4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

http://www.univ-orleans.fr/lifo

La visualisation distante

Sébastien Limet, Souley Madougou,
Emmanuel Melin et Sophie Robert
Université d'Orléans LIFO

Rapport N° **2006-12**
20/12/2006

Étude de composants logiciels pour la visualisation distribuée haute performance

Lot n°2 : État de l'art sur la visualisation distante

Rapport de synthèse sur les solutions étudiées

Préparé par

Laboratoire d'Informatique Fondamentale d'Orléans (LIFO)

Projet Parallélisme Réalité Virtuelle et Vérification (PRV)

Bât IIIA, Rue Léonard de Vinci

BP 6759

45067 Orléans Cedex

Rédigé pour

Commissariat à l'Energie Atomique

CEA/DIF/DSSI/SNEC

Centre CEA Bruyères-le-Châtel

91680 Bruyères-le-Châtel Cedex

Table des matières

I. Introduction

La visualisation de données est de plus en plus exploitée dans de nombreux domaines scientifiques (géologie, biochimie, mécanique, ...). Son utilité ne s'arrête pas à l'illustration de résultats d'expériences ou de simulation à des fins de vulgarisation mais elle devient un élément essentiel dans la compréhension de phénomènes étudiés dans les différentes disciplines scientifiques.

Dans tous les cas, face aux enjeux de la complexité des simulations sous-jacentes et de la taille des données générées, le développement des applications scientifiques se base désormais sur des systèmes parallèles et distribués. Ainsi, il devient classique d'envisager un déploiement d'une application sur trois sites par exemple. Le premier est en charge d'exécuter le code de la simulation, le deuxième permet le stockage des données générées ou paramètres de la simulation et enfin le dernier site réalise l'affichage des résultats. Cet état de l'art s'intéresse à ce dernier point. En effet, la visualisation distante [18; 19] implique des contraintes particulières qui obligent à définir des solutions spécifiques tenant compte par exemple des besoins en fluidité de l'affichage.

Nous nous sommes ainsi intéressés à différentes solutions de visualisation distante afin d'obtenir une classification permettant de juger de leur qualité en fonction des attentes des utilisateurs. Il existe plusieurs façons de classifier les différentes solutions de visualisation distante. Nous avons choisi d'aborder cette classification en fonction du lieu du rendu qui influe sur les performances en terme de taux de rafraîchissement de l'image par rapport à la qualité du réseau. Ce point de vue permet de traiter les différentes méthodes possibles pour répartir le pipeline graphique entre les deux sites impliqués et d'en voir les avantages et les inconvénients. Il fera l'objet de la première partie de ce document. Ensuite la visualisation distante s'appuie sur une infrastructure réseau dont dépendent les performances finales. Cette infrastructure peut être une simple connexion de type socket ou s'appuyer sur des concepts plus généraux qui visent à formaliser les aspects réseau pour des déploiements de type Grid. Ces aspects feront l'objet de la deuxième partie de ce document. Cette partie nous permettra de décrire différents modèles de communication qui apportent ou qui pourraient apporter à la visualisation distante un cadre de développement tout à fait pertinent. Enfin, il existe déjà aujourd'hui des systèmes que nous qualifions d'autonomes dédiés directement ou indirectement à la visualisation distante. Ces systèmes seront décrits dans la troisième partie de ce rapport. Ils viendront illustrer à la fois la classification selon le lieu du rendu et les différents modèles de communication possibles. De plus, un tableau général viendra résumer l'ensemble des systèmes cités en fonction des qualités d'une solution de visualisation distante soit :

- ✔ les performances (réseau, frame rate),
- ✔ les dépendances matérielles et logicielles,
- ✔ les dépendances graphiques
- ✔ la généralité et le niveau du système (middleware, framework, ...)
- ✔ les modalités de disponibilité (commerciales, open source) et le statut de développement et de maintenance
- ✔ l'évolutivité et l'ouverture vers des fonctions plus avancées (collaboratif, steering)

II. Classification en fonction du lieu du rendu

Dans cette première partie, il s'agit de décrire des solutions de visualisation distante en fonction du lieu d'exécution du rendu. Pour cela on suppose que les données initiales sont stockées sur une machine distante (simple PC, grappe ou super calculateur) appelée serveur et que l'affichage est effectué à partir d'une deuxième machine en local (ordinateur de bureau, grappe pilotant un mur d'images, ...) appelée client. Dans ce contexte, il est possible de décrire l'ensemble des étapes qui précédent l'affichage sous forme du pipeline donné figure 1

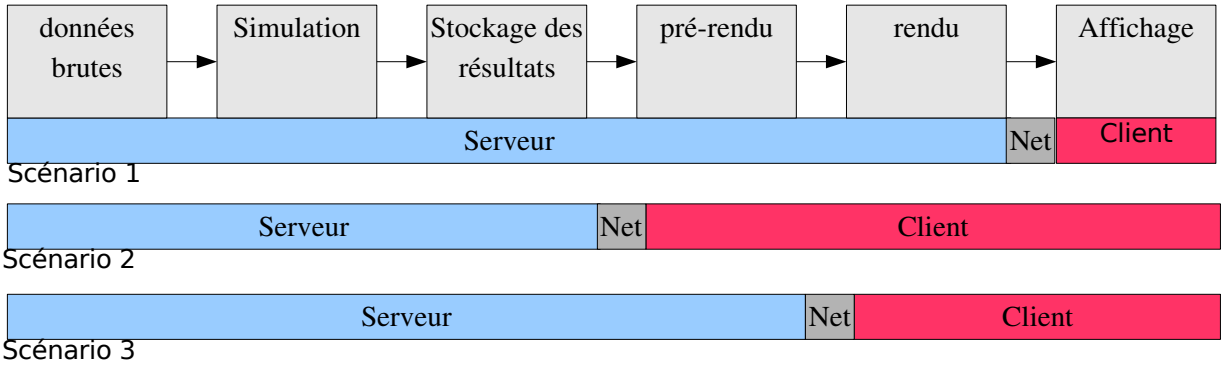

Figure 1 : Pipeline de la visualisation distante et répartition client/serveur (schéma inspiré de [47]).« Net » désigne l'opération de transfert des données entre le serveur et le client.

où

- l'étape « simulation » est facultative et désigne les calculs effectués sur les données brutes,
- l'étape « pré-rendu» désigne une partie du pipeline de visualisation avec par exemple le calcul de la géométrie, et est complétée par l'étape « rendu » (graphique).

Dans ce contexte nous avons retenu trois scénarios sur les quatre possibles :

1. Scénario 1 : le serveur effectue entièrement le rendu et envoie des images au client pour l'affichage. Cette technique est connue sous le nom d'image streaming.
2. Scénario 2 : le serveur effectue la simulation et envoie le résultat au client qui calcule entièrement le rendu avant l'affichage.
3. Scénario 3 : il y a une répartition de charge des opérations entre le serveur qui effectue le pré-rendu et le client qui réalise le rendu.

Le dernier scénario consisterait à envisager le serveur comme un simple lieu de stockage (base de données) avec un client qui effectue entièrement la simulation, le rendu et l'affichage. Dans le cadre d'une étude sur la visualisation distante, nous considérons ce scénario comme similaire au scénario 2 où le rendu est effectué en local.

1. Scénario 1 (Image streaming)

Le besoin en visualisation distante vient essentiellement de la nécessité de reporter des simulations lourdes en calcul sur des sites spécialisés équipés en super calculateurs ou en grappes. La volonté ensuite de visualiser les données générées soit sur une plate-forme locale de visualisation soit même sur sa propre machine de bureau est naturelle. La première idée dans ce contexte est de profiter de la puissance de calcul distante pour également procéder au calcul de rendu pour n'envoyer à l'utilisateur qu'un flux d'images (image streaming). Cette technique a bien sûr comme principal avantage de ne rien demander de spécifique au poste client. Elle permet également d'assurer la confidentialité des données en ne faisant transiter sur le réseau que des pixels et non des données brutes. En revanche le principal inconvénient de cette méthode est qu'elle sollicite le réseau pour l'envoi des images relatives à chaque changement d'un paramètre local de visualisation (point de vue, ...). Même si la bande passante demandée n'est dépendante que de la résolution locale et non de la taille des données ou de la complexité du rendu, ces envois répétés peuvent restreindre la qualité du taux d'images à l'arrivée. Dans ce scénario, la fluidité du rendu interactif nécessite un réseau à faible latence.
Dans cette section, nous allons décrire trois solutions d'image streaming (VNC, OpenGL Vizserver et VirtualGL) qui font référence et sont très utilisées. Par exemple, VaPOR développé par le National Center for Atmospheric Research qui a pour ambition de proposer une plate forme d'analyses et de visualisation de données fait appel à Vizserver pour l'aspect visualisation distante. Autre exemple, le projet TACC de l'université du Texas pour le calcul scientifique s'appuie sur Maverick une plate-forme logicielle et matérielle de déploiement d'applications scientifiques qui intégre VirtualGL/VNC pour le déport de l'affichage sur un poste client. Les centres de calcul nationaux CINES et IDRIS utilisent Vizserver depuis des années, exploitant ainsi des machines de puissance SGI Onyx au profit d'utilisateurs répartis sur toute la France.

1.1 -VNC

VNC pour Virtual Network Computing [58] est un protocole d'affichage distant s'appuyant sur le réseau TCP/IP. La première application visée est le bureau virtuel. Il s'agit de permettre un accès à son bureau à partir de n'importe quel poste relié à internet via un simple navigateur par exemple.

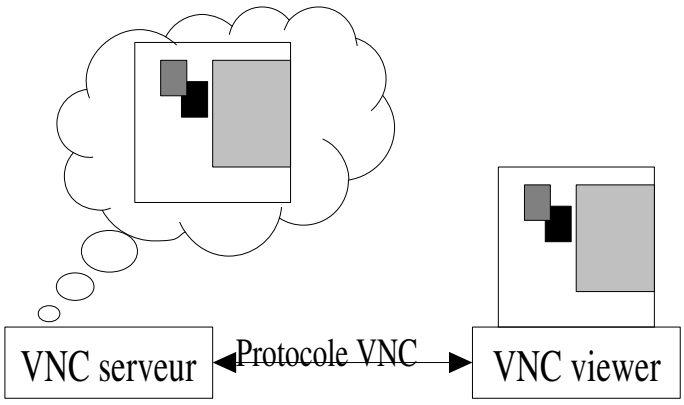

Figure 2 : Principe de l'affichage distant VNC

Le protocole VNC est basé sur une architecture client/serveur avec un protocole d'accès et de transfert du framebuffer vers la machine locale selon un encodage négocié entre le serveur et le

client. L'apport de VNC se situe à la fois sur la conception d'un client léger (viewer) (cf figure 2) et sur l'encodage des données adapté au type de la transmission, soit un ensemble de fenêtres. Les encodages possibles sont entre autres l'encodage par copie de rectangles, adapté aux modifications de type ascenseur dans une fenêtre, et l'encodage séparant les zones du bureau qui correspondent au fond d'écran et les zones qui correspondent réellement à des données.

Il s'agit donc d'une première approche de l'image streaming sachant que VNC est adapté à l'affichage du bureau d'un utilisateur, affichage qui ne nécessite pas une fréquence de rafraîchissement trop élevée. Par contre dans le cas de la visualisation scientifique, les performances de ce protocole peuvent être insuffisantes.

1.2 -OpenGL Vizserver

Cette solution est une solution commerciale développée par Silicon Graphics [57]. Elle permet à tout type de clients (PC, portable, grappe) quelque soit son système d'exploitation, d'accéder aux ressources d'une machine graphique SGI (Onyx ou Silicon Graphics Prism) et d'afficher le résultat sur le client. Deux modes d'affichage sont disponibles, soit par application soit le bureau complet.

OpenGL Vizserver est basée sur une architecture client/serveur, sur les bibliothèques OpenGL et X11 associées à des wrappers et sur des bibliothèques supplémentaires qui vont permettre de compléter certaines fonctions OpenGL et X11 pour les opérations de transfert vers le client. Le principe général est illustré par la figure 3. L'application OpenGL classique fonctionne sur le serveur en utilisant non pas les bibliothèques libGL.so et libX11.so natives mais les bibliothèques « wrappées » d'OpenGL vizserver qui agissent sur certaines fonctions OpenGL. La bibliothèque libvsx.so permet de gérer les informations nécessaires à OpenGL vizserver pour le transfert des images au client.

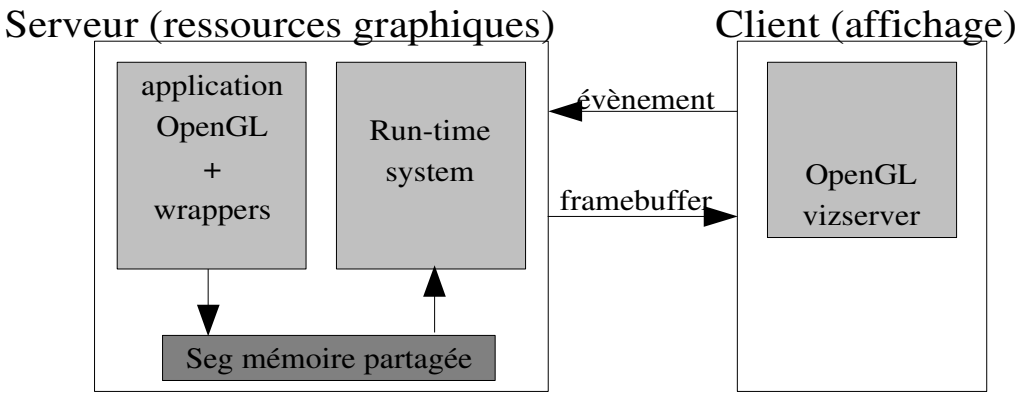

Figure 3 : Principe d'OpenGL vizserver.

Un deuxième processus vssession a la charge de compresser et d'envoyer ces données grâce à un protocole entre libvsx.so et vssession basé sur le partage d'un segment de la mémoire. A la réception d'un événement par exemple de type swap buffer de la part du client, l'application OpenGL gère le readback du framebuffer et vssession compresse et envoie l'image adaptée à la

résolution du client.

Les performances dépendent des techniques de compression utilisées (taux de compression et coût en CPU) et bien sûr de la bande passante disponible. De manière générale sur un réseau LAN 100 Mbit, les performances sont de l'ordre de 10 à 30 fps en fonction du taux de compression pour une résolution 1280x1024.

1.3 -HP Remote Graphics Software

HP propose également une solution logicielle analogue à OpenGL Vizserver basée sur de l'image streaming via des communications de type TCP/IP [27]. Le rendu effectué sur le serveur distant s'appuie sur l'accélération matérielle pour générer l'image résultante qui est compressée et envoyée au client.

