Titre: Title:	Projet D2G2 : prototype d'optimisation de la chaîne de calcul dragon-donjon
Auteur: Author:	Pascal Rhéaume
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Rhéaume, P. (2005). Projet D2G2 : prototype d'optimisation de la chaîne de calcul dragon-donjon [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7534/

UNIVERSITÉ DE MONTRÉAL

PROJET D2G2 : PROTOTYPE D'OPTIMISATION DE LA CHAÎNE DE CALCUL DRAGON-DONJON

PASCAL RHÉAUME
DÉPARTEMENT DE GÉNIE PHYSIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉNERGÉTIQUE)
AOÛT 2005

Your file Votre référence
ISBN: 978-0-494-16846-2
Our file Notre référence
ISBN: 978-0-494-16846-2

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

PROJET D2G2 : PROTOTYPE D'OPTIMISATION DE LA CHAÎNE DE CALCUL DRAGON-DONJON

présenté par: RHÉAUME Pascal
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. GAGNON Michel, Ph.D., président
M. KOCLAS Jean, Ph.D., membre et directeur de recherche
M. ROY Robert, Ph.D., membre et codirecteur de recherche
Mme. VARIN Élisabeth, D.Sc., membre

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche Robert Roy et mon co-directeur Jean Koclas qui m'ont dirigé tout au long de ce travail. Je remercie aussi Elizabeth Varin pour le précieux temps qu'elle a bien voulu m'accorder. Finalement, j'aimerais exprimer ma gratitude envers l'Institut de génie nucléaire de l'École Polytechnique de Montréal pour m'avoir acceuilli au sein du Groupe d'analyse neutronique (GAN) et envers Hydro-Québec pour m'avoir soutenu financièrement.

RÉSUMÉ

De nos jours, plusieurs logiciels de haute qualité simulant le flux neutronique d'un réacteur nucléaire sont disponibles. Les codes DRAGON-DONJON, développés à l'Institut de génie nucléaire de l'École Polytechnique de Montréal constituent justement un bon exemple. Ces applications sont gratuites et leur exactitude a depuis fort longtemps été démontrée. Cependant, comme la plupart des codes neutroniques d'aujourd'hui, ils possèdent les lacunes suivantes : piètre utilisabilité, mauvaise maintenabilité et portabilité déficiente. Le projet D2G2 se veut donc une initiative de développement logiciel permettant d'éliminer ces faiblesses en proposant une nouvelle approche informatique pour optimiser la chaîne de calcul DRAGON-DONJON.

Plus spécifiquement, les objectifs du projet D2G2 se résument à l'utilisation de l'approche orientée-objet pour moderniser l'architecture globale de la chaîne de calcul, au développement d'un pilote de contrôle pour automatiser les codes de simulation DRAGON-DONJON, à la gestion efficace du stockage des paramètres d'entrées et des résultats, à la standardisation des échanges de données et finalement à l'ajout d'une interface graphique.

Or, dans le cadre de ce travail, une architecture du type tableau noir implémentée à l'aide d'un modèle client-serveur a donné naissance à deux applications. La première, nommée D2G2Client, est l'interface graphique avec laquelle l'utilisateur interagit pour soumettre ses calculs neutroniques. Elle permet entre autres d'élaborer des requêtes de simulation, de suivre l'évolution des calculs et de naviguer le tableau noir pour visualiser les résultats. La deuxième, nommée D2G2Server, est quant à elle le gestionnaire du tableau noir. Elle s'occupe de répertorier et d'analyser les requêtes de simulation provenant du D2G2Client, de manipuler les processus DRAGON et DONJON, de sauvegarder les données de calculs et finalement de fournir les résultats aux

clients. Le D2G2Server est en fait un serveur multitâche, basé sur un modèle d'accès simultané, élaboré à partir du cadre de travail ACE (Adaptive Communication Environment).

Finalement, il est important de souligner que les deux applications développées, bien qu'elles ont été validées en déterminant le taux de combustion moyen des grappes de combustible à la sortie du réacteur lorsque celui-ci est à l'équilibre du rechargement, demeurent tout de même de simples prototypes. Plusieurs fonctionnalités doivent être ajoutées et certaines limitations corrigées. Par contre, l'architecture proposée pour l'optimisation de la chaîne de calcul DRAGON-DONJON est totalement nouvelle et facilement réutilisable dans divers contextes. Il serait alors formidable, d'ici quelques années, de constater que le projet D2G2 fut l'un des précurseurs de l'innovation informatique concernant le paradigme des méthodes de calcul neutronique.

ABSTRACT

Nowadays, high quality neutronic simulation codes are readily available. The open source software suite DRAGON-DONJON is precisely a good example. It is free, it has proven quality and correctness over the years and is still developed and maintained at l'Institut de génie nucléaire de l'École Polytechnique de Montréal. However, like most simulation codes, it has the following weaknesses : limited usability, poor maintainability, no internal data standardization and poor portability. The D2G2 project is a software development initiative which aims to create an upper layer software tool that annihilates the weakness of classic simulation codes.

More specifically, the objectives of D2G2 project sum up in the use of an object-oriented approach to modernize the computational architecture of DRAGON and DONJON, in the development of a control pilot to automate neutronic tools, in an efficient management of simulation inputs and results, in the standardization of data exchanges and finally in a graphical user interface add-on.

In fact, during this project, a blackboard pattern implemented using a client-server model gave rise to two applications. The first one, named D2G2Client, is a graphical interface which acts as a knowledge source. Indeed, it grants user the possibility to submit neutronic jobs, follow simulation progress and display results. The second application, called D2G2Server, is the blackboard controller. It is used to analyse D2G2Client requests, handle DRAGON and DONJON processes, save results and provide simulation data to clients. Technically, the D2G2Server is a multitask server based on a model of simultaneous access developed using the ACE (Adaptive Communication Environment) framework.

Finally, it is important to specify that these two applications, although they were validated by determining the time-average discharge burnup rate of fuel bundles at nominal operating conditions, are just prototypes. Several functionalities must be added and some limitations must be corrected. Nevertheless, the architecture proposed by D2G2 project to modernize legacy simulation codes is completely new and easily reusable in various contexts. So, maybe the D2G2 project will be one day recognized as one of the precursors which begin to enhance the computational paradigm of neutronic methods.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

INTRODUCTION

De nos jours, plusieurs logiciels de haute qualité simulant le flux neutronique d'un réacteur nucléaire sont disponibles. Les codes DRAGON [1] et DONJON [2], développés à l'Institut de génie nucléaire de l'École Polytechnique de Montréal, constituent justement un bon exemple. Ces applications sont gratuites et leur exactitude a depuis fort longtemps été démontrée. Cependant, comme la plupart des codes neutroniques d'aujourd'hui, ils possèdent les lacunes suivantes : piètre utilisabilité, mauvaise maintenabilité et portabilité déficiente. Le projet D2G2[1] se veut donc une initiative de développement logiciel permettant d'éliminer ces faiblesses en proposant une nouvelle approche informatique pour optimiser la chaîne de calcul DRAGON-DONJON.

Plus spécifiquement, les objectifs du projet D2G2 se résument à l'utilisation de l'approche orientée-objet pour moderniser l'architecture globale de la chaîne de calcul, au développement d'un pilote de contrôle pour automatiser les codes de simulation neutronique, à la gestion efficace du stockage des paramètres d'entrées et des résultats, à la standardisation des échanges de données afin d'éventuellement utiliser des modèles internationaux et finalement, à l'ajout d'une interface graphique pour la préparation des requêtes de simulation et la visualisation des résultats.

Bref, il est clair que l'optimisation des fonctionnalités de la chaîne de calcul DRAGON-DONJON est l'objectif principal du projet. Par contre, comme les outils développés dans le cadre de ce travail sont conçus pour s'adapter à la plupart des applications de neutronique, leur élaboration permet aussi de prouver qu'il est possible de marier des codes patrimoniaux aux techniques modernes de l'informatique. Donc, la contribution que le projet D2G2 désire apporter dépasse largement les commodités découlant de l'automatisation de l'exécution de DRAGON-DONJON, mais touche davantage au

[1] Acronyme découlant de Dragon Donjon Génération 2

paradigme dans lequel s'effectuent la plupart des développements logiciels de neutronique. En effet, il est de plus en plus urgent de moderniser les approches de conception en génie nucléaire afin de captiver la nouvelle génération de scientifiques ; beaucoup plus familière, voire même dépendante, des interfaces graphiques et des commodités de communication réseau [3].

Justement, plusieurs domaines scientifiques tels que la biotechnologie et la médecine ont déjà développé des applications offrant des interfaces graphiques et de nombreuses possibilités de visualisation. L'industrie nucléaire, qui a aussi grandement besoin de profiter de ces fonctionnalités graphiques, doit dès maintenant emboîter le pas et développer activement ces technologies [4]. Or, ces dernières années, seulement quelques outils de visualisation ont vu le jour. Par exemple, MCNP Visual Editor [5] et Moritz [6] sont deux applications fréquemment utilisées avec le code de simulation MCNP[2]. Par contre, il faut souligner que le développement des ces modules de visualisation est effectué parallèlement à ceux des codes de neutronique. Ce qui rend difficile la maintenance et l'ajout de nouvelles fonctionnalités. Il est alors essentiel qu'à l'avenir le développement des fonctionnalités graphiques et celui des codes de neutronique s'effectuent conjointement au sein de la même équipe. C'est pourquoi le projet D2G2 se doit d'intégrer des interfaces graphiques et des techniques modernes de visualisation.

Maintenant, en ce qui a trait à la structure du présent ouvrage, celui-ci est divisé en 6 chapitres. Le premier est une brève introduction à la modélisation numérique du CANDU-6. Il passe en revue quelques notions théoriques de la neutronique et aborde la chaîne de calcul DRAGON-DONJON. Le suivant expose les assises du projet D2G2. Il décrit la spécification des besoins à partir des requis de l'industrie et de certains travaux antérieurs. Le chapitre 3 traite quant à lui de la modélisation

[2]http ://www-xdiv.lanl.gov/x5/MCNP

par objet. C'est-à-dire, il présente sous forme graphique une description des objets et certaines de leurs interactions à l'aide de la notation UML. Vient ensuite le chapitre 4, pouvant être considéré comme le coeur de l'ouvrage. En effet, c'est dans de ce chapitre qu'est détaillée les deux applications informatiques réalisées au cours de ce projet. La conception du D2G2Server y est abordée et les fonctionnalités du D2G2Client sont présentées. Le chapitre 5 est pour sa part consacré à la validation des outils informatiques développés. Plus précisément, le chapitre 5 se charge de vérifier l'exactitude des résultats retournés par le D2G2Server à la suite d'une requête de simulation de gestion du combustible envoyée par le D2G2Client. Finalement, le dernier chapitre expose les limites du projet D2G2 et suggère certaines améliorations et travaux futurs.

CHAPITRE 1

MODÉLISATION NUMÉRIQUE DU CANDU-6

Ce chapitre aborde certains concepts de base liés à la modélisation numérique du CANDU-6 et effectue un survol de la chaîne de calcul DRAGON-DONJON. Il se veut aussi une brève introduction aux techniques utilisées lors de calculs neutroniques. Ce chapitre n'est cependant pas une revue exhaustive de DRAGON et DONJON et de la neutronique. Il désire seulement mettre en contexte les notions nécessaires à la compréhension du reste de l'ouvrage[1]. En effet, il est primordial de définir la structure du processus de simulation d'un réacteur CANDU-6 afin de bien identifier le rôle des divers modules de la chaîne de calcul au coeur du projet D2G2.

Or, avant d'aborder chacune des étapes de modélisation d'un réacteur CANDU-6, il est requis de préalablement effectuer une brève description des principales caractéristiques physiques de celui-ci. En effet, il est plus aisé de comprendre les notions de calcul de cellule et de supercellule en présentant préalablement une vue d'ensemble du réacteur.

1.1 Description générale du réacteur

Le réacteur CANDU-6 (figure 1.1) est un réacteur nucléaire à tubes de force, utilisant de l'uranium naturel comme combustible et de l'eau lourde comme modérateur. Le réacteur consiste principalement en une cuve cylindrique (calandre) dont le dia-

[1]Les lecteurs ayant déjà des connaissances en simulation neutronique et maîtrisant DRAGON et DONJON peuvent immédiatement passer au chapitre suivant où les assises du projet D2G2 sont abordées.

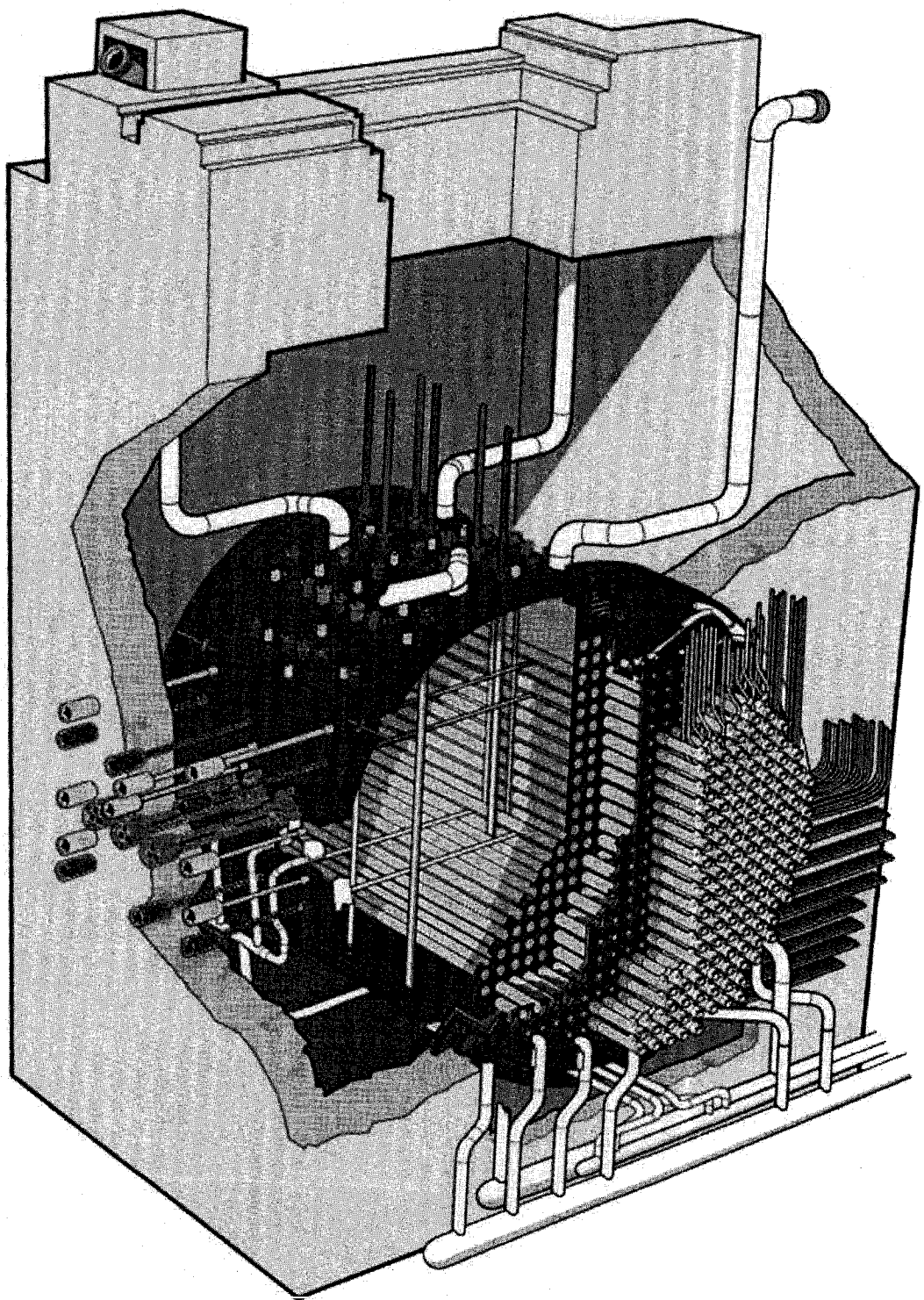

Figure 1.1 Schéma d'un réacteur CANDU-6

mètre et la longueur sont 7.6m et 6.0m respectivement. Cette cuve, remplie d'un modérateur sous forme d'eau lourde non pressurisée, est traversée axialement par un ensemble de 380 canaux (tubes de calandre) contenant les canaux de combustible (tubes de force en $Zr - Nb$). Chaque canal de combustible renferme quant à lui un ensemble de 12 grappes refroidies par un caloporteur d'eau lourde pressurisée et est isolé thermiquement du tube de calandre par un espace annulaire rempli de dioxyde de carbone. Finalement, chaque grappe de combustible est formée de 37 crayons (modèle CANDU-6) ou 43 crayons (modèle CANFLEX) gainés en Zircaloy-II et contient de l'oxyde d'uranium naturel sous forme de pastille.

Cette simple description du réacteur CANDU-6 fait ressortir une caractéristique importante de son design : celui-ci possède une séparation physique entre le caloporteur et le modérateur lui permettant de maintenir son circuit caloporteur à une pression élevée de 10MPa sans toutefois que la cuve soit elle sous pression. Le modérateur, baignant toute la cuve, est ainsi pratiquement maintenu à pression atmosphérique et sa température demeure en tout temps relativement basse ($\tilde{}70°C$). Or, cette séparation influence avantageusement le mode opérationnel du réacteur du point de vue de la gestion du combustible. Elle permet au CANDU d'être rechargé en marche ; évitant les arrêts prolongés caractéristiques de la plupart des autres réacteurs actuellement en opération dans le monde. Cette notion de rechargement en marche sera justement prise en considération lors de la validation des outils du projet D2G2. En effet, une modélisation simplifiée du réacteur découle directement de la possibilité d'effectuer des calculs où le combustible est à l'équilibre du rechargement.

1.2 Mécanismes de contrôle

Pour compléter toute description générale d'un réacteur, il est nécessaire de présenter les mécanismes de contrôle des réactions nucléaires en chaîne. En effet, toute simu-

lation numérique doit prendre en considération les différents mécanismes de contrôle assurant une distribution de puissance adéquate dans le réacteur. Leur modélisation étant effectivement primordiale pour étudier les différents phénomènes rendant le coeur sur-critique ou sous-critique.

Or, concernant le CANDU-6, le système de contrôle de réactivité est divisé en deux sous-systèmes indépendants : le système de régulation et le système d'arrêt d'urgence. Le système de régulation est composé de 14 contrôleurs liquides, 21 barres de compensation, 4 barres solides de réglage et d'un mécanisme d'addition lente de poison. Le système d'arrêt d'urgence est quant à lui composé de 28 barres d'arrêt au cadmium et de 6 tubulures d'injection rapide de gadolinium au modérateur.

Régulation

Étant donné que seul le système de régulation fait habituellement partie intégrante des simulations neutroniques d'un réacteur, une description plus détaillée de ces composantes est essentielle. Or, la régulation est assurée grâce aux lectures de flux provenant d'une panoplie de détecteurs disposés dans différents emplacements du coeur ; ces mesures étant utilisées par les divers algorithmes implémentés dans les ordinateurs de contrôle régissant le comportement des mécanismes de régulation.

- 14 contrôleurs liquides

  Les contrôleurs liquides sont des réservoirs cylindriques (barres liquides) disposés dans 14 zones distinctes du réacteur. Ils contiennent principalement de l'eau légère et peuvent varier leur niveau afin de maintenir constante la puissance globale dans le coeur. Elles permettent en effet de réduire les changements prononcés du flux entre les zones, par une variation continuelle de leur volume d'eau. Leur modélisation numérique doit faire particulièrement attention à leur description géométrique afin de pouvoir déterminer adéquatement le volume

d'eau qu'elles contiennent.

- 21 barres de compensation

  Les 21 barres de compensation sont faites d'acier ou de cobalt. Leur principale tâche est d'aplatir le flux neutronique et de compenser les effets du xénon. Normalement pleinement insérées dans le coeur, elles peuvent être extraites individuellement ou par groupe, dépendamment des consignes des ordinateurs de contrôle. Leur modélisation numérique doit prendre en considération plusieurs de leurs propriétés physiques (densité, volume, composition isotopique) ainsi que leur disposition et leur évolution spatiale.

- 4 barres solides

  Les 4 barres dites solides sont faites d'acier ou de cadmium. Elles sont habituellement hors du coeur et sont insérées lorsque certaines conditions opérationnelles très spécifiques sont présentes. Tout comme les barres de compensation, leur modélisation numérique doit prendre en considération plusieurs de leurs propriétés physiques (densité, volume, composition isotopique) ainsi que leur disposition et leur évolution spatiale. Par contre, comme ces barres sont habituellement retirées du coeur, leur modélisation est souvent ignorée.

- Bore et gadolinium

  L'ajout de bore et de gadolinium au modérateur permet un contrôle global de la réactivité du coeur, nécessaire lors de certaines manipulations de montée en puissance ou bien de gestion du combustible. La modélisation numérique de ce mécanisme est simple. Il s'agit de seulement varier la quantité de ces poisons solubles dans le modérateur.

- 28 détecteurs au platine

  La présence de 28 détecteurs au platine permet la régulation du mouvement des barres liquides. Assemblés par paires pour assurer une intégrité de leur

lecture, ces détecteurs fournissent des lectures du flux dans 14 zones distinctes du réacteur. Leur modélisation est nécessaire pour permettre une comparaison valable des valeurs simulées avec celles mesurées en centrale.

- 102 détecteurs au vanadium

  Les détecteurs au vanadium sont quant à eux distribués partout dans le coeur. Ils ont comme tâche de fournir 102 lectures au programme de cartographie du flux : nécessaire en particulier à la gestion du combustible. Tout comme les détecteurs au platine, leur modélisation permet une comparaison valable des valeurs simulées avec celles mesurées en centrale.

- Chambres d'ions

  Les chambres d'ions offrent de leur côté des mesures lorsque le réacteur est à faible puissance. En effet, grâce à leur réponse instantanée, elles permettent un suivi très précis du taux de variation logarithmique. Leur modélisation est souvent ignorée puisque généralement les calculs de réacteur ne traitent pas les montées en puissance.

Arrêt d'urgence

Les systèmes d'arrêt d'urgence, étant entièrement indépendants l'un de l'autre et complètement indépendants du système de régulation, sont habituellement ignorés de la plupart des simulations neutroniques de modes opérationnels. Ceux-ci sont cependant amplement étudiés lors d'analyses de sûreté. Donc, dans le cadre de cet ouvrage, leur modélisation a été mise à l'écart. Seul le système de régulation est pris en considération. Par contre, l'ajout de ces systèmes d'arrêt d'urgence, dans l'éventualité de l'utilisation des outils D2G2 pour simuler des accidents tels qu'une perte rapide de caloporteur (LOCA), est envisageable.

1.3 Calcul de cellule

Le but visé d'un calcul neutronique d'un réacteur complet est la détermination du flux neutronique du coeur. En effet, grâce à cette cartographie du flux, il est possible de suivre l'évolution des propriétés nucléaires du combustible, de déterminer la distribution de puissance, les profils de température et plusieurs autres paramètres opérationnels. Or, pour obtenir le champ des neutrons dans un réacteur nucléaire, l'approche de base consiste à effectuer un bilan de ces particules à l'intérieur d'un élément de volume élémentaire de l'espace de phase. La quantité fondamentale à déterminer étant la densité angulaire des neutrons $n(\vec{r}, E, \vec{\Omega}, t)$ définie de telle sorte $n(\vec{r}, E, \vec{\Omega}, t) d^3r d^2\Omega dE$ représente le nombre de neutrons au temps t à l'intérieur du volume élémentaire[2] d^3r entourant le point $\vec{r}$, dans un intervalle d'énergie dE autour de E et d'angle solide $d^2\Omega$ entourant la direction $\vec{\Omega}$. Bref, de cette approche découle l'équation de transport de Maxwell-Boltzmann [7] décrivant le comportement neutronique en fonction du temps :

$$\begin{aligned} \frac{1}{v}\frac{\partial}{\partial t}\Phi(\tilde{\mathbf{r}}, E, \vec{\Omega}, t) &= -\Sigma(\vec{r}, E, t)\,\Phi(\vec{r}, E, \vec{\Omega}, t) - \vec{\Omega}\cdot\vec{\nabla}\Phi(\vec{r}, E, \vec{\Omega}, t) \\ &\quad + \int_0^\infty dE' \int_{4\pi} d^2\Omega'\, g(\vec{r}; E' \to E, \vec{\Omega}' \to \vec{\Omega})\,\Sigma(\vec{r}, E', t)\Phi(\vec{r}, E', \vec{\Omega}', t) \\ &\quad + q(\vec{r}, E', \vec{\Omega}', t) \end{aligned} \tag{1.1}$$

où les différents termes sont :

- $\Phi(\vec{r}, E, \vec{\Omega}, t) = v\, n(\vec{r}, E, \Omega, \vec{t})$: la densité de flux angulaire où v est la vitesse
- $\Sigma(\vec{r}, E, t)$: la section efficace macroscopique totale
- $g(\vec{r}; E' \to E, \vec{\Omega}' \to \vec{\Omega})$: la section efficace différentielle
- $q(\vec{r}, E', \vec{\Omega}', t)$: une source de neutrons indépendante de la densité neutronique

[2] Ce volume élémentaire est souvent utilisé sous l'appellation volume de contrôle

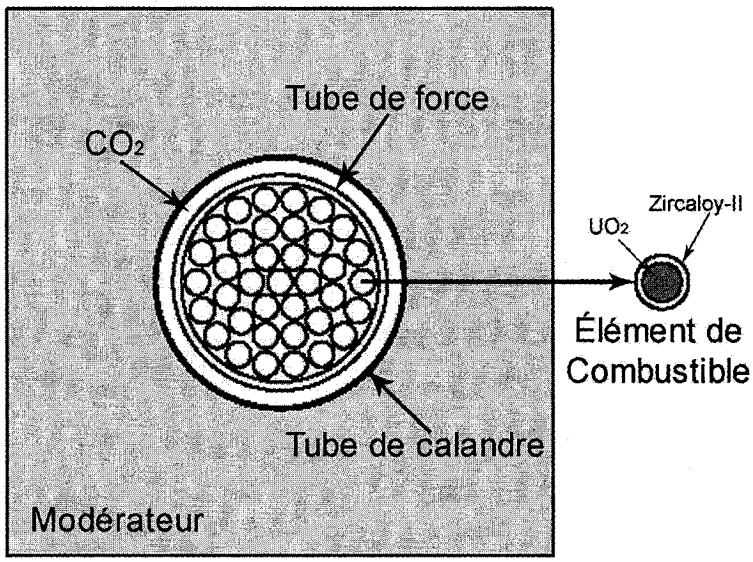

Figure 1.2 Cellule unitaire

Par contre, même s'il est possible d'obtenir une telle équation, il est malheureusement impossible de la résoudre analytiquement. Il faut en effet recourir à des méthodes numériques pour arriver à extraire le flux de cette équation à 7 dimensions. Cependant, même numériquement, l'équation 1.1 est pour le moment très difficile à traiter étant donnée la complexité de la géométrie et des matériaux d'un réacteur nucléaire. Même les plus puissantes ressources informatiques actuelles ne permettent pas de traiter le problème. Il est alors d'usage de découper le réacteur en cellules unitaires pour arriver à résoudre le coeur dans son entier, où chaque cellule unitaire est définie à partir du réseau de canaux de combustible. Une cellule contenant ainsi une seule grappe de combustible entourée de son tube de force, d'un espacement de dioxyde de carbone et du modérateur (figure 1.2). Plus précisément, une cellule possède une longueur de 49.54 cm (longueur d'une grappe), 28.575 cm de largeur et 28.575 cm de hauteur. Le but premier de cette division cellulaire étant d'obtenir les propriétés nucléaires homogénéisées en espace et condensées en énergie sur la cellule, à partir des paramètres locaux (densité de puissance, température du combustible, densité du ca-

loporteur, etc.). Il est ensuite possible d'effectuer des approximations valables sur ces géométries cartésiennes à paramètres constants telles que celles menant à l'équation de diffusion.

1.4 Calcul de supercellule

Par contre, avant d'entamer la résolution du coeur à l'aide de l'équation de diffusion, la connaissance des paramètres des cellules unitaires n'est pas suffisante. Il faut effectivement ajouter l'influence provenant du comportement des systèmes de contrôle de la réactivité décrits précédemment. En effet, il est clair que ces mécanismes jouent un rôle déterminant dans le fonctionnement du réacteur puisqu'ils ont justement été conçus pour influencer le bilan neutronique. Il est donc important de calculer leur impact en fonction de leur position dans le coeur. Par exemple, lorsqu'une barre de compensation est insérée dans le réacteur, elle provoque un changement des sections efficaces de toutes les cellules qu'elle traverse.

Or, comme il faut être en mesure de déterminer précisément les propriétés nucléaires des cellules unitaires, y compris celles influencées par les mécanismes de réactivité, pour parvenir à effectuer un calcul précis du flux, les sections efficaces des différents mécanismes (barres de compensation, contrôleurs liquides, structures, etc.) doivent être soigneusement calculées. Par contre, contrairement au calcul de transport des cellules unitaires, la détermination des paramètres nucléaires des mécanismes de contrôle doit impérativement être effectuée en trois dimensions. En effet, étant donné que ces mécanismes sont situés perpendiculairement aux canaux de combustible (figure 1.3), l'équation de transport doit être résolue sur une supercellule couvrant les 3 dimensions de l'espace.

La méthode de calcul pour y parvenir se résume à résoudre l'équation de transport

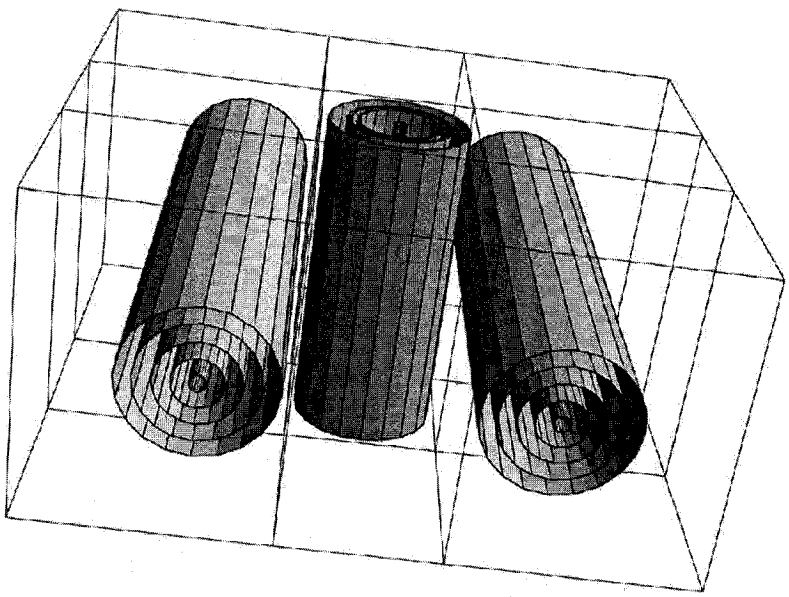

Figure 1.3 Supercellule

sur la supercellule avec le mécanisme en position souhaitée et sans le mécanisme. Les paramètres nucléaires du mécanisme étant obtenus en comparant les résultats des deux calculs. Par exemple, pour déterminer l'influence d'une barre de compensation, un calcul de supercellule est effectué avec la barre pleinement insérée dans la cellule et un deuxième sans la barre. Les sections efficaces de la barre correspondant alors aux incréments découlant de la différentiation des résultats avec et sans la barre. Finalement, suivant cette même procédure avec tous les mécanismes de contrôle, il suffit alors d'ajouter aux sections efficaces des cellules, les sections efficaces des mécanismes qui les traversent.

1.5 Calcul de réacteur

Tel que spécifié précédemment, une fois l'évaluation des différentes propriétés nucléaires de toutes les cellules unitaires effectuée, il est possible de procéder à certaines approximations permettant de résoudre le champ neutronique sur tout le réacteur.

En effet, différentes approximations (anisotropie linéaire du flux, isotropie des sources de neutrons, densité de neutron variant lentement par rapport à la fréquence de collisions [8]) permettent de simplifier considérablement l'équation de transport. L'équation résultante étant l'équation de diffusion :

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\phi(\vec{r},E,t) &= -\Sigma(\vec{r},E,t)\,\phi(\vec{r},E,t) - \vec{\nabla}\cdot D(\vec{r},E)\,\vec{\nabla}\phi(\vec{r},E,t) \\ &\quad + \int_0^\infty dE'\,\Sigma_s(\vec{r};E'\rightarrow E,t)\,\phi(\vec{r},E',t) \\ &\quad + \chi_p(E)\int_0^\infty dE'\,\nu_p\Sigma_f(\vec{r},E',t)\,\phi(\vec{r},E',t) \\ &\quad + S_d(\vec{r},E,t) + S(\vec{r},E,t) \qquad (1.2)\end{aligned}$$

où les nouveaux termes par rapport à l'équation 1.1 sont :

- $\phi(\vec{r},E,t)$: la densité de flux scalaire
- $D(\vec{r},E)$: le coefficient de diffusion
- $\Sigma_s(\vec{r},E,t)$: la section efficace de diffusion
- χ_p : le spectre de neutrons prompts
- ν_p : le nombre moyen de neutrons prompts par fission
- $\Sigma_f(\vec{r},E,t)$: la section efficace de fission
- $S_d(\vec{r},E,t)$: la source de neutrons retardés
- $S(\vec{r},E,t) = \int_{4\pi} d^2\Omega q(\vec{r},E,\vec{\Omega},t)$: source de neutrons indépendante

Or, malgré toutes ces approximations, il n'en demeure pas moins que l'équation intégro-différentielle 1.2 demeure très compliquée. Il est donc une fois de plus nécessaire d'utiliser des méthodes numériques pour la résoudre. Par conséquent, comme

la plupart des méthodes numériques traitant les équations différentielles, il faut effectuer une discrétisation en espace et en énergie. Pour ce faire, il faut condenser le spectre continu de l'énergie en un certain nombre de sous-groupes. Habituellement, pour les réacteurs CANDU-6, une condensation à deux groupes d'énergie est suffisante (groupe thermique $\leq 4eV$ et groupe rapide $> 4eV$). Du côté spatial, une discrétisation suivant le domaine cellulaire déjà utilisé pour les calculs de cellule est effectuée. Finalement, les opérateurs différentiels étant à leur tour discrédités à l'aide d'une méthode aux différences finies, rendent l'équation de diffusion transposable dans un algorithme informatique.

1.6 La chaîne de calcul

Une fois les bases théoriques, les approximations et les discrétisations connues, il est maintenant propice de présenter une des chaînes de calcul qui implémente les algorithmes permettant la résolution des équations de neutronique. La prochaine section traitera donc de deux applications informatiques développées à l'Institut de génie nucléaire de l'École Polytechnique de Montréal : DRAGON et DONJON. Cette chaîne de calcul, au coeur des développements du projet D2G2, sera brièvement abordée afin d'avoir une vue d'ensemble du contexte logiciel pour lequel une optimisation des procédés désire être apportée. Il va de soi qu'une description complète de DRAGON et DONJON dépasse largement le cadre du présent ouvrage. Il est conseillé de se référer à leur référence respective pour une étude plus exhaustive.

1.6.1 DRAGON

Le logiciel DRAGON est utilisé pour la résolution numérique des calculs de cellule et de supercellule. Plus précisément, le code de cellule DRAGON permet d'évaluer en théorie de transport les sections efficaces macroscopiques des matériaux et des régions,

en fonction de l'énergie des neutrons, ainsi que les taux de réactions dans le combustible. Techniquement, DRAGON peut résoudre l'équation de transport (éq. 1.1) dans des cellules 1D, 2D et dans des supercellules 3D en utilisant diverses méthodes algorithmiques telles que les probabilités de collision, les courants d'interface ou bien $J\pm$.

En ce qui a trait à son architecture informatique, le code DRAGON est divisé en plusieurs modules de calculs reliés entre eux via le programme de contrôle généralisé du groupe d'analyse nucléaire (GANlib [9]). L'échange des données entre ces différents modules est assuré par des structures de données hiérarchisées dépendantes aussi des outils de la GANlib. Or, bien que ces structures de données ne découlent pas d'un standard reconnu, elles sont par contre très bien documentées. En fait, elles ont été développées spécifiquement pour répondre aux besoins des utilisateurs et des développeurs de DRAGON.

1.6.2 DONJON

Le logiciel DONJON est utilisé pour la résolution numérique des équations statiques et dynamiques de diffusion (éq. 1.2). Plus précisément, le code DONJON permet entre autres de déterminer le flux neutronique d'un réacteur nucléaire à partir des sections efficaces macroscopiques homogénéisées et condensées des volumes élémentaires formant sa géométrie. DONJON peut aussi traiter l'évolution du combustible en fonction de son irradiation ou bien de son mode de rechargement. Ce code de diffusion peut également simuler différents modes opérationnels d'un réacteur en paramétrant des mécanismes de contrôle tels que des barres de compensation et des contrôleurs liquides.

L'architecture informatique de DONJON est quant à elle identique à celle de DRAGON. C'est-à-dire, qu'elle est aussi sous forme modulaire, gouvernée par la GANlib.

De plus, tout comme DRAGON, DONJON procède à des échanges intermodulaires à l'aide de structures de données hiérarchisées spécialement développées pour ses propres besoins. La figure 1.4 expose justement cette structure modulaire où les rectangles correspondent à différents modules de DONJON et les cercles représentent les structures d'échange hiérarchisées.

Finalement, comme il est nécessaire pour simuler un réacteur d'avoir préalablement déterminé les sections efficaces macroscopiques, il est clair que DONJON dépend explicitement des calculs de cellule et de supercellule. L'utilisation de DRAGON ou d'un autre logiciel de transport est donc essentielle. À ce sujet, la figure 1.5 présente sommairement l'ordre dans lequel les différentes étapes de la chaîne de calcul doivent habituellement s'accomplir.

1.6.3 GANlib

Finalement tel que spécifié précédemment, l'entité permettant la cohésion entre les différents modules de DRAGON et ceux de DONJON est la GANlib. Une explication plus détaillée de son fonctionnement est donc primordiale étant donné que tout logiciel voulant contrôler et automatiser la chaîne de calcul DRAGON-DONJON doit impérativement se plier aux exigences de cette librairie. En effet, cette librairie permet de gérer librement l'exécution de tous les modules FORTRAN implémentés dans la chaîne de calcul neutronique de l'École Polytechnique. Plus spécifiquement, grâce à de cette librairie il est possible d'organiser à souhait la séquence d'appels des modules DRAGON-DONJON et tous leurs échanges de données sans aucune recompilation. Bref, en rédigeant un fichier d'entrées GANlib à l'aide des spécifications du langage CLE-2000 [10], l'utilisateur peut agencer dans l'ordre qu'il désire les modules de calcul et en manipuler ces entrées-sorties.

Par exemple, le fichier à l'annexe I.5 correspond à un fichier d'entrées pour lancer

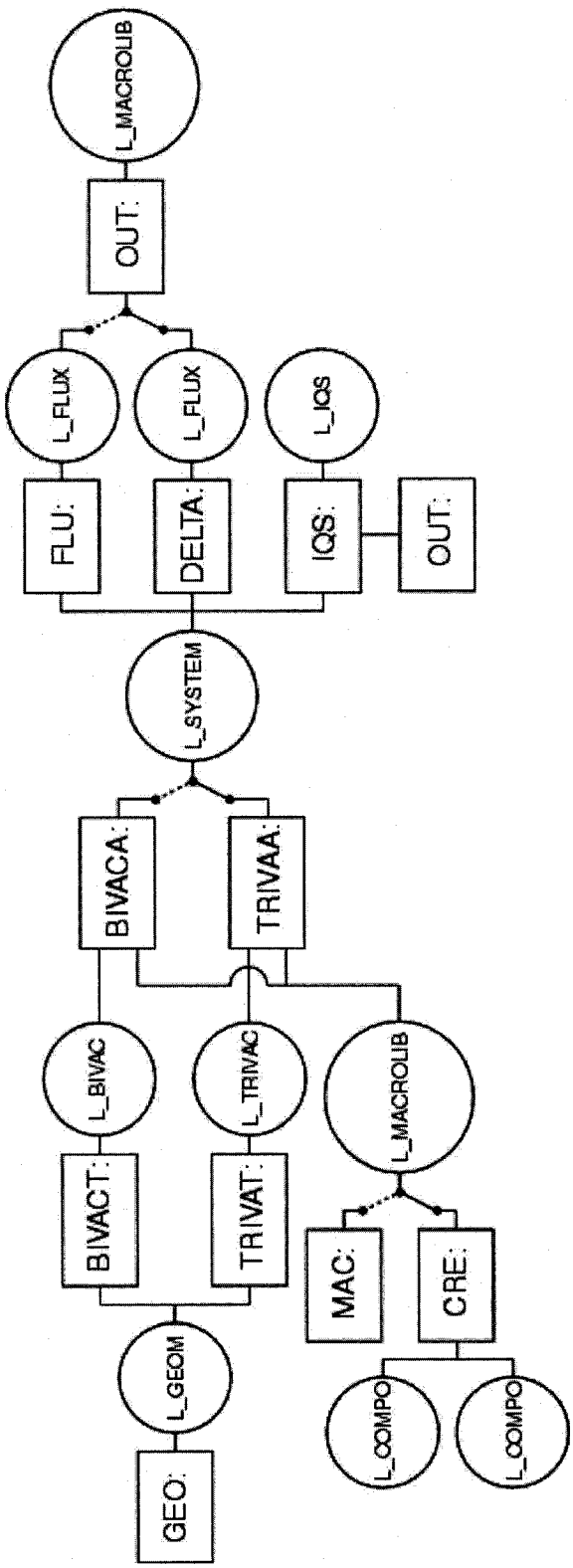

Figure 1.4 Approche modulaire de DONJON

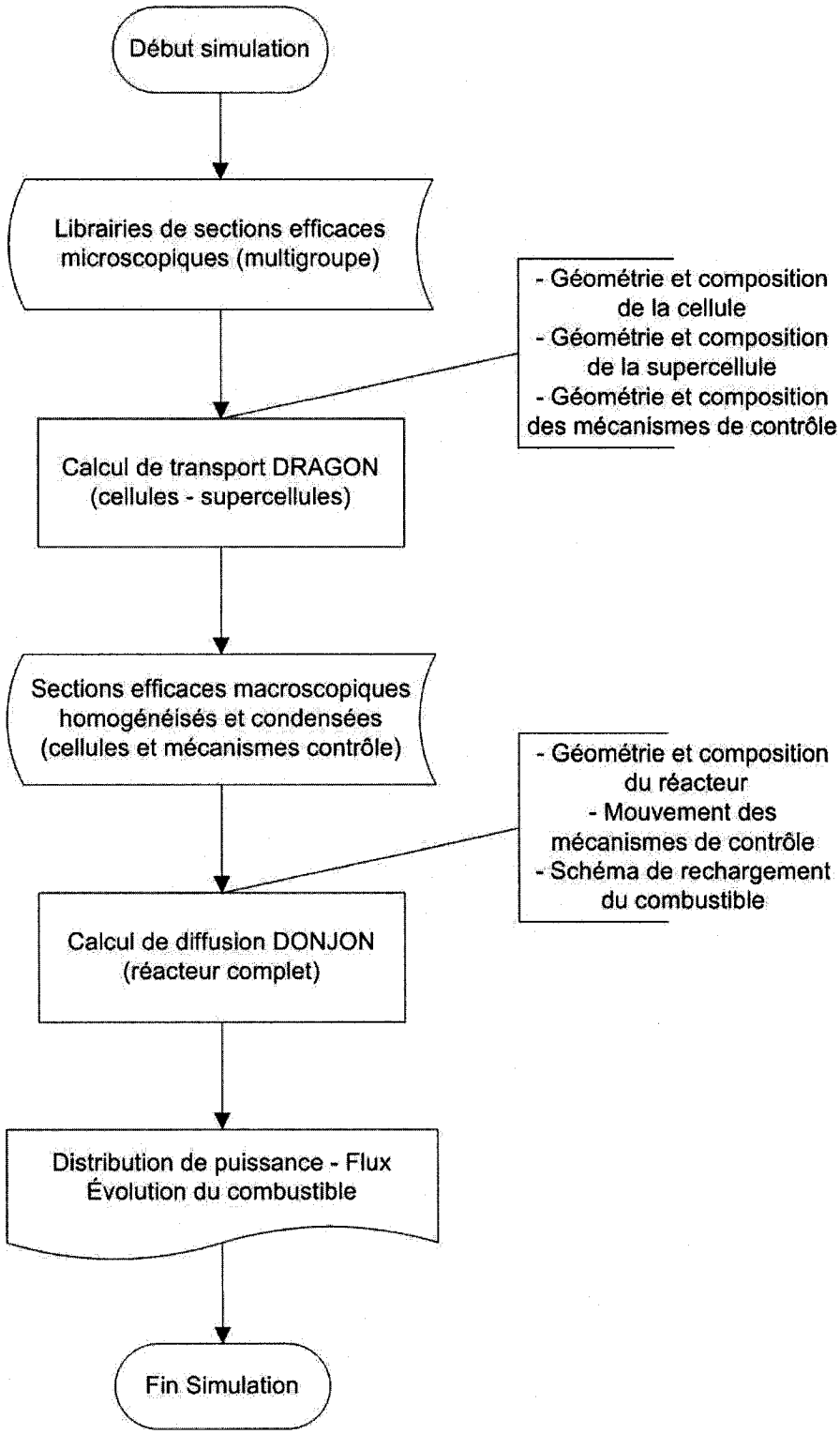

Figure 1.5 Enchaînement des calculs DRAGON-DONJON

une simulation d'un réacteur CANDU-6 avec DONJON. Or, celui-ci permet justement d'apprécier certaines possibilités de la GANlib : déclaration de procédures, déclaration de fichiers textes séquentiels d'entrées-sorties, déclaration de structures d'échanges hiérarchiques directement en mémoire (linked lists) ou sur disque (fichiers xsm), déclaration de types standards (chaînes de caractères, entiers, réels, booléens), opérations élémentaires (reverse polish notation), attribution simple entre un fichier et une structure hiérarchique, etc. Bref, plusieurs fonctionnalités très utiles permettant d'élaborer une multitude d'analyses neutroniques sans aucune recompilation de la moindre ligne de code de DRAGON ou DONJON ; une particularité des plus intéressantes lorsqu'il est question d'optimiser cette chaîne de calcul.

CHAPITRE 2

ASSISES DU PROJET D2G2

Depuis les années soixante-dix, l'industrie nucléaire canadienne a su développer et améliorer d'excellentes applications informatiques pour la physique des réacteurs. Cependant, un consensus semble dernièrement s'être établi au sein de la communauté en ce qui concerne les acquis et les besoins des différents codes de simulation actuellement utilisés. La naissance du projet D2G2 découle directement de l'apparition de ce consensus.

Il est donc pertinent dans ce chapitre d'effectuer un survol des acquis et des besoins de l'industrie afin de bien saisir l'enjeu et les différentes motivations ayant mené à la création de D2G2. De plus, il est essentiel de survoler le processus de développement logiciel utilisé dans le cadre de ce projet et de préciser son approche orientée prototype.

