	45
28	Utilisation de l'outil GRUDU	47
29	Utilisation de l'environnement g-Eclipse	47
30	Architecture globale de la nouvelle plateforme	50
31	Diagramme partiel de classes	52
32	Diagramme d'états d'une tâche	53
33	Diagramme d'états d'un processus	55
34	Modélisation du schéma de la base de données	57

35	Schéma de fonctionnement d'un script PHP	58
36	Diagramme partiel de l'ensemble des cas d'utilisation	70
37	Diagramme partiel des cas d'utilisation des utilisateurs	71
38	Diagramme partiel des cas d'utilisation communs	71
39	Diagramme partiel des cas d'utilisation des processus	72
40	Formulaire d'identification	73
41	Page d'accueil et ergonomie générale	73
42	Formulaire de création d'un hôte de type Grille	74
43	Champs complémentaires pour un hôte	74
44	Liste des hôtes	75
45	Liste des informations principales d'un hôte	75
46	Liste des grappes ou nœuds rattachés à un hôte	76
47	Liste des utilisateurs d'un hôte	76
48	Liste des projets d'un hôte	76
49	Formulaire de création d'un projet	77
50	Liste des projets	77
51	Informations principales d'un projet	78
52	Liste des séquences d'un projet	78
53	Liste des processus d'un projet	78
54	Liste des fichiers d'un projet	79
55	Liste des hôtes d'un projet	79
56	Liste des utilisateurs d'un projet	79
57	Formulaire de création de fichier	80
58	Liste des fichiers	81
59	Détails d'un fichier	81
60	Contenu d'un fichier	82
61	Formulaire de création d'une séquence de tâches	82
62	Liste des séquences	83
63	Informations principales d'une séquence	83
64	Liste des tâches d'une séquence	84
65	Liste des variables d'une séquence	84
66	Liste des processus d'une séquence	84
67	Formulaire de création d'une tâche de type commande	85
68	Champs complémentaires pour une tâche	86
69	Liste des tâches	87
70	Informations principales d'une tâche	87
71	Informations spécifiques d'une tâche	87
72	Liste des variables d'une tâche.	88
73	Formulaire de création d'une variable de séquence.	88
74	Champs complémentaires pour une variable de type Fichier.	89
75	Liste des variables.	89
76	Détails d'une variable	90
77	Surcharge d'une variable	90
78	Formulaire de création d'un processus	91
79	Liste des processus	91
80	Informations principales d'un processus	92
81	Liste des tâches d'un processus	92
82	Liste des variables d'un processus	93
83	Formulaire de création d'un utilisateur	93
84	Liste des utilisateurs	94

85	Informations principales sur un utilisateur	94
86	Liste des projets d'un utilisateur	95
87	Liste des processus d'un utilisateur	95
88	Liste des hôtes d'un utilisateur	95
89	Diagramme de GANTT	106

Laboratoire de Planétologie de Grenoble

Le Laboratoire de Planétologie de Grenoble (LPG), fondé en 1999 est situé sur le domaine universitaire de Saint-Martin d'Hères. Il est une unité de recherche du CNRS et de l'Université Joseph Fourier (UJF) ainsi qu'une composante de recherche de l'Observatoire des Sciences de l'Univers de Grenoble (OSUG) et de l'Unité de Formation et de Recherche en Physique de l'UJF. Il regroupe en moyenne une trentaine de personnes dont une vingtaine de permanents. Depuis le 1^{er} janvier 2011, le LPG a fusionné avec le Laboratoire d'AstrOphysique de Grenoble (LAOG) pour donner naissance à l'Institut de Planétologie et d'Astrophysique de Grenoble (IPAG). [1] [2]

La recherche effectuée au Laboratoire de Planétologie de Grenoble couvre un vaste domaine englobant l'étude des planètes, des comètes et des météorites du système solaire. Elle se fait dans le cadre de l'exploration du système solaire par les sondes spatiales lancées notamment par l'ESA et la NASA. Les chercheurs du laboratoire sont impliqués dans la préparation, l'opération, et l'interprétation des mesures réalisées par les missions telles que Cassini-Huygens vers Saturne et ses satellites, ROSETTA vers une comète, Mars Express et Venus Express vers nos planètes voisines.

Les moyens de recherche développés au laboratoire englobent l'observation des hautes atmosphères planétaires dont la Terre, l'analyse de données spatiales sur les surfaces et subsurfaces planétaires (Mars, Lune, Titan), la modélisation et la simulation en laboratoire de processus physiques et chimiques, l'analyse de matière extraterrestre et la conception d'instrumentation spatiale. Les objets du Système Solaire, les planètes, les satellites et les petits corps constituent les champs d'étude du LPG. Les recherches s'organisent autour de trois équipes thématiques et de deux axes transversaux.

L'équipe « Hautes Atmosphères Planétaires » étudie l'évolution, la composition et la dynamique des environnements spatiaux planétaires, en suivant plusieurs approches :

- chimie des atmosphères planétaires (expériences de laboratoire par spectrométrie de masse à très haute résolution et modélisation) ;
- caractérisation et quantification des sources d'énergies (électronique, ionique, cosmique, EUV) et leurs effets sur les hautes atmosphères planétaires. Sur Terre, cela s'inscrit dans la nouvelle discipline, la « météorologie de l'espace ».

L'équipe « Matière Moléculaire Solide du Système Solaire » étudie la surface des planètes et des satellites et la matière constituant les comètes et les météorites, qui nous renseignent sur l'origine et l'évolution ultérieure de ces objets. Les glaces en particulier participent aux grands cycles

qui conditionnent le climat de Mars à différentes échelles temporelles. Les minéraux hydratés en sont des témoins et acteurs. Les matériaux carbonés des astéroïdes et comètes reflètent leurs processus de formation, leur histoire et le degré de complexité atteint au cours du temps par la matière organique jusqu'à former parfois les briques élémentaires du vivant.

Enfin, l'équipe « Surfaces, Subsurfaces et noyaux Cométaires : Télédétection Radar » ouvre un champ nouveau d'investigation du système solaire. Étudier la structure interne du noyau de la comète 67P/Churyumov-Gerasimenko¹ à la faveur de la mission spatiale ROSETTA représente une étape clé pour percevoir les mécanismes de formation et d'altération de ces objets primitifs. Cartographier l'eau liquide ou solide dans le sous-sol martien représente un défi essentiel pour comprendre l'évolution de cette planète et pour quantifier ses réservoirs de glace d'eau. Ces travaux sont intimement liés à la recherche sur la propagation électromagnétique dans les milieux complexes.

Mission spatiale ROSETTA

Les comètes sont observées depuis de nombreux siècles mais ce n'est réellement que depuis l'avènement de la conquête spatiale que leurs mystères sont progressivement percés. Leur étude permet de mieux comprendre l'origine du système solaire. En effet, une comète est composée de matériaux primitifs de même nature que ceux ayant servi à sa formation. Il est même possible qu'une partie de la matière organique terrestre provienne d'une comète, ce qui pourrait expliquer l'apparition de la vie. L'élément principal d'une comète est son noyau solide de quelques kilomètres de diamètre. D'après les dernières observations, il s'agirait d'un objet poreux comportant des grains de silicates associés à des glaces HO, CO et CO₂ et des composés organiques plus ou moins complexes. En se rapprochant du Soleil, le noyau se réchauffe ce qui provoque la sublimation des glaces : elles s'échappent du noyau sous forme gazeuse en entraînant les poussières. Une enveloppe de gaz et de poussières se forme alors autour du noyau. C'est sa « chevelure ». Cependant, en orbitant relativement loin du soleil, les comètes se sont assez peu dégradées et restent de formidables sujets d'analyse.

En 1986, la comète 1P/Halley a été survolée par plusieurs sondes lors de son dernier passage près de la Terre. Parmi celles-ci, la sonde GIOTTO a analysé sa chevelure interne. Ayant grâce à GIOTTO la confirmation de l'existence d'un noyau cométaire, la communauté scientifique internationale envisage progressivement une nouvelle étude plus poussée. En 1993, l'ESA approuve la mission spatiale ROSETTA² dans le cadre de son programme « Horizon 2000 ». Elle a pour ambitieux objectif d'accompagner pendant plusieurs mois une comète et d'y faire atterrir un module d'analyse directement à sa surface. Avec les divers instruments développés pour l'occasion, il sera possible de mener une vingtaine d'expériences portant par exemple sur l'analyse des gaz, des poussières, ou encore l'étude de la structure du noyau.

Après divers aléas et changements de cible principale, la sonde ROSETTA fut lancée le 2 mars 2004 à destination de la comète 67P/Churyumov-Gerasimenko. Son voyage durera une dizaine d'années et sera ponctué par le survol des astéroïdes Steins en septembre 2008 et Lutetia en juillet 2010. En août 2014, la sonde approchera enfin de la véritable cible et se mettra en orbite à quelques kilomètres du noyau. La période d'observation devrait durer 18 mois. La sonde est composée de

1. Découverte en 1969 par K. Churyumov (Université de Kiev, Ukraine) et S. Gerasimenko (Institut d'Astrophysique Dushanbe, Tadjikistan), la comète 67P/Churyumov-Gerasimenko a un noyau d'environ 4 km de large. Elle orbite autour du soleil tous les 6,6 ans à une distance comprise entre 186 millions et 857 millions de kilomètres. [3].

2. Selon les scientifiques, cette mission doit « apporter la clé du système solaire ». Elle a donc été baptisée ROSETTA en référence à la pierre de Rosette, qui permit à Jean-François Champollion de décrypter les hiéroglyphes en 1822.

deux parties distinctes. L'orbiteur « ROSETTA », corps principal de la sonde, restera en orbite autour de la comète et permettra de la cartographier pendant la première phase d'observation. L'atterrisseur (ou « lander ») nommé « Philae »³ se posera sur la comète pour permettre de réaliser certaines expériences directement à sa surface.

Expérience CONSERT

Malgré les diverses observations déjà effectuées, la communauté scientifique connaît encore assez mal la structure d'un noyau cométaire. En effet, l'organisation de la matière au sein du noyau dépend de la façon dont celui-ci s'est formé. Or, ce processus est fortement lié aux conditions de la nébuleuse primitive qui donna naissance au système solaire [4]. Selon les scientifiques, l'étude de la structure d'un noyau cométaire permettra de « comprendre la mécanique d'accrétion de la matière » [5].

L'instrument CONSERT, embarqué à bord de la sonde ROSETTA, a été conçu pour réaliser une tomographie du noyau de la comète 67P/Churyumov-Gerasimenko. Ce sondage par ondes radio « permettra d'avoir accès à l'image de la structure du noyau cométaire ainsi qu'à des informations sur les propriétés physiques et électriques du noyau » [5].

Si l'équipe du LPG a défini le principe de l'expérience et son concept, la réalisation électronique et la fabrication du modèle de vol ont cependant été réalisés au Service d'Aéronomie du CNRS. Le LPG est malgré tout fortement impliqué dans l'étalonnage de l'instrument, dans l'intégration sur la sonde ROSETTA et, depuis le lancement de la mission, dans le suivi de l'instrument. CONSERT est un instrument original, unique de par sa configuration de mesure adaptée à un noyau cométaire. Afin de préparer l'analyse et le traitement des données CONSERT en 2014, des recherches théoriques sont menées sur le problème de la propagation des signaux électromagnétiques dans le noyau de la comète afin de tirer le meilleur parti du signal qui sera mesuré. En parallèle, des répliques de l'instrument CONSERT sont utilisées pour étudier des sites terrestres (glacier, dôme de centre volcanique) [1].

Comme le montre la figure 1, l'instrument CONSERT est en réalité composé de deux émetteurs-récepteurs radio[6][7] :

- le premier est logé dans le corps principal de la sonde orbitant autour de la comète ;
- le second est intégré à l'atterrisseur posé à la surface de la comète.

En position idéale pour l'expérience, le noyau se retrouve entre l'orbiteur et l'atterrisseur. Il sera alors possible depuis l'atterrisseur d'émettre une onde radio à 90 MHz après propagation à travers la matière cométaire et qui sera enregistrée par l'orbiteur. À sa sortie du noyau, « suivant que la comète est homogène, stratifiée ou qu'elle se compose de cométésimaux » [8], le signal radio pourra avoir subi diverses altérations. Celles-ci pourront par exemple être caractérisées par des échos, des réflexions, des retards causés par les variations de densité. Le principe de l'expérience repose donc sur le fait que la propagation des ondes radio sera perturbée par le noyau de la comète. Les propriétés électriques moyennes et la structure interne du noyau seront déduites à partir du temps de propagation et de l'énergie du signal reçu.[7] [1]

La figure 2 schématise une propagation d'ondes au travers d'une comète. Le point blanc matérialise la position de la sonde sur son orbite tandis que l'astérisque blanc symbolise la position de l'atterrisseur à la surface de la comète. Dans le vide, les ondes ont une vitesse constante : celle de la lumière, c'est-à-dire à 300 000 km/s. Cette vitesse notée C_0 est représentée par la couleur bleu marine (située à l'extrême droite de l'échelle). Au sein du noyau, toute différence de matériaux,

3. L'atterrisseur est nommé ainsi en hommage à l'obélisque de Philae qui permet, avec la pierre de Rosette, de déchiffrer les hiéroglyphes.

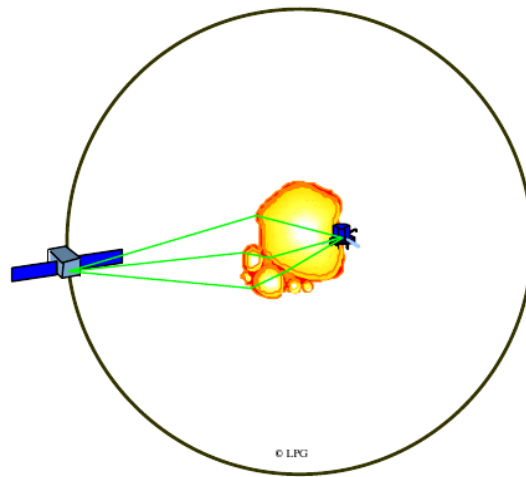


FIGURE 1 – Situation des deux modules de l’instrument CONSERT. Source : [7]

de porosité ou de densité induira une modification de la vitesse de propagation des ondes. Les différentes vitesses des signaux sont exprimées en fraction de C_0 et sont représentées par les autres couleurs.

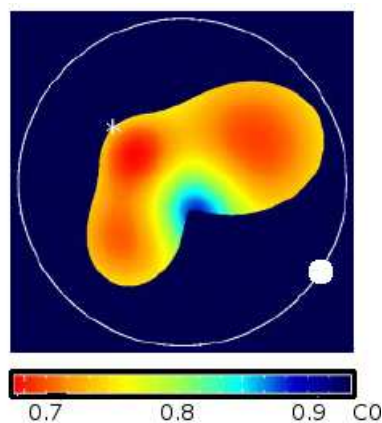


FIGURE 2 – Vitesse de propagation des ondes en présence d’une comète. Source : [7]

Disposant pour le moment d’assez peu d’informations sur la comète 67P/Churyumov-Gerasimenko, il est nécessaire de simuler en laboratoire le déroulement des différentes expériences de la mission ROSETTA. En effet, il est important de déterminer le meilleur site d’atterrissage pour le module Philae. Ce site devra offrir des conditions optimales aux différents instruments pour recueillir des données pertinentes. Il sera choisi en fonction des observations effectuées par la sonde mais également à partir des résultats de simulation. En ce qui concerne CONSERT, le site retenu doit permettre aux ondes de couvrir un volume optimal de la comète en la traversant. Il faut également que le signal reçu soit d’une puissance suffisante.

Simulation de l’expérience CONSERT

La simulation de l’expérience CONSERT utilise un algorithme de lancer de rayons [6]. Cette technique est très utilisée en imagerie de synthèse. Elle simule le parcours de la lumière entre une source et l’œil en reproduisant les phénomènes physiques de réflexion et réfraction [9]. Pour optimiser le temps de calcul, un parcours inverse est généralement préféré. Dans le cas de CONSERT,

le noyau n'ayant pas une composition homogène, un rayon sera simulé dans son sens normal de propagation pour faciliter les calculs.

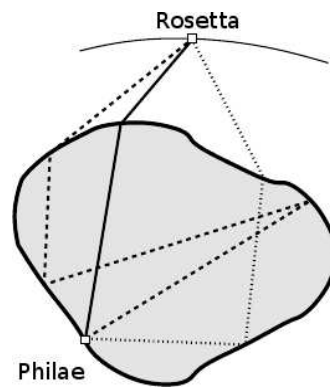


FIGURE 3 – Différents parcours de rayons au sein d'une comète. Source : [6]

La figure 3 représente différents parcours de rayons au sein d'un noyau. Le trait plein indique un rayon propagé sans réflexion. Les traits en pointillé indiquent des rayons propagés avec une ou plusieurs réflexions.

Problématiques et plan du mémoire

Dans le cadre de l'épreuve TEST [10], une étude préliminaire des données de l'expérience CONSERT a permis de proposer plusieurs techniques pour les restructurer, les pérenniser et y accéder plus facilement. En guise de transition avec le stage, il est tout d'abord demandé de mettre en œuvre les résultats de cette étude afin de déterminer quelle méthode est réellement la plus appropriée. Le premier chapitre de ce mémoire permettra de faire un rappel de cette étude puis détaillera la mise en œuvre de prototypes afin d'évaluer les solutions techniques proposées.

Le second chapitre permettra d'aborder le véritable sujet du stage. Il sera en effet consacré à l'étude de la plateforme de simulation de l'expérience CONSERT. Celle-ci utilise actuellement la grille CIMENT via son intergiciel CiGri et la chaîne de simulation ainsi paramétrée en reste très dépendante. Cette-dernière comporte plusieurs opérations manuelles ou semi-automatisées qui ne permettent pas l'utilisation intensive voulue. Il est donc nécessaire de mettre en œuvre une solution plus paramétrable et plus automatisée qui puisse être utilisée rapidement par l'équipe.

Avec le troisième chapitre, on effectuera un rapide état de l'art concernant les grilles de calcul et leurs intergiciels. Deux d'entre-eux seront plus particulièrement étudiés.

Le chapitre suivant décrira les étapes et choix concernant la réalisation de la nouvelle plateforme. Le dernier chapitre présentera l'outil réalisé et son utilisation appliquée au projet CONSERT.

Enfin, nous terminerons ce mémoire par une synthèse du travail effectué et des évolutions envisagées.

Données de simulation

Afin de préparer au mieux le déroulement de l'expérience CONSERT dans les conditions réelles de la mission ROSETTA, l'équipe du LPG a développé une suite logicielle pour assurer sa simulation. Comme le montre la figure 4, différentes données sont générées et manipulées par ces outils qui composent la « plateforme de simulation ».

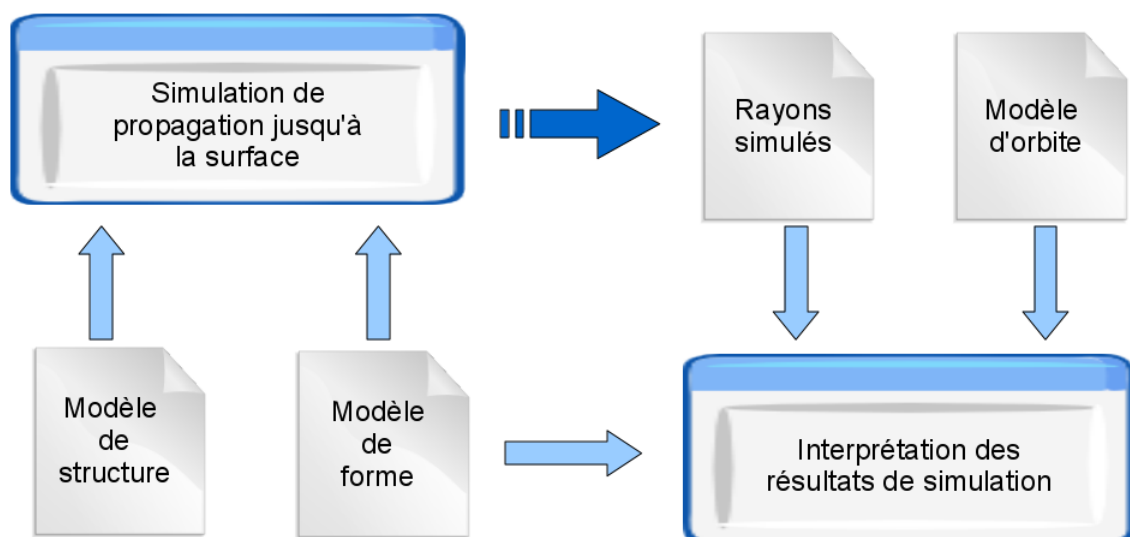


FIGURE 4 – Étapes et éléments intervenant dans la simulation CONSERT.

Afin d'appréhender le fonctionnement de la plateforme, il est tout d'abord nécessaire de présenter les données qui interviennent dans la simulation de l'expérience CONSERT.

1.1 Présentation des données

1.1.1 Modèle d'orbite

Ce modèle précise comment la sonde évolue autour de la comète. Il est généré par le logiciel GINS, un outil développé par le CNES depuis de nombreuses années et qui permet de calculer des orbites autour d'un corps du système solaire [11]. Il est essentiellement utilisé par l'outil d'interprétation pour offrir une représentation de l'orbite et ainsi permettre des traitements sur les résultats d'une simulation.

Le fichier produit contient les valeurs numériques caractérisant l'orbite de la sonde dans le référentiel de la comète [12]. Ainsi, chaque ligne du fichier définit un éphéméride, c'est-à-dire un point de l'orbite, par plusieurs paramètres dont :

- le temps ;
- la position de l'orbiteur ;
- la vitesse de l'orbiteur ;
- l'accélération de l'orbiteur.

La figure 5 est un exemple de représentation graphique d'une orbite suivie par la sonde autour de la comète.

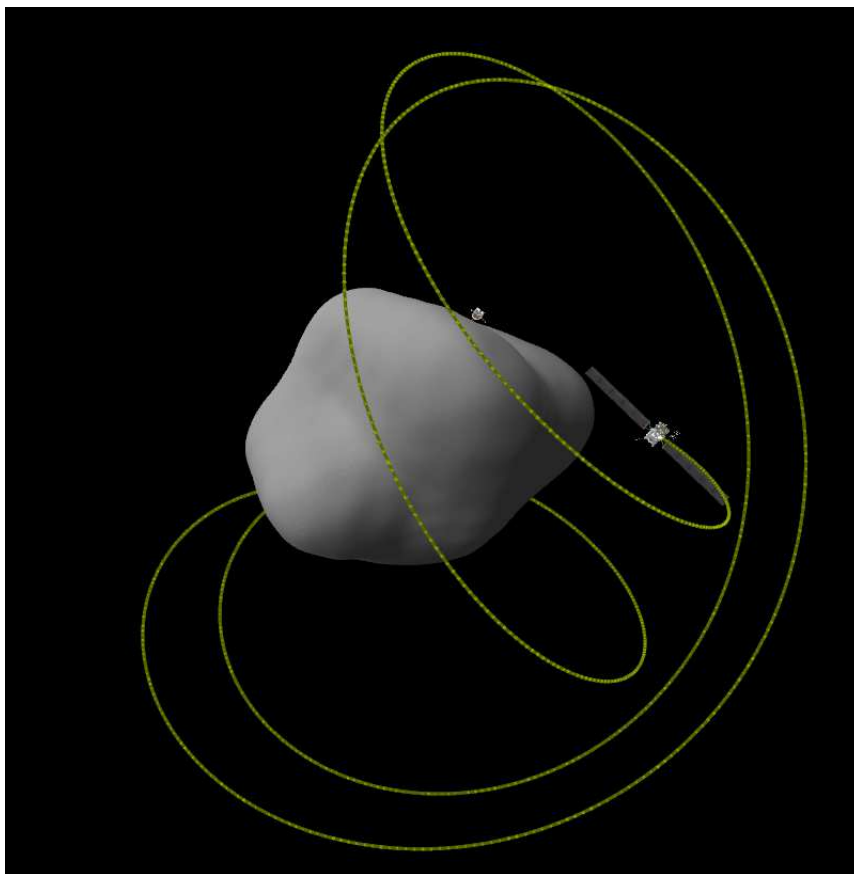


FIGURE 5 – Exemple de représentation graphique d'une orbite.

1.1.2 Modèle de forme de noyau

Ce modèle définit la forme du noyau de la comète. Il sera affiné à partir des observations de la sonde, notamment lors de la phase de cartographie de la comète. Actuellement, il est réalisé

à partir d'harmoniques sphériques [13]. Cette représentation mathématique permet d'obtenir une surface lisse utilisable par l'algorithme du lancer de rayons.

Chaque ligne présente un coefficient d'une harmonique. En partant d'une sphère, les « déformations » appliquées successivement permettront d'obtenir la forme de noyau souhaitée comme celle donnée en exemple par la figure 6.

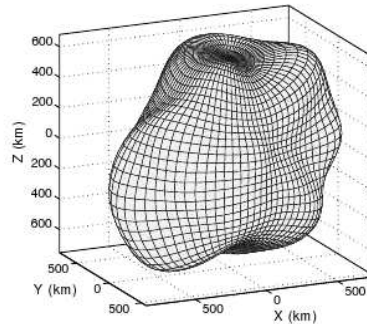


FIGURE 6 – Modélisation d'un noyau de comète en harmoniques sphériques. Source : [6]

1.1.3 Modèle de structure de noyau

Ce fichier définit un modèle structurel du noyau de comète. Il est établi à partir des différentes études scientifiques. Il est composé de deux groupes de paramètres. Le premier spécifie diverses informations générales à raison d'une par ligne :

- le ratio massique poussière / glace ;
- la densité de la glace ;
- la densité de la poussière ;
- la permittivité de la glace ;
- la permittivité de la poussière ;
- la permittivité en surface ;
- la porosité finale.

Pour la représentation actuellement retenue, la structure du noyau est ponctuée par des zones globulaires appelées « cométésimaux » [6]. Celles-ci caractérisent des parties dont la porosité est différente. Selon l'évolution des recherches scientifiques, cette modélisation pourra être remplacée par une décomposition structurelle en couches.

Dans la deuxième partie du fichier, chaque ligne définit un cométésimal en spécifiant ses caractéristiques telles que :

- sa position en coordonnées cartésiennes ;
- son rayon ;
- la porosité de son centre.

La figure 7 schématise un exemple de structure. Le noyau est représenté par la forme jaune (claire) tandis que les ronds oranges (foncés) symbolisent des cométésimaux.

1.1.4 Résultats de simulation

Les modèles de forme et de structure sont utilisés pour calculer la propagation des rayons au travers du noyau et jusqu'à sa surface [14]. Le fichier généré contient les valeurs numériques de divers calculs relatifs aux rayons simulés.

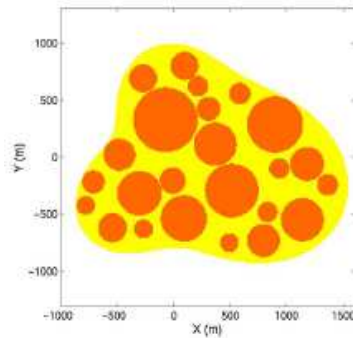


FIGURE 7 – Structure cométésimale d'un noyau. Source : [6]

Globalement, ce fichier texte contient plusieurs enregistrements. Chacun est structuré en plusieurs groupes de paramètres. Tout d'abord, sont définis :

- un numéro d'identification pour le groupe de rayons ;
- la direction en coordonnées sphériques du rayon émis ;
- l'indication de propagation ou non des quatre rayons complémentaires.

Les cinq groupes suivants caractérisent le rayon principal et ses quatre rayons complémentaires par :

- la direction en coordonnées cartésiennes du rayon à sa sortie en surface du noyau ;
- la position en coordonnées cartésiennes du point de sortie du rayon à la surface de la comète ;
- le temps de propagation du rayon ;
- la puissance du signal après propagation.

Enfin, un dernier groupe contient un nombre variable de blocs chacun rassemblant quatre valeurs de polarisation.

1.2 Problématiques et objectifs

Les données de simulation – modèles et résultats – sont structurées au sein de fichiers « plats », au format texte brut, définis pour les besoins du projet. Elles sont, pour la plupart, des données numériques complexes et en grande quantité. Certaines de ces données peuvent être superflues et ne servent pas à tous les traitements. À titre d'exemple, dans le cadre d'une simulation, un échantillon de fichier décrivant quelques 190 000 rayons atteignant la surface pèse pas moins de 430 Mo. Pour une simulation complète, le poids moyen d'un fichier de résultats est de l'ordre de plusieurs Go.

Tel l'extrait de résultats de simulation présenté par le listing 1.1, chaque ligne contient généralement une seule valeur. Parfois une ligne peut contenir plusieurs « colonnes » de valeurs, celles-ci étant séparées par un nombre variable d'espaces. Aucune donnée n'est identifiée clairement ; il est obligatoire de se référer à une notice pour en connaître la signification. Dans le cas des résultats de simulation, un « enregistrement » décrivant un groupe de rayons n'a pas un nombre de lignes fixe. Il est nécessaire de lire une ligne particulière d'un enregistrement pour déterminer le nombre de données restantes à lire...

La plupart des valeurs sont de type flottant. En lieu et place de ces flottants, il arrive cependant de rencontrer des valeurs textuelles « NaN » pour indiquer un dépassement vers l'infini. C'est une exception à prendre en compte lors du chargement des données.

```

1 800000
   8.000000000000000
   90.000000000000000
  -0.0719873309135
   0.6253086924553
6  0.7770501375198
  -297.2220764160156
   971.1893920898438
 1584.2604980468750
   0.000000000000000
11 -0.0710293650627
   0.6256966590881
   0.7768260240555
  -296.3384094238281
   972.0954589843750
16 1584.3240966796875
  -4.0038681030273
   0.000000000000000
 4771.5869140625000
 -806.7005004882813
21  1.0959184169769
  -4.0038681030273
  -2.0125210285187
  -0.0729171633720
   0.6249430179596
26  0.7772576212883
  -298.0742187500000
   970.3147583007813
 1584.1884765625000
   0.000000000000000
31  0.000000000000000
 4771.5869140625000
 -806.4244384765625
   1.0957654714584
   0.000000000000000
36  0.000000000000000
  -0.0629405677319
   0.6323003768921
   0.7721624374390
41 -288.3265075683594
   983.8140869140625
 1584.0480957031250
   0.000000000000000
   0.000000000000000
46 4771.5869140625000
  -808.0252075195313
   1.0972506999969
   0.000000000000000
   0.000000000000000

51 -0.0807229280472
   0.6184421181679
   0.7816733717918
  -305.7173461914063
   959.1517333984375
 1584.5151367187500
56  0.000000000000000
   0.000000000000000
 4771.5869140625000
 -805.2031860351563
   1.0945388078690
61                                     NaN
   0.000000000000000
 4771.5869140625000
 -806.5628051757813
   1.0958422422409
66  -17.0885791778564
   2.000000000000000
*****
   0.0000009317718
   0.7920064926147
71  1.3784048557281
   1.000000000000000
   1.000000000000000
   1.000000000000000
   1.000000000000000
76  0.1544300466776
  -522.6773681640625
 2929.6359863281250
 4017.9675292968750
 -159.4037628173828
81  7907.4653320312500
  -806.7785034179688
   1.0960068702698
 5000.0024414062500
 36.5253944396973
86  100.1157302856445
   0.0026453521568
   0.0001125790877
   0.0001544754487
   0.000000000000000
91  0.0000866806949
   0.0009688822320
  -0.0007716498221
   90.0000000000000
 99999.0000000000000

```

Listing 1.1 – Extrait de résultats de simulation

Il apparaît à l'usage que les structures des données employées ne sont pas les plus simples à manipuler et à entretenir. Pour les manipuler, les algorithmes de lecture doivent être définis précisément. Ainsi, l'évolution de ces modèles est sensible puisqu'en modifiant la structure des fichiers correspondants, il est nécessaire de répercuter le changement sur les différents logiciels.

Il est donc nécessaire de déterminer la méthode la plus adaptée pour changer les structures de données. En vue d'une intégration aux outils existants, il faut également proposer une implémentation type des algorithmes de lecture et d'écriture des fichiers de simulation. Le type de structure retenue devra alors s'appuyer sur un standard afin d'être plus évolutif. Elle devra permettre d'identifier clairement les données pour les rendre plus facilement accessibles. Le volume de données manipulées étant relativement conséquent, la performance des solutions mises en œuvre sera évaluée en fonction de la vitesse de lecture des données et la charge mémoire occasionnée.

1.3 Solutions proposées

Face aux problèmes soulevés par l'usage des structures de données actuelles, l'équipe du Laboratoire de Planétologie de Grenoble a décidé d'établir de nouvelles structures pour accueillir les informations de simulation. L'utilisation du langage XML est fortement envisagée mais d'autres techniques peuvent également répondre aux besoins. Cette partie rend compte de la démarche d'analyse de ces solutions.

1.3.1 Langage XML

XML est un langage de balisage extensible créé en 1998. Tout comme son cousin HTML, il est dérivé du langage informatique SGML et est recommandé par le W3C [15]. Son principal but est de stocker ou de transporter des données de manière structurée. En réalité, XML n'est pas vraiment un langage à part entière mais plutôt une sorte de « méta-langage » servant à définir différents dialectes selon les besoins. Il est ainsi à la base de nombreux formats standards actuels comme XHTML, Atom, OpenDocument, SVG, etc.

1.3.1.1 Structure d'un fichier XML

Un fichier XML débute par une ligne spéciale appelée « prologue ». Elle précise la version du langage XML utilisée ainsi que l'encodage des caractères. Pour une meilleure interopérabilité, il est recommandé de choisir l'UTF-8. Le prologue est généralement suivi par des instructions de traitement. Ces balises permettent par exemple de définir un schéma de validation interne ou externe, ou d'appeler une feuille de style XSLT pour appliquer une transformation.

Les lignes suivantes contiennent une arborescence de « nœuds » composée d'éléments et d'attributs. Le premier nœud est l'élément « racine » qui doit être unique. Un élément est formé par un couple de balises ouvrante `<element>` et fermante `</element>` ou par une balise auto-fermante `<element />`. Toutes les balises ouvertes doivent être fermées dans le bon ordre. La balise ouvrante (ou auto-fermante) définit l'élément par un nom d'au moins un caractère alphanumérique mais qui ne commence pas par un chiffre. Elle peut également contenir des couples de la forme `attribut="valeur"`. De plus, un élément peut contenir un ensemble d'éléments enfants ou une simple valeur [16][17][18].

Le listing 1.2 présente un extrait de la structure XML adaptée aux résultats de simulation du listing 1.1. Pour simplifier l'algorithme de lecture, les propriétés de chaque donnée ont été intégrées aux éléments sous la forme d'attributs.

<pre> 1 <?xml version="1.0" encoding="UTF-8"?> 2 <simulation> 3 <groupeRayons id="800000"> 4 <propagation> 5 <direction type="initiale" 6 coordonnee="spherique" 7 p="8.0" l="90.0"> 8 </direction> 9 <direction type="finale" 10 coordonnee="spherique" 11 r="5000.00244140625" 12 p="36.5253944396973" 13 l="100.1157302856445"> 14 </direction> 15 <point type="finale" 16 coordonnee="cartesienne" 17 x="-522.6773681640625" </pre>	<pre> 18 y="2929.635986328125" 19 z="4017.967529296875"> 20 </point> 21 <permittivite type="finale" 22 t="7907.46533203125" 23 p="-806.7785034179688" 24 c="1.0960068702698"> 25 </permittivite> 26 <attenuation type="finale" 27 valeur="159.4037628173828"> 28 </attenuation> 29 <attenuation type="surface" 30 valeur="17.0885791778564"> 31 </attenuation> 32 </propagation> 33 <rayon num="4" abouti="false"> 34 <direction type="finale" 35 coordonnee="cartesienne" </pre>
--	---

```

40     x="-0.0807229280472"
      y="0.6184421181679"
      z="0.7816733717918">
</direction>
<point type="surface"
      coordonnee="cartesienne"
      x="-305.7173461914063"
      y="959.1517333984375"
      z="1584.51513671875">
45 </point>
<permittivite type="surface"
      t="1.094538807869"
      p="-805.2031860351562"
      c="4771.5869140625">
50 </permittivite>
<transmission type="surface"
      w="0.0" tr="0.0"
      gain="NaN" pst="0.0">
</transmission>
55 </rayon>
<rayon num="3" abouti="false">
<direction type="finale"
      coordonnee="cartesienne"
      x="-0.0629405677319"
60     y="0.6323003768921"
      z="0.772162437439">
</direction>
<point type="surface"
      coordonnee="cartesienne"
65     x="-288.3265075683594"
      y="983.8140869140625"
      z="1584.048095703125">
</point>
<permittivite type="surface"
70     t="1.0972506999969"
      p="-808.0252075195312"
      c="4771.5869140625">
</permittivite>
<transmission type="surface"
75     w="0.0" tr="0.0"
      gain="0.0" pst="0.0">
</transmission>
</rayon>
<rayon num="2" abouti="false">
80 <direction type="finale"
      coordonnee="cartesienne"
      x="-0.072917163372"
      y="0.6249430179596"
      z="0.7772576212883">
85 </direction>
<point type="surface"
      coordonnee="cartesienne"
      x="-298.07421875"
      y="970.3147583007812"
90     z="1584.1884765625">
</point>
<permittivite type="surface"
      t="1.0957654714584"
      p="-806.4244384765625"
      c="4771.5869140625">
95 </permittivite>
<transmission type="surface"
      w="0.0" tr="0.0"
      gain="0.0" pst="0.0">
100 </transmission>
</rayon>
<rayon num="1" abouti="false">
105 <direction type="finale"
      coordonnee="cartesienne"
      x="-0.0710293650627"
      y="0.6256966590881"
      z="0.7768260240555">
</direction>
<point type="surface"
110     coordonnee="cartesienne"
      x="-296.3384094238281"
      y="972.095458984375"
      z="1584.3240966796875">
</point>
115 <permittivite type="surface"
      t="1.0959184169769"
      p="-806.7005004882812"
      c="4771.5869140625">
</permittivite>
120 <transmission type="surface"
      w="-4.0038681030273" tr="0.0"
      gain="-4.0038681030273"
      pst="-2.0125210285187">
</transmission>
125 </rayon>
<rayon num="0">
<direction type="finale"
      coordonnee="cartesienne"
130     x="-0.0719873309135"
      y="0.6253086924553"
      z="0.7770501375198">
</direction>
<point type="surface"
      coordonnee="cartesienne"
135     x="-297.2220764160156"
      y="971.1893920898438"
      z="1584.260498046875">
</point>
<permittivite type="surface"
140     t="1.0958422422409"
      p="-806.5628051757812"
      c="4771.5869140625">
</permittivite>
<transmission type="surface"
145     w="0.0" tr="0.7920064926147"
      gain="0.1544300466776"
      pst="9.317718E-7">
</transmission>
</rayon>
150 <polarisation num="2"
      k1="8.66806949E-5"
      k2="9.68882232E-4"
      k3="-7.716498221E-4"
      k4="90.0"></polarisation>
155 <polarisation num="1"
      k1="0.0026453521568"
      k2="1.125790877E-4"
      k3="1.544754487E-4"
      k4="0.0"></polarisation>
160 </groupeRayons>
[...]</simulation>

```

Listing 1.2 – Structure XML pour l'extrait de résultats de simulation du listing 1.1

1.3.1.2 Lecture des données et chargement mémoire

Une structure XML ne peut pas être utilisée immédiatement par un logiciel. Au préalable, il est nécessaire d'en réaliser une analyse syntaxique avec un « parseur ». Ce composant permet de lire le fichier XML, de vérifier sa cohérence syntaxique et d'en proposer ensuite une représenta-

tion logique. Par ailleurs, en utilisant une définition ou un schéma de validation, un parseur est généralement capable de valider une arborescence XML. Un parseur est en fait un élément de bibliothèque logicielle. Deux API sont généralement proposées par les parseurs : DOM et SAX.

Le langage XPath est une recommandation du W3C [19] qui permet de localiser un nœud, c'est-à-dire un élément ou un attribut, dans un document XML. Sa syntaxe n'est pas du tout basée sur XML. XPath est avant tout un langage de requête dans un arbre XML. Même si ce n'est pas son objectif principal, il peut néanmoins réaliser quelques calculs ou tests simples. Il est à considérer comme un langage intermédiaire permettant à d'autres langages d'accéder à des éléments d'une arborescence.

1.3.1.2.1 DOM

DOM, qui est une recommandation du W3C [20], est l'API la plus couramment proposée par les parseurs XML. Elle est notamment utilisée par les navigateurs Web. DOM offre une représentation objet d'une arborescence en chargeant l'intégralité du document XML dans une structure de données. De ce fait, la manipulation des données est facilitée au détriment d'une empreinte mémoire conséquente [16]. Même si DOM est très souple d'utilisation, il n'est pas vraiment adapté aux gros volumes de données comme dans le cas de l'expérience CONSERT. Il a cependant l'avantage d'offrir le support des requêtes Xpath.

La structure générée par DOM permet d'obtenir un objet pour tout nœud analysé. Cet objet possède des méthodes fournissant notamment le nom du nœud, sa valeur, la valeur d'un attribut donné, l'objet nœud parent ou encore la collection des objets nœuds enfants.

Si DOM facilite le parcours d'une arborescence XML, il permet également de la modifier tout aussi simplement. Chaque objet nœud possède des méthodes pour modifier son nom ou sa valeur ; pour ajouter, modifier ou supprimer un attribut ; ajouter ou supprimer un objet nœud enfant...

1.3.1.2.2 SAX

SAX [21] est la deuxième API proposée par certains parseurs. Son principe diffère de DOM. Au lieu de représenter la totalité d'un document XML en mémoire, SAX traite les nœuds successivement au fur et à mesure de l'analyse du fichier [16]. Le parseur travaille donc seulement sur une partie de l'arborescence. Cela a l'avantage d'être beaucoup plus rapide et plus économique en mémoire que DOM. De ce fait, cette API est recommandée pour les gros volumes de données comme dans le cas de CONSERT. Elle a cependant l'inconvénient de ne pas permettre l'utilisation de Xpath.

Généralement, un algorithme SAX fonctionne sur une approche itérative ou sur une approche événementielle, cette dernière étant un peu plus compacte que la première. A chaque pas de lecture, un évènement est déclenché (ou une méthode spécifique appelée dans le cas d'une approche itérative). Cet évènement permet d'identifier le type de l'information qui a été lue depuis le fichier : une balise ouvrante, une balise fermante, une valeur, une déclaration DTD, etc. Si l'on souhaite intervenir sur cette information, il suffit d'intercepter cet évènement. Dans ce cas, pour le nœud courant, il est possible d'obtenir son nom, sa valeur ou la valeur d'un attribut donné. Pour poursuivre l'analyse des nœud enfants, il est nécessaire d'attendre le pas de lecture suivant et de gérer le contexte précédent – le nœud parent – par l'intermédiaire d'une pile.

Contrairement à DOM, il n'est pas possible de revenir en arrière dans l'analyse du fichier. De même la modification de la structure n'est pas proposée directement.

1.3.1.3 Validation des données

Lors de l'écriture d'un fichier XML, la première règle est de s'assurer qu'il soit bien formé. Il faut notamment vérifier que toutes les balises ouvertes sont bien fermées et dans le bon ordre. Il est cependant souhaitable de pouvoir contrôler le respect de règles supplémentaires telles que la présence de tout nœud, attribut ou valeur obligatoire [17].

La création d'une nouvelle structure XML répond généralement à un besoin particulier. Cela revient en quelque sorte à inventer un nouveau dialecte. De fait, il n'y a aucune règle existante permettant de s'assurer que le contenu d'un fichier précis est bien valide. Il faut alors établir un document de référence pour décrire de façon générique la structure et ses contraintes. Cela permettra de valider tous les fichiers créés sur ce modèle.