1.4 -VirtualGL et les travaux de l'Université de Stuttgart

Le projet VIS (Visualization and Interactive Systems) de l'université de Stuttgart a pour thème de recherche la visualisation scientifique, en particulier le rendu volumique et aborde la question de la visualisation distante dans différents travaux. Le premier axe de recherche [56] aborde l'approche service web de la visualisation distante et sera décrit dans la partie 4 de ce rapport.

VIS s'intéresse avant tout à la visualisation 3D. Le deuxième axe de recherche concerne donc l'utilisation du rendu basé sur de l'accélération matérielle pour obtenir un frame rate correct, même si l'affichage n'est pas réalisé sur le site où s'effectue les calculs. Ainsi ce projet aborde le rendu distant selon le scénario 1. Leur approche est de permettre à toute application basée sur OpenGL et GLX d'être utilisée sur deux sites distants sans aucune modification du code. Elle a été implémentée (par des développeurs inconnus) sous le nom de VirtualGL, disponible en open source.

a) Description générale

Une application OpenGL/GLX peut fonctionner en mode direct comme l'illustre la figure 4. Ce mode appelé « direct rendering » permet aux commandes OpenGL d'être adressées directement à la carte graphique sans passer par le serveur X. Pour le moment ce mode est obligatoire lorsqu'on souhaite utiliser l'accélération matérielle de la carte graphique du serveur. Cependant, dans ce cas, l'affichage sur un site distant de l'image résultante n'est pas immédiat et ne peut utiliser le déport de l'affichage.

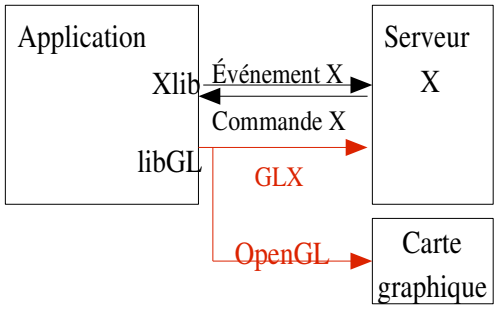

Figure 4 : Le mode direct pour OpenGL/GLX

Les travaux de VIS [52] consistent à utiliser le mode direct associé à de l'image streaming pour transmettre l'image au client pour affichage. En profitant des propriétés d'une édition de liens dynamique, leur solution permet sans aucune modification de procéder à de la visualisation distante à partir de toute application écrite en OpenGL et GLX, qui fonctionne en local.

L'architecture finale est basée sur le développement d'une bibliothèque supplémentaire à GLX (appelée ici GLX+) qui va permettre le traitement particulier de certaines fonctions GLX afin de transmettre l'image au client. Par exemple, à l'appel de la fonction glXSwapBuffers, le traitement consistera à lire le framebuffer puis à transmettre l'image résultante au client via la méthode XputImage ou XshmPutImage. La première solution proposée répond à l'architecture illustrée par la figure 5.

Cette architecture a été améliorée [53; 54] afin d'intégrer plus facilement des outils de compression adaptés à l'application. Ainsi après lecture du framebuffer, l'image est compressée et envoyée au client. A la réception de l'image par le client, elle est décompressée et traitée par son serveur X. Seul le serveur X du serveur doit reconnaître GLX.

b) VirtualGL

Cette dernière architecture a été développée sous le nom de VirtualGL en open source [49]. Le serveur utilise la bibliothèque appelée GLX+ dans ce rapport et le client est constitué principalement d'un démon en écoute. Ce dernier, à la réception de données, les décompresse et reconstitue les pixels dans la fenêtre X voulue avant traitement par le serveur X.

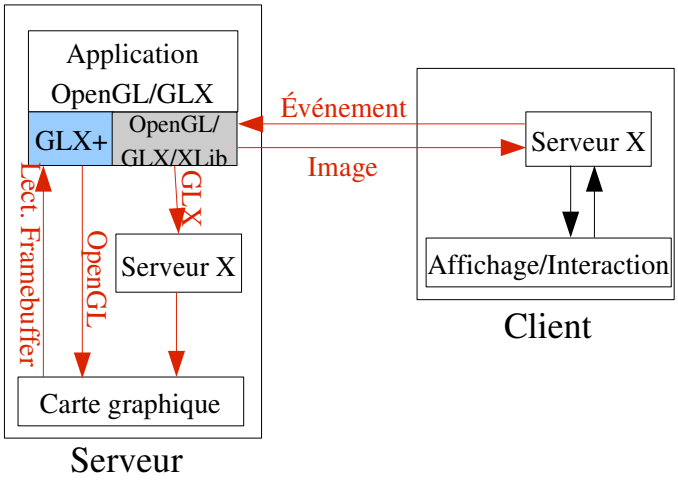

Figure 5 : Architecture OpenGL/GLX en mode direct avec image streaming

Il est également possible, à partir du mode direct, d'utiliser un proxy X qui va prendre en charge la réception des commandes et des évènements X mais également la compression et l'envoi de l'image résultante. VirtualGL propose donc une architecture supplémentaire qui intègre la bibliothèque GLX+ pour utiliser l'accélération matérielle mais VNC comme proxy afin de gérer la détection des changements d'affichage, la compression et l'envoi des images. Le client doit alors être un client VNC tel que décrit à la section 1.2. En terme de performances, il est en revanche préférable d'utiliser la solution sans proxy à celle avec VNC car ce dernier protocole est surtout adapté aux

applications 2D de type bureau (données majoritairement statiques avec de grands blocs de même couleur).

2. Scénario 2 (Rendu local)

De manière symétrique au scénario précédent, certaines solutions de visualisation distante sont basées sur un rendu en local. Les commandes graphiques sont alors envoyées au client dont les ressources (la carte graphique essentiellement) doivent être suffisantes pour assurer le rendu correspondant. L'avantage de cette famille de solutions est une utilisation moins fréquente de la ressource réseau. En effet, tant que les données ne sont pas modifiées, l'utilisateur peut interagir avec l'affichage sans demander de nouvelles communications. Cependant, les communications sous jacentes peuvent s'avérer très volumineuses et surtout beaucoup moins maîtrisées que dans le cas de l'image streaming où les données envoyées sont des pixels dont le nombre dépend uniquement de la résolution du client.

En dehors de l'approche brute, les solutions que nous allons décrire sont basées sur des techniques de multi-résolutions qui permettent de s'adapter à la bande passante disponible tout en assurant un affichage continu des données.

2.1 -Remote Display X

La solution la plus simple pour procéder à l'affichage sur un site distant est d'utiliser une connexion distante au serveur de calculs (ssh par exemple) puis d'exporter l'affichage sur la machine locale.

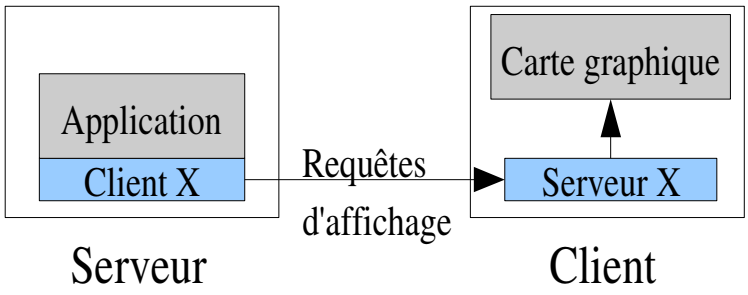

Figure 6 : Principe du Remote Display

A travers le protocole X11 toutes les requêtes d'affichage sont alors envoyées via TCP/IP au serveur X du client selon l'architecture donnée figure 6. Les images sont donc bien calculées en local et ce sont les commandes graphiques qui transitent sur le réseau. X11 est une API 2D, le protocole peut toutefois véhiculer des ordres 3D compris par une extension de type OpenGL.

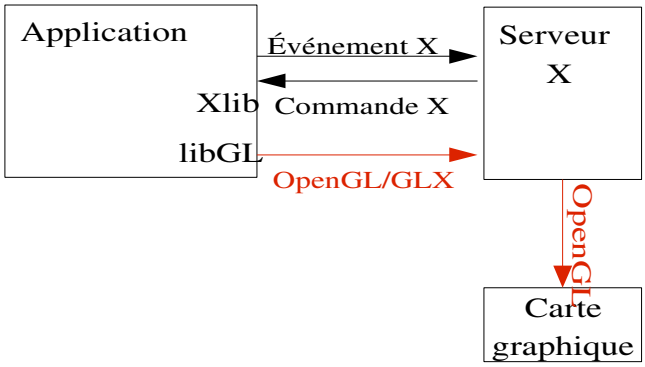

Figure 7 : Le mode indirect d'OpenGL

Pour les applications OpenGL, on retrouve ce principe dans le mode appelé mode indirect ou les commandes OpenGL issues du serveur, qui exécute l'application, sont transmises à la carte graphique du client via son serveur X comme le montre la figure 7.

2.2 -Approche data streaming

La deuxième approche de visualisation distante respectant le scénario 2, s'appuie sur un logiciel de visualisation exécuté sur le client qui communique avec le serveur contenant les données à afficher. Cette communication peut être de deux types. Soit l'écriture des résultats est effectuée sur un système de fichiers basé sur un montage réseau de type NFS soit les résultats sont transmis explicitement au logiciel de visualisation. Par exemple dans [22], le code GTC (Gyrokinetic Toroidal Code) fournit des résultats écrits dans des fichiers au format HDF5. Le client exécute AVS/Express où Express a été enrichi d'une fonction de communications par socket. Cette dernière permet à AVS/Express de recevoir les données à afficher, soit le fichier HDF5 souhaité, et de calculer le rendu correspondant pour l'affichage. Cependant, lorsque les données à visualiser sont de grande taille, le code qui les génère peut être parallèle. Dans ce cas, il est intéressant soit de faire appel à des systèmes de fichiers parallèles et distribués comme PVFS [44] ou GPFS soit d'organiser une fusion des résultats du serveur vers le client. Là encore, [22] ou [32] proposent d'utiliser GridFTP [23] qui fait partie du toolkit Globus. Par exemple, [22] associe à chaque processeur générant des données un thread supplémentaire pour les opérations de sortie qui consiste en une lecture d'un buffer mémoire alloué à l'application. Ces threads sont en charge du transfert des données en blocs successifs vers le service GridFTP du client. Ils gèrent également des reprises de transmission en cas de pertes de données.

Le principal inconvénient de ces approches est d'être dépendant de la qualité de la bande passante du réseau utilisé. En effet, la quantité de données à transmettre pour effectuer la transmission des données à visualiser peut être conséquente. En cas de données variant dans le temps, cette transmission peut devenir prohibitive pour permettre un affichage de qualité.

2.3 -Approche data streaming avec multi-résolution

Cette approche a pour objectif de remédier aux inconvénients du data streaming brut en ajoutant un

nouvel élément dans l'architecture entre la source de données et le site de visualisation. Le rôle de cet élément est d'élaguer et de simplifier les données provenant de la source de données vers le client d'affichage. Dans [39] par exemple le serveur intermédiaire, appelé serveur de données, effectue des simplifications en utilisant des techniques de multi-résolution. L'idée exploitée dans cette approche est de considérer que l'exploration des données comporte des phases de navigation durant lesquelles l'utilisateur recherche la partie des données qui l'intéresse et des phases d'analyse visuelle des données. Durant les phases de navigation, l'utilisateur peut se contenter d'une version simplifiée des données qui lui permet de se repérer dans le modèle, par contre dans les phase d'analyse les données affichées doivent être proches des données réelles. Partant de ce principe, le serveur intermédiaire calcule des représentations à différentes résolutions des données brutes. Lorsque l'utilisateur change de point de vue, les données de la résolution la plus faible lui sont envoyées, lorsque le point de vue se stabilise, le serveur de données augmente la résolution des données qu'il transmet à la visualisation. En procédant ainsi, on peut fluidifier la navigation dans les données au détriment de la qualité d'affichage, mais retrouver les données complètes lorsque le point de vue s'est stabilisé. Cette technique permet aussi d'adapter la résolution maximale des données en fonction du dispositif d'affichage. En effet, si l'utilisateur veut effectuer un affichage sur un ordinateur de bureau avec peu de ressources graphiques, le serveur intermédiaire peut éviter d'envoyer les résolutions les plus hautes des données.

Dans le projet VISUS, le rôle du serveur de données est non seulement de calculer et gérer la multi-résolution mais il permet aussi de réorganiser les données afin d'optimiser les communications entre ce serveur et la visualisation. Il s'agit de définir la structure de données qui va permettre de minimiser les phases d'exploration et d'extraction des données à visualiser. On retrouve cette idée également dans [7] qui s'appuie sur du « chessboarding » pour réduire les temps d'accès et la taille des communications ou encore dans [32] qui lui utilise le standard HDF5.

Le principal avantage de cette approche est bien évidemment d'éviter les écueils du data streaming en permettant une navigation interactive dans les données. Cette navigation se fait avec une dégradation temporaire de la qualité d'affichage, ce qui peut être une contrainte acceptable pour l'utilisateur. Il faut quand même noter que dans les système étudiés, les données variant au cours du temps ne sont pas évoquées. De plus, il subsiste une difficulté majeure dans la mise en oeuvre de cette technique, c'est l'existence d'un algorithme efficace de calcul de la multi-résolution pour le type de données que l'utilisateur veut afficher. C'est d'ailleurs le coeur du projet VISUS, qui a développé plusieurs algorithmes pour différents types de données mais qui n'est évidemment pas exhaustif.

On peut remarquer que cette approche est vraiment très proche du scénario 3 puisque le rôle du serveur intermédiaire est de préparer les données en vue de l'affichage et on peut tout à fait imaginer que dans ce cadre, ce serveur pourrait effectuer une partie du rendu.

3. Scénario 3 (Approches mixtes)

Dans le cadre du scénario 1, pour obtenir de bonnes performances, il faut un réseau de faible latence pour permettre la fluidité de l'affichage tout en offrant un taux d'interaction élevé avec l'utilisateur. Par contre il est difficilement applicable à des supports d'affichage comme le CAVE ou le mur d'images. Pour le scénario 2, un bon frame rate est dépendant de la bande passante du réseau et des ressources graphiques du client. Cependant, pour de gros volumes de données il peut atteindre rapidement ses limites et ne permettre une bonne interaction qu'au prix d'une dégradation

de la qualité de l'image (même temporaire).

Le scénario 3 consiste en un compromis entre ces deux démarches. Les solutions répondant à ce scénario peuvent être divisées en deux catégories. La première propose une répartition entre le serveur et le client basé sur un calcul de la géométrie par le premier et une prise en charge du rendu final par le deuxième. Dans ce cas, l'interaction avec l'utilisateur est complète et le réseau n'est sollicité que lorsque les données à visualiser varient dans le temps. La seconde catégorie est plus complexe et si elle permet une répartition de charges, elle ne permet qu'une interaction partielle avec l'utilisateur.