2.1 Acquis de l'industrie

Depuis sa création, l'industrie canadienne a su développer des codes de simulation performants permettant d'optimiser et de rendre sécuritaire l'exploitation de ses centrales nucléaires. En effet, l'industrie met à la disposition de la communauté nucléaire canadienne des logiciels de simulation neutronique de très hautes qualités. Cependant, pendant de nombreuses années, les équipes d'analyse des différentes centrales au Canada n'ont guère utilisé les mêmes applications. En effet, ce n'est que depuis quelques années que l'industrie s'est dotée de critères rigoureux pour standardiser ces

logiciels de simulation. Ainsi, suite à cette initiative, trois codes IST[1] de la physique des réacteurs ont été accrédités :

- WIMS-IST [11] : un code de transport 2-D multigroupe utilisé pour les calculs de cellule CANDU. Découlant d'une version développée par Énergie Atomique du Canada Limitée (WIMS-EACL), ce code permet de déterminer les sections efficaces homogénéisées et condensées d'une multitude de configurations de la cellule et de son combustible.
- DRAGON-IST [12] : un code de transport 3-D multigroupe servant à la détermination des sections efficaces incrémentales pour les différents mécanismes de réactivité du réacteur CANDU. Développé à l'Institut de génie nucléaire de l'École Polytechnique, ce code est en fait le seul à pouvoir effectuer des calculs de supercellule.
- RFSP-IST [13] : un code de diffusion 3-D pour les réacteurs CANDU. Développé par Énergie Atomique du Canada Limitée, ce code calcule le flux neutronique et la distribution de puissance dans tout le coeur. Ces analyses statiques et dynamiques à deux groupes d'énergie lui permettent d'effectuer la gestion du combustible, le suivi opérationnel et les analyses du système de régulation et d'arrêt d'urgence.

Fait à noter, ces codes sont aujourd'hui considérés comme les outils à privilégier pour effectuer les analyses de sûreté. Même que dernièrement, des démarches ont été entreprises pour introduire ces applications au sein des équipes de gestion du combustible pour accomplir les analyses quoditiennes en centrale. Cependant, bien qu'une standardisation des outils soit valable pour la cohésion entre les partenaires de l'industrie, la pérennité de ces codes est loin d'être assurée. En effet, même s'ils

[1]IST est l'acronyme anglais pour Industry Standard Toolset

ont prouvé leur qualité et ont été grandement qualifiés depuis plusieurs années, les codes IST possèdent quand même certaines lacunes.

2.2 Besoins de l'industrie

La standardisation de ses outils de simulation fut un point tournant pour l'industrie nucléaire canadienne. Cependant, malgré cette percée, il semble que l'industrie soit à l'aube d'une seconde révolution, car la plupart de ces applications informatiques ont un urgent besoin de modernisation au niveau de leur architecture. En effet, étant donné que leur conception est basée sur des langages de programmation datant de la fin des années 70, ces applications sont incompatibles avec les méthodes modernes d'implémentation. L'industrie projette donc d'améliorer ces codes afin qu'ils puissent être compatibles avec les ressources informatiques actuelles [3].

Par contre, tenter une réingénierie complète de plusieurs centaines de milliers de lignes de code FORTAN-77 demande des efforts colossaux. C'est pourquoi l'industrie envisage plutôt de migrer ces applications vers FORTRAN-95, afin de pouvoir profiter d'une approche orientée-objet, tout en préservant autant se faire que peut l'intégralité de leurs codes. En effet, ayant été révisés par de nombreux et très stricts contrôles de qualité, il est préférable de ne pas envisager une refonte majeure des codes IST. D'autant plus que certaines parties du code de ces applications peuvent être considérées comme patrimoniales[2], rendant ainsi leur modernisation très ardue. Cependant, il est clair que malgré ces contraintes, si importantes soient-elles, il est intéressant d'envisager le développement d'applications modernes contrôlant ces codes hautement standardisés. L'industrie nucléaire a effectivement un urgent besoin de faire évoluer son paradigme de simulation [15]. Le projet D2G2 est justement un exemple concret

[2]Traduction directe du terme anglais « legacy ». Pour une étude avancée des enjeux liés à la modernisation des applications patrimoniales, voir l'ouvrage de Robert C. Seacord [14].

démontrant la possibilité de manipuler une chaîne de calcul neutronique.

2.3 Spécification des besoins

Le projet D2G2 se veut avant tout un prototype démontrant la possibilité de moderniser la chaîne de calcul DRAGON-DONJON présentée au chapitre précédent. En fait, ce projet vise à démontrer que l'intégration de techniques informatiques modernes est réalisable et qu'elle permet de corriger une des lacunes majeures des chaînes de calcul patrimoniales, soit l'interopérabilité entre leurs constituants. Plus spécifiquement, les objectifs du projet D2G2 se résument ainsi : l'utilisation de l'approche orientée-objet pour moderniser l'architecture globale de la chaîne de calcul, le développement d'un pilote de contrôle pour automatiser les codes de simulation DRAGON et DONJON, la gestion du stockage des entrées et des sorties, la standardisation des échanges de données et finalement l'ajout d'une interface graphique.

2.3.1 Modernisation de l'approche informatique

La modernisation de l'architecture globale de la chaîne de calcul DRAGON-DONJON doit découler directement de l'utilisation d'une approche orientée-objet. En effet, les méthodes orientées-objet (O.O.) sont de nos jours reconnues pour leur réutilisabilité, leur modularité, leur flexibilité et leur maintenabilité. Il est donc évident que l'utilisation de techniques modernes de génie logiciel est justifiée et requise dans le cadre du projet D2G2. Ces méthodes informatiques permettent effectivement l'élaboration d'applications de très hautes qualités en optimisant et en améliorant toutes les phases du développement. Les codes de simulation neutronique actuels sont quant à eux construits à l'aide de méthodes procédurales. Leur développement et leur maintenabilité sont souvent difficiles, l'ajout de nouvelles fonctionnalités est très fastidieux et plus souvent qu'autrement une compréhension intégrale du système est presque

impossible. Or, bien que les avantages des méthodes O.O soient indéniables, il est parfois plus clair d'énumérer certaines implications pragmatiques de leur utilisation pour en apprécier davantage les biens faits. Par exemple :

- Elles permettent, grâce au découpage par objet, de regrouper les différents aspects d'une même abstraction ; ce qui procure une meilleure localisation des fonctionnalités du système. Ainsi, grâce à cette encapsulation, il est facile de cibler les endroits où il est nécessaire d'intervenir en cas d'évolution ou de maintenance [16].

- Elles permettent l'utilisation de patrons de conception (design patterns) ; ce qui simplifie la réutilisation de solutions éprouvées et performantes pour résoudre certaines situations à problématiques ou contextes similaires [17].

- Elles permettent l'utilisation de cadre de travail spécialisé (frameworks) ; ce qui permet une élaboration rapide et une grande maintenabilité dans un domaine particulier (architecture client - serveur, interface graphique, etc.) grâce à cette charpente logicielle conçue par les experts de la discipline [18].

- Elles permettent aussi une documentation compréhensible et complète ; ce qui rend plus facile l'obtention des informations relatives aux décisions de modélisation et de conception, contrairement aux langages procéduraux qui ne sont guères conçus pour mettre l'accent sur cet aspect.

De plus, bien qu'il soit possible d'utiliser un langage procédural tel que FORTRAN-77 pour mettre en oeuvre les prémisses de la programmation orientée-objet (abstraction de données, héritage, typage dynamique, polymorphisme) [19], l'utilisation d'un langage implémentant plus naturellement tous ces concepts O.O. est à privilégier. Il n'est donc pas surprenant que le projet D2G2 doive opter pour un langage tel que C++ ;

bien que la grande majorité des logiciels en relation avec les simulations des réacteurs CANDU soit actuellement écrite en FORTRAN. De toute façon, une évaluation préliminaire a permis d'établir que les bénéfices de l'utilisation du C++ seront beaucoup plus importants que les inconvénients ; d'autant plus qu'éventuellement toutes les applications de neutronique seront forcées de migrer leur code vers l'O.O.

2.3.2 Pilote de contrôle

Le pilote de contrôle doit être la composante la plus importante de projet D2G2. En effet, ce pilote, au coeur du développement de tout le système, assumera le rôle de gestionnaire de l'automatisation du lancement des simulations. Il doit en fait être responsable de la communication entre les utilisateurs et les applications DRAGON et DONJON. Ce qui implique la prise en charge de la gestion des requêtes provenant des utilisateurs, de la création de fichiers d'entrées et de l'administration des processus DRAGON et DONJON.

2.3.2.1 La gestion des requêtes

Le système de gestion des requêtes doit être le lien direct entre les fonctionnalités du pilote de contrôle et les demandes provenant des utilisateurs. Quel que soit le type de requête, le premier contact entre l'utilisateur et le pilote doit s'effectuer via ce système. Il est donc mandaté pour orienter le pilote vers l'exécution d'une simulation, l'obtention de résultats ou bien tout simplement vers la consultation du statut du serveur ou vers la modification de l'état de celui-ci.

2.3.2.2 Les fichiers d'entrées

Un autre tâche du pilote de contrôle doit être la création des fichiers d'entrées. C'est-à-dire, l'assemblage des fichiers nécessaires à l'exécution d'une simulation à partir des paramètres fournis par l'utilisateur lors d'une requête. Bien sûr, la syntaxe exigée

par la GANlib pour la création de ces entrées doit être suivie à la lettre, car ces fichiers constituent l'unique lien entre le pilote et les codes DRAGON et DONJON. Cette étape, aussi banale qu'elle peut paraître, est d'une grande utilité. En effet, les fichiers d'entrées sont encore aujourd'hui élaborés à la main, ce qui augmente considérablement le risque d'erreurs typographiques à l'origine de plusieurs simulations erronées.

2.3.2.3 Les processus DRAGON et DONJON

Le pilote de contrôle doit aussi être délégué à l'administration des processus DRAGON et DONJON. C'est-à-dire, à la préparation et au lancement des processus avec les fichiers d'entrées GANlib, à la répartition des processus sur différents ordinateurs, au suivi de l'évolution des processus jusqu'à la fin de leur exécution et finalement à la récolte des structures hiérarchisées (fichiers XSM) nécessaire à la cohésion entre DRAGON et DONJON.

En terminant, il est primordial de rappeler l'importance et la raison d'être du pilote de contrôle. En effet, celui-ci doit permettre au projet D2G2 de créer un cadre de gestion *indépendant* des applications DRAGON et DONJON, ce qui implique qu'aucune modification ne devra être apportée au niveau de ces codes neutroniques. Ainsi, toutes évolutions, autres que celles apportées au format des entrées et sorties, de DRAGON et DONJON ne devront en aucun temps influencer le pilote de contrôle. De plus, grâce à cette structure autonome, la migration du pilote vers d'autres chaînes de calcul devra être possible.

2.3.3 Stockage des entrées et sorties

Le stockage des entrées et sorties consiste principalement à la sauvegarde des paramètres d'entrées fournis par les utilisateurs et des fichiers de données issues des simulations. Cette sauvegarde doit bien sûr être répertoriée et des recherches doivent

pouvoir y être effectuées. À première vue, il est vrai que ce système de stockage s'avère une simple formalité. Cependant, il est essentiel que cette fonctionnalité soit implémentée par le projet D2G2. En effet, contrairement aux techniques qui perdurent encore aujourd'hui, avec les outils D2G2 il devra être possible de consulter la liste des simulations complétées, pour se référer aux paramètres d'entrées d'une ancienne simulation, pour en élaborer une nouvelle ou bien tout simplement pour récupérer des résultats. En fait, la gestion des entrées et sorties devra permettre la création d'un répertoire de données optimisant les temps de calculs et éliminant les simulations redondantes. Par exemple, les sections efficaces des mécanismes de contrôle sont des paramètres très longs à calculer et la plupart des utilisateurs en ont besoin. De plus, les entrées GANlib de la plupart des utilisateurs sont plus souvent qu'autrement les mêmes. Donc, grâce à ce répertoire, une fois qu'un utilisateur aura complété le calcul des mécanismes de contrôle pour la première fois, ces résultats pourront être efficacement réutilisés par le reste de la communauté.

2.3.4 Standardisation des échanges de données

Un autre objectif primordial du projet D2G2 est le développement d'une structure permettant la standardisation des échanges de données. En effet, une déficience majeure des chaînes de calcul se situe au niveau de l'interopérabilité entre leurs constituants. Par exemple, les données échangées entre les codes IST (présentés à la section 2.1), sont transformées et manipulées à l'aide de scripts Perl[3] afin d'être portables d'un module à l'autre. Il est donc difficile de suivre et de valider l'intégralité des résultats lorsque ceux-ci se métamorphosent constamment à l'intérieur de la boucle de calcul transport-diffusion-thermohydraulique. Donc, dans le cadre du projet D2G2, il est pertinent de définir un format de données commun à tous les modules de la chaîne de calcul DRAGON-DONJON. Qui plus est, une vision utopique comporterait un

[3]http ://www.perl.org

standard international permettant d'assurer la cohésion entre toutes les chaînes de calcul neutronique. Il serait alors possible d'échanger facilement des résultats entre des applications aussi hétérogènes que MCNP, WIMS, DRAGON, etc.

2.3.5 Interface graphique et visualisation

Tel qu'explicité dans l'introduction, toute tentative de modernisation de la chaîne de calcul DRAGON-DONJON doit absolument intégrer des interfaces graphiques et des techniques modernes de visualisation. En effet, il est de nos jours inacceptable que les applications de neutronique ne profitent guère des puissants outils graphiques qui sont amplement répandus. C'est pourquoi le projet D2G2, étant instigateur de ces nouveaux développements, propose le développement d'une interface utilisateur graphique. L'objectif étant bien sûr d'obtenir éventuellement une application permettant l'affichage tridimensionnel des flux et des différentes géométries des calculs de cellule et de réacteur.

2.3.6 Portabilité

Une autre préoccupation au coeur des objectifs du projet se situe au niveau de la portabilité du code de ces applications. En effet, comme la majorité des développements logiciels modernes, les outils D2G2 doivent être portables sur la plupart des systèmes d'exploitation. Plus précisément, ce niveau de portabilité doit impliquer que peu importe la plateforme sur laquelle l'utilisateur travaille, celui-ci doit être en mesure de compiler et d'exécuter les logiciels D2G2. Ainsi, une attention particulière doit être accordée au choix des librairies et des cadres de travail. C'est-à-dire que le processus de décision menant à l'utilisation ou non d'une librairie doit impérativement inclure une phase d'étude en étroite relation avec la portabilité du code de l'outil considéré. Par exemple, en ce qui a trait à l'interface graphique, il est évident que l'affichage d'une fenêtre amène une dépendance importante aux appels systèmes.

Par conséquent, le choix d'une librairie graphique implémentant une interface commune encapsulant les différences entre les systèmes d'exploitation est primordial. Ce raisonnement s'applique également dans le contexte du développement réseau où les routines d'accès aux canaux de communication sont plus souvent qu'autrement différentes d'une plateforme à l'autre. Bref, la portabilité du code des applications D2G2 est directement proportionnelle à celle des cadres de travail et des librairies à la base de son développement.

2.4 Processus de développement

L'approche orientée-objet conduit à une réorganisation complète des méthodes de développement logiciel. En effet, son utilisation a une profonde influence sur toutes les activités nécessaires à l'obtention d'un système logiciel ou à l'évolution d'un système existant [20]. Il est donc clair que son usage occasionne une refonte majeure du cycle de vie logiciel. Or, il existe plusieurs variétés de cycle de vie s'articulant toujours autour des mêmes phases de développement suivantes : l'analyse (modélisation), la conception, l'implémentation, la validation, l'exploitation et la maintenance. Cependant, même s'ils partagent les mêmes étapes du processus de développement, il n'en demeure pas moins que le niveau d'utilisation de celles-ci peut varier considérablement d'un cycle à l'autre. En effet, les cycles de vie diffèrent fréquemment en ce qui a trait à la durée consacrée à chaque phase. Par exemple, certains processus peuvent réduire considérablement l'analyse et se concentrer sur l'implémentation, alors que d'autres privilégient l'analyse et considèrent l'implémentation comme une phase secondaire. Ou bien, certains développements sont linéaires alors que d'autres optent pour un développement itératif.

Par contre, depuis quelques années, malgré l'apparition d'une grande diversité de cycles, un consensus général semble émerger au sein de la communauté du génie lo-

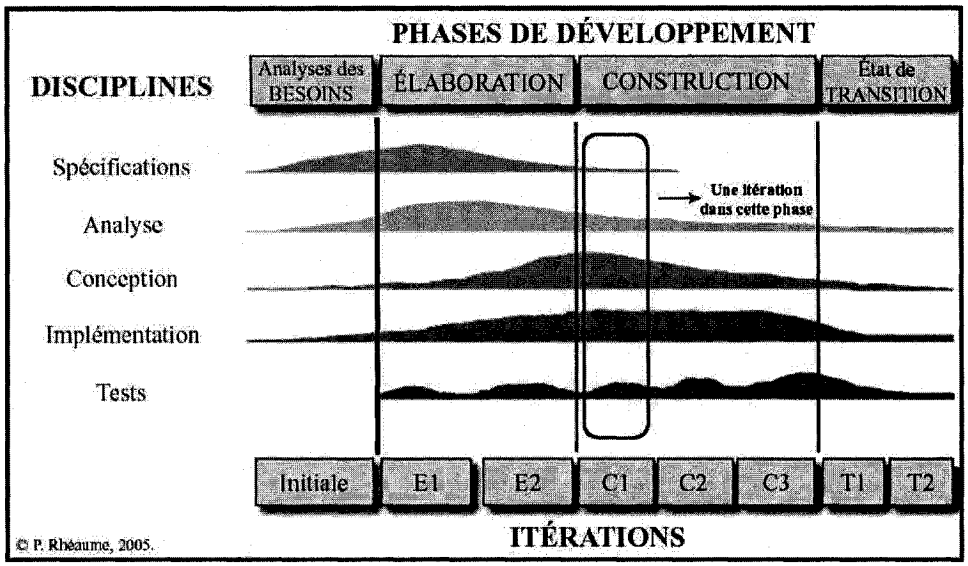

Figure 2.1 Développement itératif à travers le cycle de vie logiciel

giciel. En effet, les meilleures pratiques semblent converger vers un seul et unique processus de développement connu sous l'appellation très originale de processus unifié. Celui-ci préconise l'utilisation d'une démarche guidée par les cas d'utilisation (spécification/analyse des besoins), itérative et centrée sur l'architecture [21].

2.4.1 Besoins des utilisateurs

Des fonctionnalités logicielles précisées par les besoins des utilisateurs semblent aujourd'hui plus qu'évidentes. Cependant, pendant de nombreuses années, les concepteurs ont fréquemment négligé cet aspect du développement. En effet, bon nombre d'entre eux ont plus souvent qu'autrement opté pour leur propre vision, au détriment de la consistance et de l'interopérabilité entre les applications. Depuis, la spécification des besoins est devenue une phase à part entière du processus de développement. De plus, celle-ci doit sans cesse se peaufiner au cours des itérations de la phase d'élaboration afin préciser les requis permettant d'améliorer les fonctionnalités.

C'est pourquoi les besoins présentés dans ce chapitre, bien qu'ils découlent princi-

palement de l'étude de la chaîne de calcul DRAGON-DONJON et des objectifs de l'industrie nucléaire, ne constituent guère des fondements immuables au projet. En fait, depuis le début des présents travaux, ceux-ci n'ont jamais cessé d'évoluer. Par exemple, suite à la présentation d'un des premiers prototypes [22], des discussions avec des utilisateurs potentiels ont permis de raffiner et même d'ajouter certains besoins.

2.4.2 Itératif et incrémental

Opter pour une démarche itérative et incrémentale dans le cadre de modélisation d'un système revient à affiner l'analyse étape par étape où chaque itération porte sur des degrés d'abstraction de plus en plus précis. Or, cette démarche s'applique aussi à la conception en effectuant des cycles successifs où des solutions partielles (fonctionnellement incomplètes) apparaissent et où d'autres solutions se finalisent. Le but étant de réussir à maîtriser les inconnues et les incertitudes découlant de la complexité du système afin de réduire le risque d'erreurs majeures.

2.4.3 Centré sur l'architecture

Une démarche centrée sur l'architecture est aussi une caractéristique importante des procédés de développement modernes. Mettre l'accent sur l'architecture du logiciel permet effectivement d'améliorer grandement son adaptabilité, ces performances et sa fiabilité. Avoir une vue globale des fondements du système favorise aussi la cohésion entre les divers constituants, étant donnée l'importance d'une vision claire et précise dans l'élaboration des interactions entre les différents modules de l'application.

2.4.4 Développement XP

En pratique, bien que le développement du projet D2G2 soit guidé par les besoins des utilisateurs, itératif et centré sur l'architecture, celui est en fait plutôt basé sur

un cycle XP (extreme programming). Or, la programmation extrême ne constitue pas un modèle de processus à proprement parler, mais plutôt une méthodologie. Cette méthodologie a pour but de soutenir le développement du logiciel d'une manière moins rigide que par l'utilisation de processus formels. Le manifeste du développement agile mentionne justement les valeurs véhiculées par cette philosophie [23] :

- L'importance des individus et des interactions est accrue relativement aux processus et aux outils.
- Accentuation de l'importance d'obtenir un logiciel fonctionnel relativement à l'exhaustivité de la documentation.
- Priorité de collaboration client-fournisseur sur la négociation contractuelle.
- Priorité de la réponse au changement sur le suivi de plans.

Donc, dans l'optique de la programmation extrême, le développement du projet D2G2 se résume à l'application d'un cycle incrémental suivant des itérations extrêmement courtes, avec à chaque itération l'obtention d'une application sensiblement fonctionnelle constamment en pleine évolution.

2.4.5 Prototype

Finalement, il faut mentionner qu'étant donné que le projet D2G2 se veut une évolution générale de l'environnement de simulation neutronique DRAGON-DONJON, il est clair que cette première approche doit être envisagée sous la tutelle du prototypage. En effet, les développements actuels doivent être considérés comme une initiative démontrant la faisabilité de la modernisation de la chaîne de calcul. Donc, dans l'optique du développement itératif abordé précédemment, certaines fonctionnalités du système seront retenues et implantées pratiquement sous leur forme finale

lors de courtes itérations. Par contre, d'autres aspects seront peu abordés et légués à ceux ou celles voulant les achever ou s'en inspirer. Bref, les applications résultantes de cette phase de prototypage du projet D2G2 ne seront guère exhaustives, mais constitueront un squelette significatif pour de futurs développements.

CHAPITRE 3

MODÉLISATION OBJET DE D2G2

De nos jours, les méthodes orientées-objet (O.O.) sont reconnues pour leur réutilisabilité, leur modularité, leur approche intuitive, leur flexibilité, leur robustesse et leur maintenabilité. Ces méthodes informatiques modernes permettent en effet l'élaboration d'applications de très hautes qualités. Or, la plupart des codes de simulation neutronique actuels sont quant à eux construits à l'aide de langages procéduraux tels que FORTRAN-77. Leur développement et leur maintenance sont difficiles étant donné que l'ajout de nouvelles fonctionnalités est très souvent fastidieux. De plus, contrairement aux méthodes orientées-objet, les langages procéduraux ne sont guère conçus pour mettre l'accent sur la documentation et sur l'explication de leur architecture. Il est donc évident que l'utilisation de méthodes orientées-objet optimise et améliore le développement logiciel du projet D2G2. C'est pourquoi son analyse est présentée.

Plus spécifiquement, ce chapitre est divisé en trois sections. La première aborde le langage UML, un langage standardisé permettant entre autres de représenter les modèles d'analyse. La seconde expose le modèle statique définissant les cas d'utilisation et l'agencement des classes, alors que la dernière décrit la modélisation dynamique résumant certaines interactions entre divers objets.

3.1 UML

La modélisation (analyse) orientée objet permet de définir de façon graphique et textuelle les diverses fonctionnalités du système à partir de la définition des besoins. Le modèle qui en résulte constitue une abstraction détaillée, précise et compréhensible du problème servant de cadre à la conception et à l'implémentation [24]. C'est lors de cette étape du développement que la clarification des besoins est réalisée, que l'organisation du domaine en décomposition par objets est effectuée et la description des interactions entre les constituants du système est définie. Or, depuis les 30 dernières années, certains consensus se sont formés au sein de la communauté de génie logiciel. De ceux-ci sont ressorties certaines méthodes standardisées mettant en pratique les différents concepts issus de cette philosophie orientée-objet. Parmi ceux-ci, l'une qui semble avoir uni toutes les méthodes est UML.

UML est la forme contractée de « Unified Modeling Language ». UML est un langage de modélisation objet. Il fournit les fondements pour spécifier, construire, visualiser et décrire les artefacts[1] d'un système logiciel [16]. Il permet de modéliser de manière claire et précise la structure et le comportement d'un système, indépendamment du langage de programmation. UML offre une voie d'échange de visions systématiques et précises. UML n'est pas un langage propriétaire, il est par contre homologué par l'OMG[2] et constitue la notation internationale pour la modélisation des applications à base d'objets. En résumé, UML est un formalisme graphique standardisé permettant une représentation de la sémantique orientée-objet. C'est une représentation graphique qui s'intéresse à une perspective précise du modèle et ne constitue pas un modèle en soi.

[1] Une information qui est utilisée ou produite par une démarche de développement de logiciel.
[2] Object Management Group (http ://www.omg.org)

Techniquement, UML définit neuf types de diagrammes [25] divisés en deux grandes catégories :

- **Modèle Statique** : diagrammes de cas d'utilisation, diagrammes de composants, diagrammes de déploiement, diagrammes de classes, diagrammes objets
- **Modèle Dynamique** : diagrammes de séquence, diagrammes d'états-transitions, diagrammes de collaboration, diagrammes d'activités

3.1.1 Le processus unifié

La méthodogie UML s'associe aussi très bien avec les méthodes modernes de développement logiciel. Avec UML, ce sont les utilisateurs par le biais des diagrammes des cas d'utilisation qui guident la définition des modèles. Ils servent en effet à délimiter le périmètre du système à modéliser. Ils définissent ce que doit être le système. De plus, à chaque itération de la phase d'analyse, il y a clarification et validation des besoins des utilisateurs. Il en résulte qu'à chacune des itérations, il y a aussi clarification et validation de tous les diagrammes, puisqu'ils dépendent directement des besoins des utilisateurs. Finalement, lorsque les besoins sont clairs et précis, l'architecture du système l'est d'autant plus. Il est alors plus facile de modéliser les composantes logicielles et d'élaborer leurs interactions. Bref, les trois critères au coeur de la description du processus unifié sont bel et bien respectés par la méthode UML.

3.1.2 La Notation

La notation UML n'est pas décrite dans le cadre de cet ouvrage. Bon nombre de références se charge d'en expliciter convenablement et exhaustivement toutes les technicalités. Il est donc conseillé, pour en savoir davantage, de parcourir le livre de Booch [25] : LA référence en la matière .

3.2 Modèle Statique

3.2.1 Diagrammes de cas d'utilisation

La première étape de la modélisation statique est la détermination (capture) des requis. Cette phase est très importante, car de celle-ci découle l'ensemble des fonctionnalités du système. Il faut donc définir et préciser adéquatement tous les besoins des utilisateurs afin d'être en mesure de schématiser les différentes tâches que l'application devra pouvoir exécuter.

Pour cette schématisation, la méthode UML propose le cas d'utilisation. Celui-ci permet de décrire les interactions découlant des besoins. Il s'élabore à l'aide de la notation UML sous la forme de diagrammes de cas d'utilisation (use case diagrams). Les composantes principales de tels diagrammes sont les acteurs et les cas d'utilisation. Les acteurs sont habituellement les entités (personne physique, système distant) qui sont susceptibles d'interagir avec le système. Bref, un cas d'utilisation décrit une action spécifique que l'acteur peut effectuer par l'entremise du système.

Les figures 3.1 à 3.5 présentent les diagrammes de cas d'utilisation décrivant les besoins en relations avec le lancement de simulations DRAGON et DONJON. Le premier diagramme, à la figure 3.1, affiche les fonctionnalités permettant à l'utilisateur de suivre et influencer l'évolution de ses simulations. Tandis que les diagrammes des figures 3.2, 3.3 et 3.4 correspondent respectivement aux actions que l'utilisateur peut effectuer pour élaborer une requête de simulation de cellule, de supercellule et de réacteur. Finalement, la figure 3.5 explicite les interactions de l'administrateur avec le système.

Plus spécifiquement, l'acteur utilisateur est un simple usager pouvant seulement gérer ses propres simulations. C'est-à-dire, qu'il peut créer ses requêtes de simulation, les

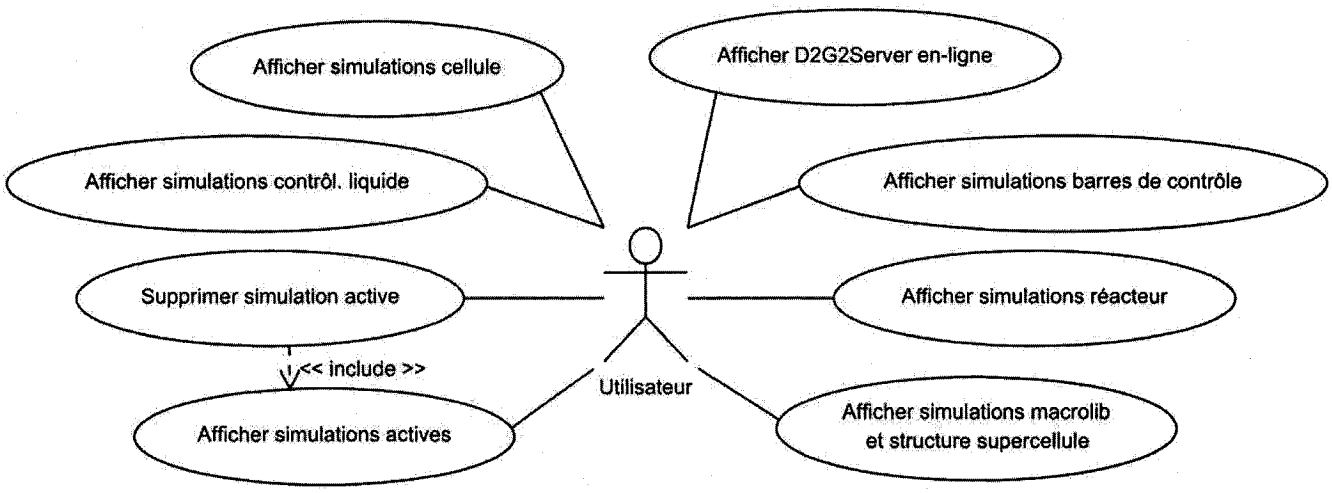

Figure 3.1 Diagramme des cas d'utilisation du suivi des simulations

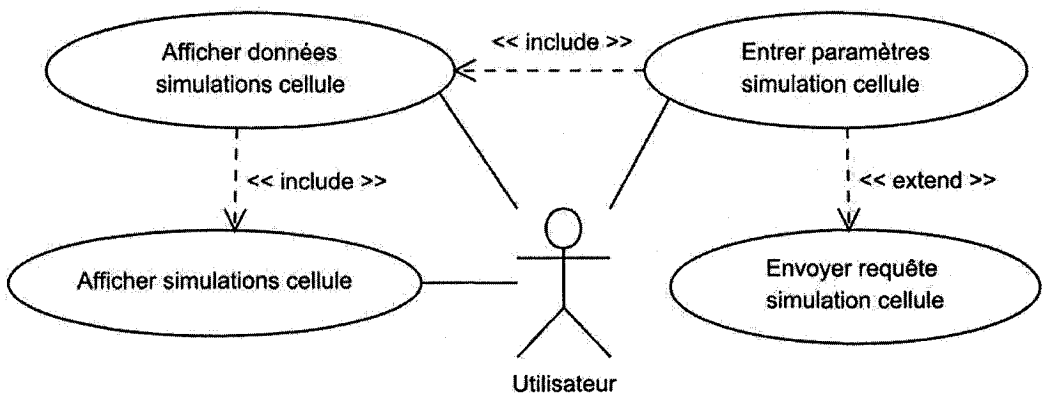

Figure 3.2 Diagramme des cas d'utilisation d'une simulation de cellule

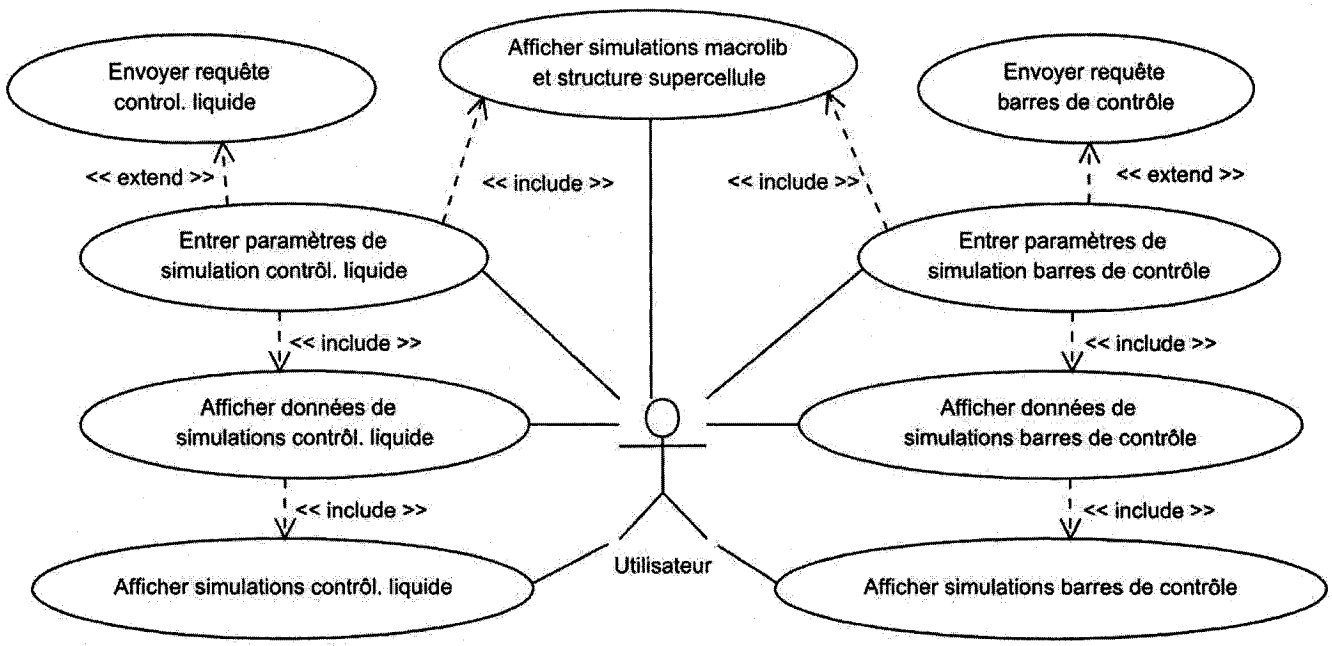

Figure 3.3 Diagramme des cas d'utilisation d'une simulation de supercellule

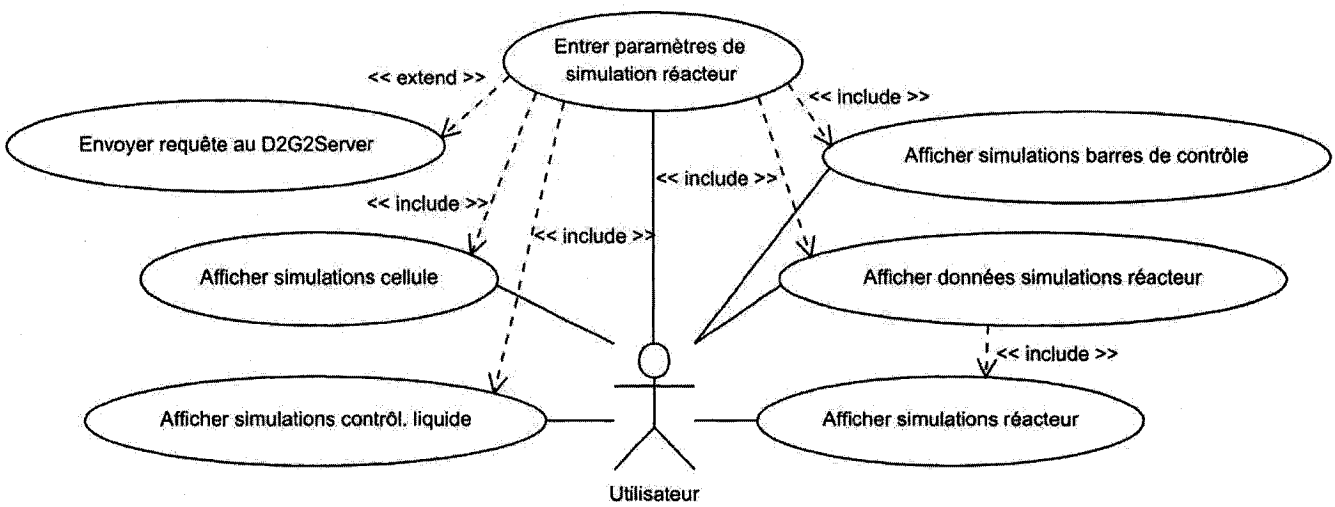

Figure 3.4 Diagramme des cas d'utilisation d'une simulation de réacteur

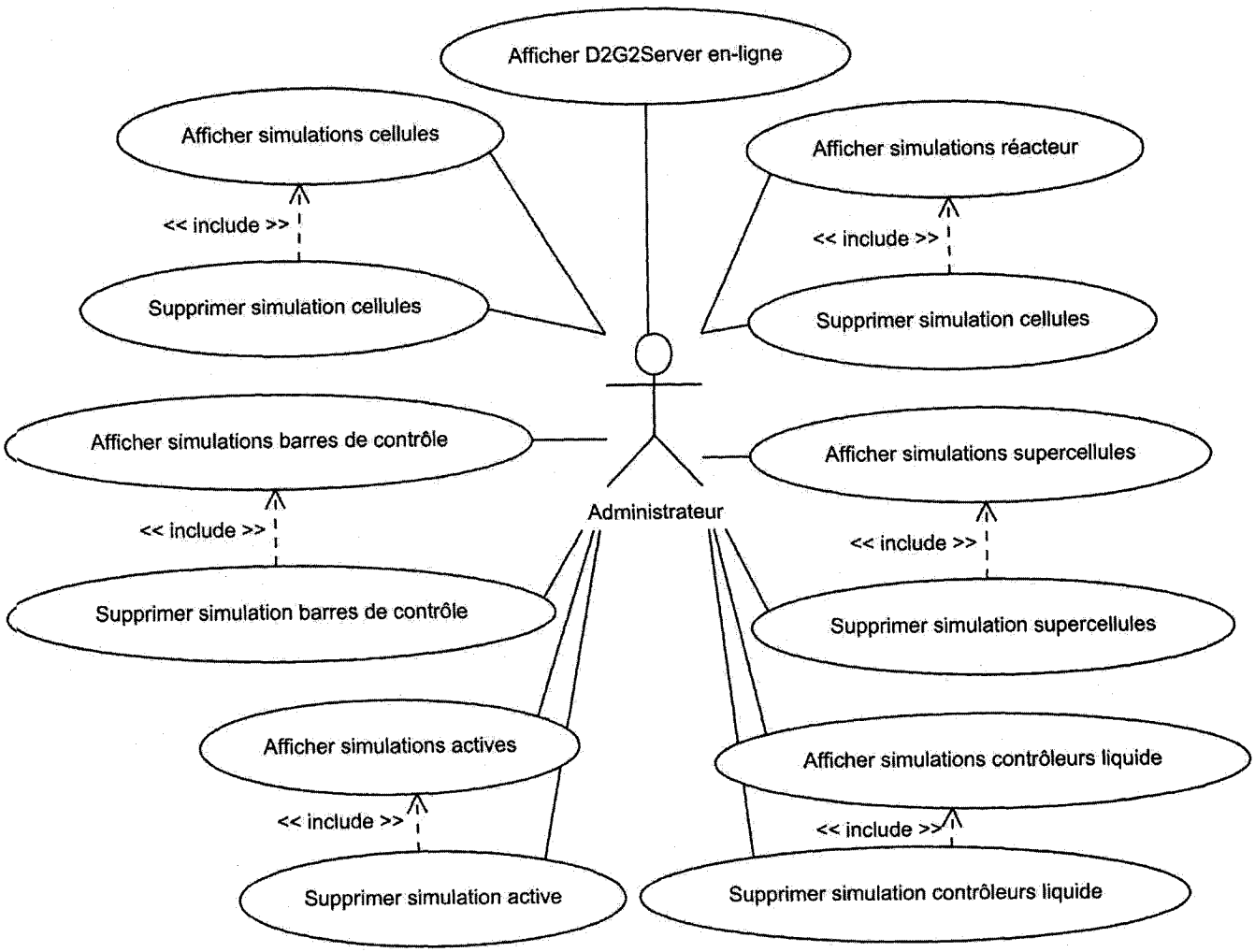

Figure 3.5 Diagramme des cas d'utilisation de l'administrateur

envoyer au serveur et les annuler lorsque nécessaire. Par contre, il lui est impossible de modifier l'état des simulations des autres utilisateurs. Ce qui est tout le contraire de l'administrateur. En effet, celui-ci correspond à un utilisateur avec pouvoir s'occupant de la gestion de toutes les simulations et de toutes les données. L'administrateur peut ainsi supprimer n'importe quelle simulation et toutes les données s'y rattachant, qu'elle soit active ou complétée.

3.2.2 Diagrammes de composants

Les diagrammes de composants décrivent les composants et leurs dépendances dans l'environnement de réalisation. Les diagrammes de composants sont des vues statiques de l'implémentation des systèmes. Ils sont par contre seulement utilisés lorsque les applications sont très complexes et qu'il est difficile d'apprécier une vue d'ensemble du domaine d'application du logiciel. C'est pourquoi un diagramme de déploiement est habituellement suffisant, car celui-ci peut aisément jouer un double rôle et représenter les deux à la fois.

3.2.3 Diagrammes de déploiement

Les diagrammes de déploiement montrent la disposition physique des différents matériels (ordinateurs, système de sauvegarde, alimentation) qui entrent dans la composition du système. Ils peuvent aussi décrire la répartition des instances de composants, de processus, de librairies. Le modèle de déploiement permet également de définir l'architecture physique : ces noeuds étant des unités matérielles sur lesquelles s'exécutent les composants logiciels et les relations entre ces noeuds représentant les moyens de communications les reliant (Internet). Finalement, chaque unité matérielle contient aussi les interactions entre les outils qu'il déploie. La figure 3.6 correspond au diagramme de déploiement des outils D2G2.

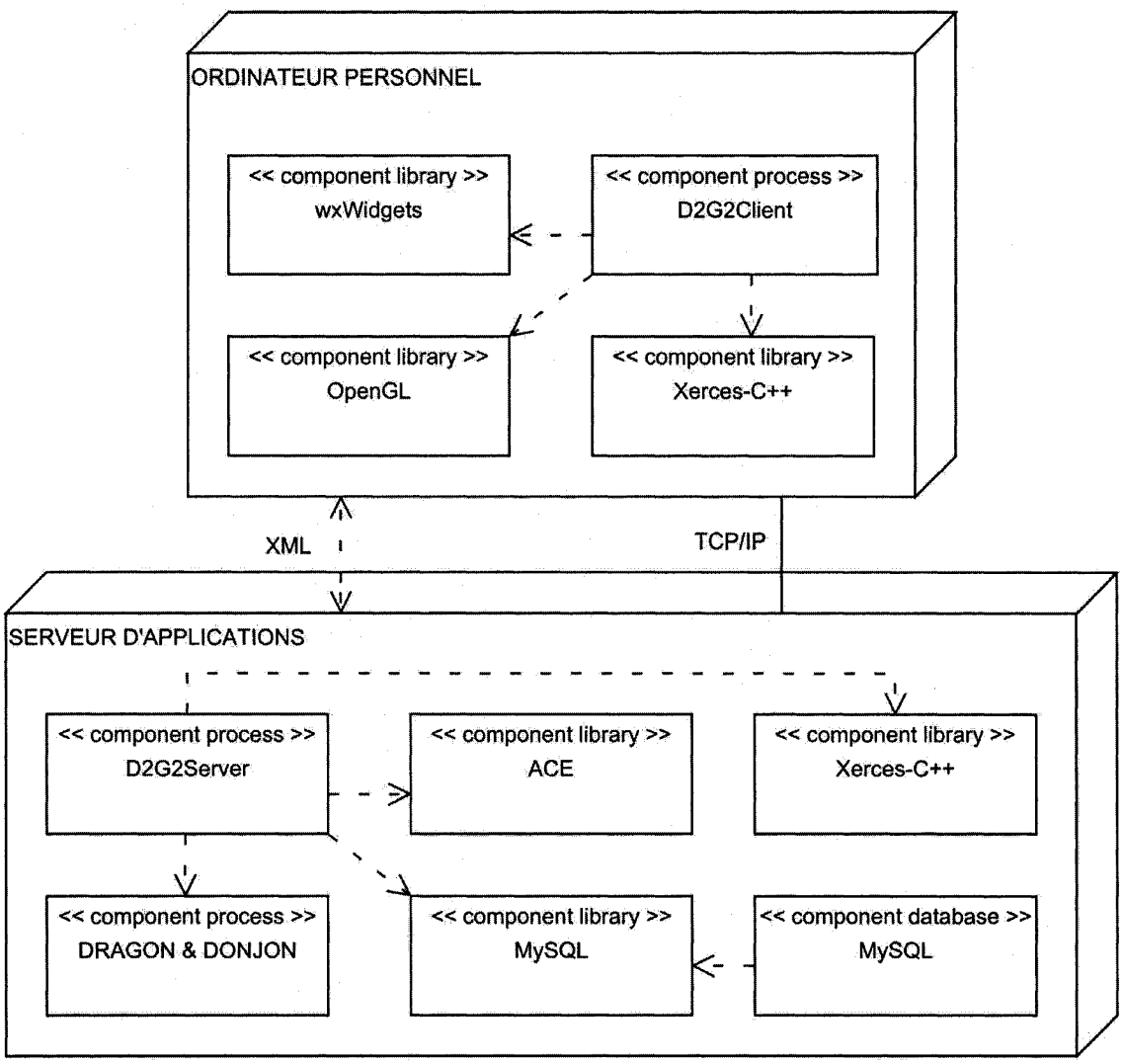

Figure 3.6 Diagramme de déploiement de l'architecture client-serveur

Ce diagramme définit deux entités matérielles : un ordinateur personnel et un serveur d'application. L'ordinateur personnel a comme composante et exécute le processus D2G2Client, faisant appel à diverses librairies. Cet ordinateur possède aussi une liaison avec le serveur d'application via le protocole TCP/IP. En fait, les échanges avec le serveur d'application sont tous sous forme de fichier XML. Le serveur d'applications exécute quant à lui le processus D2G2Server, faisant aussi appel à différentes librairies. Finalement, l'une des informations primordiales découlant de ce diagramme de déploiement est la représentation des applications DRAGON et DONJON en temps qu'instance de processus. En d'autres mots, ce déploiement signifie que les applications de neutronique sont utilisées sous leur forme originale dans l'architecture du D2G2Server. Elles correspondent en effet à des processus indépendants. Ce qui implique directement qu'aucune modification au code DRAGON ou DONJON n'est nécessaire.