1.3.1.3.1 DTD

Le langage DTD est relativement ancien. Issu de l'univers SGML, il est normalisé par l'ISO et reste encore très utilisé notamment pour la validation des documents HTML. Une DTD n'est pas un fichier XML. Elle est composée d'instructions et non de balises. Cette grammaire est simple et rapide à écrire mais présente l'inconvénient d'être assez pauvre en possibilités de contrôle. En effet, il n'est par exemple pas possible de spécifier le type des données [16][17][22].

Le listing 1.3 présente la DTD permettant de valider la structure XML du listing 1.2. Comme on peut le constater, cette grammaire n'offre la possibilité de contrôler que partiellement les données.

<pre> <!ELEMENT simulation (groupeRayons+)> <!ELEMENT groupeRayons 4 (propagation, rayon+, polarisation+)> <!ATTLIST groupeRayons id CDATA #REQUIRED> <!ELEMENT propagation 9 (direction+, point, permittivite, attenuation+)> <!ELEMENT direction EMPTY> <!ATTLIST direction 14 type (initiale finale) #REQUIRED coordonnee (cartesienne spherique) #REQUIRED x CDATA #IMPLIED y CDATA #IMPLIED 19 z CDATA #IMPLIED r CDATA #IMPLIED p CDATA #IMPLIED l CDATA #IMPLIED> 24 <!ELEMENT point EMPTY> <!ATTLIST point type (surface finale) #REQUIRED coordonnee (cartesienne) #REQUIRED x CDATA #REQUIRED 29 y CDATA #REQUIRED z CDATA #REQUIRED> <!ELEMENT permittivite EMPTY> 34 <!ATTLIST permittivite </pre>	<pre> type (surface finale) #REQUIRED t CDATA #REQUIRED p CDATA #REQUIRED c CDATA #REQUIRED> 39 <!ELEMENT attenuation EMPTY> <!ATTLIST attenuation 44 type (surface finale) #REQUIRED valeur CDATA #REQUIRED> <!ELEMENT rayon (direction, point, permittivite, transmission)> <!ATTLIST rayon 49 num (0 1 2 3 4) #REQUIRED abouti (true false) #IMPLIED> <!ELEMENT polarisation EMPTY> <!ATTLIST polarisation 54 num CDATA #REQUIRED k1 CDATA #REQUIRED k2 CDATA #REQUIRED k3 CDATA #REQUIRED k4 CDATA #REQUIRED> 59 <!ELEMENT transmission EMPTY> <!ATTLIST transmission 64 type (surface comete) #REQUIRED w CDATA #REQUIRED tr CDATA #REQUIRED gain CDATA #REQUIRED pst CDATA #REQUIRED> </pre>
---	---

Listing 1.3 – DTD validant la structure XML du listing 1.2

1.3.1.3.2 XSD

Le langage XSD est beaucoup plus récent que celui des DTD. En effet, il a été proposé en 2001 par le W3C pour répondre au problème de validation des fichiers XML [23]. Un schéma XML est d'ailleurs lui-même un fichier XML à part entière qui suit une grammaire spécifique. Il débutera donc par un prologue, sera composé de balises et aura un élément racine. En revanche, contrairement au langage DTD, l'ordre d'écriture des balises est important : un élément devra préalablement être défini avant qu'il soit possible de l'utiliser [16][17][24].

Beaucoup plus verbeux que le langage DTD, cette grammaire a l'avantage de permettre un contrôle précis de données, tant au niveau du type qu'au niveau des cardinalités. Le listing 1.4 présente le schéma XML permettant de valider la structure XML du listing 1.2. On constate aisément que la définition de la structure est beaucoup plus fine et plus stricte qu'avec le langage DTD.

```

4 <?xml version="1.0" encoding="UTF-8"?>
  <xs:schema
    elementFormDefault="qualified"
    xmlns:xs=
      "http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="TC-Direction">
      <xs:attribute name="type"
        type="TS-TypeDirection" />
      <xs:attribute name="coordonnee"
        type="TS-Coordonnee" />
      <xs:attribute name="x"
        type="xs:double" />
      <xs:attribute name="y"
        type="xs:double" />
      <xs:attribute name="z"
        type="xs:double" />
      <xs:attribute name="r"
        type="xs:double" />
      <xs:attribute name="p"
        type="xs:double" />
      <xs:attribute name="l"
        type="xs:double" />
    </xs:complexType>
    <xs:complexType name="TC-Direction0">
      <xs:attribute name="type"
        type="TS-TypeDirection" />
      <xs:attribute name="coordonnee"
        type="TS-Coordonnee" />
    </xs:complexType>
    <xs:complexType name="TC-Point">
      <xs:attribute name="type"
        type="TS-TypePoint" />
      <xs:attribute name="coordonnee"
        type="TS-Coordonnee" />
    </xs:complexType>
    <xs:complexType name="TC-Permittivite">
      <xs:attribute name="type"
        type="TS-TypePermittivite" />
      <xs:attribute name="t"
        type="xs:double" />
      <xs:attribute name="p"
        type="xs:double" />
      <xs:attribute name="c"
        type="xs:double" />
    </xs:complexType>
    <xs:complexType name="TC-Propagation">
      <xs:sequence>
      <xs:element name="direction"
        type="TC-Direction"
        minOccurs="2" maxOccurs="2" />
      <xs:element name="point"
        type="TC-PointCartesienne" />
      <xs:element name="permittivite"
        type="TC-Permittivite" />
      <xs:element name="attenuation"
        type="TC-Attenuation"
        minOccurs="2" maxOccurs="2" />
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TC-Rayon">
      <xs:sequence>
      <xs:element name="direction"
        type="TC-DirectionCartesienne" />
      <xs:element name="point"
        type="TC-PointCartesienne" />
      <xs:element name="permittivite"
        type="TC-Permittivite" />
      <xs:element name="transmission"
        type="TC-Transmission" />
      </xs:sequence>
      <xs:attribute name="num"
        type="xs:int" use="required" />
      <xs:attribute name="abouti"
        type="xs:boolean" use="optional" />
    </xs:complexType>
    <xs:complexType name="TC-Groupe">
      <xs:sequence>
      <xs:element name="propagation"
        type="TC-Propagation" />
      <xs:element name="rayon"
        type="TC-Rayon"
        minOccurs="5" maxOccurs="5" />
      <xs:element name="polarisation"
        type="TC-Polarisation"
        minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="id"
        type="xs:int"
        use="required" />
    </xs:complexType>
    <xs:complexType
      name="TC-Simulation">
      <xs:sequence>
      <xs:element name="groupeRayons"
        type="TC-Groupe"
        minOccurs="0"
        maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
    <xs:complexType
      name="TC-Transmission">
      <xs:attribute name="type"
        type="xs:string" />
      <xs:attribute name="w"
        type="xs:double" />
      <xs:attribute name="tr"
        type="xs:double" />

```

```

119 <xs:attribute name="gain"
    type="xs:double" />
    <xs:attribute name="pst"
    type="xs:double" />
  </xs:complexType>
124 <xs:complexType
    name="TC-Polarisation">
  <xs:attribute name="num"
    type="xs:int"
    use="required" />
129 <xs:attribute name="k1"
    type="xs:double"
    use="required" />
    <xs:attribute name="k2"
    type="xs:double"
134   use="required" />
    <xs:attribute name="k3"
    type="xs:double"
    use="required" />
    <xs:attribute name="k4"
139   type="xs:double"
    use="required" />
  </xs:complexType>

  <xs:complexType name="TC-Attenuation">
144 <xs:attribute name="type"
    type="TS-TypeAttenuation" />
    <xs:attribute name="valeur"
    type="xs:double" />
  </xs:complexType>
149
  <xs:element name="simulation"
    type="TC-Simulation" />

  <xs:attributeGroup
154   name="AG-CoordonneeCartesienne">
    <xs:attribute name="x"
    type="xs:double" />
    <xs:attribute name="y"
    type="xs:double" />
159 <xs:attribute name="z"
    type="xs:double" />
  </xs:attributeGroup>

  <xs:attributeGroup
164   name="AG-CoordonneeSpherique">
    <xs:attribute name="r"
    type="xs:double" />
    <xs:attribute name="p"
    type="xs:double" />
169 <xs:attribute name="l"
    type="xs:double" />
  </xs:attributeGroup>

  <xs:simpleType name="TS-TypeDirection">
174 <xs:restriction base="xs:string">
    <xs:enumeration value="initiale" />
    <xs:enumeration value="finale" />
  </xs:restriction>
  </xs:simpleType>
179
  <xs:simpleType name="TS-TypePoint">
    <xs:restriction base="xs:string">
    <xs:enumeration value="surface" />
    <xs:enumeration value="finale" />
184 </xs:restriction>
  </xs:simpleType>
  </xs:complexType>
  </xs:simpleType>
189 <xs:simpleType
    name="TS-TypeTransmission">
  <xs:restriction base="xs:string">
    <xs:enumeration value="surface" />
  </xs:restriction>
  </xs:simpleType>

194 <xs:simpleType
    name="TS-TypePermittivite">
  <xs:restriction base="xs:string">
    <xs:enumeration value="surface" />
    <xs:enumeration value="finale" />
199 </xs:restriction>
  </xs:simpleType>

  <xs:simpleType
    name="TS-TypeAttenuation">
204 <xs:restriction base="xs:string">
    <xs:enumeration value="surface" />
    <xs:enumeration value="finale" />
  </xs:restriction>
  </xs:simpleType>
209
  <xs:simpleType name="TS-Coordonnee">
    <xs:restriction base="xs:string">
    <xs:enumeration value="cartesienne" />
    <xs:enumeration value="spherique" />
214 </xs:restriction>
  </xs:simpleType>

  <xs:complexType
    name="TC-DirectionSpherique">
219 <xs:complexContent>
  <xs:extension base="TC-Direction0">
    <xs:attributeGroup
    ref="AG-CoordonneeSpherique" />
  </xs:extension>
224 </xs:complexContent>
  </xs:complexType>

  <xs:complexType
    name="TC-DirectionCartesienne">
229 <xs:complexContent>
  <xs:extension base="TC-Direction0">
    <xs:attributeGroup
    ref="AG-CoordonneeCartesienne" />
  </xs:extension>
234 </xs:complexContent>
  </xs:complexType>

  <xs:complexType
    name="TC-PointCartesienne">
239 <xs:complexContent>
  <xs:extension base="TC-Point">
    <xs:attributeGroup
    ref="AG-CoordonneeCartesienne" />
  </xs:extension>
244 </xs:complexContent>
  </xs:complexType>
</xs:schema>

```

Listing 1.4 – XSD validant la structure XML du listing 1.2

1.3.2 Base de données embarquée SQLite

Au regard des différentes problématiques, l'idée d'utiliser une base de données est à prendre en considération. Une base de données est en effet un ensemble structuré et organisé qui permet

de stocker de grandes quantités d'informations de manière cohérente. Afin d'en faciliter l'exploitation, une base est généralement utilisée au travers d'un SGBD. Il est ainsi possible d'ajouter, de modifier et surtout de rechercher des données au moyen d'une requête.

Les SGBDR sont actuellement les plus courants. Les données sont représentées sous forme de tables pouvant être mises en relation. Chaque table définit d'ailleurs une relation entre les différents champs qui la composent. Si le SGBD gère la représentation physique des données, il faut en revanche établir un modèle logique de données pour définir l'arrangement des informations au sein de la base de données [25].

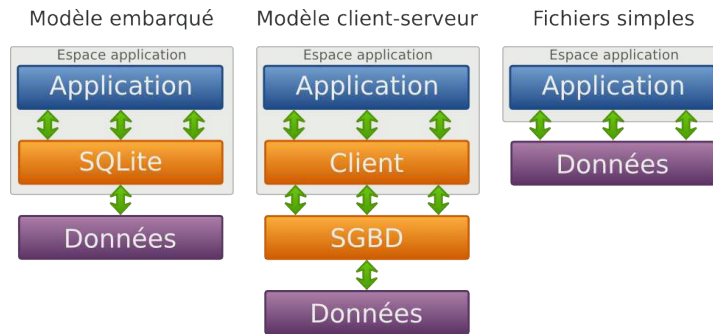


FIGURE 8 – Comparaison d'architectures de SGBD. Source : [26]

Comme le montre la figure 8, SQLite diffère des SGBD traditionnels par l'utilisation d'un modèle embarqué plutôt que le traditionnel modèle client-serveur. En fait, chaque base de données SQLite est autonome : elle est enregistrée dans un fichier dédié comprenant ses déclarations, ses tables, ses index et bien évidemment ses données. De ce fait, une base SQLite peut se substituer à une structure XML en offrant sensiblement la même autonomie aux données. [27]

1.4 Évaluation des solutions

Afin de déterminer la meilleure solution pour répondre aux problématiques, une approche itérative a été adoptée. Plusieurs prototypes ont donc été réalisés en implémentant successivement les techniques présentées ci-après. L'objectif de ces prototypes est de permettre une traduction d'un échantillon de fichier de résultats de simulation vers une nouvelle structure définie puis sa relecture pour un chargement en mémoire des données. Différents critères ont permis d'évaluer la performance de chaque solution :

- le temps d'exécution ;
- le poids des données produites ;
- la charge mémoire occasionnée ;
- la facilité d'utilisation et d'implémentation.

Pour cette phase d'étude, le langage Java est d'abord adopté car il permet la réalisation rapide de prototypes. Le concept mis en œuvre au sein du prototype doit à terme être adapté et intégré à l'outil de visualisation. Il est donc nécessaire que son implémentation soit possible dans le même langage, c'est-à-dire Delphi. Pour économiser du temps de développement, cette étape n'est réalisée que lorsque le prototype Java équivalent semble correspondre aux critères d'évaluation.

La figure 9 représente le diagramme de classes UML servant de base au prototype de transformation des données de simulation. Un groupe de rayons est composé de cinq rayons et de plusieurs polarisations. Lors du chargement des groupes de rayons, on ne sélectionne que ceux ayant une intersection avec l'orbite. L'exploration d'une des pistes d'optimisation du prototype a introduit la notion de cube afin de mailler l'espace et d'accélérer le chargement des rayons. Le développement

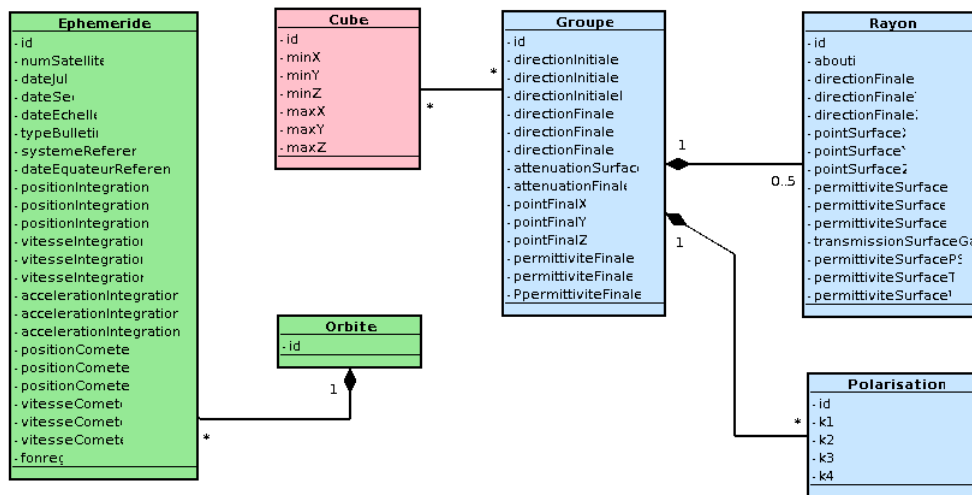


FIGURE 9 – Modèle de classes UML utilisé par les différents prototypes.

du prototype Java s'est déroulé en plusieurs étapes, chacune essayant d'améliorer les performances globales. Tout d'abord, un fichier XML contenant les résultats de simulation a été généré à partir du fichier source. On constate une augmentation de presque 50% du poids final mais cela reste acceptable. Une lecture complète du fichier a ensuite été réalisée avec SAX pour instancier les objets groupes et rayons. La figure 10 schématise ce processus.

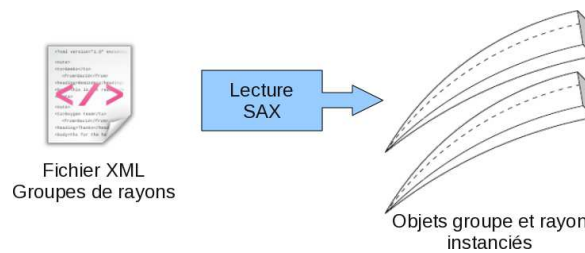


FIGURE 10 – Principe de fonctionnement du prototype n°1.

L'algorithme doit principalement être implémenté dans l'outil de visualisation. Or celui-ci ne s'intéresse qu'aux rayons croisant l'orbite. Il est cependant nécessaire de conserver l'ensemble des résultats d'une simulation. En effet, ceux-ci doivent rester indépendants de toute orbite afin de permettre un changement d'orbite à la volée au sein de l'outil de visualisation lors d'une interprétation de simulation. Il est donc nécessaire de choisir une autre approche pour faciliter une lecture rapide des seules données utiles.

La deuxième version du prototype reprend une idée mise en œuvre dans l'outil de visualisation. Comme le montre la figure 11, l'espace englobant l'orbite est « maillé » sous la forme de cubes.

Chaque cube est caractérisé par les coordonnées de ses sommets inférieur et supérieur ainsi que par la liste des groupes de rayons le traversant. Dans ce prototype, les résultats de simulation sont « éclatés » en plusieurs fichiers à raison d'un fichier par cube. Un fichier supplémentaire est constitué pour indexer les cubes, et donc les fichiers, en fonction de leurs coordonnées. Ce fichier d'index est lu avec DOM afin d'utiliser des requêtes Xpath pour obtenir le cube correspondant à chaque point d'orbite. Les fichiers de cubes sont ensuite lus avec DOM ou avec SAX. La figure 12 schématise ce processus.

L'inconvénient de cette approche est que les informations de rayons sont redondantes entre les différents fichiers de cubes traversés. Il en résulte alors une excessive augmentation de près de 1000% du poids global des données ce qui occasionnera certainement des problèmes de stockage par la suite rendant de fait cette solution inadaptée. Par ailleurs, on s'aperçoit que même

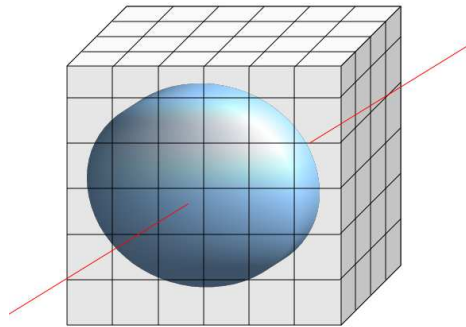


FIGURE 11 – Maillage cubique de l'espace englobant l'orbite.

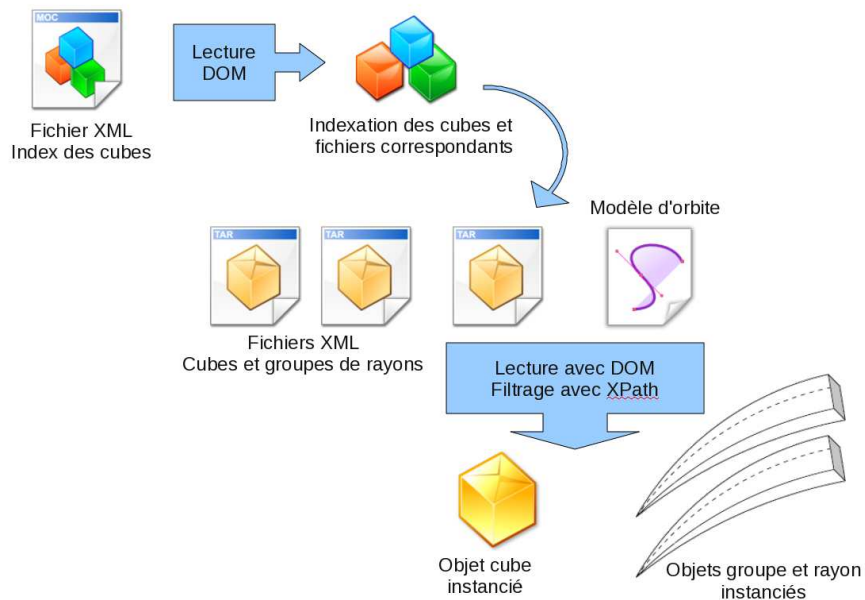


FIGURE 12 – Principe de fonctionnement du prototype n°2.

en travaillant sur des fichiers moins volumineux, l'utilisation de DOM et de requêtes Xpath ne permet pas un chargement rapide des données. La troisième version du prototype conserve l'idée du maillage mais adopte une solution différente pour le stockage des données. Chaque enregistrement de cube ne contient plus les groupes de rayons associés mais seulement leurs identifiants. Les groupes de rayons sont stockés dans un seul autre fichier à l'instar du premier prototype. On charge tout d'abord l'orbite en mémoire. Ensuite, on parcourt le fichier d'index des cubes avec SAX et on ne récupère que les cubes – et donc les identifiants de groupes de rayons – utiles pour l'orbite. Enfin, on parcourt avec SAX le fichier unique de résultats de simulation duquel on ne récupère que les groupes de rayons identifiés précédemment. La figure 13 schématise ce processus. Ainsi, le volume de données générées (un peu plus de 60% d'augmentation par rapport au poids initial), le temps de chargement et l'empreinte mémoire sont minimisés.

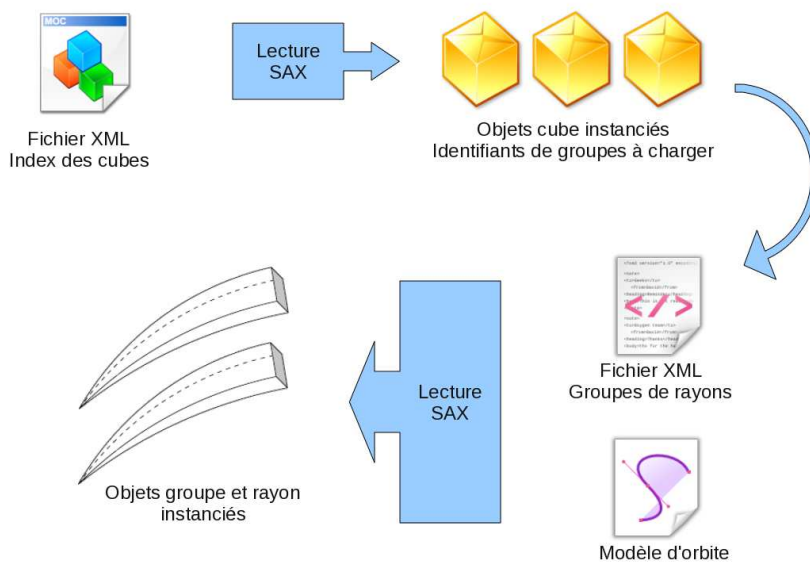


FIGURE 13 – Principe de fonctionnement du prototype n°3.

La dernière version du prototype est basée sur la précédente. On remplace simplement la structure de données XML des résultats de simulation par une base de données SQLite. La figure 14 présente le modèle conceptuel ayant servi pour la création de cette base. On récupère les informations d'un groupe de rayons par l'intermédiaire d'une requête SQL. Il est par ailleurs possible d'utiliser SQLite pour structurer l'index des cubes.

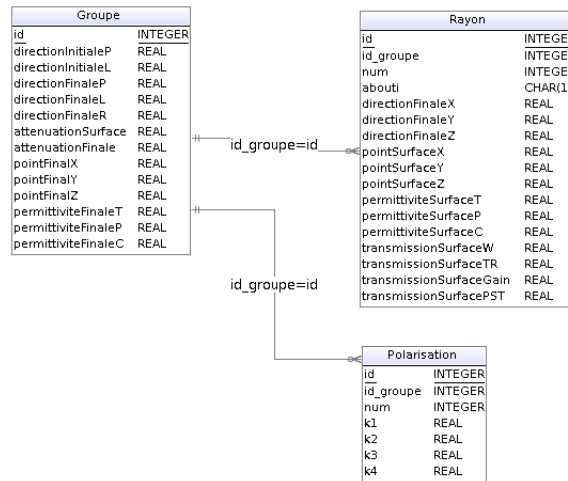


FIGURE 14 – Modèle de base de données utilisé dans le prototype n°4.

1.5 Bilan

Le tableau 1 synthétise les mesures effectuées lors du test des différents algorithmes avec un échantillon de données de 45,2 Mo, c’est-à-dire 19047 groupes, soit 95235 rayons.

Nous constatons que le prototype n°2 est vraiment à écarter. Nous pouvons noter que les résultats du prototype n°4 sont absents. En effet, suite à certains problèmes d’implémentation de SQLite, ce prototype n’a pas été réalisé entièrement. Les premiers tests sur d’autres échantillons non-comparables laissaient entrevoir une solution peu efficace en temps d’accès. Une étude plus poussée avec un schéma de base optimisé pourrait peut-être améliorer les résultats.

La meilleure solution est donc à choisir entre celles implémentées dans les prototypes n°1 et n°3. L’indexation des cubes telle que proposée dans la méthode 3 suscite une légère augmentation du volume de données générées mais permet de minimiser la charge lors du chargement. Cette dernière méthode nous semble donc celle à privilégier lorsqu’il faudra implémenter les nouvelles structures au sein des logiciels de la plateforme.

	Implémentation actuelle	Prototype 1	Prototype 2	Prototype 3	Prototype 4
Type de structure	Fichiers plats	XML	XML	XML	SQLite
Nombre de fichiers	1 fichier de résultats	1 fichier de résultats	1 index cubes et 1 fichier/cube	1 index cubes et 1 fichier de résultats	1 index cubes et 1 base SQLite
Poids global	45,2 Mo	67,5 Mo	496,2 Mo	73,2 Mo	×
Chargement des cubes	-	-	DOM et Xpath	SAX	SAX
Chargement des rayons	-	SAX	DOM	SAX	SQL
Temps global	4 s	10 s	30 s	10 s	×
Charge	Faible	Moyenne	Élevée	Faible	Faible
Facilité de mise en œuvre	XXX	XXX	XXX	XX	XX

TABLE 1 – Synthèse des différents prototypes

 Étude de la plateforme de simulation existante

2.1 Architecture de la plateforme

La plateforme de simulation actuelle est hétérogène. Comme le montre la figure 15, elle est composée principalement de deux outils logiciels mais elle fait intervenir plusieurs éléments d'infrastructure qui seront présentés ci-après.

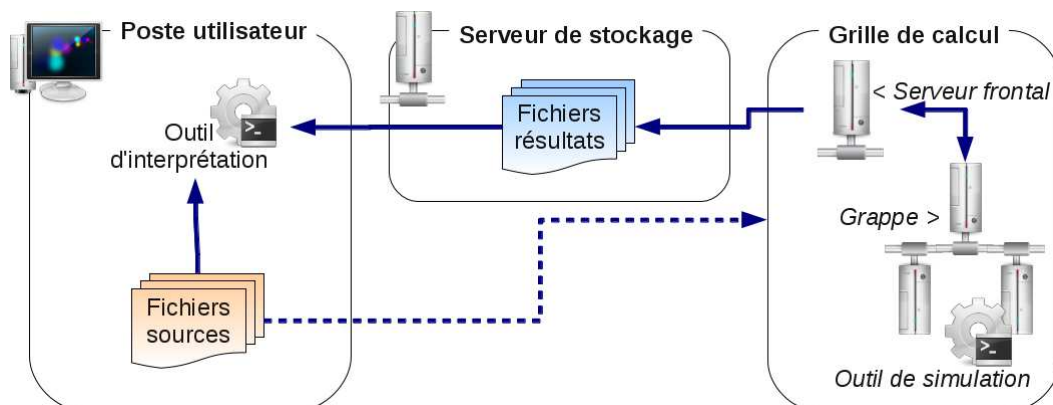


FIGURE 15 – Vue simplifiée de l'architecture actuelle de la plateforme de simulation de l'expérience CONSERT.

2.1.1 Éléments d'infrastructure

2.1.1.1 Poste utilisateur

Le poste de l'utilisateur est une simple machine du LPG fonctionnant avec le système d'exploitation Microsoft Windows. Il permet d'exécuter le logiciel de visualisation des résultats de la simulation. C'est également depuis cette machine que l'utilisateur se connecte aux différents serveurs pour échanger les données ou soumettre un calcul.

2.1.1.2 Serveur stockage

Le serveur de stockage Venus est situé au LPG. Cette machine exploitant une version 32bits de GNU/Linux Centos 5 dispose de 7 To d'espace disque utile. Les données produites par la simulation y sont automatiquement rapatriées depuis la grille de calcul via un point montage SSHFS spécifique.

2.1.1.3 Environnement de calcul intensif

La recherche scientifique contribue depuis longtemps au développement des outils informatiques. Outre le besoin évident d'outils logiciels spécifiques, c'est en effet un domaine où les exigences sont sans cesse repoussées en terme d'infrastructure, de puissance de calcul ou encore d'espace de stockage. Malgré les espoirs fondés sur la loi de Moore [28], même les machines actuelles n'offrent pas encore une puissance suffisante pour réaliser rapidement certains calculs intensifs. Avec le développement des réseaux informatiques dans les années 1960, certains ont eu l'idée de relier plusieurs machines pour obtenir une puissance de calcul plus importante. Les concepts de grappe (« cluster » en anglais) et de grille (« grid » en anglais) sont nés.

2.1.1.3.1 Grappes de calcul

Une grappe de serveurs – appelée aussi « ferme de calcul » – est une technique qui consiste à regrouper plusieurs ordinateurs indépendants – appelés nœuds (« nodes » en anglais) – qui vont apparaître comme une seule entité aux yeux des utilisateurs. Ce « super-ordinateur » dispose alors de beaucoup plus de capacités que pourrait offrir une seule machine. Chaque nœud est relié à une infrastructure réseau physique dédiée et généralement de très haute performance. Un ordinateur central – appelé « ordonnanceur » ou « frontale » – sert de point d'entrée et assure la répartition des travaux sur les différents nœuds. De ce fait, les grappes sont très utilisées pour les calculs parallèles. Cet usage optimisé des ressources permet notamment :

- d'augmenter la disponibilité ;
- de faciliter la montée en charge ;
- de mieux répartir la charge ;
- de faciliter la gestion des ressources (processeur, mémoire vive, disques dur, bande passante réseau).

La figure 16 schématise le fonctionnement d'une grappe « cluster A ». Celle-ci contient deux nœuds : « Node1 » et « Node2 ». Sur le nœud « Node1 » sont lancés deux travaux – « job1 » et « job2 » – eux-mêmes composés de processus (« process »). Les travaux des nœuds sont gérés par un ordonnanceur (« Batch scheduler ») qui lui-même est employé par les applications des utilisateurs.

2.1.1.3.2 Grilles de calcul

Une grille informatique peut être définie comme « une infrastructure virtuelle constituée d'un ensemble de ressources informatiques potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes » [30]. Une grille permet de faire du calcul distribué : elle exploite la puissance de calcul de milliers d'ordinateurs afin de donner l'illusion d'un ordinateur virtuel très puissant. Ce modèle permet de résoudre d'importants problèmes de calcul nécessitant des temps d'exécution très longs en environnement « classique ». [31]

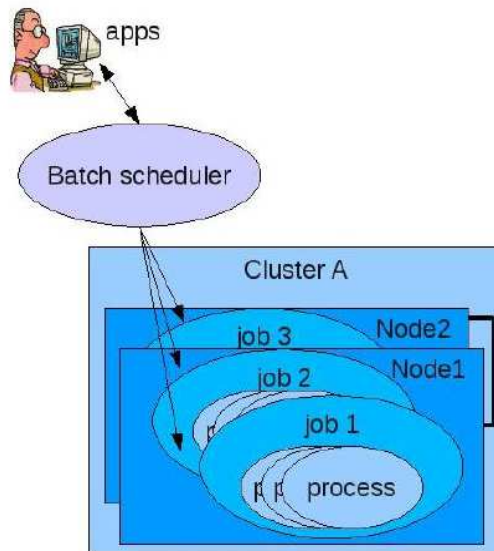


FIGURE 16 – L'architecture d'une grappe. Source : [29]

Comme le montre la figure 17, le concept de grille de calcul se rapproche d'un simple ensemble de grappes. Son infrastructure est cependant plus abstraite est plus complexe. En effet, une grille tend à apporter les mêmes services qu'une grappe mais avec des matériels et logiciels hétérogènes. Ainsi, l'utilisateur accédera aux ressources de stockage (« Storage resource ») ou de calcul – les grappes – par l'intermédiaire d'un intergiciel de grille (« middleware »). Ce-dernier est en quelque sorte un super-ordonnanceur qui va ensuite répartir les travaux entre les grappes et les transmettre aux ordonnanceurs concernés.

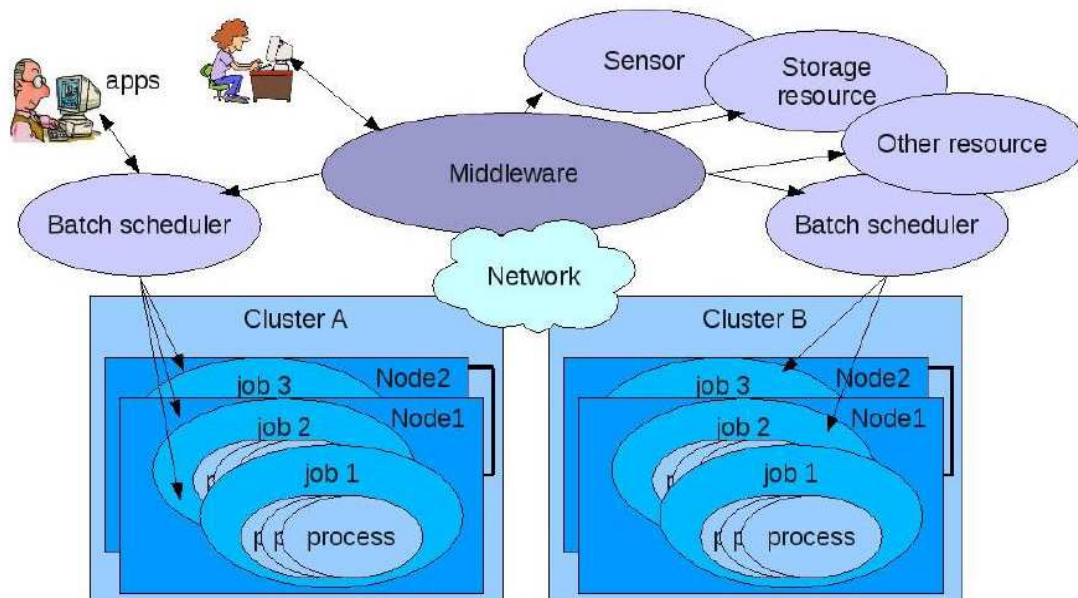


FIGURE 17 – L'architecture d'une grille. Source : [29]

2.1.1.3.3 Grille locale du projet CIMENT

Né en 1998, le projet CIMENT est le fruit de l'implication de nombreux partenaires tels que l'INPG, l'UJF, le CNRS, l'IMAG, l'INRIA, le MENRT, la Région Rhône-Alpes, la Métro et la

Ville de Grenoble. CIMENT participe également au projet CIRA avec la Fédération Lyonnaise de Calcul Haute Performance et au projet MUSCA de l'Université de Savoie. [32] [33]

Son but est de favoriser le développement cohérent de plateformes matérielles et logicielles pour la modélisation numérique et l'expérimentation du calcul intensif. Pour cela, il vise à :

- donner la possibilité aux modélisateurs de développer et tester les codes en local pour une utilisation plus rationnelle des ressources nationales ;
- fédérer les communautés et renforcer des collaborations autour du calcul intensif ;
- utiliser au mieux les ressources de calcul locales et nationales ;
- former les utilisateurs aux techniques du calcul scientifique, en particulier du calcul parallèle.

La figure 18 schématise le réseau de grappes constituant la grille CIMENT.

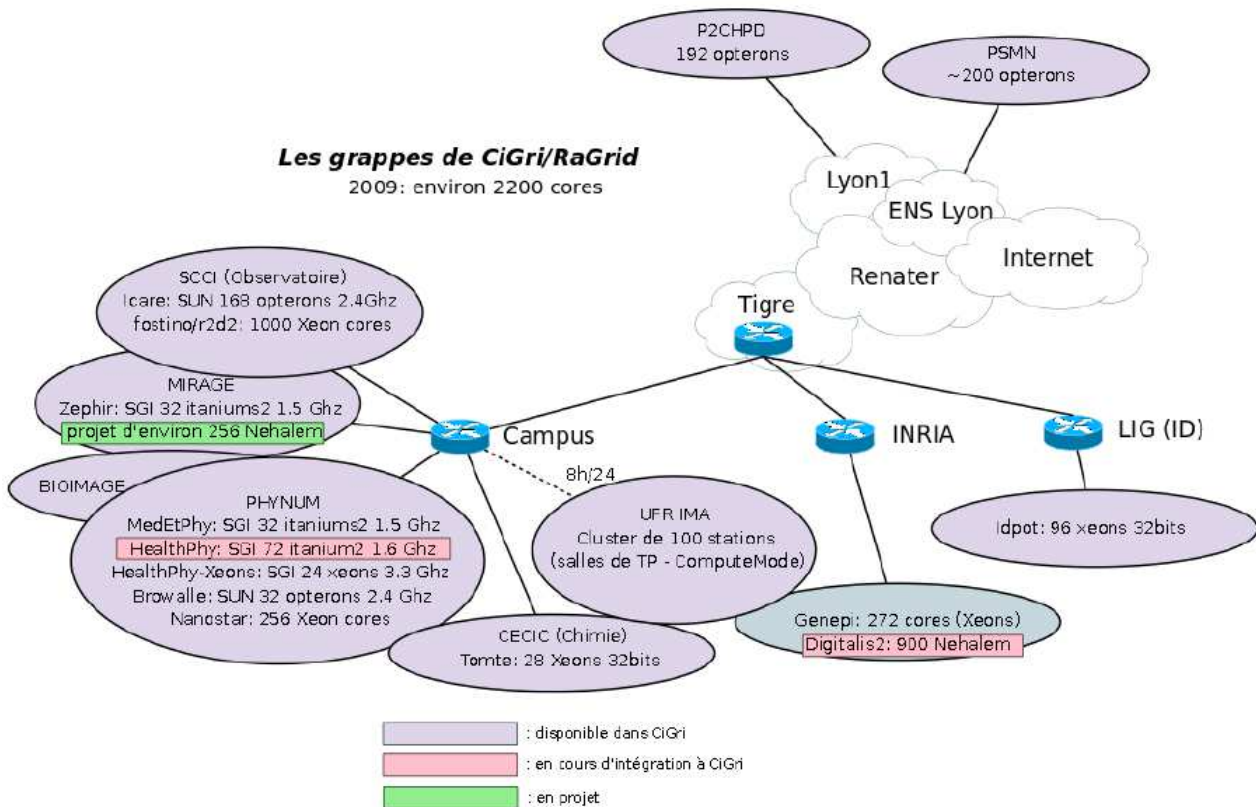


FIGURE 18 – Les grappes de CiGri/RaGrid. Source : [29]

2.1.2 Logiciels

2.1.2.1 Outil de simulation

Un premier logiciel de la plateforme utilise les modèles de forme et de composition du noyau cométaire. A partir de ces informations et de quelques paramètres, il calcule la propagation des rayons à travers le noyau jusqu'à la surface. À leur sortie, les rayons sont qualifiés avec une direction, un temps d'arrivée et une puissance. Pour déterminer la puissance d'un rayon, quatre autres rayons légèrement divergents sont également simulés [14].

Dans le cas d'un noyau homogène, les rayons sont propagés en ligne droite. En revanche, dans le cas contraire, l'algorithme prend en compte les particularités décrites dans le modèle de composition interne pour calculer la direction des rayons réfléchis et réfractés [6].

Cet outil, développé en Fortran, s'exécute en ligne de commande. Les différents paramètres nécessaires sont rassemblés au sein d'un fichier de paramétrage. Chaque ligne contient dans l'ordre un nom de paramètre, sa description textuelle, sa valeur et son unité. Il définit globalement :

- les noms des fichiers d'entrée (modèles) et de sortie (résultats) ;
- la position de l'atterrisseur dans le repère de la comète ;
- le pas de discrétisation (angle d'écart entre les différents rayons simulés) ;
- les coefficients de permittivité ;
- le nombre limite de réflexions autorisées pour un rayon ;
- la surface maximale formée par la divergence des quatre rayons secondaires ;
- le seuil limite de puissance nécessaire à l'exploitation des rayons.

L'algorithme calcule la propagation de rayons émis depuis un point donné et ce, dans toutes les directions sans contrainte particulière. Son implémentation Fortran est itérative et n'utilise pas de technique de programmation concurrente. Cependant, grâce à certains de ses paramètres, le programme peut restreindre le calcul sur un angle d'émission déterminé. La parallélisation qui en découle est seulement « externe » : il suffit de morceler le calcul global en sous-calculs d'angles réduits pour le distribuer sur les différents nœuds de la grille locale de calcul intensif CIMENT.

2.1.2.2 Outil de visualisation

À partir des données produites, ce deuxième logiciel utilise des modèles de forme et d'orbite pour offrir une représentation 3D de la scène comme le montre la figure 19. Cet outil, développé en Delphi, fonctionne sur le poste Windows de l'utilisateur.

Avec son interface graphique destinée aux scientifiques, cet outil permet surtout d'exploiter les résultats de façon synthétique et intuitive [34]. Il propose par exemple :

- une représentation 3D du noyau de la comète en fonction du modèle de forme sélectionné ;
- une représentation de la sonde et de son orbite en fonction du modèle sélectionné ;
- un affichage, après interpolation, des rayons arrivant à l'orbite ;
- une visualisation de la trajectoire des rayons à travers le noyau ;
- des informations sur un rayon précis (temps d'arrivée, puissance du signal, ...)
- un affichage de la trace au sol ;
- un affichage du signal reçu par l'orbiteur ;
- une manipulation en temps réel de la scène (déplacement, rotation, zoom, ...).

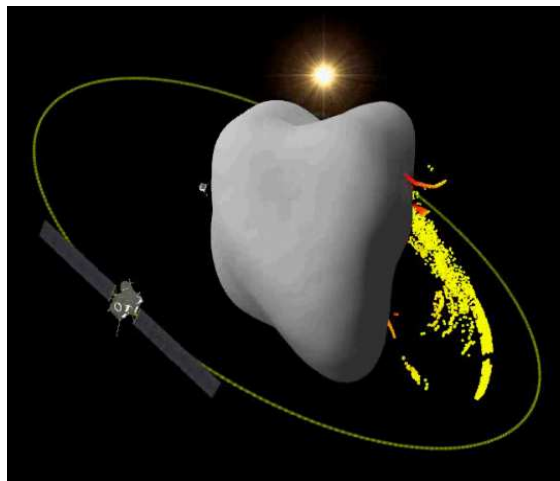


FIGURE 19 – Exemple de visualisation d'une simulation avec ondes propagées. Source : [34]

2.2 Chaîne de simulation

Comme le montre la figure 20, le processus de génération d'une simulation comprend trois étapes : la préparation du calcul et son paramétrage, l'exécution du calcul et le rapatriement des résultats puis l'interprétation des résultats. Ces étapes sont détaillées ci-après.