Dans la partie 3, nous verrons que de nombreux systèmes autonomes proposent une visualisation distante selon le scénario 3. Dans cette partie nous avons choisi de présenter Visapult car il s'agit d'une référence pour la visualisation distante, ainsi qu'une partie des travaux du SCI Institute pour les différentes répartitions de charges proposées. Dans la dernière section, nous décrirons une architecture proposée par le centre de recherche Ames de la NASA. Cette dernière est une approche mixte un peu différente dans le sens où elle exploite trois sites différents pour le génération des données à visualiser, le calcul du rendu et enfin l'affichage.

3.1 -Visapult

Visapult [4; 50; 51] est une solution complète de visualisation distante de gros volumes de données développée par le Visualization Group du Lawrence Berkeley National Laboratory. Ce projet est affiché comme un projet terminé avec comme successeur DIVA dont les principes seront décrits au chapitre suivant. Cependant, le prototype Visapult est encore disponible en open source.

Visapult propose une répartition de charges entre le serveur et le client pour le calcul de rendu. Cependant, cette solution appartient à la catégorie 2 que nous avons définie dans le sens où le degré d'interaction avec l'utilisateur est limité. Seul un ensemble de transformations applicables aux données sans dégradation de l'image sont possibles.

Le système est composé de deux parties. D'un côté le serveur, appelé le back-end, effectue les calculs de rendu volumique. De l'autre le client qui est appelé le front-end ou le viewer reconstruit l'image et l'affiche. L'originalité de cette architecture est l'algorithme de rendu volumique utilisé dit IBRAVR pour Image-Based Rendering -Assisted Volume Rendering [28]. Cet algorithme consiste à générer un ensemble d'images des données depuis un point de vue en utilisant le rendu volumique d'un certain nombre de sous volumes appelés « slabs » qui correspondent à des tranches du volume englobant. Le rendu de chaque sous volume est calculé et transmis au viewer qui reconstruit alors l'image. Cependant, puisque le rendu volumique des « slabs » est calculé par le serveur à partir d'un point de vue donné, seules des interactions de type rotation de faible amplitude peuvent être appliquées aux données sans dégradation de l'image.

L'architecture de Visapult est donnée figure 8. Le back-end est ainsi parallèle et écrit en MPI. Chaque processeur reçoit un certain nombre de sous volumes dont il calcule le rendu. Le front-end est multi-threadé en utilisant pthreads et utilise OpenRM Scene Graph. Les communications sont réalisées selon un protocole ad hoc de type TCP de telle manière que chaque image 2D générée par un processeur soit reçue et traitée par un thread au niveau du front-end.

La dernière version Visapult2, améliore le modèle en offrant une parallélisation à partir d'un

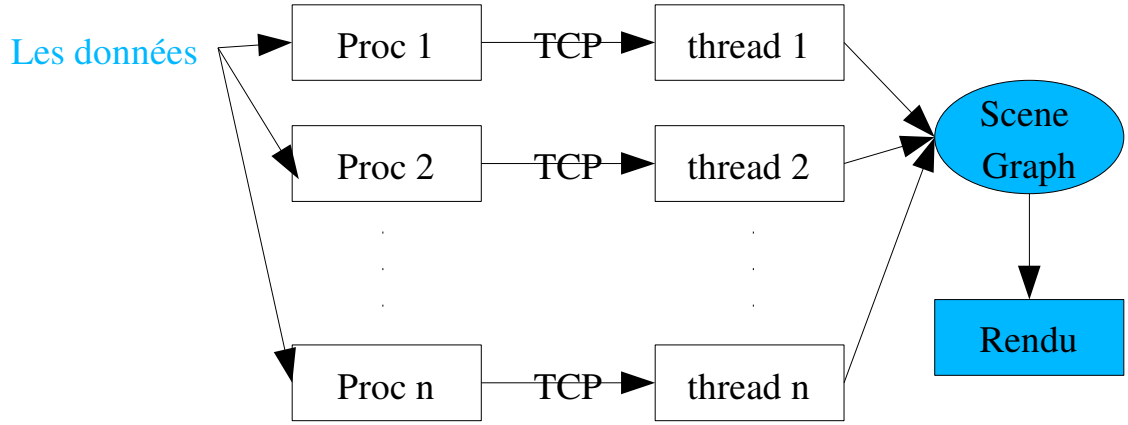

Figure 8 : Architecture de Visapult

découpage en blocs du volume à rendre afin d'assurer une meilleure scalabilité et d'être indépendant de la forme du volume. En effet, un « slab » peut être encore trop grand en fonction de cette forme pour tenir dans l'espace mémoire d'un processeur. De plus le rendu est désormais effectué en fonction des 6 directions primaires et ce sont donc 6 rendus qui sont transmis pour la recomposition de l'image finale sur le client.

3.2 -Prototype du SCI Institute – Université de l'Utah

Il s'agit d'un Master's thesis qui a été publié dans IEEE Visualization en 2002 par Eric J. Luke et Charles D. Hansen du SCI Institute [47]. Cette équipe est également à l'origine de SCIRun, un environnement de programmation pour le calcul scientifique et la visualisation et si ce master constitue leur première approche de la visualisation distante, nous verrons que désormais leur recherche sur ce thème se concentre sur l'approche par composants avec SCIRun2.

Ce travail décrit une infrastructure complète pour de la visualisation distante offrant les trois possibilités de localiser le lieu du rendu en local, à distance ou en répartissant la charge. Cette infrastructure repose sur une architecture client serveur avec un ensemble d'interfaces qui permet d'abstraire l'utilisateur des problèmes de communication par exemple. Seul un prototype a été développé mais il a permis de valider une approche mixte également basée sur le rendu d'images. Les interactions avec l'utilisateur sont illimitées dans le sens où pour chaque changement de point de vue, le serveur reçoit l'information, procède au calcul du pré-rendu et le transmet au client.

La répartition, pré-rendu sur le serveur, rendu sur le client, s'appuie sur l'algorithme suivant. Dans un premier temps le color buffer est transmis en image streaming au client. Ensuite le serveur prend en charge le calcul du depth buffer pour chaque nouveau point de vue. A la réception de ce buffer,

le client termine le calcul de la texture correspondante et affiche le résultat. Les tests de ce prototype sur un serveur SGI origin 2000, un client de type pentium III et un réseau ethernet ont montré que cette approche mixte générait un meilleur frame rate que l'image streaming ou que la solution de répartir la charge entre le serveur pour la géométrie et le client pour le rendu final.

3.3 -Visualisation concurrente de Ames (NASA Ames Research Center)

Les travaux décrits ici sont issus d'une publication récente de David Ellsworth et *al* [61]. L'objectif général est la visualisation concurrente définie comme la visualisation et l'interprétation temps réel des données au fur et à mesure de leur génération (time-varying data). Ces travaux apparaissent comme une approche mixte dans le sens où le couplage simulation/visualisation est effectué sur trois sites. Il respecte ainsi à la fois le scénario 2 pour le transfert des données au serveur de rendu et le scénario 1 pour le transfert des images au serveur d'affichage.

Le projet MAP'05[1] s'intéresse à la compréhension des phénomènes de cyclones tropicaux. Il se base sur GEOS4 (finite-volume general circulation model) une simulation parallèle hybride MPI et OpenMP qui génère à chaque itération des résultats qui doivent être visualisés (soit 19.4GB de données à traiter).

Le déploiement de l'application est le suivant :

- la simulation s'exécute sur une grappe composée de 20 noeuds SGI Altix et écrit sur un segment mémoire partagé ses résultats,
- un processus indépendant lit ses données et les transmet via un réseau haut débit infiniband à un noeud spécifique de la grappe,
- Le processus recevant les données se charge de les transmettre via un réseau privé basé sur TCP aux différents noeuds de la grappe graphique,
- le serveur de rendu effectue la génération des images et les encode au format MPEG avant d'écrire le résultat via NFS sur un serveur de fichiers,
- ce serveur de fichiers relaie ces images au noeud maître du serveur d'affichage via des requêtes ssh,
- enfin les noeuds d'affichage reçoivent à leur tour les images et procèdent à l'affichage.

L'originalité de cet exemple est de traiter des données variant dans le temps avec pour objectif de ne pas pénaliser une simulation déjà complexe, par l'extraction des données et leur visualisation. De plus elle permet d'illustrer une possible combinaison des techniques relatives au scénario 2 et au scénario 3 avec un enjeu plus fort sur le serveur d'affichage qui peut être n'importe où. Les performances restent donc dépendantes de l'efficacité de la compression MPEG utilisée. Cependant, cette approche reste ad hoc et fortement liée aux moyens matériels disponibles pour le projet, soient une grappe de calculs , une grappe graphique, un serveur de fichiers et enfin une grappe d'affichage.

1 http://map05.gsfc.nasa.gov/

III. Modèles de communications

La visualisation distante nécessite d'étudier les solutions d'un point vue répartition du pipeline graphique objet du chapitre précédent mais également d'un point de vue modèle de communications associé au réseau grande distance en jeu dans ce contexte. Les systèmes autonomes que nous allons décrire se basent tous sur un de ces modèles qui peut aller du plus simple avec l'utilisation des sockets à des modèles plus complexes qui peuvent dépasser le cadre de la visualisation distante. Cependant, ces derniers sont plus adaptés aux enjeux des réseaux de type Grid. Et on peut penser qu'ils seront de plus en plus utilisés au fur et à mesure de leur maturité surtout lorsque les systèmes de visualisation distante souhaiteront traiter des problèmes plus complexes tenant compte par exemple de la variation des données dans le temps.

1. Réseau sur le modèle TCP/IP

La plupart des systèmes de visualisation distante étudiés dans ce rapport se basent sur des communications utilisant des sockets TCP/IP. Bien que peu d'articles indiquent les raisons de ce choix, on peut facilement les deviner. Tout d'abord ce modèle de communication est un standard très largement répandu quelque soit le système d'exploitation et le type de réseau utilisé et par conséquent il est connu de tous les programmeurs. Il est de plus assez simple à mettre en oeuvre et permet de transférer efficacement de gros volume de données. Ce choix peut donc paraître très pertinent mais en fait il comporte un certain nombre d'inconvénients qu'il est important de souligner. Ces inconvénients sont liés au caractère très généraliste de TCP/IP. Afin de mieux les comprendre nous allons tout d'abord faire quelques rappels sur le modèle de réseau TCP/IP.

IP (Internet Protocol) est un protocole réseau qui a été implémenté au dessus de la quasi totalité des protocoles de liaisons existants sur le marché. TCP (Transmission Control Protocol) est un protocole de transport connecté qui se place au dessus d'IP. TCP garantit notamment à l'expéditeur la délivrance des données au destinataire. UDP (User Datagram Protocol) se situe au même niveau que TCP, il offre donc des services similaires mais il ne garantit pas la délivrance des données au destinataire. De cette manière UDP permet des transmissions plus rapides que celle de TCP, ce qui peut être très intéressant dans le cadre de l'image streaming par exemple, où on peut imaginer qu'il n'est pas très important de perdre une image de temps en temps mais où la vitesse de transmission est extrêmement importante.

Le modèle de communication TCP/IP a donc un aspect assez rudimentaire dans le sens où il permet uniquement des transferts de données non typées entre deux machines[2]. De ce fait on peut implémenter à peu près n'importe quel type de communication en se basant sur les sockets TCP/IP, mais cela signifie que TCP ne prendra pas en charge le type des données que l'on transfère. Par conséquent c'est à l'application qui utilise la liaison par socket de mettre en place toute la gestion spécifique aux données transférées. Il en résulte que chaque système de visualisation distante utilisant TCP/IP a dû implémenter des solutions ad hoc pour effectuer cette gestion. On peut noter que cette gestion n'est jamais décrite pour les systèmes étudiés. Seul le système COVISE fait état de la mise en place d'un protocole optimisé d'échanges de données entre ses composants sans le décrire.

2 UDP permet de faire de la diffusion de données à partir d'une machine à destination de plusieurs machines mais cela nécessite le plus souvent une gestion assez pointue du suivi des données envoyées et reçues.

La conséquence de tout ceci est qu'avec TCP/IP il est très simple de faire une communication point à point entre deux entités, ce qui est le scénario le plus souvent retenu dans le cadre de la visualisation distante. Cela permet rapidement de tester la validité d'une solution. Le passage de la communication point à point à une communication plus complexe (entre n serveurs de données et m machines de visualisation par exemple) est loin d'être trivial et demande des développements spécifiques complexes.

2. Approche par composants

Les composants logiciels constituent un modèle de programmation assez récent dont l'objectif est de favoriser une approche plus modulaire du développement de codes et leur ré-utilisation. Un composant se définit comme l'implémentation d'un algorithme quel qu'il soit plus un ensemble d'interfaces appelées ports, qui détermine les liens de ce composant avec l'extérieur. L'application est alors réalisée en inter-connectant un ensemble de composants grâce à la définition de leurs ports. L'ensemble des règles qui régissent ces liens et leur implémentation est appelé l'infrastructure d'intégration des composants. Enfin, l'association de l'infrastructure et de tous les outils pour développer l'application par composants constitue l'architecture par composants . Plusieurs modèles de composants logiciels existent. Parmi ceux adaptés aux calculs scientifiques, on trouve CCM (Corba Component Model) et CCA (Common Component Architecture) qui font l'objet dans les deux cas d'une volonté de normalisation via des consortiums, OMG (Object Management Group) pour CCM[3] et CCA forum pour CCA[4].

2.1 -CCM

Le modèle CCM [9] est la spécification CORBA des composants logiciels par OMG. La version CORBA 2.0 n'est pas une approche par composants mais une architecture objets distribués. La version CORBA 3.0 la complète avec de nouveaux services tels que la composition et le déploiement des objets/composants. Cette dernière devient une réelle approche par composants .

Ce modèle CCM est relativement complexe. Il s'articule autour d'un bus d'objets répartis ou ORB (Object Request Broker) qui assure le transport des requêtes entre des objets qui appartiennent à une des catégories suivantes :

- ✔ services objets communs qui fournissent des abstractions aux fonctions système telles que la sécurité ou le nommage et qui rendent les applications indépendantes des implémentations lorsque ces services sont utilisés,
- ✔ interfaces de domaines qui concernent les objets métiers relatif à un domaine d'activités,
- ✔ utilitaires communs qui peuvent offrir une abstraction supplémentaire pour des fonctionnalités communes à un grand nombre d'applications (GUI, gestion des tâches, ...),
- ✔ objets applicatifs qui sont les modules objets développés par l'utilisateur.

Dans le cas le plus simple qui nous intéresse, le bus CORBA est utilisé pour le transport des requêtes entre des objets applicatifs dont le développement est à la charge de l'utilisateur.

Un objet est défini dans un langage de programmation quelconque mais possède une interface écrite

3 http://www.omg.org
4 http://www.cca-forum.org/

en IDL (Interface Definition Language) qui permet de décrire son mode de communication avec les autres objets connectés au bus CORBA selon le modèle objet client/serveur comme le montre la figure 9.