3.2.4 Diagrammes de classes

Finalement, les derniers diagrammes de la modélisation statique, sans aucun doute les plus connus d'entre tous, sont les diagrammes de classes. Ils expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. Donc, étant donné l'importance de ces diagrammes dans le cycle de développement, il est normal qu'une analyse en possède plusieurs. Le premier diagramme (figure 3.7) correspond à une représentation générale des principales classes et de leurs interactions. Il permet d'apprécier et d'acquérir une vision globale du système. En effet, cette modélisation épurée permet de constater que le système est divisé en deux grandes parties : celle de gauche avec le singleton D2G2Server héritant des fonctionnalités de plusieurs classes lui permettant de fournir les services de serveur et celle de droite avec le Client_Handler servant de base à l'élaboration de tous les services relatifs au traitement des requêtes clientes. Les figures 3.8 et 3.9 correspondent

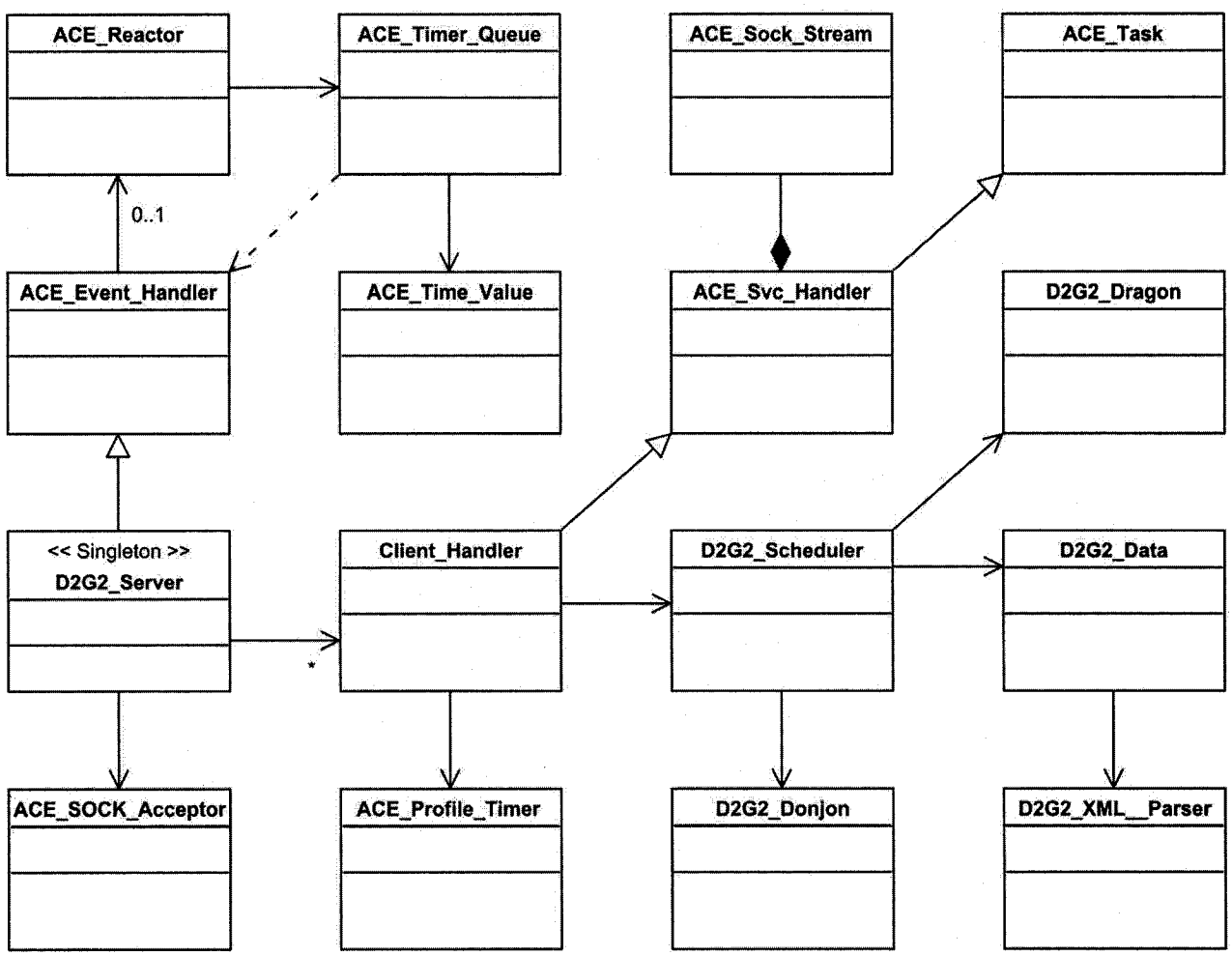

Figure 3.7 Diagramme de classes de la structure globale

quant à elles aux raffinements des classes de la figures 3.7 afin de préparer la future implémentation.

Maintenant, en ce qui a trait aux figures 3.10 et 3.11, celles-ci représentent l'agencement des classes entrant en relation dans le stockage en mémoire des données clientes. En effet, étant donné que plusieurs variables transitent par le serveur (variables de simulation DRAGON-DONJON, de contrôle, de requête), il est primordial de les regrouper dans une seule classe accessible par l'ensemble des modules de l'application.

Pour ce qui est du dernier diagramme (figure 3.12), celui-ci présente les entités entrant en relation dans le traitement des données : au niveau de la gestion du langage XML et de la sauvegarde dans la base de données.

3.2.5 Diagrammes objets

Les diagrammes objets montrent l'état du système à un instant donné à partir des modèles découlant des diagrammes de classes. Or, aucun diagramme objet n'est présenté. En effet, ceux-ci n'apporteraient aucune information nouvelle et pertinente ayant pu être ignorée par les diagrammes de classes, étant donné la simplicité de la présente analyse.

3.3 Modèle Dynamique

3.3.1 Diagrammes de séquence

Les diagrammes de séquence montrent les interactions entre les objets d'un point de vue temporel. Ils exposent en particulier les objets participant à l'interaction et la séquence des messages échangés. Ils sont généralement utilisés pour modéliser les aspects dynamiques et les scénarios complexes mettant en oeuvre peu d'objets.

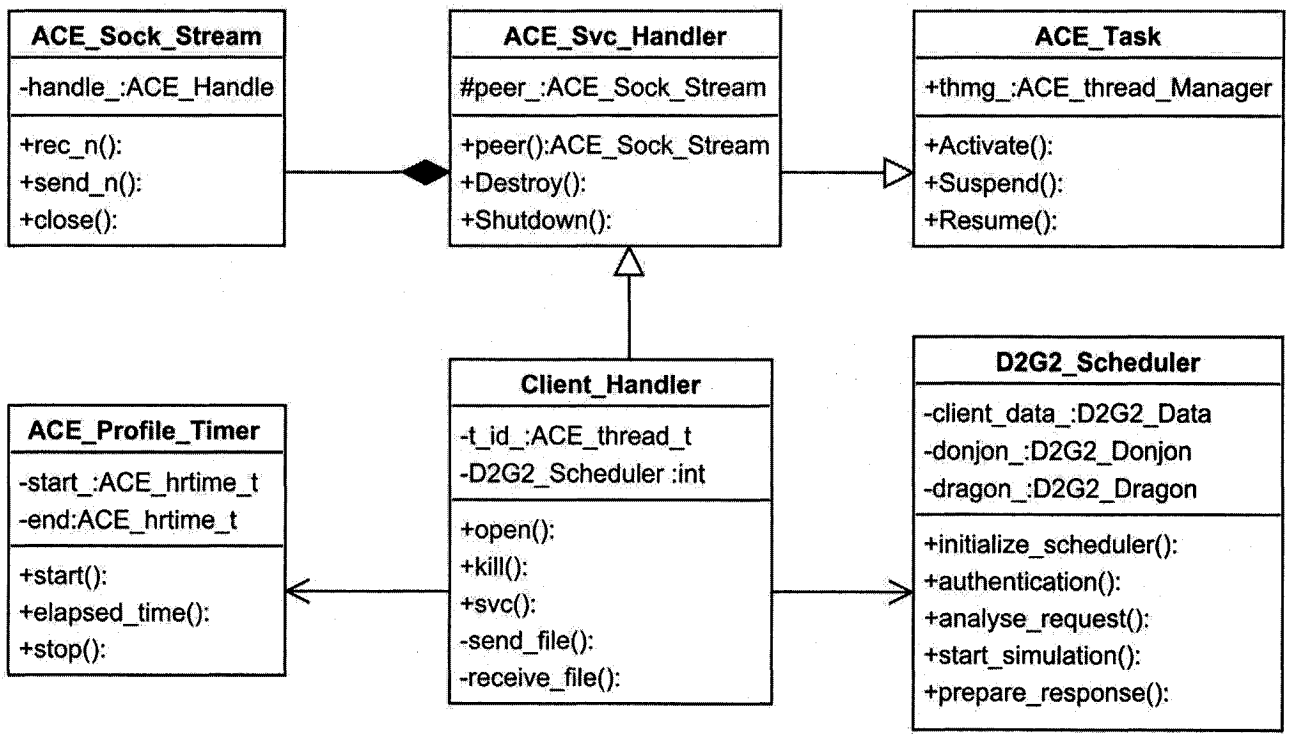

Figure 3.8 Diagramme de classes du domaine Client_Handler

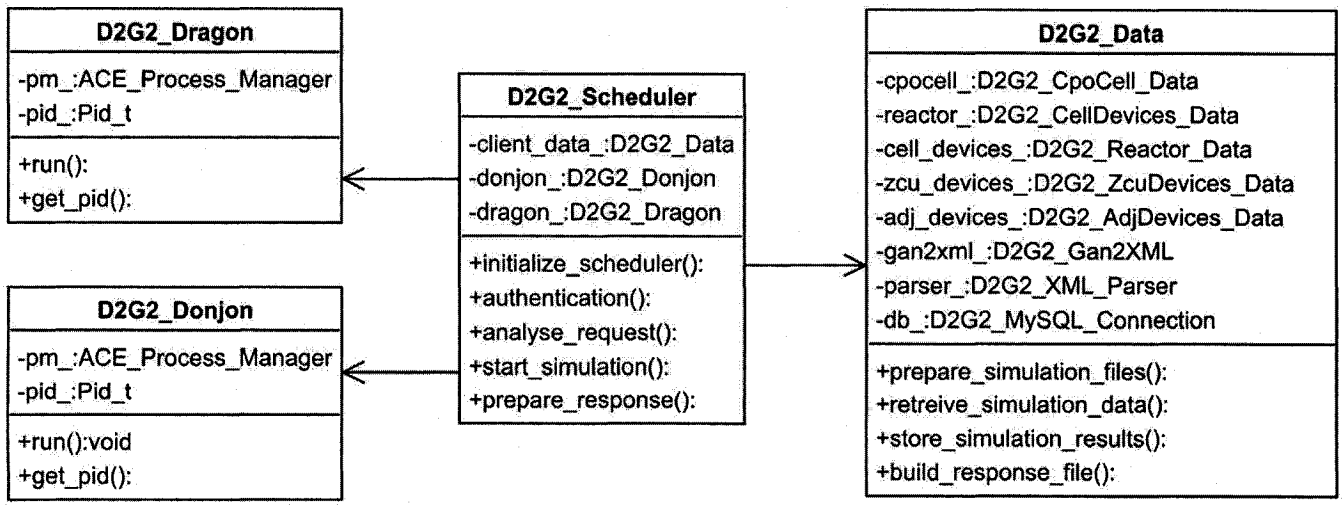

Figure 3.9 Diagramme de classes du domaine D2G2_Scheduler

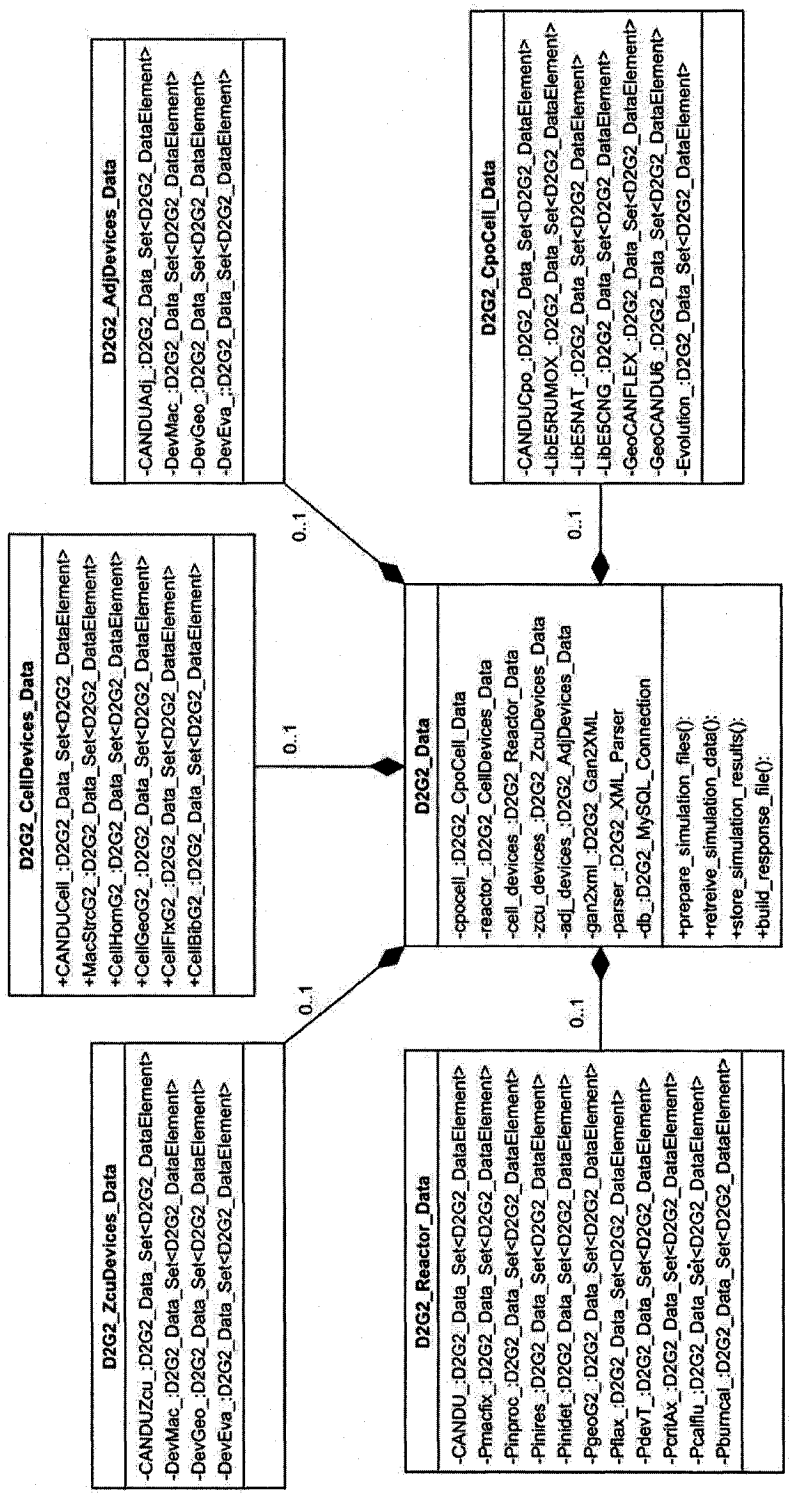

Figure 3.10 Diagramme de classes du domaine D2G2_Data (stockage des données)

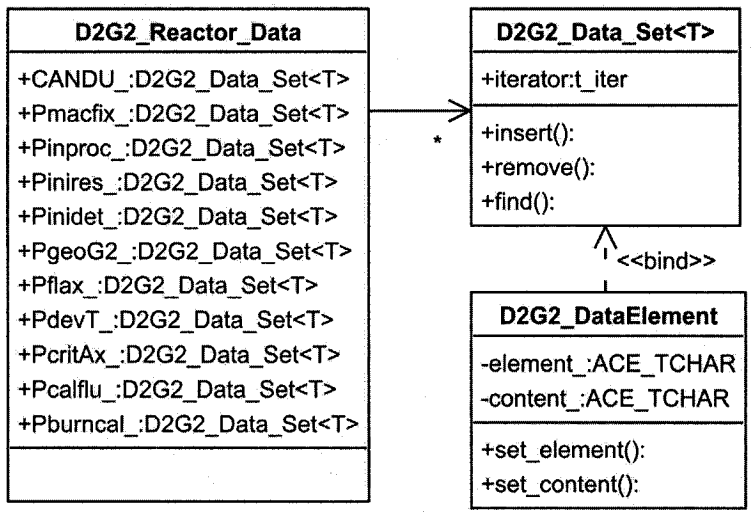

Figure 3.11 Diagramme de classes du stockage des données d'un réacteur

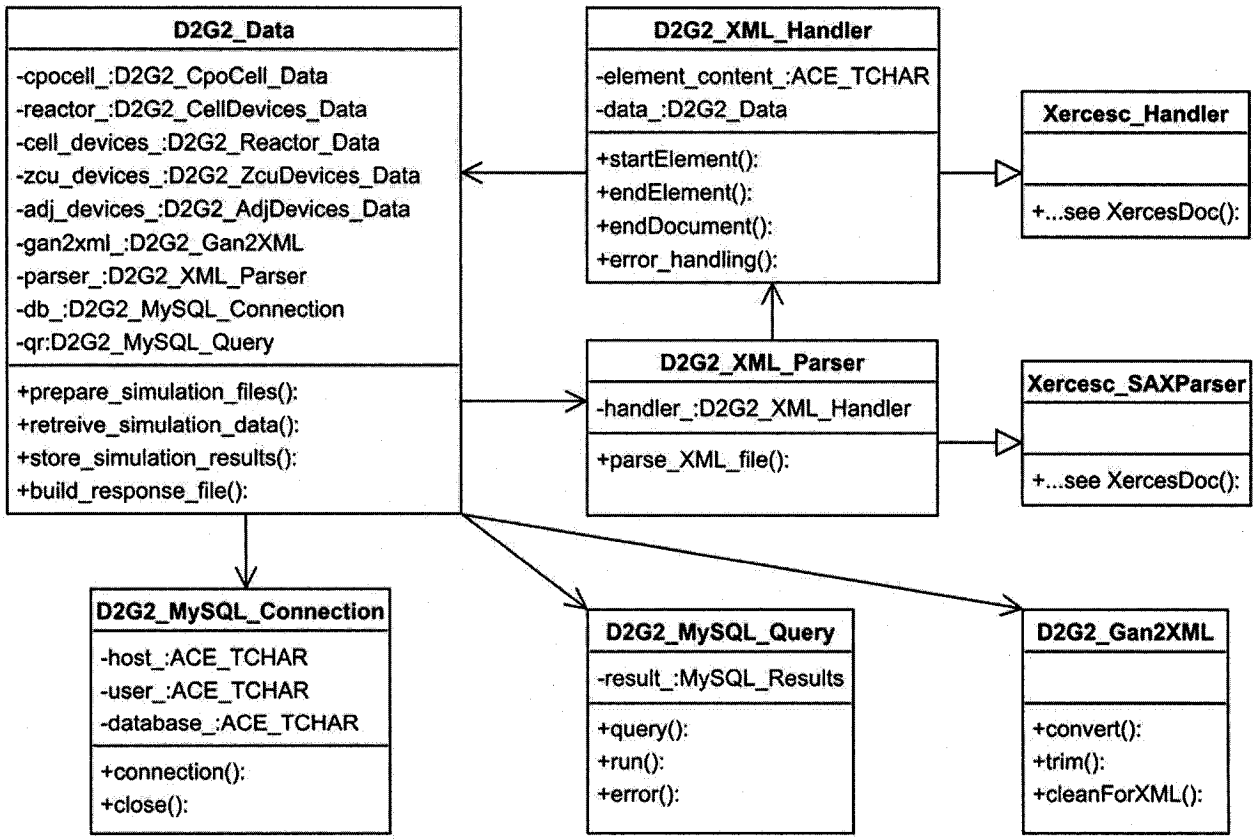

Figure 3.12 Diagramme de classes du domaine D2G2_Data (traitement de données)

Or, compte tenu des fonctionnalités des diagrammes de séquence, ils sont l'outil idéal pour modéliser un processus de requête client-serveur. Par exemple, la figure 3.13 expose une requête de simulation DRAGON. Le suivi temporel des échanges entre le client et le serveur et les différentes étapes menant à terme la simulation y sont clairement représentés. Il y est également exposé les différents moments d'activation des classes. En effet, grâce à ce diagramme, il est possible d'apprécier quand et comment les classes interviennent dans le processus de simulation.

3.3.2 Diagrammes de collaboration

Les diagrammes de collaboration tout comme les diagrammes de séquences sont des cas particuliers de diagramme d'interaction et représentent une vue dynamique du système. Les diagrammes de collaboration s'attardent sur les rôles joués par les objets dans un contexte particulier. Ils montrent également les interactions entre ces objets à travers une représentation schématisant les envoies de messages.

C'est pourquoi la figure 3.14 le diagramme de collaboration client-serveur insiste particulièrement sur la description du comportement et sur la communication entre le client et le serveur. Le diagramme expose en fait les différents messages qui peuvent être échangés entre les objets, représentant par conséquent les différents modes de requêtes pouvant être effectuées.

3.3.3 Diagrammes d'états-transitions

Les diagrammes d'états-transitions visualisent des automates à états finis du point de vue des états et des transitions des objets. Ils représentent en fait une abstraction de tous les comportements possibles d'un objet. La figure 3.15 expose justement les différents changements d'états d'une simulation.

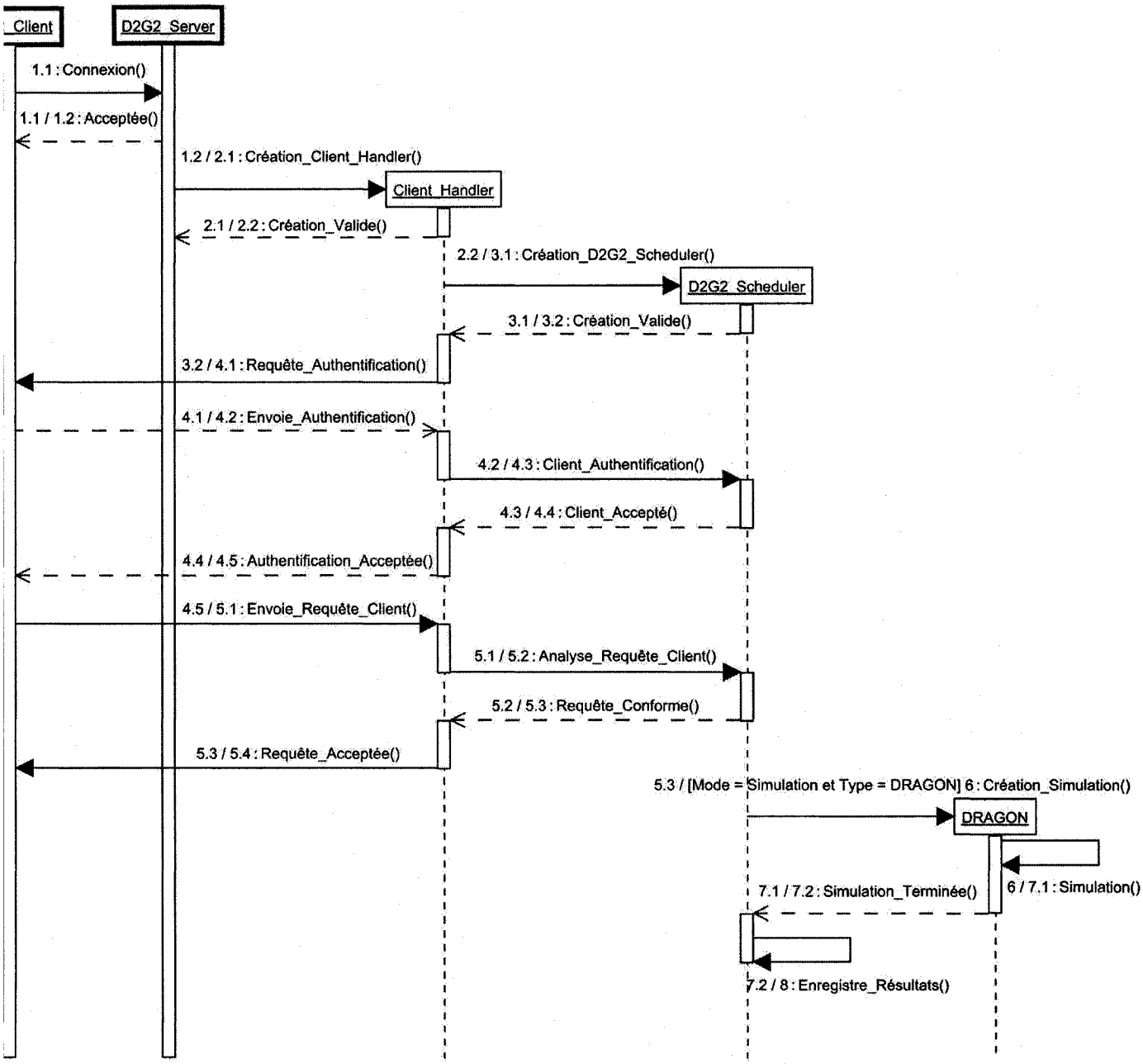

Figure 3.13 Diagramme de séquence d'un requête de simulation DRAGON

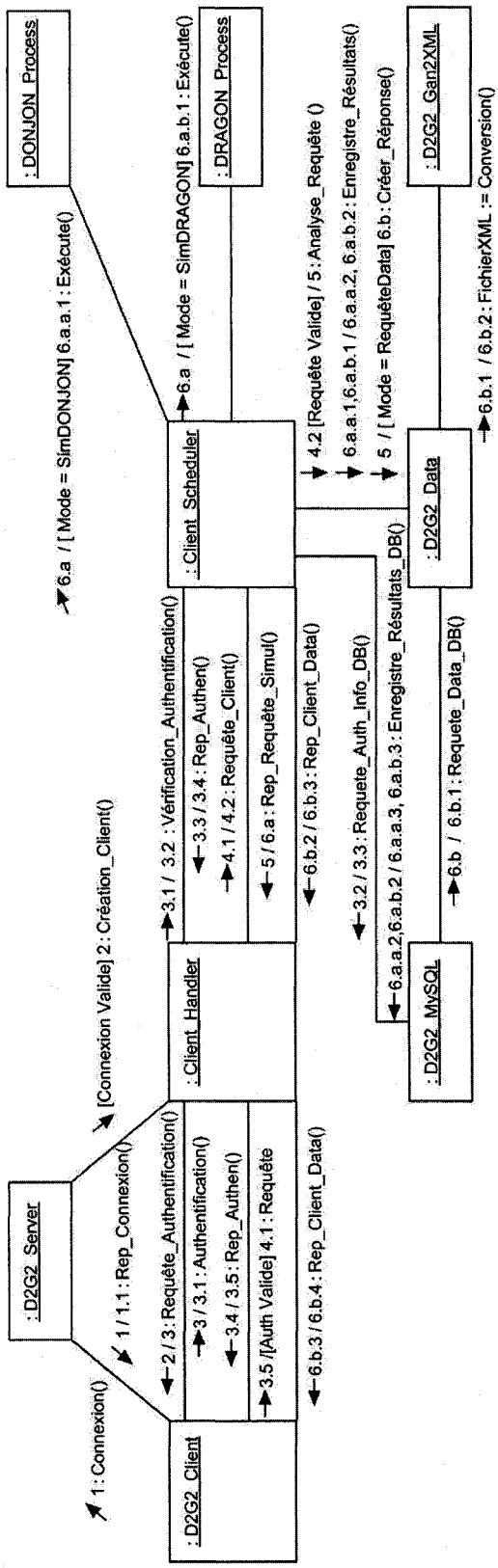

Figure 3.14 Diagramme de collaboration entre D2G2Client-D2G2Server

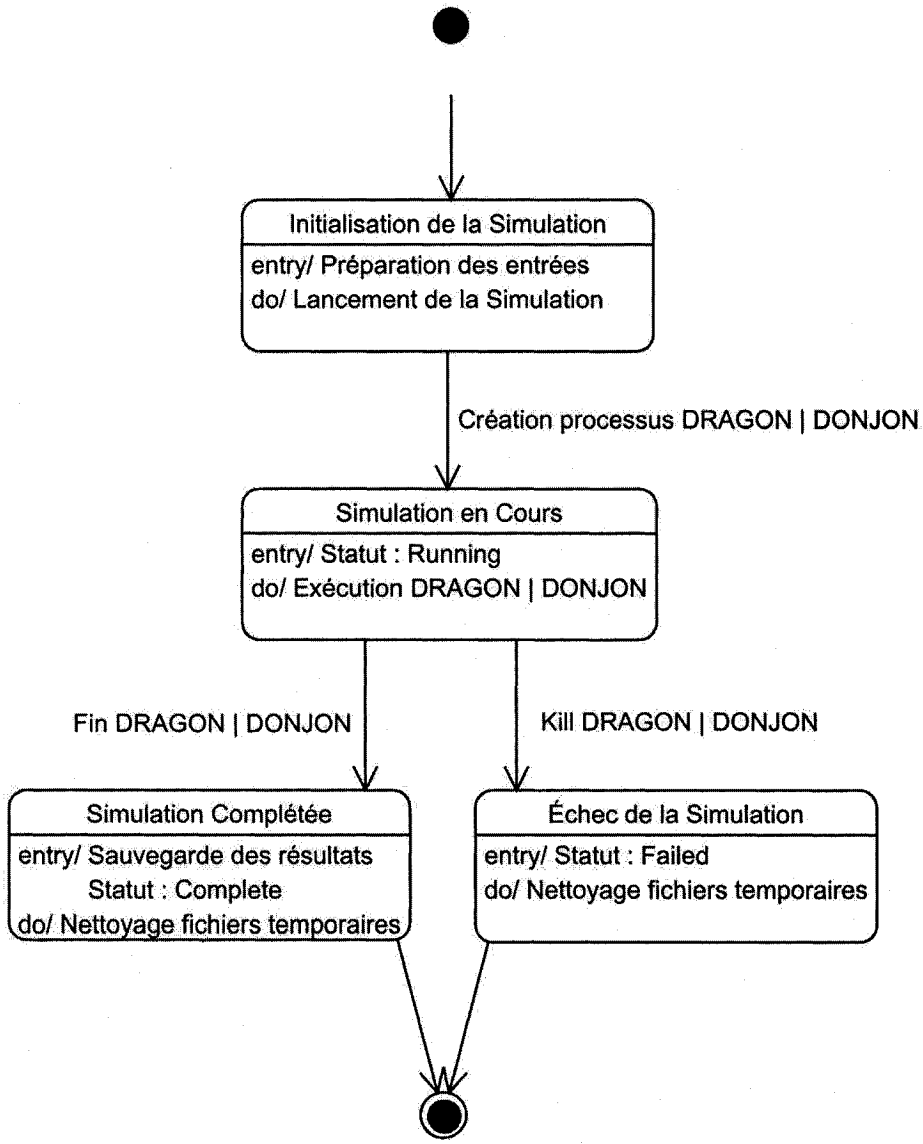

Figure 3.15 Diagramme d'états-transitions d'une simulation

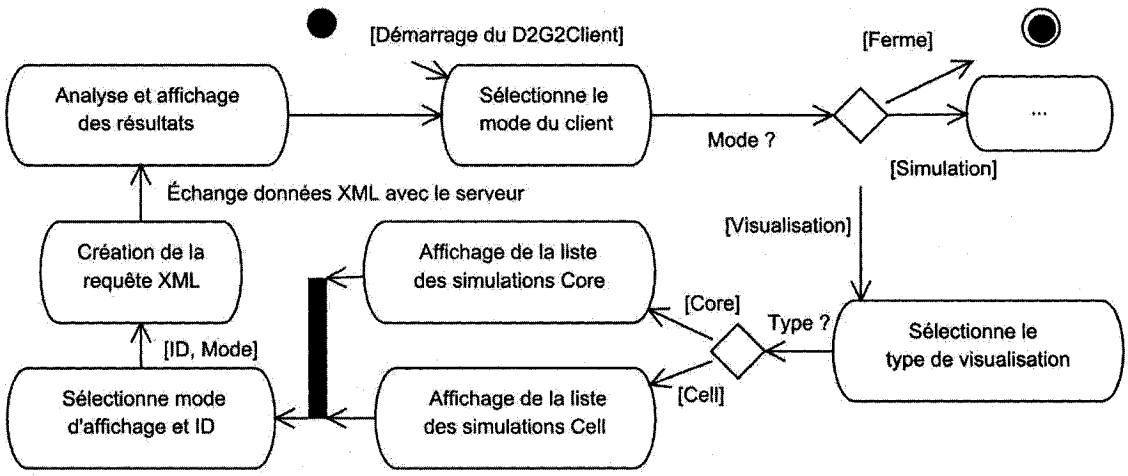

Figure 3.16 Diagramme d'activités du D2G2Client en mode visualisation

3.3.4 Diagrammes d'activités

Un diagramme d'activité est une variante des diagrammes d'états-transitions. Dans un diagramme d'états-transitions, les états et les transitions sont mis en avant alors que dans un diagramme d'activités, ce sont les activités et les transitions qui sont mises en évidence. Or, étant donné la grande similitude entre ces deux types de diagrammes, l'un d'eux est souvent ignoré de l'analyse. L'analyse du projet D2G2 a donc privilégié la création de diagramme d'activités pour la description des transitions du client. Les figures 3.16 et 3.17 retracent par conséquent les principales activités qui influencent les transitions au niveau du mode de simulation et du mode de visualisation. En effet, grâce à ces diagrammes, il est aisé de constater que lors d'une simulation ou d'une visualisation, seule la détermination des types mène vers un état différent du client. Cette modélisation suggère donc une quasi-linéarité de l'exécution du client, ce qui réduit considérablement la complexité d'implémentation.

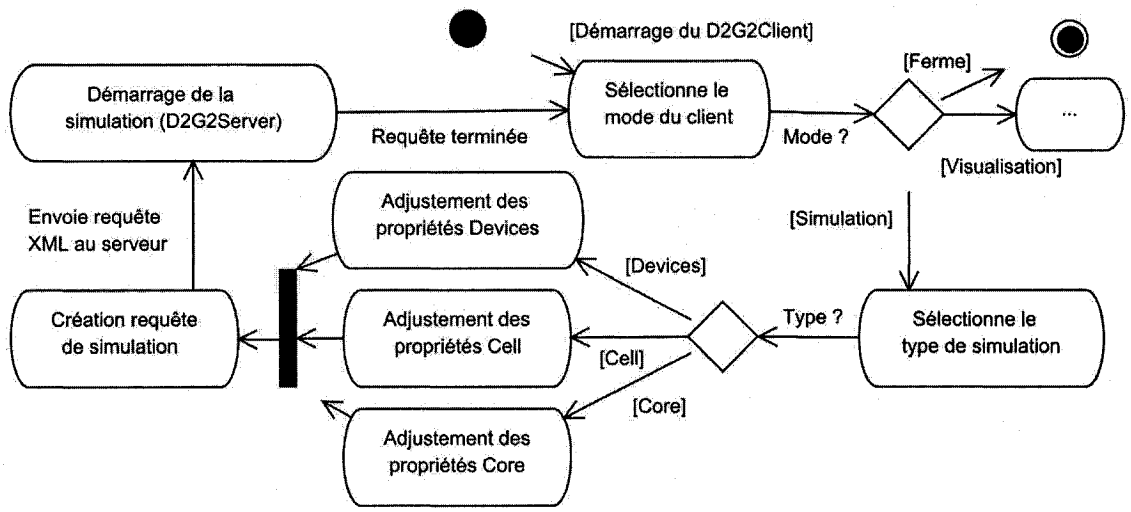

Figure 3.17 Diagramme d'activités du D2G2Client en mode simulation

3.3.5 Recoupement des phases

En terminant, il est normal de constater que les diagrammes d'analyse de ce chapitre sont passablement enrichis d'éléments de conception. En effet, le recoupement des phases d'analyse et de conception est plus qu'évident. Or, habituellement l'une des difficultés encourues durant le processus de conception logicielle est justement la perte de vue des éléments de l'analyse lors du passage à la conception. Donc, pour contrer cette tendance, un chevauchement des modèles a été privilégié. Ainsi, il est possible d'obtenir une plus grande cohésion entre les phases de développement. Il est donc conseillé de revenir fréquemment aux diagrammes présentés dans ce chapitre lors de la lecture du chapitre suivant, car celui-ci apportera plusieurs explications omises dans le présent chapitre. Celles-ci ayant été restreintes pour éliminer la redondance inhérente au processus de développement logiciel utilisé.

CHAPITRE 4

CONCEPTION

Habituellement la phase d'analyse et de conception, ainsi que la phase de conception et d'implémentation se chevauchent respectivemement. Alors, comme le chapitre précédent traitait de l'analyse et de certaines idées de conception, le présent chapitre s'attardera quant à lui au chevauchement de la conception avec l'implémentation.

La conception portera sur les décisions prises concernant les moyens nécessaires pour subvenir aux besoins spécifiés dans les chapitre 2 et 3. Bref, l'architecture du système sera décrite et l'implémentation de la composante serveur, nommée D2G2Server, et celle cliente, nommée D2G2Client, sera explicitée.

4.1 Architecture du système

L'architecture d'un système constitue la vue la plus générale de la conception d'une l'application. Elle détermine l'organisation du système en composants appelés sous-systèmes. Plus spécifiquement, elle établit le domaine d'activité de chaque sous-système et décrit leurs interactions. La conception débute donc par la détermination de l'architecture. Celle utilisée dans le cadre du projet D2G2 est pour sa part du type tableau noir.

4.1.1 Tableau noir

L'architecture du type tableau noir n'est aucunement une innovation récente, mais découle de nombreux travaux effectués depuis plusieurs années en intelligence artifi-

cielle. Plutôt méconnue et souvent confondue, l'utilisation de cette architecture n'est guère répandue hormis en AI[1]. Par contre, sa simplicité et son efficacité lui permettent peu à peu de s'imposer et d'étaler son domaine d'application. Justement, en ce qui concerne le projet D2G2, cette architecture s'adapte très bien aux contraintes de modernisation d'une chaîne de calcul patrimoniale.

Pour décrire l'architecture du type tableau noir, il est commode d'utiliser la métaphore à l'origine de son appellation.

> Imaginez un groupe de spécialistes de différents domaines d'expertise assis devant un tableau noir et utilisant celui-ci comme espace de travail pour résoudre un problème. Le fonctionnement est simple. La résolution du problème débute lorsque les données initiales sont affichées sur le tableau. Chaque spécialiste en prend alors note et tente de trouver par quel moyen il pourrait mettre son domaine d'expertise à profit. Ensuite, une fois qu'un des spécialistes juge qu'il pourrait formuler une contribution susceptible d'aider les autres membres du groupe, il s'empresse de l'inscrire sur le tableau. Ainsi, de contribution en contribution, le résolution du problème s'élabore jusqu'à l'obtention de la solution [26].

Bien qu'elle soit en apparence très banale, cette métaphore décrit fidèlement le processus de résolution au coeur de l'architecture du type tableau noir et permet de cibler adéquatement ses caractéristiques. Tout d'abord, le fait d'avoir plusieurs spécialistes de différents domaines permet d'obtenir une indépendance des expertises. Chaque membre du groupe pouvant mettre à profit son expérience personnelle sans toutefois se faire imposer des schèmes de pensée découlant d'une tendance généralisée au sein du groupe. Il est par conséquent plus aisé de regrouper des spécialistes de tout

[1]Acronyme anglais fréquemment utilisé pour référer au domaine de l'intelligence artificielle.

acabit ; ceux-ci jouant le rôle des sources de connaissances dans cette conjoncture de l'architecture du tableau noir. En somme, comme les spécialistes sont totalement indépendants, l'ajout, la modification ou le retrait d'un d'entre eux n'affecte en aucun temps la démarche de résolution via le tableau noir.

Or, quoique brève, cette introduction concernant le tableau noir permet de mieux saisir les diverses notions architecturales du projet D2G2. En effet, il est désormais plus évident (figure 4.1) de constater le rôle de gestion du tableau noir assigné au D2G2Server. Celui-ci est effectivement le pilote de contrôle du tableau et s'occupe de gérer toutes les données lui étant transmises, afin qu'elles soient disponibles à toutes les sources de connaissance telles que les D2G2Clients et les serveurs DRAGON et DONJON. Plus spécifiquement, les clients ont pour tâche de formuler et de déposer les problèmes sur le tableau, alors que les serveurs DRAGON et DONJON ont quant à eux la tâche de les résoudre en traitant les segments du problème qui entrent dans leur domaine de compétence. En fait, DRAGON récoltera les demandes de calcul de transport, alors que DONJON s'occupera de résoudre celles de diffusion. Finalement, les clients ont bien sûr la possibilité de parcourir le tableau pour évaluer l'avancement de la résolution des problèmes ou bien tout simplement pour consulter les solutions.

4.1.2 Client-serveur

Pour implémenter l'architecture du tableau noir dans le contexte du projet D2G2, une structure client-serveur est utilisée. C'est-à-dire une architecture où le client initialise toujours la communication vers le tableau noir, alors que le serveur gérant le tableau demeure continuellement en mode réactif. De plus, comme un protocole de communication entre client et serveur est défini, seulement une liste de requêtes préautorisées peut être effectuée. Donc, le D2G2Client construisant et transmettant les requêtes vers le serveur correspond bien à une source de connaissance, alors que le D2G2Server traitant les requêtes et gérant les résultats s'harmonise très bien avec

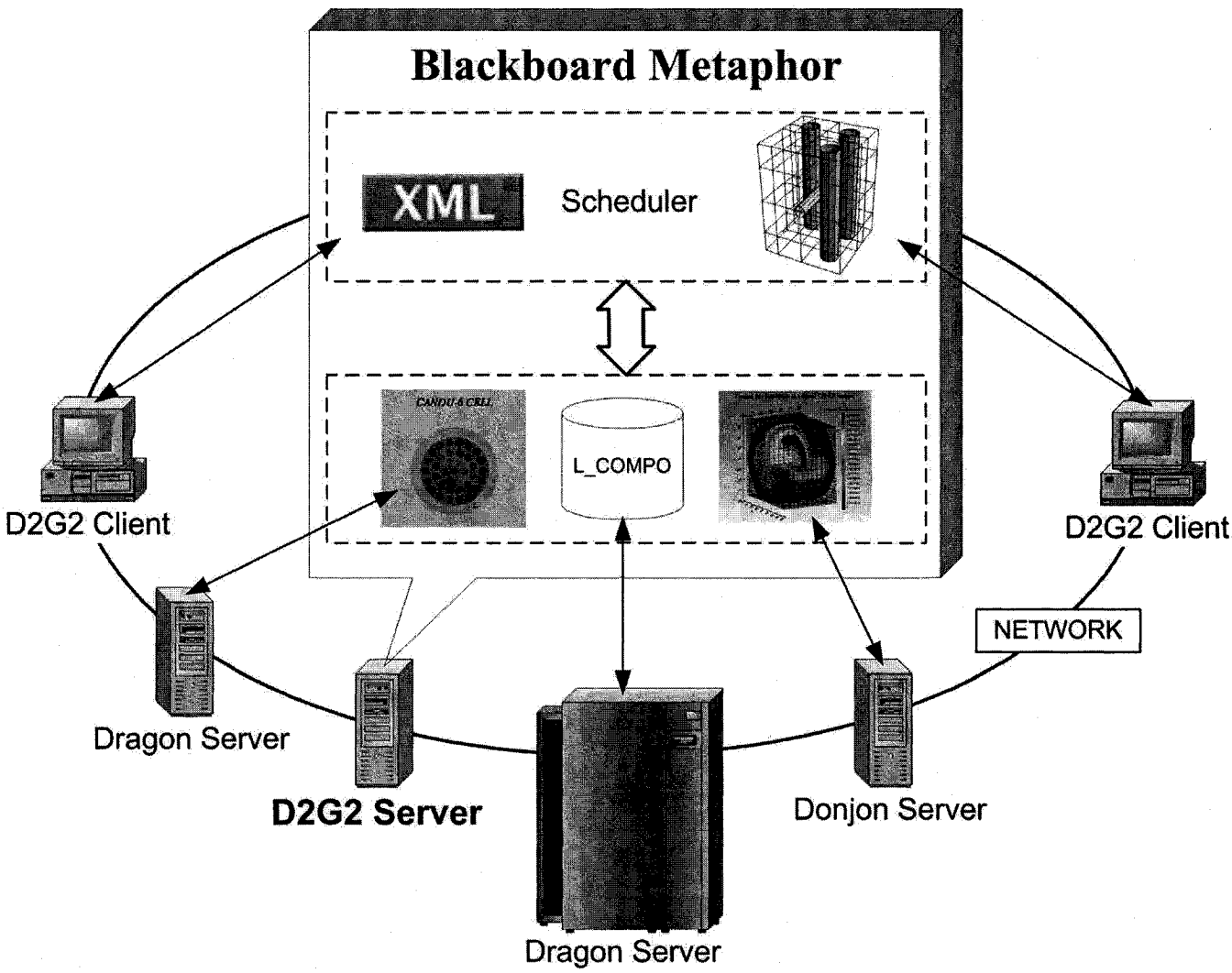

Figure 4.1 Architecture tableau noir du projet D2G2

le rôle de gestionnaire du tableau noir.

Par contre, il est bon de spécifier que lorsque le D2G2Server gère les sources de connaissances DRAGON et DONJON, il joue davantage le rôle d'un client, car c'est lui qui initialise la communication. Par conséquent, l'architecture du projet D2G2 n'est pas réellement une architecture client-serveur standard telle que spécifiée précédemment. En effet, elle est plutôt une architecture client-serveur à trois étages (figure 4.2) où le D2G2Server est en quelque sorte un intergiciel[2]. Celui-ci servant de pont entre les logiciels DRAGON et DONJON et le D2G2Client.

Donc, grâce à cette architecture client-serveur, les machines clientes comme la machine serveur fonctionnent plus efficacement pour exécuter les applications neutroniques. La partie client est optimisée pour l'interaction avec les utilisateurs, alors que le serveur possède une interface à accès unique et peut se concentrer sur le traitement à exécuter plutôt que sur l'interprétation des flots souvent irréguliers des entrées et sorties standards utilisateurs.

En résumé, l'utilisateur interagit avec le D2G2Client via une interface graphique. Celle-ci lui permet d'élaborer des requêtes de simulation parmi celles permises par l'interface du serveur et de parcourir le tableau noir pour visualiser des résultats. Le D2G2Server est quant à lui le gestionnaire du tableau noir. Il s'occupe de répertorier et analyser les requêtes de simulation, de manipuler les processus DRAGON et DONJON, de sauvegarder les données de calculs et finalement de fournir les résultats au D2G2Client.

Cependant, cette description de tâches correspond à celle s'articulant autour d'une

[2]Appellation française de *middleware*. À l'origine, le mot américain *middleware* a été crée afin de désigner une couche logicielle située entre le système d'exploitation (system software) d'un ordinateur et les logiciels d'application (software). De cette position médiane est né le mot *middleware*. De nos jours, la dénomination *middleware* est attribuable à toute application offrant une interface de communication commune pour l'intégration de logiciels d'origines diverses [27].

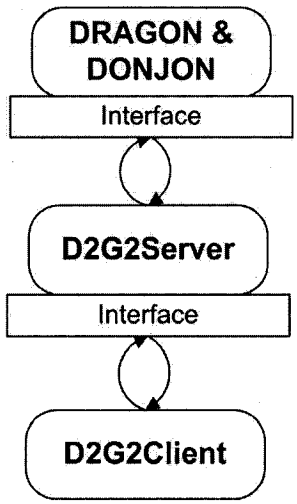

Figure 4.2 Modèle client-serveur à trois étages

relation simple entre un client et un serveur. En effet, cette description fait abstraction du nombre de clients et de serveurs. Par contre, étant donné que certains calculs DRAGON demandent énormément d'efforts de traitement, il fut impératif dans le cadre du projet D2G2 de s'attarder sur les possibilités d'optimisation des temps de calculs. En somme, après plusieurs essais, la méthode finalement privilégiée est une infrastructure de communication distribuée du type grille de calcul.

4.1.3 Calculs répartis

De nos jours, afin d'accélérer les simulations numériques requérant de longs temps de calcul, il est usuel de privilégier le traitement parallèle. Les simulateurs de neutronique n'échappent pas à cette tendance et de nombreuses recherches tentent continuellement de perfectionner la parallélisation des algorithmes de résolution des équations de transport et de diffusion [28]. Par contre, en ce qui concerne les logiciels DRAGON et DONJON, étant donné que ceux-ci n'offrent pour l'instant aucune possibilité de traitement parallèle dans leur version officielle[3], d'autres méthodes pour optimiser les

[3] À l'exception de certains travaux dont ceux sur la méthode des caractéristiques [29].

performances doivent être abordées. C'est dans ces circonstances que des techniques de calculs répartis ont été incorporées au sein du projet D2G2. En effet, la solution privilégiée consiste à installer et exécuter le D2G2Server sur le plus grand nombre de machines disponibles et de répartir automatiquement les requêtes de simulation sur ceux-ci. Plusieurs calculs peuvent ainsi être effectués simultanément, afin de profiter de tous les processeurs disponibles, sans toutefois bénéficier d'une parallélisation des logiciels DRAGON et DONJON.