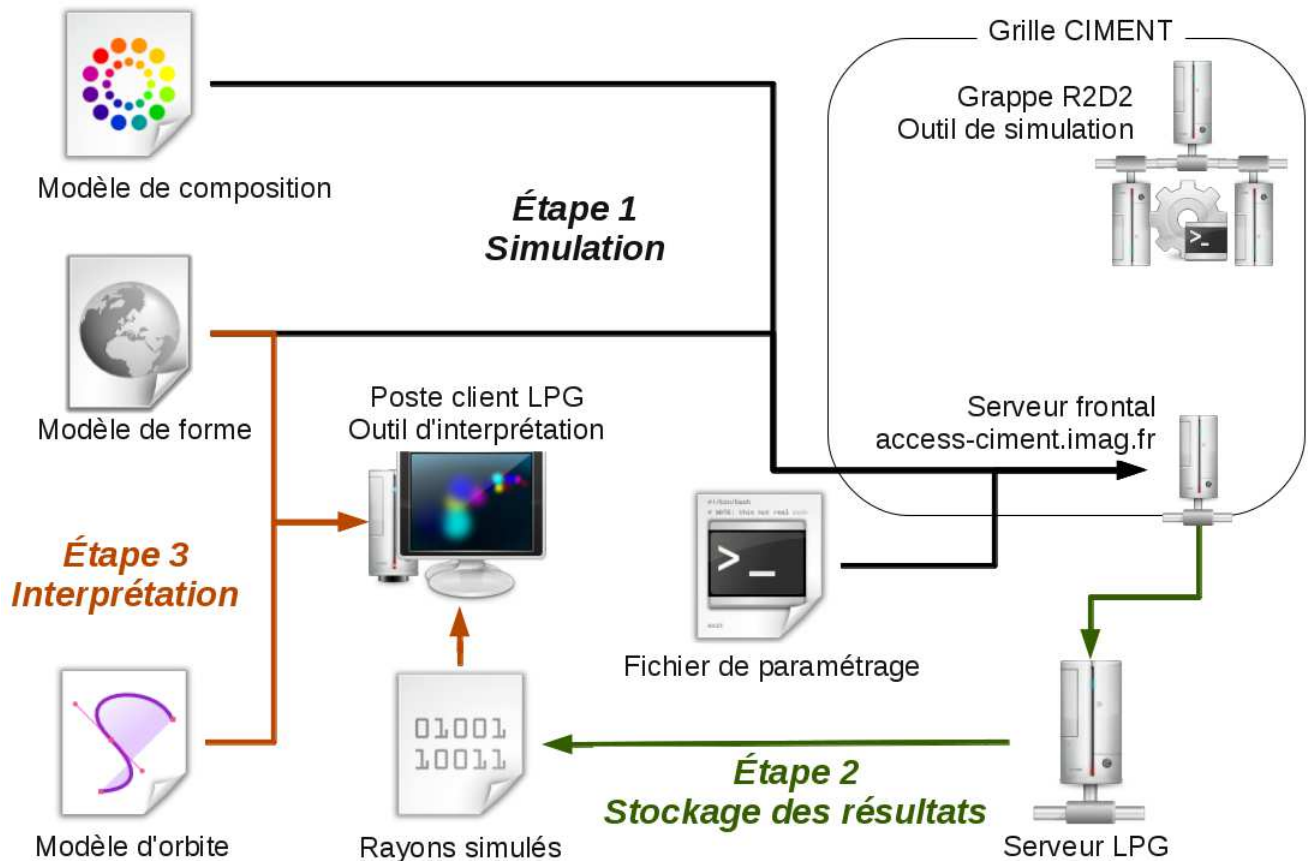


FIGURE 20 – Architecture de la plateforme actuelle de simulation.

2.2.1 Étape n°1 : déploiement des données sources sur la grappe

Lors de la première étape, on sélectionne un modèle de forme de comète et un modèle de composition. On spécifie, au sein d'un fichier spécifique, les paramètres nécessaires au programme. Ces trois fichiers sont envoyés sur le serveur frontal de la grille de calcul par l'intermédiaire d'une connexion SSH pour être ensuite déployés sur les différentes grappes.

2.2.2 Étape n°2 : lancement du calcul sur la grappe

2.2.2.1 Initialisation de l'environnement de calcul

Pour pouvoir utiliser le logiciel de simulation, il faut au préalable le déployer sur l'ensemble des hôtes utilisés pour le calcul. Pour cela, depuis le serveur frontal de la grille CIMENT, on exécute le script suivant qui, pour chaque grappe souhaitée :

- télécharge les sources du programme, le script de lancement et les fichiers de données ;

- rend exécutable le script de lancement ;
- compile le programme.

Comme on peut le constater dans le listing 2.1, les grappes n'étant pas forcément de même architecture ou de même système d'exploitation, il est nécessaire d'adapter l'environnement de compilation à chaque cas. Cela comprend donc :

- l'utilisateur ;
- le dossier ;
- le compilateur ;
- les options de compilation.

```
#!/bin/bash
APPLI=WAVE3dc3_v15.f
4 APPLIComp=WAVE3dc3_v15
LANCEUR=lance_cigri.sh

echo "copie du source sur medetphy"
9 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@medetphy.imag.fr:wave3d
scp -r a_deployer/template chaillos@medetphy.imag.fr:wave3d
ssh chaillos@medetphy.imag.fr chmod +x wave3d/$LANCEUR

echo "copie du source sur zephir"
14 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@zephir.mirage.ujf-grenoble.fr:wave3d
scp -r a_deployer/template chaillos@zephir.mirage.ujf-grenoble.fr:wave3d
ssh chaillos@zephir.mirage.ujf-grenoble.fr chmod +x wave3d/$LANCEUR

echo "copie du source sur genepi"
19 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@genepi.imag.fr:wave3d
scp -r a_deployer/template chaillos@genepi.imag.fr:wave3d
ssh chaillos@genepi.imag.fr chmod +x wave3d/$LANCEUR

echo "copie du source sur browalle"
24 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@browalle.ujf-grenoble.fr:wave3d
scp -r a_deployer/template chaillos@browalle.ujf-grenoble.fr:wave3d
ssh chaillos@browalle.ujf-grenoble.fr chmod +x wave3d/$LANCEUR

echo "copie du source sur fostino"
29 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@fostino.obs.ujf-grenoble.fr:wave3d
scp -r a_deployer/template chaillos@fostino.obs.ujf-grenoble.fr:wave3d
ssh chaillos@fostino.obs.ujf-grenoble.fr chmod +x wave3d/$LANCEUR

echo "copie du source sur r2d2"
34 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@r2d2.obs.ujf-grenoble.fr:wave3d
scp -r a_deployer/template chaillos@r2d2.obs.ujf-grenoble.fr:wave3d
ssh chaillos@r2d2.obs.ujf-grenoble.fr chmod +x wave3d/$LANCEUR

echo "copie du source sur icare"
39 scp a_deployer/$APPLI a_deployer/$LANCEUR chaillos@icare.obs.ujf-grenoble.fr:wave3d
scp -r a_deployer/template chaillos@icare.obs.ujf-grenoble.fr:wave3d
ssh chaillos@icare.obs.ujf-grenoble.fr chmod +x wave3d/$LANCEUR

echo ""
44 echo ""

echo "compilation sur medetphy"
ssh chaillos@medetphy.imag.fr ifort -g wave3d/$APPLI -o wave3d/$APPLIComp

echo "compilation sur zephir"
49 ssh chaillos@zephir.mirage.ujf-grenoble.fr ifort -g wave3d/$APPLI -o wave3d/$APPLIComp

echo "compilation sur genepi"
ssh chaillos@genepi.imag.fr g77 -g wave3d/$APPLI -o wave3d/$APPLIComp

54 echo "compilation sur browalle"
ssh chaillos@browalle.ujf-grenoble.fr g77 -g wave3d/$APPLI -o wave3d/$APPLIComp

echo "compilation sur fostino"
59 ssh chaillos@fostino.obs.ujf-grenoble.fr f95 -g wave3d/$APPLI -o wave3d/$APPLIComp

echo "compilation sur r2d2"
ssh chaillos@r2d2.obs.ujf-grenoble.fr f95 -g wave3d/$APPLI -o wave3d/$APPLIComp
```

```
64 echo "compilation sur icare"
ssh chailllos@icare.obs.ujf-grenoble.fr source .profile /opt/SUNWspro/bin/f95 wave3d/$APPLI -o
wave3d/$APPLIComp
```

Listing 2.1 – Exemple de script de déploiement

2.2.2.2 Paramétrage global du logiciel de simulation

Lors de son appel en ligne de commande, le logiciel de simulation dispose d'un certain nombre de paramètres d'entrée qui doivent être spécifiés. Pour cela, on les définit au sein du fichier « params.prg » tel que présenté dans le listing 2.2.

```
1 <----- DESCRIPTIF -----> = <----- VALEUR -----><-- DIM -->
* CFILE1 : F.I des parametres de la forme du noyau = 67PA_m.dat
* CFILE2 : F.I des parametres des cometesi. = cometis3dl.dat
* CFILE3 : F.O des mesures de phases & d'att. = fichier_sortie.dat
* CFILE4 : F.O des chemins optiques courbes = wave3d07a_cxxx.dat
6 * CFILE5 : F.O des temps de propagation = prop3d07a_cxxx.dat
* CFILE6 : F.I du diagramme de polarisation Lander = lander_polar.dat
* CFILE7 : F.O fichier log de suivi = suivi.txt

* fRMAX1 : Rayon du le cercle limite englobant le noyau= 1700. m
11 * fRMAX2 : Rayon du 2ieme cercle limite = 1800. m
* fRMAX3 : Rayon de l'orbite = 5000. m
* RMIN : Rayon de la sphere min = 680. m
* RMAX : Rayon de la sphere max = 720. m
* frX0 : Coordonne X du centre de l'orbite = 0. m
16 * frY0 : Coordonne Y du centre de l'orbite = 0. m
* frZ0 : Coordonne Z du centre de l'orbite = 0. m
* SH1 : Pas d'integration pour R<fRMAX1 = 5. m
* SH2 : Pas d'integration pour fRMAX1<R<fRMAX2 = 10. m
* SH3 : Pas d'integration final pour fRMAX2<R = 5. m
21 * fPLAND : Latitude de l'atterrisseur = 180. degrees
* fLLAND : Longitude de l'atterrisseur = 0. degrees
* XLAN_X : coordonnee X du X LANDER = -1. -
* XLAN_Y : coordonnee Y du X LANDER = 0. -
* XLAN_Z : coordonnee Z du X LANDER = 0. -
26 * YLAN_X : coordonnee X du Y LANDER = 0. -
* YLAN_Y : coordonnee Y du Y LANDER = 1. -
* YLAN_Z : coordonnee Z du Y LANDER = 0. -
* ZLAN_X : coordonnee X du Z LANDER = 0. -
* ZLAN_Y : coordonnee Y du Z LANDER = 0. -
31 * ZLAN_Z : coordonnee Z du Z LANDER = -1. -
* SCALE : Pas de discretisation en = 0.5 degrees-1
* fPstep : Pas de discretisation de la polarisation = 5. degrees
* fpmoy : Permittivite moyenne du milieu cometaire = 1.9 -
* fpvide : permittivite du vide = 1. -
36 * fLWAVE : Longueur d'onde dans le vide de l'onde = 3.3 m
* IREFMAX : Nombre de reflexions maximales = 3 -
* NBSURF : Nombre de point max a la surface = 4 -
* fpAS : Ouverture du rayon pour le calcul de la puiss= 0.01 degrees
* SMAX : Surf. max. de divergence = 80000. m2
41 * SMIN : Surf. min. de divergence = 0.00006 m2
* SEUIL : Limite de puissance des donnees exploitables= -320.
```

Listing 2.2 – Exemple de fichier de paramètres d'entrée « params.prg »

2.2.2.3 Paramétrage spécifique du logiciel de simulation

Pour rappel, la simulation de l'expérience repose sur la propagation de rayons à travers le noyau d'une comète. Le calcul s'effectue de manière séquentielle en suivant un « pas de discrétisation », c'est-à-dire que chaque rayon propagé sera espacé de x degrés du précédent sur une de ses coordonnées. Chaque rayon est donc indépendant des autres et le calcul de sa propagation peut être réalisé individuellement. L'utilisation de la grille CIMENT et de ses grappes devient évidente.

Ainsi, plutôt que d'attendre une longue succession de petits calculs sur une même machine, ceux-ci seront exécutés parallèlement sur les nombreuses machines mises à disposition pour produire un fragment du résultat global. Il faut donc générer une série de paramètres spécifiques à chaque « sous-calcul ». C'est le but du listing 2.3.

```
#!/bin/bash
thetaMin=0
3 thetaMax=180
  phiMin=0
  phiMax=360
  theta=$thetaMin
  pasTheta=2
8 pasPhi=2
  while [ $theta -lt $thetaMax ]
  do
    s=$(( $theta ))
    theta=$(( $theta + $pasTheta ))
13    if [ "$theta" -gt "180" ]
    then
      u="180"
    else
      u=$(( $theta ))
18    fi
    phi=$phiMin
    while [ $phi -lt $phiMax ]
    do
      v=$(( $phi ))
      phi=$(( $phi + $pasPhi ))
23    if [ "$phi" -gt "360" ]
    then
      w="360"
    else
      w=$(( $phi ))
28    fi
    echo output$theta_"$phi params.prg $s $u $v $w
  done
done
```

Listing 2.3 – Script générant le fichier de paramètres de lancement des tâches « params.txt »

Le listing 2.4 présente un exemple de contenu produit en sortie par ce script (fichier params.txt).

```
output2_2 params.prg 0 2 0 2
3 output2_4 params.prg 0 2 2 4
  output2_6 params.prg 0 2 4 6
  output2_8 params.prg 0 2 6 8
  output2_10 params.prg 0 2 8 10
  output2_12 params.prg 0 2 10 12
  output2_14 params.prg 0 2 12 14
8  output2_16 params.prg 0 2 14 16
  output2_18 params.prg 0 2 16 18
  output2_20 params.prg 0 2 18 20
  output2_22 params.prg 0 2 20 22
```

Listing 2.4 – Exemple de fichier params.txt

Au final, pour chaque appel au programme de simulation, il faut agréger les paramètres globaux et les paramètres spécifiques précédemment générés. Cela s'effectue par le biais du script « lancecigri.sh » présenté par le listing 2.5.

```
#!/bin/bash
set -e
4 APPLI=WAVE3dc3_v15

if [ $# -ne 6 ]
then
  echo "Usage: $0 <output> <config_file> <theta_min> <theta_max> <phi_min> <phi_max>"
9  exit 1
fi

mkdir -p $1
```

```

14 cp -f template/* $1
   cd $1
   perl -pi -e "s/%params/$2/" stdin
   perl -pi -e "s/%theta_min/$3/" stdin
19  perl -pi -e "s/%theta_max/$4/" stdin
   perl -pi -e "s/%phi_min/$5/" stdin
   perl -pi -e "s/%phi_max/$6/" stdin
   ../$APPLI < stdin

```

Listing 2.5 – Script de lancement de tâche

2.2.2.4 Lancement du logiciel de simulation

La grille CIMENT utilise un langage spécifique de description de tâches. Chaque soumission doit être spécifiée dans un fichier tel que présenté par le listing 2.6. Le bloc `DEFAULT` permet de spécifier des valeurs globales. Ainsi `name` définit le nom de la campagne de calcul et `paramFile` le fichier de paramétrage à utiliser.

Les blocs suivants concernent respectivement une grappe. Le paramètre `execDir` définit le répertoire d’exécution, `execFile` l’exécutable à lancer, `walltime` le temps maximal d’exécution autorisé avant destruction du processus et `priority` l’indice de priorité attribué à la grappe.

Le listing 2.6 décrit le contenu du fichier de paramétrage de la grille (wave3d.jdl).

```

DEFAULT{
3   name = wave3d ;
   # nbjobs = 6;
   paramFile = params.txt;
}

8 icare.obs.ujf-grenoble.fr{
   execDir = /home/ciment/chaillos/wave3d ;
   execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
   walltime = 10:00:00 ;
   priority = 3;
}

13 zephir.mirage.ujf-grenoble.fr{
   execDir = /home/ciment/chaillos/wave3d ;
   execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
   walltime = 20:00:00 ;
18 }
   priority = 10;
}

medetphy.imag.fr{
   execDir = /home/ciment/chaillos/wave3d ;
   execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
23 }
   walltime = 20:00:00 ;
   priority = 10;
}

genepi.imag.fr{
   execDir = /home/ciment/chaillos/wave3d ;
28 }
   execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
   walltime = 20:00:00 ;
   priority = 5;
}

browalle.ujf-grenoble.fr{
33 }
   execDir = /home/ciment/chaillos/wave3d ;
   execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
   walltime = 20:00:00 ;
   priority = 10;
}

38 healthphy.ujf-grenoble.fr{
   execDir = /home/ciment/chaillos/wave3d ;
   execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
   walltime = 20:00:00 ;
43 }
   priority = 10;
}

```

```

48 fostino.obs.ujf-grenoble.fr{
    execDir = /home/ciment/chaillos/wave3d ;
    execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
    walltime = 20:00:00 ;
    priority = 2;
}

53 r2d2.obs.ujf-grenoble.fr{
    execDir = /home/ciment/chaillos/wave3d ;
    execFile = /home/ciment/chaillos/wave3d/lance_cigri.sh ;
    walltime = 20:00:00 ;
    priority = 1;
}

```

Listing 2.6 – Exemple de fichier de paramétrage de la grille

La soumission d’une campagne de calcul s’effectue par la commande `gridsub` en spécifiant le fichier de paramétrage des tâches adéquat. Cela peut être automatisé par le script présenté dans le listing 2.7.

```

2 #!/bin/bash
  gridsub -f wave3d.jdl

```

Listing 2.7 – Exemple de fichier lance_cigri.sh

2.2.2.5 Contrôle de l’avancement du calcul

Le suivi d’une campagne de calcul s’effectue par la commande `gridstat` en spécifiant l’identifiant adéquat. Une interface web apporte les mêmes informations mais offre un contrôle plus fin sur la gestion d’erreur. Lorsqu’une tâche retourne une erreur, elle est signalée à l’utilisateur qui peut alors la corriger avant de re-soumettre le calcul.

La campagne peut à tout moment être stoppée soit via l’interface web, soit par la commande `griddel` en spécifiant l’identifiant adéquat.

2.2.2.6 Rapatriement des résultats

Les fichiers générés par les différents calculs sont automatiquement compactés et centralisés sur la frontale de la grille CIMENT. Pour obtenir un fichier exploitable par le reste des outils CONCERT, il est nécessaire de décompresser ces fichiers puis de les concaténer pour former un seul fichier : le fichier des résultats. Ce dernier fichier reste donc stocké sur le serveur frontal de la grille en attendant son rapatriement vers un serveur du laboratoire. Cette étape est actuellement automatisée via à un point de montage SSHFS entre le serveur frontal de la grille et le serveur de stockage du LPG.

2.2.3 Étape n°3 : interprétation des résultats

Enfin, on récupère le fichier contenant les résultats de la simulation sur le poste client pour le charger dans l’outil d’interprétation. On sélectionne un modèle d’orbite afin de filtrer certains rayons. En effet, seuls ceux atteignant l’orbiteur sont réellement intéressants pour la simulation. Il n’est donc pas nécessaire d’afficher tous les rayons. Par conséquent, filtrer est un moyen de faciliter la visualisation des données utiles.

2.3 Problématiques et objectifs

La plateforme actuelle est fonctionnelle mais son usage est un peu complexe. En effet, on manipule différentes données hétérogènes à travers divers outils indépendants. Les simulations sont soumises manuellement à la grille CIMENT et la progression des calculs est suivie par l'intermédiaire de l'interface dédiée. Les résultats rapatriés sur un serveur local sont ensuite chargés manuellement dans l'outil de visualisation. Aucun modèle, aucune simulation, aucun paramètre n'est archivé au cours de ce processus.

La principale attente des utilisateurs est d'avoir une interface unifiée et automatisée pour la plateforme afin de faciliter et d'alléger le processus de simulation de l'expérience CONSERT. Toutes les données et traitements devront être archivés afin de les retrouver par la suite.

L'interface graphique devra être ergonomique et intuitive. Elle permettra la sélection de paramètres (valeurs, scripts, logiciels de post-traitements...) afin de retrouver une ancienne simulation archivée ou d'en lancer une nouvelle. Pour chaque simulation, l'interface fournira à l'utilisateur des informations stockées en base telles que les modèles utilisés, la date de génération... Par souci de simplicité d'utilisation, un maximum de tâches devront être automatisées. Ainsi, lors d'une modification de modèles, les simulations archivées qui en dépendent devront être régénérées et archivées à leur tour. Une représentation arborescente des modèles et simulations est donc souhaitable.

Le système permettra de soumettre automatiquement la génération des simulations aux grilles de calculs configurées. L'interface devra donc permettre de gérer ces grilles et leurs paramètres de connexion tout en offrant la possibilité d'en ajouter. Les grilles et les grappes étant hétérogènes, il sera nécessaire de les initialiser notamment par la compilation du logiciel de simulation en fonction de leur architecture.

L'interface permettra de sélectionner une grille particulière pour la soumission d'une simulation. En fonction des possibilités offertes par les grilles, l'interface devra recueillir des statistiques sur chaque simulation (temps de calcul, état d'avancement...). Il sera également possible de soumettre un groupe de simulations pour étudier différents modèles, différents niveaux d'échantillonnage, différentes positions de l'atterrisseur... Les contraintes de priorités devront être prises en compte dans la stratégie de soumission choisie. En cas d'erreur, une simulation pourra être de nouveau soumise automatiquement par le système. Des alertes seront à définir suivant le degré d'erreur.

L'interface permettra à terme de générer automatiquement des pré-visualisations et des post-traitements pour chaque simulation. Idéalement intégré – du moins, en partie – à l'outil de visualisation, le système devra minimiser le volume des données stockées sur le poste client tout en permettant une utilisation autonome. La compression des données pourra donc être envisagée afin de réduire le volume de données stockées et d'optimiser les transferts réseaux suscités par leur utilisation. Par ailleurs, l'interface devra être réactive. Les plus gros calculs pourront être préparés à l'aide des grilles de calculs.

Grilles de calcul : état de l'art

D'après l'analyse de la chaîne de simulation actuelle, le principal besoin concerne le paramétrage et l'exécution distante des tâches sur la grille de calcul. La simulation CONSERT fonctionne actuellement sur la grille CIMENT avec l'intergiciel CiGri et en reste très dépendante. Afin de l'automatiser et de la rendre plus facilement adaptable à d'autres infrastructures, il est au préalable intéressant d'étudier quelques solutions d'intergiciels proposées par les grilles dont CIMENT qui a été présentée plus haut.

3.1 Quelques grilles de calcul

3.1.1 Grid'5000

Le projet français Grid'5000 est le fruit d'une collaboration nationale entre l'INRIA, le CNRS, des laboratoires, des universités et certains conseils régionaux. Depuis son origine en 2003, il vise à fournir à ses utilisateurs une plateforme d'expérimentation scientifique pour l'étude des systèmes parallèles et distribués à grande échelle. [35]

Grid'5000 utilise l'intergiciel OAR mais exploite également la possibilité d'exécuter des travaux en « best-effort ». Il rassemble près de 6000 cœurs de processeurs répartis sur neuf sites en France eux-mêmes connectés à RENATER avec un lien dédié de 10Gb/s (Lille, Rennes, Orsay, Nancy, Bordeaux, Lyon, Grenoble, Toulouse, Sophia-Antipolis), un site au Brésil (Porto Alegre) et un site au Luxembourg.

3.1.2 EGI

Initiée en 2004 à partir du projet LHC Computing Grid¹, la grille EGI (anciennement EGEE) représente actuellement plus de 100 000 processeurs répartis sur plus de 250 sites. EGI est l'aboutissement des plusieurs projets financés par la commission européenne auxquels ont participé plus de 70 institutions issues de 27 pays européens. [36] [37]

Les principaux objectifs de ce projet sont :

1. Le LCG est un réseau distribué conçu au CERN pour traiter les données massives produites par le LHC.

- construire une grille sûre, fiable et robuste ;
- développer une solution simple d'un intergiciel destiné à de nombreuses applications scientifiques ;
- attirer, engager et soutenir une grande variété d'utilisateurs, scientifiques ou industriels, et leur fournir un soutien technique et de formation.

3.1.3 Globus

Fondé en 1995, le projet Globus – devenu Globus Alliance en 2003 – est une association internationale dédiée au développement des technologies à la construction d'infrastructures de grilles de calcul. Il rassemble plusieurs laboratoires et universités américaines ainsi que l'Université d'Edimbourg et le Centre suédois pour le calcul parallèle.

3.1.4 NorduGrid

Initié en 2001, NorduGrid désignait au départ un projet devant mettre en place un prototype d'une infrastructure de calcul distribué pour répondre aux besoins des chercheurs en physique des hautes énergies de l'expérience ATLAS. Dès 2003, l'intergiciel « Advanced Resource Connector » (ARC) est proposé. NorduGrid devient un organisme regroupant des universités et laboratoires de Norvège, du Danemark, de Suède et de Finlande ayant pour but de développer et promouvoir ARC.

3.2 Quelques intergiciels de grilles de calcul

3.2.1 OAR

Créé à l'origine pour le projet CIMENT, OAR est le système d'ordonnancement de tâches qui a été sélectionné pour gérer l'ensemble des ressources des grappes Grid5000². L'équipe de développement est très active et améliore régulièrement OAR tout en lui apportant de nouvelles fonctionnalités. Récemment, il a bénéficié d'une visibilité accrue grâce au Google Summer of Code³ en 2008, 2009 et 2010. [38]

Comme représenté sur la figure 21, l'intergiciel OAR est conçu autour d'une base de données MySQL, d'un noyau applicatif écrit en Perl et d'un outil optionnel de déploiement. Il est composé de modules qui interagissent uniquement avec la base de données et qui sont exécutés comme des programmes indépendants. Formellement, il n'y a pas d'API, le système est complètement défini par le schéma de la base de données. Cette approche facilite le développement de modules spécifiques car chaque module, tel qu'un planificateur, peut être développé dans n'importe quel langage disposant d'une librairie d'accès à la base de données.

Ses principales fonctionnalités sont :

- deux modes de soumission : par script et en mode interactif ;
- des files d'attente multiples avec des priorités différentes pour chaque file ;
- des règles d'admission : grande souplesse de configuration de l'admission des jobs dans les files d'attente ;

2. Grid'5000 est une grille de calcul française d'environ 6000 cœurs dédiée à l'expérimentation scientifique de systèmes parallèles et distribués à grande échelle. Elle utilise l'intergiciel OAR mais les travaux en « best-effort » sont possibles.

3. Programme annuel organisé par Google visant à promouvoir le développement du logiciel libre

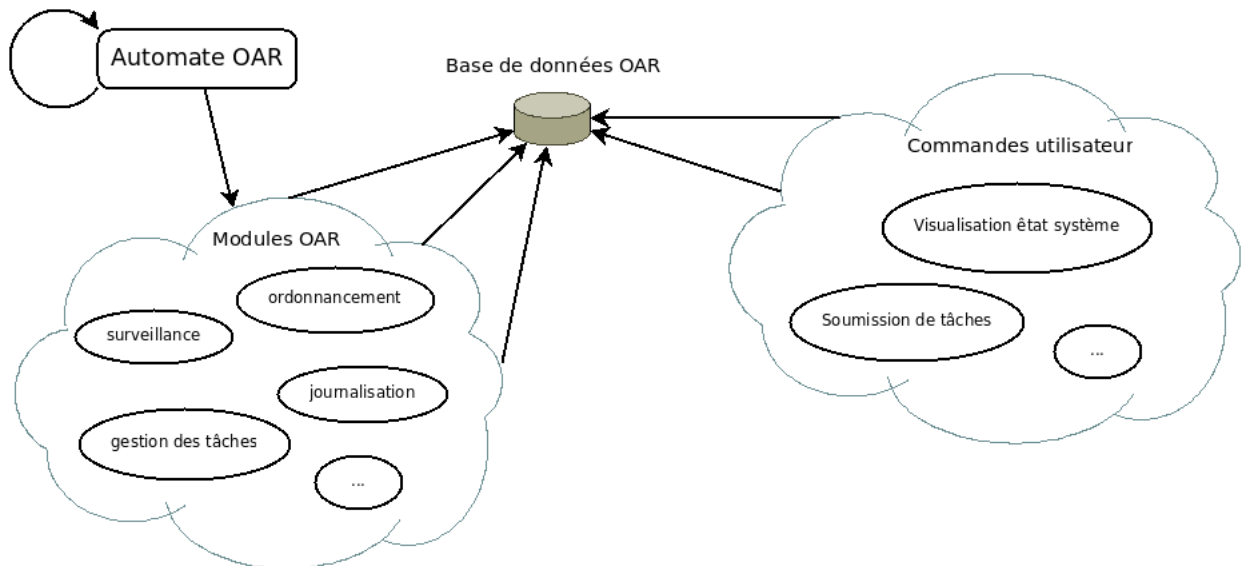


FIGURE 21 – Architecture de l'intergiciel OAR. Source : [39]

- une soumission par réservation : un job peut être lancé à une date définie à l'avance (advance reservation) ;
- une dépendances inter-jobs : un job ne démarre que si un ensemble défini de jobs a pu se terminer avec succès ;
- des tableaux de jobs : possibilité de soumettre en une seule fois de nombreux jobs aux propriétés similaires dépendants éventuellement de paramètres ;
- un type de job spécifique « best-effort » qui permet d'exploiter les ressources inutilisées ;
- un partage intelligent des ressources en fonction du taux d'utilisation par utilisateur et par projet.

3.2.1.1 Lancement de tâche

Pour soumettre un job interactif, on utilise la commande `oarsub` avec le paramètre `-I`. OAR retourne alors un identifiant unique permettant de repérer le job dans le système.

```
submachine:~$> oarsub -I
OAR_JOB_ID=1234
```

Une fois que le job est planifié et lorsque les ressources souhaitées sont enfin disponibles, OAR se connecte au premier nœud alloué. OAR initialise des variables d'environnement qui informent l'utilisateur des propriétés de sa soumission. Par exemple, la liste des nœud alloués est contenue dans la variable `$OAR_NODEFILE`.

```
node:~$> env | grep OAR
node:~$> cat $OAR_NODEFILE
```

3.2.1.2 Contrôle de l'exécution

L'état d'un job peut à tout moment être contrôlé en appelant la commande suivante :

```
submachine:~$> oarstat -fj OAR_JOB_ID
```

Il est également possible d'utiliser les outils graphiques, si disponibles, tels que Monika ou DrawGantt.

3.2.1.3 Arrêt de tâche

Afin de terminer un job interactif, il est nécessaire de se déconnecter de la ressource. Cela se fait grâce à la commande suivante :

```
node:~$> exit
```

Il est également possible d'utiliser la syntaxe suivante :

```
submachine:~$> oardel JOB_ID
```

3.2.2 CiGri

CiGri est un outil complémentaire à OAR développé par les mêmes auteurs sous la forme d'un logiciel libre. Il permet de constituer une grille de calcul dite « légère » en exploitant les cycles CPU libres des grappes de calcul. Cette architecture s'appuie sur le type de jobs « best-effort » que OAR propose. Avec CiGri, on peut optimiser la charge d'un ensemble de grappes et offrir de nombreuses ressources aux utilisateurs ayant des applications de type multi-paramétriques ou dites « sac de tâches ». [39]

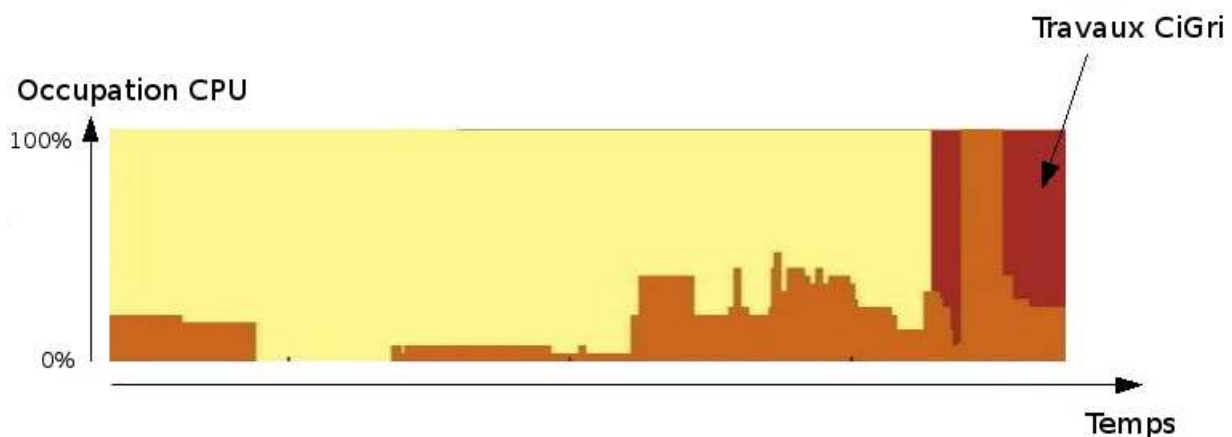


FIGURE 22 – Évolution de la charge d'une grappe. Source : [40]

CiGri est largement utilisé dans le projet CIMENT et a exécuté plus de 3 millions de jobs depuis 2002 sur ce site. La charge des grappes de calcul est souvent faite de pics et de creux comme le montre la figure 22. Lorsque CiGri entre en jeu, on constate que la charge est lissée.

L'emploi de cette grille est quasi transparente vis-à-vis des utilisateurs locaux d'un calculateur donné puisque les jobs best-effort sont d'une priorité nulle et immédiatement tués lorsque des jobs

locaux ont besoin des ressources. CiGri se charge de re-soumettre automatiquement les jobs qui ont été tués. Un système de collecte automatique des résultats est par ailleurs disponible.

Même si CiGri est un sous-projet de OAR, ces deux outils évoluent en parallèle. Certaines fonctionnalités de OAR sont parfois issues d'un besoin de CiGri et le développement d'un outil influe alors sur l'autre. Ainsi lors du Google Summer Of Code de 2009, CiGri a bénéficié d'un stage au sein du projet OAR qui a permis de réécrire entièrement le mécanisme de planification. Prochainement, CiGri tirera parti de la nouvelle API REST disponible dans les dernières versions de OAR, sécurisant, simplifiant et fiabilisant la couche de communication entre CiGri et OAR.

3.2.2.1 Lancement de campagne

Afin de lancer une campagne, il faut au préalable créer un fichier au format JDL pour décrire les différentes options.

La soumission de la campagne est ensuite effectuée au moyen de la commande suivante :

```
gridsub -t TYPE FICHIER.JDL
```

Le paramètre de type détermine le mode de soumission de la campagne :

- TEST pour ne lancer qu'une tâche par grappe configurée et en haute priorité pour tester la campagne ;
- BESTEFFORT pour lancer la campagne dans le mode par défaut de la grille.

3.2.2.2 Contrôle de l'exécution

L'exécution d'une campagne peut être suivie en ligne de commande par l'outil `gridstat`. Sans paramètre, il affiche une liste de campagnes actives sur la grille en précisant pour chacune :

- l'identifiant ;
- l'auteur ;
- le statut ;
- le nombre total de tâches ;
- le nombre de tâches en cours ;
- le nombre de tâches terminées ;
- le nombre de tâches erronées.

Le statut peut prendre les valeurs suivantes :

- WAITING lorsque la campagne est planifiée mais pas encore lancée ;
- IN_TREATMENT lorsque la campagne est en cours d'exécution ;
- TERMINATED lorsque la campagne s'est terminée avec succès ;
- FAILED lorsque l'exécution de la campagne a généré au moins une erreur.

En précisant l'identifiant de campagne en premier paramètre, l'outil `gridstat` affiche des informations plus détaillées sur ladite campagne. Cela donne donc un appel de la forme suivante :

```
gridstat IDCAMPAGNE
```

Voici un exemple d'exécution de cette commande :

```

calculer:~> gridstat
Id      User      Status      Type      Waiting  Running  Finished  ToFix  Err
-----
4  2716    bollardp  IN_TREATMENT default    0        10        0
   2717    bollardp  IN_TREATMENT default    0        10        0
-----

calculer:~> gridstat 2716
9 Multiple job 2716
   Type:                default
   Status:              IN_TREATMENT
   Submitted:          Mon Nov 22 21:41:21 +0100 2010
14  Last terminated job: Thu Jan 01 01:00:00 +0100 1970
   Errors to Fix:      false
   Running:           10
   Waiting:           0
   Terminated:       0
19  Duration:           2395660 s

Forecasts:
   Throughput (last hour): 0.00 jobs/hour
   Average:               0 s
24  Stddev:              0.00

calculer:~>

```

Dans les informations affichées, on retrouve :

- le type (« default » ou « test ») ;
- le statut ;
- la date et l'heure de soumission ;
- la date et l'heure à laquelle la dernière tâche s'est terminée ;
- un indicateur de présence d'erreurs à corriger ;
- le nombre de tâches en cours d'exécution ;
- le nombre de tâches en attente ;
- le nombre de tâches terminées ;
- la durée d'exécution en secondes ;
- quelques indicateurs statistiques tels que le nombre de tâches traitées par heure.

3.2.2.3 Arrêt de campagne

Lorsqu'une campagne a été planifiée sur la grille, il est possible de la supprimer en appelant le programme `griddel` en ligne de commande avec l'identifiant de la campagne comme premier paramètre. Cela donne donc une saisie de la forme suivante :

```
griddel -m IDCAMPAGNE
```

3.2.2.4 Interface web

CiGri propose également une interface Web. Celle-ci est accessible à l'adresse suivante : <https://ciment-grid.ujf-grenoble.fr/cigri-web>

Multijob #	Multijob name	Submission date	State	
2457	4bd8353b00c09	2010-08-03 17:22:29	TERMINATED	Statistics
2456	4bd8353b00c09	2010-08-03 16:35:20	TERMINATED	Statistics
2455	4bd8353b00c09	2010-08-03 16:29:41	TERMINATED	Statistics
2454	4bd8353b00c09	2010-08-03 16:23:41	TERMINATED	Statistics
2453	4bd8353b00c09	2010-08-03 16:05:48	TERMINATED	Statistics

FIGURE 23 – Liste des campagnes d’un utilisateur de CiGri.

Cluster Name	Execution Command	Exec Directory	Wall Time	Weight	Prio
fostino.obs.ujf-grenoble.fr	/home/ciment/bollardp /4bcf1bb4c0683/4c58341fd0209 /run_job_cigrn.sh	/home/ciment/bollardp /4bcf1bb4c0683 /4c58341fd0209	10:00:00	0	1
nanostar.ujf-grenoble.fr	/home/ciment/bollardp /4bcf1bb4c0683/4c58341fd0209 /run_job_cigrn.sh	/home/ciment/bollardp /4bcf1bb4c0683 /4c58341fd0209	10:00:00	0	1
r2d2.obs.ujf-grenoble.fr	/home/ciment/bollardp /4bcf1bb4c0683/4c58341fd0209 /run_job_cigrn.sh	/home/ciment/bollardp /4bcf1bb4c0683 /4c58341fd0209	10:00:00	0	1

FIGURE 24 – Informations sur une campagne, paramètres d’exécution.

The screenshot shows the 'My Account Jobs' page for Multijob #2457. The page has a navigation bar with 'General information', 'Statistics', 'Events', and 'My account'. Below the navigation, there are links for 'Multijobs', 'Statistics', 'Status', 'Errors', and 'bollardp: logout'. The breadcrumb trail indicates the user is in 'My account > Multijobs > Multijob #2457 > Executed jobs'. The main heading is 'Multijob #2457' with sub-sections for 'Running Jobs', 'Executed Jobs', and 'Waiting Parameters'. Below this, it says 'Executed Jobs 1 - 3 out of 3'. There are pagination controls showing 'Page 1 / 1 with 20 items per page'. The main content is a table with the following data:

Job #	Job name	Start date	End date	Duration	Cluster	Node	Collect #
3615893	output2_6	2010-08-04 02:15:10	2010-08-04 02:15:10	00:00:00	nanostar.ufj-grenoble.fr	r1i1n14	2
3615841	output2_4	2010-08-03 17:23:07	2010-08-03 17:23:08	00:00:01	r2d2.obs.ufj-grenoble.fr	n43	1
3615840	output2_2	2010-08-03 17:22:46	2010-08-03 17:22:46	00:00:00	nanostar.ufj-grenoble.fr	r1i1n0	1

FIGURE 25 – Informations sur une campagne, liste des tâches.

3.2.3 gLite

gLite est un intergiciel pour les grilles de calcul proposé dans le cadre du projet EGEE. Il fournit un cadriciel pour construire des applications de grille tirant partie de la puissance de calcul distribué et des ressources de stockage via l'Internet. [41]

La distribution gLite est un ensemble homogène de composants conçus pour partager leurs ressources. En d'autres termes, cet intergiciel permet de construire une grille. En plus du code développé dans le cadre du projet, la distribution gLite rassemble les contributions de bien d'autres projets y compris LCG. Le modèle de distribution est de construire différents services (« nœud-type ») à partir de ces composants et d'assurer une installation facile et une configuration sur les plateformes choisies (actuellement Scientific Linux 4 et 5 ainsi que Debian GNU/Linux 4 pour les WN).

3.2.4 Globus Toolkit

Le Globus Toolkit est l'intergiciel développé pour la grille Globus. Comme le montre la figure 26, cet intergiciel est en réalité composé de différents logiciels et services. Cela inclut les bibliothèques de sécurité distribuée, la gestion des ressources, la surveillance et la gestion des données. Sa dernière version, GT4, comprend des composants pour systèmes de bâtiments qui suivent l'Open Grid Services Architecture (OGSA) cadre défini par le Global Grid Forum (GGF), dont l'Alliance Globus est un membre éminent. GT4 composants et outils de développement logiciel sont également conformes aux Web Services Framework de ressources (WSRF), un ensemble de normes en matière de développement dans OASIS. [42]

3.2.5 ARC

L'intergiciel ARC⁴, issu du projet NorduGrid, est une solution logicielle qui utilise les technologies de grille pour permettre le partage et la fédération de ressources de calcul et de stockage

4. Acronyme de « Advanced Resource Connector » qui signifie « Connecteur de ressource avancé ».

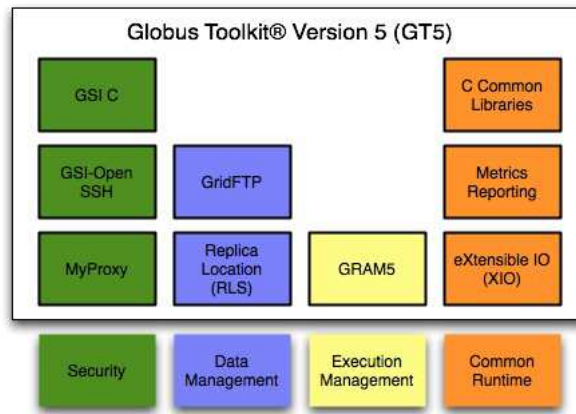


FIGURE 26 – Architecture de Globus Toolkit. Source : [42]

distribué. ARC est utilisé pour créer des infrastructures de grille d’envergure et de complexité différentes (de la grille universitaire à la grille nationale). Il est distribué sous la licence Apache 2.0 [43].

ARC est :

- une solution intergicIELle portable, légère, open source et générique ;
- une implémentation des services fondamentaux d’une grille extensible et de qualité de production ;
- un facilitateur de solutions d’infrastructure de calcul distribué inter-organisation ;
- fortement attaché aux standards ouverts et à l’interopérabilité ;
- en utilisation depuis 2002.

En ce qui concerne son architecture, comme on peut le voir sur la figure 27, toutes les communications entre les parties client et serveur se font via des services web. ARC offre une interface en ligne de commande et une interface graphique. Python et Java peuvent notamment être utilisés pour développer des clients à partir des bibliothèques mises à disposition.