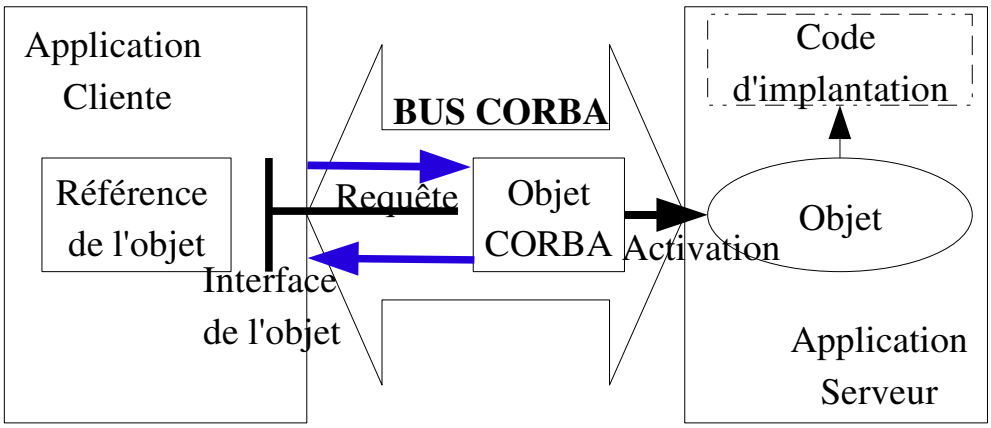

Figure 9 : Le modèle objet de CORBA

Une pré-compilation des descriptions IDL des interfaces permet de les construire en tant que souches de communication que l'utilisateur peut désormais utiliser pour appeler un objet dans son application cliente. Lors de l'exécution de l'application, cette invocation est alors prise en charge par le bus CORBA de manière transparente pour l'utilisateur dans le sens où pour lui tout se passe comme si l'objet était local. Le bus applique alors la bonne invocation en fonction du lieu d'exécution de l'objet appelé qui peut être sur le même processus, sur un processus différent mais sur une même machine ou sur une machine différente. Cependant, cette transparence dépend de l'implémentation CORBA utilisée qui doit prendre en charge les trois lieux d'exécution de l'objet invoqué. Il est en particulier important de choisir une implémentation de CORBA qui respecte le protocole IIOP (Internet Inter-ORB protocol) qui définit la communication pour des objets distribués utilisant le protocole TCP/IP d'internet.

Cette technologie est utilisée dans de nombreuses solutions ad hoc ('petites' applications) réalisées par les utilisateurs eux-mêmes pour répondre à un besoin de généricité de leurs applications ou de leurs déploiements sur systèmes distribués telles que [34] ou [56] ou encore [20].

Par exemple, [56] décrit un framework permettant de contrôler et d'afficher le résultat d'une application OpenInventor selon l'architecture illustrée figure 10.

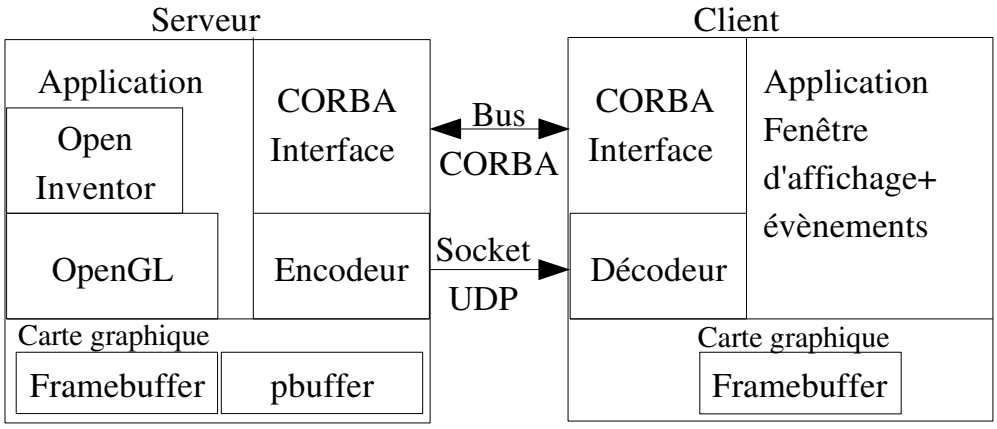

Figure 10 : Architecture du framework basée sur OpenInventor [56]

Cette approche suit un modèle de programmation orienté objet permettant au client via une interface CORBA d'accéder à distance à des méthodes disponibles sur le serveur. La réception d'un événement de la part du client sous forme de requête CORBA, déclenche sa traduction en événement OpenInventor. L'image est alors générée dans le pbuffer qui permet d'utiliser l'accélération matérielle de la carte graphique sans effectuer d'affichage. L'image générée est encodée puis envoyée au client par une connexion de type socket UDP. L'application d'affichage du client peut être soit une applet java appliquée à un navigateur web soit un système ad hoc.

Cette solution répond donc au scénario 1 de l'image streaming. De plus, un schéma de compression basé sur les techniques de flux vidéo sur internet est proposé. L'idée est que dans le contexte d'une telle diffusion sur plusieurs clients, il faut tenir compte de leur hétérogénéité en termes de puissance de calculs et de bande passante. Le codec s'appuie sur une pyramide de résolution spatio-temporelle qui transforme le signal original en deux sous-couches de résolutions différentes qui sont toutes les deux transmises. De cette manière tout client est au moins capable de traiter la couche de plus faible résolution.

Autre exemple, dans [20], Shumilov et *al* proposent une architecture basée sur une transformation de GeoToolkit (bibliothèque de transformation et représentation d'objets spatio-temporels géologiques) en composant CORBA via un adaptateur appelé GTA pour GeoToolKit/CORBA Adapter. Ainsi tout client compatible CORBA est à même d'accéder à la base de données et de visualiser les données correspondantes dans un navigateur. En particulier, leur implémentation propose un client basé sur une interface utilisateur graphique (GUI) qui effectue la requête à la base de données, s'appuie sur GOCAD pour le calcul de la géométrie et GRAPE pour le rendu. Ensuite un navigateur compatible VRML enrichi d'une applet java permet l'affichage de ces données spatio-temporelles.

- CORBA pour le calcul haute performance

CORBA est plus souvent associé à des applications de type client/serveur simple qu'à des applications scientifiques pour des raisons d'adaptation à un modèle fortement parallèle ou distribué ou de performances. Cependant, il existe des modèles de programmation adaptés aux applications scientifiques haute performance qui sont basés sur CORBA, dont certaines implémentations ont des performances très proches de celles de MPI [10].

Dans ce sens, le projet PARIS[5] de l'IRISA propose différentes extensions de CORBA aux objets ou aux composants distribués.

Par exemple, dans le cadre du modèle orienté objet, ce projet propose plusieurs définitions d'un objet parallèle. Dans [40], chaque objet parallèle est vu comme une collection d'objets CORBA et le langage IDL est complété de mots clés supplémentaires pour exprimer les nouvelles communications parallèles entre objets. L'inconvénient de cette approche est de modifier l'implémentation de CORBA et de nécessiter de nouveaux compilateurs tenant compte des nouvelles définitions du langage IDL. Avec PACO++ [41], le choix consiste à ajouter un niveau d'abstraction au dessus des implémentations existantes de CORBA. Ainsi tout objet parallèle est également une collection d'objets CORBA selon le principe SPMD, qui peuvent utiliser MPI par exemple pour des communications internes. On associe à cet objet parallèle deux contextes partagés par tous les objets CORBA qui le composent. Le premier « global context » permet à tout objet parallèle d'être un objet CORBA et le deuxième « operation context » permet d'exprimer toutes les opérations parallèles. PACO++ est disponible en OpenSource et a également fait l'objet de tests sur une application d'EADS dans le cadre de l'ACI Grid-RMI[6].

2.2 -CCA

Le modèle CCA [6] est un modèle par composants logiciels qui vise la performance dans le cadre du calcul scientifique. Par rapport aux spécifications CCM, il simplifie le modèle de composant de CORBA 3.0. Par contre CORBA 2.0 en fonction de son implémentation peut être conforme au standard CCA.

Ce modèle se base sur un composant plus simple que CORBA 3.0 et ne contient que deux types de ports pour invoquer une méthode (« uses port ») ou fournir une méthode (« provides port »). Il s'appuie sur un langage IDL appelé SIDL (Scientific Interfaces Description Language) adapté à la description des données manipulées par les applications scientifiques comme les tableaux de dimension *n* par exemple et supportant l'héritage à la Java. Une implémentation par composants conforme au modèle CCA fournit donc ce langage SIDL, les différentes API et les services nécessaires à la construction et la gestion de l'application CCA comme le montre la figure 11.

La particularité de l'approche CCA est d'adapter un modèle par composants aux besoins des applications scientifiques et en particulier au couplage de codes en vu des contraintes de performances de ces applications. Cette particularité s'exprime au niveau du SIDL adapté aux structures de données scientifiques mais également sur la définition des ports qui peuvent être directs ou collectifs. Dans le premier cas, un « uses » port n'admet une connexion qu'à un unique « provides » port. Dans le deuxième cas c'est un ensemble de *n* « uses » ports qui peuvent être connectés à *m* « provides » ports avec dans ce cas la possibilité de représenter tous les schémas de communications *n*×*m* classiques (gather, scatter, ...) aux applications parallèles.

5 http://www.irisa.fr/paris/web/

6 http://www.irisa.fr/Grid-RMI/www/fr/

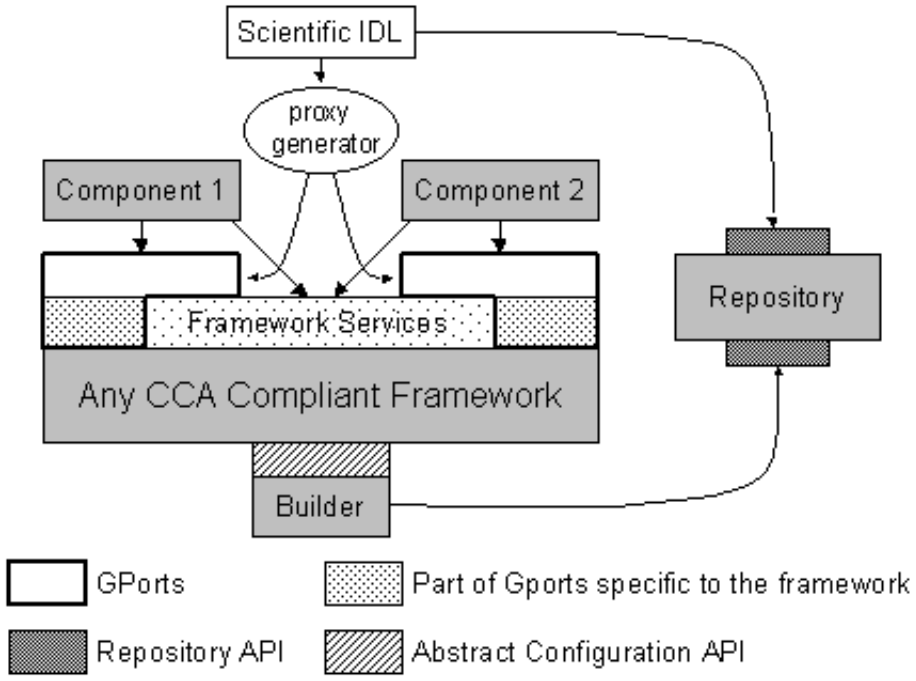

Figure 11 : L'architecture d'une application CCA

Les développements CCA disponibles actuellement sont encore peu nombreux et souvent de l'ordre du prototype. De plus, cette approche est bien plus générale que la visualisation distante qui est le thème de ce rapport. Nous en verrons cependant un exemple dans le chapitre « Systèmes autonomes » avec SCIRun2. Ce modèle en tant que tel est mentionné dans cet état de l'art, car il est prometteur pour tout projet scientifique dont l'ambition est de déployer des applications complexes de type couplage de codes intégrant des outils de visualisation, ces derniers étant des composants comme les autres. Il devrait permettre d'assurer des mises en oeuvre d'applications qui prennent en compte les besoins en performances et en inter-opérabilité tout en abstrayant le scientifique d'une large partie des problèmes de développement relatifs à l'hétérogénéité des plate-formes et aux modèles d'échanges entre les composants.

3. Services web

De manière générale, un service web est défini comme un mode d'échanges de données via « internet » entre différents logiciels tournant sur des machines hétérogènes. Il s'agit donc d'une implémentation dans un langage quelconque qui délivre un service qui doit pouvoir être découvert et invoqué dynamiquement. Un ensemble de normes et de protocoles au sein du « Web Services Protocol Stack » sont disponibles (ou en cours de spécification) pour assurer les échanges et l'inter-opérabilité. Parmi ces normes, on retrouve par exemple XML pour le format de données, VRML pour les données graphiques [59], SOAP (Simple Object Access Protocol) [48] qui implémente un protocole RPC (Remote Procedure Call) basé sur XML pour les données et http pour les communications.

Dans le cadre de la visualisation, nous pouvons définir les services web comme une mise à disposition de données via internet pour un affichage dans un simple navigateur compatible. Les

solutions qui peuvent se revendiquer de type service web pour de la visualisation distante semblent appartenir à deux familles distinctes. La première utilise CORBA avec une implémentation respectant la IIOP et donc une approche par objets/composants pour assurer l'inter-opérabilité entre le serveur distant et le client. Nous retrouvons ici les travaux du projet VIS de l'université de Stuttgart. La deuxième s'appuie sur les outils du « Web Services Protocol Stack » de type SOAP. Le système gViz qui sera décrit dans la partie suivante peut être associé à cette famille.

Cette approche existe également pour accéder à des bases de données volumineuses et permettre l'affichage de ces données sur des clients locaux. Par exemple pour le domaine de l'astronomie, au sein du CSIRO (Australian national scientific research organisation), l'ATNF (Australia Telescope National Facility) propose un outil de visualisation distante de type service web RVS[7] (Remote Visualisation System). Cette solution logicielle est basée sur un serveur accédant aux bases de données des images FITS et effectuant les calculs et sur un client de type viewer java qui se connecte au serveur RVS en lui indiquant la base de données de travail et procède à l'affichage des images reçues. On retrouve cette approche aussi dans Aladin développé par le centre de données astronomiques de Strasbourg dont une démonstration est disponible à l'adresse http://aladin.u-strasbg.fr/java/nph-aladin.pl. Là encore un viewer plugin java d'un navigateur web permet d'accéder au serveur d'images digitalisées du ciel. De plus cette version permet également un affichage de ses propres données par superposition.