Bref, ce modèle de répartition des tâches s'apparente fortement à une grille de calcul. Par contre, les détails de l'implémentation de cette infrastructure de calcul réparti ne peuvent être présentés dès maintenant. Une bonne compréhension du fonctionnement du D2G2Server est d'abord nécessaire pour être en mesure d'apprécier ce mécanisme de calcul distribué.

4.2 D2G2Server

Le D2G2Server est la composante principale du projet D2G2. Son implémentation est de loin plus avancée que celle du D2G2Client. Il peut même être considéré comme une application quasi complète malgré son développement plutôt orienté vers l'obtention d'un proprotype.

Pour présenter l'application, un découpage par classe et par fonctionnalités est privilégié. Toutes les classes ayant un rôle majeur au sein de la conception du serveur sont abordées : les classes implémentant les communications réseau (D2G2_Server), la gestion parallèle des clients (Client_Handler), la prise en charge des requêtes (D2G2_Scheduler). En ce qui a trait aux fonctionnalités, le transfert des données, le traitement des requêtes de simulation, la sauvegarde des résultats, le contrôle des applications neutroniques sont entre autres explicités.

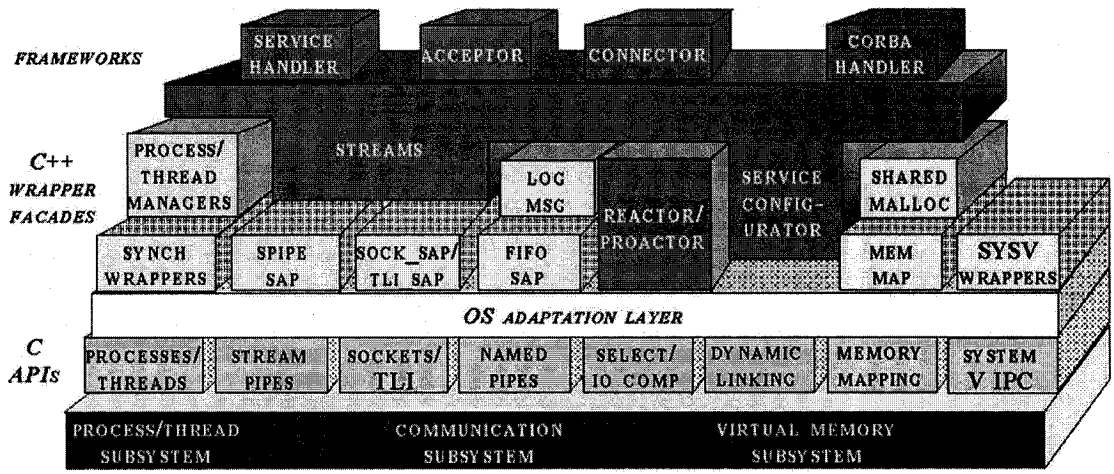

Figure 4.3 Structure générale de ACE

Or, il est tout d'abord important de spécifier que le choix d'un langage de programmation pour l'implémentation du D2G2Server n'a pas fait l'objet d'une longue étude. En réalité, l'utilisation du C++ découle directement du langage utilisé dans le cadre de travail[4] choisi. À vrai dire, c'est la recherche du cadre de travail le plus approprié qui a été très laborieuse. Par contre, depuis la découverte de ACE [30, 31](Adaptive Communication Environment), plus aucun doute ne persiste quant à savoir quel est le meilleur outil pour développer une application réseau stable et performante.

4.2.1 Application réseau multitâche

4.2.1.1 Présentation de ACE

ACE est un cadre de travail objet en C++ implémentant divers services de concurrence et de communication. Son utilisation permet de réaliser, dans un court laps de temps, des applications réseau portables, stables et performantes sur une grande variété de systèmes d'exploitation. Composé d'approximativement 500 classes, ce cadre de travail d'environ un quart de million de lignes de code découle directement des

[4]Traduction directe de l'expression anglaise framework

principes fondamentaux du paradigme objet. L'héritage, l'encapsulation et le polymorphisme permettent à ACE d'utiliser efficacement l'abstraction et les patrons de conception (design patterns). Il est donc possible avec ACE de réutiliser aisément des schémas d'implémentation complexes et éprouvés depuis plusieurs années. Par exemple, s'appuyant sur le patron de conception façade[17], ACE offre la même spécification pour tous les appels systèmes. La façade cachant tous les problèmes de portabilité en fournissant une spécification unifiée pour l'accès à l'ensemble des fonctionnalités des différents systèmes d'exploitation supportés par ACE. Ce cadre de travail possède donc une interface commune (figure 4.3) pour exploiter la plupart des fonctions systèmes, sans être contraint par les disparités inhérentes provenant de l'utilisation de systèmes d'exploitation différents. Les services réseau de haut niveau, offerts par la couche supérieure de ACE, sont ainsi implémentés uniformément quelle que soit la plateforme sur laquelle s'exécute le code. Plus spécifiquement, avec ACE, il est possible de créer des applications portables mettant en oeuvre des interfaces de connexion réseau (network sockets), des lancements de processus, des multitâches, de la synchronisation inter-processus, des accès fichiers, etc.

Par contre, un inconvénient majeur de ACE, comme la plupart des cadres de travail complexes, est qu'il demande un effort d'apprentissage considérable pour parvenir à le maîtriser convenablement. Néanmoins, une fois les connaissances acquises, son utilisation permet au développeur de bénéficier d'un savoir inégalé pour concevoir des applications réseau multitâches de haute qualité. Justement, le guide du programmeur[32] permet de bien comprendre la collaboration entre les classes et du même coup d'envisager les diverses possibilités de conception offertes par le cadre de travail ACE et celles utilisées pour concevoir le D2G2Server.

4.2.1.2 Conception avec ACE

Quant au projet D2G2, étant donné que son architecture est essentiellement basée sur des notions client-serveur, il est clair que l'utilisation de ACE est tout à fait propice pour son développement. En effet, l'utilisation de ce cadre de travail a permis de grandement accélérer le conception du D2G2Server, tout en assurant une implémentation de qualité axée vers une maintenance aisée.

Plus spécifiquement, le D2G2Server est un serveur multitâche basé sur un modèle d'accès simultané créant une tâche pour chaque demande de connexion. Le serveur peut ainsi prendre en charge plusieurs connexions simultanément[5]. L'implémentation du serveur découle directement de l'héritage des classes ACE_Event_Handler et ACE_Svc_Handler définies dans le contexte Acceptor-Connector de ACE. Cet héritage permet d'acquérir tous les services de communication réseau, d'accès simultané multitâche et de synchronisation. En d'autres mots, la création des fonctionnalités réseau multitâches du D2G2Server se résume à une redéfinition des fonctions membres de certaines classes du contexte Acceptor-Connector de ACE. Il est ainsi inutile de se préoccuper des routines de bas niveau nécessaires à l'ouverture des canaux de communication TCP/IP et de la synchronisation des ressources. Par conséquent, les erreurs relatives aux appels systèmes sont drastiquement réduites. De plus, il importe de rappeler que le code résultant de l'utilisation de ACE est portable sur la plupart des plateformes modernes : Win32, MacOS, OS/2, Linux, etc.

[5]L'utilisation du terme simultanément est plutôt abusif. Par contre, depuis l'avènement des systèmes d'exploitation uniprocesseur multitâche, cet abus de langage relatif à la simultanéité est plus souvent qu'autrement toléré. En fait, étant donné que les ordinateurs qui exécutent le D2G2Server possèdent pour la plupart seulement un processeur, il est impossible de traiter deux connexions réellement simultanément. Il faudrait plutôt mentionner que toute nouvelle connexion peut être acceptée par le serveur sans attendre que le traitement de celle en cours d'exécution se termine. Les demandes de connexion sont donc effectuées en alternance par échange mutuel des ressources du processeur, mais paraissent simultanées du point de vue du client.

4.2.1.3 Classe D2G2_Server

Plus spécifiquement, d'après la figure 3.7, la classe D2G2_Server hérite de la classe ACE_Event_Handler qui est enregistrée en temps que membre réactif dans la file d'attente événementielle du ACE Reactor. La classe D2G2_Server est donc à la base de toutes réactions entreprises par le serveur suite à la réception de stimuli extérieurs via son port de communication TCP/IP. Elle s'occupe aussi de mettre en place tous les mécanismes destinés à la gestion des flots de données (streams) entre les clients et le serveur. Elle joue ainsi le rôle d'aiguilleur en créant une instance de la classe Client_Handler pour chaque connexion cliente et en manipulant et dirigeant les différents flots de données vers cette nouvelle instance. De plus, étant donné son rôle de gestion avancée, il apparaît justifié que la classe D2G2_Server soit élaborée à partir d'un patron de conception singleton. En effet, pour assurer un contrôle uniforme des connexions actives sur le serveur, il est primordial d'avoir un seul et unique point d'entrée au coeur de l'application.

Finalement, parmi les autres tâches de la classe D2G2_Server, il y a entre autres l'initialisation du serveur à l'aide d'un fichier de configuration et la prise en charge des signaux de contrôle du système d'exploitation. Cette dernière caractéristique ayant été particulièrement ardue à implémenter pour le contrôle des signaux d'arrêt du serveur, il en sera question plus en détail à la section 4.2.4.3.

4.2.1.4 Classe Client_Handler

La classe Client_Handler, qui hérite quant à elle de la classe ACE_Svc_Handler, prend en charge toutes les connexions qui lui ont été attribuées par la classe D2G2_Server. C'est justement grâce à cet héritage de la classe ACE_Svc_Handler que le traitement multitâche du serveur peut s'articuler. En effet, il est possible de lancer des tâches parallèles (multithreads) en invoquant la fonction SVC() implémentée dans la classe

ACE_Task dont ACE_Svc_Handler hérite. Ainsi, pour chaque connexion assignée à une instance Client_Handler, il est possible de créer automatiquement à l'aide de la fonction SVC() de nouvelles tâches indépendantes. Ces nouvelles tâches s'occupant de gérer les requêtes provenant du client, libérant du même coup la tâche principale (main thread) de l'application. Le D2G2Server peut ainsi accepter promptement de nouvelles connexions.

Bref, une fois la construction et l'initialisation de la classe Client_Handler effectuée, la communication entre le client et le serveur peut alors débuter. Le client a accès à un canal dédié lui permettant de faire appel à toutes les fonctionnalités du D2G2Server.

4.2.2 Transfert de données

Or, une fois l'interface de connexion entre le client et le serveur bien établie, ces deux entités peuvent maintenant échanger librement des données. Par contre, afin que le serveur puisse interpréter adéquatement les demandes des clients, il faut qu'il ait été prélablablement convenu de la structure des données qu'ils échangeront. Il est donc normal qu'une normalisation des messages soit développée et si possible un langage standardisé utilisé.

En ce qui concerne le projet D2G2, l'utilisation du métalangage XML, avec lequel il est possible de concevoir sa propre normalisation de données, a été privilégée. Le serveur est alors en mesure de connaître à l'avance l'ensemble des messages que le client est susceptible de lui transmettre, grâce à la définition de type de document (DTD)[33]. En effet, l'utilisation d'une DTD permet de définir exactement la structure du document d'échange XML et pour que celui-ci soit valide, il est nécessaire que tous ces éléments soient indiqués dans la DTD. Ainsi, un nombre restreint de messages est permis et une vérification des données transférées est effectuée. De plus, l'utilisation d'une DTD permet l'ajout de valeurs par défaut pour les différents éléments du fichier

XML. Même s'il y a omission de certains paramètres dans le requête client, ceux-ci prendront alors automatiquement une valeur par défaut. Par exemple, la DTD utilisée dans le cadre des travaux de validation du présent ouvrage (annexe II.1) possède justement une valeur par défaut pour chaque variable paramétrée. Bref, grâce aux fonctionnalités du métalangage XML, le D2G2Client peut transférer sans faute et efficacement ces requêtes au D2G2server.

4.2.2.1 Classe D2G2_XML_Parser

L'utilisation d'un métalangage XML implique inévitablement l'emploi d'un analyseur (parser). Cet outil permet en effet de lire et de manipuler les documents *.xml.* Or, celui dont fait appel le projet D2G2 est développé par la société Apache Software Foundation[6] sous l'appellation Xerces-c. Par contre, bien que Xerces-c offre une panoplie de fonctionnalités pour le traitement du XML, par souci de simplification, le type d'analyseur employé dans le D2G2Server est un SAX (Simple API for XML). En effet, l'API de SAX est relativement élémentaire puisqu'il se caractérise par un traitement séquentiel des documents et par l'appel pour chaque balise XML d'une méthode correspondante. La simplicité de SAX a grandement facilité l'implémentation de procédures d'analyse XML, par contre comme tout développement simpliste, ce type d'analyseur possède un manque frappant de flexibilité. C'est justement en observant la classe D2G2_XML_Handler que cette lacune de SAX est frappante.

4.2.2.2 Classe D2G2_XML_Handler

La classe D2G2_XML_Handler est en quelque sorte une liste exhaustive des routines permettant la traduction en mémoire vive de chaque élément de la structure de document D2G2 (la DTD). C'est cette classe qui a pour tâche de traduire systématiquement toutes les données XML en objet dynamique stocké en mémoire. Par exemple, dans

[6]htttp ://www.apache.org

cette classe il y a une ligne qui teste si la balise soumise par l'analyseur SAX équivaut bien à *username*. Si le test est positif, alors le nom de l'usager placé comme élément de la balise est alors attribué au membre *username_* de la classe D2G2_Data.

En somme, il est clair que cette méthode d'analyse XML possède un manque flagrant de flexibilité, car un simple changement à la DTD doit impérativement être traduit dans la classe D2G2_XML_Handler. Sinon, les lectures subséquentes ne pourront guère être assurées, même que des erreurs d'exécution sont fortement probables. Par conséquent, il est juste d'admettre que c'est une faiblesse majeure au niveau des caparacités évolutives du D2G2Server. Ce sera justement un des sujets abordés dans le chapitre 6 traitant des limitations et des travaux futurs.

4.2.2.3 Classe D2G2_Data

La classe D2G2_Data est quant à elle en charge du stockage de toutes les informations découlant de l'analyse de la requête XML envoyée par le client. En fait, chaque élément du fichier XML est attribué à un membre privé de la classe D2G2_Data. Cette classe conserve toutes sortes d'informations, partant du nom de l'usager jusqu'au numéro d'identification du sous-calcul de cellule utilisé dans un calcul de réacteur. Donc, plusieurs structures de données peuplent cette classe et chaque module du D2GServer peut accéder à celle-ci pour obtenir les données envoyées par le client.

Par contre, il est important d'attirer l'attention sur certains membres des classes : Reactor_Data, CpoCell_Data, CellDevices_Data,ZcuDevices_Data et AdjDevices_Data. Ceux-ci sont effectivement très importants puisqu'ils sont en charge de stocker tous les paramètres de simulations DRAGON et DONJON. Ces objets sont en fait constitués d'une série d'ensembles indéfinis que l'on peut remplir et parcourir à l'aide d'étiquette et de leur valeur. Leur fonctionnement est simple. Pour ajouter une valeur à l'un de ces ensembles, il suffit d'insérer la valeur et son étiquette correspondante. Pour ex-

traire une valeur, il faut tout simplement utiliser l'étiquette associée. Ces ensembles jouent un rôle déterminant, car leur flexibilité aide grandement à l'élaboration des fichiers d'entrées du type GANlib. Il en sera question à la section 6.1.2.

4.2.3 Gestionnnaire de requêtes

4.2.3.1 Classe D2G2_Scheduler

Maintenant, une fois la connexion établie entre le client et le serveur et que les échanges de données sont standardisés, le traitement des requêtes du client peut s'amorcer. C'est la classe D2G2_Scheduler qui s'occupe de gérer le tout.

La première étape est l'authentification du client. Le client envoie, dans un fichier texte formaté en XML, son nom d'utilisateur et son mot de passe encodé en MD5[7]. Le serveur interroge alors sa base de données MySQL (annexe III) pour vérifier l'exactitude des informations fournies. Bien que simpliste, cette authentication est très importante. En effet, il est primordial que le client soit authentifié avant que le serveur réponde à ses requêtes, car certaines fonctionnalités, dont la suppression de simulations actives, doivent seulement s'opérer sur les données appartenant à l'utilisateur. Ce qui évite, par exemple, des pertes occasionnées par un utilisateur mal attentionné désirant libérer certaines ressources de calculs pour des fins personnelles.

Une fois l'authentification terminée, le serveur est maintenant en mesure d'accepter la requête du client. Celui-ci peut donc envoyer sa requête sous forme de fichier texte XML suivant la structure de document spécifiée par la DTD D2G2. Cette requête est ensuite analysée à l'aide de la classe D2G2_XML_Parser. Une fois l'analyse terminée et les données traduites en mémoire dans la classe D2G2_Data, la classe D2G2_Scheduler teste alors quel est le mode de requête dans lequel le client s'est engagé.

[7]http ://rfc.net/rfc1321.html

Or, d'après la DTD, il est convenu que l'utilisateur peut opter pour trois modes de requête : Simulation, RequestData et KillSimulation. Le premier mode correspondant à une demande de simulation DRAGON ou DONJON, le deuxième à une demande de données de simulation (flux, map, sortie standard) et le dernier traite l'interruption (KILL) d'une simulation. La classe D2G2_Scheduler oriente alors les efforts du serveur en fonction du mode requête demandé.

4.2.3.2 Requête de Simulation

Parmi les modes de requête énumérés précédemment, celui qui est le plus important est sans aucun doute la simulation. Il est non seulement le mode qui consomme le plus de ressources du serveur, mais il est également le mode dont le traitement doit être accompli sans faille. Chaque phase doit s'exécuter correctement et à la moindre irrégularité, le processus doit se terminer convenablement sans paralyser le serveur. Les étapes de la requête doivent donc s'enchaîner comme suit :

- État initial

  Le serveur a reçu la demande de connexion, a authentifié l'usager et vient d'analyser le fichier XML du client pour découvrir qu'il doit se mettre en mode simulation. Cette analyse (*parsing* XML) a du même coup modifié la classe D2G2_Data qui contient désormais tous les paramètres de simulation transmis par le client.

- Calculs répartis

  Une fois l'état initial du mode simulation atteint, le serveur entame la phase de répartition réseau des efforts de calculs. Il interroge alors la base de données pour connaître le nombre de simulations en cours d'exécution sur les autres D2G2Servers actifs. Il détermine ensuite s'il doit soumettre la tâche à un autre serveur ou bien s'il doit l'exécuter localement. La décision est prise en fonction du nombre de simulations actives sur les autres serveurs. Or, si un serveur pos-

sède un nombre de simulations inférieur à celui en cours d'exécution localement, la requête doit alors être transférée. Pour ce faire, le serveur étant le moins occupé est contacté et le fichier XML que le client avait transmis auparavant lui est intégralement envoyé.

Ainsi, aucune routine spécifique de communication serveur-serveur n'a dû être ajoutée. Le serveur à qui la requête est resoumise, la traite comme si elle provenait d'un client. Les requêtes se propagent donc automatiquement sur tous les D2G2Servers actifs. Ce partage est en fait la base du mécanisme de calculs répartis du projet D2G2. Grâce à cette fonctionnalité, il est possible d'optimiser la puissance des calculs DRAGON et DONJON en profitant au maximum de tous les ordinateurs possédant une installation du D2G2Server.

Cependant, il est important de souligner qu'il existe une contrainte d'expansion au niveau de cette grille de calcul. En effet, tous les serveurs doivent avoir accès en écriture à un emplacement commun pour l'enregistrement des données. C'est pourquoi la configuration testée dans le cadre de cet ouvrage se limite à quelques ordinateurs connectés sur un réseau local à un espace disque partagé (NFS[8]).

- Simulations actives

Maintenant, peu importe si la requête a été retransmise vers un autre serveur ou non, lorsque celle-ci atteint le serveur sur lequel elle doit s'exécuter, la procédure de traitement demeure la même. La première tâche que le serveur exécute est celle de l'inscription de la requête de simulation dans la base de données. Pour ce faire, le serveur insère dans la table des simulations actives divers paramètres de contrôle : adresse IP du serveur qui exécute la simulation, nom de l'usager qui en fait la demande, commentaires et statut de la simulation (*running*). Ensuite, le serveur modifie la table des D2G2Servers actifs pour y mettre à jour le nombre de simulations qu'il est en train d'exécuter. Bref, grâce à ces

[8]Network File System : http ://nfs.sourceforge.net

informations dans la base de données, il est possible de connaître en tout temps l'état des simulations actives et leur répartition sur les différents D2G2Servers.

- Construction des fichiers d'entrées

  Une fois les variables d'état du serveur mises à jour, l'étape suivante est de construire les fichiers d'entrées GANlib requis pour le lancement de DRAGON et DONJON. Or, à ce stade-ci, tous les paramètres transmis par le client sont accessibles via la classe D2G2_Data. Par contre, bien que toutes les données de simulation du fichier XML soient en mémoire, il n'en demeure pas moins que l'élaboration des fichiers GANlib est une tâche ardue.

  En effet, bien qu'une procédure de génération automatique de ces fichiers à partir de la sémantique et de la syntaxe des modules DRAGON-DONJON aurait été la méthode la plus flexible, le trop grand nombre de possibilités à considérer dépasse largement le cadre du présent ouvrage. Donc, une méthode moins élégante mais très fonctionnelle a été privilégiée. Elle correspond à une élaboration des fichiers GANlib par substitution de variables.

  Plus précisément, grâce à la librairie libTPT[9], il est possible d'utiliser un fichier modèle (*template*) dans lequel certaines variables peuvent être paramétrées. Le mécanisme de substitution est le suivant. Chaque variable à paramétrer à l'intérieur du fichier modèle doit être entourée d'accolades et précédée du signe de dollars : "${FOO}". Ensuite, un appel à la fonction SET de la librairie libTPT permet de faire la substititution de la variable avec la valeur soumise en argument. Par exemple, la variable FNAT retrouvée dans le fichier modèle sous la forme ${FNAT} sera remplacée par « MonFichierFnat » à l'aide de la commande suivante : SET(ACE_TEXT("FNAT"), ACE_TEXT("MonFichierFnat")).

  Bref, grâce à ce mécanisme de substitution, il suffit d'inclure aux fichiers de

[9] http ://tazthecat.net/~isaac/libtpt/

simulation GANlib les variables libTPT devant être modifiées par les données de la classe D2G2_Data. À ce titre, la figure 4.4 résume les phases de construction en partant des valeurs soumises par le client dans le document XML, en passant par le fichier modèle libTPT, jusqu'au fichier d'entrées GANlib.

- Lancement de la simulation

  L'étape suivante est le lancement de la simulation. En effet, ayant élaboré à l'étape précédente tous les fichiers d'entrées GANlib, il est maintenant possible d'exécuter DRAGON ou DONJON. Ainsi, dépendamment du type de simulation demandée (réacteur, cellule, mécanismes de contrôle) le D2G2Server prépare un dossier temporaire où tous les fichiers d'entrées y sont stockés. Le serveur s'occupe aussi d'y copier les librairies de sections efficaces microscopiques requises si la simulation est exécutée par DRAGON. Il se charge également d'y copier tous les fichiers dépendants provenant de simulations antérieures. Par exemple, il y copie les fichiers provenant des simulations de mécanismes de contrôle et d'évolution du combustible nécessaire lors d'une simulation DONJON d'un réacteur complet.

  Or, une fois tous les fichiers requis en place dans le dossier temporaire, le serveur lance un processus externe DRAGON ou DONJON et attend la fin de son exécution. Il est important de rappeler que ce processus est réellement externe. Le D2G2Server utilise effectivement les applications de neutronique sans aucune modification. En effet, il les utilise exactement la méthode spécifiée dans la documentation DRAGON-DONJON : soit de soumettre un fichier texte via l'entrée standard de l'environnement (SHELL) duquel l'application est lancée.

  Bref, une fois le lancement effectué, le D2G2Server étant multitâche, il n'est guère paralysé durant l'attente de la fin de l'exécution de DRAGON ou DONJON. Il peut ainsi continuer à accepter d'autres requêtes clientes.

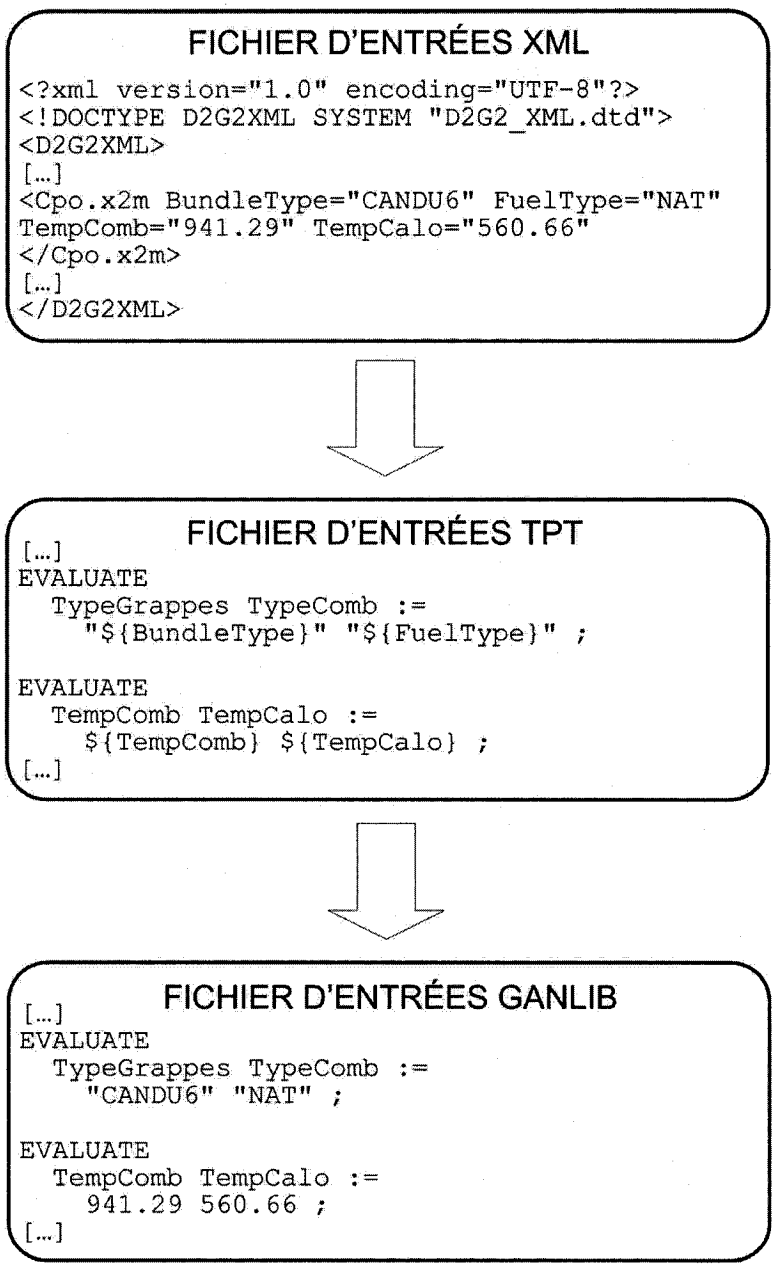

Figure 4.4 Construction d'un fichier GANlib

- Sauvegarde des résultats

  La dernière étape de traitement lors d'une requête de simulation est la sauvegarde des résultats. Ainsi, lorsque le processus DRAGON ou DONJON se termine, le D2G2Server enregistre dans la base de données MySQL les différentes données de contrôle : nom de l'usager, commentaires, adresse IP du client, adresse IP du serveur, date et durée de l'exécution et finalement le statut (complete). Le serveur enregistre de plus les résultats spécifiques à la simulation exécutée. Par exemple, dans le cas d'une simulation de cellule il y a sauvegarde du fichier contenant l'évolution des sections efficaces en fonction du taux de combustion, alors que dans le cas d'une simulation de réacteur il y a sauvegarde des fichiers de flux et de celui contenant la distribution de puissance et les données relatives au combustible.

- Fin de la requête

  Finalement, une fois la sauvegarde des résultats achevée, la requête est terminée. Le serveur supprime alors la simulation de la liste des simulations actives et diminue son compteur, dans la table des D2G2Servers actifs, indiquant le nombre de simulations qu'il est en train d'exécuter. Le serveur effectue ensuite sa dernière tâche qui consiste en la suppression de tous les fichiers temporaires qui étaient nécessaires à l'exécution du processus DRAGON ou DONJON. Par exemple le fichier de cellule (FuelCompo), ceux des mécanismes de régulation (Adjuster1, Ajuster2...) et tous les fichiers d'entrées GANlib sont supprimés du dossier temporaire.

4.2.3.3 Requête de Données

Le mode de requête de données peut s'effectuer sur deux groupes de données. Le premier groupe correspond aux résultats de simulation. Le client fournit alors le type et le numéro ID de la simulation pour laquelle il désire consulter les résultats. Le

deuxième groupe correspond quant à lui à toutes les données relatives au contrôle des simulations (liste des simulations actives, liste des serveurs actifs, liste des simulations de réacteurs exécutées, liste de simulations de cellule, etc). Le client soumet alors une requête de données du type liste et le serveur lui retourne toutes les informations contenues dans la base de données. L'utilisateur n'a par conséquent pas à fournir de numéro ID de simulation puisque le serveur retourne toute la liste des entrées de la table interrogée.

Maintenant, en ce qui a trait aux opérations effectuées par le serveur pour répondre aux requêtes de données, elles sont peu nombreuses comparativement à celles durant la phase de simulation. Le serveur reçoit comme toujours la requête XML et l'analyse. Lorsqu'il constate que le client est en mode requête de données, il vérifie quel type de données il doit fournir, extrait les informations contenues dans la table de la base de données MySQL, construit le fichier XML de réponse et le retourne au client.

Il est cependant important de spécifier que le traitement des requêtes de résultats de simulation est quelque peu différent de celui correspondant aux listages de simulations. En effet, étant donné que ces résultats sont stockés dans des fichiers à l'extérieur de la base de données, le serveur ne peut qu'extraire le chemin d'accès (*path*) de ces fichiers. Il doit donc par la suite accéder à ce fichier à l'emplacement indiqué (habituellement le chemin d'accès pointe vers un espace disque partagé NFS) avant de construire le fichier XML de réponse. Il est aussi important de noter qu'étant donné que la plupart des fichiers de résultats DRAGON-DONJON se retrouvent sous le format GANlib, il est requis d'effectuer un traitement supplémentaire pour les convertir au format XML, avant de les retourner aux clients. Pour ce faire, le D2G2Server utilise des classes de conversion nommée GantoXML[10]. Ces outils de conversion ont été spécialement conçus pour lire une structure hiérarchique (fichier XSM) et la transposer

[10]Ces outils ont été développés par le professeur Koclas. Aucune modification majeure n'y a été effectuée.

dans un fichier XML.

4.2.3.4 Annulation de simulation

Le dernier mode de requête accessible par le D2G2Client est celui de l'annulation d'une simulation. Ce mode est effectivement très utile lorsqu'un utilisateur s'aperçoit qu'il a soumis une requête erronée et que ce calcul prend habituellement plusieurs minutes, voir même plusieurs heures à se compléter. L'annulation de ces simulations inexactes aide donc à libérer les ressources de calcul. Bien entendu, seul le propriétaire ou l'administrateur peut stopper une simulation. En aucun temps un utilisateur n'est en mesure d'annuler une exécution soumise par un autre utilisateur.

Le processus d'annulation est simple. Lorsque le D2G2Server reçoit une requête et constate que le client est en mode d'annulation (*kill* simulation), il utilise le numéro ID soumis par le client et extrait les informations requises à partir de la table des simulations actives. Ensuite, il vérifie si l'usager qui a soumis la simulation est bel et bien le même qui désire l'annuler. Finalement, si l'identité concorde, le serveur envoie alors un signal d'arrêt au processus DRAGON ou DONJON en cours d'exécution, change le statut de la simulation pour « failed » dans la table des simulations actives de la base de données et supprime tous les fichiers temporaires (figure 3.15).

4.2.4 Détails techniques

4.2.4.1 Fichier de configuration

Parmi les fonctionnalités intéressantes du D2G2Server, il y a en une qui paraît banale à première vue, mais qui est en fait d'une grande importance. Le fichier D2G2Server.conf, contenant tous les paramètres d'initialisation du serveur, est effectivement crucial. En effet, c'est dans ce fichier, lu par le serveur à son démarrage, que tous les paramètres du serveur, ceux de l'accès à la base de données MySQL, du

réseau TCP/IP, des applications DRAGON et DONJON, des fichiers de simulation, de l'emplacement de la DTD et des modes de débogage sont spécifiés. Il est donc primordial de s'assurer de la conformité des informations inscrites dans ce fichier de configuration. Certes, bien qu'une procédure de vérification de ces paramètres soit lancée au démarrage du serveur, l'utilisation d'une mauvaise version de DRAGON est une erreur humaine que le serveur ne peut guère détecter.

4.2.4.2 Sauvegardes binaires MySQL

Tel que spécifié précédemment, tous les D2G2Servers doivent avoir accès à un emplacement disque partagé pour sauvegarder les résultats (NFS). Le pourquoi d'une telle restriction découle de piètres performances de stockage de champs binaires (blob) en utilisant une base de données MySQL. En effet, plusieurs essais [34] ont démontré que sauvegarder seulement les chemins d'accès (path) des fichiers de résultats et d'y accéder autrement que via MySQL augmente considérablement les performances.

4.2.4.3 Arrêt du serveur

Une autre technicalité à préciser au sujet de l'implémentation du D2G2Server est sa routine d'arrêt. En effet, l'arrêt d'un serveur multitâche comme le D2G2Server comporte plusieurs difficultés. Il est effectivement primordial d'arriver à synchroniser l'interruption des tâches parallèles en cours d'exécution afin de ne pas compromettre l'intégrité de leurs données et le suivi de celles-ci. Ainsi, lorsqu'un signal d'arrêt est envoyé au serveur (SIGINT, SIGTERM, SIGQUIT) celui-ci s'occupe de tuer tous les processus DRAGON ou DONJON en cours de simulation avant de s'arrêter. Du coup, cette terminaison forcée déclenche une cascade événementielle découlant de l'arrêt des processus DRAGON ou DONJON. Cette série d'événements est entre autres constituée de la suppression des fichiers temporaires de la simulation active et de modifications à la base de données afin de changer le statut à *failed* pour toutes les

simulations perdues. Ainsi, l'utilisateur qui parcourt la liste des simulations actives est informé du non-succès de l'une de ces requêtes. Il peut donc resoumettre sa requête si d'autres serveurs sont actifs ou lorsque celui qui vient de s'arrêter est redémarré.

Par contre, cette procédure d'arrêt, terminant adéquatement toutes les simulations en cours de calcul, ne peut guère être entamée lorsque l'arrêt du serveur n'est pas souhaité. C'est-à-dire, lorsqu'une cause extérieure, autre que la réception de signaux d'arrêts, est à l'origine de la fermeture du serveur. Il est alors impossible de terminer, de nettoyer et de mentionner la perte des simulations en cours d'exécution. Il serait donc souhaitable d'implémenter une procédure de récupération plus poussée. Or, la non-existence de cette procédure de récupération dans l'implémentation du D2G2Server s'inscrit dans la liste des limitations fonctionnelles du serveur et sera abordée dans le dernier chapitre du présent ouvrage.

4.2.4.4 Mode de débogage

Finalement, une autre particularité intéressante pour le suivi de l'exécution du serveur découle de l'utilisation du cadre de travail ACE. Manifestement, il est essentiel de pouvoir s'assurer du bon déroulement des traitements effectués par le serveur. Pour ce faire, de nombreuses sorties de débogage parsèment le code du D2G2Server. Celles-ci s'avèrent justement très utiles pour la détection des erreurs, mais aussi pour tout développeur désirant comprendre le fonctionnement du serveur. Bien entendu, il est possible de les désactiver ou de les rediriger vers les fichiers de *logs* du système d'exploitation.

4.3 D2G2Client

Le D2G2Client est l'interface utilisateur graphique[11] du projet D2G2. Il a pour objectif de diriger les utilisateurs dans les phases de prétraitement et de post-traitement de calculs neutroniques DRAGON et DONJON. Son développement, quoi qu'aujourd'hui au stade embryonnaire, est d'une importance capitale. En effet, le manque d'outils graphiques dans les codes neutroniques actuels constitue en soi l'une de leurs principales lacunes. Sans un support graphique adéquat, l'utilisation de logiciels comme DRAGON et DONJON est effectivement vouée à un avenir très incertain. Le D2G2Client se veut donc un précurseur de ces fonctionnalités graphiques permettant une approche plus intuitive aux nouveaux utilisateurs, tout en offrant un contrôle rigoureux pour les utilisateurs expérimentés.

Il est important de souligner que le D2G2Client est seulement un prototype et que plusieurs de ses fonctionnalités restent à être développées. En particulier l'un de ses deux modes d'opération, la visualisation des résultats, est tout à fait inopérant. Son développement est par contre en cours de réalisation et pourra éventuellement se greffer au client [35]. Par conséquent, l'implémentation du D2G2Client qui sera présenté couvre seulement l'autre mode d'opération du client : le pré-traitement. Par contre, l'architecture et les décisions de conception au coeur du développement du D2G2Client sont tout aussi valables peu importe le mode d'opération. Le choix des librairies utilisées étant en soi une base solide favorisant l'élaboration et l'intégration du mode de visualisation. C'est pourquoi, il est essentiel de débuter la présentation du client en abordant la librairie sur laquelle reposent toutes les fonctionnalités graphiques.

[11] Traduction directe de l'expression anglaise Graphical User Interface (GUI)

4.3.1 WxWidgets

Tel que soutenu précédemment, l'un des objectifs du projet D2G2 est de favoriser une grande portabilité de son code afin de permettre une utilisation accrue de la chaîne de calcul DRAGON-DONJON quel que soit le système d'exploitation de ces utilisateurs. Or, pour y parvenir dans le contexte de développement du D2G2Client, l'utilisation d'une librairie graphique telle que wxWidgets[12][36] est essentielle.

WxWidgets, anciennement connu sous l'appellation wxWindows, est une bibliothèque C++ multiplateformes offrant une panoplie de fonctionnalités liées au développement des interfaces utilisateurs graphiques. Cette libraire fournit une interface de programmation unique permettant de construire des interfaces dont le code est portable sur la plupart des systèmes d'exploitation. Ainsi, à l'aide de wxWidgets, il est possible de concevoir rapidement une application utilisant des structures graphiques telles que des fenêtres, des menus, des boutons, des icônes, des barres de défilement, des images, et ce, sans se soucier des détails d'implémentation propres à chacun des systèmes d'exploitation. Le code écrit avec wxWidgets se compile donc sur toutes les plateformes supportées par la librairie, sans en changer la moindre ligne.

4.3.1.1 Avantages

Le choix de l'utilisation de cette librairie est vraiment justifié car elle possède plusieurs avantages. Sa documentation est précise et très exhaustive. Elle est en développement depuis 1992; elle est donc très stable et l'une des plus matures actuellement disponible. Elle est de plus portable sur la plupart des systèmes d'exploitation (Win32, MacOS, OS/2, Linux, etc.). Elle offre toutes les fonctionnalités voulues : aide en ligne, programmation réseau, flux de données (streams), presse-papier, glisser-déposer (drag'n drop), multitâche, traitement d'images et surtout le support OpenGL[13]. Fi-

[12]http ://www.wxwidgets.org
[13]http ://www.opengl.org

nalement, le plus important, wxWidgets est totalement gratuit, son code source est libre d'accès et sa communauté de développement est très active.

4.3.1.2 Inconvénients

Parmi les inconvénients de la librairie wxWidgets, mentionnons qu'elle ne permet pas de changer les thèmes d'affichage, qu'elle ne supporte pas la prise en charge des exceptions et qu'elle n'offre pas d'espace de nommage. De plus, sa rapidité d'exécution, quoi déjà l'une des plus rapides de sa catégorie, pourrait être considérablement améliorée si elle utilisait les fonctions de bas niveau de l'interface de programmation système (OS API). En effet, elle repose sur gtk+ et MFC sur les plateformes Linux and Win32 respectivement. Donc, si la librairie reposait par exemple sur Xt, les gains de performances serait assez appréciables sous X11.

4.3.1.3 Comparaison avec QT

Finalement, pour jusfitier davantage le choix de la librairie wxWidgets, il est intéressant de présenter une comparaison avec sa grande rivale et aussi le deuxième choix envisageable pour la conception du D2G2Client : Qt[14]. En effet, Qt est aussi une très bonne librairie multiplatformes qui offre pratiquement les mêmes services que wxWidgets. Cependant, l'une de ces lacunes majeures est son orientation commerciale. En fait, dépendamment du système d'exploitation ou bien de la version utilisés, plusieurs licences peuvent s'appliquer (professionelle-entreprise-communauté). Donc, Qt n'est pas totalement gratuit et il faut être très vigilant avec l'application de ses licences. De plus, contrairement à wxWidgets, les applications créées avec Qt ne sont pas entièrement faites de code C++, mais nécessitent une phase de pré-compilation. Par contre, Qt offre une approche orientée-objets plus avancée se traduisant par une diminution du nombre de lignes de code grâce aux nombreux appels aux fonctions

[14]http ://www.trolltech.com

virtuelles de la librairie. Il en résulte cependant une perte de performances comparativement à wxWidget, qui utilise pour sa part un système de macros. Finalement, il est important de mentionner que contrairement à wxWidgets, Qt offre un excellent support commercial et des outils de développement intégré du type Visual Studio.

4.3.2 Description générale du client

Tel que spéficié dans à la section 4.1, le projet D2G2 possède une architecture client-serveur. Le D2G2Client sert donc à transmettre les requêtes des utilisateurs au serveur. Or, l'explication du transfert des données, du protocole de communication et de la structure de données échangées ayant déjà été abordée dans les sections précédentes, il ne reste plus qu'à présenter l'interface à partir de laquelle l'utilisateur effectue toutes ces opérations.

Le D2G2Client est conçu sur un modèle d'interface à documents multiples. C'est-à-dire, dans une fenêtre principale (fenêtre mère) qui s'affiche dès l'ouverture de l'application via laquelle il est possible d'ouvrir plusieurs sous-fenêtres (fenêtres enfants). En ce qui a trait à la notion de document, celle-ci découle du fait que chaque sous-fenêtre est en charge de l'affichage du contenu d'un document bien précis qui lui est attribué. Ainsi, chaque sous-fenêtre possède les routines nécessaires pour ouvrir, sauvegarder et manipuler les différents éléments ajoutés à la structure de son document associé.

En ce qui concerne le D2G2Client, plusieurs types de documents y sont implémentés. L'ouverture de ceux-ci s'effectuant par l'entremise d'un menu de la fenêtre principale (figure 4.5). Or, dès l'ouverture d'une des sous-fenêtres, une procédure propre à chaque type de document se met en branle. Celle-ci a pour tâche de communiquer avec le serveur pour échanger les données nécessaires à l'élaboration de son contenu. Plus spécifiquement, afin de pouvoir effectuer l'affichage de son interface, chaque fenêtre est en charge du traitement d'un certain contenu qui lui est attribué par l'entremise

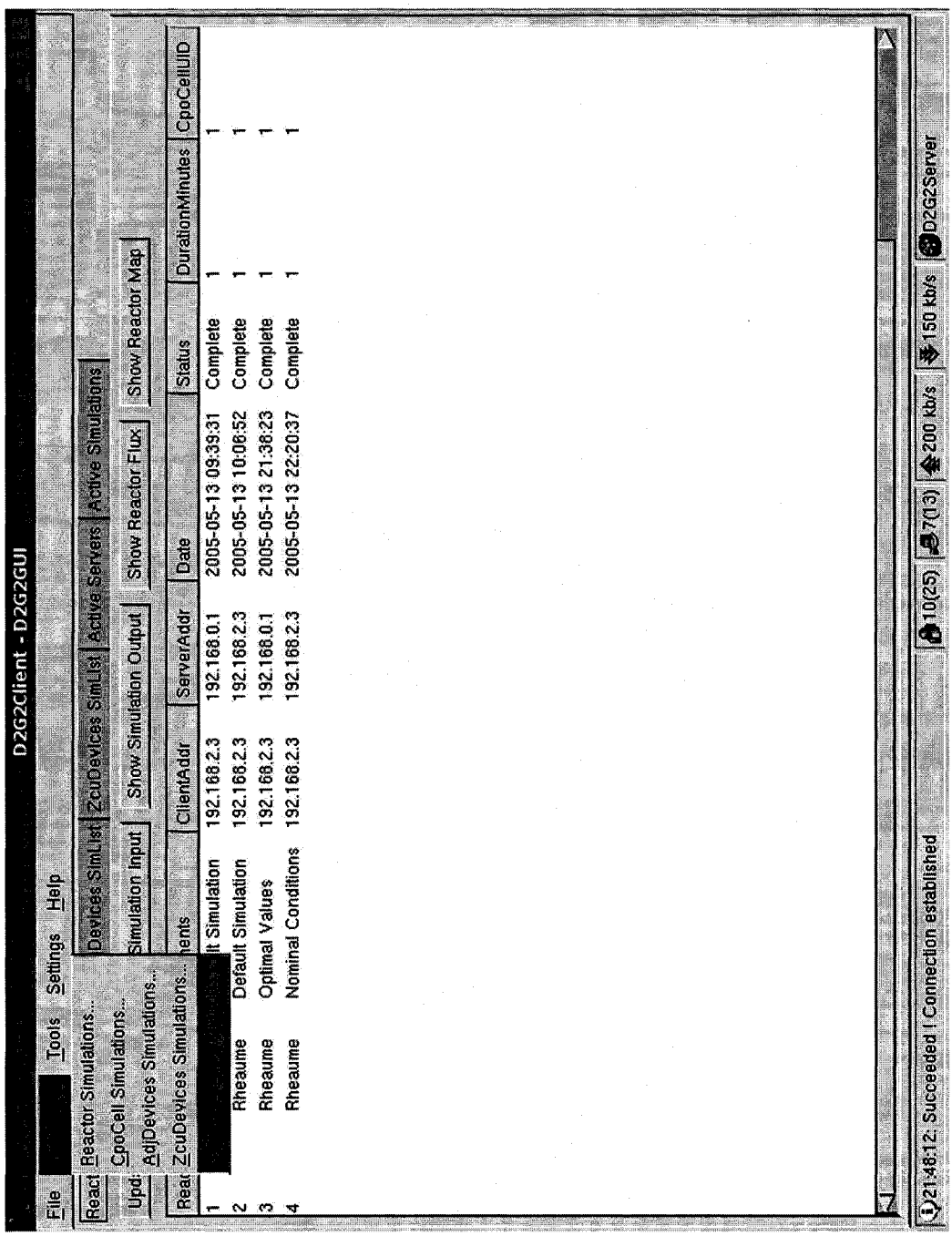

Figure 4.5 Ouverture d'un document via le menu principal

de fichiers XML échangés entre le client et le serveur. En d'autres mots, lorsqu'une des sous-fenêtres du client s'ouvre, un canal de communication entre le client et le serveur est créé, une requête relative au type de la fenêtre en cours d'ouverture est envoyée et finalement un fichier XML contenant les résultats de la requête est retourné par le serveur.

Bref, chaque fenêtre communique par l'entremise d'une requête bien précise parmi celles disponibles sur le serveur. Or, bien que chaque fenêtre soit indépendante, il est possible de les regrouper en deux catégories distinctes : les pré-traitements et les post-traitements.