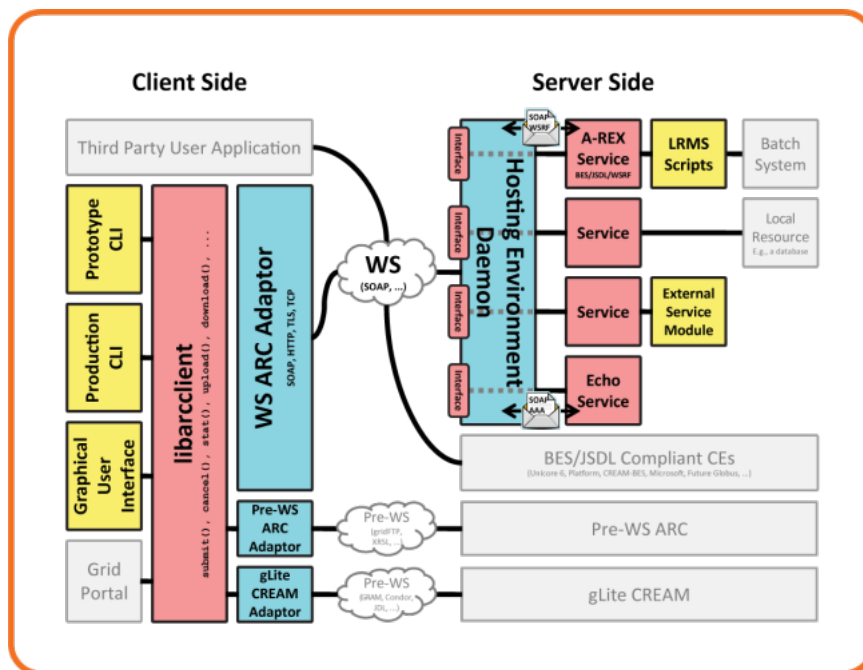


FIGURE 27 – Architecture de l’intergiciel ARC. Source : [43]

3.3 Quelques outils complémentaires de gestion de grilles

3.3.1 OARgrid

L'outil OARgrid n'est actuellement utilisé qu'avec la grille Grid'5000 pour laquelle il a été spécifiquement créé. Il permet de gérer de manière centralisée les ressources de plusieurs serveurs OAR interconnectés entre eux via une interface en ligne de commande ou une API REST. OARgrid n'est pas un intergiciel de grilles ; il ne propose aucune fonctionnalité d'ordonnancement. Il offre seulement une vision globale d'un ensemble de grappes et permet de faire des réservations de ressources sur cet ensemble. Il s'agit donc d'un outil simple pour soumettre des travaux simultanément sur plusieurs grappes en exploitant le mode « advanced reservation » de OAR.

3.3.2 GRUDU

GRUDU est un outil de gestion de Grid'5000. Il permet de gérer et réserver des ressources mais également de déployer des environnements de calcul personnalisés. Développé comme un complément à DIET, il peut être utilisé soit via l'espace de travail DIET, soit en version autonome comme sur la figure 28.

Le projet DIET se focalise sur le développement d'un intergiciel extensible en se fondant sur une distribution du problème d'ordonnancement sur de multiples agents. DIET consiste en un ensemble d'éléments pouvant être utilisés conjointement pour construire des applications employant le paradigme Grid-RPC. Cet intergiciel est capable de trouver le serveur approprié à partir des informations issues de la requête du client – par exemple, le problème à résoudre, la taille des données impliquées –, de la performance de la plateforme cible – par exemple, la charge du serveur, la mémoire disponible, les performances de communication – et la disponibilité locale des données stockées lors des précédents calculs. L'ordonnanceur est distribué en utilisant plusieurs hiérarchies collaboratrices connectées de manière statique ou dynamique (de style peer-to-peer). La gestion des données est assurée pour permettre aux données persistantes de rester sur le système pour une utilisation future. Cette fonctionnalité évite les communications superflues lorsque les dépendances existent entre les différentes requêtes. [44]

3.3.3 g-Eclipse

Le projet g-Eclipse est dirigé par un consortium constitué de sept partenaires de cinq pays européens différents. Il vise à construire un cadre d'environnement de travail intégré pour accéder aux infrastructures de grilles existantes. Comme le montre la figure 29, cet environnement offre à l'utilisateur une interface standardisée qui rassemble divers outils permettant de gérer les ressources de la grille tout en assurant le cycle de développement et la personnalisation des applications devant fonctionner sur la grille. [45]

L'environnement g-Eclipse est basé sur l'écosystème éprouvé de la communauté Eclipse, ce qui lui assure un développement durable sur le long terme. Il interface principalement les infrastructures de grilles telles EGEE. Par conséquent, il cible en premier lieu les outils génériques de gestion de grilles tels que gLite ou Globus Toolkit. Il peut cependant être étendu à d'autres intergiciels.

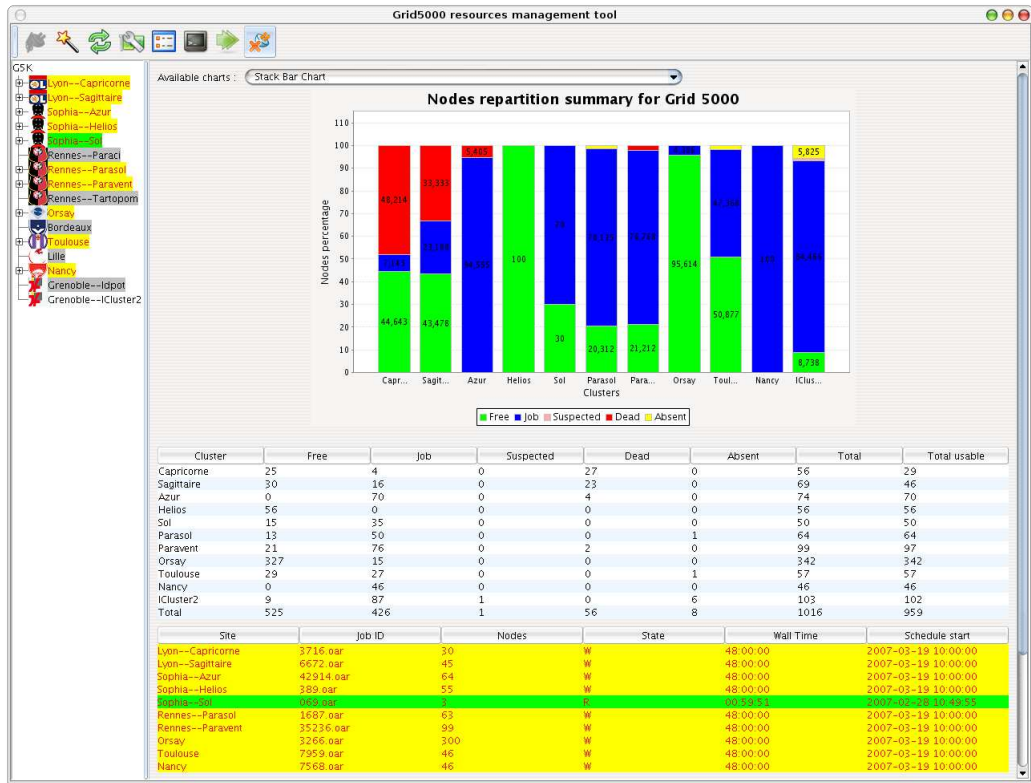


FIGURE 28 – Utilisation de l’outil GRUDU. Source : [44]

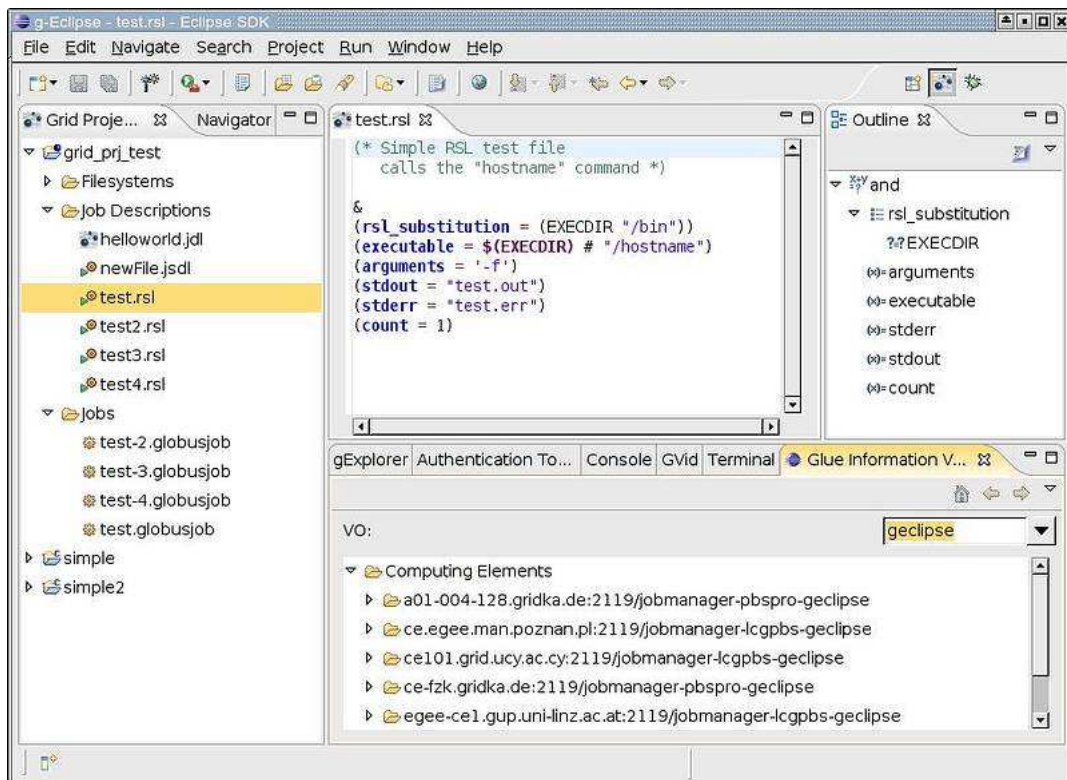


FIGURE 29 – Utilisation de l’environnement g-Eclipse. Source : [45]

3.4 Bilan

On s'aperçoit assez vite que globalement à chaque grille correspond une solution intergicielle qui lui est propre. Il n'existe pour le moment pas de standard reconnu comme tel et supporté par toutes les infrastructures. OAR – et potentiellement CiGri – semble devenir l'outil de référence pour la communauté française puisqu'il est employé par le projet Grid'5000. Poussé par la communauté européenne via le projet EGI, gLite est quant à lui un intergiciel reconnu. Enfin, le Globus Toolkit, d'origine américaine, est amené à s'implanter en Europe... Même s'il est difficile de voir émerger une solution unique, on peut espérer que les diverses communautés développeront des solutions intermédiaires pour favoriser la collaboration entre les infrastructures.

Le rôle d'un intergiciel est avant tout de gérer les ressources d'une grille et d'ordonner les travaux soumis par les utilisateurs. L'architecture d'une telle solution est donc conçue autour de l'infrastructure de la grille et non autour des travaux. Il est donc normal que l'interface ou l'API proposées aux utilisateurs ne permettent pas de gérer les problématiques de paramétrages telles que nous le rencontrons dans la chaîne de simulation CONSERT. L'outil qui sera développé au cours de ce stage devra donc se focaliser sur la chaîne de simulation plutôt que sur l'intergiciel.

De ce fait, il est envisageable de réutiliser un des outils de gestion de grilles. Cependant, ceux-ci sont trop liés à une architecture et seul g-Eclipse semble suffisamment générique pour permettre une adaptation à CiGri. De plus, ils ne sont en réalité que des compléments graphiques aux couches d'intergiciels et se révèlent donc inadaptés au présent projet.

Réalisation de la nouvelle plateforme de simulation

4.1 Méthodologie

4.1.1 Déroulement du projet

Comme le montre le diagramme de Gantt situé en annexe, le projet a suivi un cycle itératif. Chaque itération a duré entre trois et quatre semaines,

Une itération débute par une réunion de cadrage afin de valider certains choix, de définir les nouveaux objectifs et ainsi d'établir la liste des tâches à effectuer. La première semaine est consacrée à la phase d'analyse et de conception. À partir de la deuxième itération, tout travail précédent est systématiquement revu et amélioré lors de cette étape. La phase de développement de l'application s'étire ensuite sur les semaines suivantes.

A l'issue de l'itération, une réunion d'équipe est organisée pour valider le travail effectué. De ce fait, cette réunion de fin d'itération est généralement confondue avec celle du début de l'itération suivante.

Tout au long du développement, le même jeu de tests est utilisé. Celui-ci se compose des scripts de la plateforme d'origine ainsi que des exemples de modèles de formes et de modèles de structures. Les différentes variations de jeu de tests sont effectuées sur les paramètres numériques des scripts. Dès qu'un prototype est « prêt », il est présenté à l'équipe et laissé en test si souhaité.

4.1.2 Outils et techniques utilisés

Un dépôt Subversion a été ouvert pour le projet sur le serveur du LPG. Il a ainsi été utilisé pour centraliser le code source de l'application et ainsi gérer ses évolutions.

Dans la mesure du possible, les logiciels et technologies libres ont été privilégiés. Ainsi, la plateforme Eclipse a été retenue comme environnement de développement. Le langage PHP est supporté via le greffon PDT. De plus, la modélisation UML est intégrée par le greffon TopCased. La modélisation de la base de données a été réalisée via l'outil MySQL Workbench. La planification a été réalisée à l'aide de l'outil GanttProject.

4.2 Conception de la plateforme

Afin de poser les bases de l'application, il est nécessaire d'effectuer une phase de conception. À partir des spécifications générales, elle permet d'obtenir les spécifications détaillées relatives à l'itération courante. Seule l'architecture finale et aboutie sera présentée dans la suite de ce document.

4.2.1 Conception de l'architecture globale

Afin de répondre aux différents besoins d'amélioration de la plateforme existante, l'architecture présentée sur la figure 30 a été adoptée. Elle se décompose en quatre blocs distincts :

- le poste client ;
- le serveur web ;
- le serveur de stockage ;
- la grille de calcul.

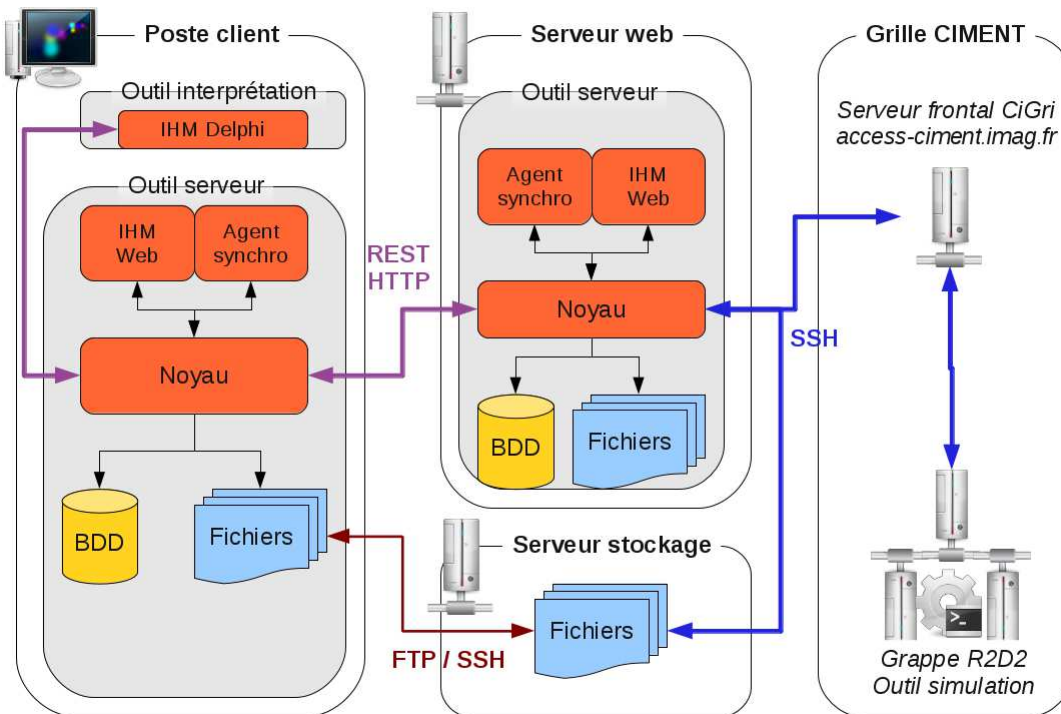


FIGURE 30 – Architecture globale de la nouvelle plateforme.

4.2.1.1 Poste client

Le poste client est toujours requis pour exécuter l'outil de visualisation. Désormais, il offre à l'utilisateur la possibilité de gérer les simulations en accédant à l'interface de gestion située sur le serveur web. Dans une version plus évoluée, l'outil de visualisation peut embarquer une interface native.

Si le poste client est mobile, il peut aussi héberger une version autonome de l'application serveur. Celle-ci embarque toute la logique applicative pour gérer la base de données et les fichiers

mais elle n'est pas en relation directe avec les hôtes. Au contraire, elle se synchronise avec l'application serveur pour partager les modifications locales ainsi que pour actualiser la base de données et récupérer les nouveaux fichiers.

4.2.1.2 Serveur web

Cette machine est la pierre angulaire de la plateforme. Il permet d'exécuter l'application serveur, principale brique de la plateforme. Il accueille également quelques fichiers et la base de données. L'application serveur est la partie centrale de la plateforme. Elle contient toute la logique applicative, elle gère la base de données et les fichiers, elle est en relation directe avec les hôtes tels que la grille de calcul.

La base de données contient toutes les informations paramétrées via l'application serveur. Actuellement, aucun fichier n'est stocké en base afin de ne pas la surcharger.

4.2.1.3 Serveur de stockage

Le serveur de stockage est actuellement une machine hébergée au LPG. Il est utilisé par la plateforme pour accueillir les différents fichiers produits. Les échanges se font via des connexions SSH.

4.2.1.4 Grille de calcul

La grille de calcul est accessible par son serveur frontal via une connexion SSH. Chaque grappe communique avec ce serveur frontal en SSH et exécute le programme de simulation.

4.2.2 Modélisation de l'architecture de l'application

La figure 31 présente un diagramme partiel de classes de l'application serveur. Chaque classe est décrite dans ce qui suit. Par souci de lisibilité, les accesseurs sont été omis et les seules méthodes « métier » sont listées. Ainsi, pour chaque attribut, deux méthodes publiques de la forme `getAttribut()` et `setAttribut(valeur)` permettent respectivement d'obtenir et de définir la valeur de l'attribut.

L'usage de patrons de conception – plus connus sous le terme « design patterns » – a été favorisé. Par exemple, le patron « composite », qui est visible sur ce diagramme partiel de classes, se retrouve au niveau des groupes de tâches et de la spécialisation des hôtes. Le « patron de méthode » a été utilisé sur les tâches pour déléguer aux sous-classes spécialisées l'implémentation du comportement des méthodes `start()`, `stop()` et `checkState()`. Par ailleurs, le patron « prototype » a été appliqué à presque toutes les classes principales ; cela rend par exemple possible la duplication d'une séquence avec toutes ses tâches et ses variables. Enfin, le patron « fabrique abstraite » – qui par souci de lisibilité, n'est pas visible sur ce schéma – a été utilisé sur toutes les classes principales pour isoler les classes « métier » de leur initialisation via le modèle de données du cadriciel.

4.2.2.1 Tâche

Une tâche est une entité type définissant une action particulière d'une chaîne de traitements. Elle est caractérisée par :

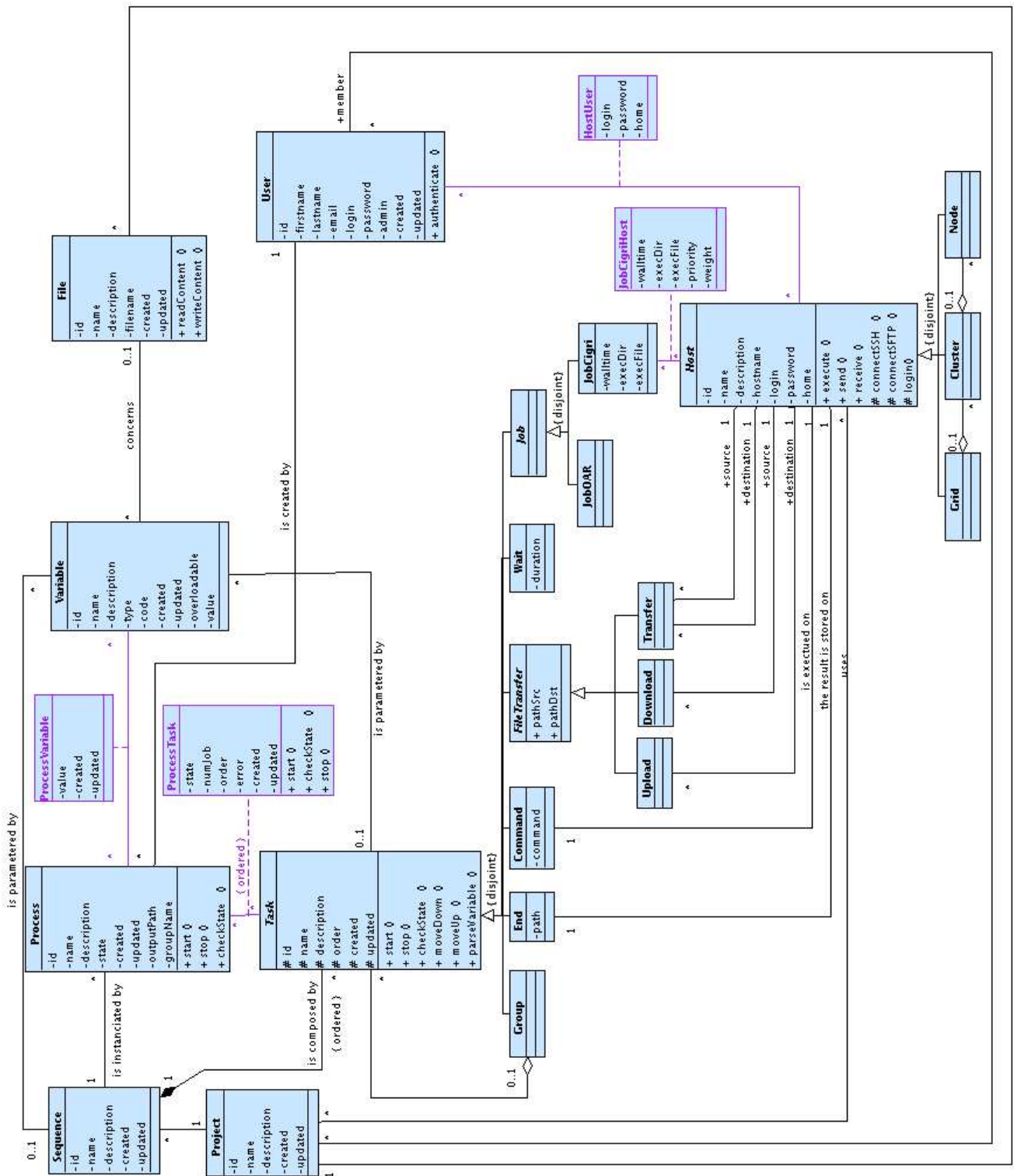


FIGURE 31 – Diagramme partiel de classes.

- un identifiant ;
- un nom ;
- une description ;
- une date de création ;
- une référence vers la séquence ;
- un numéro d'ordre.

Lorsqu'une tâche est paramétrée au sein d'un processus, elle peut être exécutée. Elle suit alors un cycle composé de plusieurs états comme représenté sur la figure 32. Dès la création de l'instance, la tâche est *initialisée*. Lorsque l'exécution du processus est planifiée, c'est-à-dire mise en attente du prochain déclenchement automatique, chaque tâche passe à l'état *en attente*. Une tâche ne passe à l'état *en cours* que lorsqu'elle est lancée soit manuellement par l'utilisateur, soit automatiquement par la plateforme. Si l'exécution s'est correctement déroulée, la tâche passe dans l'état final *terminée*, ou sinon *échouée*. Une tâche en attente ou en cours d'exécution peut être arrêtée ; elle passera alors à l'état *annulée*.

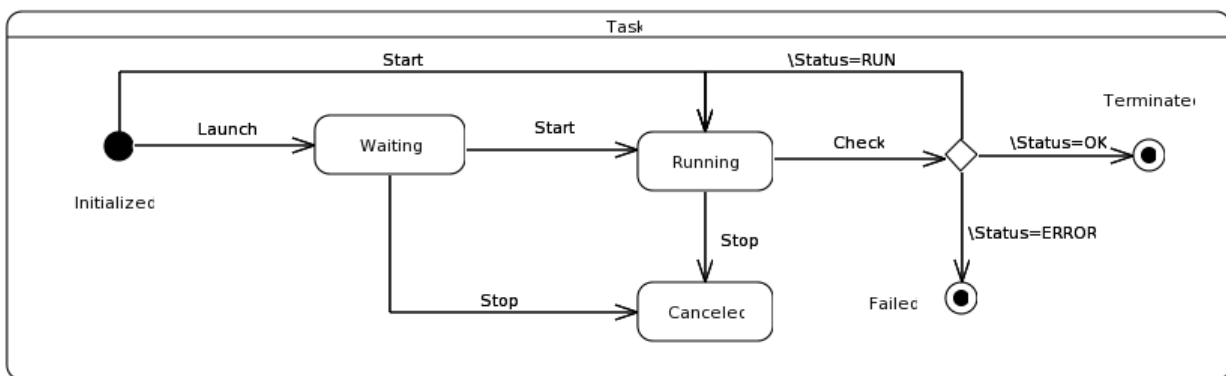


FIGURE 32 – Diagramme d'états d'une tâche.

Plusieurs types de tâches sont prévus : *commande*, *transfert*, *envoi*, *réception*, *job CiGri*, *attente*, *fin*. Il est cependant possible d'étendre les possibilités de la plateforme en créant de nouvelles tâches en fonction des besoins rencontrés.

Chaque sous-classe définissant un type particulier de tâche implémente son comportement spécifique au sein des méthodes suivantes :

- `start()` pour lancer l'exécution de la tâche ;
- `stop()` pour arrêter l'exécution de la tâche ;
- `checkState()` pour actualiser l'état de la tâche en fonction de l'avancement de son exécution.

4.2.2.1.1 Commande

Une tâche de type *commande* permet l'exécution distante d'une commande paramétrée sur un hôte donné. Elle est caractérisée par :

- un hôte ;
- une chaîne de caractères qui sera interprétée comme une « commande » sur l'hôte.

4.2.2.1.2 Transfert

Une tâche de type *transfert* permet de transférer en FTP un fichier donné entre un hôte source et un hôte cible. Elle est caractérisée par :

- un chemin source ;
- un hôte source ;
- un chemin cible ;
- un hôte cible.

4.2.2.1.3 Envoi

Une tâche de type `envoi` permet d'envoyer en FTP un fichier donné depuis le serveur de la plateforme vers un hôte cible. Elle est caractérisée par :

- un chemin source ;
- un chemin cible ;
- un hôte cible.

4.2.2.1.4 Réception

Une tâche de type `réception` permet de recevoir par FTP sur le serveur de la plateforme un fichier donné stocké sur un hôte source. Elle est caractérisée par :

- un chemin source ;
- un hôte source ;
- un chemin cible.

4.2.2.1.5 Job CiGri

Une tâche de type `job CiGri` permet l'exécution d'une campagne de calculs en mode grille en utilisant CiGri. Elle est caractérisée par :

- un hôte ;
- un chemin pour le répertoire d'exécution ;
- un chemin pour le script à exécuter ;
- un temps maximal d'exécution.

4.2.2.1.6 Attente

Une tâche de type `attente` permet de marquer une pause d'une durée donnée avant l'enchaînement de la tâche suivante. Elle est caractérisée par une durée exprimée en secondes.

4.2.2.1.7 Fin

Une tâche de type `fin` permet de marquer de terminer la séquence de tâches en spécifiant, pour le processus, le chemin et l'hôte de stockage du fichier de résultats.

4.2.2.2 Séquence

- Une séquence est une liste ordonnée de tâches. Elle est caractérisée par :
- un identifiant ;
 - un nom ;

- une description ;
- une date de création ;
- une référence vers le projet.

4.2.2.3 Processus

Un processus est une instance paramétrée d'une séquence de tâches. Il est caractérisé par :

- un identifiant ;
- un nom ;
- une description ;
- une date de création ;
- un état ;
- une référence vers la séquence ;
- un nom de groupe ;
- un chemin et une référence vers l'hôte stockant le fichier final.

Lorsqu'un processus est paramétré au sein d'un processus, il peut être exécuté. Il suit alors un cycle composé de plusieurs états comme représenté sur la figure 33. Dès la création de l'instance, le processus est *initialisé*. Lorsque l'exécution est planifiée, c'est-à-dire mise en attente du prochain déclenchement automatique, le processus et toutes ses tâches passent à l'état *en attente*. Un processus ne passe à l'état *en cours* que lorsqu'il est lancé soit manuellement par l'utilisateur, soit automatiquement par la plateforme. Si l'exécution de toutes les tâches s'est correctement déroulée, le processus passe dans l'état final *terminé*, ou sinon *échoué*. Un processus en attente ou en cours d'exécution peut être arrêté ; il passera alors à l'état *annulé*.

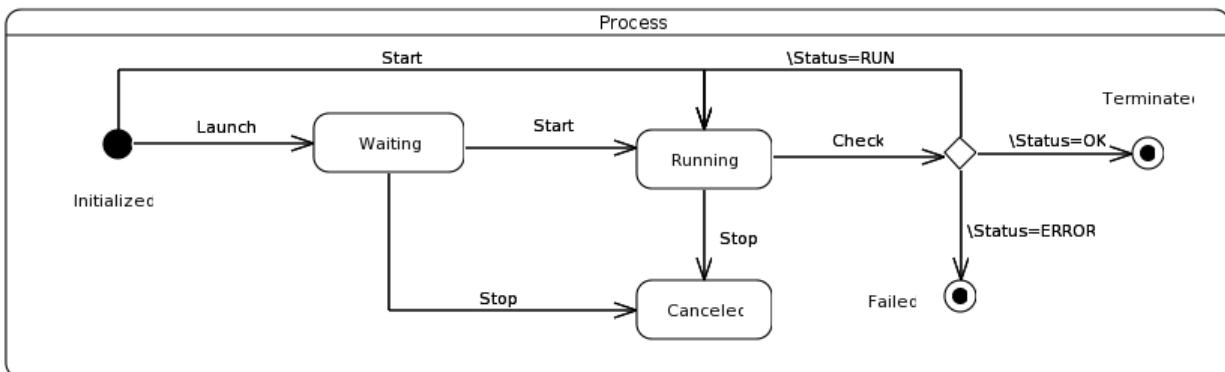


FIGURE 33 – Diagramme d'états d'un processus.

4.2.2.4 Variable

Une variable permet de paramétrer une tâche en remplaçant un code par une valeur. Elle peut être définie surchargée par l'utilisateur lors de l'instanciation d'une tâche dans un processus. Elle est caractérisée par :

- un identifiant ;
- un nom ;
- une description ;
- une date de création ;
- un type (chaîne, entier, flottant, fichier, ...);
- un code ;
- une valeur ;

- un drapeau indiquant si la variable peut être surchargée ou non.

4.2.2.5 Hôte

Un hôte est une entité permettant de définir toute machine à laquelle l'application accède pour y exécuter des tâches. Un hôte est caractérisé par :

- un identifiant ;
- un nom ;
- une description ;
- un type (grille, grappe ou nœud) ;
- un nom d'hôte.

Sur la figure 31, la classe « host » représente cette entité hôte. Elle est spécialisée selon le type d'hôte par les sous-classes `grid`, `cluster` ou `node`.

4.2.2.5.1 Grille

Un hôte de type `grille` définit une grille de calcul et peut contenir des grappes. On utilisera ce type d'hôte pour paramétrer le serveur frontal d'une grille de calcul.

4.2.2.5.2 Grappe

Un hôte de type `grappe` définit une grappe de calcul. Une grappe peut être rattachée à une grille et peut contenir des nœuds.

4.2.2.5.3 Nœud

Un hôte de type « nœud » définit un nœud particulier d'une grappe ou toute machine indépendante. Ainsi, on utilisera ce type d'hôte pour paramétrer un serveur de stockage.

4.2.3 Base de données

La base de données modélisée à partir du diagramme de classes est représentée sur la figure 34. Le modèle est donc relativement proche de la structure objet. Dans l'ensemble, chaque table correspond à une classe ou à une association n-aire.

Les identifiants sont générés par l'application sous la forme d'une chaîne unique de plusieurs caractères. Les champs correspondants sont de type `VARCHAR` limité à 65 caractères.

Les champs de date sont au format `DATETIME`. Les champs textes de moins de 255 caractères sont de type `TINYTEXT` ; au-delà, le format `TEXT` est préféré.

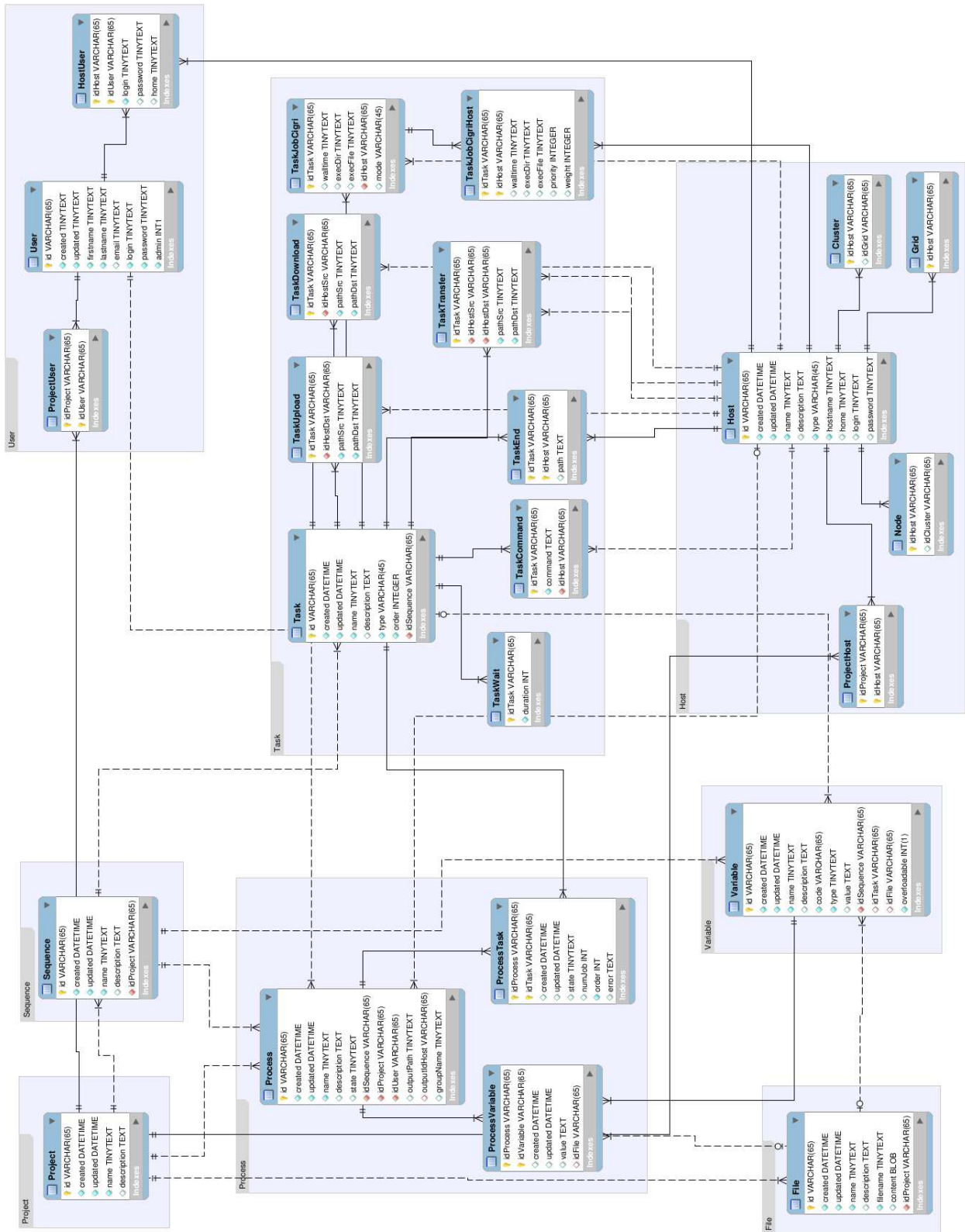


FIGURE 34 – Modélisation du schéma de la base de données.

4.3 Implémentation de l'architecture logicielle

4.3.1 Choix techniques

4.3.1.1 Langage PHP

PHP est un langage interprété principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP. Il peut également fonctionner de façon locale en exécutant les programmes en ligne de commande. PHP est un langage impératif disposant depuis la version 5 de fonctionnalités complètes d'un modèle objet. Sa syntaxe découle du langage C et est donc assez proche de Java par exemple. [46] [47]

La figure 35 représente le déroulement de l'exécution du code PHP dans une utilisation Web. Lorsqu'un visiteur demande à consulter une page Web, son navigateur envoie une requête au serveur HTTP correspondant. Si la page est identifiée comme un script PHP (généralement grâce à l'extension .php), le serveur appelle l'interprète PHP qui va traiter et générer le code final de la page (constitué généralement d'HTML ou de XHTML, mais aussi souvent de CSS et de JS). Ce contenu est renvoyé au serveur HTTP, qui l'envoie finalement au client.



FIGURE 35 – Schéma de fonctionnement d'un script PHP. Source : [46]

PHP a été retenu car il s'agit d'un langage libre et multiplateforme. Dans le cas d'une application Web, il est beaucoup plus simple à déployer que Java qui, lui, nécessite l'installation d'un serveur intermédiaire tel que Tomcat. De plus, ce langage était connu et maîtrisé avant ce stage ce qui permettait d'être plus efficace qu'en découvrant un nouveau langage comme Python ou Ruby.

4.3.1.2 Cadriciel Zend Framework

Aujourd'hui, une application Web est généralement basée sur un cadriciel. En effet, ce « squelette » permet d'accélérer le développement en fournissant une structure de base qui implémente notamment le patron MVC. De nombreuses classes apportent des solutions éprouvées et prêtes à l'emploi pour répondre aux besoins courants et récurrents d'un tel développement.

La communauté PHP étant très active, une multitude de cadriciels ont vu le jour en quelques années. Ne pouvant les comparer exhaustivement, le choix a été fortement influencé par l'avenir du projet. En effet, le cadriciel Symfony a un temps été envisagé mais finalement, Zend Framework lui a été préféré car ce-dernier est développé depuis 2006 par l'éditeur-même du langage PHP. En plus de ce gage de sérieux, Zend Framework est utilisé dans le cadre d'un autre projet au LPG. Dans la perspective de la poursuite du développement, la personne en charge aurait certainement plus de facilité à dépanner un système développé autour d'un noyau connu.

4.3.2 Initialisation du projet

4.3.2.1 Mise en place du framework

La dernière version de Zend Framework est disponible sur le site `http://framework.zend.com` en plusieurs éditions : une version minimale que nous choisirons ou une version complète qui intègre notamment des exemples supplémentaires. L'archive ainsi récupérée doit être décompressée dans le dossier qui accueillera l'instance du framework pour le projet.

Suivant le système d'exploitation utilisé sur la machine – par exemple Debian GNU/Linux –, le framework peut être disponible sous la forme d'un paquet. Il suffit alors d'installer le paquet correspondant pour disposer d'une instance du framework commune à plusieurs projets. Même si cette approche semble préférable, elle présente l'inconvénient de séparer le framework du projet. En cas de mise à jour importante et de rupture de compatibilité, certains problèmes peuvent alors apparaître au sein de l'application si celle-ci n'est pas corrigée en même temps.

Avant de démarrer le développement du projet ou l'installation de l'application définitive, il est nécessaire de contrôler la présence de plusieurs éléments sur la machine hôte. Ainsi, un serveur Apache doit être installé et configuré avec les module de réécriture d'url `mod_rewrite` ainsi que le module PHP 5. Ce dernier doit proposer les modules :

- `simplexml`;
- `pdo_mysql`.

Il faut ensuite adapter la configuration du serveur Apache et paramétrer un `virtualhost` pour le projet. Celui-ci aura la forme suivante :

```

<VirtualHost *:80>
  ServerName    consert.localhost
  SetEnv        APPLICATION_ENV "development"
  DocumentRoot /home/philippe/Workspace/CONCERT/public
  <Directory   "/home/philippe/Workspace/CONCERT/public">
    Options     Indexes MultiViews FollowSymLinks
    DirectoryIndex index.php
    AllowOverride All
    Order       allow,deny
    Allow       from All
  </Directory>
</VirtualHost>

```

4.3.2.2 Création du projet

Pour créer le squelette d'un nouveau projet, il est bien évidemment possible de tout faire manuellement. Heureusement, Zend Framework propose une interface en ligne de commande qui permet d'automatiser cette tâche rébarbative. Ainsi, en se plaçant dans le répertoire adéquat, il suffit d'appeler :

```
zf create project NOMDUPROJET
```

Cela a pour effet d'initialiser l'arborescence suivante [48] :

```

NOMDUPROJET
|-- application
|  |-- Bootstrap.php
4 |  |-- configs

```

```

| |   |-- application.ini
| |   |-- controllers
| |   |   |-- ErrorController.php
| |   |   |-- IndexController.php
9 |   |-- models
| |   |-- views
| |       |-- helpers
| |       |-- scripts
| |           |-- error
14 |           |   |-- error.phtml
| |           |-- index
| |               |-- index.phtml
|-- library
|-- public
19 |   |-- .htaccess
| |   |-- index.php
|-- tests
| |   |-- application
| |   |   |-- bootstrap.php
24 |   |-- library
| |   |   |-- bootstrap.php
| |   |-- phpunit.xml

```

Les classes du Zend Framework doivent être disponibles dans le dossier `library` du projet pour que celui-ci puisse les utiliser. Pour cela, il faut soit rajouter le chemin adéquat dans la variable d'environnement `include_path`, soit lier ou copier le dossier du cadriciel à cet endroit. Cela est possible via les commandes suivantes :

```

# Symlink:
cd library; ln -s path/to/ZendFramework/library/Zend .
4 # Copy:
cd library; cp -r path/to/ZendFramework/library/Zend .

```

4.3.2.3 Description de l'arborescence

Le cadriciel Zend Framework offre une certaine souplesse dans la structure d'un projet. Celle-ci peut être modifiée à souhait tout en respectant quelques règles. Dans le cas de notre projet, la nouvelle arborescence se découpe en plusieurs sous-dossiers :

- application;
- data;
- docs;
- library;
- public;
- scripts.

Le dossier `application` contient le cœur même de l'application. Celui-ci se décompose par les sous-dossiers :

- `config` pour les fichiers de configuration (acl, navigation...);
- `controllers` pour les contrôleurs de l'application ;
- `forms` pour les formulaires ;
- `layout` pour les squelettes principaux des pages ;
- `locales` pour les dictionnaires d'internationalisation ;
- `models` pour les classes du modèle de données ;
- `plugins` pour les greffons ;
- `views` pour les vues.

Le dossier `public` est enrichi par des sous-dossiers `css`, `img` et `js` afin de séparer respectivement les feuilles de style, les images et les composants javascript.

Le dossier `data` contient une version des scripts adaptés pour la simulation CONSERT. Cependant, tout fichier téléchargé depuis l'interface vers le serveur web sera stocké dans le sous-dossier `uploads`.

Le dossier `docs` contient toute la documentation du projet, notamment les diagrammes UML et le schéma de la base de données.