4. Globus[8]

Globus ne peut être défini sur le même plan que les autres projets de recherche. C'est plus que cela puisque ses concepteurs le pensent en tant qu'*écosystème* de composants et outils qui inter-opèrent ensemble. Le projet est constitué : d'une *communauté* d'utilisateurs et de développeurs open source centrés sur les thèmes du calcul distribué ; d'organisations virtuelles ; et d'une fédération de ressources. La partie *software* est constituée du Globus Toolkit qui comporte un ensemble de bibliothèques et de programmes qui répondent aux problèmes rencontrés lorsque l'on conçoit des systèmes distribués. Globus est aussi l'*infrastructure* qui supporte la communauté : répertoires de codes, système de débogage, listes, etc... De cet ensemble doit émerger des solutions, des savoir-faire classiques sur lesquels s'appuyer pour faciliter la conception des futures applications sur Grille.
La partie concrète et visible pour les utilisateurs reste principalement le **Globus Toolkit**. Il fournit une variété de services allant d'implémentations de services tels que la gestion des ressources, la transmission des données et la découverte de services jusqu'aux outils pour en construire de nouveaux : Web Services (Java, C, Python). Globus fournit une infrastructure de sécurité pour l'authentification, l'identification, l'autorisation d'accès. Enfin il fournit aux clients à la fois des API (dans différents langages) ainsi que des programmes en ligne de commande pour accéder aux services Globus.
La dernière version du Golbus Toolkit est la numéro quatre (GT4). Cette dernière fait un usage extensif des services web afin de définir ses interfaces et structurer ses composants. Leurs protocoles sont adaptés pour les interactions faiblement couplées que certains argumentent comme préférables pour concevoir des systèmes distribués robustes. Leur but est de concevoir des architectures orientées services, structurées en services communicants décrits de manière uniforme.

7 http://www.atnf.csiro.au/vo/rvs/
8 http://www.globus.org

A partir du constat que même les applications très spécifiques à un domaine partagent entre elles la nécessité de manipuler une infrastructure, GT4 fournit des services d'infrastructure qui implémentent la gestion du calcul, du stockage et des autres ressources.

- **Architecture GT4**

La figure 12 illustre les différents aspects du Globus Toolkit 4 constitué de trois ensembles de composants :

- ✔ Services (en bas à gauche) : gestion de l'exécution (GRAM), accès aux données (GridFTP, RFT, OGSA-DAI), gestion de la réplication (RLS, DRS), monitorage et découverte (Index, Trigger, WebMDS), gestion de la sécurité (MyProxy, Delegation, SimpleCA), gestion des instruments (GTCP) ; La plupart sont des services web mais certains (en bas à droite) sont implémentés dans d'autres langages et/ou protocoles.
- ✔ Trois containers sont réservés pour héberger les services des utilisateurs. Ils fournissent les mécanismes (compatible Web Services) fréquents lorsque l'on conçoit des services.
- ✔ Des bibliothèques clientes en C, JAVA et Python pour invoquer des opérations GT4 ou celles développées par les utilisateurs.

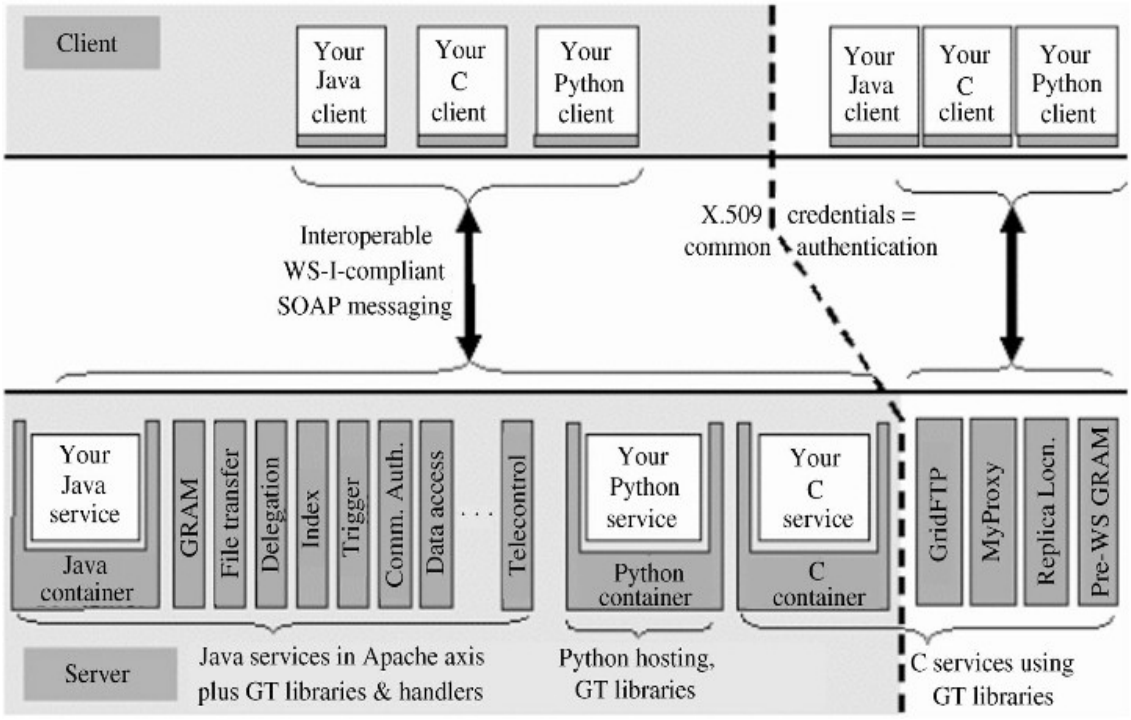

Figure 12 : Les composants GT4 et leurs interactions (les boîtes blanches sont les codes utilisateurs)

De par sa généricité, son architecture et ses services, Globus semble se prêter à une utilisation pour visualiser des données massives à distance. Pourtant peu de solutions sont citées. Parmi celles-ci une expérimentation retient l'attention [21] puisqu'elle met en oeuvre une connexion entre les Etats-Unis et la Hollande aux fins de coupler un composant de collaboration avec un composant de rendu haute résolution à distance. Le composant de collaboration rend et diffuse des images en faible résolution, calculées sur un sous échantillonnage des données. Il utilise VTK (extraction d'isosurfaces sur des données sous échantillonnées) et Chromium [8] pour paralléliser le rendu et

décomposer les images en sort-last afin d'expédier des flux vidéos séparés et compressés en H261. Le composant de collaboration est typiquement capable de rendre une image 3x2 pour atteindre 1056x576. Le composant de rendu haute résolution est un prototype de parallélisation de visualisation utilisant l'ensemble complet des données. Les pièces de l'image sont ensuite envoyées en streaming puis composées pour former l'image finale. Cela fournit aux utilisateurs finaux des images (2048x1536) qui s'affinent progressivement. Les calculs des images et leurs transferts sont réalisés via MPICH-G2 une implémentation de MPI-1 qui utilise Globus pour adapter MPI à la Grille. MPICH-G2 est à même d'utiliser des sockets parallèles via GridFTP. Cet exemple illustre bien que si Globus n'est pas explicitement un environnement de visualisation, il en facilite notablement la conception si celui-ci doit prendre en charge la visualisation distante, la parallélisation du rendu, la sécurité etc.

5. Approche FlowVR

Le développement d'applications de visualisation scientifique distribuées avec streaming et gestion de la latence rencontre des difficultés similaires à celles que l'on doit surmonter lors du développement des applications de Réalité Virtuelle (RV) ambitieuses qui incluent de nombreuses simulations et des interactions complexes. On doit dans les deux cadres, faire face à deux difficultés majeures :

- gestion ardue du génie logiciel où de multiples morceaux de codes, développés par différents acteurs, à différentes périodes, doivent être assemblés dans un cadre unique et fonctionner ensemble ;
- surmonter les limites du matériel en multipliant les unités de traitement (GPU, CPU,....) mais avec le désavantage majeur d'ajouter des difficultés supplémentaires liées à la parallélisation de tâches, pipeline ou de données.

Il existe de nombreux outils qui facilitent l'utilisation des machines parallèles actuelles (en général des architectures faiblement couplées de type cluster) pour la RV. Par exemple Chromium [8] porte la machinerie OpenGL sur un cluster de visualisation. D'autres outils proposent des diffusions afin de synchroniser de multiples copies d'applications plus ou moins parallèles [37]. Comme dans le cas de la visualisation distribuée, les outils sont multiples et tous sont utiles à divers titres. Il est simplement difficile de choisir parmi eux et de les combiner. On notera que le parcours dataflow strictement FIFO des applications classiques de visualisation scientifique est plus contraint et rend difficile la gestion de la visualisation distribuée sans ralentissement lié à des fortes synchronisations et un trafic réseau lourd. Les applications classiques de RV relâchent ces contraintes non en proposant une gestion fine de synchronisation mais en proposant d'extrapoler les comportements calculés à distance pour absorber la latence du réseau (estimation de position).

5.1 -Description générale

FlowVR propose un cadre pour le développement d'applications ambitieuses de RV [17]. Les applications sont fortement modulaires pour faciliter le développement et pour être aisément déployées sur les architectures des clusters de PC. FlowVR étend le cadre dataflow traditionnel en autorisant des communications et des synchronisations bien plus complexes que celles des API classiques parallèles. Pour autant, la sémantique des applications n'est pas forcément laissée sous la responsabilité des développeurs mais elle peut être gérée via des éléments pré-intégrés à FlowVR. Ainsi, il est possible d'utiliser FlowVR pour réaliser des codes non spécifiquement RV. Tout code parallèle sur cluster (et par extension, probablement sur la grille) peut tirer avantage de FlowVR,

soit pour ses capacités de génie logiciel et la vision globale qu'il apporte du cluster, soit, plus finement, lorsque la gestion des interactions, des framerates interactifs ou du pilotage des applications devient complexe.

5.2 -Architecture FlowVR

Une application FlowVR est composée de *modules* qui échangent des données via un *réseau FlowVR* (voir figure 13).

Un module est typiquement un code existant (éventuellement lui-même parallèle) modifié à la marge par l'ajout d'appels à des fonctions FlowVR. Un module s'exécute dans son processus, il y a donc peu d'effort à fournir pour transférer d'anciens codes vers FlowVR. Les modules n'ont pas à tenir compte directement du calcul des autres modules.

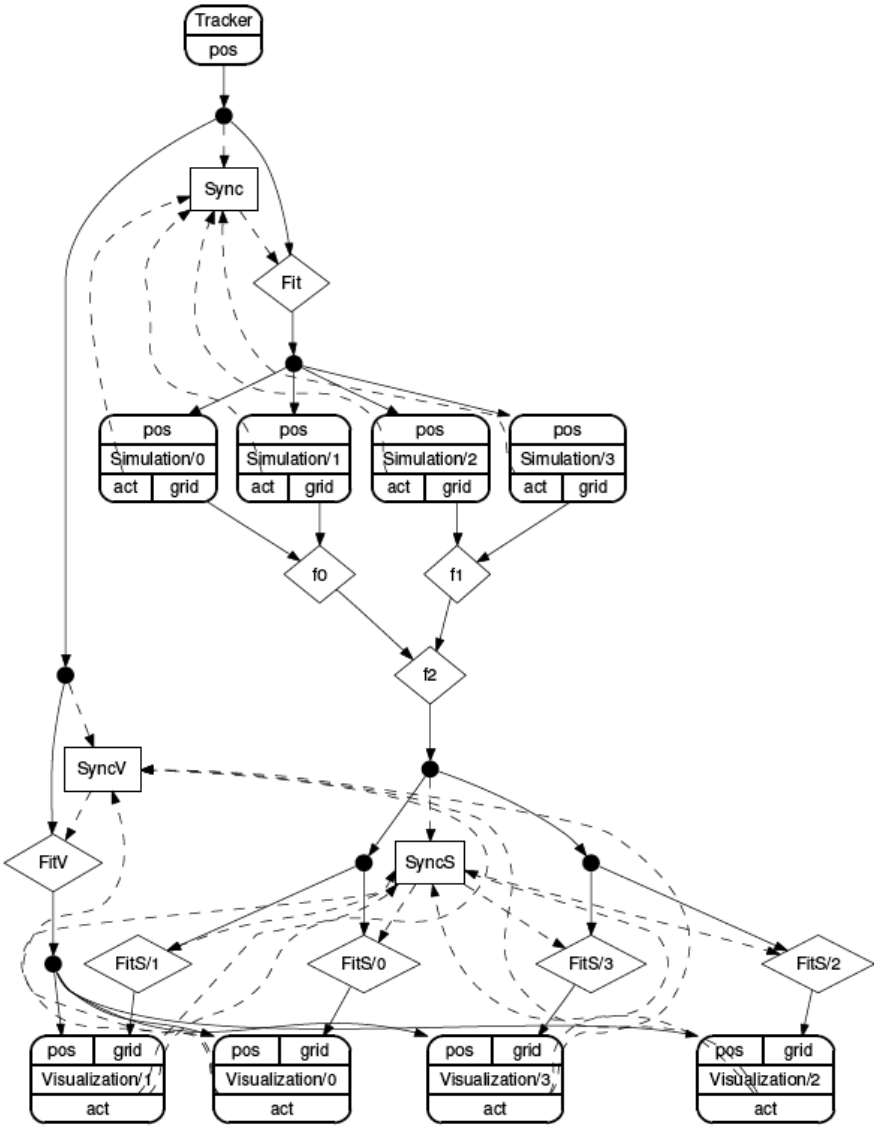

Figure 13 : Exemple d'un réseau FlowVR composé de liens entre des modules séquentiels ou parallèles

Ces derniers n'échangent leurs données et ne se synchronisent que via des *Démons FlowVR* qui

s'exécutent sur chacune des machines du cluster. Ce sont les démons qui gèrent les transferts de données entre les machines si nécessaire. Cette approche permet des déploiements différents et même des agencements multiples des modules de l'application FlowVR sans nouvelle compilation. Les connexions entre les modules peuvent être simples (fifo de type MPI par exemple) ou bien plus complexes : filtrage sur les données, prélèvement, estimation de position, découpe selon le cône de vision, communications collectives, etc. Ce contrôle fin permet de tirer avantage de la spécificité de l'application, de s'adapter à l'architecture matérielle tout en tenant compte des besoins réels de l'utilisateur en termes de latence et de taux de rafraîchissement. Cette expressivité doit probablement pouvoir offrir les mêmes avantages dans le cadre de la visualisation distribuée et distante. Une chaîne de visualisation distante doit pouvoir être transposée en graphe FlowVR (presque littéralement dans un premier temps) puis utiliser les mécanismes FlowVR et adapter les transferts de données à la bande passante du réseau, ou à d'autres critères tels que la pertinence des résultats d'un calcul ou bien la forte/faible demande en interactivité à un instant donné. Par exemple, un filtrage mettant en oeuvre un mécanisme de LOD en fonction d'informations externes qui proviennent de modules pourrait simplifier à la demande, les informations en sortie de simulations. Et dans le cadre de FlowVR ce mécanisme pourrait être développé indépendamment des codes de simulations et mis en oeuvre en fonction du type de déploiement.