4.3.3 Mode pré-traitement

Avant d'aborder le mode de pré-traitement, il est bon de faire un rappel sur l'architecture du type tableau noir du projet D2G2. Ce rappel permet de bien comprendre l'importance de la navigation des structures de connaissance par le client afin de concevoir une simulation. En effet, tel que spécifié au début de ce chapitre, l'un des aspects du concept de tableau noir implique que les sources de connaissance puissent naviguer librement le tableau en quête d'informations utiles à la résolution du problème. Ainsi, ayant accès aux travaux des autres sources de connaissances, l'utilisateur peut profiter de l'expertise des autres tout en offrant la sienne. Or, l'accès aux listes des simulations complétées et autres informations de contrôle est justement un exemple concret de cette notion de partage entre les sources de connaissances.

4.3.3.1 Liste des simulations complétées

Parmi les fonctionnalités relatives au mode de pré-traitement, l'affichage de la liste des requêtes qui ont été soumises au serveur est primordial. En effet, la possibilité d'avoir accès à la liste des simulations qui ont déjà été effectuées est non seulement nécessaire pour sélectionner les données que l'on veut éventuellement analyser, mais

offre aussi un autre avantage non négligeable : la possibilité d'avoir accès à la liste de simulations effectuées, quel que soit l'usager qui en a fait la requête et quel que soit le serveur qui les a exécutés. L'utilisation des résultats de simulation d'un autre usager pour élaborer une nouvelle simulation est effectivement très utile pour éliminer le dédoublement des efforts de calcul ou pour servir de modèle.

C'est notamment le cas de plusieurs simulations qui dépendent de sous-calculs. Il est alors nécessaire d'y avoir accès pour réaliser une nouvelle simulation. Plus la liste des simulations complétées est exhaustive, plus il y a de chances de réaliser des simulations complémentaires innovatrices. Par exemple, tel que présenté dans le premier chapitre, pour effectuer un calcul de réacteur, il faut fournir les résultats d'un calcul de cellule, d'un calcul des sections efficaces des barres de contrôle et de celles des contrôleurs liquides. Il est donc primordial de pouvoir efficacement afficher la liste des simulations CpoCell, AdjDevices et ZcuDevices déjà effectuées.

Maintenant, pour revenir aux possibilités du D2G2Client, celui-ci permet entre autres l'affichage de la liste des simulations effectuées sur les barres de contrôle (figure 4.6), les contrôleurs liquides (figure 4.7), les cellules unitaires (figure 4.8), les macrolibs et les structures pour les calculs de supercellule (figures 4.9) et les simulations de réacteur (figure 4.10). Chaque liste possédant une série d'attributs : un numéro d'identification unique (ID), le nom de l'usager qui a soumis la requête, les commentaires relatifs aux paramètres de la simulation, l'adresse IP du D2G2Client à partir duquel a été transmise la requête, l'adresse IP du D2G2Server qui a exécuté la simulation, la date de soumission, le temps de calcul en minutes, le statut final de la requête et en dernier lieu les numéros d'identification unique (s'il y a lieu) des simulations dont dépend la simulation affichée. Par exemple, la figure 4.6 montre une liste de deux entrées de simulation de barres de contrôle. Prenant, la première entrée, il y est affichée le numéro ID (1), le nom de l'usager (Rheaume), l'adresse du D2G2Client à partir duquel Rheaume à soumis sa requête (192.168.2.3), l'adresse du serveur sur

lequel la simulation a été effectuée (192.168.0.1), la date de soumission (2005-05-07 23 :48 :06), le statut (Complete), la durée (765 minutes) et pour terminer le numéro ID de la simulation déterminant la macrolib et la structure de supercellule de laquelle dépend ce calcul de barres de contrôle (1).

4.3.3.2 Simulations et serveurs actifs

Une autre liste de simulations très utiles est celle des simulations actives. En effet, il est très commode de pouvoir suivre l'évolution des requêtes en cours d'exécution sur le D2G2Server. La liste des simulations actives (figure 4.11) fournit les renseignements suivants : le nom du fichier temporaire en cours d'exécution, l'adresse IP du serveur qui exécute le calcul, le nom de l'usager qui a soumis la requête, des commentaires relatifs à la simulation en cours, la date à laquelle elle a été soumise et finalement son statut. Or, l'attribut de cette liste qui peut être considéré comme le plus utile est bien entendu le statut de la simulation. Il est effectivement primordial de savoir si la simulation est en train de s'exécuter ou bien si elle s'est arrêtée pour une raison ou une autre.

Finalement, la dernière liste d'informations présentée favorisant un bon suivi de la puissance de calcul disponible, est la liste des D2G2Server actifs (figure 4.12). Il est effectivement intéressant de pouvoir connaître le nombre de serveurs actifs et le nombre de simulations que chacun d'eux est en train d'exécuter. Par exemple, sachant qu'il y a seulement deux serveurs actifs et qu'ils ont déjà plusieurs simulations actives, il est évident que la durée d'exécution des prochaines simulations soumises sera allongée.

Par contre, il est important de rappeler qu'aucune parallélisation de DRAGON ou de DONJON n'a été effectuée dans le cadre de cet ouvrage. Ainsi, même s'il y a plusieurs serveurs actifs, la durée d'un calcul ne sera pas influencée. Seul le nombre

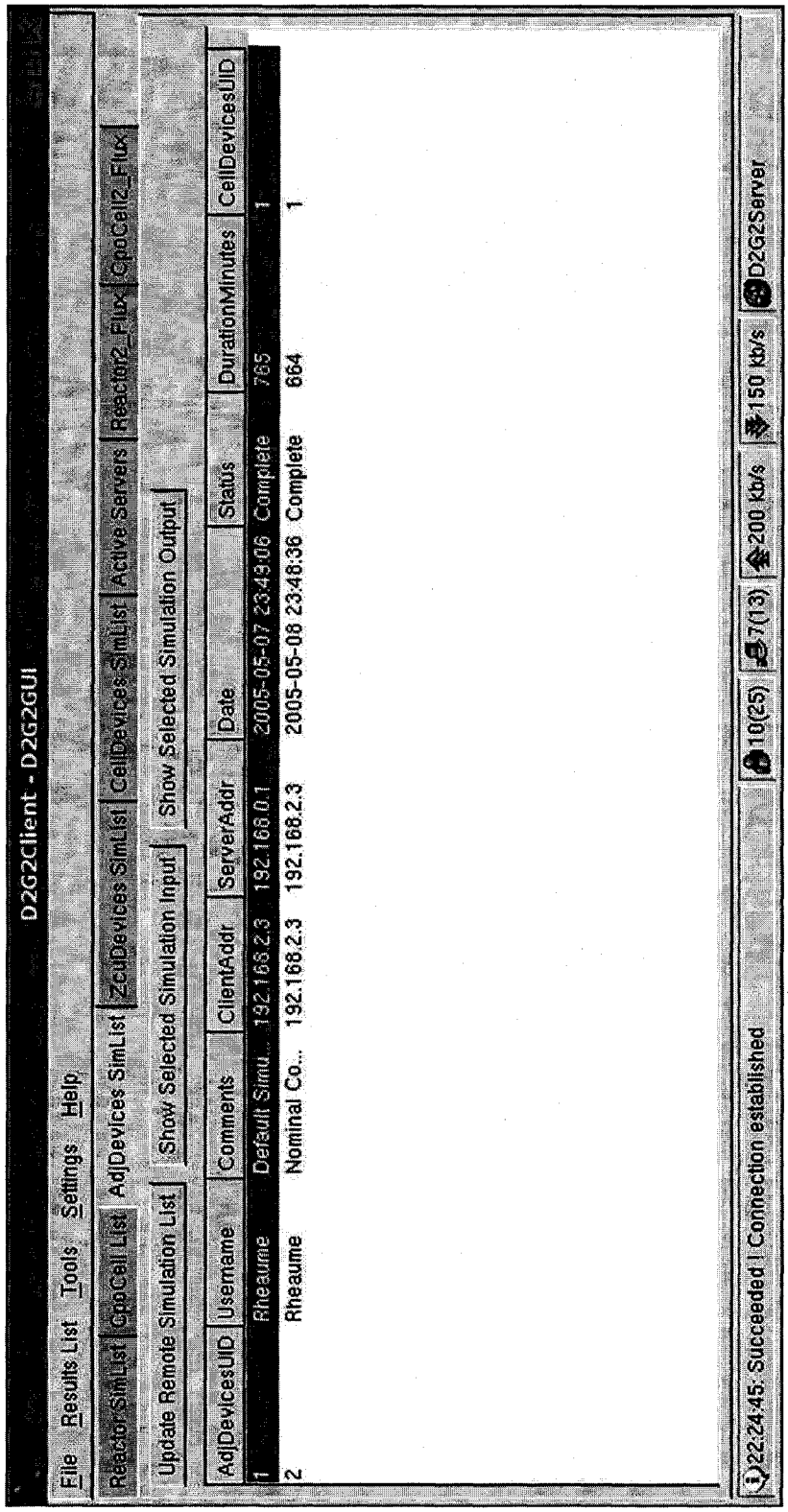

Figure 4.6 Liste des simulations de barres de contrôle

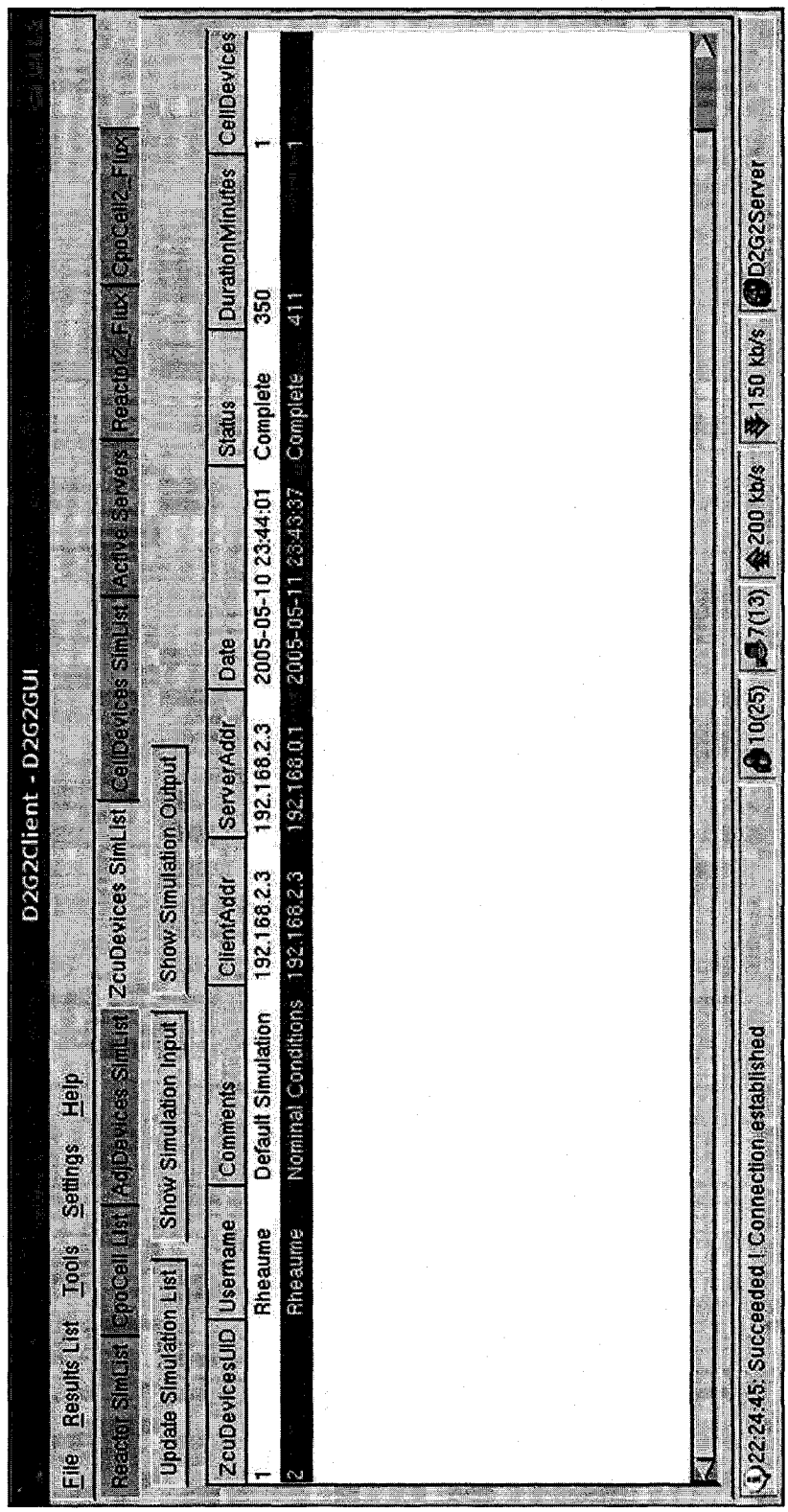

Figure 4.7 Liste des simulations de contrôleurs liquides

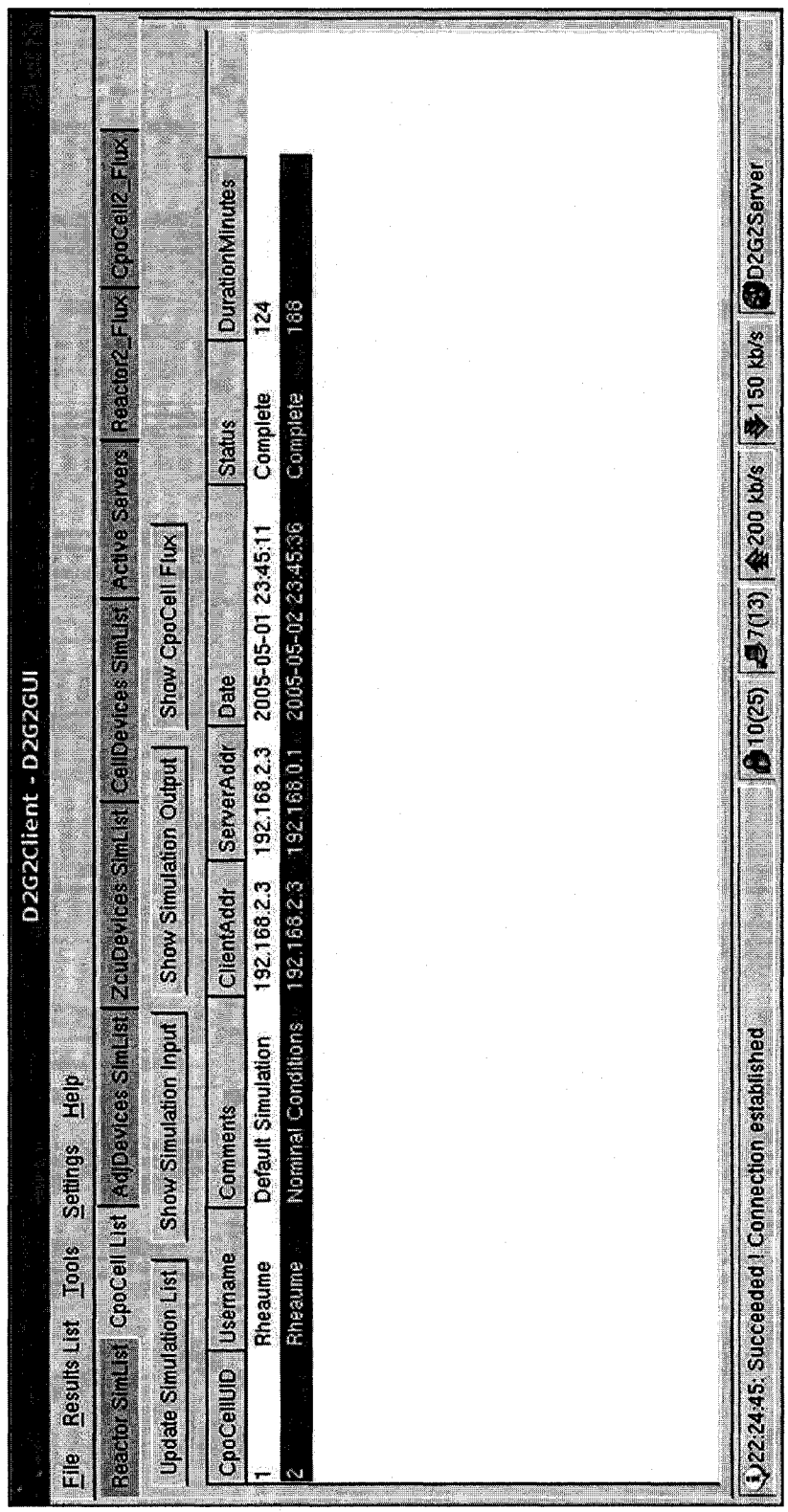

Figure 4.8 Liste des simulations de cellule unitaire

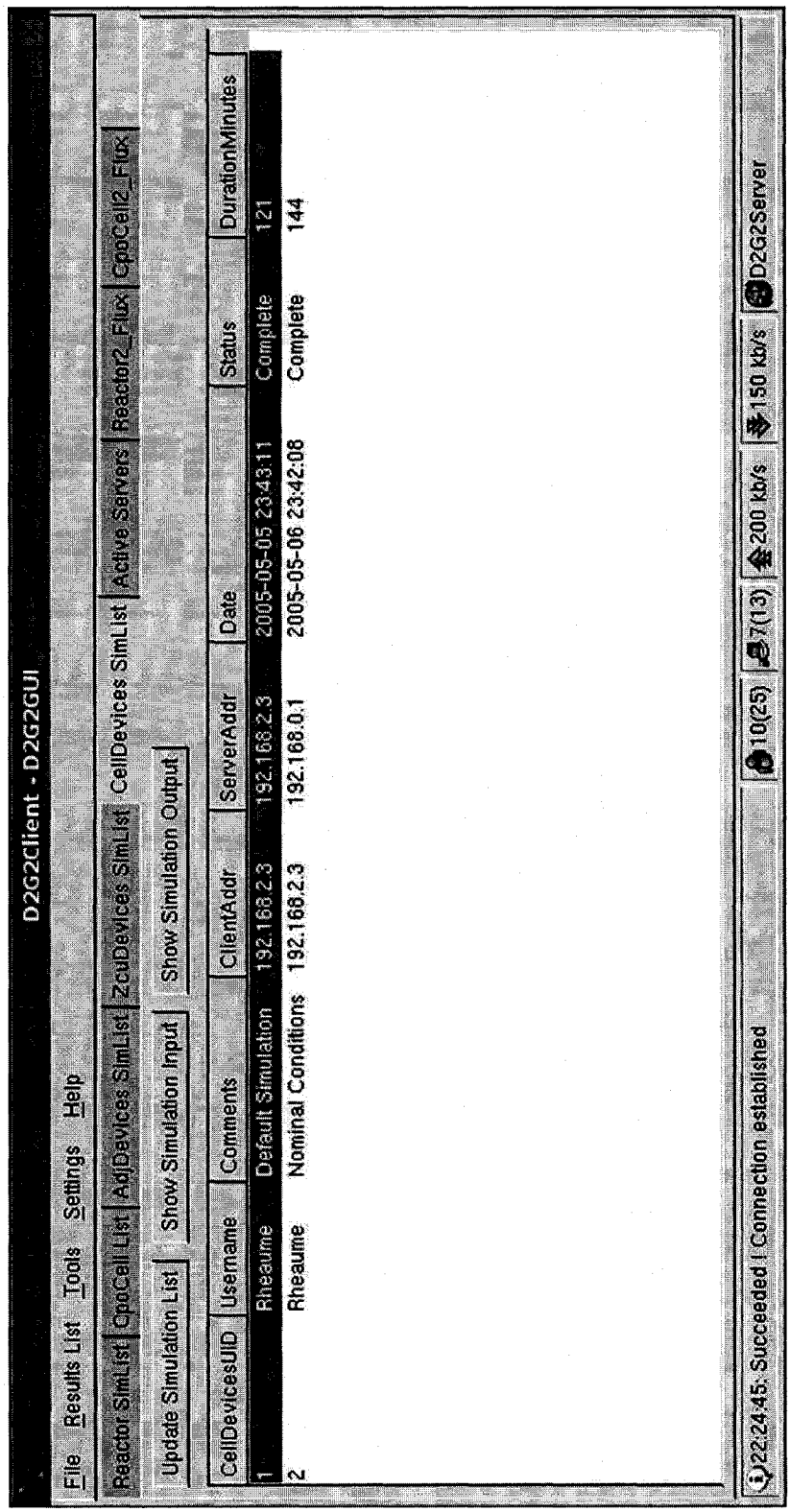

Figure 4.9 Liste des simulations macrolib et structure supercellule

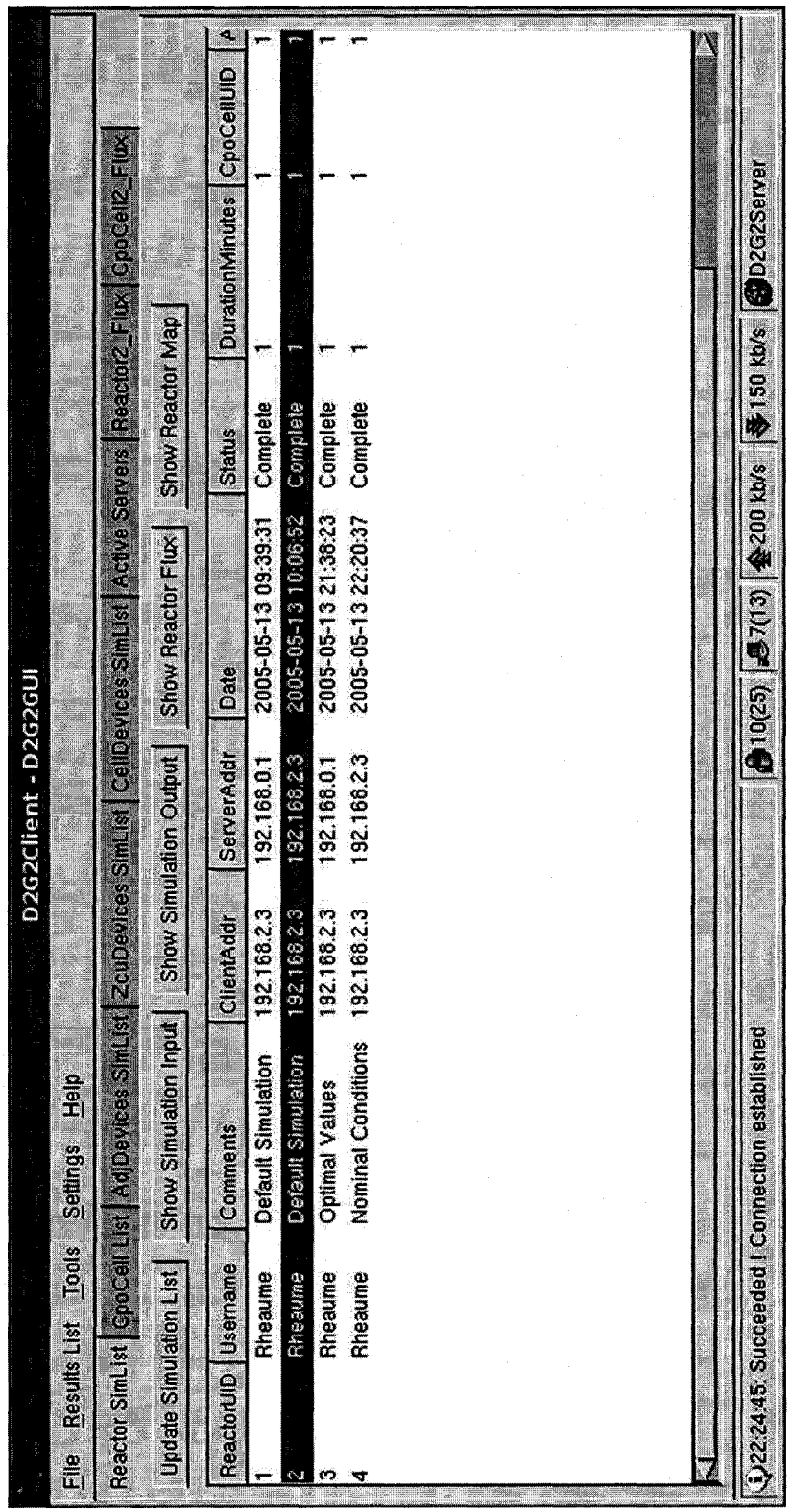

Figure 4.10 Liste des simulations de réacteur

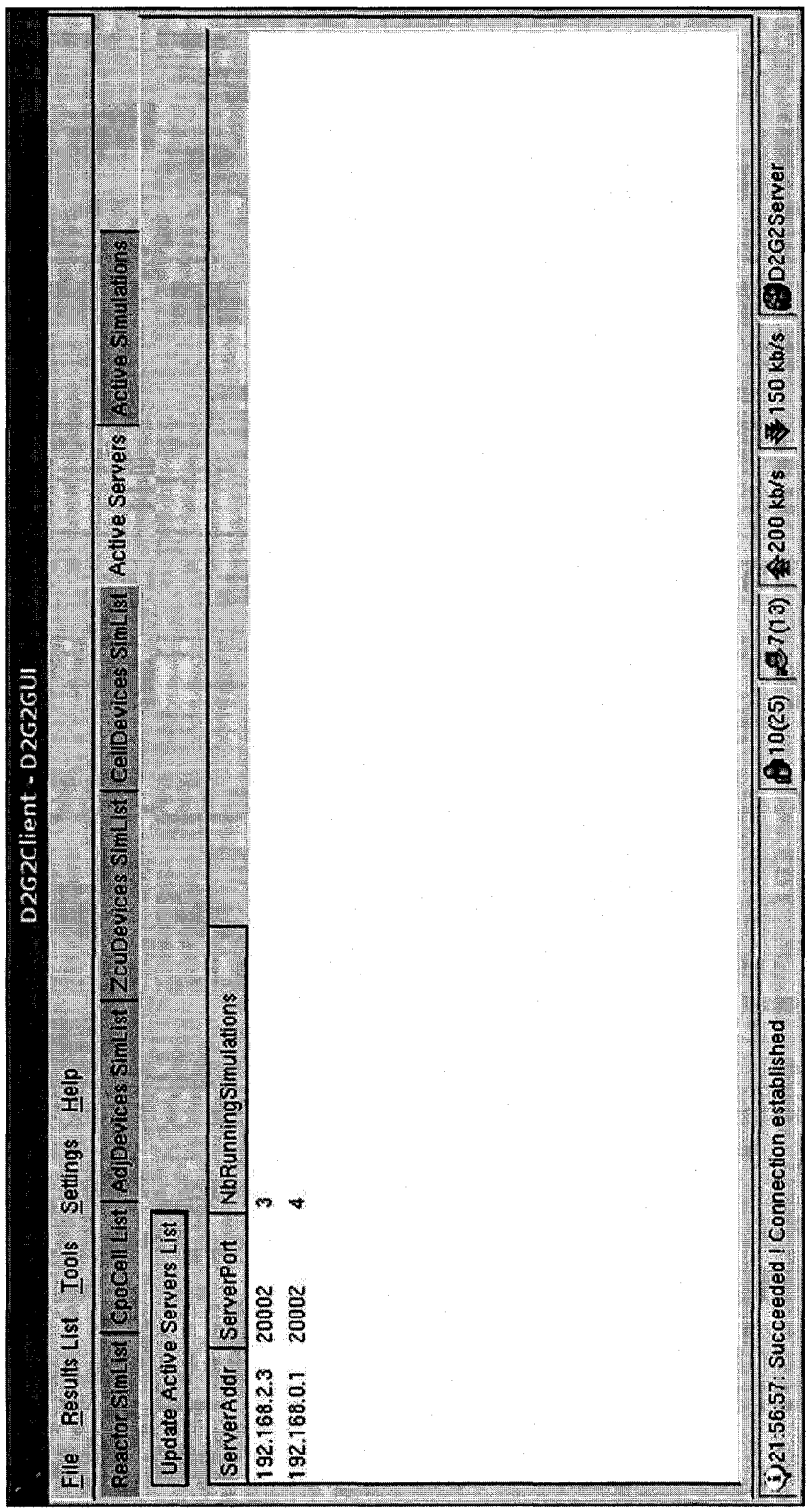

Figure 4.11 Liste des serveurs actifs

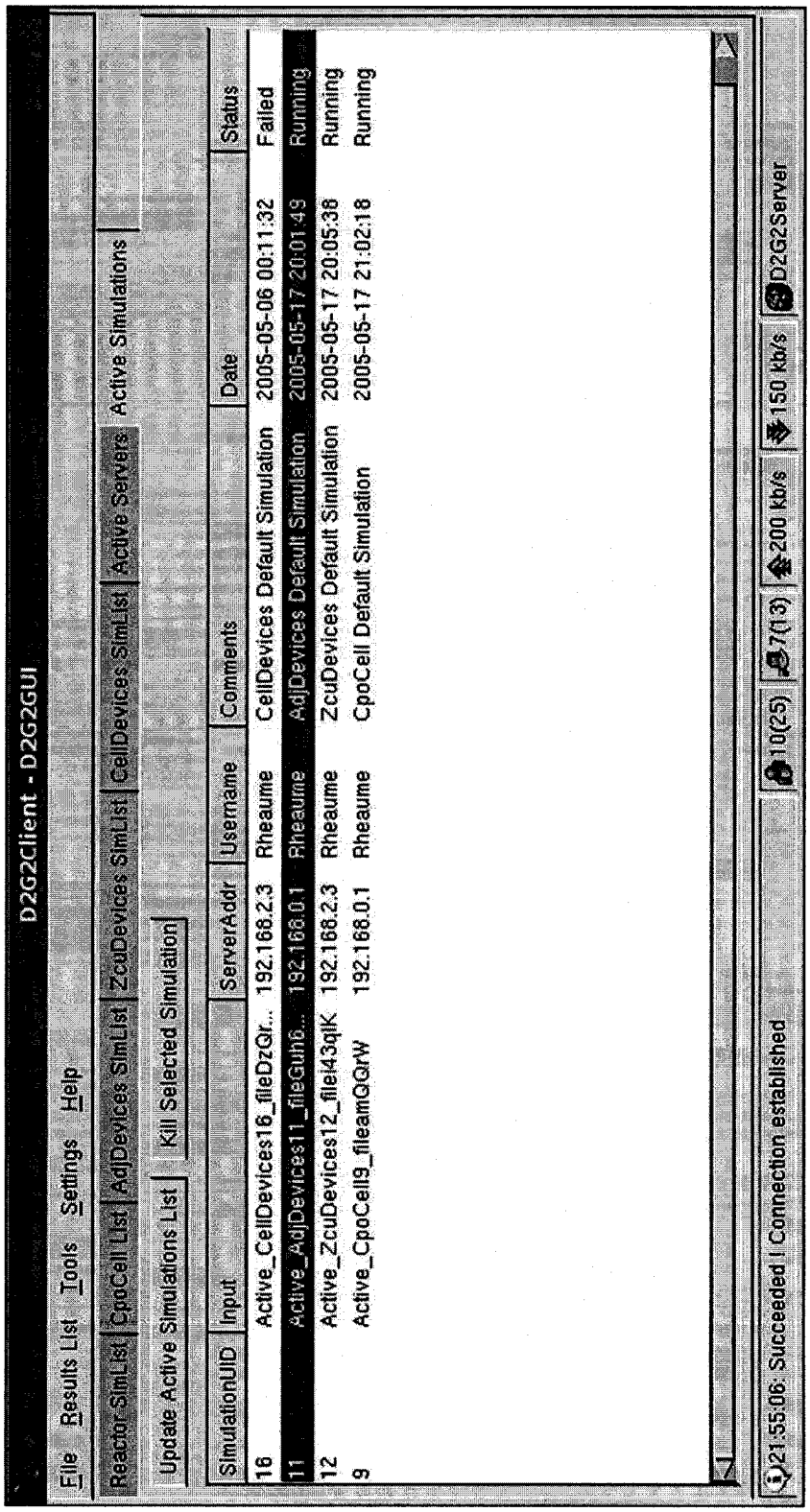

Figure 4.12 Liste des simulations actives

de simulations en cours d'exécution sur un même serveur influence les performances moyennes de ce serveur. Cependant, il est clair que s'il y a plusieurs serveurs actifs, le nombre de simulations actives par serveur diminue. Ceci augmente par le fait même les performances de la grille de calcul D2G2Server, étant donné l'équipartition avec laquelle les requêtes sont partagées sur les différents serveurs actifs.

4.3.3.3 Requête de simulation

Une fois les mécanismes d'interrogation de l'état des simulations explicités, il est maintenant temps d'aborder la raison d'être de l'implémentation de ces services : la requête de simulation. Cette phase est l'étape la plus importante de tout le processus client. La moindre erreur lors de la construction de la requête entraîne des résultats erronés, même si les calculs sont effectués par un code neutronique validé et stable.

C'est justement l'un des principaux objectifs d'optimisation du projet D2G2 de minimiser autant se faire que peut la probabilité de soumettre une fausse requête à DRAGON ou DONJON. Pour y parvenir, il faut donc encadrer au maximum les utilisateurs afin qu'ils ne puissent en aucun temps commettre des erreurs banales lors l'élaboration de leurs fichiers d'entrées. Par exemple, une anecdote du passé mettant en cause une simple erreur typographique sur la densité du chrome a requis d'énormes efforts de débogage. Or, il est clair que la confection manuelle des fichiers d'entrées GANlib doit tendre à disparaître. En ce sens, le D2G2Client propose une approche encadrant constamment les entrées des utilisateurs en leur fournissant une description détaillée des variables de simulation et en validant systématiquement toutes leurs entrées dans des plages opérationnelles d'exploitation normale ou de conception réaliste. La figures 4.13 présente un exemple d'interface où l'utilisateur est dirigé dans la construction de sa requête. Cette interface se divise en trois parties. Celle de gauche constitue la barre de navigation permettant à l'utilisateur de sélectionner les modules d'entrées qu'il désire paramétrer. La partie supérieure droite affiche quant à elle les

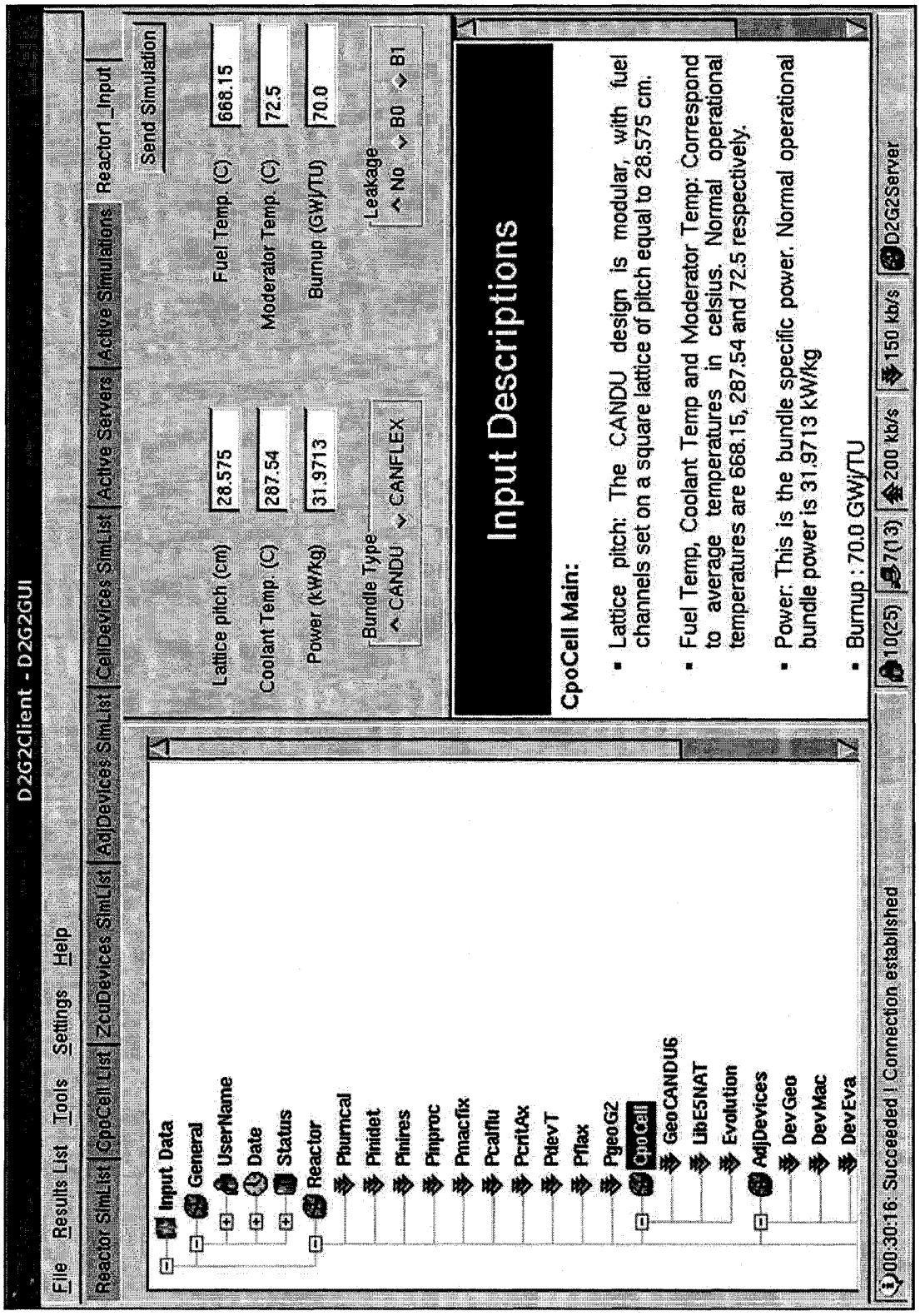

Figure 4.13 Saisie des paramètres d'une requête de simulation

variables modifiables du module sélectionné. C'est en remplissant cette section que l'utilisateur est informé de son erreur ou bien de la plage des valeurs possibles, avant même d'avoir soumis sa requête au serveur. En effet, c'est lors de la modification de l'une des variables de cette portion de l'interface que des routines de validation testent les entrées du client. Et finalement, la partie inférieure droite affiche pour sa part des informations relatives à chacune des variables paramétrées : descriptions, unités, plage de valeurs admises. Bref, l'utilisateur est constamment informé de la validité de ces entrées, tout en étant renseigné sur la nature de celles-ci.

À noter aussi que même si les fonctionnalités de construction des requêtes de simulation ne sont guères très évoluées et pas entièrement implémentées, il est toujours possible pour l'instant de soumettre une requête au D2G2Server sous son format natif en transférant directement un fichier XML (figure 4.14). Ainsi, même si l'utilisateur ne profite pas des fonctionnalités de l'interface graphique et des renseignements sur les variables de simulation, il n'en demeure pas moins que la validation découlant de l'utilisation de DTD associée au fichier XML est toujours accomplie. Ce qui demeure quand même un avantage considérable comparativement à l'édition manuelle des fichiers GANlib.

En terminant, tel que mentionné à maintes reprises précédemment, le D2G2Client élaboré à ce jour n'est qu'un simple prototype. Par contre, certains développements en cours [35] touchant certains aspects de la visualisation permettront d'améliorer considérablement l'interface de conception des requêtes. Par exemple, une navigation tridimentionnelle à l'intérieur du réacteur où il y est possible de sélectionner les constituants à paramétrer ou bien d'interroger leur valeur pourrait bientôt être envisageable. L'intégration de ces travaux dans le D2G2Client jouera alors un rôle déterminant dans l'évolution de celui-ci.

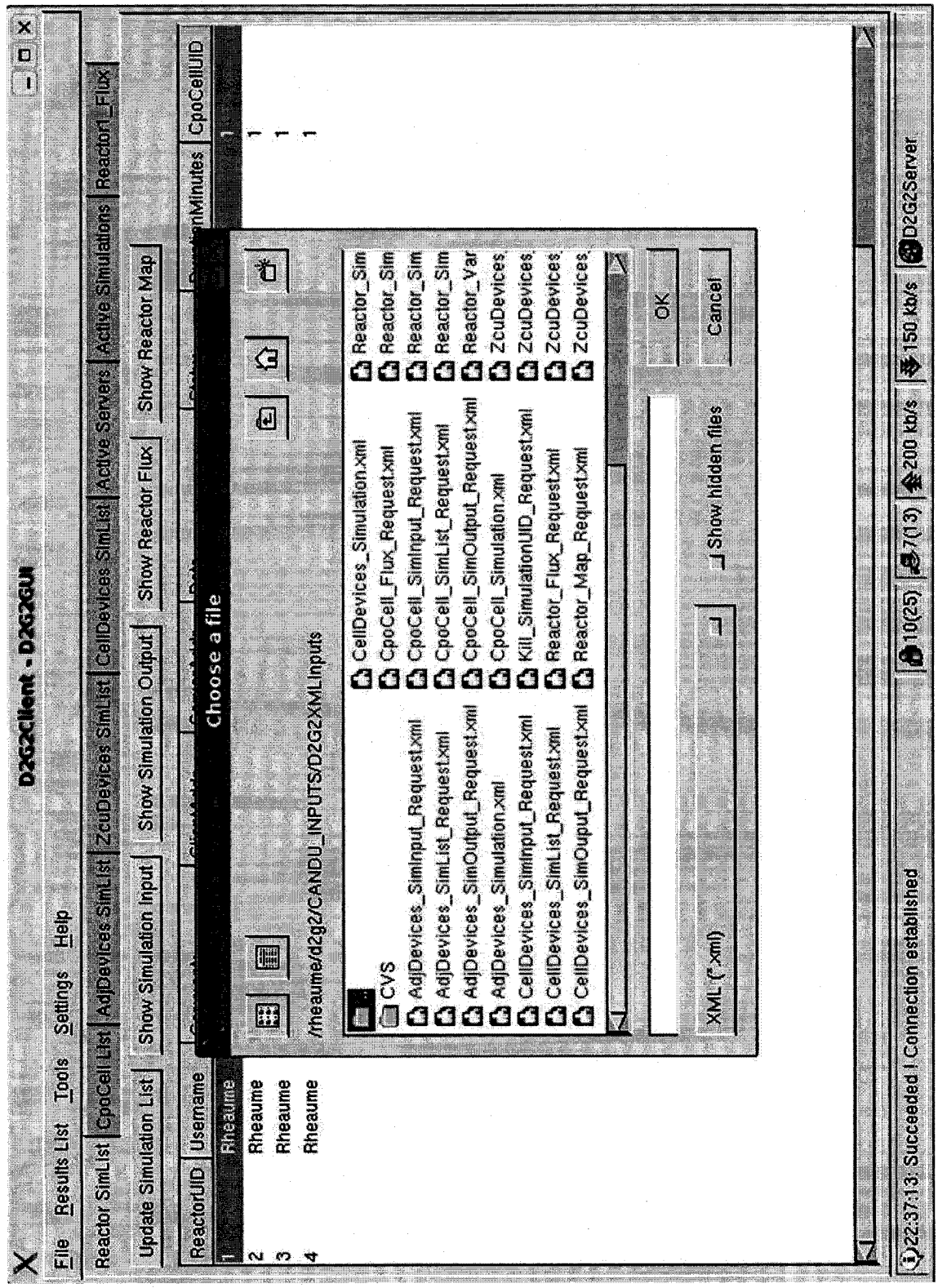

Figure 4.14 Envoi d'un fichier XML

4.3.4 Mode post-traitement

Comme la plupart des chaînes de calcul neutronique, la suite DRAGON-DONJON n'offre aucune interface utilisateur telle que présentée à la section précédente. Elle n'offre de plus aucun support avancé pour la visualisation des résultats. En effet, il faut extraire manuellement les résultats (flux, taux de réactions, épuisement du combustible) des fichiers de sortie et les importer dans un logiciel spécialisé (Mathematica, TecPlot). Donc, de nombreuses étapes transitoires sont requises entre la fin de la simulation et la visualisation des résultats. Le D2G2Client veut donc pallier cet inconvénient en intégrant la visualisation dans son interface.

Par contre, ce mode de post-traitement n'a pas encore été implémenté dans le cadre de ce travail. Cependant, l'élaboration de l'architecture du client a été soigneusement prise en considération pour permettre son ajout éventuel. C'est dire que le client a été conçu afin de rendre l'intégration de ces outils de visualisation relativement simple. En effet, étant donné que la librairie wxWidgets possède le support OpenGL, un canevas graphique a été mis en place parmi les types de documents disponibles du client. Le D2G2Client étant par conséquent déjà apte à coexister avec des outils de visualisation développés à partir d'OpenGL. Par exemple, la figure 4.15 présente un cas test où une simple représentation du flux thermique d'un réacteur complet est affiché sur le canevas GL de l'interface du D2G2Client. La figure 4.16 présente quant à elle un autre cas test avec l'affichage du flux thermique dans une grappe à 37 éléments. Or, il est clair que ces exemples statiques sont très rudimentaires et qu'aucune navigation tridimensionnelle n'est permise, contrairement aux fonctionnalités qui seront éventuellement disponibles lors de l'intégration des outils de visualisations développés en parallèle avec le projet D2G2.