Le dossier `library` est enrichi par les composants tiers utilisés par le projet.

Enfin, le dossier `scripts` contient quelques scripts permettant d'automatiser le déploiement de l'application.

4.3.2.4 Configuration du framework

Lorsqu'un appel est effectué à l'application, celui-ci est tout d'abord intercepté par le point d'entrée – le fichier `index.php` – qui initialise le cadriciel avant de lui passer le relais. Lors de cette phase d'initialisation, on instancie un objet `Zend_Application` pour charger certains paramètres de configuration contenus dans un fichier `application.ini`. Dans le cas présent, on initialise quelques composants de base du cadriciel tels que le contrôleur principal, l'accès à la base de données, ... Le code source du fichier `application.ini` est disponible en annexe.

Bien sûr, ces différents paramètres peuvent changer entre la machine de développement et le serveur de production. Plusieurs contextes d'exécution sont donc prévus :

- `production` pour l'environnement définitif ;
- `staging` et `testing` pour des phases intermédiaires ;
- `development` pour la phase de développement.

Le contexte `production` permet par exemple d'initialiser l'accès à la base de données à travers les clés débutant par `resources.db`. Celles-ci sont surchargés dans le contexte `development` qui hérite de `production`.

Une fois que le fichier `application.ini` a été traité, le processus se poursuit avec l'exécution du « bootstrap ». Ce dernier est contenu dans une classe du cadriciel qu'il faut étendre pour les besoins de l'application. Il sert à enrichir la configuration en offrant au développeur toute la souplesse du PHP mais surtout l'accès aux différentes briques du cadriciel. Pour cela, il suffit de définir une méthode publique dont le nom commence par « `init` » et contenant les instructions souhaitées.

Dans le cas de l'application serveur, le fichier `Bootstrap.php` – dont le code source est disponible en annexe – permet de définir plusieurs méthodes :

- `__initDoctype()` pour sélectionner le standard XHTML strict lors de la génération des vues ;
- `__initTranslate()` pour définir le chemin de recherche et le type de structure utilisée par le composant d'internationalisation ;
- `__initNavigation()` pour définir le chemin de recherche et le type de structure utilisée par le composant de navigation ;
- `__initAcl()` pour charger les profils des droits utilisateur depuis un fichier externe et attribuer les droits correspondants à l'utilisateur courant ;
- `__initHelpers()` pour charger différentes aides de vues.

4.3.3 Modèle de données

Le modèle de données est en réalité découpé en trois couches qui seront présentées ci-après :

- le modèle de bas niveau, très proche de la base de données ;
- le modèle de haut niveau, très proche de la logique métier de l'application ;
- la couche d'adaptation, nécessaire pour la « traduction » entre les modèles de bas et haut niveaux.

4.3.3.1 Modèle de bas niveau

Pour réaliser le modèle de données, on utilise le composant `Zend_Db` du cadre. Chaque table issue de la modélisation est décrite dans une classe dérivée de `Zend_Db_Table_Abstract`. Même s'il est possible de décrire les relations avec les autres tables, l'implémentation retenue se limite à préciser le nom de la table ainsi que la ou les colonnes composant la clé primaire. Par exemple, la classe `process` aura la forme suivante :

```

5 <?php
  class Application_Model_Process_Db_Table_Process extends Zend_Db_Table_Abstract
  {
    protected $_name = 'Process';
  }
  ?>

```

Une telle classe offre en standard plusieurs méthodes pour effectuer des requêtes sur la table adéquate. Ainsi sont disponibles :

- `insert()` pour insérer de nouvelles données ;
- `update()` pour mettre à jour des données ;
- `delete()` pour supprimer des données ;
- `find()` pour rechercher des données par leur clé primaire ;
- `fetchAll()` pour lister toutes les données correspondant aux critères de recherche ;
- `fetchRow()` pour lister une ligne de données correspondant aux critères de recherche.

4.3.3.2 Modèle de haut niveau

Le modèle de haut niveau est établi à partir du diagramme de classes. Chaque classe permet de décrire une entité logique et apporte ainsi une structure « métier » au modèle de données.

Un objet sera instancié à partir d'une de ces classes via une fabrique. De cette manière, toutes les requêtes faisant appel au modèle de bas niveau sont rassemblées au sein de cette fabrique. L'objet instancié ne sert que de conteneur pour l'information et la logique métier.

A titre d'exemple, les codes sources de la classe `process` et de sa fabrique d'objets sont disponibles en annexe.

4.3.3.3 Couche d'adaptation

Pour chaque table du modèle, une classe `mapper` est créée. Elle offre quelques méthodes pour chercher (`fetch`), insérer (`insert`), mettre à jour (`update`) ou supprimer (`delete`) des informations dans la table adéquate. Chaque méthode effectue une correspondance entre les colonnes de la table et les attributs (ou accesseurs) d'un objet métier.

A titre d'exemple, le code source de l'adaptateur de la classe `process` est disponible en annexe.

4.3.4 Contrôleurs

L'application est découpée en plusieurs blocs fonctionnels correspondant aux différentes entités logiques du modèle. Chaque bloc dispose d'un « contrôleur », c'est-à-dire d'une classe dérivée de `Zend_Controller` qui regroupe les « actions » possibles.

Une action décrit un comportement particulier de l'application. Elle peut par exemple initialiser un formulaire, solliciter le modèle pour effectuer un traitement,... Sauf si cela est explicitement précisé, une action initialise une vue qui sera utilisée pour l'affichage en fin de processus.

A titre d'exemple, le code source du contrôleur des processus est disponible en annexe.

4.3.5 Vues

Le squelette général des pages est contenu dans un « layout ». C'est à cet endroit que sont définies les différentes zones telles que le bandeau, le corps principal ou encore le pied de page.

Au sein du Zend Framework, une vue est une portion de page qui s'inscrit dans le layout. Celle-ci est définie dans un fichier qui ne contient que le code nécessaire à la génération de portion de page attendue. Ce code peut être du simple HTML ou une structure PHP plus évoluée. Dans ce cas, il est notamment possible d'utiliser un objet `this` contenant la vue instanciée par le contrôleur.

4.3.5.1 Gestion des droits et des utilisateurs

Afin de contrôler finement l'accès aux données, il est nécessaire de définir chaque action possible dans l'application. Dans notre cas, la limitation se fera selon le contrôleur demandé. Chaque contrôleur correspond alors à une « ressource ». Il est ensuite nécessaire de lier les utilisateurs à ces ressources. Cependant, il n'est pas judicieux de lister exhaustivement les utilisateurs. On définit alors des rôles correspondant chacun à une catégorie d'utilisateurs donnés possédant un ensemble de droits.

Chaque rôle est déclaré par un bloc `[rôle]`. Si le rôle surcharge un autre rôle, la déclaration aura plutôt la forme `[rôle_enfant : rôle_enfant]`. Les lignes suivantes définissent chacune un droit de ce rôle. Elles peuvent prendre plusieurs formes :

- `allow.ressource = null`, pour autoriser tous les accès à une ressource ;
- `allow.ressource = droit1, droit2, droit3`, pour autoriser certains accès à une ressource ;
- `deny.ressource = null`, pour refuser tous les accès à une ressource ;
- `deny.ressource = droit1, droit2, droit3`, pour refuser certains accès à une ressource.

L'ensemble de la structure est disponible en annexe.

4.3.5.2 Navigation

Le terme « navigation » n'est pas à confondre avec l'ergonomie de l'application même s'il est lié. Dans le cadre du Zend Framework, cela correspond en réalité à l'arborescence des pages accessibles via le ou les menus.

On utilise une structure XML pour décrire cet arbre. Chaque page est caractérisée par :

- un titre ;
- une « route » ;

- un contrôleur ;
- une action ;
- une ressource ;
- un droit d'accès ;
- un ensemble de sous-pages.

Le fichier créé pour l'application serveur est disponible en annexe.

4.3.5.3 Internationalisation

Afin de rendre l'interface plus agréable à utiliser, il est possible de traduire les différents textes qu'elle contient. Ainsi, suivant le choix de l'utilisateur – ou à défaut, suivant les réglages du navigateur – l'interface pourra être localisée par exemple en français ou en anglais.

Il faut au préalable constituer un dictionnaire pour contenir toutes les chaînes à traduire. Chaque chaîne sera identifiable par une « clé ». Pour chaque langue, on y associera le texte traduit. Avec Zend Framework, ce dictionnaire peut être stocké sous différentes formes :

- une structure PHP native de type tableau ;
- une structure XML ;
- une liste de couples de la forme `clé = valeur`.

La structure PHP a été adoptée car elle offre un chargement des données plus rapide que les autres lors desquelles une étape d'analyse syntaxique supplémentaire est requise.

Pour tirer partie de l'internationalisation au sein de l'application, on commence par instancier un objet `Zend_Translate` depuis le bootstrap pour charger en mémoire le dictionnaire. Ensuite, à tout endroit où un message localisé doit être inclus, il suffit de récupérer l'objet `Zend_Translate` et d'appeler la méthode `translate` en précisant la clé en paramètre.

4.3.6 Automatisation

Pour automatiser la plateforme, il est nécessaire de mettre en place un déclenchement régulier. Pour cela, on utilise le système de planification Cron sur le serveur. Ce service permet d'exécuter une commande à des intervalles définis. Il suffit donc de planifier un appel vers un contrôleur de l'application chargé de déclencher les processus en attente.

4.4 Implémentation des couches de communication

Les couches de communication représentent une partie importante de l'application. Il faut distinguer plusieurs aspects :

- la communication entre le serveur applicatif et les hôtes ;
- la synchronisation entre les parties serveur et client de l'application ;
- la gestion des ressources de calcul.

4.4.1 Communication entre le serveur applicatif et les hôtes

Le premier aspect dépend essentiellement des systèmes d'exploitation utilisés et des possibilités offertes pour l'exécution des commandes. Les grappes de la grille de calcul utilisant nativement le protocole SSH, celui-ci a été retenu comme protocole principal pour l'application.

Pour la communication entre l'application et les différentes machines, le protocole SSH a été retenu. En effet, il allie souplesse et sécurité. Une connexion SSH permet de se connecter à distance sur un hôte et d'y exécuter des commandes. Par le biais d'une couche FTP (SFTP), il est possible d'échanger des fichiers.

Différentes implémentations sont disponibles pour le langage PHP. La première est une extension à activer dans la configuration. Elle est basée sur libssh et offre ainsi de bonnes performances. Malheureusement, elle souffre de quelques défauts et n'est pas forcément disponible sur tous les systèmes.

La deuxième solution, phpSecLib, est une implémentation native en PHP. De ce fait, son utilisation offre des performances moins bonnes que libssh puisque son code doit être interprété à chaque appel. En revanche, son principal avantage est d'être ainsi disponible sur n'importe quel système sans aucune contrainte de configuration puisqu'elle se trouve directement intégrée au code source de l'application.

La bibliothèque phpSecLib est disponible sur le site <http://phpseclib.sf.net> en licence LGPL. Après avoir téléchargé la dernière version, il faut la décompresser dans un sous-dossier du projet tel que `/library/ssh`. Afin d'utiliser phpSecLib dans le code de l'application, il est nécessaire d'inclure les fichiers adéquats.

4.4.1.1 Ouverture de connexion

Selon qu'on souhaite ouvrir une connexion SSH ou SFTP, la bibliothèque nous propose plusieurs constructeurs, chacun ayant les mêmes paramètres : l'hôte, le numéro de port (22 par défaut) et le délai d'attente (10 secondes par défaut). L'instance ainsi créée est ensuite utilisée dans tous les traitements jusqu'à la fermeture de la connexion. Pour terminer la phase d'ouverture de connexion, il est nécessaire de s'authentifier auprès de l'hôte soit avec un couple nom d'utilisateur/mot de passe, soit avec un couple nom d'utilisateur/clé privée.

Voici un exemple de connexion SFTP avec authentification par mot de passe :

```
include('Net/SFTP.php');

$sftp = new Net_SFTP('www.domain.tld');
4 if (!$sftp->login('username', 'password')) {
    exit('Login Failed');
}
```

Voici un exemple de connexion SSH avec authentification par clé privée :

```
include('Crypt/RSA.php');
include('Net/SSH2.php');

4 $key = new Crypt_RSA();
  // $key->setPassword('whatever');
  $key->loadKey(file_get_contents('privatekey'));

5 $ssh = new Net_SSH2('www.domain.tld');
9 if (!$ssh->login('username', $key)) {
    exit('Login Failed');
}
```

4.4.1.2 Exécution de commandes

Une fois que la connexion SSH est établie, il suffit d'utiliser la méthode `exec()` de l'objet instancié pour exécuter une commande sur l'hôte distant. Cela donne par exemple :

```
echo $ssh->exec('pwd');  
echo $ssh->exec('ls -la');
```

4.4.1.3 Gestion de fichiers

Une connexion ouverte en mode SSH ne permet que d'exécuter des commandes. Le mode SFTP présente l'avantage de proposer des méthodes dédiées à la gestion de fichiers.

La méthode `pwd()` retourne le chemin courant. `chdir($chemin)` permet de changer de dossier avec la chaîne passée en paramètre. `mkdir($chemin)` crée un dossier et `rmdir($chemin)` le supprime.

Pour renommer un fichier, il faut utiliser la méthode `rename($ancien_nom, $nouveau_nom)`. La méthode `delete($chemin)` permet quant à elle de supprimer un fichier dont le chemin est passé en paramètre.

Pour déposer un fichier, il faut utiliser la méthode `put(chemin, contenu)` en spécifiant en paramètre, le chemin de destination et le contenu à écrire. A l'inverse, la méthode `get($chemin_source, $chemin_cible)` récupère le contenu du chemin source pour l'écrire dans le chemin cible.

4.4.1.4 Fermeture de connexion

La connexion est interrompue automatiquement à la destruction de l'objet. Il est cependant possible de le faire manuellement en appelant la méthode `disconnect()` comme dans cet exemple :

```
$ssh->disconnect();
```

4.4.2 La synchronisation entre les parties serveur et client

La communication entre l'application serveur et une application cliente itinérante sera basée sur le protocole HTTP via le port 80 afin qu'elle ne soit pas bloquée par d'éventuels pare-feu ou proxies comme c'est le cas au LPG. Pour cela, l'architecture REST a été retenue pour sa facilité d'utilisation et de mise en œuvre.

En fait, le terme REST ne désigne pas un protocole ou un format mais un style d'architecture. Tout est basé sur l'URI et les différentes opérations fournies par le protocole HTTP. Les données échangées peuvent être formatées via les langages XML ou JSON.

Par exemple, l'URI `http://simulation-consert.obs.ujf-grenoble.fr/rest/project/1` désigne la ressource composée par l'objet projet d'identifiant « 1 ». Les opérations HTTP suivantes sont possibles :

- GET pour obtenir le contenu de la ressource ;
- PUT et POST pour modifier la ressource en envoyant un nouveau contenu ;
- DELETE pour supprimer la ressource.

Ainsi, en établissant certaines règles de construction d'URI, il est possible de définir une API qui sera utilisable pour l'échange entre l'application principale et ses versions clientes itinérantes mais également pour faciliter la création d'une interface native en Delphi.

Malheureusement, cet aspect n'a été qu'abordé pendant le stage et le développement de cette couche n'a pas été entièrement réalisé. Celle-ci ne fait donc pas partie de la version de l'application livrée en fin de stage.

4.4.3 Gestion des ressources de calcul

La grille CIMENT propose deux intergiciels pour soumettre et gérer des calculs. Leur utilisation est rendue possible grâce à la couche de communication SSH présentée plus haut. Il s'agit OAR et CiGri dont les commandes ont été décrites au chapitre précédent.

4.5 Bilan

La nouvelle plateforme se décompose en quatre blocs distincts :

- la grille de calcul sur laquelle fonctionne l'outil de simulation ;
- le serveur de stockage sur lequel sont centralisés les fichiers produits par les simulations ;
- le serveur web accueillant l'application principale de la plateforme et sa base de données ;
- le poste client accueillant l'outil d'interprétation des résultats et servant de point d'accès à l'application principale.

L'application « principale » est donc au cœur de la plateforme. Son architecture interne a été conçue afin de paramétrer aisément les différentes tâches composant une chaîne de simulation générique. Pour cela, plusieurs types de tâches ont été définis :

- `commande` pour l'exécution distante d'une commande quelconque ;
- `transfert, envoi, réception` pour échanger des fichiers entre les hôtes ;
- `job CiGri` pour gérer spécifiquement l'intergiciel de grille CiGri ;
- `attente` pour marquer une pause dans la séquence.

L'application a été réalisée dans le langage PHP à l'aide du cadre Zend Framework. L'architecture respecte le patron MVC et le découpage en couches proposé par le cadre.

La communication entre l'application et les différents hôtes se fait via une connexion SSH. La librairie `phpSecLib` a ainsi été utilisée pour faciliter l'implémentation de cette couche au sein de l'application. Par ailleurs, seul l'intergiciel CiGri a été nativement interfacé. Son utilisation se fait également au travers d'une liaison SSH.

Utilisation de la nouvelle plateforme de simulation

Avant de présenter les différents écrans de l'application Web réalisée, il peut être utile de préciser leurs enchaînements. Nous utiliserons pour cela les diagrammes de cas d'utilisation issus du formalisme UML. Ainsi les principaux cas d'utilisation seront présentés afin de préciser les possibilités offertes aux utilisateurs en fonction de leurs rôles.

5.1 Principaux cas d'utilisation

La figure 36 présente une vue partielle de l'ensemble des cas d'utilisation de l'application Web. Les interactions entre les différentes « actions », c'est-à-dire les enchaînements entre les différents écrans, sont ainsi matérialisées par des relations d'extension. La relation d'inclusion est utilisée pour représenter une réutilisation d'un écran externe au sein d'une action. Les acteurs ont volontairement été omis par souci de lisibilité mais ils seront présents sur les figures suivantes.

La figure 37 présente une vue partielle des cas d'utilisation spécifiques aux utilisateurs. Un *invité*, c'est-à-dire un utilisateur non authentifié, ne peut qu'accéder à l'écran de connexion. Une fois authentifié, il devient *membre* et modifier son profil (et uniquement le sien). Seul un *administrateur* est habilité à créer de nouveaux utilisateurs.

La figure 38 présente une vue partielle des cas d'utilisation communs à tous les modules de l'application. En général, un *membre*, c'est-à-dire un utilisateur authentifié mais non-administrateur, ne peut que lister et consulter les éléments. Un administrateur peut en revanche créer, modifier et supprimer des éléments. Ce diagramme s'applique notamment aux projets, aux séquences, aux variables et aux hôtes.

La figure 39 présente une vue partielle des cas d'utilisation spécifiques aux processus. Les membres peuvent créer un processus ou un groupe de processus. Ils peuvent également modifier un processus, le planifier – c'est-à-dire le mettre en attente d'exécution –, lancer ou arrêter son exécution. Le « système » peut quant à lui lancer l'exécution des processus en attente ou dans certains cas, arrêter leur exécution. Le diagramme des cas d'utilisation des tâches est quasiment similaire à au diagramme concernant les processus ; il ne sera donc pas présenté.

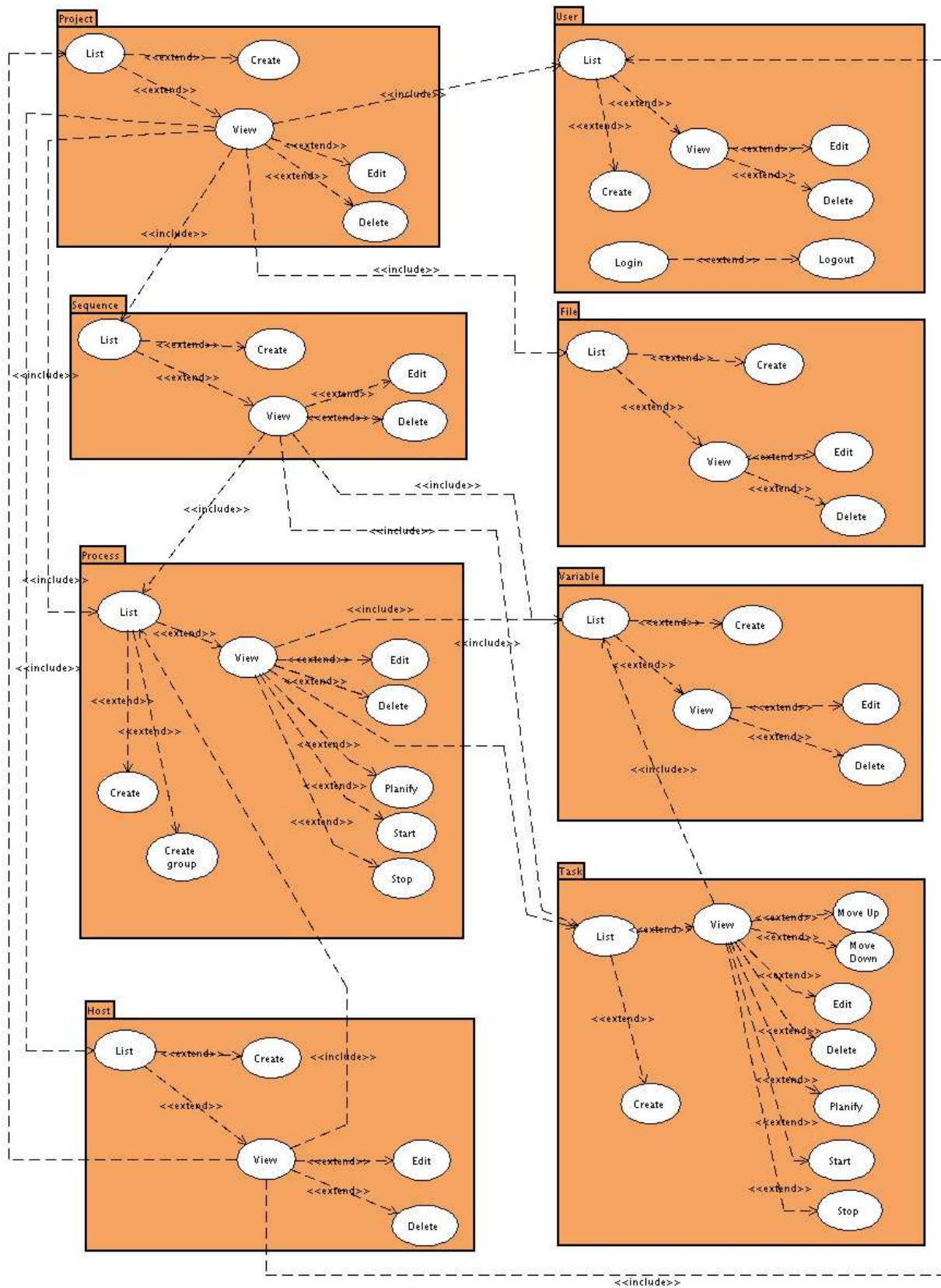


FIGURE 36 – Diagramme partiel de l'ensemble des cas d'utilisation.

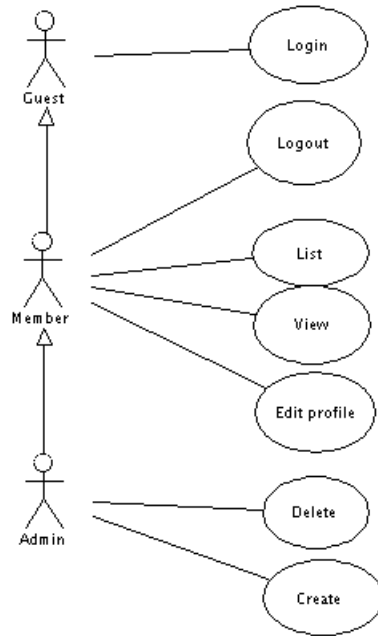


FIGURE 37 – Diagramme partiel des cas d'utilisation des utilisateurs.

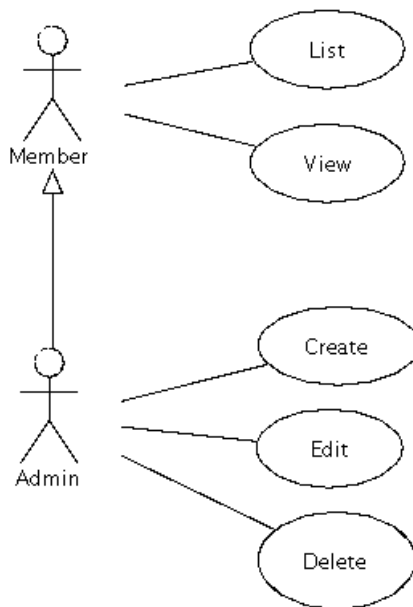


FIGURE 38 – Diagramme partiel des cas d'utilisation communs.

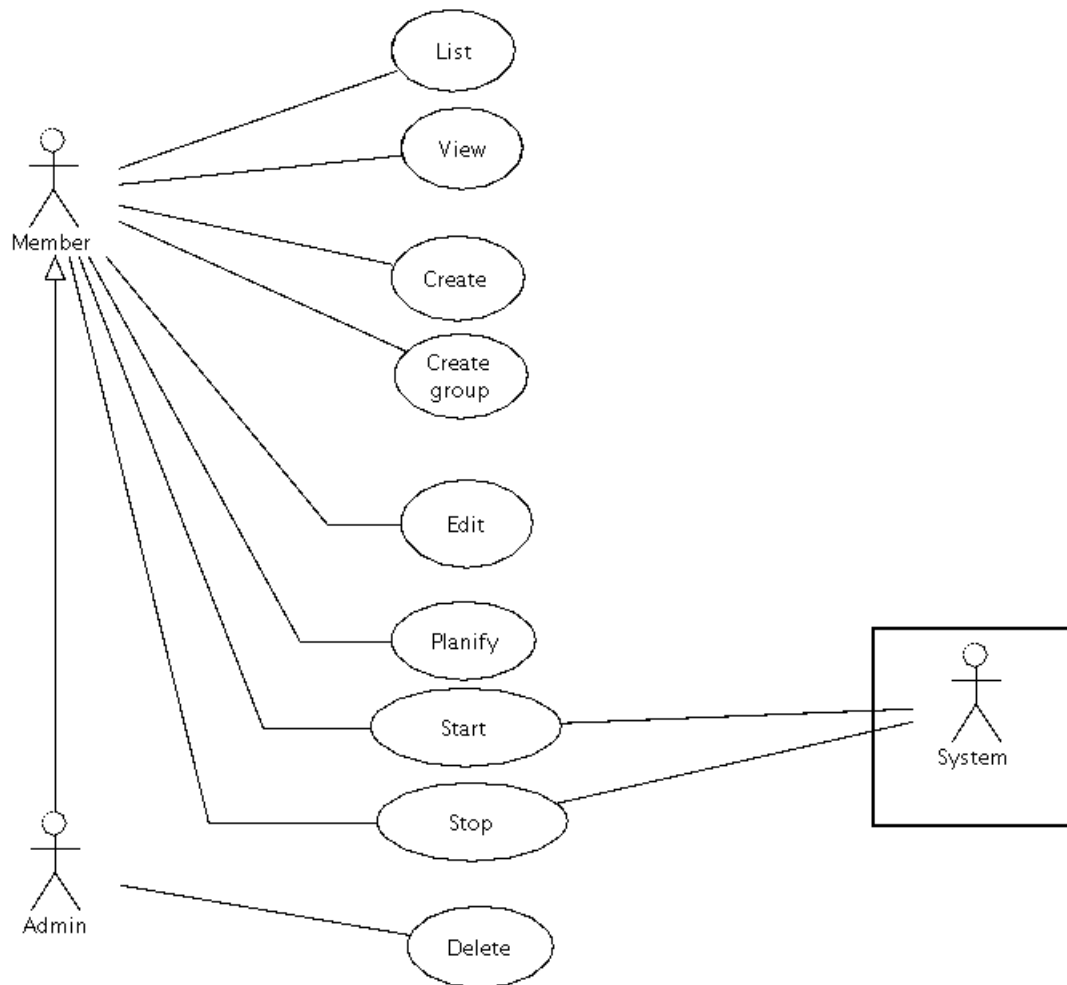


FIGURE 39 – Diagramme partiel des cas d'utilisation des processus.

5.2 Utilisation de l'interface

Cette partie présente les différents écrans de l'application Web principale réalisée pour la plateforme. Elle peut donc être considérée comme un manuel utilisateur.

5.2.1 Connexion

Pour accéder à la plateforme, il est tout d'abord nécessaire de s'identifier via le formulaire de connexion représenté sur l'écran 40. Celui-ci ne contient que deux champs respectivement pour le nom d'utilisateur et pour le mot de passe.



Écran 40 – Formulaire d'identification.

Une fois authentifié, l'utilisateur découvre la page d'accueil telle que présentée sur l'écran 41. En zone principale, deux liens permettent respectivement d'accéder à la recherche de processus et à la liste des processus.

Chaque page respecte la même disposition. En haut, une barre contient de gauche à droite :

- un lien pour revenir à la page d'accueil ;
- le menu déroulant avec les entrées principales pour projet, hôte et utilisateur ;
- le nom de l'utilisateur courant ;
- un lien permettant d'accéder au profil de l'utilisateur courant ;
- un lien pour se déconnecter.

La partie centrale contient la zone principale. Enfin, en bas, une barre contient un lien permettant de revenir à tout moment à la page précédente.



Écran 41 – Page d'accueil et ergonomie générale.

5.2.2 Hôte

5.2.2.1 Création d'un hôte

Le formulaire de création d'un hôte contient les champs de base tels que le nom et la description. Comme le montre l'écran 42, le champ suivant permet de saisir le nom canonique identifiant la machine sur un réseau.

Écran 42 – Formulaire de création d'un hôte de type Grille.

Un champ déroulant permet de sélectionner le type de l'hôte à créer. Lorsque l'utilisateur sélectionne une valeur dans la liste, l'interface effectue une requête Ajax pour obtenir les champs supplémentaires correspondants. Les écrans 43a et 43b présentent respectivement les compléments dans le cas des types grappe et nœud.

Écran 43 – Champs complémentaires pour un hôte

5.2.2.2 Liste des hôtes

L'écran 44 représente la page listant les hôtes configurés dans l'interface. Chaque ligne indique :

- le nom de l'hôte avec un lien permettant d'accéder à sa page de détails ;
- le type ;
- la date de dernière mise à jour ;
- la description.

En bas de page, deux liens amènent respectivement sur les pages de création et de recherche d'hôtes.

Liste des hôtes

Nom	Type	Mise à jour	Description
Plateforme	Noeud	2010-07-29 15:07:59	Serveur exécutant cette application
Frontale CiGri	Grille	2010-07-29 15:08:31	Frontale de la grille de calcul intensif CIMENT
Venus	Noeud	2010-06-15 15:37:34	

Nouveau | Rechercher
Page 1 | Page 2

Écran 44 – Liste des hôtes.

5.2.2.3 Détails d'un hôte

L'écran 45 représente l'onglet principal de la page listant les informations d'un hôte. On y retrouve donc :

- l'identifiant ;
- le nom de l'hôte ;
- la description ;
- le type ;
- la date de création ;
- la date de dernière mise à jour ;
- le nom de machine ;
- le dossier par défaut.

En bas de page, trois liens permettent respectivement de modifier, supprimer et dupliquer l'hôte courant.

Détails d'un hôte

Hôte		Grappes	Utilisateurs	Projets
Id	4bcf1c3fca93e			
Nom	Frontale CiGri			
Description	Frontale de la grille de calcul intensif CIMENT			
Création	2010-04-21 17:39:43			
Mise à jour	2010-07-29 15:08:31			
Type	Grille			
Nom de machine	calcule.imag.fr			
Dossier par défaut				

Modifier | Supprimer | Dupliquer

Écran 45 – Liste des informations principales d'un hôte.

L'écran 46 représente le deuxième onglet sur lequel sont listés les grappes ou les nœuds associés à l'hôte courant selon son type. La structure de la page est identique à celle de la liste des hôtes.

L'écran 47 représente le troisième onglet sur lequel sont listés les utilisateurs associés à l'hôte courant. La structure de la page est identique à celle de la liste des utilisateurs. Seul un lien `retirer` sur chaque ligne permet de détruire l'association entre l'utilisateur et l'hôte.

L'écran 48 représente le quatrième onglet sur lequel sont listés les projets associés à l'hôte courant. La structure de la page est identique à celle de la liste des projets. Seul un lien `retirer` sur chaque ligne permet de détruire l'association entre l'utilisateur et le projet.

Détails d'un hôte

Hôte			Grappes	Utilisateurs	Projets
Nom	Création	Description			
Cluster R2D2	2010-04-21 17:40:25	Grappe de calcul faisant partie de la grille CiGri			
Cluster Fostino	2010-04-21 17:41:05	Grappe de calcul faisant partie de la grille CiGri			

Écran 46 – Liste des grappes ou nœuds rattachés à un hôte.

Détails d'un hôte

Hôte				Grappes	Utilisateurs	Projets
Login	Prénom	Nom	Actions			
bollardp	Philippe	Bollard	Retirer			

Ajouter

Écran 47 – Liste des utilisateurs d'un hôte.

Détails d'un hôte

Hôte			Grappes	Utilisateurs	Projets
Nom	Description			Actions	
Consert	Simulation de l'expérience CONCERT			Retirer	

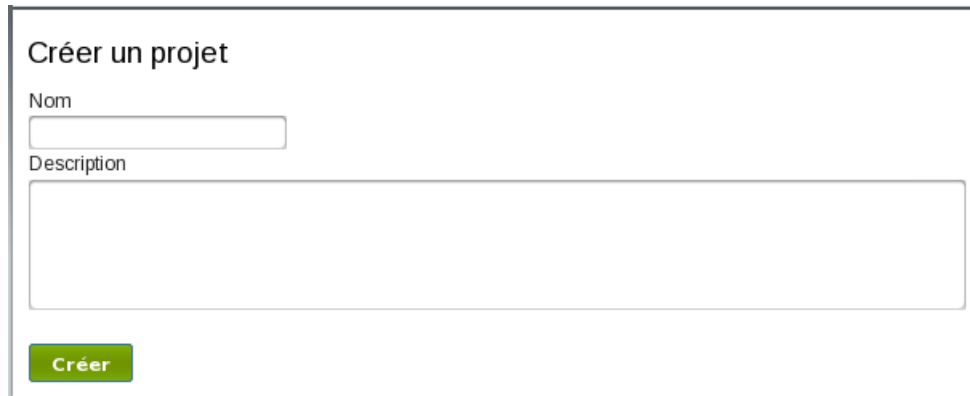
Ajouter

Écran 48 – Liste des projets d'un hôte.

5.2.3 Projet

5.2.3.1 Création d'un projet

Comme le montre l'écran 49, le formulaire de création d'un projet ne contient que les champs de base pour le nom et la description.



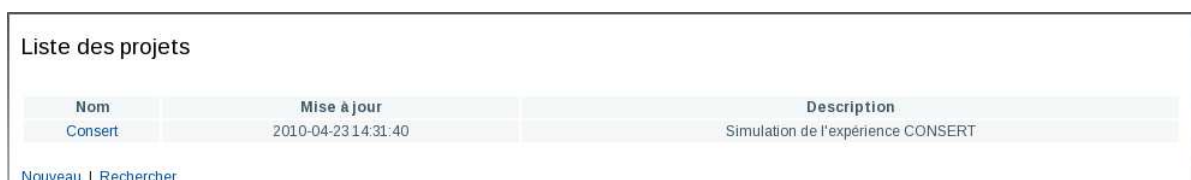
Écran 49 – Formulaire de création d'un projet.

5.2.3.2 Liste des projets

L'écran 50 représente la page listant les projets configurés dans l'interface. Chaque ligne indique :

- le nom du projet avec un lien permettant d'accéder à sa page de détails ;
- la date de dernière mise à jour ;
- la description.

En bas de page, deux liens amènent respectivement sur les pages de création et de recherche de projet.



Nom	Mise à jour	Description
Consert	2010-04-23 14:31:40	Simulation de l'expérience CONSERT

Nouveau | Rechercher

Écran 50 – Liste des projets.

5.2.3.3 Détails d'un projet

L'écran 51 représente l'onglet principal de la page listant les informations d'un projet. On y retrouve donc :

- l'identifiant ;
- le nom du projet ;
- la description ;
- la date de création.

En bas de page, trois liens permettent respectivement de modifier, supprimer et dupliquer le projet courant.



Écran 51 – Informations principales d'un projet.

L'écran 52 représente le deuxième onglet sur lequel sont listées les séquences associées au projet courant. La structure de la page est identique à celle de la liste des séquences. Un lien en bas de page permet d'accéder à page de création d'une séquence.



Écran 52 – Liste des séquences d'un projet.

L'écran 53 représente le troisième onglet sur lequel sont listés les processus associés au projet courant. La structure de la page est identique à celle de la liste des processus. Deux liens en bas de page permettent respectivement d'accéder aux pages de création d'un processus et d'une série de processus.



Écran 53 – Liste des processus d'un projet.

L'écran 54 représente le quatrième onglet sur lequel sont listés les fichiers associés au projet courant. La structure de la page est identique à celle de la liste des fichiers. Un lien en bas de page permet d'accéder à la page de création d'un fichier.

L'écran 55 représente le cinquième onglet sur lequel sont listés les hôtes associés au projet courant. La structure de la page est identique à celle de la liste des hôtes. Un lien en bas de page permet d'accéder à la page d'ajout d'un hôte.

L'écran 56 représente le sixième onglet sur lequel sont listés les utilisateurs associés au projet courant. La structure de la page est identique à celle de la liste des utilisateurs. Un lien en bas de page permet d'accéder à la page d'ajout d'un utilisateur.

Détails d'un projet

Projet	Séquences	Processus	Fichiers	Hôtes	Utilisateurs
Nom	Fichier	Mise à jour	Description		
Source Fortran V17	WAVE3dc3_v17.f	2010-08-04 11:30:36	Code de simulation limité à une sphère		
Modele structure interne homogene	cometis3d1.dat	2010-08-04 11:30:43	modèle de structure interne homogène		
Modele de forme 67P	67PA_m.dat	2010-08-04 11:30:51	Modèle de forme basé sur le modèle de Lamy		
Script build_params_txt	build_params_txt.sh	2010-08-04 11:38:50	Script paramétrant le fichier params.txt nécessaire au lancement des jobs d'une campagne CiGri pour la simulation fortran		
Données polarisation	lander_polar.dat	2010-04-23 17:20:39			
Script build_params_prg	build_params_prg.sh	2010-08-04 11:35:55	Script paramétrant le fichier params.prg nécessaire à la simulation Fortran		
Script deploy_soft	deploy_software_cigri.sh	2010-08-04 11:36:30	Script déployant et compilant la simulation fortran sur les clusters de CiGri		
Script run_job	run_job_cigri.sh	2010-08-04 11:37:19	Script permettant de lancer l'exécution d'un job de simulation fortran sur un cluster de CiGri		
Script fetch_results	fetch_results_cigri.sh	2010-08-04 11:38:12	Script permettant de concaténer et de compresser les résultats d'une campagne CiGri pour la simulation Fortran		
Modèle sphère	sphere_harmonique.dat	2010-08-04 11:30:19	Forme sphérique		
Modele structure interne Blob V1	cometis3d1_blob.dat	2010-08-04 11:31:36	Modèle de structure interne avec blob de perturbation		
Porosité Interne 1blob	interne2.dat	2010-08-04 11:32:21	Fichier de porosité interne avec un seul blob centré sur <0,0,0> et de rayon 1000		

Nouveau fichier

Écran 54 – Liste des fichiers d'un projet.

Détails d'un projet

Projet	Séquences	Processus	Fichiers	Hôtes	Utilisateurs
Nom	Type	Description			Actions
Plateforme	Noeud	Serveur exécutant cette application			Retirer
Frontale CiGri	Grille	Frontale de la grille de calcul intensif CIMENT			Retirer
Cluster R2D2	Grappe	Grappe de calcul faisant partie de la grille CiGri			Retirer
Cluster Fostino	Grappe	Grappe de calcul faisant partie de la grille CiGri			Retirer
Venus	Noeud				Retirer

Ajouter

Écran 55 – Liste des hôtes d'un projet.

Détails d'un projet

Projet	Séquences	Processus	Fichiers	Hôtes	Utilisateurs
Login	Prénom	Nom	Actions		
bollardp	Philippe	Bollard	Retirer		
chaillos	Stéphane	Chaillo	Retirer		

Ajouter

Écran 56 – Liste des utilisateurs d'un projet.

5.2.4 Fichier

5.2.4.1 Création d'un fichier

Comme le montre l'écran 57, le formulaire de création d'un fichier contient les champs de base pour le nom et la description. Un champ supplémentaire permet de sélectionner un fichier sur le poste de l'utilisateur afin de l'envoyer vers le serveur.



The screenshot shows a web form titled 'Consert > Créer un fichier'. It contains the following elements: a 'Nom' label above a text input field; a 'Description' label above a larger text area; a 'Fichier' label above a file selection input field with a 'Parcourir...' button; and a green 'Créer' button at the bottom left.

Écran 57 – Formulaire de création de fichier.

5.2.4.2 Liste des fichiers

L'écran 58 représente la page listant les fichiers configurés dans l'interface. Chaque ligne indique :

- le nom du fichier avec un lien permettant d'accéder à sa page de détails ;
- le nom d'origine du fichier ;
- la date de dernière mise à jour ;
- la description.

En bas de page, deux liens amènent respectivement sur les pages de création et de recherche de fichier.

5.2.4.3 Détails d'un fichier

L'écran 59 représente l'onglet principal de la page listant les informations d'un fichier. On y retrouve donc :

- l'identifiant ;
- le nom du fichier (interne à l'interface) ;
- la description ;
- la date de création ;
- le nom du projet associé avec un lien permettant d'accéder à sa page de détails ;
- le nom d'origine du fichier ;
- le chemin de stockage du fichier.

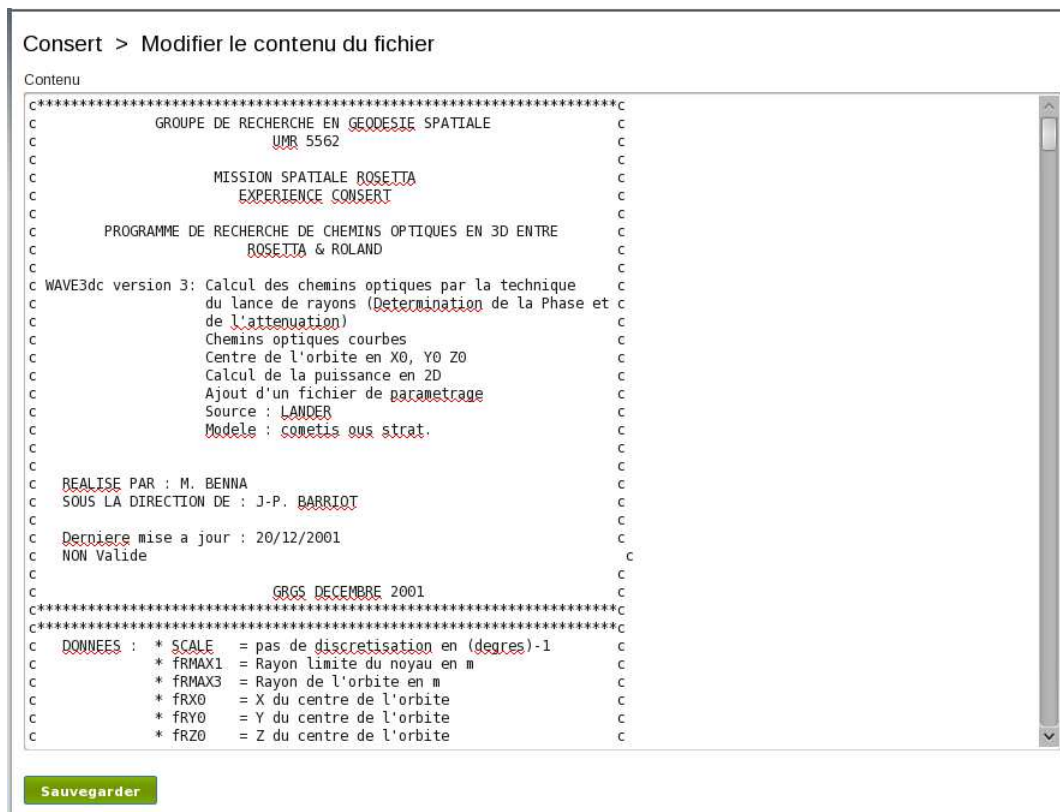
En bas de page, cinq liens permettent respectivement de modifier le fichier courant, de voir et éditer son contenu (comme le représente l'écran 60), de le supprimer, de le dupliquer et enfin de le télécharger.