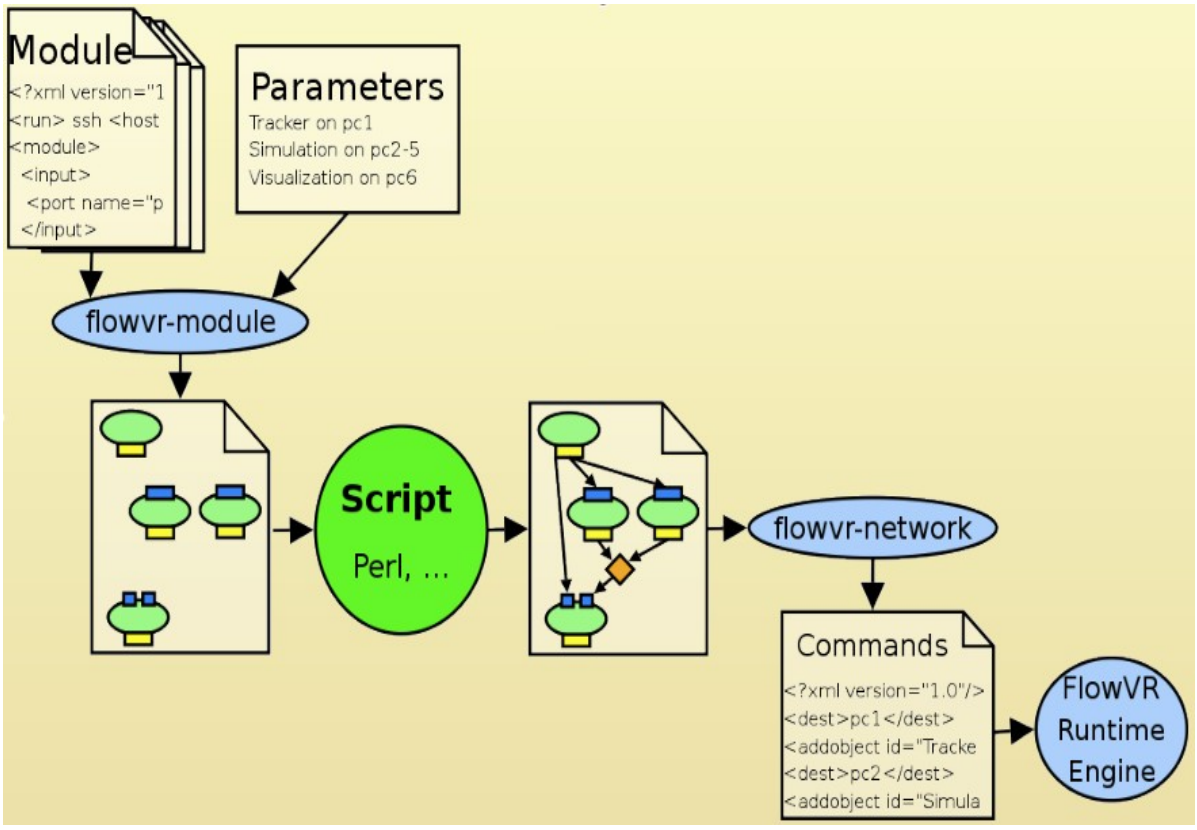

Figure 14 : Processus de création d'une application FlowVR

FlowVR est une suite complète d'outils pour développer les modules, les réseaux d'interconnexion, pour plaquer l'application complète sur un cluster, la lancer et pour contrôler son exécution. De plus il existe également des outils pour tracer et analyser l'exécution. Là encore ces outils devraient aider fortement le déploiement d'applications de visualisations distantes. La figure 14 illustre le processus de développement FlowVR depuis le code des modules jusqu'aux commandes générées à destination du démon FlowVR. Notons que le passage à un code FlowVR sur la grille nécessiterait

une adaptation dans la mesure où FlowVR n'offre pas les fonctionnalités propres à ce support comme la sécurité, la tolérance aux pannes, la découverte de service et l'équilibrage de charge.

Une extension de FlowVR permettrait d'exprimer et de déployer rapidement des pipelines distribués de traitement des données plus complexes et hétérogènes que ceux classiquement proposés dans les environnements modulaires de visualisation. Par ailleurs l'aspect « composants » de FlowVR devrait permettre d'intégrer, en fonction des besoins, les niveaux de détails dynamiques pertinents, qui sont au coeur des approches qui se préoccupent de la visualisation des gros volumes de données. Enfin la conception typiquement FlowVR de code sous forme de modules réactifs vis-à-vis d'événements externes ouvre la voie vers le pilotage et le travail collaboratif et pourrait constituer une approche par composants pertinente pour la visualisation distante et ses perspectives.

IV. Systèmes autonomes

Il existe nombre de systèmes de visualisation distribuée. La plupart sont basés à l'origine sur un système de visualisation modulaire lui-même fondé sur la notion de modèle de flot de données [31]. Dans ces applications, le processus entier de visualisation est scindé en parties, ou encore modules, qui sont inter-connectées via un réseau. Les données suivent le réseau et sont modifiées progressivement jusqu'à être transformées en images visionnées par le scientifique (figure 15).

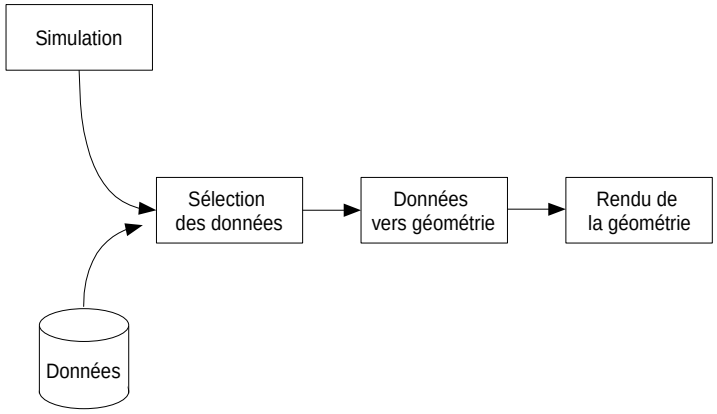

Figure 15 : Pipeline graphique des systèmes de visualisation modulaires

Si ce scénario a été créé initialement pour visualiser des données enregistrées, il est aussi adapté aux données générées à la volée par des simulations (voir figure 15). La plupart des applications de visualisation sont conçues sur ce modèle. Chronologiquement, un des premiers exemples est AVS (version 1) [2]. Typiquement, il fournit un éditeur visuel afin de construire les applications en connectant entre eux des blocs de traitement. Il est probable que l'expressivité et la facilité d'utilisation de cette métaphore de conception soient à l'origine du succès des environnements de visualisation modulaires. De par les notions de réseau et de modules de traitement, il devient naturel de passer de systèmes mono-machine à des applications distribuées (AVS, VTK, IRIS Explorer, OpenDX...) [3; 33; 60]. Ce sont principalement ces applications, au fonctionnement relativement autonomes, qui sont décrites dans cette partie.

1. Amira

Amira est un système orienté objet de visualisation basé sur Open Inventor [1; 62]. A la différence des autres systèmes de visualisation discutés ici, il n'est pas basé sur le modèle à flot de données. Au lieu de cela les objets sont persistants et accessibles via les interfaces C++ des classes données. Pour cette raison, il est mal adapté à la distribution du travail en mode pipeline. Par contre il est extensible par l'utilisateur et le mode de conception est basé sur une métaphore proche de celle adoptée par les environnements modulaires de visualisation soit des modules connectés entre eux. Toutefois, on peut adapter Amira à la visualisation distante en écrivant des modules de communication avec une simulation (pourquoi pas en utilisant une libraire de communication et de synchronisation telle que FlowVR [17]) ou même en dialoguant avec une autre instance d'Amira (méthode adoptée dans la version Amira VR à rendu parallèle reprenant ainsi une partie des

fonctionnalités et des solutions du logiciel de RV parallèle Netjuggler [37]).

2. Covise

Covise [12] est un environnement de travail collaboratif développé à l'origine par des chercheurs du laboratoire HLRS de l'Université de Stuttgart. Les dernières versions de ce logiciel sont devenues commerciales, elles sont développées par la société VISENSO après l'arrêt des activités de Vircinity.

2.1 -Approche

L'idée de COVISE est de proposer un environnement de travail collaboratif distribué qui permette à des chercheurs et des ingénieurs de travailler à distance sur des modélisations éventuellement complexes. COVISE est conçu pour permettre à l'ensemble des participants de visualiser et d'interagir sur les modèles en temps réel.

2.2 -Architecture

COVISE se base sur une représentation centralisée et propriétaire du modèle d'une part et sur un protocole optimisé de mise à jour de cette représentation par les différents acteurs d'autre part. Le schéma de la figure 16 ci-dessous donne une idée de l'architecture de COVISE.

Chaque machine participant à la session de travail collaboratif possède un espace mémoire réservé à COVISE ainsi qu'un processus particulier appelé "courtier" (Broker) chargé des mises à jours des modèles manipulés. Une machine particulière joue le rôle de serveur (la machine 1 dans la figure 16). Elle maintient à jour le modèle manipulé (l'objet rouge de la figure). Chaque machine participante conserve une copie du modèle. Le courtier du serveur va diffuser les informations de mise à jour à l'ensemble des courtiers des autres machines qui seront chargés d'effectuer les changements nécessaires. Chaque machine peut ensuite manipuler sa propre version du modèle en extrayant par exemple une iso-surface. Ce travail sera effectué en local mais n'aura pas de répercussions sur les autres participants.

Dans cette architecture, il est clair que le serveur stocke les données issues de la simulation et effectue la mise en forme de ces données dans le modèle de COVISE. Chaque client doit lui aussi stocker le modèle COVISE et il doit aussi avoir les ressources nécessaires pour mettre à jour ce modèle. C'est aussi le client qui est chargé du calcul de rendu (scénario 2). Ce type d'architecture permet donc de s'affranchir des contraintes de latence du réseau pour toutes les manipulations locales du modèle mais nécessite des ressources en mémoire et en calcul sur les clients. Le serveur doit aussi être capable, en plus de la gestion du modèle, de transmettre efficacement les mises à jour à tous ces clients.

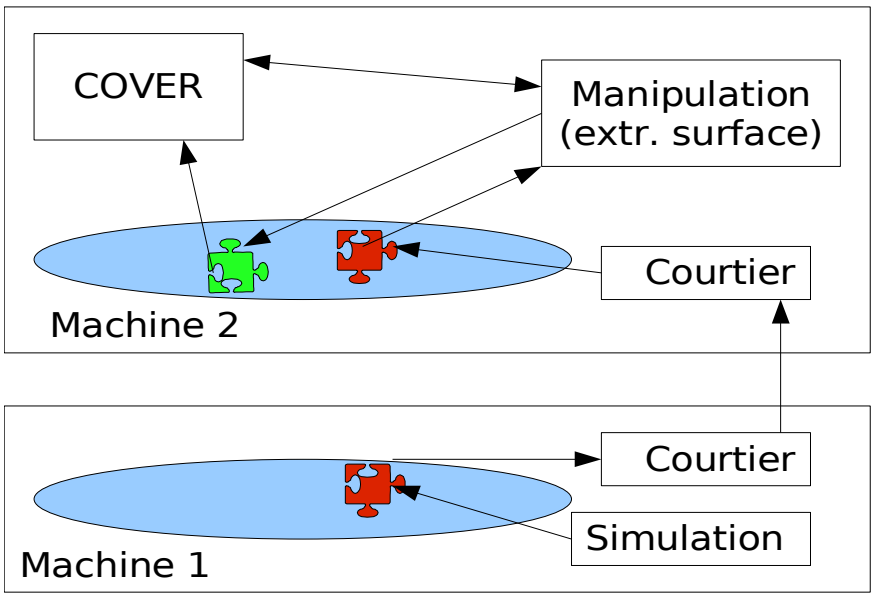

Figure 16 : Architecture de COVISE

3. EnSight

EnSight [14] est un logiciel commercial de visualisation scientifique développé aux Etats-Unis par la société CEI. EnSight est très largement utilisé par de grands organismes de recherche américains comme la NASA ou le laboratoire de Los Alamos (LANL) et des organismes français tels que EDF, SNECMA/SAFRAN ou le CEA/DAM.

3.1 -Approche

Ce logiciel se décline en plusieurs versions allant de EnSight personal monoposte, à EnSight DR qui propose des fonctionnalités pour le traitement de données et le rendu distribué. Il semble que la partie distribuée d'EnSight a été largement développée au sein d'une collaboration entre CEI et Los Alamos.

3.2 -Architecture

L'architecture d'EnSight peut aller du simple mode client/serveur à une architecture complètement distribuée où la source de données et le rendu sont distribués comme illustré figure 17. EnSight permet notamment de distribuer uniquement la partie cliente ou la partie serveur en fonction des besoins de l'application et des ressources matérielles disponibles. Dans cette architecture, la partie serveur va jusqu'au calcul de la géométrie qui sera envoyée à la partie client qui elle s'occupera du rendu et de l'affichage.

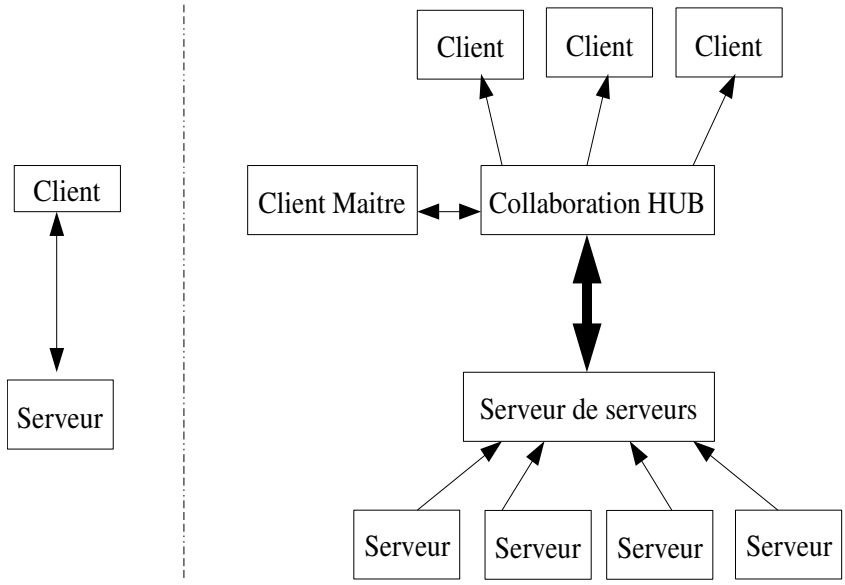

Figure 17 : Différentes architectures d'EnSightDR

L'architecture distribuée de la partie serveur s'appuie sur un serveur de serveurs (abrégé en SOS) dont le rôle principal est de coordonner le travail de chacun des serveurs. Il peut par exemple prendre en charge la répartition des données entre chacun des serveurs. C'est aussi le SOS qui prend en charge la recomposition de la géométrie calculée par chacun des serveurs et la répartition vers les différents clients en essayant d'équilibrer les charges.