Bref, pour l'instant, la visualisation des résultats est vraiment limitée. Elle constitue un simple affichage du fichier de la sortie standard récupérée lors de l'exécution d'une

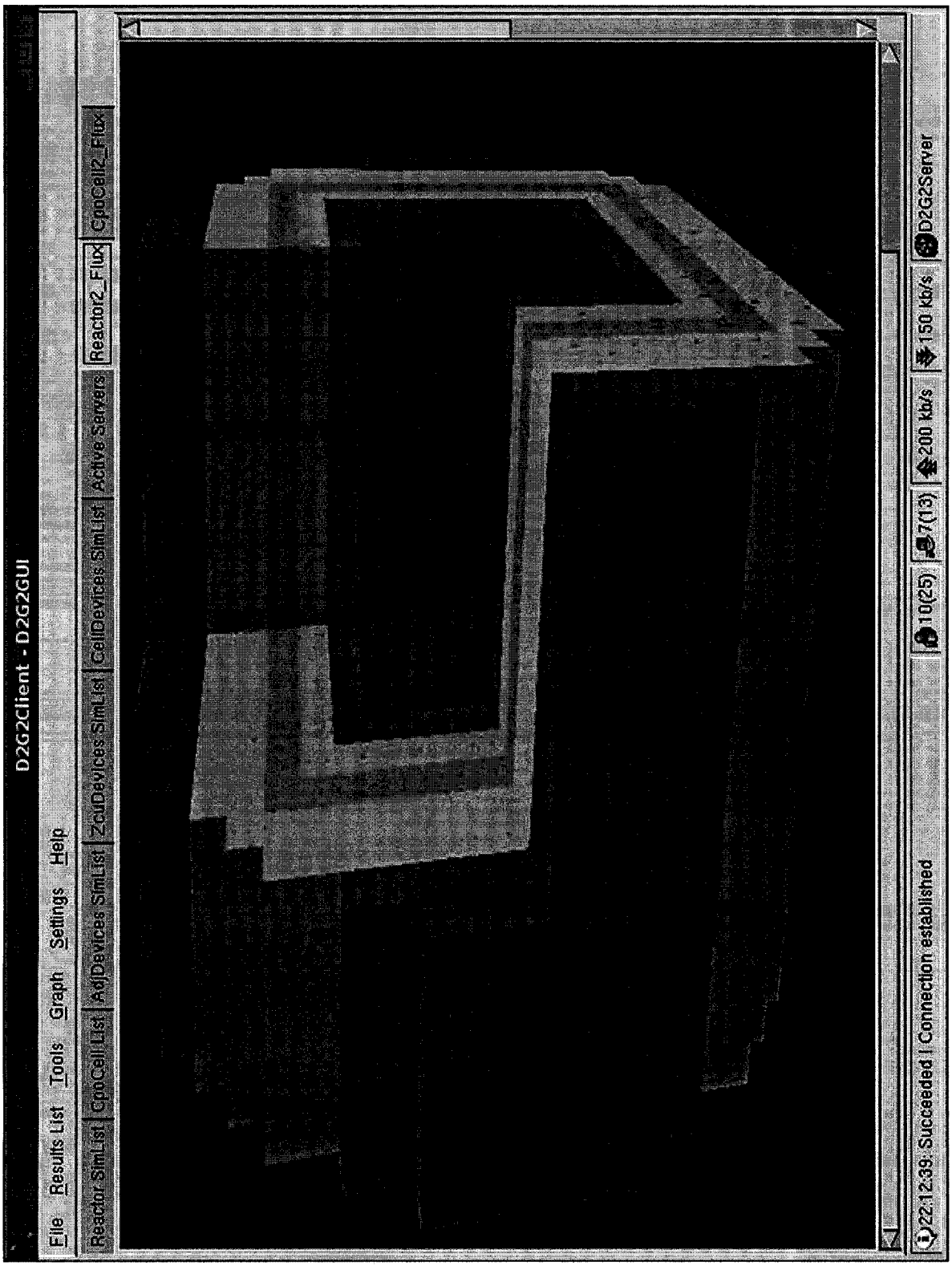

Figure 4.15 Affichage du flux d'un calcul de réacteur

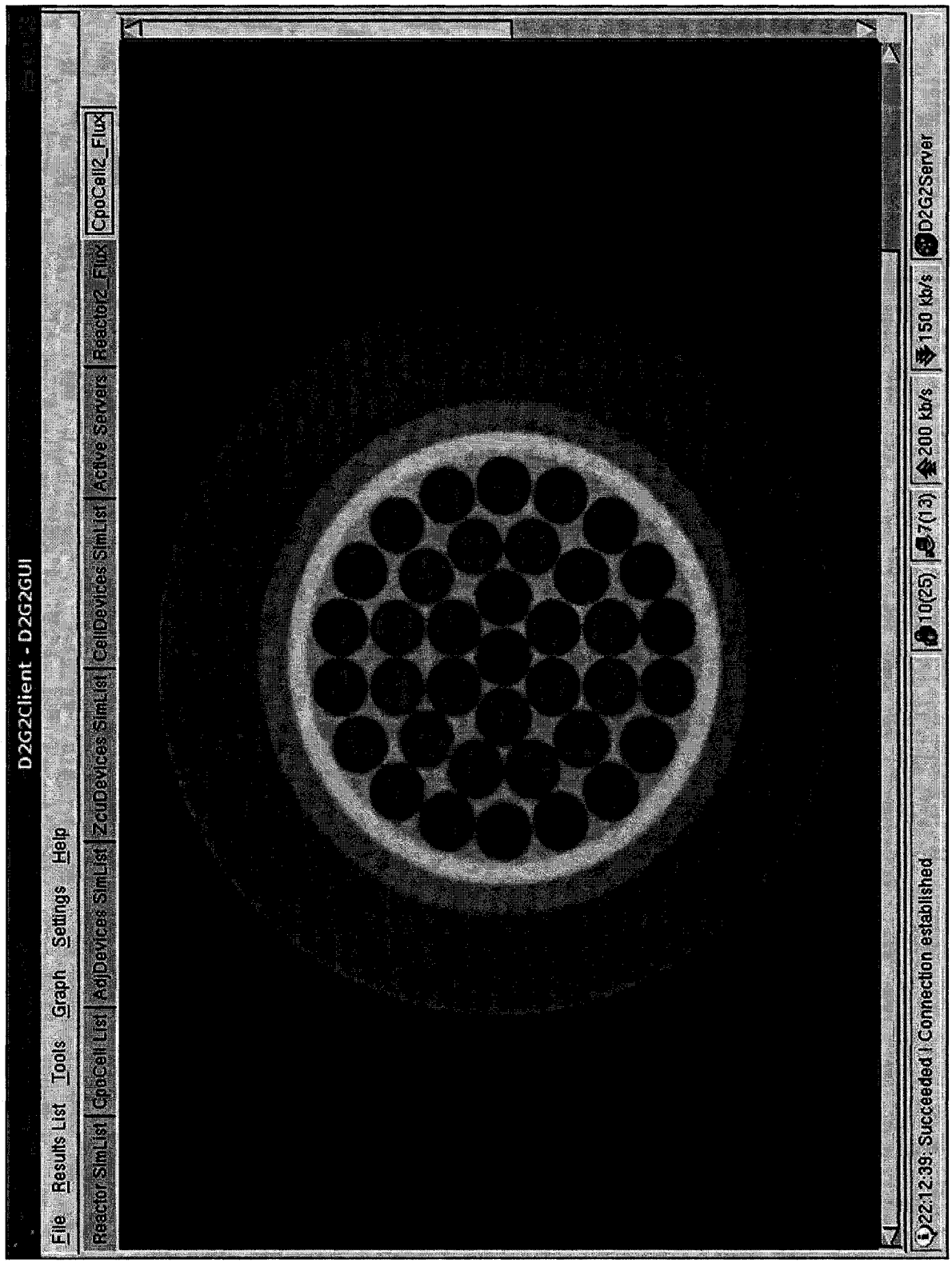

Figure 4.16 Affichage du flux d'un calcul de cellule

simulation DRAGON ou DONJON (voir figure 4.17). Grâce à ce fichier texte, il est du moins possible de visualiser certains résultats intermédiaires et l'essentiel, de voir si la simulation a été terminée avec succès. Bien entendu, il est aussi possible d'avoir accès aux fichiers de flux, mais aucune visualisation graphique de ceux-ci n'est implémentée. Seulement les fichiers sous format texte (figure 4.18) sont présentement disponibles.

4.3.5 Paramètres de configuration

Il est aussi pertinent de spécifier que le D2G2Client possède une interface de configuration (figure 4.19). En effet, il est important de pouvoir spécifier l'adresse IP ou le nom du serveur sur lequel il faut se connecter, ainsi que le port TCP à utiliser. De plus, l'interface permet aussi de s'identifier et d'inscrire son mot de passe. Cette étape est cruciale, car sans ces informations il est impossible de communiquer avec le serveur puisqu'il effectue une routine d'authentification à chacune de ces communications avec le client.

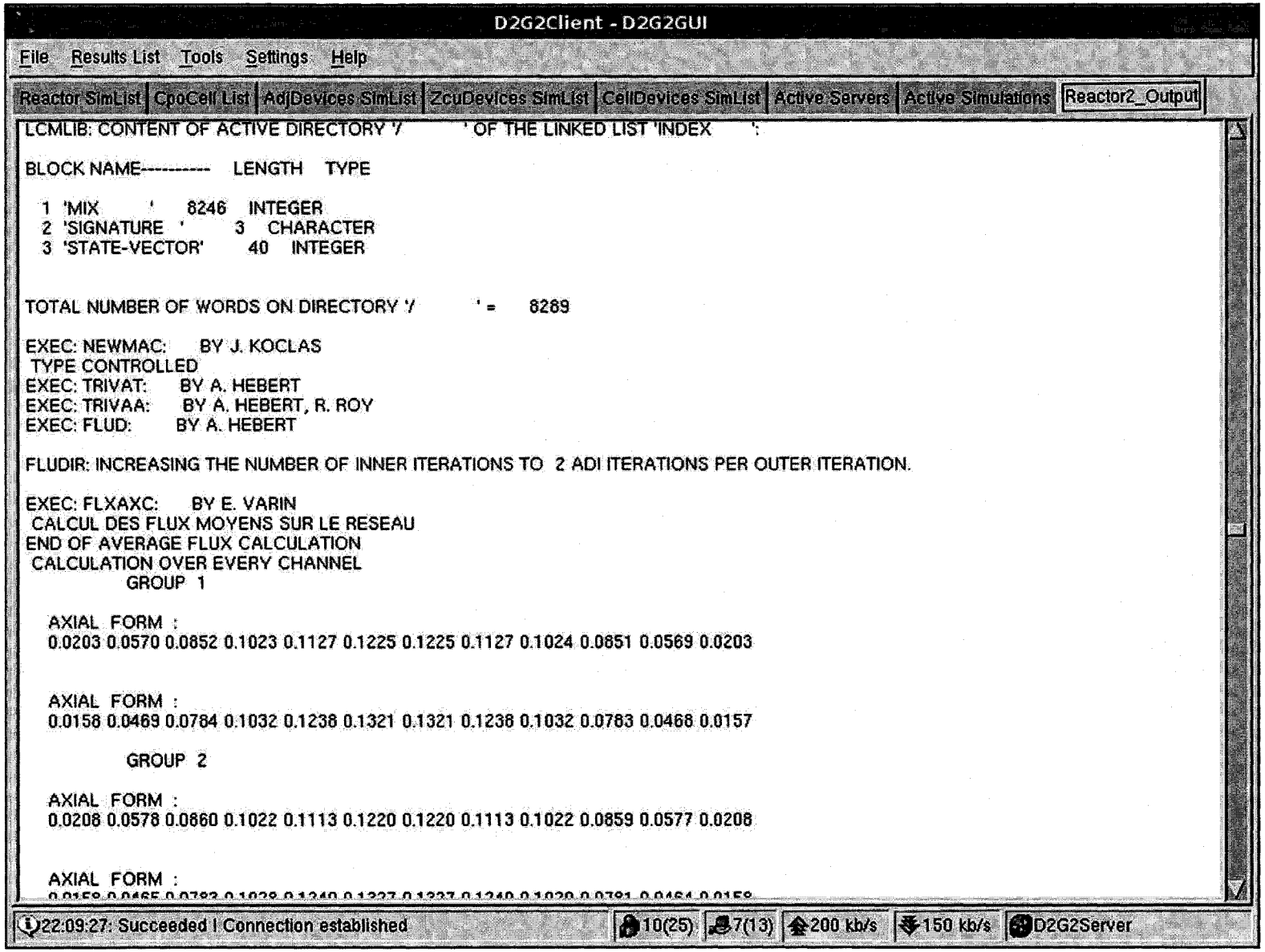

Figure 4.17 Sortie standard d'une simulation réacteur

Figure 4.18 Affichage du flux d'un réacteur (format texte)

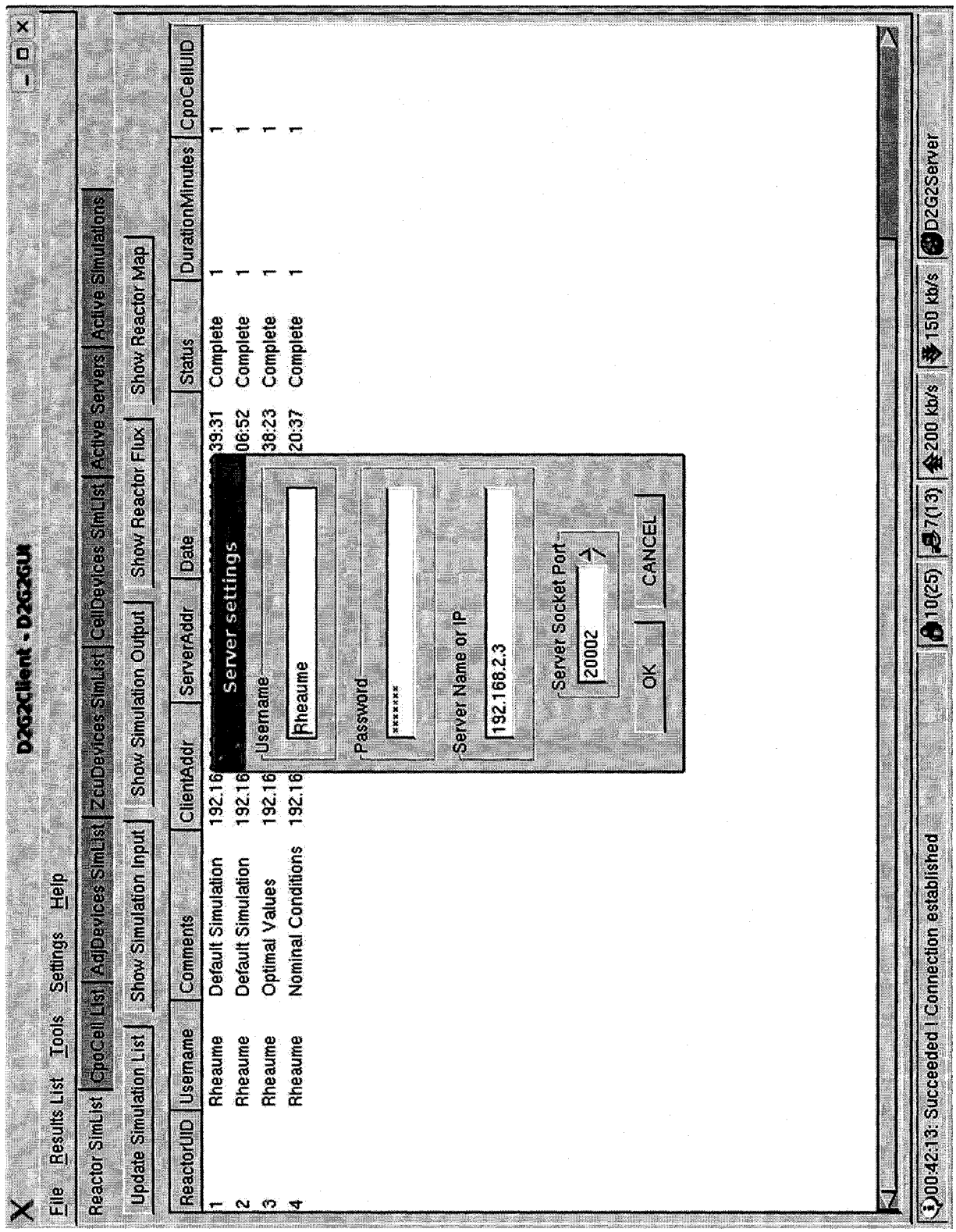

Figure 4.19 Interface de configuration du client

CHAPITRE 5

VALIDATION LOGICIELLE

La validation logicielle du projet D2G2 doit permettre de tester la plupart des fonctionnalités développées. Elle doit recourir à des requêtes de simulation DRAGON et DONJON, vérifier l'état des simulations et finalement recueillir les résultats. Cette validation doit de plus s'assurer que les résultats obtenus concordent bel et bien avec ceux provenant des mêmes calculs effectués à l'aide des méthodes usuelles d'utilisation de la chaîne de calcul.

Cela dit, pour valider les outils D2G2, la problématique choisie doit inclure des calculs de transport et de diffusion. Elle doit de plus avoir été, par le passé, complétée à maintes reprises avec succès. Par conséquent, parmi les problèmes permettant de vérifier le bon fonctionnement du D2G2Server et du D2G2Client, celui déterminant le taux de combustion moyen des grappes de combustible à la sortie du réacteur, lorsque celui-ci est à l'équilibre du rechargement, est tout à fait adéquat.

5.1 Contexte neutronique des tests d'intégration

5.1.1 Réacteur à l'équilibre du rechargement

Un réacteur neuf (toutes ces grappes de combustible sont neuves[1]) peut fonctionner normalement pendant environ 6 mois à pleine puissance en raison de sa réactivité excédentaire initiale. Cependant, après quelques mois en marche, l'évolution du com-

[1]En pratique quelques grappes appauvries en U-235 sont disposées au centre du réacteur

bustible et l'accumulation des produits de fissions dans le combustible irradié doivent être compensées. Comme le CANDU se distingue des autres réacteurs nucléaires par la présence d'une séparation physique entre le caloporteur et le modérateur (tubes de force), il peut être rechargé en marche. Ainsi, même à pleine puissance, l'addition quotidienne de grappes neuves et le retrait des plus irradiées sont possibles. Une stratégie de rechargement est cependant nécessaire afin qu'un suivi judicieux soit constamment effectué pour éviter toute distorsion du flux. Le choix des emplacements à recharger et le nombre de grappes à échanger doivent effectivement être planifiés avec soin. L'objectif étant de conserver un coeur critique à pleine puissance malgré l'évolution du combustible.

Toutefois, après plusieurs mois de rechargement, la fréquence de recharge des grappes devient constante. Le taux de rechargement du réacteur est donc à l'équilibre du rechargement et le degré d'irradiation des grappes est uniformément distribué entre 0 et le taux de combustion de sortie. Le réacteur possède alors la distribution de puissance nominale découlant des paramètres de design. C'est justement cet état nominal qui sert de référence pour les analyses de sûreté, l'optimisation de la conception du coeur ou bien la gestion du combustible.

5.1.2 Taux de combustion de sortie moyen

La distribution de puissance nominale et le concept du réacteur à l'équilibre du rechargement permettent plusieurs approximations menant à des schémas simplifiés de gestion du combustible. Grâce à ces notions, il est effectivement possible d'effectuer des calculs de diffusion statique décrivant l'état du réacteur moyenné dans le temps afin de déterminer plusieurs paramètres opérationnels. Par exemple, le taux de combustion moyen des grappes à leur sortie du réacteur peut être trouvé [37]. La méthode pour l'obtenir est la suivante. Il faut tout d'abord définir quelques paramètres :

- ϕ_{jk} : flux thermique moyenné dans le temps (cellule unitaire de la grappe k dans le canal j)
- B_j : taux de combustion moyen de sortie pour le canal j
- n_j : nombre de grappes neuves introduites dans le canal j à chaque rechargement
- T_j : intervalle de temps entre deux rechargements dans le canal j
- $H_{jk} = \kappa\Sigma_{f,jk}$: énergie produite par fission multipliée par la section efficace de fission (moyenne sur le cycle)
- $p_{jk} = H_{jk}\phi_{jk}$: distribution de puissance de grappe moyennée dans le temps

L'incrément du taux de combustion pour la grappe k sur un cycle défini par l'intervalle entre deux rechargements est alors donné par :

$$\Delta B_{jk} = B_{T,jk} - B_{0,jk} = H_{jk}\phi_{jk}T_j = p_{jk}T_j \tag{5.1}$$

Ainsi, à l'équilibre de rechargement, le taux de combustion moyen de sortie des grappes retirées du canal j est l'énergie produite dans le canal durant le cycle, divisé par n_j :

$$B_j = \frac{T_j}{n_j}\sum_k H_{jk}\phi_{jk} = \frac{1}{n_j}\sum_k \Delta B_{jk} \rightarrow \Delta B_{jk} = n_j B_j \Psi_{jk} \tag{5.2}$$

où

$$\Psi_{jk} = \frac{H_{jk}\phi_{jk}}{\sum_k H_{jk}\phi_{jk}} \tag{5.3}$$

avec Ψ_{jk} qui est le rapport de la puissance de la grappe sur celle du canal (facteur de forme axiale). Finalement, à partir des sections efficaces de la cellule contenant la grappe k du canal j :

$$\Sigma_{jk}(t) = \Sigma^*[B_{jk}(t)] + \Delta\Sigma_{jk}(t) \tag{5.4}$$

où Σ_{jk} correspond à l'incrément de section efficace prenant en considération la présence ou non d'un mécanisme de régulation. Il est alors possible de déterminer les sections efficaces moyennes pour un taux de combustion spécifique :

$$\Sigma_{jk} = \frac{1}{n_j B_j \Psi_{jk}} \int_{B_{0,jk}}^{B_{T,jk}} \Sigma^*(\theta) d\theta + \Delta\Sigma_{jk} \tag{5.5}$$

où $\Delta\Sigma_{jk}$est considéré constant étant donné que le réacteur est toujours à l'état nominal. Bref, une fois les sections efficaces moyennes déterminées, il est possible de résoudre l'équation de diffusion et de vérifier si les différentes contraintes d'opération du coeur sont respectées (criticité du coeur, puissance maximale de grappe et de canal en deçà des limites opérationnelles). Si c'est le cas, le taux de combustion moyen de sortie du combustible est alors déterminé. Or, la résolution n'est cependant pas si simple, car les limites d'intégration de l'équation 5.5 dépendent implicitement du flux par l'entremise de l'équation 5.3. Ce problème est effectivement non linéaire et des itérations supplémentaires pour faire converger la forme axiale du flux sont nécessaires. À ce sujet, un schéma algorithmique de calcul sera présenté à la section 5.2 explicitant davantage la procédure.

5.1.3 Conditions nominales

Pour obtenir la distribution de puissance nominale du réacteur et effectuer la gestion du combustible à l'aide de ce modèle, il est bien sûr nécessaire que les différentes composantes du réacteur soient elles aussi en condition nominale. L'état nominal pour modéliser le coeur est donc caractérisé par certains paramètres physiques provenant des diverses études de design : températures, dimensions, propriétés nucléaires, densités, concentrations. Plus spécifiquement, le réacteur CANDU-6 est considéré à l'état nominal lorsque ses 14 contrôleurs liquides sont remplis à 50%, ses 21 barres de compensation sont pleinement insérées, ses 4 barres absorbantes solides sont reti-

Tableau 5.1 Valeurs nominales du CANDU-6

Paramètres de modélisation	Nominales
Puissance thermique coeur (MW)	2061.4
Rapport puissance thermique / fission	0.946
Pas du réseau : cellule unitaire (cm)	28.575
Puissance spécifique du grappe (kW/kg)	31.9713
Masse totale UO_2d'une grappe (kg)	19.236
Densité UO_2 (g/cm)	10.4371
Enrichissement du combustible$(\%poids)$	0.7114
Température du combustible (K)	941.29
Température du caloporteur (K)	560.66
Température du modérateur (K)	345.66
Densité du caloporteur (g/cm)	0.81212
Densité du modérateur (g/cm)	1.0829
Pureté caloporteur $(\%\, D_2O)$	99.30
Pureté du modérateurr $(\%\, D_2O)$	99.92

rées, sa concentration de poison (bore ou gadolinium) dans le modérateur est nulle et finalement lorsque les différents paramètres du tableau 5.1 sont aux valeurs spécifiées.

5.2 Méthode de résolution

Le modèle de gestion du combustible présentée à la section 5.1 découle d'une problématique nécessitant différents calculs de transport afin d'obtenir les sections efficaces requises pour effectuer le calcul de réacteur à l'équilibre du rechargement. Tous les types de calculs implémentés dans le D2G2Server sont donc sollicités. La figure 5.1 résume justement la procédure à suivre pour déterminer le taux de combustion moyen du combustible à sa sortie du réacteur.

Ce schéma algorithmique expose bien les différentes étapes à suivre, en particulier les itérations externes sur la forme axiale. Il est aussi pertinent de spécifier l'apport en sections efficaces. En effet, au milieu gauche de la figure, la présence des entrées DRAGON implique que des calculs de cellule et de supercellule doivent préalablement

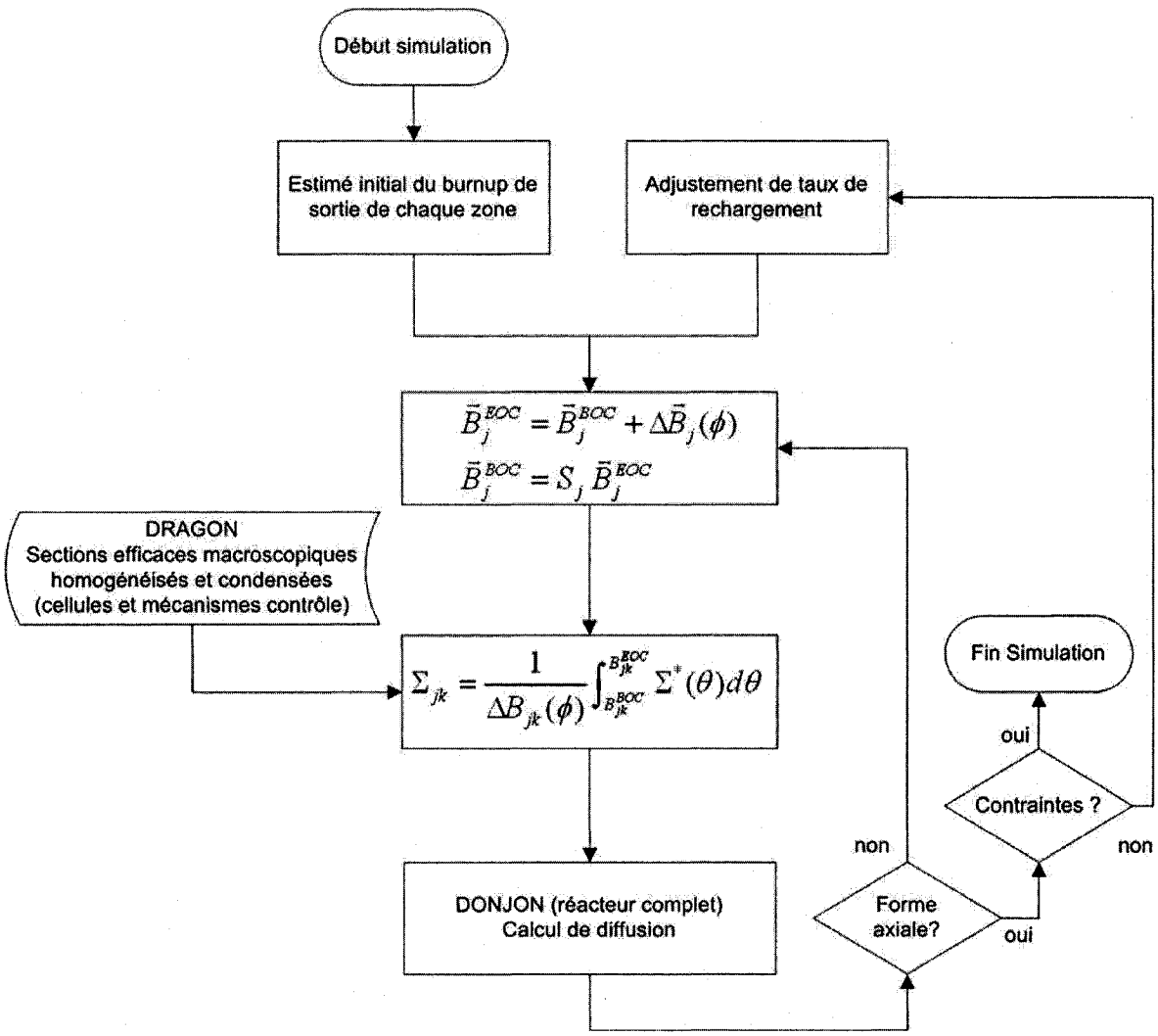

Figure 5.1 Calcul de la puissance en fonction de l'épuisement du combustible

être accomplis avant de pouvoir effectuer le calcul DONJON.

5.2.1 Calculs de cellule

D'après l'équation 5.4, la détermination des sections efficaces nécessite une tabulation de celles-ci en fonction du taux de combustion. Or, pour obtenir une telle tabulation, il faut effectuer plusieurs calculs de cellule en fonction du temps de résidence du combustible. La requête envoyée par le D2G2Client pour effectuer de tels calculs à l'état nominal est présentée à l'annexe II.5. Celle-ci permet en effet de lancer un calcul de cellule DRAGON (annexe I.1) en fournissant les différents paramètres aux fichiers modèles de simulation : CpoCell_Cpo.x2m, Evolution.c2m, GeoCANDU6.c2m, GeoCANFLEX.c2m, LibE5CNG.c2m, LibE5NAT.c2m, LibE5RUMOX.c2m. Ensuite, une fois l'exécution terminée (état constaté à l'aide de la liste de simulations actives), il est alors possible d'obtenir les résultats par l'entremise d'une requête de données sur le flux de la cellule (annexe II.6) ou demander le fichier de la sortie standard (annexe II.7) pour vérifier l'exactitude des résultats ou extraire certains paramètres tels que le facteur de multiplication K_∞ou bien le Laplacien critique B^2.

5.2.2 Calculs de supercellule

Une fois les propriétés nucléaires évolutives de la cellule déterminées, les incréments de section efficace des mécanismes de régulation doivent maintenant être déterminés. Pour y arriver, il faut tout d'abord calculer la macrolib et les éléments de structure de la supercellule (annexe I.2). La requête D2G2Client nécessaire pour lancer ce calcul DRAGON est à l'annexe II.8. Celle-ci permet de fournir les paramètres de simulation aux fichiers modèles : CellDev.x2m, CellBibG2.c2m, CellFlxG2.c2m, CellGeoG2.c2m, CellHomG2.c2m, MacStrcG2.c2m.

5.2.2.1 Mécanismes de régulation

Une fois la macrolib et la structure de supercellule trouvées, il est alors possible de lancer les calculs déterminant les sections efficaces des barres de compensation (annexe I.3) et ceux des contrôleurs liquides (annexe I.4). Les requêtes D2G2Client requises se retrouvent à l'annexe II.9 et II.10. Celles-ci permettent de spécifier les différents paramètres à traduire dans les fichiers modèles : AdjDev.x2m, AdjDevEva.c2m, AdjDevGeo.c2m, AdjDevMac.c2m, ZcuDev.x2m, ZcuDevEva.c2m, ZcuDevGeo.c2m, ZcuDevMac.c2m.

Finalement, il est important de spécifier que les calculs de mécanismes de régulation, comme ceux de la cellule unitaire, sont effectués en utilisant les conditions nominales d'opération. Par contre contrairement aux calculs de cellule, ceux des mécanismes de contrôle sont effectués une seule fois à un taux de combustion donné, car ces incréments de sections efficaces sont considérés constants lors des calculs de réacteur. C'est-à-dire qu'ils sont considérés indépendants du taux de combustion. Il sont donc calculés à un taux fixe correspondant à un temps de résidence de 130 jours à 31.9713 kW/kg de puissance développée.

5.2.3 Calculs de réacteur

La dernière étape à accomplir pour obtenir le taux de combustion de sortie du combustible est bien sûr le calcul de réacteur prenant en considération l'évolution du combustible, les mécanismes de contrôle et toute la structure du coeur. Or, pour lancer un tel calcul à l'aide du D2G2Client, la requête de l'annexe II.2 doit être soumise. Celle-ci, en plus de spécifier les no. ID des simulations de cellule et de supercellule dont les résultats sont requis pour effectuer le calcul DONJON (annexe I.5), fournit également les différentes variables nominales des fichiers modèles : CANDU.x2m, Pburncal.c2m, Pcalflu.c2m, PcritAx.c2m, PdevT.c2m, Pflax.c2m, PgeoG2.c2m, Pi-

nidet.c2m, Pinires.c2m, Pinproc.c2m, Pmacfix.c2m. Finalement, comme les autres simulations supportées par le D2G2Server, il est aussi possible d'obtenir le fichier de sortie standard de l'exécution de DONJON (annexe II.3) pour y extraire entre autres le taux de combustion moyen de sortie du combustible. Il est aussi possible d'obtenir le fichier de flux (annexe II.4) correspondant à la distribution de puissance au taux de combustion trouvé.

5.3 Spécification des systèmes

Maintenant, avant d'exposer les résultats, une présentation de la spécification des systèmes utilisés est nécessaire. En effet, il est important de connaître en détail le matériel et les logiciels de l'environnement de test afin de pouvoir éventuellement refaire la procédure ou bien comparer celle-ci avec d'autres méthodes de validation.

- Matériel

  Les équipements utilisés dans le cadre de la validation sont deux ordinateurs uniprocessseur de l'Institut de génie nucléaire de l'École Polytechnique de Montréal. Plus précisément, il s'agit des postes « kushneriuk » (Intel Pentium 4 - 2.53GhZ - cache 512KB) et de « dirac » (Intel Pentium 4 - 3.20GhZ - cache 1024KB). Tous deux sont conectés via le sous-réseau recherche.polymtl.ca (132.207.60), ont accès au même espace disque réseau (NFS) et à la même base de données MySQL résidant sur le serveur bickley.recherche.polymtl.ca.

- Logiciel

  La configuration logicielle du poste kushneriuk et de celle de dirac est la même. Le système d'exploitation utilisé est Fedora Core 3. La version du noyau (kernel) Linux est v2.6.7-1.494.2.2. Le compilateur ayant servi à la construction des outils D2G2 et des librairies avec lesquelles ce dernier se lie dynamiquement

est GCC v3.3.3 (posix-thread). La libraire pour la traduction XML (parser) est Xerces-c v2.6 (libxerces-c.so.26). Celle pour l'accès client à la base de données MySQL v4.1.1 (libmysqlclient.so.14). Tandis que la version de la librairie graphique wxWidgets est 2.4.2 (libwx_gtkd-2.4 et libwx_gtkd_gl-2.4) et celle de ACE (Adaptive Communication Environment) est quant à elle la 5.4.1 (libACE.so.5.4.1). Finalement, la version des codes neutroniques est 304S et 300F pour DRAGON et DONJON respectivement.

5.4 Résultats et Discussions

Une fois les requêtes soumises au D2G2Server, les calculs terminés et les résultats extraient à l'aide du D2G2Client, il est maintenant possible de comparer ces derniers avec les valeurs provenant des mêmes calculs effectués à l'aide des méthodes usuelles d'utilisation de DRAGON-DONJON.

Or, cette comparaison ne sera guère explicitée puisque les résultats des deux méthodes concordent parfaitement. Toutes les valeurs obtenues[2] sont identiques d'une technique à l'autre. Bref, l'usage du D2G2Server et du D2G2Client n'entraîne aucune erreur de manipulation de données lors de la confection des fichiers d'entrées GANlib et lors de la récupération des résultats.

Cependant, bien que le traitement des entrées-sorties avec les outils D2G2 soit adéquat, il est tout de même pertinent de vérifier si les calculs effectués dans le cadre de cette validation sont valables. C'est-à-dire, si les résultats obtenus par l'entremise des fichiers modèles libTPT présentés à la section 5.2, coïncident avec ceux provenant d'études antérieures utilisant la chaîne de calcul DRAGON-DONJON. Ainsi, sachant que les simulations de cellule, de supercellule et de réacteur produisent des résultats

[2]Un *diff* <D2G2> <Standard> a été effectué sur tous les fichiers de résultats

cohérents, la validation sera alors plus crédible.

5.4.1 Validation des résultats des calculs de cellule

La première vérification permet d'assurer que la variation du facteur de multiplication K_∞ en fonction du taux de combustion du combustible de la cellule unitaire concorde bel et bien avec le comportement attendu [37]. En effet, il doit y avoir une diminution rapide du K_∞ due à l'accumulation du xénon-135 au tout début, suivi du pic de réactivité produit par la variation de la concentration en plutonium (Pu-239) et de la chute graduelle résultante de l'épuisement de U-235 et de l'accumulation du Pu-240 et des autres produits de fission. Or, la variation K_∞ découlant des calculs de cellule D2G2 présentés à la figure 5.2 coïncident très bien avec la tendance prescrite.

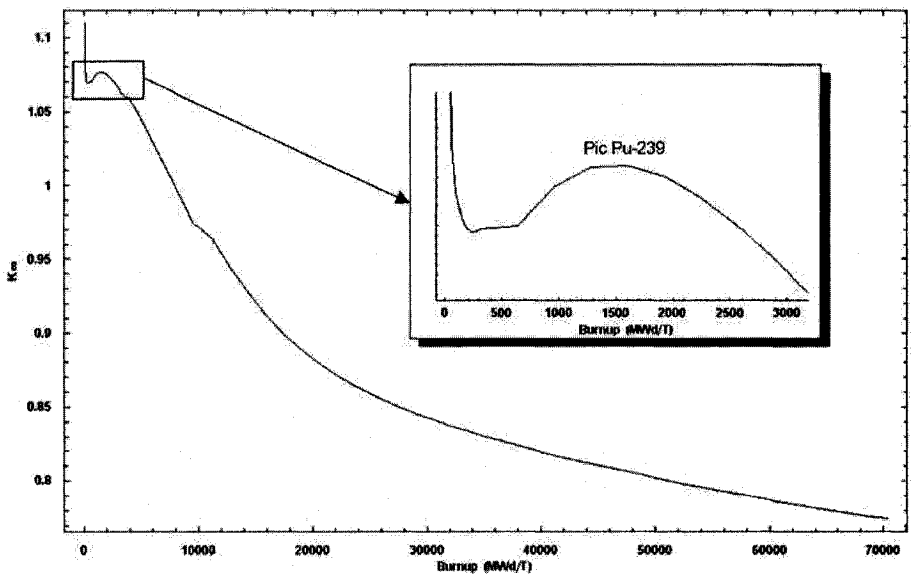

Figure 5.2 K_∞ en fonction du taux de combustion d'une cellule CANDU-6

5.4.2 Validation des résultats des calculs de supercellule

Maintenant, pour vérifier l'exactitude des résultats des calculs relatifs aux mécanismes de réactivité, leur apport en réactivité a été déterminé. Un premier calcul a

Tableau 5.2 Sections efficaces incrémentales des barres de compensation

Incremental cross-section	Adjuster(1) BCAINT	Adjuster(2) BCAOUT	Adjuster(3) BCBINT
$\Delta\Sigma_{tr}^{1}$	5.294E-04	4.297E-04	1.026E-03
$\Delta\Sigma_{t}^{1}$	7.463E-04	6.083E-04	1.374E-03
$\Delta\nu\Sigma_{f}^{1}$	-4.259E-06	-3.508E-06	-6.831E-06
$\Delta\Sigma_{s}^{1\leftarrow 1}$	6.551E-04	5.331E-04	1.233E-03
$\Delta\Sigma_{s}^{1\leftarrow 2}$	2.483E-06	2.043E-06	4.038E-06
$\Delta\Sigma_{tr}^{2}$	4.171E-04	3.403E-04	7.061E-04
$\Delta\Sigma_{t}^{2}$	4.575E-04	3.737E-04	7.402E-04
$\Delta\nu\Sigma_{f}^{2}$	2.369E-05	1.982E-05	3.897E-05
$\Delta\Sigma_{tr}^{2\leftarrow 1}$	7.320E-05	6.059E-05	1.089E-04
$\Delta\Sigma_{tr}^{2\leftarrow 2}$	1.767E-04	1.419E-04	2.979E-04

Incremental cross-section	Adjuster(4) BCCINT	Adjuster(5) BCCOUT	Adjuster(6) BCDINT
$\Delta\Sigma_{tr}^{1}$	8.102E-04	2.811E-04	4.366E-04
$\Delta\Sigma_{t}^{1}$	1.111E-03	4.056E-04	6.173E-04
$\Delta\nu\Sigma_{f}^{1}$	-5.896E-06	-2.448E-06	-3.549E-06
$\Delta\Sigma_{s}^{1\leftarrow 1}$	9.869E-04	3.526E-04	5.413E-04
$\Delta\Sigma_{s}^{1\leftarrow 2}$	3.467E-06	1.417E-06	2.068E-06
$\Delta\Sigma_{tr}^{2}$	5.960E-04	2.322E-04	3.448E-04
$\Delta\Sigma_{t}^{2}$	6.383E-04	2.590E-04	3.780E-04
$\Delta\nu\Sigma_{f}^{2}$	3.313E-05	1.373E-05	2.007E-05
$\Delta\Sigma_{tr}^{2\leftarrow 1}$	9.767E-05	4.314E-05	6.120E-05
$\Delta\Sigma_{tr}^{2\leftarrow 2}$	2.528E-04	9.665E-05	1.437E-04

permis de déterminer que les barres de compensation pleinement insérées dans le coeur ajoutent $16.3mk$ en réactivité, alors que les volumes d'eau des contrôleurs liquides permettent quant à eux de varier la réactivité de $6.7mk$. Or, ces valeurs correspondent à environ+10% pour les barres de compensation et +8% pour les contrôleurs liquides par rapport aux calculs de Roy [38]. Ce qui est sensiblement satisfaisant étant donné les nombreuses modifications qui ont été apportées au code DRAGON depuis 1994.

Finalement, les incréments des sections efficaces des différents mécanismes de régulation sont présentés au tableau 5.2 et 5.3. Ils permettent d'effectuer une comparaison

Tableau 5.3 Sections efficaces incrémentales des contrôleurs liquide remplis

Incremental cross-section	Liquid Zone Controller 32	Liquid Zone Controller 21	Liquid Zone Controller 10
$\Delta\Sigma_{tr}^{1}$	1.775E-03	1.897E-03	2.011E-03
$\Delta\Sigma_{t}^{1}$	1.112E-02	1.147E-02	1.179E-02
$\Delta\nu\Sigma_{f}^{1}$	3.126E-05	3.190E-05	3.246E-05
$\Delta\Sigma_{s}^{1\leftarrow 1}$	1.017E-02	1.049E-02	1.079E-02
$\Delta\Sigma_{s}^{1\leftarrow 2}$	-2.064E-06	-2.072E-06	-2.067E-06
$\Delta\Sigma_{tr}^{2}$	5.811E-02	6.135E-02	6.455E-02
$\Delta\Sigma_{t}^{2}$	8.862E-02	9.347E-02	9.828E-02
$\Delta\nu\Sigma_{f}^{2}$	3.840E-06	3.906E-06	4.065E-06
$\Delta\Sigma_{tr}^{2\leftarrow 1}$	9.108E-04	9.398E-04	9.673E-04
$\Delta\Sigma_{tr}^{2\leftarrow 2}$	8.799E-02	9.283E-02	9.761E-02

supplémentaire des calculs de supercellule avec les valeurs du rapport IGE-227 [39]. Or, il y a encore une légère disparité entre les valeurs présentées et celles des documents de référence. Cet écart est une fois de plus attribuable à l'évolution de DRAGON et aux techniques de simulations qui ont changées depuis les dernières années. Bref, les résultats de supercellule provenant des outils D2G2 donnent des valeurs dans les mêmes ordres de grandeur que les études antérieures, ce qui permet de conclure que l'automatisation des calculs DRAGON est valide.

5.4.3 Validation des résultats des calculs de réacteur

Maintenant, pour consolider davantage tous les calculs effectués, les résultats de simulation du réacteur sont présentés. En effet, étant donné qu'une simulation de réacteur requiert des calculs de cellule et de supercellule, l'exactitude des valeurs trouvées permet à la fois de confirmer les résultats de réacteur DONJON et ceux des sous-calculs DRAGON.

La première validation concerne la cartographie de la puissance moyenne par canal. La figure 5.3 expose les valeurs obtenues pour le calcul complet du réacteur aux

conditions nominales avec deux zones de combustion (figure 5.4). Or, les puissances trouvées coïncident assez bien avec celles données à la référence [37].

Finalement, la dernière comparaison traite des valeurs découlant de la problématique exposée à la section 5.1.2, c'est-à-dire la détermination du taux de combustion moyen de sortie des grappes. Or, le taux de combustion trouvé avec un mode de rechargement bidirectionnel à 8 grappes est de $205.56\,MWh/kg$ $(8.5651E+03\,MWd/T)$ pour la zone de combustion 1 et de $164.45\,MWh/kg$ $(6.8521E+03\,MWd/T)$ pour la zone 2. Ce qui coïncide bien avec le taux de combustion moyen provenant des valeurs expérimentales [40].

TIME-AVG CHANNEL POWERS (kW)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
A									3429	3699	3806	3806	3698	3429								
B						3069	3757	4360	4771	5027	5098	5098	5027	4771	4360	3757	3069					
C					3439	4261	4990	5534	5895	6067	6053	6053	6067	5895	5534	4990	4261	3439				
D				3510	4452	5279	5932	6359	6598	6661	6527	6527	6661	6598	6358	5932	5279	4452	3510			
E			3291	4383	5294	5985	6477	6633	6715	6687	6517	6517	6687	6715	6633	6476	5985	5293	4383	3290		
F			4073	5112	5863	6259	6526	6626	6512	6431	6338	6338	6431	6512	6626	6526	6259	5863	5112	4073		
G		3507	4703	5637	6170	6390	6547	6561	6377	6292	6297	6297	6292	6377	6561	6547	6390	6170	5637	4703	3507	
H		4108	5227	5981	6150	6324	6450	6424	6229	6145	6167	6167	6145	6229	6424	6450	6324	6150	5981	5227	4108	
J	3284	4564	5620	6226	6200	6244	6247	6190	6056	5970	5961	5961	5970	6056	6190	6247	6244	6200	6225	5620	4564	3284
K	3601	4890	5885	6389	6195	6151	6140	6054	5912	5796	5715	5715	5796	5912	6054	6140	6150	6194	6389	5884	4890	3601
L	3763	5049	6025	6514	6354	6265	6130	5990	5819	5668	5518	5518	5668	5819	5990	6129	6265	6354	6514	6024	5049	3763
M	3752	5037	6019	6529	6400	6308	6135	5970	5785	5623	5461	5461	5623	5784	5970	6135	6308	6400	6528	6019	5037	3752
N	3567	4848	5855	6415	6358	6306	6141	5982	5804	5662	5532	5531	5661	5804	5982	6141	6306	6358	6415	5855	4848	3567
O	3222	4476	5523	6165	6265	6296	6148	6026	5866	5769	5750	5750	5769	5866	6025	6148	6295	6265	6164	5523	4476	3221
P		3962	5025	5735	5890	6043	6152	6123	5942	5871	5903	5903	5871	5942	6123	6152	6043	5889	5735	5024	3962	
Q		3313	4402	5207	5582	5783	6046	6121	5988	5940	5976	5976	5940	5988	6121	6046	5783	5582	5207	4402	3312	
R			3719	4581	5113	5478	5890	6077	6029	6000	5973	5972	6000	6029	6077	5890	5478	5113	4581	3719		
S			2952	3864	4559	5191	5789	6020	6148	6152	6008	6008	6152	6148	6020	5789	5191	4559	3864	2952		
T				3099	3918	4675	5311	5746	5995	6062	5923	5923	6062	5995	5746	5311	4675	3918	3099			
U					3061	3813	4480	4994	5334	5481	5409	5409	5481	5334	4994	4480	3813	3061				
V						2754	3379	3935	4315	4551	4622	4622	4551	4315	3935	3379	2754					
W									3105	3359	3479	3479	3359	3105								

MAXIMUM CHANNEL POWER 6715. E9
MAXIMUM BUNDLE POWER 875. D11 PLANE 7

Figure 5.3 Distribution de puissance par canal (moyenne dans le temps)

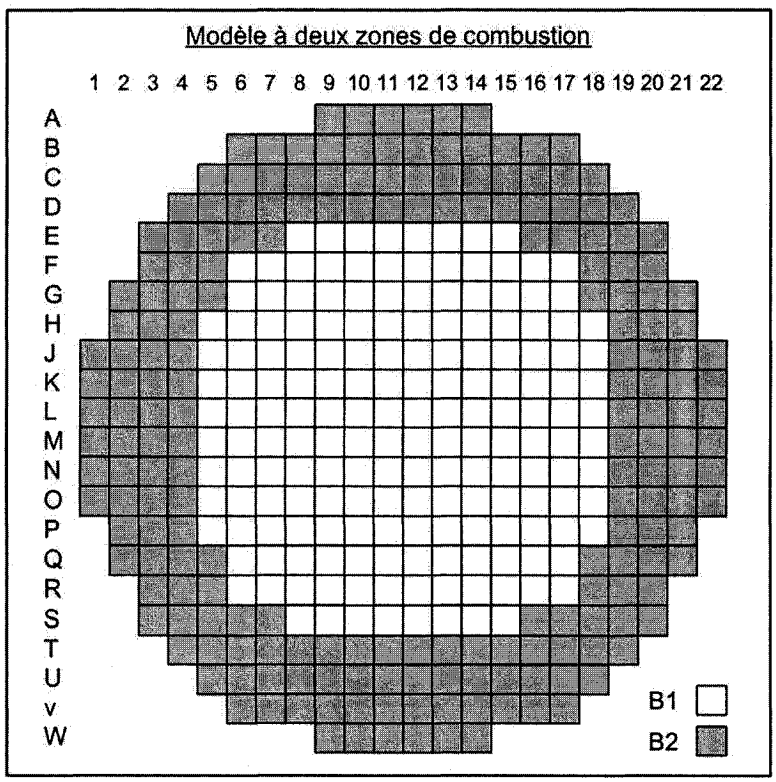

Figure 5.4 Modèle à deux zones de combustion

CHAPITRE 6

LIMITATIONS ET TRAVAUX FUTURS

La construction d'un prototype permet une création rapide et efficace. Elle permet aussi de définir et affiner les besoins des utilisateurs. En effet, expérimenter le plus tôt possible les fonctionnalités d'un prototype permet de découvrir les incomplétudes et les inconsistances de la spécification. Une rectification précoce de ces erreurs écarte alors de lourdes modifications que leur correction dans une phase plus avancée du cycle de développement occasionnerait.