Accueil Projet Hôte Utilisateur Philippe Bollard : Utilisateur Déconnexion			
Liste des fichiers			
Nom	Nom du fichier	Mise à jour	Description
Source Fortran V17	WAVE3dc3_v17.f	2010-08-04 11:30:36	Code de simulation limité à une sphère
Modele structure interne homogene	cometis3d1.dat	2010-08-04 11:30:43	modèle de structure interne homogène
Modele de forme 67P	67PA_m.dat	2010-08-04 11:30:51	Modèle de forme basé sur le modèle de Lamy
Script build_params_txt	build_params_txt.sh	2010-08-04 11:38:50	Script paramétrant le fichier params.txt nécessaire au lancement des jobs d'une campagne CiGri pour la simulation fortran
Données polarisation	lander_polar.dat	2010-04-23 17:20:39	
Script build_params_prg	build_params_prg.sh	2010-08-04 11:35:55	Script paramétrant le fichier params.prg nécessaire à la simulation Fortran
Script deploy_soft	deploy_software_cigri.sh	2010-08-04 11:36:30	Script déployant et compilant la simulation fortran sur les clusters de CiGri
Script run_job	run_job_cigri.sh	2010-08-04 11:37:19	Script permettant de lancer l'exécution d'un job de simulation fortran sur un cluster de CiGri
Script fetch_results	fetch_results_cigri.sh	2010-08-04 11:38:12	Script permettant de concaténer et de compresser les résultats d'une campagne CiGri pour la simulation Fortran
Modèle sphère	sphere_harmonique.dat	2010-08-04 11:30:19	Forme sphérique
Page 1 Page 2			
Retour			

Écran 58 – Liste des fichiers.

Consert > Détails d'un fichier	
Fichier	
Id	4bd16723a4cba
Nom	Source Fortran V17
Description	Code de simulation limité à une sphère
Création	2010-04-23 11:23:47
Mise à jour	2010-04-23 11:23:47
Projet	Consert
Nom du fichier	WAVE3dc3_v17.f
Chemin du fichier	/home/bollardp/Workspace/Consert/Consert6/data/uploads/4bd16723a4cba/WAVE3dc3_v17.f
Modifier Contenu Supprimer Dupliquer Télécharger	

Écran 59 – Détails d'un fichier.



Écran 60 – Contenu d’un fichier.

5.2.5 Séquence

5.2.5.1 Création d’une séquence

Comme le montre l’écran 61, le formulaire de création d’une séquence ne contient que les champs de base pour le nom et la description.



Écran 61 – Formulaire de création d’une séquence de tâches.

5.2.5.2 Liste des séquences

L’écran 62 représente la page listant les séquences configurées dans l’interface. Chaque ligne indique :

- le nom de la séquence avec un lien permettant d’accéder à sa page de détails ;

- la date de dernière mise à jour ;
- la description.

Liste des séquences

Nom	Mise à jour	Description
Simulation Fortran	2010-04-21 17:47:27	
Déploiement simulation fortran	2010-07-26 16:57:30	Déploiement de la simulation Consert en Fortran

Page 1 | Page 2

Écran 62 – Liste des séquences.

5.2.5.3 Détails d'une séquence

L'écran 63 représente l'onglet principal de la page listant les informations d'une séquence. On y retrouve donc :

- l'identifiant ;
- le nom du projet associé avec un lien permettant d'accéder à sa page de détails ;
- le nom de la séquence ;
- la description ;
- la date de création ;
- la date de dernière mise à jour.

En bas de page, trois liens permettent respectivement de modifier, supprimer et dupliquer la séquence courante.

Consert > Détails d'une séquence

Séquence	Tâches	Variabiles	Processus
Id	4c4da25a244be		
Projet	Consert		
Nom	Déploiement simulation fortran		
Description	Déploiement de la simulation Consert en Fortran		
Création	2010-07-26 16:57:30		
Mise à jour	2010-07-26 16:59:02		

[Modifier](#) | [Supprimer](#) | [Dupliquer](#)

Écran 63 – Informations principales d'une séquence.

L'écran 64 représente le deuxième onglet sur lequel sont listés les tâches associées à la séquence courante. La structure de la page est identique à celle de la liste des tâches. Celles-ci apparaissent dans l'ordre d'exécution. Pour chaque tâche, deux liens supplémentaires permettent respectivement de la descendre et de la monter. Un lien en bas de page permet d'accéder à page de création d'une tâche.

L'écran 65 représente le troisième onglet sur lequel sont listés les variables associées à la séquence courante. La structure de la page est identique à celle de la liste des variables. Un lien en bas de page permet d'accéder à page de création d'une variable.

L'écran 66 représente le quatrième onglet sur lequel sont listés les processus associés à la séquence courante. La structure de la page est identique à celle de la liste des processus. Deux liens en bas de page permettent respectivement d'accéder aux pages de création d'un processus et d'une série de processus.

Consert > Détails d'une séquence

Séquence Tâches Variables Processus

Nom	Type	Mise à jour	Actions
CIGRI - MakeDir Project	Commande	2010-07-29 15:02:14	Descendre Monter
CIGRI - Upload soft source	Upload	2010-07-29 11:49:12	Descendre Monter
CIGRI - Upload script deploy_soft	Upload	2010-07-29 11:49:12	Descendre Monter
CIGRI - Upload script build_params_txt	Upload	2010-07-29 11:49:12	Descendre Monter
CIGRI - Upload script build_params_prg	Upload	2010-07-29 11:49:12	Descendre Monter
CIGRI - Upload script run_job	Upload	2010-08-03 11:28:53	Descendre Monter
CIGRI - Upload script fetch_results	Upload	2010-08-03 11:28:53	Descendre Monter
CIGRI - Chmod +x project/*.sh	Commande	2010-08-03 11:28:53	Descendre Monter
CIGRI Run script deploy_soft.sh	Commande	2010-08-03 15:11:35	Descendre Monter

[Nouvelle tâche](#)

Écran 64 – Liste des tâches d'une séquence.

Consert > Détails d'une séquence

Séquence Tâches Variables Processus

Nom	Type	Code	Valeur	Tâche	Actions
Fichier soft source	Fichier	SOFTSRC	Source Fortran V17		Modifier Supprimer
Soft bin	Texte	SOFTBIN	soft_consert.bin		Modifier Supprimer
Fichier - script deploy_soft	Fichier	FILE	Script deploy_soft	CIGRI - Upload script deploy_soft	Modifier Supprimer
Fichier - script build_paramtxt	Fichier	FILE	Script build_params_txt	CIGRI - Upload script build_params_txt	Modifier Supprimer
Fichier - script build_paramprg	Fichier	FILE	Script build_params_prg	CIGRI - Upload script build_params_prg	Modifier Supprimer
Fichier - script run_job	Fichier	FILE	Script run_job	CIGRI - Upload script run_job	Modifier Supprimer
Fichier - script fetch_result	Fichier	FILE	Script fetch_results	CIGRI - Upload script fetch_results	Modifier Supprimer

[Nouvelle variable](#)

Écran 65 – Liste des variables d'une séquence.

Consert > Détails d'une séquence

Séquence Tâches Variables Processus

Nom	État	Mise à jour	Actions
SimuTest	Initialisé	2011-01-28 15:50:13	TODO

[Nouveau processus](#) | [Nouvelle série de processus](#)

Écran 66 – Liste des processus d'une séquence.

5.2.6 Tâche

5.2.6.1 Création d'une tâche

Comme le montre l'écran 67, le formulaire de création d'une tâche contient les champs de base pour le nom et la description.

Écran 67 – Formulaire de création d'une tâche de type commande.

Un champ déroulant permet de sélectionner le type de tâche à créer. Lorsque l'utilisateur sélectionne une valeur dans la liste, l'interface effectue une requête Ajax pour obtenir les champs supplémentaires correspondants. Les écrans 68a, 68b, 68c, 68d et 68e présentent respectivement les compléments dans le cas des types `job cigri`, `transfert`, `upload`, `download` et `attente`.

Il est possible de paramétrer une tâche. Dans le cas d'une commande, il est par exemple nécessaire de rendre générique une partie de la chaîne saisie et ainsi dépendante d'une « variable ». Pour cela, il suffit d'isoler ce segment et d'utiliser un code. La valeur sera automatiquement résolue lors de l'exécution de la tâche. Le code est attribué par l'utilisateur et peut prendre l'une des deux formes suivantes :

- `#CODE#` pour désigner la valeur d'une variable en utilisant son code ;
- `@OBJET-ATTRIBUT@` pour désigner la valeur d'un attribut d'un objet en fonction du contexte (par exemple `@task-id@` pour l'identifiant de la tâche).

5.2.6.2 Liste des tâches

L'écran 69 représente la page listant les tâches configurées dans l'interface. Chaque ligne indique :

- le nom de la tâche avec un lien permettant d'accéder à sa page de détails ;
- le type ;

(a) Type job CiGri

(b) Type transfert

(c) Type upload

(d) Type download

(e) Type attente

Écran 68 – Champs complémentaires pour une tâche

- le nom de la séquence avec un lien permettant d’accéder à sa page de détails ;
- la date de dernière mise à jour ;
- la description.

5.2.6.3 Détails d’une tâche

L’écran 70 représente l’onglet principal de la page listant les informations d’une tâche. On y retrouve donc :

- l’identifiant ;
- le nom de la tâche ;
- le type ;
- le nom de la séquence associée avec un lien permettant d’accéder à sa page de détails ;
- la description ;
- la date de création ;
- la date de dernière mise à jour.

En bas de page, trois liens permettent respectivement de modifier, supprimer et dupliquer la tâche courante.

L’écran 71 représente le deuxième onglet sur lequel sont listés les informations spécifiques dépendant du type de la tâche.

Liste des tâches

Nom	Type	Séquence	Mise à jour	Description
CIGRI Run build_paramsTxt.sh	Commande	Simulation Fortran	2010-04-22 17:49:11	
CIGRI Upload data forme	Upload	Simulation Fortran	2010-04-23 15:14:40	
CIGRI Upload data cometesimaux	Upload	Simulation Fortran	2010-04-23 15:15:28	
CIGRI Upload data polarisation	Upload	Simulation Fortran	2010-04-23 17:21:06	
CIGRI Run build_paramsPrg.sh	Commande	Simulation Fortran	2010-04-26 15:57:30	
CIGRI Deploy data on clusters	Commande	Simulation Fortran	2010-04-27 17:39:03	
CIGRI Submit job	Job CiGri	Simulation Fortran	2010-04-28 15:16:43	
CIGRI MakeDir Process	Commande	Simulation Fortran	2010-04-28 16:10:23	
CIGRI Run script.fetch_results.sh	Commande	Simulation Fortran	2010-05-20 15:32:42	
CIGRI-VENUS Transfer Results	Transfert	Simulation Fortran	2010-05-20 17:35:12	

Page 1 | Page 2

Écran 69 – Liste des tâches.

Consert > Simulation Fortran > Détails d'une tâche

Tâche	Commande	Variables
Id	4bd06ff7545a6	
Nom	CIGRI Run build_paramsTxt.sh	
Type	Commande	
Séquence	Simulation Fortran	
Description		
Création	2010-04-22 17:49:11	
Mise à jour	2010-08-02 10:08:32	

[Modifier](#) | [Supprimer](#) | [Dupliquer](#)

Écran 70 – Informations principales d'une tâche.

Consert > Simulation Fortran > Détails d'une tâche

Tâche	Commande	Variables
Hôte	Frontale CiGri	
Commande	@PROJECT-ID@/build_params_txt.sh #SOFTBIN# #THETAMIN# #THETAMAX# #THETAINC# #PHIMIN# #PHIMAX# #PHIINC# > @PROJECT-ID@/@PROCESS-ID@/params.txt	

Écran 71 – Informations spécifiques d'une tâche.

L'écran 72 représente le troisième onglet sur lequel sont listées les variables associées à la tâche courante. La structure de la page est identique à celle de la liste des variables. Un lien en bas de page permet d'accéder à page de création d'une variable.

Consert > Simulation Fortran > Détails d'une tâche

Tâche | Commande | Variables

Nom	Type	Code	Valeur	Actions
Theta Min	Entier	THETAMIN	0	Modifier Supprimer
Theta Max	Entier	THETAMAX	180	Modifier Supprimer
Theta Inc	Entier	THETAINC	2	Modifier Supprimer
Phi Min	Entier	PHIMIN	0	Modifier Supprimer
Phi Max	Entier	PHIMAX	360	Modifier Supprimer
Phi Inc	Entier	PHIINC	2	Modifier Supprimer

[Nouvelle variable](#)

Écran 72 – Liste des variables d'une tâche.

5.2.7 Variable

5.2.7.1 Création d'une variable

Le formulaire de création d'une variable contient les champs de base tels que le nom et la description. Comme le montre l'écran 73, le formulaire présente également un champ pour le code de la variable et une case à cocher déterminant si la variable est surchargeable ou non.

Consert > Simulation Fortran > Créer une variable

Nom

Description

Code

Surchargeable

Type

Valeur

Écran 73 – Formulaire de création d'une variable de séquence.

Un champ déroulant permet de sélectionner le type de variable à créer. Lorsque l'utilisateur sélectionne une valeur dans la liste, l'interface effectue une requête Ajax pour obtenir les champs supplémentaires correspondants. L'écran 74 présente par exemple le complément dans le cas du type fichier.

Écran 74 – Champs complémentaires pour une variable de type Fichier.

5.2.7.2 Liste des variables

L'écran 75 représente la page listant les variables configurées dans l'interface. Chaque ligne indique :

- le nom de la variable avec un lien permettant d'accéder à sa page de détails ;
- le nom de la tâche avec un lien permettant d'accéder à sa page de détails ;
- le type ;
- le code ;
- la date de dernière mise à jour.

Liste des variables				
Nom	Tâche	Type	Code	Mise à jour
Theta Min	CIGRI Run_build_paramsTxt.sh	Entier	THETAMIN	2010-04-23 09:38:55
Theta Max	CIGRI Run_build_paramsTxt.sh	Entier	THETAMAX	2010-07-02 16:05:05
Theta Inc	CIGRI Run_build_paramsTxt.sh	Entier	THETAINC	2010-04-23 09:39:37
Phi Min	CIGRI Run_build_paramsTxt.sh	Entier	PHIMIN	2010-04-23 09:39:54
Phi Max	CIGRI Run_build_paramsTxt.sh	Entier	PHIMAX	2010-07-02 16:05:11
Phi Inc	CIGRI Run_build_paramsTxt.sh	Entier	PHIINC	2010-04-23 09:40:35
Fichier entree forme noyau		Fichier	CFILE1	2010-04-23 17:10:59
Fichier entree cometesimaux		Fichier	CFILE2	2010-04-29 13:40:29
Fichier entree polarisations		Fichier	CFILE6	2010-04-23 17:22:18
Liste des clusters		Texte	CLUSTERS	2010-04-30 10:56:05

Page 1 | Page 2

Écran 75 – Liste des variables.

5.2.7.3 Détails d'une variable

L'écran 76 représente l'onglet principal de la page listant les informations d'une variable. On y retrouve donc :

- l'identifiant ;
- le type ;
- le nom de la variable ;
- le nom de la séquence associée avec un lien permettant d'accéder à sa page de détails ;
- le nom de la tâche associée avec un lien permettant d'accéder à sa page de détails ;
- le code ;
- la valeur ;
- la description ;
- la date de création ;
- la date de dernière mise à jour.

En bas de page, trois liens permettent respectivement de modifier, supprimer et dupliquer la variable courante.

Consert > Simulation Fortran > Détails d'une variable

Variable	
Id	4bd14e8f4a5ca
Type	Entier
Nom	Theta Min
Séquence	Simulation Fortran
Tâche	CIGRI Run build_paramsTxt.sh
Code	THETAMIN
Valeur	0
Description	
Création	2010-04-23 09:38:55
Mise à jour	2010-04-23 09:38:55

[Modifier](#) | [Supprimer](#) | [Dupliquer](#)

Écran 76 – Détails d'une variable.

5.2.7.4 Surcharge d'une variable

L'écran 77 représente le formulaire de surcharge de variable pour un processus.

Consert > SimuTest > Ajouter une variable à un processus

Variable
 Theta Min

Valeur
 0

[Ajouter](#)

Écran 77 – Surcharge d'une variable.

5.2.8 Processus

5.2.8.1 Création d'un processus

Comme le montre l'écran 78, le formulaire de création d'un processus ne contient les champs de base pour le nom et la description. Un champ déroulant permet de sélectionner la séquence que le processus suivra. Enfin, un bouton *Ajouter une variable* permet, via une requête Ajax, d'enrichir le formulaire avec de nouveaux champs pour surcharger une variable.

5.2.8.2 Liste des processus

L'écran 79 représente la page listant les processus configurés dans l'interface. Chaque ligne indique :

- le nom du processus avec un lien permettant d'accéder à sa page de détails ;
- le nom du projet avec un lien permettant d'accéder à sa page de détails ;
- le nom de la séquence avec un lien permettant d'accéder à sa page de détails ;
- l'état ;
- la date de dernière mise à jour ;
- la description.

Consert > Créer un processus

Nom

Description

Séquence
 Simulation Fortran

Ajouter une variable

Créer

Écran 78 – Formulaire de création d'un processus.

Liste des processus

Nom	Projet	Séquence	État	Mise à jour	Description	Actions
SimuTest	Consert	Simulation Fortran	Initialisé	2011-01-28 15:50:13		Planifier

Page 1 | Page 2

Écran 79 – Liste des processus.

5.2.8.3 Détails d'un processus

L'écran 80 représente l'onglet principal de la page listant les informations d'un processus. On y retrouve donc :

- l'identifiant ;
- le nom du projet associé avec un lien permettant d'accéder à sa page de détails ;
- le nom de la séquence associée avec un lien permettant d'accéder à sa page de détails ;
- le nom du processus ;
- l'état ;
- le nom de l'utilisateur associé avec un lien permettant d'accéder à sa page de détails ;
- la description ;
- la date de création ;
- la date de dernière mise à jour.

En bas de page, quatre liens permettent respectivement de modifier, supprimer, dupliquer et planifier le processus courant.

L'écran 81 représente le deuxième onglet sur lequel sont listées les tâches associées au processus courant. Chaque ligne contient les champs suivants :

- le nom de la tâche avec un lien permettant d'accéder à sa page de détails ;
- la date de dernière mise à jour ;
- l'état ;
- un ensemble de liens contrôlant la tâche en fonction de son état.

Selon l'état de chaque tâche, le lien :

- `planifier` met la tâche en attente d'exécution ;
- `démarrer` lance immédiatement l'exécution de la tâche ;
- `arrêter` suspend l'exécution de la tâche lorsque celle-ci est en cours ;

Consert > Simulation Fortran > Détails d'un processus

Processus | Tâches | Variables

Id	4d42d7a5a76bd
Projet	Consert
Séquence	Simulation Fortran
Nom	SimuTest
État	Initialisé
Utilisateur propriétaire	Philippe Bollard
Description	
Création	2011-01-28 15:50:13
Mise à jour	2011-01-28 15:50:13

[Modifier](#) | [Supprimer](#) | [Dupliquer](#) | [Planifier](#) |

Écran 80 – Informations principales d'un processus.

– annuler annule l'exécution de la tâche.

Dans le cas d'une tâche en erreur, une bulle d'information apparaît en précisant le message récupéré par l'interface. Enfin, un lien en bas de page permet d'accéder à page de création d'une variable.

Consert > Simulation Fortran > Détails d'un processus

Processus | Tâches | Variables

Nom	Mise à jour	État	Actions
CIGRI MakeDir Process	2011-01-28 15:50:13	Initialisé	Planifier Démarrer
CIGRI Upload data forme	2011-01-28 15:50:13	Initialisé	Planifier Démarrer
CIGRI Upload data cometesimax	2011-01-28 15:50:13	Initialisé	Planifier Démarrer
CIGRI Upload data polarisation	2011-01-28 15:50:13	Initialisé	Planifier Démarrer
CIGRI Run build_paramsTxt.sh	2011-01-28 15:50:13	Initialisé	Planifier Démarrer
CIGRI Run build_paramsPrg.sh	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI Deploy data on clusters	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI Deploy runjob.sh on clusters	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI Submit job	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
Wait 1h	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI Run script fetch_results.sh	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
VENUS MakeDir Process	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI-VENUS Transfer Results	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
Wait 5min	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
Set Output	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI Clean data	2011-01-28 15:50:14	Initialisé	Planifier Démarrer
CIGRI Clean result	2011-01-28 15:50:14	Initialisé	Planifier Démarrer

[Nouvelle tâche](#)

Écran 81 – Liste des tâches d'un processus.

L'écran 82 représente le troisième onglet sur lequel sont listées les variables associées au processus courant. La page présente d'abord les variables surchargées par le processus puis les variables héritées de la séquence ou des tâches. Chaque ligne contient les champs suivants :

- le nom de la variable avec un lien permettant d'accéder à sa page de détails ;
- le code de la variable ;
- la valeur avec éventuellement un lien vers la page de détails dans le cas d'une variable de type fichier ;
- un ou plusieurs liens pour surcharger une variable héritée ou voir, modifier, supprimer une variable surchargée.

Un lien en bas de page permet d'accéder à page de création d'une variable.

Consert > Simulation Fortran > Détails d'un processus

Nom	Code	Valeur	Actions
Theta Min	THETAMIN	10	Voir Modifier Supprimer
Theta Max	THETAMAX	170	Voir Modifier Supprimer
Variables non-surchargées			
Theta Inc	THETAINC	2	Surcharger
Phi Min	PHIMIN	0	Surcharger
Phi Max	PHIMAX	360	Surcharger
Phi Inc	PHIINC	2	Surcharger
Fichier entree forme noyau	CFILE1	Modele de forme 67P	Surcharger
Fichier entree cometesimaux	CFILE2	Modele structure interne homogene	Surcharger
Fichier entree polarisations	CFILE6	Données polarisation	Surcharger
Liste des clusters	CLUSTERS	fostino r2d2	Surcharger

Écran 82 – Liste des variables d'un processus.

5.2.9 Utilisateur

5.2.9.1 Création d'un utilisateur

Comme le montre l'écran 83, le formulaire de création d'un utilisateur se compose des champs suivants :

- le prénom ;
- le nom ;
- l'adresse de courriel ;
- le nom d'utilisateur ;
- le mot de passe ;
- une case à cocher pour activer ou non le rôle d'administrateur.

Créer un utilisateur

Prénom

Nom

Courriel

Nom d'utilisateur

Mot de passe

Est administrateur

Créer

Écran 83 – Formulaire de création d'un utilisateur.

5.2.9.2 Liste des utilisateurs

L'écran 84 représente la page listant les utilisateurs configurés dans l'interface. Chaque ligne indique :

- le nom d'utilisateur avec un lien permettant d'accéder à sa page de détails ;

- le prénom ;
- le nom ;
- la date de création.

En bas de page, deux liens amènent respectivement sur les pages de création et de recherche d'utilisateur.

Liste des utilisateurs

Nom d'utilisateur	Prénom	Nom	Création
bollardp	Philippe	Bollard	2010-05-17 15:18:57
chaillos	Stéphane	Chaillol	2010-06-15 15:37:56
usertest	usertest	usertest	2010-07-29 10:26:13

[Nouveau](#) | [Rechercher](#)
Page 1 | [Page 2](#)

Écran 84 – Liste des utilisateurs.

5.2.9.3 Détails d'un utilisateur

L'écran 85 représente l'onglet principal de la page listant les informations d'un utilisateur. On y retrouve donc :

- l'identifiant ;
- le nom d'utilisateur ;
- le prénom ;
- le nom ;
- l'adresse de courriel ;
- le statut ou non d'administrateur ;
- la date de création ;
- la date de dernière mise à jour.

En bas de page, trois liens permettent respectivement de modifier, supprimer et dupliquer l'utilisateur courant.

Détails d'un utilisateur

Utilisateur		Projets	Processus	Hôtes
Id	4bf1424198d1d			
Nom d'utilisateur	bollardp			
Prénom	Philippe			
Nom	Bollard			
Courriel	philippe.bollard@obs.ujf-grenoble.fr			
Est administrateur	Oui			
Création	2010-05-17 15:18:57			
Mise à jour	2010-07-27 15:23:59			

[Modifier](#) | [Supprimer](#) | [Dupliquer](#)

Écran 85 – Informations principales sur un utilisateur.

L'écran 86 représente le deuxième onglet sur lequel sont listés les projets associés à l'utilisateur courant.

L'écran 87 représente le troisième onglet sur lequel sont listés les processus associés à l'utilisateur courant. La structure de la page est identique à celle de la liste des processus.

Détails d'un utilisateur

Utilisateur Projets Processus Hôtes

Nom	Description	Actions
Consert	Simulation de l'expérience CONCERT	Retirer

Ajouter

Écran 86 – Liste des projets d'un utilisateur.

Détails d'un utilisateur

Utilisateur Projets Processus Hôtes

Nom	Projet	Séquence	État	Mise à jour	Description	Actions
SimuTest	Consert	Simulation Fortran	Initialisé	2011-01-28 15:50:13		Planifier

Écran 87 – Liste des processus d'un utilisateur.

L'écran 88 représente le quatrième onglet sur lequel sont listés les hôtes associés à l'utilisateur courant.

Détails d'un utilisateur

Utilisateur Projets Processus Hôtes

Nom	Type	Description	Actions
Frontale CiGri	Grille	Frontale de la grille de calcul intensif CIMENT	Modifier Retirer
Venus	Noeud		Modifier Retirer

Ajouter

Écran 88 – Liste des hôtes d'un utilisateur.

5.3 Mise en œuvre de la simulation CONCERT

Afin de mettre en place la simulation Consert sur cette plateforme, il a été nécessaire d'adapter au préalable les différents scripts Bash. La généricité a ainsi été améliorée en vue d'une réutilisation de ce travail par d'autres projets.

5.3.1 Amélioration des scripts Cigri

En étudiant la chaîne de simulation existante, on s'aperçoit que certaines étapes sont directement liées à l'outil CiGri de gestion de la grille CIMENT. Celles-ci ont donc été isolées et généralisées au sein de scripts spécifiques. Ces derniers feront partie de l'installation de base de la plateforme et seront ainsi communs à tous les projets qui l'utiliseront.

5.3.1.1 Génération de la liste de grappes

Le premier script permet de lister les grappes réellement actives et accessibles à un utilisateur de la grille. Pour cela, on interroge la base de données de l'intergiciel CiGri. Un compte a été

créé spécifiquement pour notre plateforme.

Via une requête SQL, le script obtient la liste des grappes déclarées. Cette liste est ensuite filtrée pour éliminer les grappes volontairement désactivées par l'administrateur de la plateforme. Enfin, le script contacte chaque grappe via SSH. Si la connexion échoue – à cause d'un délai de réponse trop long ou de refus d'authentification –, la grappe est écartée. Au final, la liste des grappes restantes est écrite dans un fichier `clusters.txt`.

Le code source de ce script est disponible en annexe.

5.3.1.2 Génération du fichier JDL

Ce script est utilisé pour produire un fichier JDL permettant de configurer une campagne CiGri sur la grille CIMENT. Il prend plusieurs paramètres en ligne de commande :

- le chemin du fichier JDL à produire ;
- le nom de la campagne ;
- le chemin du dossier d'exécution ;
- le chemin du fichier à exécuter ;
- la durée maximale d'exécution.

Le script commence par écrire le bloc `DEFAULT` définissant quelques valeurs globales du fichier JDL. Ensuite, il charge la liste des grappes à partir du fichier `clusters.txt` établi précédemment. Chaque grappe est contactée via SSH. Si la connexion réussit, le script génère le bloc correspondant pour le fichier JDL.

Le code source de ce script est disponible en annexe.

5.3.1.3 Déploiement sur les grappes

Ce script est utilisé pour déployer un dossier ou un fichier entre le serveur frontal de la grille et ses grappes. Il prend plusieurs paramètres en ligne de commande :

- le chemin source ;
- le chemin cible.

Le script commence par charger la liste des grappes à partir du fichier `clusters.txt` établi précédemment. Chaque grappe est contactée via SSH. Si la connexion réussit, le script effectue une copie récursive du chemin source vers le chemin cible.

Le code source de ce script est disponible en annexe.

5.3.1.4 Nettoyage des fichiers

Ce script est utilisé pour nettoyer le serveur frontal de la grille et ses grappes en fin de processus. Il prend pour seul paramètre de ligne de commande le chemin à nettoyer.

Le script commence par charger la liste des grappes à partir du fichier `clusters.txt` établi précédemment. Chaque grappe est contactée via SSH. Si la connexion réussit, le script effectue une suppression récursive du chemin passé en paramètre. Enfin, le script effectue la même suppression récursive du chemin situé sur le serveur frontal de la grille.

Le code source de ce script est disponible en annexe.

5.3.2 Amélioration des scripts Concert

5.3.2.1 Génération du fichier params.prg

Ce script est utilisé pour générer le fichier `params.prg` permettant de configurer l'outil de simulation. Les différentes valeurs sont passées en paramètres de ligne de commande. Le script génère le fichier à partir d'un modèle. Il effectue ensuite un remplacement des champs par les valeurs correspondantes.

Le code source de ce script est disponible en annexe.

5.3.2.2 Génération du fichier params.txt

Ce script est utilisé pour générer le fichier `params.txt` nécessaire au lancement de la campagne de calcul. Il prend plusieurs paramètres en ligne de commande :

- le chemin de l'exécutable ;
- la valeur minimale de theta ;
- la valeur maximale de theta ;
- le pas d'incrémentation pour theta ;
- la valeur minimale de phi ;
- la valeur maximale de phi ;
- le pas d'incrémentation pour phi.

L'algorithme contient deux boucles imbriquées permettant d'obtenir toutes les combinaisons en faisant varier theta et phi entre leurs valeurs minimales et maximales respectives. Le fichier produit contient alors une succession de lignes respectant la forme suivante :

```
outputTHETA_PHI SOFT params.prg THETAMIN THETAMAX PHIMIN PHIMAX
```

Le code source de ce script est disponible en annexe.

5.3.2.3 Compilation et déploiement de l'outil de simulation sur les grappes

Ce script est utilisé pour compiler le code source de la simulation et le déployer sur les grappes de la grille CIMENT. Il prend plusieurs paramètres en ligne de commande :

- le chemin vers le fichier contenant la liste des grappes ;
- le chemin source ;
- le chemin cible ;
- le nom du fichier source ;
- le nom du binaire.

Le script ouvre le fichier `clusters.txt` pour récupérer la liste des grappes accessibles. Pour chacune d'entre elles, il tente de se connecter via SSH. S'il y parvient, il copie récursivement le chemin source – de la frontale – vers le chemin cible – sur la grappe – puis exécute la ligne de compilation spécifique à la grappe.

Le code source de ce script est disponible en annexe.

5.3.2.4 Lancement d'une tâche sur une grappe

Ce script est utilisé pour lancer une tâche de calcul sur une grappe. Il prend plusieurs paramètres en ligne de commande :

- le chemin de sortie ;
- le chemin de l'exécutable ;
- la valeur minimale de theta ;
- la valeur maximale de theta ;
- la valeur minimale de phi ;
- la valeur maximale de phi.

Le script formate ces valeurs puis lance l'exécutable en lui passant l'ensemble des paramètres via l'entrée standard.

Le code source de ce script est disponible en annexe.

5.3.2.5 Récupération et concaténation des résultats

Ce script est utilisé pour rassembler les résultats d'une campagne de calcul. Il prend plusieurs paramètres en ligne de commande :

- le chemin vers un fichier contenant l'identifiant de la campagne ;
- le nom du fichier contenant les résultats d'un job ;
- le chemin vers le fichier final.

Lorsqu'une campagne est lancée via l'interface, un fichier est créé pour stocker l'identifiant fourni par CiGri. Ce fichier est lu par le script pour récupérer cette valeur. Il vérifie ensuite que les fichiers de résultats existent avant de les extraire. Il concatène les fichiers contenant les résultats des jobs puis compresse le fichier final. Enfin, il nettoie les fichiers sources.

Le code source de ce script est disponible en annexe.

5.3.3 Paramétrage de l'interface

Il faut en premier lieu initialiser quelques éléments au sein de l'interface. Ensuite, il faut entrer la nouvelle chaîne de simulation qui a été définie sur la base de la chaîne d'origine. La plupart des scripts ont été repris et adaptés. Ainsi, ceux-ci sont plus génériques et offrent plus de paramètres afin que l'utilisateur les configure depuis l'interface.

La chaîne est découpée en deux sous-chaînes :

- une première pour déployer l'environnement de calcul sur la grille ;
- une seconde pour paramétrer et exécuter le calcul proprement dit.

5.3.3.1 Initialisation de l'interface

Pour initialiser la plateforme, il faut créer :

- un projet « Consert » ;
- un hôte « Plateforme » de type `nœud` pour définir le serveur exécutant l'application ;
- un hôte « Frontale CiGri » de type `grille` pour définir le point d'entrée de la grille de calcul CIMENT ;
- un hôte « Venus » de type `nœud` pour définir le serveur de stockage des fichiers ;
- les comptes des utilisateurs ;
- les fichiers.

Ensuite, au sein du projet, il faut associer les hôtes au projet puis les utilisateurs aux hôtes.

5.3.3.2 Sous-chaîne de déploiement de l'environnement de calcul

Cette séquence agit sur la frontale de la grille de calcul. Elle ne doit généralement être exécutée qu'une seule fois à l'initialisation d'un projet par un utilisateur. Elle comporte les tâches suivantes :

1. une `commande` pour créer l'arborescence adéquate ;
2. un `envoi` pour copier le fichier source du logiciel de simulation dans le dossier précédemment créé ;
3. un `envoi` pour copier le script de compilation et déploiement de l'outil ;
4. un `envoi` pour copier le script de génération des paramètres d'exécution d'un job ;
5. un `envoi` pour copier le script de génération des paramètres spécifiques à l'outil de simulation ;
6. un `envoi` pour copier le script de lancement d'un job ;
7. un `envoi` pour copier le script de concaténation et de récupération des résultats ;
8. une `commande` pour rendre ces scripts exécutables ;
9. une `commande` pour lancer la compilation et le déploiement de l'outil de simulation sur les différentes grappes de la grille de calcul.

5.3.3.3 Sous-chaîne de calcul de la simulation

Cette séquence doit être exécutée pour chaque « simulation ». Elle comporte les tâches suivantes :

1. une `commande` pour créer l'arborescence adéquate ;
2. un `envoi` pour copier le fichier du modèle de forme de comète ;
3. un `envoi` pour copier le fichier du modèle de composition interne de la comète ;
4. un `envoi` pour copier le fichier des données de polarisation ;
5. une `commande` pour exécuter le script de génération des paramètres de lancement des jobs ;
6. une `commande` pour exécuter le script de génération des paramètres spécifiques à l'outil de simulation ;
7. une `commande` pour déployer ces données sur les grappes de la grille de calcul ;
8. un `job Cigri` pour lancer l'exécution de la campagne de calcul ;
9. une `attente` d'une heure pour temporiser le déroulement de la séquence et ainsi laisser le temps aux automates de la grille de rassembler les résultats dans le dossier de l'utilisateur ;
10. une `commande` pour exécuter le script de concaténation et de récupération des résultats ;
11. une `commande` pour créer l'arborescence adéquate sur le serveur de stockage ;
12. un `transfert` pour transférer les résultats de la grille vers le serveur de stockage ;
13. une `attente` de cinq minutes pour s'assurer que les connexions sont bien terminées ;
14. une `fin` permettant d'enregistrer le chemin final des résultats ;
15. une `commande` pour lancer un nettoyage des données sources sur la grille ;
16. une `commande` pour lancer un nettoyage des données générées sur la grille.

5.4 Bilan

L'application principale de la nouvelle plateforme s'utilise via une interface Web. Celle-ci a été réalisée de façon à être la plus claire et la plus intuitive possible même si elle reste perfectible. Les différents écrans présentés permettent aux utilisateurs de gérer l'ensemble des informations en créant des éléments, en les listant, en les affichant, en les modifiant, en les supprimant, etc.

Lorsqu'une chaîne de simulation est initialisée, c'est-à-dire lorsqu'un projet est créé avec ses séquences, leurs tâches et leurs variables respectives, il est possible d'en créer une ou plusieurs « instances » sous la forme de « processus ».

Dans le cas de la simulation CONSERT, le paramétrage de la chaîne de simulation a nécessité la modification des scripts Bash existants. Ceux-ci sont désormais plus génériques mais restent adaptés à l'intergiciel de grille CiGri. La chaîne de simulation est maintenant découpée en deux sous-chaînes :

- une première sous-chaîne déploie l'environnement de calcul sur les grappes de la grille ;
- une seconde sous-chaîne définit la succession d'étapes nécessaires à l'exécution du calcul de la simulation.

En conclusion, nous revenons sur les différents aspects du stage en établissant une synthèse des objectifs et du travail effectué puis en proposant quelques perspectives d'évolution. Enfin, un bilan personnel termine ce mémoire.

Structures de données

Dans la poursuite de l'étude préliminaire menée lors de l'épreuve TEST, la première phase du stage fut centrée sur les structures de données avec pour but de valider certaines décisions prises dans le développement de l'outil de visualisation. La simulation de l'expérience CONSERT utilisait différents fichiers de données. Leurs structures n'étaient pas homogènes et leur usage au sein de l'outil de visualisation n'était pas optimal. Il était demandé d'étudier ces fichiers puis de proposer de nouvelles structures de données basées sur XML et de définir des méthodes de traitement permettant un accès rapide et précis à certaines données.

Des structures XML ont été assez facilement créées pour contenir les données. Elle peuvent être validées par les DTD ou les XSD correspondants. La partie la plus longue concerna le choix de la méthode la plus adaptée pour lire les informations. L'étude menée s'est focalisée sur les données les plus importantes : les résultats de simulation. Différentes approches ont été proposées à travers des prototypes successifs en implémentant des analyseurs syntaxiques basés sur DOM et SAX ou en transformant la structure de données XML en une base de données SQLite.

La méthode qui semble réunir à la fois un faible poids de fichiers, une faible empreinte mémoire et une assez bonne vitesse d'exécution est caractérisée par l'usage :

- d'un fichier XML décrivant les rayons simulés ;
- d'un fichier XML décrivant des cubes de maillage de l'espace de calcul en cubes avec la liste des identifiants des rayons les traversant ;
- de SAX.

Suite à cette étude, il est encore nécessaire d'affiner les nouvelles structures établies avant leur réelle utilisation par le projet. De plus, il convient de noter que leur emploi requiert une adaptation d'au moins une partie des logiciels et scripts existants pour la plateforme. Cela dépasse le cadre du stage et reste donc à effectuer. Plusieurs possibilités sont envisageables :

- l'implémentation du nouveau format en plus de l'existant ;
- la création et l'utilisation de scripts de conversion des données d'un format à l'autre ;

- la réécriture des algorithmes de génération ou de chargement des données.

Dans le cas de l’outil de visualisation, l’implémentation du nouveau format ne devrait pas poser de problème. En revanche, l’outil de simulation doit être adapté pour produire directement les données au format XML. Dans tous les cas, l’utilisation de convertisseurs externes est une alternative valable et moins envahissante.

Plateforme

Le cœur même du stage est directement lié à la plateforme de simulation de l’expérience CONSERT. Celle-ci était jusque-là composée de deux logiciels :

- le logiciel de simulation écrit en Fortran et fonctionnant sur la grille de calcul CIMENT ;
- le logiciel d’interprétation des résultats écrit en Delphi et fonctionnant sur le poste de l’utilisateur.

Pour lancer une simulation, il était nécessaire de paramétrer manuellement un certain nombre de scripts. Cela rendait la plateforme complexe, peu intuitive et inadaptée à une utilisation intensive. L’objectif principal du stage fut donc de compléter cette plateforme en concevant et en réalisant un outil qui automatise la soumission des simulations sur la grille de calcul. Cet outil devait permettre également de constituer un historique des simulations effectuées.

L’outil a été réalisé sous la forme d’une application Web basée sur le cadriciel PHP Zend Framework. Il communique en SSH avec les différents hôtes configurés pour interagir avec la grille en ligne de commande ou télécharger des fichiers en SFTP. L’application est multi-utilisateurs et multi-projets. Un projet peut contenir plusieurs séquences de tâches. Ces-dernières sont paramétrables par des variables personnalisables par l’utilisateur. Différents types de tâches sont prévus pour exécuter une commande, échanger des fichiers, lancer un job CiGri sur une grille de calcul, etc. Une séquence n’est qu’un « modèle » qu’il faut instancier en « processus » afin de pouvoir exécuter une succession de tâches paramétrées.

L’utilisation de ce nouvel outil permet aux utilisateurs de gagner beaucoup de temps. En effet, il est désormais facile et rapide de lancer une série de simulations en faisant varier quelques paramètres via l’interface. Il est également aisé de retrouver une simulation et d’obtenir des informations sur celle-ci puisque tout est stocké automatiquement en base de données, ce qui n’était pas le cas avant,

La plateforme ayant été conçue en vue d’une utilisation générique, son utilisation n’est pas limitée à la seule expérience CONSERT. D’autres projets et d’autres équipes du LPG sont désormais intéressés par cet outil. Dans le cas de du projet MARSIS¹, la chaîne de simulation existante est composée de différents programmes dont des scripts IDL. Ceux-ci ont la particularité d’être exécutés interactivement par l’utilisateur via un environnement de calcul scientifique qui n’est disponible que sur certains serveurs. Pour adapter la chaîne à la plateforme, il faut donc soit :

- réécrire les scripts IDL dans un autre langage tel que C++ ou Python afin qu’ils soient facilement déployables sur toute grappe de calcul ;
- s’assurer de la disponibilité de l’environnement IDL sur les grappes et l’interfacer avec la plateforme en rendant les scripts non-interactifs.

Par ailleurs, seule la grille CIMENT et son intergiciel CiGri ont été paramétrés au sein de la plateforme. L’architecture a été volontairement conçue de manière générique pour interfacer d’autres intergiciels comme OAR.

1. Le LPG a développé le programme de simulation des échos MARSIS. Ce radar basse fréquence permet d’imager la structure interne du sous-sol martien jusqu’à des profondeurs kilométriques, ainsi que celle des calottes polaires. [1]

L'architecture prévue pour la plateforme prévoyait une version de l'application serveur déportée sur un poste client pour permettre aux utilisateurs itinérants de préparer leurs simulations en mode « déconnecté ». Cette application découle directement de la version serveur puisqu'elle reprend la même pile logicielle (serveur Apache, serveur MySQL, PHP...). La seule différence réside dans le paramétrage initial de l'interface pour activer l'automate de synchronisation. Faute de temps, cet aspect n'a pas été totalement terminé et testé pour être intégré dans la version livrée.