Dans la partie cliente, plusieurs scénarios sont envisagés en fonction du matériel et des besoins d'affichage.

- Le client s'exécute sur une machine parallèle à mémoire partagée. Dans ce cas EnSight utilise plusieurs threads pour accéder aux différentes cartes graphiques de la machine qui fait le rendu.
- Le client s'exécute sur une machine parallèle à mémoire distribuée (type cluster de PC). Dans ce cas la partie cliente d'EnSight est distribuée. Tous les clients, à part le client-maître, sont passifs du point de vue de l'interaction. Le client-maître par contre a pour rôle principal de prendre en charge les interactions de l'utilisateur. Ces interactions sont diffusées via le HUB de collaboration vers les différents clients. Le HUB de collaboration et le SOS échangent des informations afin de contrôler et d'effectuer la répartition de la géométrie entre les différents clients. La distribution des données peut aller de la simple réplication des primitives graphiques à une réelle distribution avec équilibrage de charge. Le mode *réplication* ne permet pas d'afficher plus de données que ce qu'on pourrait faire sur un seul client, mais il permet de gérer simplement et efficacement l'affichage multi-écran. On peut aussi confier à Chromium la gestion du rendu parallèle pour le rendu multi-ecran.

  Dans le cas d'un affichage sur un seul écran la recomposition de l'image produite par les clients EnSight peuvent se faire soit par l'intermédiaire du « CEI Parallel Compositor », soit

en utilisant Chromium.

Toutes les communications entre les différents acteurs de l'application EnSight se font via TCP/IP. Peu d'informations sont données sur le protocole utilisé, mais il semble que la connexion entre le SOS et le HUB de collaboration est le point sensible de cette architecture car toute la géométrie passe par ce lien. Robert J. Kares du laboratoire de Los Alamos [15] donne cependant un certain nombre d'indications sur les possibilités offertes par cette architecture dans le cadre de très gros volumes de données. Par contre, aucune information en termes de frame rate ou de latence de l'interaction n'est donnée. Parmi les exemples cités par Kares, on trouve la visualisation à Los Alamos, du résultat d'une simulation ayant générée 21TB de données au LANL à Livermore à plus de 1600 km de distance. La partie serveur contenait 50 serveurs et la partie cliente était "réduite" à une SGI Origine 2000. Les deux sites étant reliés par un réseau spécifique de plusieurs Gigabits. Cette architecture est maintenant utilisée en production au sein du laboratoire de Los Alamos pour visualiser les données générées par divers codes de calcul. L'objectif actuel est de remplacer la machine SGI par une grappe de PC sous linux en utilisant les fonctionnalités client réparti d'EnSight.

4. ParaView

ParaView est une plate-forme de visualisation distribuée destinée à la manipulation efficace de grands volumes de données qui repose sur VTK pour les algorithmes de traitement et de rendu. Elle est développée par Kitware sur conception initale de Jim Ahrens du LANL. Le laboratoire SNL a ensuite beaucoup contribué à ParaView.

4.1 -Description générale

ParaView [42] comme illustré par la figure 18, est construit au dessus de VTK, lui-même implémenté en C++ et fournissant une plate-forme de visualisation parallèle et/ou distribuée. Afin de fournir une solution générique aux applications de visualisation, ParaView propose différents modes d'exécution du calcul du rendu et de l'affichage. Le mode le plus simple « stand-alone » consiste en une exécution de ParaView en local sur une machine parallèle ou un cluster. De l'autre côté ParaView propose un mode basé sur une séparation du serveur en un serveur de données et un serveur de rendu plus un client qui procède à l'affichage. Dans ce cas, l'approche adoptée est l'approche client/serveur avec une transmission entre le serveur de rendu et le client soit de la géométrie en accord avec notre scénario 3 soit des images résultantes en accord avec notre scénario 1.

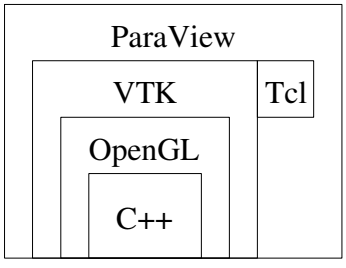

Figure 18 : Organisation de ParaView

Dans le cadre d'une transmission de la géométrie, ParaView fournit un algorithme optimisé de redistribution des données des *M* noeuds du serveur de rendu aux *N* noeuds du client qui compléteront le rendu avant de procéder à l'affichage.

4.2 -Infrastructure

Pour exprimer ces différents modes de visualisation, ParaView s'organise autour

- ✔ d'un module appelé CSS pour Client/Serveur Streaming qui gère l'invocation des méthodes fournies par le serveur qu'il soit local ou sur un site distant,
- ✔ d'un gestionnaire de serveur au dessus des modules CSS qui prend en charge l'interconnexion des serveurs au client et qui gère la création des objets VTK, les requêtes sur les différents objets, etc.
- ✔ d'une hiérarchie de modules implémentant les différents modes de rendu.

Cette hiérarchie de modules illustrée par la figure 19 a pour racine le module abstrait Render Module (RM) qui joue le rôle d'interface avec l'ensemble des objets de rendu VTK. Le module Simple RM (SRM) en hérite et englobe les fonctionnalités élémentaires de rendu telles que collection de visuels, lumière, caméra, interaction. Le seul module fils de SRM est LOD RM qui ajoute des fonctionnalités de gestion de niveaux de détail pour accélérer l'affichage pendant les phases d'interaction.

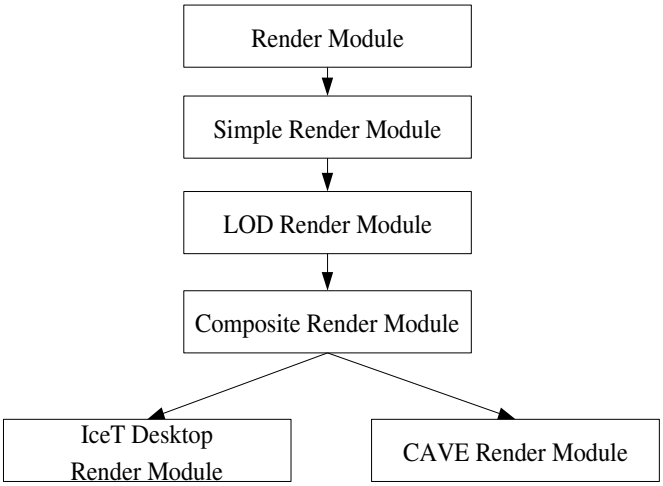

Figure 19 : La hiérarchie de modules de ParaView

Ensuite le module Composite RM ajoute des outils de rendu parallèle dans le contexte client/serveur, notamment les communications inter-processus et le LOD parallèle. C'est ce module qui implémente l'algorithme optimisé de transfert de géométrie entre le serveur de données et celui de rendu ainsi qu'un algorithme sort-last pour l'affichage multi-écrans. Le module Composite étant abstrait, les implémentations sont fournies par les sous-modules : IceT Desktop et IceT RM d'une part et CAVE de l'autre. Les premiers utilisent la bibliothèque IceT [29; 30] pour implémenter le module Composite et proposent une facilité de compression à travers l'algorithme SQUIRT (Sequential Unified Image Run Transfert). Le second fournit une implémentation CAVE du

Composite.

Ainsi l'architecture fortement distribuée de ParaView et l'abstraction du rendu à travers les différents modes de son exécution font que cet environnement est a priori parfaitement adapté à la visualisation distante en général, et en particulier pour l'affichage multi-écrans distant.

5. VisIt

VisIt [55] est un outil d'analyse et de visualisation de gros volumes de données développé par le Lawrence Livermore National Laboratory ; VisIt est disponible en open source. C'est un système autonome mais qui permet l'utilisation, sous forme de plug-ins, d'outils personnels à l'utilisateur.

5.1 -Approche

Cet outil se base sur une approche Data Flow similaire à celle d'AVS ou VTK. Plus largement prévu pour de la visualisation classique de données, il permet la visualisation distante en intégrant les fonctionnalités nécessaires pour l'exécution du viewer et de l'interface graphique intégrés à VisIt sur un poste client distant.

5.2 -Architecture

VisIt est un environnement complet de développement d'applications d'analyse et de visualisation de données scientifiques. C'est à la fois un ensemble de bibliothèques graphiques basées sur VTK et un environnement de programmation sur le modèle Data Flow. L'architecture générale est illustrée par la figure 20. Ainsi à partir de l'interface graphique (GUI) sur le client, il est possible de se connecter à distance sur le serveur VisIt et de lui indiquer les données sur lesquelles on souhaite travailler. L'application est alors lancée sur le site distant comprenant à la fois la base de données voulue et les machines de calculs ou graphiques. Les opérations sont exécutées au sein du « compute engine » qui peut être parallèle et les résultats sont envoyés au viewer local pour l'affichage. Il est également possible de visualiser des données générées en temps réel par une simulation en instrumentant le code de cette dernière. Le « compute engine » a alors la charge d'accéder aux résultats de la simulation sans écriture intermédiaire de fichiers de stockage. Toutes les communications sont basées sur des sockets et sur RPC pour l'invocation distante des méthodes

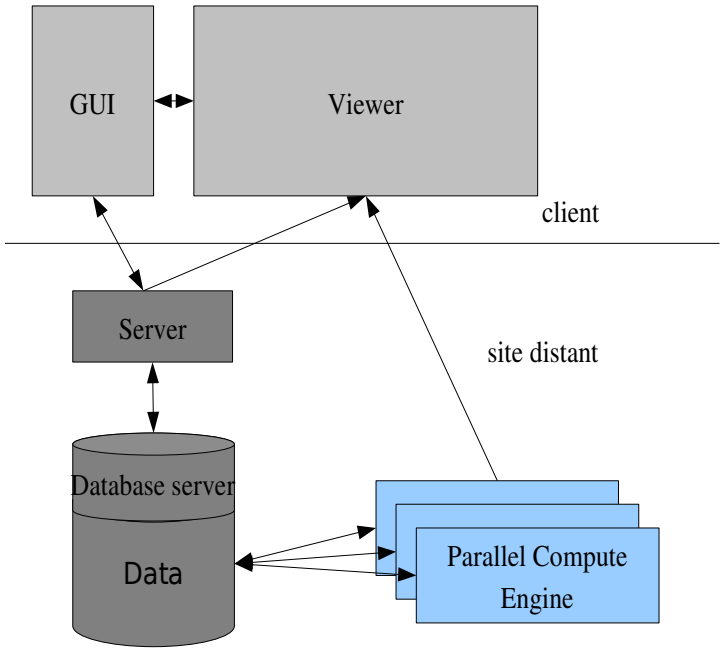

Figure 20 : Architecture générale de VisIt

Un important effort a été fait sur l'optimisation des algorithmes de pré-rendu et de rendu. Essentiellement, elles s'appuient sur des techniques de rendu de scènes. Elles respectent donc soit le scénario 2 lorsque les commandes OpenGL sont directement transmises à la carte graphique du client soit le scénario 3 lorsque sur le site distant, via Mesa par exemple, le rendu est effectué en parallèle et que le résultat recomposé est envoyé au client après optimisation des objets qui font partis ou non de la scène.

VisIt est donc très complet et constitue un environnement de développement autonome pour toute application de visualisation de données scientifiques. Pour la lecture des données, VisIt est compatible avec de nombreux formats dont les principaux sont le format VTK et le format SILO (développé par le LLNL pour les grilles 2D ou 3D et le stockage distribué) mais également GIS, FITS ou encore le format propriétaire d'Ensight Gold. Il supporte également Python comme langage de script permettant aux utilisateurs de générer des scénarios complexes de visualisation de leurs données (animations, ...). Enfin, puisque toutes les fonctionnalités sont implémentées sous forme de plug-ins, il est possible d'enrichir VisIt de ses propres développements. Par conséquent, VisIt est principalement utilisé comme logiciel intégré à interface applicative prédéfinie et langage de script.

6. Covisa/CovisaG/gViz

6.1 -Covisa/CovisaG

Le modèle à flot de données a été conçu initialement pour un seul utilisateur, mais son principe

permet, avec quelques extensions, de l'adapter à plusieurs utilisateurs qui inter-connectent leurs pipelines graphiques de façon à ce que les données puissent voyager d'un utilisateur à l'autre. Une étape supplémentaire est franchie si on donne les moyens aux scientifiques de piloter les simulations en modifiants leurs paramètres, là encore à la volée (voir figure 21).

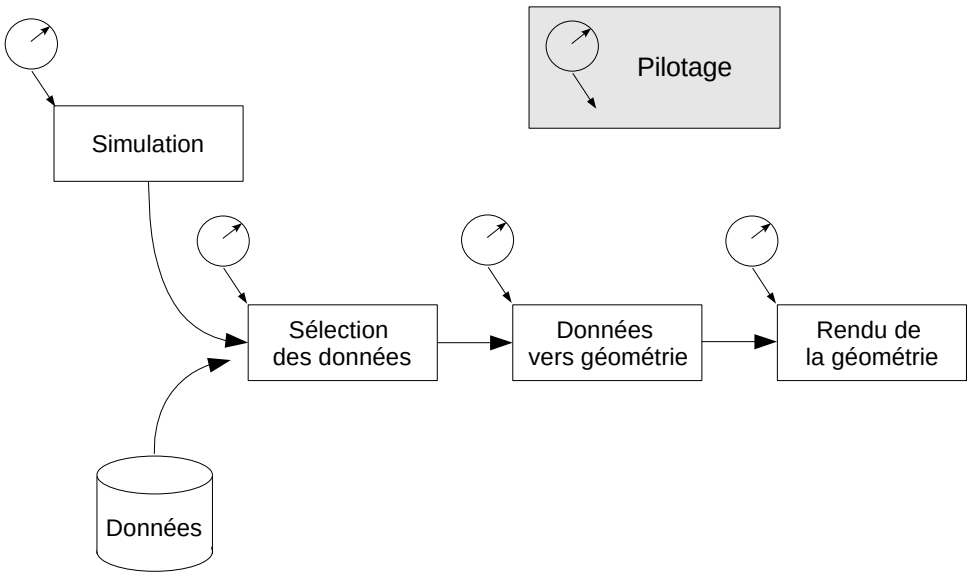

Figure 21 : le pipeline du flot de données piloté à chaque étape

Dans ce sens, COVISA [11; 33] est une extension d'IRIX Explorer qui offre, en outre, la possibilité de rendre collaborative la visualisation et le pilotage. De plus, l'émergence du calcul sur la Grille conduit à adapter les solutions de visualisation distribuée. D'où une première extension avec CovisaG dont le but est d'utiliser Globus pour les communications de COVISA . Ce dernier projet s'est, semble-t-il, vite métamorphosé en gViz.