En ce sens, les prototypes D2G2Server et D2G2Client ont justement permis d'expérimenter rapidement diverses possibilités d'optimisation de la chaîne de calcul DRAGON-DONJON. Donc, pour profiter des avantages relatifs aux efforts de création en mode prototype, il est normal d'expliciter les limitations et proposer certains travaux futurs.

6.1 Limitations

6.1.1 Association de données

Une des principales limitations du D2G2Server est son association de données entre les fichiers XML, les données en mémoire vive et la base de données. En effet, étant donné que les accès à la base de données sont implémentés statiquement, des modifications au niveau de la structure des requêtes XML (DTD) entraîne impérativement une réécriture de certaines routines du serveur. Par exemple, voici ce qui doit être modifié lors de l'ajout d'un champ de commentaires dans les tables de sauvegarde des données de

simulation. Bien que l'impact de ces modifications soit tout de même assez faible, il permet de bien cerner la limitation et de comprendre le parcours de l'information le long de la chaîne de traitement du D2G2Server.

- Premièrement, pour ajouter la gestion des commentaires, il faut tout d'abord modifier le fichier D2G2_XML.dtd. En effet, c'est ce fichier qui décrit la structure d'une requête client et dicte ce qu'elle peut contenir. Dans ce cas-ci, l'ajout d'un attribut de commentaires doit s'effectuer au niveau des différents sous éléments de l'élément père Simulation : Reactor, CpoCell, CellDevices, AdjDevices, ZcuDevices.

- Deuxièmement, il faut ajouter la gestion de ce nouvel attribut de commentaires dans la lecture (parsing) XML implémenté dans le fichier XML_Handler.cpp. En effet, comme chaque élément de simulation doit posséder des commentaires, il faut être en mesure de traiter ce nouvel attribut lors de la traduction du fichier XML en mémoire vive.

- Troisièmement, l'ajout de cet attribut implique aussi l'ajout d'une nouvelle variable et de ces méthodes de gestion dans le fichier de données D2G2_Data.cpp : la classe en charge de la traduction et des manipulations en mémoire vive des différentes données soumises dans les requêtes XML.

- Ensuite, pour être en mesure de conserver les commentaires, il faut évidemment modifier la base de données D2G2 en ajoutant un champ commentaire à chacune des tables de simulation. De plus, toute modification à la structure de la base de données doit aussi être effectuée au fichier D2G2.sql (annexe III) afin que ces modifications soient prises en compte lors d'une réinstallation de la base de données.

- Finalement, il faut aussi modifier toutes les fonctions qui enregistrent dans la base de données les résultats de simulation, en l'occurrence les fonctions du type

store_results. Il faut aussi modifier les fonctions qui extraient les données de la base de données soient les fonctions retreive_simlist. Les fonctions en relation avec la détermination des simulations active doivent aussi être modifiées afin que le client qui désire afficher la liste des simulations actives puisse avoir accès aux commentaires de la simulation en cours d'exécution.

Bref, il est clair que la modification de la structure des requêtes clientes XML (DTD) entraîne des modifications considérables au niveau du D2G2Server. En effet, il y a des modifications au lecteur XML, dans la classe D2G2_Data et à la base de données. C'est pourquoi le développement d'une association automatique des données (data binding) serait une solution à envisager pour corriger cette lacune. Ainsi, grâce à cette technique, il serait possible de soumettre une nouvelle structure de document XML (DTD) au D2G2Server et celui-ci se reconfigurerait automatiquement pour prendre en charge toutes les nouvelles possibilités de transfert.

6.1.2 Fichiers d'entrées statiques

Une autre limitation du D2G2Server est sa méthode de création des fichiers d'entrées DRAGON-DONJON. En effet, tel que présenté dans la section 4.2.3, les paramètres de simulation soumis par le client sont transposés directement en fichier GANlib à l'aide de fichiers modèles (templates) et de la librairie libTPT. Ainsi, l'ordre d'appel des modules est fixe et est déterminé par le fichier modèle. En d'autres mots, il est possible, en cours d'exécution du serveur, de modifier le contenu des simulations, mais pas la forme.

La raison d'être de cette implémentation provient de l'agencement quasi illimité des modules et surtout du nombre considérable d'options envisageables lors de l'appel des modules DRAGON-DONJON. Un traitement exhaustif de toutes les possibilités, sans utiliser cette technique de fichier modèle, équivaudrait effectivement à une réécriture

de la GANlib et de son langage CLE-2000. Donc, devant le travail immense qu'une telle tâche demande, cette solution statique a été privilégiée. Les outils D2G2 peuvent par conséquent être davantage comparés à des outils d'analyse que de recherche.

Par contre, il est bon de souligner que bien que cette technique de création de fichiers d'entrées puisse paraître assez restrictive, certaines décisions de conception lui confèrent quand même une certaine flexibilité. En effet, l'utilisation dans les classes Reactor_Data, CpoCell_Data, CellDevices_Data, AdjDevices_Data et ZcuDevices_Data d'ensembles indéfinis permet d'y sauvegarder un nombre indéterminé de paramètres. Par conséquent, il est possible d'assigner un nombre quelconque de variables dans un fichier modèle. Ce qui implique aussi qu'il est possible d'ordonnancer à souhait un nombre indéterminé de modules sans devoir modifier le code du D2G2Server, de la base de données ou bien de la structure de document XML (DTD). Il faut cependant rappeler que malgré cette possibilité, le nom et le nombre de fichiers modèles restent toujours statiques (modification au code source du D2G2Server nécessaire) et qu'une fois ces fichiers rédigés, leur contenu est fixe et ne peut en aucun temps être modifié lors de l'exécution du serveur.

6.1.3 Géometries variables

Une autre lacune des outils D2G2 se situe au niveau de l'organisation de la définition géométrique des domaines de simulation. En effet, bien qu'il soit possible de modifier plusieurs paramètres tels que la longueur de grappe, le pas du réseau, le diamètre des barres de compensation, la géométrie demeure difficilement modifiable. Or, ce manque de flexibilité est en partie attribuable aux méthodes de définition géométrique assscociées aux fichiers d'entrées GANlib et à l'utilisation de fichiers modèles (libTPT) par le D2G2Server. Par contre, un jour lorsque DRAGON-DONJON pourront lire des fichiers géométriques standardisés tels que STEP ou IGES, il sera alors possible d'intégrer des outils de conception assistée par ordinateur (CAO) au D2G2Client. L'utilisateur

pourra alors créer la géométrie de son choix et la soumettre au D2G2Server qui la soumettra à son tour aux applications de neutronique. À ce sujet, dans le cadre du développement du projet D2G2, des outils permettant d'intégrer de la CAO dans des applications C++ ont été mis à l'étude. Parmi les cadres de travail prometteurs, OpenCascade[1] est sans aucun doute celui qui s'est démarqué le plus. Il est entièrement gratuit, son code source est disponible et offre une grande variété de fonctionnalités toutes supportées par une excellente documentation.

6.2 Travaux futurs

6.2.1 Visualisation graphique

L'un des travaux futurs qui apportera grandement aux D2G2Client est sans aucune l'intégration des outils de visualisation graphique. La possibilité de visualiser la carte de flux d'une cellule ou bien du réacteur sera effectivement une amélioration importante pour le client. Aussi, la possibilité de parcourir les différentes géométries à simuler et de pouvoir y sélectionner certains éléments (grappe, modérateur, caloporteur) afin de modifier leurs attributs (température, densité, irradiation) sera aussi une innovation majeure facilitant grandement la conception des requêtes de simulation. Bref, une fois la visualisation ajoutée, les utilisateurs se demanderont à coup sûr comment il ont pu se passer de ces fonctionnalités.

6.2.2 Sauvegarde des données

Une autre amélioration dont le D2G2Server pourrait éventuellement profiter se situe au niveau de la sauvegarde des données. Actuellement, cette sauvegarde est assurée par une base de données MySQL, couplée à un espace disque réseau NFS. Cette

[1] http ://www.opencascade.org

méthode possède l'avantage d'être simple et son implémentation est très rapide. Par contre, afin d'accélérer le stockage et standardiser davantage les échanges de données, l'utilisation d'une librairie plus puissante conçue pour le traitement des données scientifiques doit être envisagée. À ce sujet, la librairie HDF[2], un standard de plus en plus répandu dans la communauté scientifique, semble aujourd'hui un choix incontournable[41].

6.2.3 Interface administrateur

Le développement d'une interface administrateur pourrait aussi être une fonctionnalité très intéressante pour gérer l'intégrité des résultats de simulation, ajouter des usagers, démarrer ou changer le statut d'un D2G2Server ou bien modifier la liste des simulations actives. Actuellement une interface de gestion PHP[3] est utilisée pour effectuer ces diverses tâches de gestion. Il serait cependant beaucoup plus pratique d'ajouter ces outils de gestion au D2G2Client. Certains utilisateurs, authentifiés dans le groupe d'administration, pourraient alors soumettre des requêtes administratives au D2G2Server.

6.2.4 Récupération des simulations

Un autre service très utile pouvant être ajouté au D2G2Server est la capacité de récupération des simulations actives advenant un arrêt non désiré du serveur. Présentement le serveur est seulement conçu pour gérer en toute sécurité des arrêts commandés (SIGINT, SIGTERM, SIGQUIT). Par contre, lors d'une panne d'alimentation électrique par exemple, aucun mécanisme de récupération n'a été implémenté. Or, la solution proposée est simple. Après un arrêt non désiré, lors du redémarrage du serveur, il faudrait parcourir la liste des simulations actives et vérifier celles qui ne

[2]http ://hdf.ncsa.uiuc.edu/HDF5
[3]http ://www.phpmyadmin.net

sont pas en train de s'exécuter, mais qui possèdent quand même la mention *running*. Dans ce cas précis, il suffirait de les redémarrer toutes les simulations répondant à ces deux critères, puisqu'elles ont été momentanément interrompues pour une raison inconnue. En effet, contrairement aux simulations qui ont été volontairement annulées par son utilisateur (statut *failed*), les fichiers de simulations non actives qui auraient encore une inscription dans la liste des simulations actives peuvent encore être récupérés dans le dossier de sauvegarde des données. Le redémarrage du calcul DRAGON ou DONJON serait donc facilement réalisable sans que l'utilisateur ait à resoumettre sa requête.

6.2.5 Automatisation des requêtes

Finalement, l'automatisation de la construction des requêtes de simulation est l'une des fonctionnalités, avec l'intégration de la visualisation 3D, qui fera considérablement évoluer le D2G2Client. En effet, grâce à cette avancée, il sera possible de soumettre des requêtes de simulation s'exécutant automatiquement et dont l'arrêt sera gouverné par une condition d'optimisation. C'est-à-dire, une requête de simulation pourra être soumise au D2G2Server avec non seulement les paramètres d'entrées habituels, mais aussi avec des intervalles à travers lesquels ces derniers pourront varier. Il sera alors possible d'effectuer une boucle sur la même simulation en faisant évoluer les paramètres d'entrées jusqu'à l'obtention d'un résultat souhaité. Par exemple, il pourra être possible de faire varier la géométrie des barres de compensation, les paramètres locaux de la cellule ou bien simplement le pas du réseau jusqu'à l'obtention de la forme de flux désirée.

Or, pour implémenter directement cette fonctionnalité d'optimisation dans les outils D2G2, des routines avancées de traitement des fichiers de résultats XML devront être ajoutées. Il devra alors être possible d'extraire certaines valeurs bien précises pour vérifier si le calcul évolue bel et bien vers la condition d'arrêt. Maintenant, en ce qui

a trait à la construction des requêtes devant faire varier les paramètres, la plupart des utilitaires sont déjà présents dans le code du D2G2Server et cet ajout ne nécessitera pas de longues modifications. Justement, la possibilité de relancer une simulation déjà effectuée est déjà implémentée dans le serveur. Concrètement, il est possible, lors de la soumission d'une nouvelle requête, de fournir le numéro ID d'une simulation présente dans le la liste des simulations complétées. Le serveur se charge alors de remplacer les données de l'ancienne simulation par celles de la nouvelle. Bref, il est déjà possible d'optimiser les résultats d'un calcul en faisant varier ces paramètres d'entrées, sans toutefois que la procédure soit automatisée.

CONCLUSION

L'objectif initial du projet D2G2 a été atteint. Une optimisation de la chaîne de calcul DRAGON-DONJON a été réalisée. En effet, grâce à de cette initiative de développement logiciel, plusieurs innovations relatives au contrôle de l'exécution des simulations neutroniques ont été achevées, certaines envisagées et d'autres pourront s'en inspirer.

Plus précisément, ce projet a donné naissance à deux applications : le D2G2Server et le D2G2Client. Implémentées suivant une architecture du type tableau noir client-serveur, celles-ci ont été créées à l'aide de puissants cadres de travail orientés-objet. Elles profitent par conséquent de tous les avantages reliés à l'utilisation de cette approche informatique caractérisée par une grande utilisabilité, maintenabilité et portabilité.

Techniquement, le D2G2Client est en fait l'interface graphique avec laquelle l'utilisateur interagit pour soumettre des calculs neutroniques. Il a été créé à l'aide du cadre de travail wxWidgets. Il permet entre autres d'élaborer des requêtes de simulation, de suivre l'évolution des calculs et de visualiser certains résultats. Éventuellement, l'intégration d'outils graphiques plus avancés viendra assurément enrichir les possibilités de visualisation. Un affichage tridimensionnel des cellules et du réacteur permettra alors de confectionner plus rapidement de nouvelles requêtes ou bien d'interpréter plus aisément les résultats.

Le D2G2Server est quant à lui un serveur multitâche, basé sur un modèle d'accès simultané, élaboré à partir du cadre de travail ACE. Il s'occupe de répertorier et d'analyser les requêtes de simulation neutroniques provenant du D2G2Client, de manipuler les processus DRAGON et DONJON, de sauvegarder les données de calculs

et finalement de fournir aux clients les résultats. De son côté, certaines limitations reliées à l'association des données, à l'élaboration des fichiers d'entrées GANlib et à la flexibilité des géométries étudiées devront être corrigées. De plus, d'intéressantes améliorations touchant la sauvegarde des données et l'automatisation des requêtes pourront être ajoutées.

Bref, les deux applications développées dans le cadre du projet D2G2, bien qu'elles ont été validées en déterminant le taux de combustion moyen des grappes de combustible à la sortie du réacteur lorsque celui-ci est à l'équilibre du rechargement, demeurent tout de même de simples prototypes. Plusieurs travaux doivent encore être entamés et certaines limitations corrigées. Par contre, la standardisation des échanges de données, l'ajout d'une interface graphique et l'architecture du tableau noir méritent une attention particulière. En effet, ces solutions d'optimisation de la chaîne de calcul DRAGON-DONJON sont totalement nouvelles et de plus facilement réutilisables dans divers contextes. Il serait alors formidable, d'ici quelques années, de constater que le projet D2G2 fut l'un des précurseurs de l'innovation informatique concernant le paradigme des méthodes de calcul neutronique. En effet, l'industrie nucléaire a grandement besoin de se munir d'un cadre de travail flexible et performant pour l'élaboration de ces futures applications informatiques. Elle a en effet la tâche de rendre accessibles, à la génération de non-initiés, plusieurs connaissances très spécialisées, afin d'assurer la pérennité de ses développements logiciels.

En définitive, l'industrie nucléaire doit absolument réussir à partager efficacement cette longue expertise qu'elle a su acquérir depuis les 30 dernières années. Plusieurs autres domaines scientifiques, dont l'aérospatiale et la physique de particules, ont depuis fort longtemps atteint cet objectif. Le projet D2G2 se veut donc une initiative logicielle voulant aider l'industrie nucléaire à atteindre ces ambitions de modernisation.

RÉFÉRENCES

[1] G. Marleau, R. Roy, and A. Hébert. *DRAGON : A Collision Probability Transport Code for Cell and Supercell Calculations.* Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, 1994.

[2] E. Varin, A. Hébert, R. Roy, and J. Koclas. *A User Guide for DONJON.* Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, 2000.

[3] R. Roy, J. Koclas, W. Shen, D. A. Jenkins, D. Altiparmakov, and B. Rouben. Reactor Core Simulations in Canada. In *PHYSOR - The Physics of Fuel Cycles and Advanced Nuclear Systems*, Chicago, 2004.

[4] Lewis Hanes and Joseph Naser. Use of Visualization Techniques to Improve Human Decision-Making in Nuclear Power Plants. In *Trans. American Nuclear Society*, Washington D.C., 2004. 91, 34-35.

[5] A. L. Schwarz, R. A. Schwarz, and L. L. Carter. 3-D Plotting Capabilities in the Visual Editor for Release 5 of MCNP. In *Proceedings of the 2003 Topical on Mathematical and Computational Sciences*, Gatlinburg, Tennessee, April 2003.

[6] Kenneth A. Van Riper. Mesh and Volume Fraction Capabilities in Moritz. In *Workshop on Common Tools and Interfaces for Deterministic Radiation Transport, for Monte Carlo, and Hybrid Codes (3D-TRANS-2003)*, Issy-les Moulineaux, France, September 2003.

[7] J. R. Lamarsh. *Introduction to Nuclear Reactor Theory.* Addison-Wesley, 1966.

[8] Daniel Rozon. *Introduction à la Cinétique des Réacteurs Nucléaires.* Éditions de l'École Polytechnique de Montréal, 1992.

[9] R. Roy and A. Hébert. *The GAN Generalized Driver.* Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, 2000.

[10] R. Roy. *The CLE-2000 Tool-Box.* Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, 1999.

[11] J. D. Irish and S. R. Douglas. Validation of WIMS-IST. In *23rd Annual Conference of the Canadian Nuclear Society*, Toronto, June 2-5 2002.

[12] G. Marleau. The Verification of DRAGON : Progress and Lessons Learned. In *23rd Annual Conference of the Canadian Nuclear Society*, Toronto, Canada, June 2-5 2002.

[13] B. Rouben. RFSP-IST, The Industrial Standard Tool Computer Program for CANDU Reactor Core Design and Analysis. In *13rd Pacific Basin Nuclear Conference*, Shenzhen, China, October 21-25 2002.

[14] Robert C. Seacord. *Modernizing legacy systems : software technologies, engineering processes, and business practices.* Addison-Wesley, 2003.

[15] Cassiano de Oliviera. GERARD : A General Environment for Radiation Analysis, Research and Design. In *Workshop on Common Tools and Interfaces for Deterministic Radiation Transport, for Monte Carlo, and Hybrid Codes (3D-TRANS-2003)*, Issy-les Moulineaux, France, September 2003.

[16] Rémy Fannader. *UML : Principes de Modélisation.* Dunod, 1999.

[17] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[18] Michel Ezran, Maurizio Morisio, and Colin Tully. *La Réutilisation Logicielle.* Eyrolles, Paris, 1999.

[19] Robert W. Sebesta. *Concepts of programming languages.* Addison-Wesley, 5th edition, 2002.

[20] P. Roques and F. Vallée. *UML en Action.* Eyrolles, 2nd edition, 2002.

[21] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process.* Addison-Wesley, 1999.

[22] P. Rheaume, J. F. Lefebvre, J. Koclas, and R. Roy. The D2G2 Project : A New Software Tool for Nuclear Engineering Design in Canada. In *6th International Conference on Simulation Methods in Nuclear Engineering - Canadian Nuclear Society*, Montreal, October 12-15 2004.

[23] Craig Larman. *Agile and iterative development.* Addison-Wesley, Boston, 2004.

[24] James Rumbaugh. *OMT : Modélisation et Conception Orientées objets.* Masson, Paris, 1997.

[25] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[26] Daniel Corkill. Blackboard systems. *AI Expert*, 6(9), January 1991.

[27] Daniel Serain. *Middleware et Internet.* InterEditions, 1999.

[28] M. Tatsumi and A. Yamamoto. SCOPE2 : Object-Oriented Parallel Code for Multi-Group Diffusion/Transport Calculations in Three-Dimensional Fine-Mesh Reactor Core Geometry. In *Physor-2002*, Seoul, Korea, 2002.

[29] M. Dahmani, R. Roy, and J. Koclas. New Computational Methodology for Large 3D Neutron Transport Problems. In *PHYSOR-2004*, Chicago, 2004.

[30] D. C. Schmidt and S. D. Huston. *C++ Network Programming : Resolving Complexity Using Ace and Patterns (C++ in-Depth Series).* Addison-Wesley Longman Publishing Co., Inc., 2001.

[31] D. C. Schmidt, S. D. Huston, and F. Buschmann. *C++ Network Programming : Systematic Reuse with ACE and Frameworks, Vol. 2.* Pearson Education, 2002.

[32] S. D. Huston, J. C. Johnson, and U. Syyid. *The ACE Programmer's Guide : Practical Design Patterns for Network and Systems Programming.* Addison-Wesley Longman Publishing Co., Inc., 2003.

[33] Simon St-Laurent. *XML : A Primer.* Hungry Minds, Incorporated, 1999.

[34] Jeremy D. Zawodny and Derek J. Balling. *High Performance MySQL.* O'Reilly, 2005.

[35] Jean-François Lefebvre. Outils de Visualisation pour DRAGON et DONJON. Master's thesis, Institut de génie nucléaire de l'École Polytechnique de Montréal, en cours.

[36] Taran Rampersad. wxWindows for Cross-platform Coding. *Linux J.*, 2003(111) :6, 2003.

[37] Daniel Rozon. *Gestion du Combustible Nucléaire.* École Polytecnique de Montréal, 2003.

[38] R. Roy, G. Marleau, J. Tajmouati, and D. Rozon. Modelling of CANDU Reactivity Control Devices with the Lattice Code DRAGON. *Annals of Nuclear Energy*, 21 :115–132, 1994.

[39] R. Roy and E. Varin. DRAGON Models Used for Producing Two-Group CANDU-6 Nuclear Properties. Technical Report IGE-227, École Polytechnique de Montréal, 1999.

[40] G. Hotte, M. Beaudet, J. Koclas, and D. Rozon. Accuracy of Burnup History Simulations Integrating In-Core Flux Detector Measurements and Diffusion Theory. In *PHYSOR - Physics of Reactors*, Marseille, 1990.

[41] R. Moffett and D. Shalaby. Assessment of Standard Formats for Data Exchange Between Scientific Codes. In *6th International Conference on Simulation Methods in Nuclear Engineering - Canadian Nuclear Society*, Montreal, October 2004.

ANNEXE I

FICHIERS D'ENTRÉES GANLIB

Listing I.1 Fichier d'entrées GANlib avec variables libTPT pour la simulation de cellule unitaire avec DRAGON

```
*----------
*  Nom             : CPOCANDU.x2m
*  Type            : Input DRAGON
*  Usage           : Generer des bases de donnees reacteur CPO
*                    pour du conbustible de type de CANDU
*  Auteur          : G. Marleau (2001/04/02)
*                    P. Rheaume (2004/09/02)
*----------
*  Procedures utilisees :
*
*  GeoCANDU6.c2m   : Definir geometrie CANDU a 37 elements
*  GeoCANFLEX.c2m  : Definir geometrie CANDU a 43 elements
*  LibE5NAT.c2m    : Definir bibliotheque ENDF/B-V avec
*                    uranium naturel
*  LibE5RUMOX.c2m  : Definir bibliotheque ENDF/B-V avec
*                    uranium recupere de REP
*  LibE5CNG.c2m    : Definir bibliotheque ENDF/B-V avec
*                    combustible CANDU-NG
*  Evolution       : Resoudre probleme et evoluer
*----------
*  Parametres de simulation :
*
*  TypeGrappes     : Type de grappes
*                    CANDU6  -> Cellule CANDU-6 a 37 elements
*                    CANFLEX -> Cellule CANFLEX a 43 elements
*  TubePress       : Chaine de caracteres decrivant le type de
*                    tubes de pression
*                    options :
*                    G2      -> Tube de pression CANDU-6 de Gentilly-2
*                    CNG     -> Tube de pression de CANDU-NG
*  PasReseau       : Pas de reseau
*  TypeComb        : Chaine de caracteres decrivant le type de combustible
*                    options :
*                    NAT     -> Uranium naturel
*                    RUMOX   -> Uranium reccupere des REP
*                               (voir tableau page 5)
*                    CNG     -> CANDU-NG
*  TempComb        : Temperature combustible (K)
*  PurtCalo        : % D2O caloporteur
*  TempCalo        : Temperature caloporteur (K)
*  TempModer       : Temperature moderateur (K)
```

```
*  PurtModer      : % D2O moderateur
*  TitreCalo      : Titre du caloporteur (%)
*  Fuites         : Modele de fuites
*                   AUCUNE -> Aucune fuite
*                   B0     -> B0 homogene
*                   B1     -> B1 homogene
*  Puissance      : Puissance specifique de grappe (kw/kg)
*  Burnup         : Burnup moyen de sortie (GWj/TU)
*----------
*  Output Fichiers ASCII :
*
*  fnat        : Fichier CPO pour caloporteur present
*  CPOV        : Fichier CPO pour vidange caloporteur
*  rnat        : Fichier CPO pour le rÃl'flecteur
*----------
* Define procedures and modules
* Procedures pouvant etre appelees par cette procedure
*----------
PROCEDURE
  GeoCANDU6    GeoCANFLEX
  LibE5NAT     LibE5RUMOX
  LibE5CNG
  Evolution
  ;
MODULE
  DELETE:
  ;
*----
*  Define input and output data structures
*----
STRING
  TypeGrappes TubePress
  TypeComb Fuites
  ;
EVALUATE
  TypeGrappes TubePress
  TypeComb Fuites :=
  "${BundleType}" "${TubePresType}"
  "${FuelType}" "${Leakage}"
  ;
REAL
  PasReseau TempComb PurtCalo
  TempCalo TitreCalo TempModer PurtModer
  Puissance Burnup
  ;
EVALUATE
  PasReseau TempComb PurtCalo
  TempCalo TitreCalo TempModer PurtModer
  Puissance Burnup :=
  ${LatticePitch} ${TempComb} ${PurtCalo}
  ${TempCalo} ${TitreCalo} ${TempModer} ${PurtModer}
  ${Power} ${Burnup}
  ;
SEQ_ASCII fnat CPOV rnat :: FILE
          "${fnat}"
          "CPOV"
```

```
          "${rnat}" ;
*-----
* Define local data structures needed
* for execution
* Define and evaluate local variables
*-----
LINKED_LIST
  GEOFLXC GEOSHIC GEOFLXV GEOSHIV
  Biblio
  ;
SEQ_BINARY
  TRKFLXC TRKSHIC TRKFLXV TRKSHIV
  ;
*
*  acceder aux informations de geometrie
*  defaut: CANDU6
*
IF TypeGrappes "CANFLEX" = THEN
  GEOFLXC GEOSHIC GEOFLXV GEOSHIV
  TRKFLXC TRKSHIC TRKFLXV TRKSHIV := GeoCANFLEX ::
  <<TubePress>> <<PasReseau>>
  ;
ELSE
  GEOFLXC GEOSHIC GEOFLXV GEOSHIV
  TRKFLXC TRKSHIC TRKFLXV TRKSHIV := GeoCANDU6  ::
  <<TubePress>> <<PasReseau>>
  ;
ENDIF ;
*
*  preparer bibliotheques
*  defaut: NAT
*
IF TypeComb "RUMOX" = THEN
  Biblio := LibE5RUMOX ::
  <<TempComb>> <<PurtCalo>> <<TempCalo>> <<TitreCalo>>
  <<TempModer>>   <<PurtModer>>
    ;
ELSEIF TypeComb "CNG" = THEN
  Biblio := LibE5CNG   ::
  <<TempComb>> <<PurtCalo>> <<TempCalo>> <<TitreCalo>>
  <<TempModer>>   <<PurtModer>>
    ;
ELSE
  Biblio := LibE5NAT   ::
  <<TempComb>> <<PurtCalo>> <<TempCalo>> <<TitreCalo>>
  <<TempModer>>   <<PurtModer>>
    ;
ENDIF ;
*
*  Resoudre probleme et produire base de donnee reacteur
*
fnat CPOV rnat := Evolution Biblio
               GEOFLXC GEOSHIC GEOFLXV GEOSHIV
               TRKFLXC TRKSHIC TRKFLXV TRKSHIV ::
  <<Fuites>>
  <<Puissance>>   <<Burnup>>
```

```
  ;
*
*  eliminer fichiers perimes
*
Biblio
GEOFLXC TRKFLXC GEOSHIC TRKSHIC
GEOFLXV TRKFLXV GEOSHIV TRKSHIV
  := DELETE:
Biblio
GEOFLXC TRKFLXC GEOSHIC TRKSHIC
GEOFLXV TRKFLXV GEOSHIV TRKSHIV
  ;
*
*  terminer execution
*
QUIT "LIST" .
```

Listing I.2 Fichier d'entrées GANlib avec variables libTPT pour la simulation de macrolib et des structures de supercellule avec DRAGON

```
*DECK Device_CellDev.x2m
*----
*  Nom            : Device_CellDev.x2m
*  Type           : Input DRAGON
*  Usage          : Generer les macrolib de cellule Gentilly-2
*                   pour un burnup de 4.156 GWj/T
*                   (130 jours a 31.9713 kW/kg)
*                   et la macrolib pour les materiaux de structure
*  Auteur         : G. Marleau
*  Date           : 2002/12/03
*                 : P. Rheaume (2004/09/02)
*
*  Procedure requise  :
*    MacStrcG2    : macrolib pour le materiaux de structure
*    CellBibG2    : bibliotheque de section efficace
*                   pour le calcul de cellule
*    CellGeoG2    : Geometries pour le calcul de cellule
*    CellFlxG2    : Calcul de flux et evolution pour une cellule
*    CellHomG2    : Homogeneisation et generation de la macrolib
*                   pour la cellule
*
*  Output Fichiers ASCII :
*
*  MacroC         : Macrolib de la cellule pour le calcul
*                    des devices
*  MacroS         : Macrolib de la structure pour le calcul
*                   des devices
*----
*  Procedures et Modules
*----
PROCEDURE    MacStrcG2   CellBibG2
             CellGeoG2   CellFlxG2   CellHomG2          ;
MODULE       DELETE: END:                               ;
```

```
*----
*  Definir le type de geometrie et de bibliotheque
*  RefLib        : Type de bibliotheque
*                  "WL"    -> Bibliotheque Winfrith WIMS-AECL
*                  "E5"    -> Bibliotheque ENDF/B-V WIMS-AECL
*                             (defaut)
*                  "E6"    -> Bibliotheque ENDF/B-VI WIMS-AECL
*  TempComb      : Temperature combustible (K)
*  EnriComb      : Enrichissement combustible (%)
*  DensComb      : Densite combustible (g/cc)
*  TempCalo      : Temperature caloporteur (K)
*  PurtCalo      : % D2O caloporteur
*  TempMod       : Temperature moderateur (K)
*  PurtMod       : % D2O moderateur
*  BoreMod       : Bore dans le moderateur (ppm)
*----
STRING       RefLib                                    ;
EVALUATE     RefLib                                    :=
            "${Microlib}" ;
*----
*  Definir les options locales pour les geometries
*----
STRING       Calo     Auto    Flux   Homo              ;
EVALUATE     Calo     Auto    Flux   Homo              :=
            "Present" "Auto" "Flux"  "Homo"            ;
*----
*  Definir les options locales pour les sections efficaces
*----
REAL         TempComb EnriComb  DensComb               ;
REAL         TempCalo PurtCalo                         ;
REAL         TempMod  PurtMod   BoreMod                ;
INTEGER      Impression                                ;
EVALUATE     TempComb EnriComb  DensComb               :=
             ${TempComb} ${EnriComb} ${DensComb} ;
EVALUATE     TempCalo PurtCalo                         :=
             ${TempCalo} ${PurtCalo} ;
EVALUATE     TempMod  PurtMod   BoreMod                :=
             ${TempMod} ${PurtMod} ${BoreMod} ;
*----
*  Definir les options locales pour les calculs de cellule
*----
REAL         Power   Timei  Timef                      ;
EVALUATE     Power   Timei  Timef                      :=
             ${Power} ${Timei} ${Timef} ;
*----
*  Definir les parametrees de debug
*----
EVALUATE     Impression                                :=
             1                                         ;
*----
*  Extraire la macrolib pour les materiaux de structure
*  et la bibliotheque pour le calcul de cellule
*  le defaut est ENDFB5
*----
SEQ_ASCII    MacroC MacroS :: FILE
             "${MacroC}"
```

```
            "${MacroS}" ;
LINKED_LIST  Biblio Macrolib MacroEdi                          ;
Macrolib := MacStrcG2                                          ::
  <<RefLib>>
  <<TempMod>> <<PurtMod>> <<BoreMod>> <<Impression>>           ;
Biblio   := CellBibG2                                          ::
  <<RefLib>>
  <<TempComb>>  <<EnriComb>>  <<DensComb>>
  <<TempCalo>>  <<PurtCalo>>
  <<TempMod>>   <<PurtMod>>   <<BoreMod>>
  <<Impression>>                                               ;
*-----
*  Sauver la macrolib pour le materiaux de structure
*-----
MacroS := Macrolib ;
Macrolib := DELETE: Macrolib                                   ;
*-----
*  Obtenir les Geometries
*-----
LINKED_LIST  GeoS    GeoF    GeoH                              ;
LINKED_LIST  TrkS    TrkF    TrkH                              ;
SEQ_BINARY   IntLinS IntLinF IntLinH                           ;
*-----
*  Autoprotection
*-----
GeoS TrkS IntLinS := CellGeoG2                                 ::
  <<Calo>> Auto <<Impression>>                                 ;
*-----
*  Calcul de cellule
*-----
GeoF TrkF IntLinF := CellGeoG2                                 ::
  <<Calo>> Flux <<Impression>>                                 ;
*-----
*  Homogeneisation
*-----
GeoH TrkH IntLinH := CellGeoG2                                 ::
  <<Calo>> Homo <<Impression>>                                 ;
*-----
*  Calcul de flux
*-----
LINKED_LIST  Flux  Burnup                                      ;
Flux Burnup Biblio := CellFlxG2 Biblio TrkS    TrkF
                                       IntLinS IntLinF         ::
  <<Power>> <<Timei>> <<Timef>> <<Impression>>                 ;
Burnup      := DELETE:  Burnup                                 ;
*-----
*  Homogenisation et creation de la macrolib
*-----
MacroEdi := CellHomG2 Flux Biblio GeoH TrkF  IntLinF           ::
  <<Impression>>                                               ;
MacroC := MacroEdi  ;
MacroEdi := DELETE: MacroEdi                                   ;
*-----
*  Eliminer les structures inutiles et terminer
*-----
Flux         := DELETE: Flux                                   ;
```

```
Biblio       := DELETE: Biblio                          ;
GeoS TrkS IntLinS := DELETE: GeoS TrkS IntLinS          ;
GeoF TrkF IntLinF := DELETE: GeoF TrkF IntLinF          ;
GeoH TrkH IntLinH := DELETE: GeoH TrkH IntLinH          ;
END:                                                    ;
QUIT "LIST"                                             .
```

Listing I.3 Fichier d'entrées GANlib avec variables libTPT pour la simulation de barres de compensation avec DRAGON

```
*DECK AdjDev.x2m
*----
*  Name           : AdjDev.x2m
*  Use            : Perform 3-D reactivity device analysis
*                   Adjuster bars
*  Date           : 2002/10/09
*                 : 2004/09/02 Modified to export CPO
*  Input          : Procedures  -->
*                   DevGeo      ->  generate 3-D reactivity
*                                   device geometry
*                   DevMac      ->  generate 3-D macrolib
*                                   for reactivity devices
*                   DevEva      ->  solve 3-D transport problem
*                                   for reactivity devices
*                   ASCII_FILES --> required by DevMac
*                   ${MacroC} ${MacroS} ->
*                                   General macrolib for cell
*  Output         : ASCII_FILES -->  adj1nat, adj2nat, adj3nat
*                                    adj4nat, adj5nat, adj6nat
*----
* Define procedures and modules
*----
PROCEDURE  DevEva     DevGeo     DevMac  ;
MODULE     EDI:       CPO:
           DELETE:    BACKUP:    END:    ;
*----
*  Define input and output data structures
*----
STRING     RefLib                        ;
EVALUATE   RefLib                        :=
          "${Microlib}" ;
SEQ_ASCII adj1nat adj2nat adj3nat
          adj4nat adj5nat adj6nat
          MacroC MacroS
          :: FILE
          "${adj1nat}"
          "${adj2nat}"
          "${adj3nat}"
          "${adj4nat}"
          "${adj5nat}"
          "${adj6nat}"
          "${MacroC}"
          "${MacroS}" ;
```

```
*-----
* Define local data structures needed
* Create all adjuster compo ADJ1 to ADJ6,
* Initialize device type
* Initialize device location
*    Start with "NO" then consider
*   "OUT" and "IN" where
*   "NO" is for device and structure absent
*   "OUT" is for device OUT with structure present
*   "IN" is for device IN with structure present
*    Note: For LZC, IN means filled with water
*    and OUT means empty
*-----
* ADJ1
*-----
STRING     Device    :=  "ADJ1" ;
INTEGER    PrtLev     := 2                ;
STRING     DevLocation                    ;
STRING     NameRec                        ;
*-----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*-----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                       ;
SEQ_BINARY Tracks                         ;
*-----
*  Analyse and track device
*-----
Geometry Volumes Tracks  := DevGeo        ::
  <<Device>> <<PrtLev>>                   ;
Geometry := DELETE: Geometry              ;
Backfile := BACKUP: Volumes               ;
*-----
*  Import ASCII library
*-----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*-----
*  Analyze device in "NO" location
*-----
EVALUATE   DevLocation                     :=
          "NO"                             ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>> <<PrtLev>>    ;
Fluxes   := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                               ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Fluxes Macrolib Volumes   ::
```

```
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlNO                        ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO      ;
*-----
*  Analyze device in "OUT" location
*-----
EVALUATE   DevLocation                      :=
          "OUT"                             ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>>  <<PrtLev>>    ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                         ;
Backfile := BACKUP: Backfile  FlOUT                       ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT    ;
*-----
*  Analyze device in "IN" location
*-----
EVALUATE   DevLocation                      :=
          "IN"                              ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>> <<PrtLev>>     ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlIN                        ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN      ;
*-----
*  Export result files
*-----
adj1nat  := Compo                    ;
*-----
*  Remove remaining temporary files
*-----
```

```
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*----
* ADJ2
*----
EVALUATE   Device   :=  "ADJ2" ;
*----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                          ;
SEQ_BINARY Tracks                            ;
*----
*  Analyse and track device
*----
Geometry Volumes Tracks  := DevGeo           ::
  <<Device>> <<PrtLev>>                      ;
Geometry := DELETE: Geometry                 ;
Backfile := BACKUP: Volumes                  ;
*----
*  Import ASCII library
*----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*----
*  Analyze device in "NO" location
*----
EVALUATE   DevLocation                       :=
          "NO"                               ;
Macrolib := DevMac MACCELL MACSTRC           ::
  <<Device>> <<DevLocation>> <<PrtLev>>      ;
Fluxes  := DevEva Macrolib Volumes Tracks    ::
  <<PrtLev>>                                 ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                     ;
Edition := EDI: Fluxes Macrolib Volumes    ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>     ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                             ;
Backfile := BACKUP: Backfile  FlNO                           ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO         ;
*----
*  Analyze device in "OUT" location
*----
EVALUATE   DevLocation                       :=
          "OUT"                              ;
```

```
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>>  <<PrtLev>>   ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                                ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes            ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>   ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlOUT                        ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT     ;
*----
*  Analyze device in "IN" location
*----
EVALUATE   DevLocation                      :=
         "IN"                               ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>> <<PrtLev>>     ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                                ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes            ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>   ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                           ;
Backfile := BACKUP: Backfile  FlIN                         ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN       ;
*----
*  Export result files
*----
adj2nat  := Compo                    ;
*----
*  Remove remaining temporary files
*----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*----
* ADJ3
*----
EVALUATE   Device   :=  "ADJ3" ;
*----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*----
```

```
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                        ;
SEQ_BINARY Tracks                          ;
*----
*  Analyse and track device
*----
Geometry Volumes Tracks  := DevGeo         ::
  <<Device>> <<PrtLev>>                    ;
Geometry := DELETE: Geometry               ;
Backfile := BACKUP: Volumes                ;
*----
*  Import ASCII library
*----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*----
*  Analyze device in "NO" location
*----
EVALUATE   DevLocation                     :=
          "NO"                             ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>> <<PrtLev>>    ;
Fluxes   := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                               ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Fluxes Macrolib Volumes    ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                            ;
Backfile := BACKUP: Backfile  FlNO                          ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO        ;
*----
*  Analyze device in "OUT" location
*----
EVALUATE   DevLocation                     :=
          "OUT"                            ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>>  <<PrtLev>>   ;
Fluxes   := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                               ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                     ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>   ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
```

```
  ;
FlOUT   := Fluxes                                                    ;
Backfile := BACKUP: Backfile  FlOUT                                  ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT               ;
*-----
*  Analyze device in "IN" location
*-----
EVALUATE   DevLocation                        :=
          "IN"                                ;
Macrolib := DevMac MACCELL MACSTRC            ::
  <<Device>> <<DevLocation>> <<PrtLev>>       ;
Fluxes  := DevEva Macrolib Volumes Tracks     ::
  <<PrtLev>>                                  ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                              ;
Edition := EDI: Edition Fluxes Macrolib Volumes                       ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>              ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                                     ;
Backfile := BACKUP: Backfile  FlIN                                   ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN                 ;
*-----
*  Export result files
*-----
adj3nat  := Compo                        ;
*-----
*  Remove remaining temporary files
*-----
Edition Backfile Compo := DELETE:  Edition Backfile Compo             ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                           ;
Volumes Tracks := DELETE: Volumes Tracks                             ;
*-----
* ADJ4
*-----
EVALUATE   Device   :=  "ADJ4" ;
*-----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*-----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                        ;
SEQ_BINARY Tracks                          ;
*-----
*  Analyse and track device
*-----
Geometry Volumes Tracks  := DevGeo         ::
  <<Device>> <<PrtLev>>                    ;
Geometry := DELETE: Geometry               ;
```

```
Backfile := BACKUP: Volumes                ;
*-----
*  Import ASCII library
*-----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*-----
*  Analyze device in "NO" location
*-----
EVALUATE   DevLocation                     :=
          "NO"                             ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>> <<PrtLev>>    ;
Fluxes   := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                               ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Fluxes Macrolib Volumes    ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO    := Fluxes                                       ;
Backfile := BACKUP: Backfile  FlNO                      ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO    ;
*-----
*  Analyze device in "OUT" location
*-----
EVALUATE   DevLocation                     :=
          "OUT"                            ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>>  <<PrtLev>>   ;
Fluxes   := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                               ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                   ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>   ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT    := Fluxes                                      ;
Backfile := BACKUP: Backfile  FlOUT                     ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT  ;
*-----
*  Analyze device in "IN" location
*-----
EVALUATE   DevLocation                     :=
          "IN"                             ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>> <<PrtLev>>    ;
Fluxes   := DevEva Macrolib Volumes Tracks ::
```

```
  <<PrtLev>>                                    ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                            ;
Backfile := BACKUP: Backfile  FlIN                          ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN        ;
*----
*  Export result files
*----
adj4nat  := Compo                    ;
*----
*  Remove remaining temporary files
*----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*----
* ADJ5
*----
EVALUATE   Device   :=  "ADJ5" ;
*----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                         ;
SEQ_BINARY Tracks                           ;
*----
*  Analyse and track device
*----
Geometry Volumes Tracks  := DevGeo          ::
  <<Device>> <<PrtLev>>                     ;
Geometry := DELETE: Geometry                ;
Backfile := BACKUP: Volumes                 ;
*----
*  Import ASCII library
*----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*----
*  Analyze device in "NO" location
*----
EVALUATE   DevLocation                       :=
          "NO"                               ;
Macrolib := DevMac MACCELL MACSTRC           ::
```

```
  <<Device>> <<DevLocation>> <<PrtLev>>    ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                              ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Fluxes Macrolib Volumes    ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlNO                        ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO      ;
*-----
*  Analyze device in "OUT" location
*-----
EVALUATE   DevLocation                    :=
          "OUT"                           ;
Macrolib := DevMac MACCELL MACSTRC        ::
  <<Device>> <<DevLocation>>  <<PrtLev>>  ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                              ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                   ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                         ;
Backfile := BACKUP: Backfile  FlOUT                       ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT    ;
*-----
*  Analyze device in "IN" location
*-----
EVALUATE   DevLocation                    :=
          "IN"                            ;
Macrolib := DevMac MACCELL MACSTRC        ::
  <<Device>> <<DevLocation>> <<PrtLev>>   ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                              ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                   ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlIN                        ;
```

```
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN     ;
*-----
*  Export result files
*-----
adj5nat  := Compo                    ;
*-----
*  Remove remaining temporary files
*-----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*-----
* ADJ6
*-----
EVALUATE   Device   :=  "ADJ6" ;
*-----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*-----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                  ;
SEQ_BINARY Tracks                    ;
*-----
*  Analyse and track device
*-----
Geometry Volumes Tracks  := DevGeo       ::
  <<Device>> <<PrtLev>>                  ;
Geometry := DELETE: Geometry             ;
Backfile := BACKUP: Volumes              ;
*-----
*  Import ASCII library
*-----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*-----
*  Analyze device in "NO" location
*-----
EVALUATE   DevLocation                    :=
          "NO"                            ;
Macrolib := DevMac MACCELL MACSTRC        ::
  <<Device>> <<DevLocation>> <<PrtLev>>   ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                              ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Fluxes Macrolib Volumes   ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
```

```
FlNO   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlNO                        ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO      ;
*-----
*  Analyze device in "OUT" location
*-----
EVALUATE   DevLocation                      :=
          "OUT"                             ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>>  <<PrtLev>>    ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                         ;
Backfile := BACKUP: Backfile  FlOUT                       ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT    ;
*-----
*  Analyze device in "IN" location
*-----
EVALUATE   DevLocation                      :=
          "IN"                              ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>> <<PrtLev>>     ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>> ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlIN                        ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN      ;
*-----
*  Export result files
*-----
adj6nat  := Compo                    ;
*-----
*  Remove remaining temporary files
*-----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*-----
```

```
*  Finish execution
*-----
END: ;
QUIT "LIST" .
```