Enfin, l'application serveur doit à terme s'intégrer à l'outil d'interprétation des résultats. Pour cela, il est nécessaire de réaliser une interface native en Delphi qui utilisera la couche de communication REST mise en place. D'un commun accord, cette réalisation est laissée à la charge de l'équipe de développement de l'outil concerné. Une solution alternative plus rapide à mettre en œuvre est également proposée : il est possible d'intégrer l'interface Web de l'application serveur au travers d'un navigateur embarqué au sein de l'outil d'interprétation.

Chaîne de simulation CONSERT

La chaîne de simulation de l'expérience CONSERT est désormais scindée en deux sous-chaînes : la première permet d'initialiser l'environnement de calcul en déployant et en compilant l'outil de simulation sur les grappes de la grille CIMENT. La seconde chaîne décrit réellement les étapes nécessaires au lancement de la simulation. Cette dernière sous-chaîne est en fait un modèle qui doit être dupliqué et paramétré pour chaque instance de calcul souhaitée.

La nouvelle chaîne de simulation CONSERT reste toujours adaptée aux seuls outils actuels. Le programme de simulation est en cours de réécriture en C++. Il est fort probable que la chaîne nécessite d'être modifiée de nouveau pour prendre en compte l'ajout, le retrait ou la modification d'étapes ou de paramètres. Il faudra alors adapter les scripts puis, via l'interface reconfigurer les tâches et les variables de la chaîne.

Bilan personnel

Ce stage fut pour moi une très bonne expérience. En effet, grâce à ce travail, j'ai pu étudier et me former au fonctionnement d'une grille de calcul intensif et découvrir quelques intergiciels existants même si nous nous sommes focalisés sur CiGri. Ce domaine est en pleine évolution et il y a encore beaucoup de choses à faire pour qu'un véritable standard émerge et permette une interconnexion entre les différentes grilles.

Ce fut aussi pour moi l'occasion de découvrir de plus près le milieu de la recherche scientifique. Cela m'a donné l'envie de poursuivre dans cette voie. Suite à cette mission, j'ai été recruté en tant qu'ingénieur d'études contractuel par une autre équipe du même laboratoire. Je poursuis l'utilisation de Zend Framework que j'ai appris à connaître lors de ce stage. J'espère par la suite être en mesure de valoriser mon expérience acquise récemment sur les grilles de calcul en postulant aux concours CNRS.

Annexes

A Diagramme de Gantt

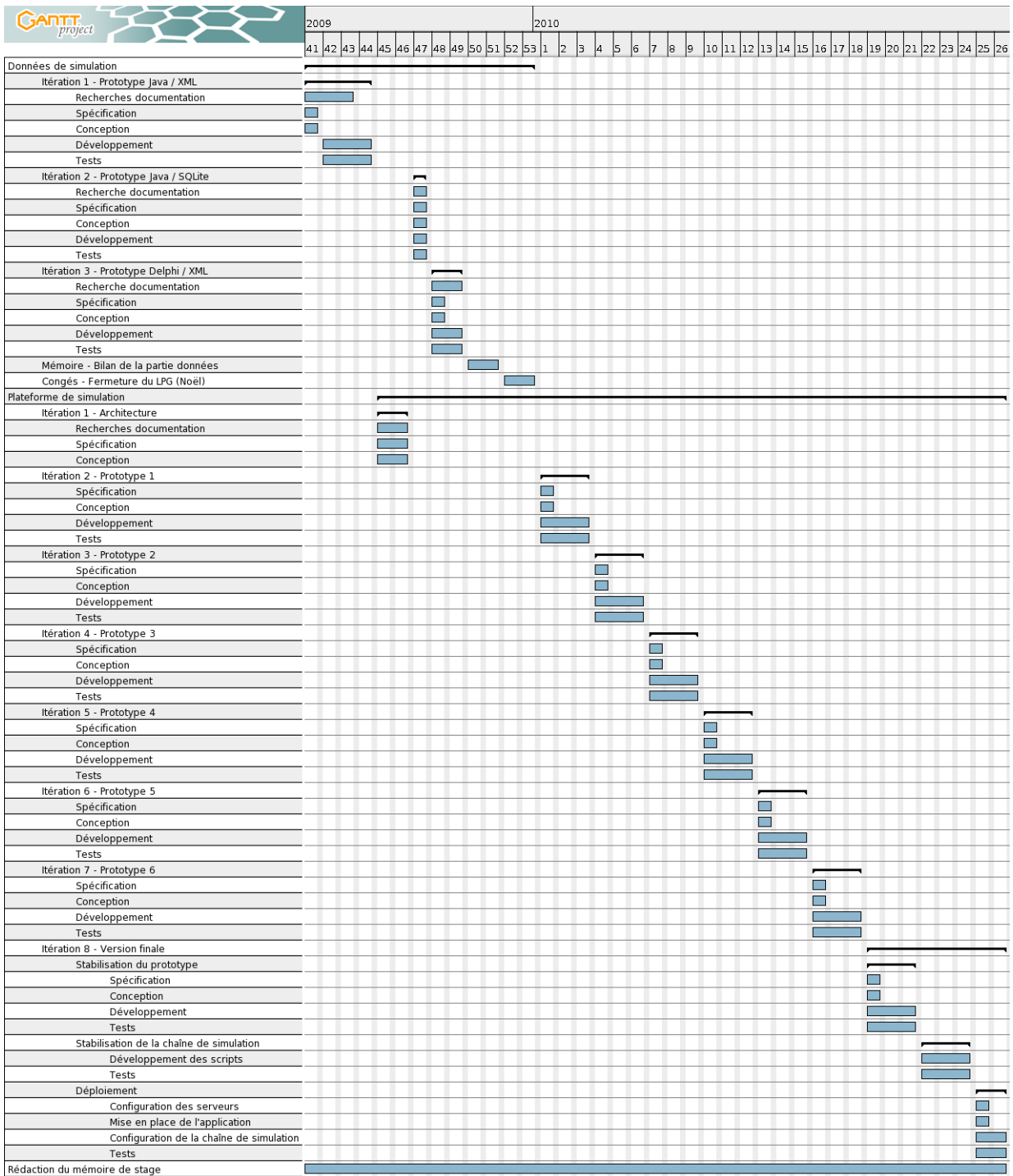


Figure 89 – Diagramme de GANTT.

B Scripts Bash

B.1 Scripts Cigri

B.1.1 Génération de la liste des clusters

```
#!/bin/bash
#####
#####          C O N S E R T          #####
4  #####
  ## Generation de la liste des clusters accessibles
  ##
  #####
  ## Auteurs : Philippe BOLLARD
  9  ## Date   : 28-07-2010
  ## Version : 1.0
  #####

#####
14 # CONFIG SQL
  #####
  SQLUSER=cigri_ro
  SQLPASS=read
  19  SQLBASE=cigri2

  ## Parametres
  #####

  PATHOUT=~/.clusters.txt
24

  ## On liste tous les clusters
  QUERY="select clusterName from clusters;"

  ## On genere la requete
  29  #####
  echo $QUERY > clusters.in

  ## On se connecte a la base et on execute la requete
  #####
  34  mysql -b -s -u$SQLUSER -p$SQLPASS $SQLBASE < clusters.in > clusters.out
  rm clusters.in

  CLUSTERS=`cat clusters.out`
  39  rm clusters.out

  ## On efface l'eventuel ancien fichier
  #####
  rm -f $PATHOUT

44  ## On recupere l'eventuelle liste blacklistee
  #####
  if [[ -e blackclusters.txt ]]
  then

  49  BLACKCLUSTERS=`cat blackclusters.txt`
    for BLACK in $BLACKCLUSTERS;
    do
      CLUSTERS=${CLUSTERS/$BLACK}
    done
  54 fi

  ## On teste chaque cluster
  #####
  59  for HOST in $CLUSTERS;
  do

    ##On ping le cluster
    ping -q -c 1 $HOST &> ping.out
    pingCount=$(cat ping.out | grep 'received' | awk -F',' '{ print $2 }' | awk '{ print $1 }')
```

```

69  ## Test de la connexion SSH
    ssh -q -o NumberOfPasswordPrompts=0 -o ConnectTimeout=5 -o StrictHostKeyChecking=no
        $USER@$HOST touch .testHost

    if [[ $? -eq 0 ]]
    then
74      ## On peut utiliser ce cluster
        echo $HOST >> $PATHOUT
    fi
    fi
79 done

## C'est fini
#####
exit 0;

```

B.1.2 Génération du JDL

```

2  #!/bin/bash
   ##=====
   #####          C O N S E R T          #####
   ##=====
   ## Deploiement sur la grille des fichiers
   ## utilises par une simulation
7  ##
   ##=====
   ## Auteurs : Philippe BOLLARD, Stephane CHAILLOL
   ## Date    : 23-04-2010
   ## Version : 1.0
12 #####

   ## On controle le nombre de parametres
   if [[ $# -ne 5 ]]
   then
17   echo "Usage: $0 <jdlfilename> <jobname> <execDir> <execFile> <walltime>"
     exit 1
   fi

   ## Parametres
22 #####

   ## Liste des clusters
   CLUSTERS=`cat ~/clusters.txt `

27 ##Nom du fichier jdl
   JDLFILENAME=$1

   ## Nom de la campagne
   JOBNAME=$2
32

   ## Chemin d'execution
   EXECDIR=$3

   ## Fichier a executer
37 EXECFILE=$4

   ## Temps d'execution
   WALLTIME=$5

42 ## On ecrit le debut du JDL
   #####
   echo 'DEFAULT {' > $JDLFILENAME
   echo " name=$JOBNAME;" >> $JDLFILENAME
   echo ' paramFile=params.txt;' >> $JDLFILENAME
47 echo '}' >> $JDLFILENAME
   echo '' >> $JDLFILENAME

   ## On ecrit les blocs pour les clusters disponibles
   #####

```

```

52 for HOST in $CLUSTERS;
do
    ## Test de la connexion SSH
    HOSTHOMEDIR=$(ssh -q -o NumberOfPasswordPrompts=0 -o ConnectTimeout=2 -o StrictHostKeyChecking=
        no $USER@$HOST pwd)
57 if [[ $? -eq 0 ]]
    then
        echo "$HOST {" >> $JDLFILENAME
        echo " walltime=$WALLTIME;" >> $JDLFILENAME
62 echo " execDir=$HOSTHOMEDIR/$EXECDIR;" >> $JDLFILENAME
        echo " execFile=$HOSTHOMEDIR/$EXECFILE;" >> $JDLFILENAME
        echo '}' >> $JDLFILENAME
        echo '' >> $JDLFILENAME
    fi
67 done

## C'est fini
#####
exit 0;

```

B.1.3 Déploiement sur les clusters

```

#!/bin/bash
#####
4 #####          C O N S E R T          #####
#####
## Deploiement sur la grille du script
## de lancement d'une simulation
##
#####
9 ## Auteurs : Philippe BOLLARD, Stephane CHAILLOL
## Date      : 02-08-2010
## Version   : 1.1
#####
14 ## On controle le nombre de parametres
if [[ $# -ne 2 ]]
then
    echo "Usage: $0 <pathIn> <pathOut>"
    exit 1
19 fi

## Parametres
#####
24 ## Liste des clusters
CLUSTERS=`cat ~/clusters.txt`

## Chemin source
PATHIN=$1
29 ## Chemin destination
PATHOUT=$2

## Deploiement sur les clusters
34 #####
for HOST in $CLUSTERS;
do
    ## Test de la connexion SSH
39 ssh -q -o NumberOfPasswordPrompts=0 -o ConnectTimeout=2 -o StrictHostKeyChecking=no $USER@$HOST
        touch .testHost
    if [[ $? -eq 0 ]]
    then
        PATHDIR=`dirname "$PATHOUT"`
44
        ## On cree l'arborescence de destination
        ssh $USER@$HOST mkdir -p $PATHDIR

```



```

49     ## Copie des fichier
        scp -qr $PATHIN $USER@$HOST:$PATHOUT
    fi
done
54 ## C'est fini
#####
exit 0;

```

B.1.4 Nettoyage

```

#!/bin/bash
#####
4  #####          C O N S E R T          #####
#####
## Nettoyage des clusters
## utilise par une simulation
##
##-----
9  ## Auteurs : Philippe BOLLARD
## Date      : 02-07-2010
## Version   : 1.0
#####
14 ## On controle le nombre de parametres
if [[ $# -ne 1 ]]
then
    echo "Usage: $0 <path>"
    exit 1
19 fi

## Parametres
#####
24 ## Liste des clusters
CLUSTERS=`cat ~/clusters.txt`

## Chemin
PATHOUT=$1
29 ## Nettoyage des clusters
#####
for HOST in $CLUSTERS;
do
34     ## Test de la connexion SSH
    ssh -q -o NumberOfPasswordPrompts=0 -o ConnectTimeout=2 -o StrictHostKeyChecking=no $USER@$HOST
        touch .testHost
    if [[ $? -eq 0 ]]
    then
39         ## Nettoyage
            ssh $USER@$HOST rm -rf $PATHOUT

    fi
44 done

## Nettoyage de la frontale
#####
49 rm -rf $PATHOUT

## C'est fini
#####
exit 0;

```

B.2 Scripts Consert

B.2.1 Génération du fichier params.prg

```
#!/bin/bash
2 ##=====
#####          C O N S E R T          #####
#####
## Genere le fichier de parametrage des jobs
##
7 ##=====
## Auteurs : Philippe BOLLARD, Stephane CHAILLOL
## Date    : 20-04-2010
## Version : 1.0
#####
12 ## On controle le nombre de parametres
#nbParams=41
nbParams=41
if [ $# -ne $nbParams ]
17 then
    echo "Usage: $0 <PATHOUT> <CFILE1> <CFILE2> <CFILE3> <CFILE4> <CFILE5> <CFILE6> <CFILE7> <
    fRMAX1> <fRMAX2> <fRMAX3> <RMIN> <RMAX> <fRX0> <fRY0> <fRZ0> <SH1> <SH2> <SH3> <fPLAND> <
    fLLAND> <XLAN_X> <XLAN_Y> <XLAN_Z> <YLAN_X> <YLAN_Y> <YLAN_Z> <ZLAN_X> <ZLAN_Y> <ZLAN_Z> <
    SCALE> <fPstep> <fpmoy> <fpvide> <fLWAVE> <IREFMAX> <NBSURF> <fPAS> <SMAX> <SMIN> <SEUIL>"
    exit 1
fi
22 ## Modele du fichier param.prg
#####
(
cat <<' EOF'
27 <----- DESCRIPTIF -----> = <----- VALEUR -----><-- DIM --->
* CFILE1   : F.I des parametres de la forme du noyau           = %1%
* CFILE2   : F.I des parametres des cometesi.                 = %2%
* CFILE3   : F.O des mesures de phases & d'att.               = %3%
* CFILE4   : F.O des chemins optiques courbes                 = %4%
* CFILE5   : F.O des temps de propagation                     = %5%
32 * CFILE6   : F.I du diagramme de polarisation Lander         = %6%
* CFILE7   : F.O fichier log de suivi                         = %7%

* fRMAX1   : Rayon du le cercle limite englobant le noyau= %8% m
* fRMAX2   : Rayon du 2ieme cercle limite                     = %9% m
37 * fRMAX3   : Rayon de l'orbite                               = %10% m
* RMIN     : Rayon de la sphere min                           = %11% m
* RMAX     : Rayon de la sphere max                           = %12% m
* fRX0     : Coordonne X du centre de l'orbite                = %13% m
* fRY0     : Coordonne Y du centre de l'orbite                = %14% m
42 * fRZ0     : Coordonne Z du centre de l'orbite                = %15% m
* SH1      : Pas d'integration pour R<fRMAX1                  = %16% m
* SH2      : Pas d'integration pour fRMAX1<R<fRMAX2          = %17% m
* SH3      : Pas d'integration final pour fRMAX2<R            = %18% m
* fPLAND   : Latitude de l'atterrisseur                       = %19% degres
47 * fLLAND   : Longitude de l'atterrisseur                     = %20% degres
* XLAN_X   : coordonnee X du X LANDER                         = %21% -
* XLAN_Y   : coordonnee Y du X LANDER                         = %22% -
* XLAN_Z   : coordonnee Z du X LANDER                         = %23% -
* YLAN_X   : coordonnee X du Y LANDER                         = %24% -
52 * YLAN_Y   : coordonnee Y du Y LANDER                         = %25% -
* YLAN_Z   : coordonnee Z du Y LANDER                         = %26% -
* ZLAN_X   : coordonnee X du Z LANDER                         = %27% -
* ZLAN_Y   : coordonnee Y du Z LANDER                         = %28% -
* ZLAN_Z   : coordonnee Z du Z LANDER                         = %29% -
57 * SCALE    : Pas de discretisation en                       = %30% degres-1
* fPstep   : Pas de discretisation de la polarisation         = %31% degres
* fpmoy    : Permittivite moyenne du milieu cometaire        = %32% -
* fpvide   : permittivite du vide                             = %33% -
* fLWAVE   : Longueur d'onde dans le vide de l'onde          = %34% m
62 * IREFMAX  : Nombre de reflexions maximales                 = %35% -
* NBSURF   : Nombre de point max a la surface                 = %36% -
* fPAS     : Ouverture du rayon pour le calcul de la puiss= %37% degres
* SMAX     : Surf. max. de divergence                          = %38% m2
* SMIN     : Surf. min. de divergence                          = %39% m2
67 * SEUIL    : Limite de puissance des donnees exploitables= %40% dB
EOF
```

```

) > paramPrg.tmp
#####
72 index=1
   indexLoop=1
   for param in "$@"
   do
77     if [[ $indexLoop -gt 1 ]]
       then
         variable=`echo "$param" | sed -e :z -e 's/^\.{1,21}\$/& /;tz'`;
         #sed -e "s/%$index%/$variable/g" paramPrg.tmp > paramPrg.tmp2 && mv -f paramPrg.tmp2 paramPrg
           .tmp;
         perl -pi -e "s/%$index%/$variable/" paramPrg.tmp
         let "index += 1";
82     fi
       let "indexLoop += 1";
   done
mv paramPrg.tmp $1
87 exit 0;

```

B.2.2 Génération du fichier params.txt

```

#!/bin/bash
#####
3 #####          C O N S E R T          #####
#####
## Genere le fichier de parametrage des jobs
##
#####
8 ## Auteurs : Philippe BOLLARD, Stephane CHAILLOL
## Date      : 19-04-2010
## Version   : 1.0
#####
13 ## On definit les bornes
thetaMaxLimit=180
phiMaxLimit=360

## On controle le nombre de parametres
18 if [ $# -ne 7 ]
then
  echo "Usage: $0 <soft> <thetaMin> <thetaMax> <thetaInc> <phiMin> <phiMax> <phiInc>"
  exit 1
fi
23 ## On recupere les parametres
soft=$1
thetaMin=$2
thetaMax=$3
28 thetaInc=$4
phiMin=$5
phiMax=$6
phiInc=$7
params=params.prg
33 ## On initialise la variable theta
theta=$thetaMin;

## On boucle jusqu'a ce que theta atteigne la borne maximale
38 while [ $theta -lt $thetaMax ]
do

  ## On garde la valeur actuelle de theta dans la variable s
  s=$((theta));
43

  ## On incremente theta du pas defini
  theta=$((theta+$thetaInc));

  ## Si theta depasse la borne superieure (180)
48 if [ "$theta" -gt "$thetaMaxLimit" ]

```

```

then

  ## On garde la borne superieure dans la variable u
  u=$thetaMaxLimit;
53
else

  ## On garde la valeur actuelle de theta dans la variable u
  u=$((theta));
58
fi

## On initialise la variable theta
phi=$phiMin;
63

## On boucle jusqu'a ce que theta atteigne la borne maximale
while [ $phi -lt $phiMax ]
do
68
  ## On garde la valeur actuelle de phi dans la variable v
  v=$((phi));

  ## On incremente phi du pas defini
  phi=$((phi+$phiInc));
73

  ## Si phi depasse depasse la borne superieure (360)
  if [ "$phi" -gt "$phiMaxLimit" ]
  then
78
    ## On garde la borne superieure dans la variable u
    w=$((phiMaxLimit));

  else
83
    ## On garde la valeur actuelle de phi dans la variable w
    w=$((phi));

  fi

  ## On assemble la ligne de parametres a ecrire
  echo output$theta_"$phi $soft $params $s $u $v $w
88

done;
done;
93
exit 0;

```

B.2.3 Compilation et déploiement de l'outil de simulation sur les clusters

```

1 #!/bin/bash
##=====
#####          C O N S E R T          #####
##=====
## Deploiement d'une simulation sur la grille
6 ##
##=====
## Auteurs : Philippe BOLLARD, Stephane CHAILLOL
## Date   : 23-04-2010
## Version : 1.0
11 #####

## On controle le nombre de parametres
if [[ $# -ne 5 ]]
then
16   echo "Usage: $0 <clusters> <pathIn> <pathOut> <source> <binary>"
   exit 1
fi

## Parametres
21 #####

```

```
## Liste des clusters
CLUSTERS=`cat $1`

26 ## Chemin source
PATHIN=$2

## Chemin destination
PATHOUT=$3
31

## Source
SRC=$4

## Binaire
36 BIN=$5

## Deploiement sur les clusters
#####
for HOST in $CLUSTERS;
41 do

## Test de la connexion SSH
ssh -q -o NumberOfPasswordPrompts=0 -o ConnectTimeout=5 -o StrictHostKeyChecking=no $USER@$HOST
touch .testHost
46 if [[ $? -eq 0 ]]
then

## Copie du source
ssh $USER@$HOST mkdir -p $PATHOUT
scp -qr $PATHIN/* $USER@$HOST:$PATHOUT
51

## Compilation du source
case $HOST in

"medetphy.imag.fr" | "zephir.mirage.ujf-grenoble.fr" | "healthphy.ujf-grenoble.fr" | "
56 medetphy" | "zephir" | "healthphy" )
ssh $USER@$HOST ifort -g $PATHOUT/$SRC -o $PATHOUT/$BIN
;;

"genepi.imag.fr" | "browalle.ujf-grenoble.fr" | "genepi" | "browalle" )
61 ssh $USER@$HOST g77 -g $PATHOUT/$SRC -o $PATHOUT/$BIN
;;

"fostino.obs.ujf-grenoble.fr" | "r2d2.obs.ujf-grenoble.fr" | "fostino" | "r2d2" )
66 ssh $USER@$HOST f95 -g $PATHOUT/$SRC -o $PATHOUT/$BIN
;;

"icare.obs.ujf-grenoble.fr" | "icare" )
ssh $USER@$HOST source .profile /opt/SUNWspro/bin/f95 -g $PATHOUT/$SRC -o $PATHOUT/$BIN
71 ;;

"nanostar.ujf-grenoble.fr" | "nanostar" )
ssh $USER@$HOST gfortran -o $PATHOUT/$BIN $PATHOUT/$SRC
76 ;;
esac

## Changement du droit d'execution du binaire
ssh $USER@$HOST chmod +x $PATHOUT/$BIN

fi
done
81

## C'est fini
#####
exit 0;
```

B.2.4 Lancement d'une tâche sur un cluster

```
1 #!/bin/bash
  ##=====
  #####          C O N S E R T          #####
  ##=====
  ## Lancement d'un job
6 ##
  ##=====
  ## Auteurs : Philippe BOLLARD, Stephane CHAILLOL
  ## Date   : 20-04-2010
  ## Version : 1.0
11 #####

  ## On controle le nombre de parametres
  #####
  if [[ $# -ne 7 ]]
16 then
    echo "Usage: $0 <output> <soft> <config_file> <theta_min> <theta_max> <phi_min> <phi_max>"
    exit 1
  fi

21 ## On recupere les parametres
  #####
  DIR=$1
  APPLI=$2
  CONFIG=$3
26 THETAMIN=$4
  THETAMAX=$5
  PHIMIN=$6
  PHIMAX=$7

31 ## Creation du repertoire
  #####
  mkdir -p $DIR
  cp -f template/* $DIR
  cd $DIR

36 ## Creation du fichier stdin
  #####
  (
41  cat <<'EOF'
    %paramsPrg
    %thetaMin
    %thetaMax
    %phiMin
    %phiMax
46  EOF
  ) > stdin

  ## Remplacement des valeurs
  #####
51 perl -pi -e "s/%paramsPrg/$CONFIG/" stdin
  perl -pi -e "s/%thetaMin/$THETAMIN/" stdin
  perl -pi -e "s/%thetaMax/$THETAMAX/" stdin
  perl -pi -e "s/%phiMin/$PHIMIN/" stdin
  perl -pi -e "s/%phiMax/$PHIMAX/" stdin

56 ## On assemble la commande
  #####
  abspath="$(cd "${0%*/}"/.. 2>/dev/null; echo "$PWD"/"${0##*/}")"
  path_only=`dirname "$abspath"`
61 $path_only/$APPLI < stdin
```

B.2.5 Récupération et concaténation des résultats

```

#!/bin/bash
#####
3 #####          C O N S E R T          #####
#####
## Recuperation des donnees d'un job
##
#####
8 ## Auteurs : Philippe BOLLARD
## Date   : 03-05-2010
## Version : 1.0
#####

13 ## On controle le nombre de parametres
#####
if [[ $# -ne 3 ]]
then
18   echo "Usage: $0 </path/numjob.txt> <joboutputfilename.dat> </path/out.dat.gz>"
   exit 1
fi

## On recupere les parametres
#####
23 CIGRI=/var/lib/cigri/results/$USER
NUMJOBFILE=$1
DATFILE=$2
PATHOUT=$3

28 ## Liste des jobs
JOBS=`cat $NUMJOBFILE`
for JOBID in $JOBS;
do

33   ## On verifie si les resultats ont ete recuperes sur la frontale
   if [[ -d $CIGRI/$JOBID ]]; then
   {

38     ## Creation du dossier temporaire
     #####
     cd $CIGRI;
     touch $JOBID.dat
     mkdir -p $JOBID.tmp
     cd $JOBID.tmp

43     ## Extraction des resultats
     #####
     find $CIGRI/$JOBID/ -name "*tgz" -exec tar zxf {} -C ./ \;

48     ## Concatenation des resultats
     #####
     for fichier in ./output*/$DATFILE; do cat $fichier >> ../$JOBID.dat; done;

53     ## Compression du fichier final
     #####
     cd ..
     gzip -c $JOBID.dat > $PATHOUT

58     ## Nettoyage des fichiers/dossiers temporaires
     rm -rf $JOBID.tmp
     rm $JOBID.dat

     ## Nettoyage des fichiers sources
     rm -rf $CIGRI/$JOBID

63   }
   fi
done

68 exit 0;

```

C Extraits de code source

C.1 Fichier de configuration

```
[production]
2  phpSettings.display_startup_errors = 0
  phpSettings.display_errors = 0
  includePaths.library = APPLICATION_PATH "../library"
  bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
  bootstrap.class = "Bootstrap"
7  appnamespace = "Application"
  resources.frontController.controllerDirectory = APPLICATION_PATH "/controllers"
  resources.frontController.params.displayExceptions = 0
  resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/"
  resources.view[] =
12 resources.db.adapter = "PDO_MYSQL"
  resources.db.isDefaultTableAdapter = true
  resources.db.params.charset = utf8
  resources.db.params.host = "localhost"
  resources.db.params.username = "consert"
17 resources.db.params.password = "myconsertsql"
  resources.db.params.dbname = "consert"

[staging : production]

22 [testing : production]
  phpSettings.display_startup_errors = 1
  phpSettings.display_errors = 1

[development : production]
27 phpSettings.display_startup_errors = 1
  phpSettings.display_errors = 1
  resources.frontController.params.displayExceptions = 1
  resources.db.params.username = "consert"
  resources.db.params.password = "consert"
32 resources.db.params.dbname = "consert"
```

C.2 Fichier de bootstrap

```
<?php
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap {
3
  /**
   * Initialize Doctype
   */
  protected function _initDoctype() {
8    $this->bootstrap('view');
    $view = $this->getResource('view');
    $view->doctype('XHTML1_STRICT');
  }

13 /**
   * Initialize Zend_Translate
   */
  protected function _initTranslate() {
    $translate = new Zend_Translate('array', APPLICATION_PATH . '/locales', null, array('scan' =>
18     Zend_Translate::LOCALE_DIRECTORY));
    Zend_Registry::set('Zend_Translate', $translate);
    return $translate;
  }

23 /**
   * Initialize Zend_Navigation
   */
  protected function _initNavigation() {
    $view = $this->bootstrap('layout')->getResource('layout')->getView();
    $config = new Zend_Config_Xml(APPLICATION_PATH . '/configs/navigation.xml', 'nav');
28    $navigation = new Zend_Navigation($config);
  }
}
```



```

    Zend_Registry::set('Zend_Navigation', $navigation);
}

/**
 * Initialize Zend_Acl
 */
33 protected function _initAcl() {
    $acl = new Application_Model_User_Acl_Acl(APPLICATION_PATH . '/configs/acl.ini') ;
    Zend_Registry::set('Zend_Acl', $acl);
38
    $front = $this->bootstrap('layout')->getResource('frontController');
    $front->registerPlugin(new Application_Model_User_Acl_PluginAuth($acl)) ;

    $this->bootstrap('navigation');
43 $view = $this->getResource('view');

    $auth = Zend_Auth::getInstance();
    if ($auth->hasIdentity()) {
        $user = $auth->getStorage()->read() ;
        $role = ($user->isAdmin()) ? 'admin' : 'member';
48     } else {
        $role = 'guest';
    }

53 $view->navigation()->setAcl($acl)->setRole($role);
}

/**
 * Initialize Helpers
 */
58 protected function _initHelpers() {
    $this->bootstrap('view');
    $view = $this->getResource('view');
    $view->addHelperPath(APPLICATION_PATH . '/views/helpers', 'Application_View_Helper');
63 }
}
?>

```

C.3 Classe processus

```

<?php
/**
 * Process class
 * @author Philippe BOLLARD
5 */
class Application_Model_Process_Class_Process extends Application_Model_AbstractClass {

// PROPERTIES
//-----
10
/**
 * ID of the related project
 * @var string
 */
15 protected $_idProject;

/**
 * ID of the related Chain
 * @var string
 */
20 protected $_idChain;

/**
 * State
 * @var string
 */
25 protected $_state;

/**
 * Output path
30

```

```

    * @var string
    */
protected $_outputPath;
35 /**
    * Output idHost
    * @var string
    */
40 protected $_outputIdHost;

    /**
    * Group name
    * Enter description here ...
    * @var string
45 */
protected $_groupName;

    /**
    * ID of the related user
    * @var string
50 */
protected $_idUser;

// STATIC
55 //-----

// METHODS
//-----

60 /**
    * Csonstructor
    * @param array $options
    */
65 public function __construct(array $options = null) {
    $this->_state = Application_Model_Process_Factory_Process::STATE_INITIALIZED;
    parent::__construct($options);
}

    /**
70 * Set de ID of the related project
    * @param $id
    */
    public function setIdProject($value) {
75     $this->_idProject = (!empty($value)) ? $value : null;
    return $this;
}

    /**
80 * Get the ID of the related project
    */
    public function getIdProject() {
    return $this->_idProject;
}

    /**
85 * Set de ID of the related project with an object
    * @param $project
    */
90 public function setProject($project) {
    return $this->setIdProject($project->getId());
}

    /**
95 * Get the related project
    */
    public function getProject() {
    return Application_Model_Project_Factory_Project::find($this->_idProject);
}

    /**
100 * Set de ID of the related chain
    * @param $id
    */
105 public function setIdChain($value) {
    $this->_idChain = (!empty($value)) ? $value : null;
    return $this;
}

```

```
    }

    /**
    * Get the ID of the related chain
    */
110 public function getIdChain() {
    return $this->_idChain;
    }

115 /**
    * Set de ID of the related chain with an object
    * @param $chain
    */
120 public function setChain($chain) {
    return $this->setIdChain($chain->getId());
    }

125 /**
    * Get the related chain
    */
    public function getChain() {
    return Application_Model_Chain_Factory_Chain::find($this->_idChain);
    }

130 /**
    * Set the state
    * @param $value
    */
135 public function setState($value) {
    $this->_state = (string) $value;
    return $this;
    }

140 /**
    * Get the state
    */
    public function getState() {
145     if (empty($this->_state)) {
        $this->_state = Application_Model_Process_Factory_Process::STATE_INITIALIZED;
    }

    return $this->_state;
    }

150 /**
    * Set the output path
    * @param $value
    */
155 public function setOutputPath($value) {
    $this->_outputPath = (string) $value;
    return $this;
    }

160 /**
    * Get the output path
    */
    public function getOutputPath() {
165     return $this->_outputPath;
    }

170 /**
    * Set the output host
    * @param $value
    */
    public function setOutputHost($value) {
175     if (is_null($value)) {
        $this->_outputIdHost = $value->getId();
    }

    return $this;
    }

180 /**
    * Get the output host
    */
    public function getOutputHost() {
    if (!empty($this->_outputIdHost)) {
```

```

        return Application_Model_Host_Factory_Host::find($this->_outputIdHost);
    }
185     return false;
    }

    /**
     * Set the output idHost
     * @param $value
     */
190     public function setOutputIdHost($value) {
        $this->_outputIdHost = $value;
        return $this;
195     }

    /**
     * Get the output idHost
     */
200     public function getOutputIdHost() {
        return $this->_outputIdHost;
    }

    /**
     * Get the output
     */
205     public function getOutput() {
        return true;
    }
210

    /**
     * Set the groupName
     * @param $value
     */
215     public function setGroupName($value) {
        $this->_groupName = (string) $value;
        return $this;
    }

    /**
     * Get the groupName
     */
220     public function getGroupName() {
        return $this->_groupName;
225     }

    /**
     * Set de ID of the related user
     * @param $id
     */
230     public function setIdUser($value) {
        $this->_idUser = (!empty($value)) ? $value : null;
        return $this;
235     }

    /**
     * Get the ID of the related user
     */
240     public function getIdUser() {
        return $this->_idUser;
    }

    /**
     * Set de ID of the related user with an object
     * @param $user
     */
245     public function setUser($user) {
        return $this->setIdUser($user->getId());
    }

    /**
     * Get the related user
     */
250     public function getUser() {
        return Application_Model_User_Factory_User::find($this->_idUser);
255     }

    /**

```

```

260     * Get the list of associated tasks
    */
    public function getProcessTasks() {
        return Application_Model_Process_Factory_ProcessTask::fetchAll(array('process'=>$this, '
            order'=>'order ASC'));
    }
265 /**
    * Get the list of associated variables
    */
    public function getProcessVariables() {
270         return Application_Model_Process_Factory_ProcessVariable::fetchAll(array('process'=>$this));
    }

    /**
    * Check the state - INITIALIZED
    */
275 public function isInitialized() {
        return ($this->getState() == Application_Model_Process_Factory_Process::STATE_INITIALIZED);
    }

    /**
    * Check the state - WAITING
    */
280 public function isWaiting() {
        return ($this->getState() == Application_Model_Process_Factory_Process::STATE_WAITING);
    }
285

    /**
    * Check the state - RUNNING
    */
290 public function isRunning() {
        return ($this->getState() == Application_Model_Process_Factory_Process::STATE_RUNNING);
    }

    /**
    * Check the state - TERMINATED
    */
295 public function isTerminated() {
        return ($this->gete($processState);
            Application_Model_Process_Factory_Process::save($this);
            //exit;
300     }
    }

    /**
    * Stop the process
    */
305 public function stop() {
        //We assume that this process is running or is waiting
        if ($this->isRunning() || $this->isWaiting()) {
310
            //We initialize a flag
            $stopWhile = false;
            $indexTask = 0;

            //We'll check all tasks
315             $tasks = $this->code source deState() == Application_Model_Process_Factory_Process::
                STATE_TERMINATED);
        }

    /**
    * Check the state - FAILED
    */
320 public function isFailed() {
        return ($this->getState() == Application_Model_Process_Factory_Process::STATE_FAILED);
    }

    /**
    * Check the state - CANCELED
    */
325 public function isCanceled() {
        return ($this->getState() == Application_Model_Process_Factory_Process::STATE_CANCELED);
330     }
    }

    /**

```

```

    * Check the state - DELETED
    */
335 public function isDeleted() {
    return ($this->getState() == Application_Model_Process_Factory_Process::STATE_DELETED);
}

/**
 * Resolve a context variable string
 * @param $arg
 */
340 public function resolveContext($arg) {
    switch (strtolower($arg)) {
345     case 'idproject':
        $stoReturn = $this->_idProject;
        break;

        case 'idchain':
350     $stoReturn = $this->_idChain;
        break;

        default:
355     $stoReturn = parent::resolveContext($arg);
        break;
    }
    return $stoReturn;
}

/**
 * Start the process
 * @param $forced
 */
360 public function start($forced=false) {

    //We assume that this process is waiting to start
    if ($this->isWaiting() || $this->isInitialized() || $forced) {

370     //We change the state for this process
        $this->setState(Application_Model_Process_Factory_Process::STATE_RUNNING);
        Application_Model_Process_Factory_Process::save($this);

        //We check the state
        $this->checkState();
375     } else {
        $this->checkState();
    }
}

/**
 * Check the state - launch tasks
 */
380 public function checkState() {

    //We assume that this process is running
    if ($this->isRunning()) {

385     //We initialize a flag
        $stopWhile = false;
        $indexTask = 0;
390     $processState = Application_Model_Process_Factory_Process::STATE_RUNNING;

        //We'll check all tasks
        $tasks = $this->getChain()->getTasks();
395     while ($indexTask < count($tasks) && !$stopWhile) {

        //We select a task
        $task = $tasks[$indexTask];
400     $processTask = Application_Model_Process_Factory_ProcessTask::find($this->getId(), $task
        ->getId());

        //We check its state
        $task->checkState($processTask);
        //echo $indexTask.' | '.$task->getId().' - '.$task->getName().' : '.$task->getState()."\n";
405     n<br>";

        //If the task is waiting / initialized

```

```

    if ($processTask->isWaiting() || $processTask->isInitialized()) {
        //We start the task
        $task->start($processTask);
    }

    //If the task is running
    elseif ($processTask->isRunning()) {
        //We stop here, the process is running
        $stopWhile = true;

        //We choose the state for the process
        $processState = Application_Model_Process_Factory_Process::STATE_RUNNING;
    }

    //If the task is terminated / canceled / deleted
    elseif ($processTask->isTerminated() || $processTask->isCanceled() || $processTask->
        isDeleted()) {
        //We jump to the next task
        $indexTask++;
    }

    //If the task is failed
    elseif ($processTask->isFailed()) {
        //We stop here, the process is failed
        $stopWhile = true;

        //We choose the state for the process
        $processState = Application_Model_Process_Factory_Process::STATE_FAILED;
    }
}

//If all is completed
if (!$stopWhile) {
    //We choose the state for the process
    $processState = Application_Model_Process_Factory_Process::STATE_TERMINATED;
}

//We change the state for this process
$this->setState($processState);
Application_Model_Process_Factory_Process::save($this);
//exit;
}
}

/**
 * Stop the process
 */
public function stop() {
    //We assume that this process is running or is waiting
    if ($this->isRunning() || $this->isWaiting()) {

        //We initialize a flag
        $stopWhile = false;
        $indexTask = 0;

        //We'll check all tasks
        $tasks = $this->getChain()->getTasks();

        while ($indexTask < count($tasks) && !$stopWhile) {
            //We select a task
            $task = $tasks[$indexTask];
            $processTask = Application_Model_Process_Factory_ProcessTask::find($this->getId(), $task
                ->getId());

            //We check its state
            $task->checkState($processTask);

            //If the task is running
            if ($processTask->isRunning()) {

```

```

        //We stop the task
        $task->stop($processTask);

        //We check the state another time
485     $task->checkState($processTask);

        //If the task is running
        if ($processTask->isRunning()) {
490         $stopWhile = true;
        }

        //We jump to the next task
        $indexTask++;
495     }

        //If the task is waiting / initialized / canceled / failed / ...
        else {

            //We jump to the next task
500         $indexTask++;
        }
    }

    if (!$stopWhile) {
505        //We change the state for this process
        $this->setState(Application_Model_Process_Factory_Process::STATE_CANCELED);
        Application_Model_Process_Factory_Process::save($this);
    }
}

/**
 * Reset the process
 */
515 public function reset() {
    if (!$this->isDeleted()) {
        $this->setState(Application_Model_Process_Factory_Process::STATE_INITIALIZED);
        Application_Model_Process_Factory_Process::save($this);
520        foreach ($this->getProcessTasks() as $processTask) {
            $processTask->reset();
        }
    }
}
}

```

C.4 Classe fabrique d'objets processus

```

1 <?php
  /**
   * Process factory
   * @author Philippe BOLLARD
   */
6 class Application_Model_Process_Factory_Process {

  // CONSTS
  //-----

11  /**
   * State - Initialized
   */
   const STATE_INITIALIZED = 'INITIALIZED';

16  /**
   * State - Waiting
   */
   const STATE_WAITING = 'WAITING';

21  /**
   * State - Running
   */

```



```
const STATE_RUNNING = 'RUNNING';
26 /**
   * State - Terminated
   */
const STATE_TERMINATED = 'TERMINATED';
31 /**
   * State - Failed
   */
const STATE_FAILED = 'FAILED';
36 /**
   * State - Canceled
   */
const STATE_CANCELED = 'CANCELED';
41 /**
   * State - Deleted
   */
const STATE_DELETED = 'DELETED';
46 // PROPERTIES
//-----

// STATIC
//-----
51 /**
   * Return the lists of states
   */
static public function getStates() {
56     $translate = Zend_Registry::get('Zend_Translate');
    return array(
        self::STATE_CANCELED => self::getStateLabel(self::STATE_CANCELED),
        self::STATE_DELETED => self::getStateLabel(self::STATE_DELETED),
61     self::STATE_FAILED => self::getStateLabel(self::STATE_FAILED),
        self::STATE_INITIALIZED => self::getStateLabel(self::STATE_INITIALIZED),
        self::STATE_RUNNING => self::getStateLabel(self::STATE_RUNNING),
        self::STATE_TERMINATED => self::getStateLabel(self::STATE_TERMINATED),
        self::STATE_WAITING => self::getStateLabel(self::STATE_WAITING),
66     );
}

/**
 * Return the lable for a state
 */
71 static public function getStateLabel($state) {
    $translate = Zend_Registry::get('Zend_Translate');
    switch($state) {
        case self::STATE_CANCELED:
76         return $translate->translate('Process.Factory.State.Canceled');
        break;

        case self::STATE_DELETED:
            return $translate->translate('Process.Factory.State.Deleted');
81         break;

        case self::STATE_FAILED:
            return $translate->translate('Process.Factory.State.Failed');
86         break;

        case self::STATE_INITIALIZED:
            return $translate->translate('Process.Factory.State.Initialized');
91         break;

        case self::STATE_RUNNING:
            return $translate->translate('Process.Factory.State.Running');
96         break;

        case self::STATE_TERMINATED:
            return $translate->translate('Process.Factory.State.Terminated');
            break;

        case self::STATE_WAITING:
            return $translate->translate('Process.Factory.State.Waiting');
```

```

101         break;
102     default:
103         return '';
104         break;
105     }
106 }

/**
 * Instance of DB Mapper (Singleton)
 * @var unknown_type
111 */
static private $_dbMapper = null;

/**
 * Get the instance of DB Mapper
116 */
static public function getDbMapper() {
    if (is_null(self::$_dbMapper)) {
        self::$_dbMapper = new Application_Model_Process_Db_Mapper_Process();
    }
121     return self::$_dbMapper;
}

/**
 * Create an instance
126 * @return Application_Model_Process_Class_Process
 * @param $options
 */
static public function create($options) {
131     return new Application_Model_Process_Class_Process($options);
}

/**
 * Find an object by ID
136 * @param unknown_type $id
 */
static public function find($id) {
    $object = self::getDbMapper()->find($id);
    if (is_null($object)) {
141         throw new Exception('Process not found');
        return null;
    }
    return $object;
}

146 /**
 * Fetch all objects
 */
static public function fetchAll($options=array()) {
151     if (is_array($options)) {
        $where = self::getDbMapper()->getDbTable()->select(Zend_Db_Table_Abstract::
            SELECT_WITH_FROM_PART);
        $order = null;
        $count = null;
        $offset = null;
        $limit = null;

156         if (count($options)>0) {
            foreach ($options as $option=>$value) {
                switch ($option) {
                    case 'name':
161                     if (!empty($value)) {
                        $where->where('name like ?', '%'.$value.'%');
                    }
                    break;

                    case 'description':
166                     if (!empty($value)) {
                        $where->where('description like ?', '%'.$value.'%');
                    }
                    break;

                    case 'state':
171                     if (!empty($value)) {
                        $where->where('state = ?', $value);
                    }
                }
            }
        }
    }
}