Listing I.4 Fichier d'entrées GANlib avec variables libTPT pour la simulation de contrôleurs liquides avec DRAGON

```
*DECK ZcuDev.x2m
*-----
*  Name           : ZcuDev.x2m
*  Use            : Perform 3-D reactivity device analysis
*                   for Liquid zone controlllers
*  Date           : 2002/10/09
*                 : 2004/09/02 Modified to export CPO
*  Input          : Procedures  -->
*                   DevGeo      ->  generate 3-D reactivity
*                                   device geometry
*                   DevMac      ->  generate 3-D macrolib
*                                   for reactivity devices
*                   DevEva      ->  solve 3-D transport problem
*                                   for reactivity devices
*                   ASCII_FILES --> required by DevMac
*                   MacroC MacroS ->
*                                   General macrolib for cell
*  Output         : ASCII_FILES -->  zcu1nat,zcu2nat,zcu3nat
*-----
* Define procedures and modules
*-----
PROCEDURE  DevEva     DevGeo     DevMac  ;
MODULE     EDI:       CPO:
           DELETE:    BACKUP:    END:    ;
*-----
*  Define input and output data structures
*-----
STRING     RefLib                        ;
EVALUATE   RefLib                        :=
          "${Microlib}" ;
SEQ_ASCII zcu1nat zcu2nat zcu3nat
          MacroC MacroS
          :: FILE
          "${zcu1nat}"
          "${zcu2nat}"
          "${zcu3nat}"
          "${MacroC}"
          "${MacroS}" ;
*-----
* Define local data structures needed
*  Create all ZCU device type
* Initialize device type
* Initialize device location
*    Start with "NO" then consider
*   "OUT" and "IN" where
```

```
*    "NO" is for device and structure absent
*    "OUT" is for device OUT with structure present
*    "IN" is for device IN with structure present
*     Note: For LZC, IN means filled with water
*     and OUT means empty
*----
* zcu1nat
*----
STRING     Device   :=  "LZC1" ;
INTEGER    PrtLev    := 2                ;
STRING     DevLocation                   ;
STRING     NameRec                       ;
*----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                      ;
SEQ_BINARY Tracks                        ;
*----
*  Analyse and track device
*----
Geometry Volumes Tracks  := DevGeo       ::
  <<Device>> <<PrtLev>>                  ;
Geometry := DELETE: Geometry             ;
Backfile := BACKUP: Volumes              ;
*----
*  Import ASCII library
*----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*----
*  Analyze device in "NO" location
*----
EVALUATE   DevLocation                    :=
          "NO"                            ;
Macrolib := DevMac MACCELL MACSTRC        ::
  <<Device>> <<DevLocation>> <<PrtLev>>   ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                              ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Fluxes Macrolib Volumes   ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                           ;
Backfile := BACKUP: Backfile  FlNO                         ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO       ;
```

```
*----
*  Analyze device in "OUT" location
*----
EVALUATE   DevLocation                      :=
          "OUT"                             ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>>  <<PrtLev>>    ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                           ;
Backfile := BACKUP: Backfile  FlOUT                         ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT      ;
*----
*  Analyze device in "IN" location
*----
EVALUATE   DevLocation                      :=
          "IN"                              ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>> <<PrtLev>>     ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                            ;
Backfile := BACKUP: Backfile  FlIN                          ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN        ;
*----
*  Export result files
*----
zcu1nat  := Compo                       ;
*----
*  Remove remaining temporary files
*----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*----
* zcu2nat
*----
EVALUATE     Device   :=  "LZC2" ;
```

```
*----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                        ;
SEQ_BINARY Tracks                          ;
*----
*  Analyse and track device
*----
Geometry Volumes Tracks  := DevGeo         ::
  <<Device>> <<PrtLev>>                    ;
Geometry := DELETE: Geometry               ;
Backfile := BACKUP: Volumes                ;
*----
*  Import ASCII library
*----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*----
*  Analyze device in "NO" location
*----
EVALUATE   DevLocation                     :=
          "NO"                             ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>> <<PrtLev>>    ;
Fluxes  := DevEva Macrolib Volumes Tracks  ::
  <<PrtLev>>                               ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Fluxes Macrolib Volumes    ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                           ;
Backfile := BACKUP: Backfile  FlNO                         ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO       ;
*----
*  Analyze device in "OUT" location
*----
EVALUATE   DevLocation                     :=
          "OUT"                            ;
Macrolib := DevMac MACCELL MACSTRC         ::
  <<Device>> <<DevLocation>>  <<PrtLev>>   ;
Fluxes  := DevEva Macrolib Volumes Tracks  ::
  <<PrtLev>>                               ;
*----
*  Store results in editing file
*----
```

```
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                           ;
Backfile := BACKUP: Backfile  FlOUT                         ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT      ;
*----
*  Analyze device in "IN" location
*----
EVALUATE   DevLocation                      :=
          "IN"                              ;
Macrolib := DevMac MACCELL MACSTRC          ::
  <<Device>> <<DevLocation>> <<PrtLev>>     ;
Fluxes  := DevEva Macrolib Volumes Tracks   ::
  <<PrtLev>>                                ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                            ;
Backfile := BACKUP: Backfile  FlIN                          ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN        ;
*----
*  Export result files
*----
zcu2nat  := Compo                          ;
*----
*  Remove remaining temporary files
*----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*----
* zcu3nat
*----
EVALUATE     Device   :=  "LZC3" ;
*----
* Define local data structures needed
* for execution
*  Define and evaluate local variables
*----
XSM_FILE   Geometry Volumes Macrolib Edition
           Fluxes Compo
           MACCELL MACSTRC
           FlIN FlOUT FlNO
           Backfile                        ;
SEQ_BINARY Tracks                          ;
*----
```

```
*  Analyse and track device
*----
Geometry Volumes Tracks  := DevGeo       ::
  <<Device>> <<PrtLev>>                  ;
Geometry := DELETE: Geometry             ;
Backfile := BACKUP: Volumes              ;
*----
*  Import ASCII library
*----
MACCELL := MacroC ;
MACSTRC := MacroS ;
*----
*  Analyze device in "NO" location
*----
EVALUATE   DevLocation                   :=
          "NO"                           ;
Macrolib := DevMac MACCELL MACSTRC       ::
  <<Device>> <<DevLocation>> <<PrtLev>>  ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                             ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Fluxes Macrolib Volumes   ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlNO   := Fluxes                                          ;
Backfile := BACKUP: Backfile  FlNO                        ;
Macrolib Fluxes FlNO := DELETE: Macrolib Fluxes FlNO      ;
*----
*  Analyze device in "OUT" location
*----
EVALUATE   DevLocation                   :=
          "OUT"                          ;
Macrolib := DevMac MACCELL MACSTRC       ::
  <<Device>> <<DevLocation>>  <<PrtLev>> ;
Fluxes  := DevEva Macrolib Volumes Tracks ::
  <<PrtLev>>                             ;
*----
*  Store results in editing file
*----
EVALUATE NameRec := Device DevLocation +                  ;
Edition := EDI: Edition Fluxes Macrolib Volumes           ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>  ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlOUT   := Fluxes                                         ;
Backfile := BACKUP: Backfile  FlOUT                       ;
Macrolib Fluxes FlOUT := DELETE: Macrolib Fluxes FlOUT    ;
*----
*  Analyze device in "IN" location
*----
```

```
EVALUATE   DevLocation                          :=
          "IN"                                  ;
Macrolib := DevMac MACCELL MACSTRC              ::
  <<Device>> <<DevLocation>> <<PrtLev>>         ;
Fluxes   := DevEva Macrolib Volumes Tracks      ::
  <<PrtLev>>                                    ;
*-----
*  Store results in editing file
*-----
EVALUATE NameRec := Device DevLocation +                    ;
Edition := EDI: Edition Fluxes Macrolib Volumes             ::
  COND ${CondEnergyArray} MERGE COMP SAVE ON <<NameRec>>    ;
Compo := CPO: Compo Edition  ::
  STEP <<NameRec>>  NAME <<NameRec>>
  ;
FlIN   := Fluxes                                            ;
Backfile := BACKUP: Backfile  FlIN                          ;
Macrolib Fluxes FlIN := DELETE: Macrolib Fluxes FlIN        ;
*-----
*  Export result files
*-----
zcu3nat  := Compo                       ;
*-----
*  Remove remaining temporary files
*-----
Edition Backfile Compo := DELETE:  Edition Backfile Compo    ;
MACCELL MACSTRC := DELETE: MACCELL MACSTRC                   ;
Volumes Tracks := DELETE: Volumes Tracks                     ;
*-----
*  Finish execution
*-----
END: ;
QUIT "LIST" .
```

Listing I.5 Fichier d'entrées GANlib avec variables libTPT pour la simulation d'un réacteur avec DONJON

```
!****************************************************************
!* Calcul DONJON a l'equilibre du rechargement a deux (2) zones **
!*                                                              **
!* Determination du burnup moyen de sortie pour un coeur        **
!* avec des grappes quelconques provenant de ${fnat}            **
!*                                                              **
!* Options:                                                     **
!*   1- Rapport B1/B2 (B1 zone centrale, B2 zone peripherique)  **
!*   2- Position des BC                                         **
!****************************************************************
!* B. Dionne (03/04/2001)                                       **
!* E. Varin (23/02/2004)                                        **
!* P. Rheaume (2004/09/02) (pour D2G2Server)                    **
!****************************************************************
 PROCEDURE  Pburncal Pcalflu Pinidet ;
```

```
 MODULE        END: CRE: FLXAXC: REFUEL: INIMAC:
               POWER: FLUNRM: DETUTL: DETECT: PLOT: ;

 SEQ_ASCII fnat van map flux plot :: FILE
           "${fnat}"
           "${van}"
           "${map}"
           "${flux}"
           "plot" ;

 LINKED_LIST  NFUEL G23D2 INDEX MACRO DEVICE PROCEDE RESEAU TAB  ;
 LINKED_LIST  MACRO2 MACRES FLUX TRACK MACOLD
              DETEC VAN ;

!* Declaration des variables
!**************************
 REAL Rapport BCinout ;
 INTEGER Maxreg  ;
 REAL B1 B2      ;
 STRING        Refl ;
 REAL          Xfacc Xfacp ;
 REAL          Keff ;
 REAL          Kref ;
 REAL          ThPow ;
 REAL          FisPow ;
 REAL          Fsth ;
 INTEGER       iter := 1 ;
 REAL          Precf := ${FluxPrecision} ;

!* Parametres de controle de la simulation
!****************************************
 EVALUATE Rapport := ${Rapport} ; ! Rapport B1 / B2
 EVALUATE BCinout := ${BCinout} ; ! BC entrees = 1.0, BC sorties = 0.0
 EVALUATE Maxreg  := ${Maxreg} ; ! nombre de regions dans la geometrie
 EVALUATE ThPow   := ${ThermalPower} ;
 EVALUATE Fsth    := ${ThermaltoFissionRatio} ;
 EVALUATE FisPow  := ThPow Fsth / ;
 EVALUATE Kref := ${Kref} ;

!* Liaison avec le fichier de composition
!****************************************
 NFUEL := fnat ;

!* Appel de la procedure principale
!*********************************
G23D2 INDEX MACRO DEVICE PROCEDE RESEAU TAB
           Pburncal NFUEL ::
    <<Rapport>> <<BCinout>> <<Maxreg>> <<Precf>>
    <<FisPow>> <<Kref>> ;

 EVALUATE  Refl  := "${Refl}" ;
 EVALUATE  Xfacp := ${Xfacp} ;
 EVALUATE  Xfacc := ${Xfacc} ;

!*
!* Construction des detecteurs
```

```
!*******************************
 DETEC := Pinidet :: 1 ;

!* Prorietes en fonction du burnup actuel
!*************************************
 MACRES := CRE: TAB RESEAU ::
     EDIT 0
     READ
      TABLE TAB  MIX  1 'NATURAL'
                 BURNUP TAVGC ENDMIX ;

 MACRO2 := INIMAC: INDEX MACRO MACRES ;

!* Calcul du flux pour la repartition convergee du burnup
!*********************************************************
 FLUX MACOLD TRACK := Pcalflu MACRO2 DEVICE G23D2 ::
    <<iter>>  <<Maxreg>> <<Refl>> <<Xfacc>> <<Xfacp>> <<Precf>> ;

!* EXPORT FLUX FILE
 flux := FLUX ;

 RESEAU := FLXAXC: RESEAU FLUX TRACK INDEX MACRES ::   AXIAL COMP ;


!* Calcul de la repartition de la puissance dans le reacteur
!*********************************************************
 RESEAU := POWER: RESEAU MACRES :: EDIT 2 POWER ${ThermalPower} FSTH ${
  ThermaltoFissionRatio} ;

!* EXPORT MAP FILE
 map := RESEAU ;

!* EXPORT GNUPLOT AND VU
!* plot := PLOT: RESEAU :: GNUPLOT ZERO CHANNEL POWER-BUN ;
!* plot := PLOT: RESEAU :: VU TOTAL POWER-BUN ;


!* Calcul de la lecture aux 102 detecteurs vanadium
!*********************************************************
 FLUX := FLUNRM: FLUX MACOLD TRACK ::   EDIT 5 POWER ${
  NormalisationFluxPower}E+6 REF ${RefNormFlux} ;

 DETEC := DETECT: DETEC FLUX TRACK G23D2 ::
    EDIT 0   REF 0 TIME ${DetectTime} SIMEX ;

!* EXPORT VAN FILE
 van := DETEC ;

 END: ;

QUIT .
```

ANNEXE II

CODES SOURCES XML

Listing II.1 DTD du Projet D2G2

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT D2G2XML (UserInfo, ClientMode)>
<!ELEMENT UserInfo EMPTY>
<!ATTLIST UserInfo
Username CDATA #REQUIRED
Password CDATA #FIXED "Dummy"
>
<!ELEMENT ClientMode (Authentication | Simulation | RequestData |
 KillSimulation)>
<!ELEMENT Authentication EMPTY>
<!ATTLIST Authentication
>
<!ELEMENT Simulation (Reactor | CpoCell | CellDevices | AdjDevices |
 ZcuDevices)>
<!ATTLIST Simulation
SimulationUID CDATA "NewSimulation"
>

<!-- ################################## -->
<!-- # REACTOR SIMULATION PARAMETERS # -->
<!-- ################################## -->

<!ELEMENT Reactor (CANDU.x2m,Pburncal.c2m,Pcalflu.c2m,PcritAx.c2m, PdevT
 .c2m, Pflax.c2m, PgeoG2.c2m, Pinidet.c2m, Pinires.c2m, Pinproc.c2m,
 Pmacfix.c2m)>
<!ATTLIST Reactor
CpoCellUID CDATA "1"
AdjDevicesUID CDATA "1"
ZcuDevicesUID CDATA "1"
Comments CDATA "Reactor Simulation"
>
<!ELEMENT CANDU.x2m EMPTY>
<!-- ThermalPower: Total thermal power of the reactor in (MW) -->
<!-- ThermaltoFissionRatio: Thermal to fission power ratio. By default
 this value is 1.0. -->
<!-- NormalisationFluxPower: Power used to normalize the flux (MW) -->
<!-- RefNormFlux: =0 for normalization of fluxes to power and =1 power
 calculation using a steady normalization factor. -->
<!-- Kref: Reference multiplication factor, usually set to 1 -->
<!-- DetectTime: DetectTime is time step between to call of DETECT
 module (s) -->
<!-- Rapport: Ratio B1 / B2 B1 = average exit burnup for zone # 1 B2 =
 average exit burnup for zone # 2  (default 20% more in B1)-->
```

```
<!-- BCinout : Adjuster control bar in = 1.0, out = 0.0 -->
<!-- Maxreg: Maximum dimensions of the problem to be considered. The
 default value is set to the number of regions previously computed by
 the GEOD: module but this value is insufficient if symmetries or mesh-
 splitting are specified. -->
<!-- FluxPrecision: The FLUD module is used to compute the flux solution
  to an eigenvalue problem corresponding to a set of system matrices and
  we set the precision on the solution -->
<!-- Refl: REFL-SIGF key word used to add explicitly device incremental
 fission cross sections to non-fissile properties. This option is
 available to compare with other reactor simulation codes in which
 incremental fission cross sections are put in the reflector. NOSIGF key
  word used to set that no fission are added to non-fissile properties.
 It is the default option. -->
<!-- Xfacp: Device incremental cross section correction. For
 perturbation, it is usually set to 1. By default, xfacp is equal to
 xfacc. -->
<!-- Xfacc : Lattice number. For DRAGON code, xfacc must be equal to 2.
 and for MULTICELL code, equal to 1. The default value is 1. -->
<!ATTLIST CANDU.x2m
ThermalPower CDATA "2061.4"
ThermaltoFissionRatio CDATA "0.946"
NormalisationFluxPower CDATA "2061.4"
RefNormFlux CDATA "0"
Kref CDATA "1.0"
DetectTime CDATA "0.25"
Rapport CDATA "1.25"
BCinout CDATA "1.0"
Maxreg CDATA "27700"
FluxPrecision CDATA  "1.E-6"
Refl CDATA "REFL-SIGF"
Xfacp CDATA "1."
Xfacc CDATA "2."
>
<!ELEMENT Pburncal.c2m EMPTY>
<!-- DBurn: Give to INTRPL keyword used to order a linear interpolation
 of fuel properties with respect to burnup. delta fuel depletion
 variable increment used in the property interpolation. If fuel burnup
 is the interpolation parameter, this value will be in MWd/t ; If fuel
 irradiation is the interpolation parameter, this value will be in
 neutron/kb. -->
<!-- Timbc: Maximum time for complete insertion or withdrawal of a rod (
 s). -->
<!-- B1min: Min exit burnup for zone # 1 to compute root -->
<!-- B1max: Max exit burnup for zone # 1 to compute root -->
<!-- DevicesInitPosition: Devices initial position (%) -->
<!-- BrentRootPrec: Root precision when finding with Brent method -->
<!-- BrentItMax: Maximum number of iteration with Brent method -->
<!ATTLIST Pburncal.c2m
Dburn CDATA "250.0"
Timbc CDATA "60.0"
fmean CDATA "0.5"
B1Min CDATA "4000.0"
B1Max CDATA "12000.0"
DevicesInitPosition CDATA "0.5"
BrentRootPrec CDATA "5.0E-4"
```

```
BrentItMax CDATA "10"
>
<!ELEMENT Pcalflu.c2m EMPTY>
<!ATTLIST Pcalflu.c2m
>
<!ELEMENT PcritAx.c2m EMPTY>
<!-- AxialFormFluxError: Maximum error on axial form flux -->
<!-- B1 BundleShifts: Refueling schemes for combustion zone #1 -->
<!-- B2 BundleShifts: Refueling schemes for combustion zone #2 -->
<!ATTLIST PcritAx.c2m
AxialFormFluxError CDATA "0.001"
B1BundleShifts CDATA "8"
B2BundleShifts CDATA "8"
>
<!ELEMENT PdevT.c2m EMPTY>
<!-- A lot of devices settings, many parameters can be modified in this
 file. -->
<!-- LatticePitch: Reactor lattice pitch -->
<!ATTLIST PdevT.c2m
LatticePitch CDATA "28.5750"
>
<!ELEMENT Pflax.c2m EMPTY>
<!ATTLIST Pflax.c2m
>
<!ELEMENT PgeoG2.c2m EMPTY>
<!-- We use two combustion zones. Geo modifications influence many
 parameters in others files... -->
<!ATTLIST PgeoG2.c2m
>
<!ELEMENT Pinidet.c2m EMPTY>
<!-- A lot of vanadium detector settings, many parameters can be
 modified in this file... -->
<!ATTLIST Pinidet.c2m
>
<!ELEMENT Pinires.c2m EMPTY>
<!-- Fuel map description for Gentilly2 reactor. We use two combustion
 zones. If geo modifications are made, many parameters will be modified
 in this file... -->
<!ATTLIST Pinires.c2m
>
<!ELEMENT Pinproc.c2m EMPTY>
<!-- yful1 and yful3: Values for 100% of the BL in DONJON length -->
<!ATTLIST Pinproc.c2m
yful1 CDATA "170.1827"
yful3 CDATA "53.8898"
>
<!ELEMENT Pmacfix.c2m EMPTY>
<!ATTLIST Pmacfix.c2m
>


<!-- ################################## -->
<!-- # CPOCELL SIMULATION PARAMETERS # -->
<!-- ################################## -->
```

```
<!ELEMENT CpoCell (Cpo.x2m,Evolution.c2m, GeoCANDU6.c2m, GeoCANFLEX.c2m,
 LibE5CNG.c2m, LibE5NAT.c2m, LibE5RUMOX.c2m)>
<!ATTLIST CpoCell
Comments CDATA "CpoCell␣Simulation"
>
<!ELEMENT Cpo.x2m EMPTY>
<!-- BundleType: Bundle type, could be CANDU6-> CANDU-6 with 37 elements
 , CANFLEX-> CANFLEX with 43 elements -->
<!-- TubePresType: Pressure type, could be G2-> Tube CANDU-6 Gentilly
 -2, CNG Tube CANDU-NGor CNG -->
<!-- FuelType: Fuel type, could be NAT-> Natural Uranium , RUMOX->
 Recycled Uranium from REP, CNG->CANDU-NG -->
<!-- Leakage: Leakage model, could be AUCUNE=no leakage, B0= B0 homogene
  B1= B1 homogene -->
<!-- LatticePitch: Reactor Lattice pitch -->
<!-- TempComb: Fuel Temperature (K) -->
<!-- PurtCalo: Coolant purity (% of D20) -->
<!-- TempCalo: Coolant Temperature (K) -->
<!-- TitreCalo: Coolant titre (%) -->
<!-- TempModer: Moderator Temperature (K) -->
<!-- PurtModer: Moderator purity (%) -->
<!-- Power: Specific fuel bundle power (kw/kg) -->
<!-- Burnup: Fuel Exit burnup (GWj/TU) -->
<!ATTLIST Cpo.x2m
BundleType CDATA "CANDU6"
TubePresType CDATA "G2"
FuelType CDATA "NAT"
Leakage CDATA "B1"
LatticePitch CDATA "28.575"
TempComb CDATA "941.29"
PurtCalo CDATA "99.30"
TempCalo CDATA "560.66"
TitreCalo CDATA "0.0"
TempModer CDATA "345.66"
PurtModer CDATA "99.92"
Power CDATA "31.9713"
Burnup CDATA "70.0"
>
<!ELEMENT Evolution.c2m EMPTY>
<!-- A lot of fuel evolution settings, many parameters can be modified
 in this file... -->
<!ATTLIST Evolution.c2m
>
<!ELEMENT GeoCANDU6.c2m EMPTY>
<!-- A lot of 37 elements fuel bundle geometry settings, many parameters
  can be modified in this file. -->
<!ATTLIST GeoCANDU6.c2m
>
<!ELEMENT GeoCANFLEX.c2m EMPTY>
<!-- A lot of 43 elements fuel bundle geometry settings, many parameters
  can be modified in this file. -->
<!ATTLIST GeoCANFLEX.c2m
>
<!ELEMENT LibE5CNG.c2m EMPTY>
<!-- A lot of microlib access stuff, could be change if microlib access
 parameters change -->
```

```
<!ATTLIST LibE5CNG.c2m
>
<!ELEMENT LibE5NAT.c2m EMPTY>
<!-- A lot of microlib access stuff, could be change if microlib access
 parameters change -->
<!ATTLIST LibE5NAT.c2m
>
<!ELEMENT LibE5RUMOX.c2m EMPTY>
<!-- A lot of microlib access stuff, could be change if microlib access
 parameters change -->
<!ATTLIST LibE5RUMOX.c2m
>


<!-- ###################################### -->
<!-- # CELLDEVICES SIMULATION PARAMETERS # -->
<!-- ###################################### -->

<!ELEMENT CellDevices (CellDev.x2m, CellBibG2.c2m, CellFlxG2.c2m,
 CellGeoG2.c2m,CellHomG2.c2m, MacStrcG2.c2m)>
<!ATTLIST CellDevices
Comments CDATA "CellDevices␣Simulation"
>
<!ELEMENT CellDev.x2m EMPTY>
<!-- Microlib to use could be WL-> Winfrith WIMS-AECL, E5-> ENDF/B-V
 WIMS-AECL, E6-> ENDF/B-VI WIMS-AECL -->
<!-- TempComb: Fuel Temperature (K) -->
<!-- EnriComb: Fuel enrichment (%) -->
<!-- DensComb: Fuel density (g/cc) -->
<!-- TempCalo: Coolant Temperature (K) -->
<!-- PurtCalo: Coolant purity (% of D20) -->
<!-- TempMod: Moderator Temperature (K) -->
<!-- PurtMod: Moderator purity (%) -->
<!-- Power: Specific fuel bundle power (kw/kg) -->
<!-- Timei: Initial fuel evolution time (day) -->
<!-- Timef: Initial fuel evolution time (day) -->
<!ATTLIST CellDev.x2m
Microlib CDATA "E5"
TempComb CDATA "941.29"
EnriComb CDATA "0.7114"
DensComb CDATA "10.4375"
TempCalo CDATA "560.66"
PurtCalo CDATA "99.30"
TempMod CDATA "345.662"
PurtMod CDATA "99.922"
BoreMod CDATA "0.0"
Power CDATA "31.97132"
Timei CDATA "0.02"
Timef CDATA "130.0"
>
<!ELEMENT CellBibG2.c2m EMPTY>
<!-- A lot of microlib access stuff, could be change if microlib access
 parameters change -->
<!ATTLIST CellBibG2.c2m
>
<!ELEMENT CellFlxG2.c2m EMPTY>
```

```
<!-- A lot of fuel evoution settings, many parameters can be modified in
  this file... -->
<!ATTLIST CellFlxG2.c2m
>
<!ELEMENT CellGeoG2.c2m EMPTY>
<!-- A lot of cell geometry settings, many parameters can be modified in
  this file... -->
<!ATTLIST CellGeoG2.c2m
>
<!ELEMENT CellHomG2.c2m EMPTY>
<!-- HomoRegMerge: Cell homogenisation region merge -->
<!-- EXCELT parameters could be change TRAK TISO ... -->
<!ATTLIST CellHomG2.c2m
HomoRegMerge CDATA "
 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6 6 6 6 2 2 2 1 1 1 1 1 1 1 1 1 1 1"
>
<!ELEMENT MacStrcG2.c2m EMPTY>
<!-- A lot of reactor structure settings, many parameters can be
 modified in this file... -->
<!ATTLIST MacStrcG2.c2m
>


<!-- ######################################### -->
<!-- # ADJUSTER DEVICES SIMULATION PARAMETERS # -->
<!-- ######################################### -->

<!ELEMENT AdjDevices (AdjDev.x2m, AdjDevEva.c2m, AdjDevGeo.c2m,
 AdjDevMac.c2m)>
<!ATTLIST AdjDevices
CellDevicesUID CDATA "1"
Comments CDATA "AdjDevices Simulation"
>
<!ELEMENT AdjDev.x2m EMPTY>
<!-- Microlib to use could be WL-> Winfrith WIMS-AECL, E5-> ENDF/B-V
 WIMS-AECL, E6-> ENDF/B-VI WIMS-AECL -->
<!-- CondEnergyArray: Array of increasing energy limits that will be
 associated with each condensed groups (eV) -->
<!ATTLIST AdjDev.x2m
Microlib CDATA "E5"
CondEnergyArray CDATA "0.625"
>
<!ELEMENT AdjDevEva.c2m EMPTY>
<!-- Leakage model, could be B0= B0 homogene, B1= B1 homogene -->
<!ATTLIST AdjDevEva.c2m
Leakage CDATA "B1"
>
<!ELEMENT AdjDevGeo.c2m EMPTY>
<!-- XYLatticePitch: XY Reactor lattice pitch (cm) -->
<!-- ZLatticePitch: Z Reactor lattice pitch (cm) -->
<!-- CellSize: Size of device cell (cm) -->
<!-- A lot of ADJ|LZU geometry settings, many parameters can be modified
  in this file... -->
<!ATTLIST AdjDevGeo.c2m
XYLatticePitch CDATA "28.5750"
ZLatticePitch CDATA "49.53"
```

```
CellSize CDATA "7.00"
>
<!ELEMENT AdjDevMac.c2m EMPTY>
<!-- A lot of devices settings, many parameters can be modified in this
 file...-->
<!ATTLIST AdjDevMac.c2m
>


<!-- ##################################### -->
<!-- # ZCU DEVICES SIMULATION PARAMETERS # -->
<!-- ##################################### -->

<!ELEMENT ZcuDevices (ZcuDev.x2m, ZcuDevEva.c2m, ZcuDevGeo.c2m,
 ZcuDevMac.c2m)>
<!ATTLIST ZcuDevices
CellDevicesUID CDATA "1"
Comments CDATA "ZcuDevices Simulation"
>
<!ELEMENT ZcuDev.x2m EMPTY>
<!-- Microlib to use could be WL-> Winfrith WIMS-AECL, E5-> ENDF/B-V
 WIMS-AECL, E6-> ENDF/B-VI WIMS-AECL -->
<!-- CondEnergyArray: Array of increasing energy limits that will be
 associated with each condensed groups (eV) -->
<!ATTLIST ZcuDev.x2m
Microlib CDATA "E5"
CondEnergyArray CDATA "0.625"
>
<!ELEMENT ZcuDevEva.c2m EMPTY>
<!-- Leakage model, could be B0= B0 homogene, B1= B1 homogene -->
<!ATTLIST ZcuDevEva.c2m
Leakage CDATA "B1"
>
<!ELEMENT ZcuDevGeo.c2m EMPTY>
<!-- XYLatticePitch: XY Reactor lattice pitch (cm)-->
<!-- ZLatticePitch: Z Reactor lattice pitch (cm)-->
<!-- CellSize: Size of device cell (cm)-->
<!-- A lot of ADJ|LZU geometry settings, many parameters can be modified
  in this file... -->
<!ATTLIST ZcuDevGeo.c2m
XYLatticePitch CDATA "28.5750"
ZLatticePitch CDATA "49.53"
CellSize CDATA "7.00"
>
<!ELEMENT ZcuDevMac.c2m EMPTY>
<!-- A lot of devices settings, many parameters can be modified in this
 file...
-->
<!ATTLIST ZcuDevMac.c2m
>


<!-- ###################### -->
<!-- # REQUEST DATA MODE # -->
<!-- ###################### -->
```

```
<!ELEMENT RequestData (ActiveServersList | ActiveSimList| ReactorSimList
  | ReactorSimInput | ReactorSimOutput | CpoCellSimList |
 CpoCellSimInput | CpoCellSimOutput | CellDevicesSimList |
 CellDevicesSimInput | CellDevicesSimOutput | AdjDevicesSimList |
 AdjDevicesSimInput |
AdjDevicesSimOutput | ZcuDevicesSimList | ZcuDevicesSimInput |
 ZcuDevicesSimOutput | VanadiumFlux | ReactorMap |ReactorFlux |
 CpoCellFlux)>
<!ELEMENT ActiveServersList EMPTY>
<!ATTLIST ActiveServersList
>
<!ELEMENT ActiveSimList EMPTY>
<!ATTLIST ActiveSimList
>
<!ELEMENT ReactorSimList EMPTY>
<!ATTLIST ReactorSimList
MinDate CDATA "1970-01-01"
MaxDate CDATA "2070-01-01"
>
<!ELEMENT ReactorSimInput EMPTY>
<!ATTLIST ReactorSimInput
ReactorUID CDATA "1"
>
<!ELEMENT ReactorSimOutput EMPTY>
<!ATTLIST ReactorSimOutput
ReactorUID CDATA "1"
>
<!ELEMENT CpoCellSimList EMPTY>
<!ATTLIST CpoCellSimList
MinDate CDATA "1970-01-01"
MaxDate CDATA "2070-01-01"
>
<!ELEMENT CpoCellSimInput EMPTY>
<!ATTLIST CpoCellSimInput
CpoCellUID CDATA "1"
>
<!ELEMENT CpoCellSimOutput EMPTY>
<!ATTLIST CpoCellSimOutput
CpoCellUID CDATA "1"
>
<!ELEMENT CellDevicesSimList EMPTY>
<!ATTLIST CellDevicesSimList
MinDate CDATA "1970-01-01"
MaxDate CDATA "2070-01-01"
>
<!ELEMENT CellDevicesSimInput EMPTY>
<!ATTLIST CellDevicesSimInput
CellDevicesUID CDATA "1"
>
<!ELEMENT CellDevicesSimOutput EMPTY>
<!ATTLIST CellDevicesSimOutput
CellDevicesUID CDATA "1"
>
<!ELEMENT AdjDevicesSimList EMPTY>
<!ATTLIST AdjDevicesSimList
MinDate CDATA "1970-01-01"
```

```
MaxDate CDATA "2070-01-01"
>
<!ELEMENT AdjDevicesSimInput EMPTY>
<!ATTLIST AdjDevicesSimInput
AdjDevicesUID CDATA "1"
>
<!ELEMENT AdjDevicesSimOutput EMPTY>
<!ATTLIST AdjDevicesSimOutput
AdjDevicesUID CDATA "1"
>
<!ELEMENT ZcuDevicesSimList EMPTY>
<!ATTLIST ZcuDevicesSimList
MinDate CDATA "1970-01-01"
MaxDate CDATA "2070-01-01"
>
<!ELEMENT ZcuDevicesSimInput EMPTY>
<!ATTLIST ZcuDevicesSimInput
ZcuDevicesUID CDATA "1"
>
<!ELEMENT ZcuDevicesSimOutput EMPTY>
<!ATTLIST ZcuDevicesSimOutput
ZcuDevicesUID CDATA "1"
>
<!ELEMENT VanadiumFlux EMPTY>
<!ATTLIST VanadiumFlux
ReactorUID CDATA "1"
Format CDATA "GAN"
>
<!ELEMENT ReactorMap EMPTY>
<!ATTLIST ReactorMap
ReactorUID CDATA "1"
Format CDATA "GAN"
>
<!ELEMENT ReactorFlux EMPTY>
<!ATTLIST ReactorFlux
ReactorUID CDATA "1"
Format CDATA "GAN"
>
<!ELEMENT CpoCellFlux EMPTY>
<!ATTLIST CpoCellFlux
CpoCellUID CDATA "1"
Format CDATA "GAN"
>
<!ELEMENT KillSimulation EMPTY>
<!ATTLIST KillSimulation
SimulationUID CDATA "1"
>
<!-- ################ -->
<!-- FIN DTD D2G2_XML -->
<!-- ################ -->
```

Listing II.2 Fichier de simulation de réacteur

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <Simulation SimulationUID="NewSimulation">
   <Reactor CpoCellUID="1" AdjDevicesUID="1" ZcuDevicesUID="1">
    <CANDU.x2m ThermalPower="2061.4" ThermaltoFissionRatio="0.946"
     NormalisationFluxPower="2061.4" RefNormFlux="0" Kref="1.0"
     DetectTime="0.25" Rapport="1.25" BCinout="1.0" Maxreg="27700"
     FluxPrecision="1.E-6" Refl="REFL-SIGF" Xfacp="1." Xfacc="2."></
     CANDU.x2m>
    <Pburncal.c2m Dburn="250.0" Timbc="60.0" fmean="0.5" B1Min="4000.0"
     B1Max="12000.0" DevicesInitPosition="0.5" BrentRootPrec="5.0E-4"
     BrentItMax="10"></Pburncal.c2m>
    <Pcalflu.c2m></Pcalflu.c2m>
    <PcritAx.c2m AxialFormFluxError="0.001" B1BundleShifts="8"
     B2BundleShifts="8"></PcritAx.c2m>
    <PdevT.c2m LatticePitch="28.5750"></PdevT.c2m>
    <Pflax.c2m></Pflax.c2m>
    <PgeoG2.c2m></PgeoG2.c2m>
    <Pinidet.c2m></Pinidet.c2m>
    <Pinires.c2m></Pinires.c2m>
    <Pinproc.c2m yful1="170.1827" yful3="53.8898"></Pinproc.c2m>
    <Pmacfix.c2m></Pmacfix.c2m>
   </Reactor>
  </Simulation>
 </ClientMode>
</D2G2XML>
```

Listing II.3 Requête de résultats de simulation réacteur

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <RequestData>
   <ReactorSimOutput ReactorUID="1"></ReactorSimOutput>
  </RequestData>
 </ClientMode>
</D2G2XML>
```

Listing II.4 Requête de résultats du flux réacteur

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
```

```
 <ClientMode>
  <RequestData>
   <ReactorFlux ReactorUID="1" Format="GAN"></ReactorFlux>
  </RequestData>
 </ClientMode>
</D2G2XML>
```

Listing II.5 Fichier de simulation de cellule

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <Simulation SimulationUID="NewSimulation">
   <CpoCell>
    <Cpo.x2m BundleType="CANDU6" TubePresType="G2" FuelType="NAT"
     Leakage="B1" LatticePitch="28.575" TempComb="941.29" PurtCalo="
     99.30" TempCalo="560.66" TitreCalo="0.0" TempModer="345.66"
     PurtModer="99.92" Power="31.9713" Burnup="70.0" CondEnergyArray="
     0.625"></Cpo.x2m>
    <Evolution.c2m></Evolution.c2m>
    <GeoCANDU6.c2m></GeoCANDU6.c2m>
    <GeoCANFLEX.c2m></GeoCANFLEX.c2m>
    <LibE5CNG.c2m></LibE5CNG.c2m>
    <LibE5NAT.c2m></LibE5NAT.c2m>
    <LibE5RUMOX.c2m></LibE5RUMOX.c2m>
   </CpoCell>
  </Simulation>
 </ClientMode>
</D2G2XML>
```

Listing II.6 Requête de résultats de flux cellule

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
<UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <RequestData>
   <CpoCellFlux CpoCellUID="1" Format="GAN"></CpoCellFlux>
   </RequestData>
 </ClientMode>
</D2G2XML>
```

Listing II.7 Requête de résultats de simulation cellule

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
```

```
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <RequestData>
   <CpoCellSimOutput CpoCellUID="1"></CpoCellSimOutput>
  </RequestData>
 </ClientMode>
</D2G2XML>
```

Listing II.8 Fichier de simulation de macrolib et des structures de supercellule

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <Simulation SimulationUID="NewSimulation">
   <CellDevices>
    <CellDev.x2m Microlib="E5" TempComb= "941.29" EnriComb="0.7114"
     DensComb="10.4375" TempCalo="560.66" PurtCalo="99.30" TempMod="
     345.66" PurtMod="99.92" BoreMod="0.0" Power="31.9713" Timei="0.0"
     Timef="130.0"></CellDev.x2m>
    <CellBibG2.c2m></CellBibG2.c2m>
    <CellFlxG2.c2m></CellFlxG2.c2m>
    <CellGeoG2.c2m></CellGeoG2.c2m>
    <CellHomG2.c2m HomoRegMerge="
     3␣3␣3␣3␣4␣4␣4␣4␣4␣5␣5␣5␣5␣5␣6␣6␣6␣6␣6␣2␣2␣2␣1␣1␣1␣1␣1␣1␣1␣1␣1␣1">
     CellHomG2.c2m>
    <MacStrcG2.c2m></MacStrcG2.c2m>
   </CellDevices>
  </Simulation>
 </ClientMode>
</D2G2XML>
```

Listing II.9 Fichier de simulation de barres de contrôle

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <Simulation SimulationUID="NewSimulation">
   <AdjDevices CellDevicesUID="1">
    <AdjDev.x2m Microlib="E5" CondEnergyArray="0.625"></AdjDev.x2m>
    <AdjDevEva.c2m Leakage="B1"></AdjDevEva.c2m>
    <AdjDevGeo.c2m XYLatticePitch="28.5750" ZLatticePitch="49.53"
     CellSize="7.00"></AdjDevGeo.c2m>
    <AdjDevMac.c2m></AdjDevMac.c2m>
   </AdjDevices>
  </Simulation>
 </ClientMode>
</D2G2XML>
```

Listing II.10 Fichier de simulation de contrôleurs liquides

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE D2G2XML SYSTEM "D2G2_XML.dtd">
<D2G2XML>
 <UserInfo Username="Rheaume"></UserInfo>
 <ClientMode>
  <Simulation SimulationUID="NewSimulation">
   <ZcuDevices CellDevicesUID="1">
    <ZcuDev.x2m Microlib="E5" CondEnergyArray="0.625"></ZcuDev.x2m>
    <ZcuDevEva.c2m Leakage="B1"></ZcuDevEva.c2m>
    <ZcuDevGeo.c2m XYLatticePitch="28.5750" ZLatticePitch="49.53"
     CellSize="7.00"></ZcuDevGeo.c2m>
    <ZcuDevMac.c2m></ZcuDevMac.c2m>
   </ZcuDevices>
  </Simulation>
 </ClientMode>
</D2G2XML>
```

ANNEXE III

CODE SOURCE MYSQL

Listing III.1 Structure de la base de données D2G2

```
--
-- Structure de la table 'ActiveServersList'
--
DROP TABLE IF EXISTS 'ActiveServersLists';
CREATE TABLE IF NOT EXISTS 'ActiveServersList' (
  'ServerAddr' varchar(250) NOT NULL default '',
  'ServerPort' int(11) NOT NULL default '20001',
  'NbRunningSimulations' int(11) NOT NULL default '0',
  UNIQUE KEY 'ServerAddr' ('ServerAddr')
) TYPE=MyISAM;
-- --------------------------------------------------------
--
-- Structure de la table 'ActiveSimList'
--

DROP TABLE IF EXISTS 'ActiveSimList';
CREATE TABLE IF NOT EXISTS 'ActiveSimList' (
  'SimulationUID' bigint(20) NOT NULL auto_increment,
  'Input' varchar(250) NOT NULL default '',
  'ServerAddr' varchar(250) NOT NULL default '',
  'Username' varchar(30) default NULL,
  'Comments' text,
  'Date' datetime default NULL,
  'Status' varchar(15) default NULL,
  PRIMARY KEY  ('SimulationUID')
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;

-- --------------------------------------------------------
--
-- Structure de la table 'AdjDevicesSimList'
--
DROP TABLE IF EXISTS 'AdjDevicesSimList';
CREATE TABLE IF NOT EXISTS 'AdjDevicesSimList' (
  'AdjDevicesUID' bigint(20) NOT NULL auto_increment,
  'Username' varchar(30) NOT NULL default 'Alice',
  'Comments' text,
  'ClientAddr' varchar(250) default NULL,
  'ServerAddr' varchar(250) default NULL,
  'Date' datetime NOT NULL default '0000-00-00 00:00:00',
  'Status' varchar(15) NOT NULL default 'Running',
  'DurationMinutes' bigint(20) default NULL,
  'CellDevicesUID' bigint(20) NOT NULL default '0',
  'InputFilePath' varchar(250) default NULL,
  'OutputFilePath' varchar(250) default NULL,
```

```
  `Adj1natFilePath` varchar(250) default NULL,
  `Adj2natFilePath` varchar(250) default NULL,
  `Adj3natFilePath` varchar(250) default NULL,
  `Adj4natFilePath` varchar(250) default NULL,
  `Adj5natFilePath` varchar(250) default NULL,
  `Adj6natFilePath` varchar(250) default NULL,
  PRIMARY KEY  (`AdjDevicesUID`)
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;
-- --------------------------------------------------------
--
-- Structure de la table `CellDevicesSimList`
--
DROP TABLE IF EXISTS `CellDevicesSimList`;
CREATE TABLE IF NOT EXISTS `CellDevicesSimList` (
  `CellDevicesUID` bigint(20) NOT NULL auto_increment,
  `Username` varchar(30) NOT NULL default 'Alice',
  `Comments` text,
  `ClientAddr` varchar(250) default NULL,
  `ServerAddr` varchar(250) default NULL,
  `Date` datetime NOT NULL default '0000-00-00 00:00:00',
  `Status` varchar(15) NOT NULL default 'Running',
  `DurationMinutes` bigint(20) default NULL,
  `InputFilePath` varchar(250) default NULL,
  `OutputFilePath` varchar(250) default NULL,
  `MacroCFilePath` varchar(250) default NULL,
  `MacroSFilePath` varchar(250) default NULL,
  PRIMARY KEY  (`CellDevicesUID`)
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;
-- --------------------------------------------------------
--
-- Structure de la table `CpoCellSimList`
--
DROP TABLE IF EXISTS `CpoCellSimList`;
CREATE TABLE IF NOT EXISTS `CpoCellSimList` (
  `CpoCellUID` bigint(20) NOT NULL auto_increment,
  `Username` varchar(30) NOT NULL default 'Alice',
  `Comments` text,
  `ClientAddr` varchar(250) default NULL,
  `ServerAddr` varchar(250) default NULL,
  `Date` datetime NOT NULL default '0000-00-00 00:00:00',
  `Status` varchar(15) NOT NULL default 'Running',
  `DurationMinutes` bigint(20) default NULL,
  `InputFilePath` varchar(250) default NULL,
  `OutputFilePath` varchar(250) default NULL,
  `FnatFilePath` varchar(250) default NULL,
  `RnatFilePath` varchar(250) default NULL,
  PRIMARY KEY  (`CpoCellUID`)
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;
-- --------------------------------------------------------
--
-- Structure de la table `ReactorSimList`
--
DROP TABLE IF EXISTS `ReactorSimList`;
CREATE TABLE IF NOT EXISTS `ReactorSimList` (
  `ReactorUID` bigint(20) NOT NULL auto_increment,
  `Username` varchar(30) NOT NULL default 'Alice',
```

```
  `Comments` text,
  `ClientAddr` varchar(250) default NULL,
  `ServerAddr` varchar(250) default NULL,
  `Date` datetime NOT NULL default '0000-00-00 00:00:00',
  `Status` varchar(15) NOT NULL default 'Running',
  `DurationMinutes` bigint(20) default NULL,
  `CpoCellUID` bigint(20) NOT NULL default '0',
  `AdjDevicesUID` bigint(20) NOT NULL default '0',
  `ZcuDevicesUID` bigint(20) NOT NULL default '0',
  `InputFilePath` varchar(250) default NULL,
  `OutputFilePath` varchar(250) default NULL,
  `VanFilePath` varchar(250) default NULL,
  `MapFilePath` varchar(250) default NULL,
  `FluxFilePath` varchar(250) default NULL,
  PRIMARY KEY  (`ReactorUID`)
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;
-- --------------------------------------------------------
--
-- Structure de la table `UserList`
--
DROP TABLE IF EXISTS `UserList`;
CREATE TABLE IF NOT EXISTS `UserList` (
  `UserUID` bigint(20) NOT NULL auto_increment,
  `Username` varchar(30) NOT NULL default '',
  `FirstName` varchar(30) default NULL,
  `LastName` varchar(30) default NULL,
  `Email` varchar(250) default NULL,
  `MD5Password` varchar(32) NOT NULL default '',
  PRIMARY KEY  (`UserUID`),
  UNIQUE KEY `UserName` (`Username`)
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;
--
-- Contenu de la table `UserList`
--
-- --------------------------------------------------------
--
-- Structure de la table `ZcuDevicesSimList`
--
DROP TABLE IF EXISTS `ZcuDevicesSimList`;
CREATE TABLE IF NOT EXISTS `ZcuDevicesSimList` (
  `ZcuDevicesUID` bigint(20) NOT NULL auto_increment,
  `Username` varchar(30) NOT NULL default 'Alice',
  `Comments` text,
  `ClientAddr` varchar(250) default NULL,
  `ServerAddr` varchar(250) default NULL,
  `Date` datetime NOT NULL default '0000-00-00 00:00:00',
  `Status` varchar(15) NOT NULL default 'Running',
  `DurationMinutes` bigint(20) default NULL,
  `CellDevicesUID` bigint(20) NOT NULL default '0',
  `InputFilePath` varchar(250) default NULL,
  `OutputFilePath` varchar(250) default NULL,
  `Zcu1natFilePath` varchar(250) default NULL,
  `Zcu2natFilePath` varchar(250) default NULL,
  `Zcu3natFilePath` varchar(250) default NULL,
  PRIMARY KEY  (`ZcuDevicesUID`)
) TYPE=MyISAM PACK_KEYS=0 AUTO_INCREMENT=1 ;
```

ANNEXE IV

FICHIER DE CONFIGURATION

Listing IV.1 Exemple de fichier de configuration du D2G2Server

```
# ------------------------------------------------------------------
# D2G2Server.conf file (ini config)
# ------------------------------------------------------------------
# We can use whitespace (tabs and spaces) freely.
# ------------------------------------------------------------------

[NETWORK]
# Put ip address rather than hostname
LocalServerAddr=132.207.60.45
ListenPort=30000

[DONJON]
DonjonPath=/home/nucl/etc/binLinux/donjon300F

[DRAGON]
DragonPath=/home/nucl/etc/binLinux/dragon304S
NuclearLibrariesPath=/home/nucl/lib/librariesLinux

[MYSQL]
ServerAddress=bickley.recherche.polymtl.ca
Database=D2G2
User=d2g2
Password=d2g2

[SIMULATIONFILES]
SimulationTemplatesPath=/home/parhe/d2g2/D2G2SimulationTemplates
SimulationDataPath=/home/parhe/d2g2/D2G2SimulationData

[XML]
DTDPath=/home/parhe/d2g2/D2G2XMLInputs
DTDFile=D2G2_XML.dtd

[MISC]
Verbose=FALSE
Syslog=TRUE
KeepTmpFiles=FALSE
```