```

```
176         }
        break;

        case 'idProject':
            $where->where('idProject = ?', $value);
            break;
181
        case 'project':
            $where->where('idProject = ?', $value->getId());
            break;

186
        case 'idChain':
            $where->where('idChain = ?', $value);
            break;

        case 'chain':
191            $where->where('idChain = ?', $value->getId());
            break;

        case 'group':
196        case 'groupName':
            $where->where('groupName = ?', $value);
            break;

        case 'order':
201            $where->order($value);
            break;

        case 'count':
            $count = $value;
            break;
206
        case 'limit':
            $limit = $value;
            break;

211
        case 'offset':
            $offset = $value;
            break;
    }
}
216
}

if (!is_null($count) && !is_null($offset)) {
    $where->limit($count, $offset);
}
221
} else {
    $where = $options;
}

226
return self::getDbMapper()->fetchAll($where);
}

/**
 * Save an object
 * @param $object
 */
static public function save($object) {
236     if (null === ($id = $object->getId())) {
        $idUser = $object->getIdUser();
        if (empty($idUser)) {
            $object->setUser(Zend_Auth::getInstance()->getIdentity());
        }
        self::getDbMapper()->save($object);
241
        /*foreach ($object->getChain()->getTasks() as $task) {
            $ptaskOptions = array('idProcess'=>$object->getId(), 'idTask'=>$task->getId());
            $ptask = Application_Model_Process_Factory_ProcessTask::create($ptaskOptions);
            Application_Model_Process_Factory_ProcessTask::insert($ptask);
246        }*/
    }
    return self::getDbMapper()->save($object);
}
```

```

251  /**
     * Delete an object
     * @param $object
     */
    static public function delete($object) {
256      return self::getDbMapper()->delete($object->getId());
    }

    /**
     * Clone and save a process
     * @param $object
     */
261    static public function cloneAndSave($object) {
        $objectNew = clone $object;
        $objectNew->setName("NEW-".$object->getName());
266      self::save($objectNew);

        /*foreach ($object->getProcessVariables() as $var) {
            $varNew = clone $var;
            $varNew->setIdProcess($objectNew->getId());
271          Application_Model_Process_Factory_ProcessVariable::insert($varNew);
        }*/

        foreach ($object->getProcessTasks() as $task) {
276          $taskNew = clone $task;
          $taskNew->setIdProcess($objectNew->getId());
          Application_Model_Process_Factory_ProcessTask::insert($taskNew);
        }

        return $objectNew;
281    }
}

```

C.5 Classe d'adaptation entre le modèle de données et le modèle objet pour les processus

```

<?php
/**
3  * Process database mapper
  * @author Philippe BOLLARD
  */
class Application_Model_Process_Db_Mapper_Process {

8  //PROPERTIES\subsection{}
  //-----

  /**
  * Dbtable
13  */
  protected $_dbTable;

  //METHODS
  //-----

18  /**
  * Set the dbtable
  * @param $dbTable
  */
23  public function setDbTable($dbTable) {
      if (is_string($dbTable)) {
          $dbTable = new $dbTable();
      }
      if (!$dbTable instanceof Zend_Db_Table_Abstract) {
28          throw new Exception('Invalid table data gateway provided');
      }
      $this->_dbTable = $dbTable;
      return $this;
33  }
}

```

```

/**
 * Get the dbtable
 */
public function getDbTable() {
38     if (null === $this->_dbTable) {
        $this->setDbTable('Application_Model_Process_Db_Table_Process');
    }
    return $this->_dbTable;
43 }

/**
 * Save the object
 * @param Application_Model_Process_Process $object
 */
48 public function save(Application_Model_Process_Class_Process $object) {
    $dateCreated = $object->getCreated();
    if (empty($dateCreated)) $dateCreated = date('Y-m-d H:i:s');
    $data = array(
53     'name' => $object->getName(),
    'description' => $object->getDescription(),
    'created' => $dateCreated,
    'updated' => date('Y-m-d H:i:s'),
    'idProject' => $object->getIdProject(),
58     'idChain' => $object->getIdChain(),
    'state' => $object->getState(),
    'outputPath' => $object->getOutputPath(),
    'outputIdHost' => $object->getOutputIdHost(),
    'groupName' => $object->getGroupName(),
63     'idUser' => $object->getIdUser(),
    );

    if (null === ($id = $object->getId())) {
        $data['id'] = $object->generateId();
        $object->setId($data['id']);
68     $this->getDbTable()->insert($data);
    } else {
        $this->getDbTable()->update($data, array('id = ?' => $id));
    }
73 }

/**
 * Find an object in the database
 * @param $id
 * @param Application_Model_Process_Process $process
78 */
public function find($id, Application_Model_Process_Class_Process $object=null) {
    $result = $this->getDbTable()->find($id);
    if (0 == count($result)) {
83     return;
    }
    $row = $result->current();
    if (is_null($object)) {
        $object = new Application_Model_Process_Class_Process();
88     $object->setId($row->id)
        ->setName($row->name)
        ->setDescription($row->description)
        ->setCreated($row->created)
        ->setUpdated($row->updated)
93     ->setIdProject($row->idProject)
        ->setIdChain($row->idChain)
        ->setState($row->state)
        ->setOutputPath($row->outputPath)
        ->setOutputIdHost($row->outputIdHost)
98     ->setGroupName($row->groupName)
        ->setIdUser($row->idUser);

    return $object;
103 }

/**
 * Fetch all objects from database
 */
108 public function fetchAll($where=null, $order=null, $count=null, $offset=null) {
    $resultSet = $this->getDbTable()->fetchAll($where, $order, $count, $offset);
    $entries = array();
}

```

```

    foreach ($resultSet as $row) {
        $entry = new Application_Model_Process_Class_Process();
        $entry->setId($row->id)
113         ->setName($row->name)
            ->setDescription($row->description)
            ->setCreated($row->created)
            ->setUpdated($row->updated)
            ->setIdProject($row->idProject)
118         ->setIdChain($row->idChain)
            ->setState($row->state)
            ->setOutputPath($row->outputPath)
            ->setOutputIdHost($row->outputIdHost)
            ->setGroupName($row->groupName)
123         ->setIdUser($row->idUser);
        $entries[] = $entry;
    }
    return $entries;
}
128
/**
 * Delete
 */
public function delete($id) {
133     $where = $this->getDbTable()->getAdapter()->quoteInto('id = ?', $id);
    return $this->getDbTable()->delete($where);
}
}

```

C.6 Contrôleur de processus

```

<?php
/**
 * Process controller
4  * @author Philippe BOLLARD
 */
class ProcessController extends Zend_Controller_Action {

    /**
9     * Init
    */
    public function init() {
        $this->_redirector = $this->_helper->getHelper('Redirector');

14        //Store the back url
        if (in_array($this->_request->getActionName(), array('index', 'list', 'show'))) {
            $sessionNP = new Zend_Session_Namespace();
            if (isset($sessionNP->currentRequest)) {
                $sessionNP->backRequest = $sessionNP->currentRequest;
19            }
            $sessionNP->currentRequest = $this->getRequest();
        }
    }

24    /**
     * Pre-dispatch
    */
    public function preDispatch() {
        if (!Zend_Auth::getInstance()->hasIdentity()) {
29            $this->_helper->redirector('login', 'user');
        }
    }

34    /**
     * Index
    */
    public function indexAction() {
        $this->_helper->redirector('list', 'process');
39    }
}
/**

```

```

    * List
    */
44 public function listAction() {
    $page = $this->getRequest()->getParam('page', 1);
    $count = $this->getRequest()->getParam('count', 10);
    $offset = $count*($page-1);
    $this->view->page = $page;

49     try {
        $this->view->entries = Application_Model_Process_Factory_Process::fetchAll(array('count'=>
            $count, 'offset'=>$offset));
    } catch (Exception $e) {
        throw $e;
    }
54 }

/**
 * New
 */
59 public function newAction() {
    $request = $this->getRequest();
    $form = new Application_Form_Process_New();

64     try {

        //If a "project" parameter is set
        if (isset($request->project)) {

69             //We overload the "idProjet" element in the form
            $form->addElement('hidden', 'idProject');
            $form->setDefaults(array('idProject'=>$request->project));

            //We get the project object
            $this->view->project = Application_Model_Project_Factory_Project::find($request->project)
74             ;

            // Add an chain element
            $chains = array();
            foreach ($this->view->project->getChains() as $c) {
                $chains[$c->getId()] = $c->getName();
79             }
            $form->addElement('select', 'idChain', array(
                'label' => 'Process.Form.New.Chain',
                'required' => false,
                'multiOptions'=> $chains,
84             ));
        }

        //If a "chain" parameter is set
        if (isset($request->chain)) {

89             //We get the chain object
            $chain = Application_Model_Chain_Factory_Chain::find($request->chain);
            $this->view->chain = $chain;
            $this->view->project = $chain->getProject();

94             //We overload the "idChain" element in the form
            $form->addElement('hidden', 'idChain');
            $form->setDefaults(array('idChain'=>$chain->getId()));

99             //We overload the "idProjet" element in the form
            $form->addElement('hidden', 'idProject');
            $form->setDefaults(array('idProject'=>$chain->getIdProject()));
            // $form->removeElement('idProject');
104         }

        //If the form is posted, we'll check the data
        if ($this->getRequest()->isPost()) {
            if ($form->isValid($request->getPost())) {

109                 //We initialize a process object
                $object = Application_Model_Process_Factory_Process::create($form->getValues());
                Application_Model_Process_Factory_Process::save($object);

                //We initialize the tasks for this process
114                 foreach ($object->getChain()->getTasks() as $task) {

```

```

        $ptaskOptions = array('idProcess'=>$object->getId(), 'idTask'=>$task->getId(), 'order
            '>=> $task->getOrder());
        $ptask = Application_Model_Process_Factory_ProcessTask::create($ptaskOptions);
        Application_Model_Process_Factory_ProcessTask::insert($ptask);
    }
119
    //Subfunction for isolate data from variables to overload
    function findFields($field) {
        if (strpos($field, '_processVariable') !== false) {
124
            return $field;
        }
    }

    // Search $data for dynamically added fields using findFields callback
    $newFields = array_filter(array_keys($request->getPost()), 'findFields');
129

    foreach ($newFields as $fieldName) {
        $subformData = $request->getParam($fieldName);
        $objectVariable = Application_Model_Process_Factory_ProcessVariable::create(
            $subformData);
        $objectVariable->setProcess($object);
134
        Application_Model_Process_Factory_ProcessVariable::insert($objectVariable);
    }

    //We redirect to the process show action
    $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$object->getId()));
139
    }
} catch (Exception $e) {
    throw $e;
}
144

//We set the form to the view
$this->view->form = $form;
}

149
/**
 * New (multi-process)
 */
public function newmultiAction() {
154
    $request = $this->getRequest();
    $form = new Application_Form_Process_NewMulti();

    try {

159
        //If a "project" parameter is set
        if (isset($request->project)) {

            //We overload the "idProjet" element in the form
            $form->addElement('hidden', 'idProject');
164
            $form->setDefaults(array('idProject'=>$request->project));

            //We get the project object
            $this->view->project = Application_Model_Project_Factory_Project::find($request->project)
                ;
169
        }

        //If a "chain" parameter is set
        if (isset($request->chain)) {

            //We get the chain object
174
            $chain = Application_Model_Chain_Factory_Chain::find($request->chain);
            $this->view->chain = $chain;
            $this->view->project = $chain->getProject();

            //We overload the "idChain" element in the form
179
            $form->addElement('hidden', 'idChain');
            $form->setDefaults(array('idChain'=>$chain->getId()));

            //We overload the "idProjet" element in the form
            $form->addElement('hidden', 'idProject');
184
            $form->setDefaults(array('idProject'=>$chain->getIdProject()));
            // $form->removeElement('idProject');
        }
    }
}

```



```

189 //If the form is posted, we'll check the data
    if ($this->getRequest()->isPost()) {
        if ($form->isValid($request->getPost())) {

194 //We get the id of the project (from a parameter)
            $idProject = $form->getValue("idProject");

//Subfonction for isolate data from variables to overload
199 function findFields($field) {
    if (strpos($field, '_processVariable') !== false) {
        return $field;
    }
}

//Subfonction for isolate data from variables to overload and to build multi-processes
204 function findFieldMulti($field) {
    if (strpos($field, '_variableMulti') !== false) {
        return $field;
    }
}

209 // Search $data for dynamically added fields using findFields callback
    $newFieldsMulti = array_filter(array_keys($request->getPost()), 'findFieldMulti');
    foreach ($newFieldsMulti as $fieldMultiName) {

214 //We get the data for this field
        $sfMultiData = $request->getParam($fieldMultiName);

//We get the set of values for this field
219 switch ($sfMultiData['type']) {

    case Application_Model_Variable_Factory_Variable::TYPE_INT:
    case Application_Model_Variable_Factory_Variable::TYPE_FLOAT:
        $valsMulti = array();
        for ($v = $sfMultiData['valueMin']; $v <= $sfMultiData['valueMax']; $v +=
224 $sfMultiData['valueInc']) {
            $valsMulti[] = $v;
        }
        break;

    case Application_Model_Variable_Factory_Variable::TYPE_FILE:
229 $valsMulti = $sfMultiData['idFile'];
        break;

    case Application_Model_Variable_Factory_Variable::TYPE_TEXT:
234 $valsMulti = $sfMultiData['value'];
        break;

    default:
239 $valsMulti = array();
        break;
}

//For each value
    foreach ($valsMulti as $valMulti) {

244 //We initialize a process object
        $object = Application_Model_Process_Factory_Process::create($form->getValues());
        Application_Model_Process_Factory_Process::save($object);

//We initialize the tasks for this process
249 foreach ($object->getChain()->getTasks() as $task) {
            $ptaskOptions = array('idProcess'=>$object->getId(), 'idTask'=>$task->getId(), '
                order'=> $task->getOrder());
            $ptask = Application_Model_Process_Factory_ProcessTask::create($ptaskOptions);
            Application_Model_Process_Factory_ProcessTask::insert($ptask);
254 }

//-----
//We overload the value for this iteration
    $objectVariable = Application_Model_Process_Factory_ProcessVariable::create(
259 $sfMultiData);
    $objectVariable->setProcess($object);
    $objectVariable->setValue($valMulti);
    Application_Model_Process_Factory_ProcessVariable::insert($objectVariable);

```

```

//-----

//We overload the values for some other variables
264 $newFields = array_filter(array_keys($request->getPost()), 'findFields');
foreach ($newFields as $fieldName) {
    $subformData = $request->getParam($fieldName);

    $objectVariable = Application_Model_Process_Factory_ProcessVariable::create(
        $subformData);
269 $objectVariable->setProcess($object);
    Application_Model_Process_Factory_ProcessVariable::insert($objectVariable);
}
}
274 }

//We redirect to the project show action
$this->_redirector->gotoSimple('show', 'project', null, array('id'=>$idProject));
279 }
}

//We set the form to the view
$this->view->form = $form;
284 } catch (Exception $e) {
    throw $e;
}
}

/**
 * Edit
 */
public function editAction() {
294 try {
    $request = $this->getRequest();
    $form = new Application_Form_Process_Edit();

    if ($this->getRequest()->isPost()) {
299     if ($form->isValid($request->getPost())) {
        $object = Application_Model_Process_Factory_Process::create($form->getValues());
        Application_Model_Process_Factory_Process::save($object);

        $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$object->getId()));
304     } else {
        $object = Application_Model_Process_Factory_Process::find($request->id);
        $form->populate($object->toArray());

        $this->view->project = $object->getProject();
309     $this->view->chain = $object->getChain();
    }

    $this->view->form = $form;
314 } catch (Exception $e) {
    throw $e;
}
}

/**
 * Show
 */
public function showAction() {
324 try {
    $request = $this->getRequest();
    $this->view->process = Application_Model_Process_Factory_Process::find($request->id);
} catch (Exception $e) {
    $this->_redirector->gotoSimple('index', 'index');
}
}
329 }

/**
 * Delete
 */
public function deleteAction() {
334 try {
    $request = $this->getRequest();

```

```

    if ($object = Application_Model_Process_Factory_Process::find($request->id)) {
        if (!Application_Model_Process_Factory_Process::delete($object)) {
339             //ERROR
                return $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$object->
                    getId()));
            }
            return $this->_redirector->gotoSimple('show', 'project', null, array('id'=>$object->
                getIdProject()));
        }
        return $this->_redirector->gotoSimple('index');
344    } catch (Exception $e) {
        throw $e;
    }
}

349 /**
    * Start
    */
public function launchAction() {
354     try {
        $request = $this->getRequest();
        $process = Application_Model_Process_Factory_Process::find($request->id);
        if ($process->isInitialized()) {
            $process->setState(Application_Model_Process_Factory_Process::STATE_WAITING);
            Application_Model_Process_Factory_Process::save($process);
359        }

        return $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$process->getId
            ());
        } catch (Exception $e) {
            throw $e;
364        }
    }

    /**
    * Start
    */
369 public function startAction() {
    try {
        $request = $this->getRequest();
        $process = Application_Model_Process_Factory_Process::find($request->id);
374        $process->start();

        return $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$process->getId
            ());
        } catch (Exception $e) {
            throw $e;
379        }
    }

    /**
    * Start
    */
384 public function restartAction() {
    try {
        $request = $this->getRequest();
        $process = Application_Model_Process_Factory_Process::find($request->id);
389        $process->start(true);

        return $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$process->getId
            ());
        } catch (Exception $e) {
            throw $e;
394        }
    }

    /**
    * Reset
    */
399 public function resetAction() {
    try {
        $request = $this->getRequest();
        $process = Application_Model_Process_Factory_Process::find($request->id);
404        $process->reset();
    }
}

```

```

        return $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$process->getId
            ());
    } catch (Exception $e) {
        throw $e;
    }
}

/**
 * Start
 */
414 public function stopAction() {
    try {
        $request = $this->getRequest();
        $process = Application_Model_Process_Factory_Process::find($request->id);
419         $process->stop();

        return $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$process->getId
            ());
    } catch (Exception $e) {
        throw $e;
    }
}

/**
 * Clone
 */
429 public function cloneAction() {
    try {
        $request = $this->getRequest();
        $object = Application_Model_Process_Factory_Process::find($request->id);
434         $objectNew = Application_Model_Process_Factory_Process::cloneAndSave($object);
        $objectNew->reset();

        if (!is_null($objectNew)) {
            $this->_redirector->gotoSimple('show', 'process', null, array('id'=>$objectNew->getId()))
                ;
439         } else {
            $this->_redirector->gotoSimple('show');
        }
    } catch (Exception $e) {
        throw $e;
    }
}

/**
 * Search
 */
449 public function searchAction() {
    try {
        $sessionNP = new Zend_Session_Namespace('ProcessSearch');
        $request = $this->getRequest();
454         $form = new Application_Form_Process_Search();

        //If the form is posted, we'll check the data
        if ($this->getRequest()->isPost()) {
459             if ($form->isValid($request->getPost())) {

                function parseData($col, $cond, $value) {
                    switch ($cond) {
                        case 'eq': $c = $col. " = ?"; $v = $value; break;
                        case 'ne': $c = $col. " <> ?"; $v = $value; break;
464                         case 'lt': $c = $col. " < ?"; $v = $value; break;
                        case 'gt': $c = $col. " > ?"; $v = $value; break;
                        case 'ends': $c = $col. " like ?"; $v = "%".$value; break;
                        case 'begins': $c = $col. " like ?"; $v = $value."%"; break;
469                         case 'contains': $c = $col. " like ?"; $v = "%".$value."%"; break;
                        default: $c=""; $v="";
                    }
                    return array("condition"=>$c, "value"=>$v);
                }

474             function findSubforms($field) {
                if (stripos($field, '_sfSearch') !== false) {
                    return $field;
                }
            }
        }
    }
}

```

```

479 //We isolate data from variable subforms
    $subforms = array_filter(array_keys($request->getPost()), 'findSubforms');

484 //We initialize the process search query
    $selectProcess = Application_Model_Process_Factory_Process::getDbMapper()->getDbTable()
        ->select(Zend_Db_Table_Abstract::SELECT_WITH_FROM_PART);

    //We build queries with values get by the form
    $selectPV = array();
    $selectV = array();
489 foreach ($subforms as $sub) {
        $subformData = $request->getParam($sub);
        $sform = $sessionNP->subforms[$subformData['uniqid']];
        if ($sform->isValid($subformData)) {

494         if ($subformData['criteria']=='variable') {
            $stoWhere = parseData("value", $subformData['condition'], $subformData['value']);

            $select1 = Application_Model_Variable_Factory_Variable::getDbMapper()->getDbTable()
                ->select(Zend_Db_Table_Abstract::SELECT_WITH_FROM_PART);
            $select1->where("id = ?", $subformData['variable']);
499 $select1->where($stoWhere['condition'], $stoWhere['value']);
            $selectV[] = $select1;

            $select2 = Application_Model_Process_Factory_ProcessVariable::getDbMapper()->
                getDbTable()->select(Zend_Db_Table_Abstract::SELECT_WITH_FROM_PART);
            $select2->where("idVariable = ?", $subformData['variable']);
504 $select2->where($stoWhere['condition'], $stoWhere['value']);
            $selectPV[] = $select2;
        } else {
            $stoWhere = parseData($subformData['criteria'], $subformData['condition'],
                $subformData['value']);
            $selectProcess->where($stoWhere['condition'], $stoWhere['value']);
509     }
    }
    //We search the processes corresponding to these criterias
    $processes = Application_Model_Process_Factory_Process::fetchAll($selectProcess);
514 if (count($processes)>0) {
        $stoReturn = $processes;
    } else {
        $stoReturn = array();
    }

519 //We search the variables overloaded by these criterias and then the processes
    associated
    $processVars = array();
    foreach ($selectPV as $sel) {
        foreach (Application_Model_Process_Factory_ProcessVariable::fetchAll($sel) as $pvar)
524     {
        $p = $pvar->getProcess();
        if (!is_null($p) && !in_array($p, $processVars)) {
            $processVars[] = $p;
        }
    }
529 }
    if (count($processVars)>0) {
        if (count($stoReturn)>0) {
            $stoReturn = array_intersect($stoReturn, $processVars);
        } else {
534 $stoReturn = $processVars;
        }
    }

    //We search the variables filled by these criterias and then the tasks and the
    processes associated
539 $processTasks = array();
    foreach ($selectV as $sel) {
        foreach (Application_Model_Variable_Factory_Variable::fetchAll($sel) as $var) {
            $chain = $var->getChain();
            foreach (Application_Model_Process_Factory_Process::fetchAll(array('chain'=>$chain)
544 ) as $p) {
                if (!is_null($p) && !in_array($p, $processTasks)) {
                    $processTasks[] = $p;
                }
            }
        }
    }

```

```

    }
  }
549 }
    if (count($processTasks)>0) {
      if (count($toReturn)>0) {
        $toReturn = array_intersect($toReturn, $processTasks);
      } else {
554 $toReturn = $processTasks;
      }
    }

    $this->view->entries = $toReturn;
559 $this->render('list');

  }
} else {
564 unset($sessionNP->subforms);
}

$this->view->form = $form;
} catch (Exception $e) {
  throw $e;
569 }
}

/**
 * Show
 */
574 public function outputAction() {
  try {
    $this->_helper->layout->disableLayout();
    $this->_helper->viewRenderer->setNoRender();

579 $request = $this->getRequest();
    $process = Application_Model_Process_Factory_Process::find($request->process);
    $host = $process->getOutputHost();

584 $response = $this->getResponse();
    $response
      ->setHeader('Cache-Control', 'public', true)
      ->setHeader('Content-Description', 'File Transfer', true)
      ->setHeader('Content-Disposition', 'attachment; filename=' . basename($process->
589 getOutputPath()), true)
      ->setHeader('Content-Transfer-Encoding', 'binary', true)
      ->appendBody($host->receive($process->getUser(), $process->getOutputPath()));
  } catch (Exception $e) {
    throw $e;
594 }
}
}

```

C.7 Ressources, rôles et droits des utilisateurs

```

[ressources]
ajax = null
chain = null
file = null
5 host = null
  hostuser = null
  index = null
  jobcigrihost = null
  process = null
10 processtask = null
  processvariable = null
  project = null
  projecthost = null
  projectuser = null
15 task = null
  user = null
  variable = null

```

```
[roles]
20 guest = null
   member = null
   admin = null

[guest]
25 deny.ajax = null
   deny.chain = null
   deny.file = null
   deny.host = null
   deny.hostuser = null
30 deny.index = null
   deny.jobcigrihost = null
   deny.process = null
   deny.processtask = null
   deny.processvariable = null
35 deny.project = null
   deny.projecthost = null
   deny.projectuser = null
   deny.task = null
   deny.user = null
40 deny.variable = null

   allow.user = login

[member : guest]
45 allow.ajax = null
   allow.chain = index, show
   allow.file = index, show
   allow.host = index, show
   allow.hostuser = show
50 allow.index = null
   allow.jobcigrihost = index, show
   allow.process = index, show
   allow.processtask = index, show
   allow.processvariable = index, show
55 allow.project = index, show
   allow.projecthost = index, show
   allow.projectuser = show
   allow.task = index, show
   allow.user = index, login, show, logout
60 allow.variable = index, show

; admin herite de member
[admin : member]
65 allow.ajax = null
   allow.chain = null
   allow.file = null
   allow.host = null
   allow.hostuser = null
   allow.index = null
70 allow.jobcigrihost = null
   allow.process = null
   allow.processtask = null
   allow.processvariable = null
   allow.project = null
75 allow.projecthost = null
   allow.projectuser = null
   allow.task = null
   allow.user = null
   allow.variable = null
```

C.8 Structure XML pour la navigation

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <configdata>
    <nav>
      <project>
        <label>Layout.Menu.Project</label>
        <route>project</route>
        <resource>project</resource>
        <privilege>index</privilege>
        <pages>
          <list>
            <label>Layout.Menu.Project.List</label>
            <route>project-list</route>
            <action>list</action>
            <resource>project</resource>
            <privilege>index</privilege>
          </list>
          <new>
            <label>Layout.Menu.Project.New</label>
            <route>project-new</route>
            <action>new</action>
            <resource>project</resource>
            <privilege>new</privilege>
          </new>
          <edit>
            <label>Layout.Menu.Project.Edit</label>
            <route>project-edit</route>
            <action>edit</action>
            <resource>project</resource>
            <privilege>edit</privilege>
          </edit>
          <show>
            <label>Layout.Menu.Project.Show</label>
            <route>project-show</route>
            <action>show</action>
            <resource>project</resource>
            <privilege>show</privilege>
          </show>
        </pages>
      </project> <privilege>new</privilege>
      </new>
      <logout>
        <label>Layout.Menu.User.Logout</label>
        <controller>user</controller>
        <action>logout</action>
        <resource>user</resource>
        <privilege>logout</privilege>
      </logout>
    </pages>
  </user>
</nav>>
  <file>
    <label>Layout.Menu.File</label>
    <controller>file</controller>
    <resource>host</resource>
    <privilege>index</privilege>
    <pages>
      <list>
        <label>Layout.Menu.File.List</label>
        <controller>file</controller>
        <action>index</action>
        <resource>file</resource>
        <privilege>index</privilege>
      </list>
      <new>
        <label>Layout.Menu.File.New</label>
        <controller>file</controller>
        <action>new</action>
        <resource>file</resource>
        <privilege>new</privilege>
      </new>
    </pages>
  </file>
  <host>
    <label>Layout.Menu.Host</label>
```



```
76 <controller>host</controller>
<resource>host</resource>
<privilege>index</privilege>
<pages>
  <list>
81   <label>Layout.Menu.Host.List</label>
   <controller>host</controller>
   <action>index</action>
   <resource>host</resource>
   <privilege>index</privilege>
  </list>
86 <new>
   <label>Layout.Menu.Host.New</label>
   <controller>host</controller>
   <action>new</action>
   <resource>host</resource>
91   <privilege>new</privilege>
  </new>
</pages>
</host>
<user>
96 <label>Layout.Menu.User</label>
<controller>user</controller>
<resource>user</resource>
<privilege>index</privilege>
<pages>
101 <list>
   <label>Layout.Menu.User.List</label>
   <controller>user</controller>
   <action>index</action>
106 <resource>user</resource>
   <privilege>index</privilege>
  </list>
  <new>
   <label>Layout.Menu.User.New</label>
111 <controller>user</controller>
   <action>new</action>
   <resource>user</resource>
   <privilege>new</privilege>
  </new>
116 <logout>
   <label>Layout.Menu.User.Logout</label>
   <controller>user</controller>
   <action>logout</action>
   <resource>user</resource>
121 <privilege>logout</privilege>
  </logout>
</pages>
</user>
</nav>
</configdata>
```

Ajax (Asynchronous Javascript and XML), combinaison de technologies telles que Javascript, CSS, XML DOM et XMLHttpRequest permettant de réaliser es applications Web à l'ergonomie enrichie.

Atom format de document basé sur XML conçu pour la syndication de contenu périodique, tel que les blogs ou les sites d'actualités.

best-effort type de tâche autorisé à être tué par une ressource s'il faut laisser la priorité d'exécution aux tâches locales.

CSS (Cascading Style Sheet), langage informatique servant à décrire la présentation des documents HTML et XML.

DOM (Document Object Model), recommandation du W3C décrivant une interface standard permettant de manipuler une structure XML.

DTD (Document Type Defintion), document permettant de décrire une structure modèle de document SGML ou XML.

HTML (HyperText Markup Language), langage dérivé de SGML et conçu pour écrire des pages Web.

HTTP (HyperText Transfer Protocol), protocole de transfert de données hypertextes utilisées dans les sites Web.

intergiciel (plus connu sous le terme anglais « middleware »), logiciel tiers créant un réseau d'échange d'informations entre différentes applications informatiques.

JS voir Javascript.

OpenDocument format ouvert de données pour les applications bureautiques. OpenDocument est la désignation d'usage d'une norme dont l'appellation officielle est « OASIS Open Document Format for Office Applications », également abrégée par le sigle ODF.

PHP (PHP : Hypertext Preprocessor), langage de programmation interprété dont la syntaxe se rapproche du C, très utilisé pour le Web.

SAX (Simple API for XML), interface de programmation permettant de lire une structure XML.

SGML (Standard Generalized Markup Language), langage normalisé de balisage généralisé.

SQL (Structured Query Language), langage normalisé servant à effectuer des opérations sur des bases de données.

SSH (Secure Shell), protocole sécurisé de communication utilisé par le logiciel du même nom.

SSHFS (Secure shell file system), technologie permettant le montage de système de fichiers distant venant s'intégrer dans une arborescence via le protocole SSH.

SVG (Scalable Vector Graphics), format de données conçu pour décrire des ensembles de graphiques vectoriels et basé sur XML.

UML (Unified Modeling Language), langage de modélisation graphique à base de pictogrammes très utilisé en génie logiciel.

URI (Uniform Resource Identifier), courte chaîne de caractères identifiant une ressource physique ou abstraite sur un réseau et dont la syntaxe respecte une norme d'Internet mise en place pour le World Wide Web (voir RFC 3986).

UTF (Unicode Transformation Format), norme visant à attribuer à chaque caractère un identifiant numérique unique et indépendamment du système d'exploitation.

XHTML (eXtensible Hypertext Markup Language), langage de balisage servant à écrire des pages Web, dérivé de XML destiné à remplacer HTML.

XML (eXtensible Markup Language), langage extensible de balisage, très utilisé pour stocker ou transférer des données structurées en champs arborescents.

XSD (XML Schema), langage de description de format XML permettant de définir la structure et le type de contenu d'un document XML.

Bibliographie

- [1] DUTUIT, Odile *et al.*: *Laboratoire de Planétologie de Grenoble*. [en ligne]. <http://www-lpg.obs.ujf-grenoble.fr/>, visité le 1^{er} février 2011.
- [2] MONIN, Jean Louis *et al.*: *Institut de Planétologie et d'Astrophysique de Grenoble*. [en ligne]. <http://ipag.osug.fr>, visité le 1^{er} février 2011.
- [3] ESA: *ESA – Rosetta*. [en ligne]. <http://www.esa.int/esaMI/Rosetta>, visité le 1^{er} février 2011.
- [4] COTARDIERE (DE LA), Philippe et LEVASSEUR-REGOURD, Anny Chantal: *Les comètes et les astéroïdes*. Éditions du Seuil, Manecourt, collection points, série sciences édition, 1997.
- [5] UJF, Info Hebdo: *Mission ROSETTA : tout sur les comètes*. [en ligne], mars 2004. <http://www.obs.ujf-grenoble.fr/osug/content/view/53/>, visité le 1^{er} février 2011.
- [6] BENNA, Mehdi: *Génération et inversion de données de propagation d'ondes radio à travers un noyau cométaire (Simulation de l'expérience CONSERT)*. Thèse de doctorat, Université Paul Sabatier, Toulouse 3, 2002.
- [7] PIOT, Alexandre *et al.*: *Consert : Laboratoire de Planétologie de Grenoble*. [en ligne]. <http://consert-lpg.obs.ujf-grenoble.fr>, visité le 1^{er} février 2011.
- [8] PIOT, Alexandre: *Consert : sonder un noyau cométaire*. [en ligne]. <http://www.obs.ujf-grenoble.fr/osug/content/view/53/>, visité le 1^{er} février 2011.
- [9] JANEY, Nicolas: *Le rendu par lancer de rayons (ray tracing)*. [en ligne]. <http://raphaello.univ-fcomte.fr/Ig/RayTracing/LancerDeRayons.htm>, visité le 1^{er} février 2011.
- [10] BOLLARD, Philippe: *Étude des données de l'expérience CONSERT*. rapport technique, CNAM Grenoble, juin 2009.
- [11] CNES: *Le logiciel GINS, Formation GINS-PC, 25-26 octobre 2006*.
- [12] CNES: *Documentation algorithmique GINS*. [en ligne]. http://igsac-cnes.cls.fr/documents/gins/GINS_Doc_Algo_V4.html, visité le 1^{er} février 2011.
- [13] LASUE, Jeremie *et al.*: *Spherical harmonics decomposition : Application for CONSERT*. rapport technique, Laboratoire de Planétologie de Grenoble.
- [14] CHAILLOL, Stéphane: *CONSERT Meeting, Lindau, 12-13 February 2008*.

- [15] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. [en ligne]. <http://www.w3.org/TR/xml/>, visité le 1^{er} février 2011.
- [16] BRILLANT, Alexandre: *XML : Cours et exercices*. Eyrolles, Paris, 2007.
- [17] CHAGNON, Gilles et NOLOT, Florent: *XML*. Pearson Education France, Marsat, collection synthex édition, 2007.
- [18] STINNER, Victor: *Introduction à XML*. [en ligne]. <http://haypo.developpez.com/tutoriel/xml/introduction/>, visité le 1^{er} février 2011.
- [19] W3C: *XML Path Language (XPath) 2.0*. [en ligne]. <http://www.w3.org/TR/xpath20/>, visité le 1^{er} février 2011.
- [20] W3C: *Document Object Model (DOM) Level 1 Specification*. [en ligne]. <http://www.w3.org/TR/REC-DOM-Level-1/>, visité le 1^{er} février 2011.
- [21] MEGGINSON, David: *SAX*. [en ligne]. <http://www.saxproject.org/>, visité le 1^{er} février 2011.
- [22] W2SCHOOLS: *DTD Tutorial*. [en ligne]. <http://www.w3schools.com/dtd/default.asp>, visité le 1^{er} février 2011.
- [23] W3C: *W3C XML Schema Definition Language (XSD) 1.1 Part 1 : Structures*. [en ligne]. <http://www.w3.org/TR/xmlschema11-1/>, visité le 1^{er} février 2011.
- [24] VLIST, Eric VAN DER: *XML.com : Using W3C XML Schema*. [en ligne]. <http://xml.com/pub/a/2000/11/29/schemas/part1.html>, visité le 1^{er} février 2011.
- [25] COMMENTCAMARCHE: *Bases de données – Introduction*. [en ligne]. <http://www.commentcamarche.net/contents/bdd/bddintro.php3>, visité le 1^{er} février 2011.
- [26] WIKIPEDIA: *SQLite - Wikipédia*. [en ligne]. <http://fr.wikipedia.org/wiki/SQLite>, visité le 1^{er} février 2011.
- [27] HIPPI, Richard: *SQLite Home Page*. [en ligne]. <http://sqlite.org/>, visité le 1^{er} février 2011.
- [28] INTEL: *Perspectives de la loi de Moore : technologie et recherche chez Intel*. [en ligne]. <http://www.intel.com/cd/corporate/techtrends/emea/fra/209837.htm>, visité le 1^{er} février 2011.
- [29] BZEZNIK, Bruno: *Computing grids*. [en ligne], octobre 2009. https://ciment.ujf-grenoble.fr/docs/presentations-de-oar-et-cigri/computing_grids_2009-10-05.pdf, visité le 1^{er} février 2011.
- [30] WIKIPEDIA: *Grille informatique - Wikipédia*. [en ligne]. http://fr.wikipedia.org/wiki/Grille_informatique, visité le 1^{er} février 2011.
- [31] DROTHIER, Yves: *Découvrir le monde des serveurs*. [en ligne], février 2006. <http://www.journaldunet.com/solutions/dossiers/pratique/serveurs.shtml>, visité le 1^{er} février 2011.
- [32] CHALJUB, Emmanuel et al.: *Calcul Intensif / Modélisation / Expérimentation Numérique et Technologique - Site du projet CIMENT*. [en ligne]. <https://ciment.ujf-grenoble.fr/>, visité le 1^{er} février 2011.
- [33] *CIMENT (Calcul Intensif, Modélisation, Expérimentation Numérique - [Groupe Calcul]*. [en ligne]. <http://calcul.math.cnrs.fr/spip.php?article47>, visité le 1^{er} février 2011.
- [34] CHAILLOL, Stéphane: *CONSERT Meeting, London, 16-17 February 2009*.
-

-
- [35] GRID5000: *Grid5000 :Home - Grd5000*. [en ligne]. <https://www.grid5000.fr>, visité le 1^{er} février 2011.
- [36] EGI: *EGI*. [en ligne]. <http://www.egi.eu>, visité le 1^{er} février 2011.
- [37] EGEE: *EGEE Portal : Enabling Grids for E-sciencE*. [en ligne]. <http://www.eu-egee.org>, visité le 1^{er} février 2011.
- [38] OAR-TEAM: *OAR*. [en ligne]. <http://oar.imag.fr>, visité le 1^{er} février 2011.
- [39] BZEZNIK, Bruno, CAVAGNA, Romain, EMERAS, Joseph et RICHARD, Olivier: *OAR : un gestionnaire de ressources pour grandes grappes de calcul*. [en ligne], décembre 2009. <https://ciment.ujf-grenoble.fr/docs/presentations-de-oar-et-cigri/article.odt>, visité le 1^{er} février 2011.
- [40] BZEZNIK, Bruno et DESBAT, Laurent: *CIMENT - Un regroupement de pôles meso-info mutualisés par une grille légère CIGRI*. [en ligne], février 2008. https://ciment.ujf-grenoble.fr/docs/presentations-de-oar-et-cigri/cigri-2008-02_journee_meso.pdf, visité le 1^{er} février 2011.
- [41] CERN: *gLite - Lightweight Middleware for Grid Computing*. [en ligne]. <http://glite.cern.ch>, visité le 1^{er} février 2011.
- [42] GLOBUS: *The Globus Alliance*. [en ligne]. <http://www.globus.org/>, visité le 1^{er} février 2011.
- [43] NORDUGRID: *NorduGrid | Advanced Resource Connector*. [en ligne]. <http://www.nordugrid.org/arc>, visité le 1^{er} février 2011.
- [44] DIET: *DIET - DIET*. [en ligne]. <http://graal.ens-lyon.fr/DIET>, visité le 1^{er} février 2011.
- [45] GECLIPSE: *g-Eclipse - Access the Grid : Home*. [en ligne]. <http://www.geclipse.eu>, visité le 1^{er} février 2011.
- [46] WIKIPEDIA: *PHP - Wikipédia*. [en ligne]. <http://fr.wikipedia.org/wiki/Php>, visité le 1^{er} février 2011.
- [47] ARCHOUR, Mehdi, BRIET, Vincent, SEGUY, Damien *et al.*: *PHP : Manuel PHP*. [en ligne]. <http://fr.php.net/manual/fr/>, visité le 1^{er} février 2011.
- [48] ZEND: *Zend Framework Documentation : Zend Framework Manual*. [en ligne]. <http://framework.zend.com/manual/en>, visité le 1^{er} février 2011.

MÉMOIRE D'INGÉNIEUR C.N.A.M. en INFORMATIQUE

Conception et réalisation d'une interface unifiée et automatisée pour la plateforme de simulation de l'expérience CONSERT.

Philippe BOLLARD

Grenoble, le 8 avril 2011

RÉSUMÉ

Dans le cadre de la mission spatiale ROSETTA, l'instrument CONSERT permettra de déterminer la composition interne du noyau de la comète 67P/Churyumov-Gerasimenko. Une plateforme de simulation de cette expérience a été développée par le Laboratoire de Planétologie de Grenoble. Elle est composée de deux logiciels manipulant des données numériques complexes et volumineuses.

La première partie de ce stage concerne l'étude des données de l'expérience CONSERT. De nouvelles structures basées sur XML sont proposées afin d'améliorer l'accessibilité et la pérennité des informations. Différentes méthodes de lecture sont mises en œuvre et testées.

La suite du stage concerne l'automatisation de la plateforme de simulation. En effet, pour lancer une simulation, il est nécessaire de modifier manuellement plusieurs scripts afin d'utiliser la grille de calcul CIMENT et son intergiciel CiGri. Ce mémoire présente la réalisation de l'outil Web conçu autour du cadre Zend Framework. Cet outil générique, multi-utilisateurs et multi-projets offre une interface permettant de définir des séquences de tâches paramétrables. En quelques sélections, il est désormais possible d'interagir avec la grille pour soumettre et gérer les simulations.

MOTS CLÉS : CONSERT, CIMENT, CiGri, Zend Framework, plateforme, grille, grappe, calcul intensif, simulation, XML, PHP, SSH.

ABSTRACT

As part of ROSETTA, the mission to space, the device called CONSERT will allow to determine the inner composition of the core of Comet 67P/Churyumov-Gerasimenko. A simulation of this experiment was developed by the Laboratory of Planetology of Grenoble. The platform consists of two softwares handling complex and bulky digital data.

The first part of this study is focused on data used by the CONSERT experience. New XML patterns are proposed to improve the accessibility and continuity of data. Different parsing methods are implemented and tested.

The next part is focused on the automation of the simulation platform. Indeed, in order to run a simulation, it is necessary to manually edit several scripts before using the CIMENT grid computing and its CiGri middleware. This study presents the development of a Web software designed around the Zend Framework. This generic tool, multi-user and multi-project, provides a graphical interface which allows the definition of customizable task sequences. With a few selections, it is now possible to interact with the grid in order to submit and manage the simulations.

KEYWORDS : CONSERT, CIMENT, CiGri, Zend Framework, platform, grid, cluster, super-computing, simulation, XML, PHP, SSH.