6.2 -Projet gViz : Intergiciel de visualisation pour les e-Sciences

Le projet gViz [19; 25] a été créé par le UK e-Science Core Programme, et a deux objectifs principaux. En premier lieu, le projet vise à adapter à la grille deux systèmes existants de visualisation, IRIS Explorer et pV3, de sorte que les outils de visualisation soient disponibles dès que possible pour des utilisateurs des grilles informatiques [26; 43]. En second lieu, gViz développe la réflexion à plus long terme sur la visualisation distribuée et collaborative, avec l'emploi du XML pour décrire des données à visualiser ainsi que les programmes de visualisation eux-même. Autour d'IRIS Explorer un démonstrateur a été conçu de façon à distribuer à travers la grille un réseau de modules commandé via un ordinateur de bureau. Ceci est réalisé d'une façon sécurisée en utilisant le logiciel Globus pour remplacer le peu sûr mécanisme de rsh. Les capacités de collaboration offertes par COVISA sont immédiatement disponibles, et ainsi nous obtenons un cadre pour la visualisation collaborative distribuée sur Grille. Une application importante de ceci est le pilotage lorsque le modèle de simulation fonctionne sur un serveur distant, mais est commandé à partir d'un ordinateur de bureau. La version grille de pV3 est créée en remplaçant ses communications basées

sur PVM par un mécanisme de service web qui utilise le paquet de gSOAP [24]. Le calcul distribué fourni des données via des offres de service web pour la visualisation et les serveurs pV3 se relient à celui-ci en tant que clients de service web. Notons que gSOAP fournit une implantation C/C++ efficace de service web, faisant usage de SSL et de GSI (Grid Security Infrastructure) aux fins de sécurité.

6.3 -Modèle en couches

Le modèle gViz est conçu autour de trois couches :

- ✔ une couche conceptuelle ;
- ✔ une couche logique ;
- ✔ une couche physique.

La couche conceptuelle décrit le réseau de flot de données en termes de processus abstraits indépendants des choix logiciels et matériels. La couche logique plaque sur ces processus les choix logiciels. La couche matérielle répartit les processus sur la grille en décrivant la localisation des logiciels qui vont construire l'application globale de visualisation collaborative (voir figure 22).

La couche conceptuelle décrit comment les données numériques doivent être transformées en données visualisables. La collaboration est décrite en associant des réseaux de flots de données à des utilisateurs et en spécifiant les liens entre ces réseaux. Le réseau de flot de données peut être modifié de façon dynamique en fonction de l'évolution des besoins de visualisation au cours d'une analyse.

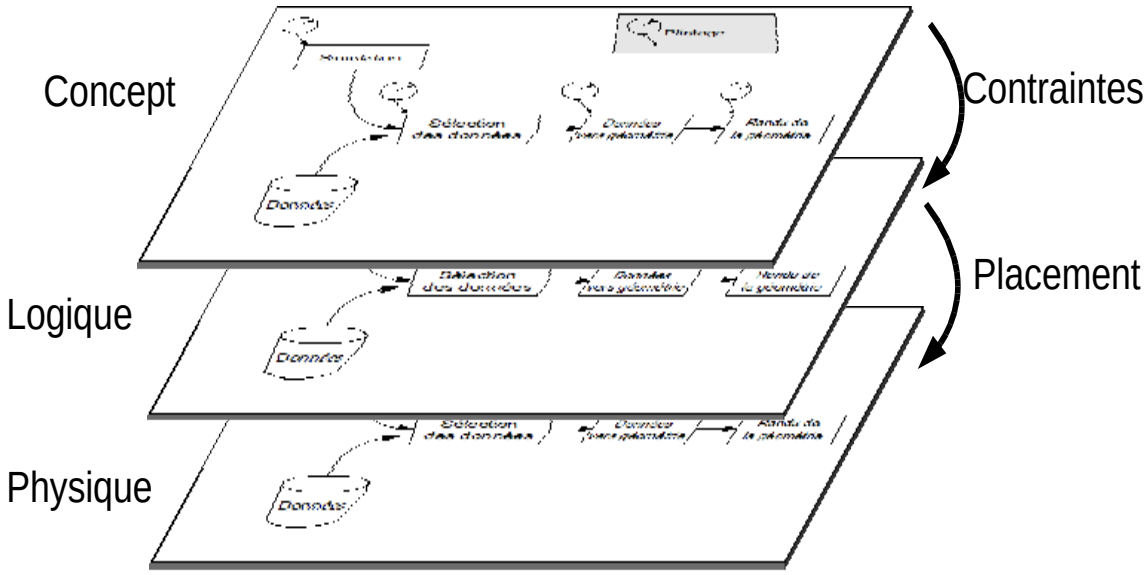

Figure 22 : Modèle de référence en couches

L'utilisation du langage XML pour décrire cette couche doit conduire à une meilleure inter-opérabilité des systèmes de visualisation et à un développement plus pertinent des nouveaux logiciels de visualisation (qui pourraient être développés pour inter-opérer avec ces nouveaux standards). L'avantage d'utiliser une couche conceptuelle (en XML) pour décrire les programmes de

visualisation (par exemple la façon dont les pipelines graphiques sont construits par les environnements modulaires de visualisation) est d'ouvrir la voie à des échanges de visualisations entre les utilisateurs de différents systèmes. On notera toutefois que l'éventuel succès d'une telle spécification formelle est lié à l'émergence progressive d'une ontologie de la visualisation bien décrite et partagée (ontologie = modèle général ou taxonomie des types de données et des méthodes et techniques de visualisation [19]).

La couche logique peut être réalisée de différentes façons. Par exemple, chaque entité peut correspondre à un module au sein d'une environnement modulaire de visualisation, ou à une fonction de bibliothèque ou à une combinaison des deux. Si le principe de cette couche est de ne pas préciser la localisation des exécutions, le choix des logiciels introduit des contraintes sur les ressources requises. De plus ces contraintes ne sont pas statiques puisque de nouvelles données peuvent être mises en jeux lesquelles nécessitent une migration logicielle pour satisfaire des contraintes de performance.

Au final la couche physique interprète la spécification de la couche logique en terme de ressources sur un environnement « Grille » particulier. Ici non plus les choix ne sont pas statiques.

6.4 -Mise en pratique du modèle

Chaque couche du modèle a besoin d'être représentée pour pouvoir exprimer les instances de réseau mais aussi les transformations d'une couche dans l'autre. Si la plupart des environnements de visualisation modulaire à base de flots de données privilégient une présentation par diagrammes, gViz explore un moyen de représentation à la fois diagrammatique mais aussi basé sur un langage « skML ». La description des modules gViz ressemble à celle des modules FlowVR, avec des définitions de ports d'entrée et de sortie. Un souci particulier des concepteurs de gViz a été de proposer des outils généraux tout en les illustrant avec de véritables environnements de visualisation. C'est dans cet esprit qu'ils ont proposé un module IRIS Explorer qui lui permet de lire les fichiers skML pour lancer des sessions collaboratives COVISA. Un outil inverse permet de générer du skML pour exprimer des sessions IRIS Explorer. Le concept est annoncé comme transférable à VTK [60], Open DX [38], ou tout environnement modulaire de visualisation.

Le « g » de gViz signifie « Grille » et, hormis le souci de généralité, c'est le principal apport de cet environnement. Ce dernier autorise l'allocation de modules en local mais aussi à distance sur la grille. De façon évidente, c'est particulièrement utile pour les simulations qui génèrent les données, mais aussi pour les calculs intensifs de visualisation tels que les extractions d'iso-surfaces sur des données massives. D'autres environnements modulaires de visualisation implantent ce type de distribution mais gViz apporte la technologie « grille » en portant l'effort sur la sécurité, l'authentification, l'identification et la découverte de ressource. Par exemple IRIS Explorer se contente du mécanisme de RSH pour les appels distants, alors que gViz privilégie SSH ou Globus.

6.5 -Bibliothèque gViz

La bibliothèque gViz est composée de deux parties : une pour le pilotage des applications qui permet d'appareiller le code des scientifiques, l'autre pour autoriser les connexions entre les composants généraux et les codes de visualisations. L'implantation des mécanismes de communication vise à minimiser les interruptions de la simulation. L'architecture fournit des

processus légers pour recevoir les changements sur les paramètres de pilotage et pour les conserver dans une queue jusqu'à ce qu'ils soient requis par la simulation. De même le système fournit des processus légers en guise de serveurs pour les paramètres à destination des clients connectés et d'autres processus légers pour fournir les données calculées. La seconde grande fonctionnalité de la bibliothèque est de fournir des routines d'API pour les clients afin qu'ils les utilisent pour créer des communications avec la simulation. L'apport de gViz, vis-à-vis d'autres environnements de visualisation de données autorisant le pilotage, est de fournir les moyens d'externaliser la simulation vis-à-vis du processus de flot de données (voir figure 23).

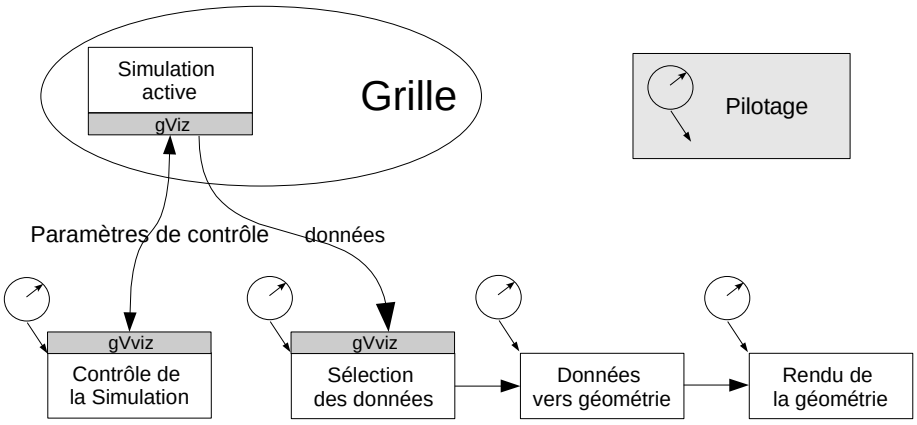

Figure 23 : Utilisation de gViz pour piloter et visualiser des données depuis une simulation exécutée sur la grille

Les communications entre les simulations et les codes clients peuvent être gérées de différentes manières. Ce sont les concepteurs des simulations qui choisissent les modes de communication en choisissant les méthodes. Le moyen le plus simple est d'utiliser les sockets Unix mais cela devrait être réservé à un environnement sécurisé. L'environnement offre une interface avec les services web via la bibliothèque gSOAP. Là où l'accès est réservé à certains mais où l'environnement est ouvert, des sockets GLOBUS sont fournis. Une version des gSOAP utilise un module GSI afin de garantir l'authentification. En outre il est possible d'adjoindre des moyens d'encryptage pour sécuriser les transferts de données.

La bibliothèque gViz contient également les outils gVizDS et gVizProxy. Le premier est un « directory service » qui collecte l'existence et les modalités de connexion des simulations disponibles. Un système gViz peut ainsi trouver les simulations en cours d'exécution en contactant un ou plusieurs serveurs gVizDS. Une fois la simulation découverte elle peut être directement contactée. GvisDS est fourni sous la forme d'un ensemble de services web accédé via SOAP.

Une des difficultés bien connue lorsqu'on doit accéder à une ressource de calcul sur la Grille est qu'elle est souvent localisée sur un cluster configuré en réseau privé. Il est alors difficile de communiquer directement avec une machine ne disposant que d'une adresse privée. L'outil

gVizProxy réalise un pont entre processus interne et processus externe. Lorsqu'une application de visualisation contacte gVizDS pour trouver une simulation, celui-ci retourne les modalité de contact de gVizProxy au travers duquel il est possible d'accéder à la simulation. La visualisation se connecte alors à gVizProxy comme si elle accédait à la simulation et gVizProxy transfère les données de l'un à l'autre.

Les moyens fournis par gViz permettent de réaliser entièrement à distance le déploiement et le pilotage de la simulation et du processus de visualisation pour ne gérer en local que les paramètres de pilotage, la définition du pipeline et le rapatriement final des images rendues. C'est le scénario de la figure 24. Ce schéma correspond à un démonstrateur où IRIS explorer est utilisé comme front-end pour lancer à distance un code de simulation et un pipeline de traitement visuel et de pilotage. Il est alors possible de se déconnecter pour un temps de la simulation avant d'y revenir à sa guise pour récupérer les images courantes et ajuster les paramètres de visualisation.

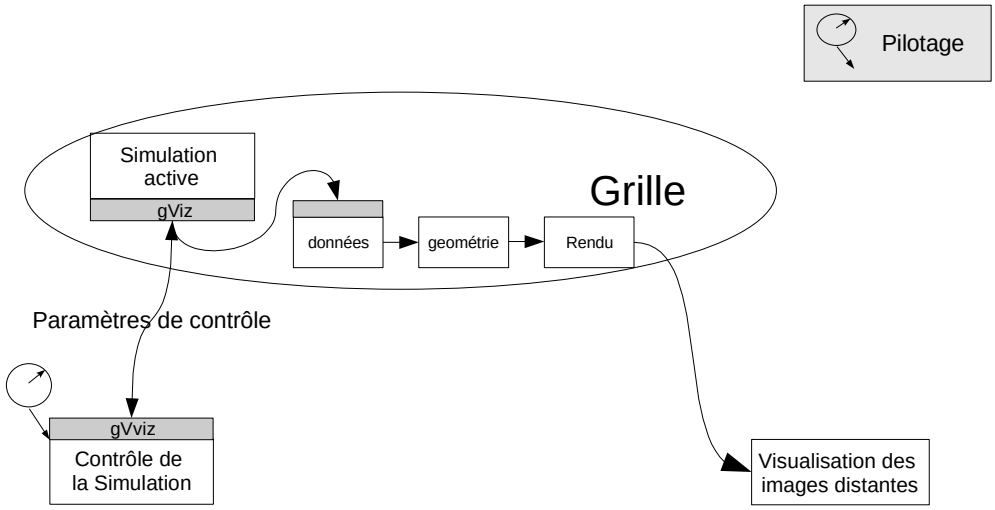

Figure 24 : Connexion de composants de simulations et de visualisations sur la grille

Le projet est open source (dernière release : gViz 1.1 août 2005) et est poursuivi via le « Integrative Biology Project » qui est actif (dernier rapport datant de 2006).

7. DIVA[9]

Le but initial de ce projet est de promouvoir la définition et l'émergence d'un « standard virtuel » pour la visualisation scientifique haute performance. Le projet est issu du Département d'Energie Américain (DOE) qui a travaillé sur les algorithmes et l'infrastructure de visualisation après avoir constaté que les produits commerciaux ne répondaient pas à ses besoins. Le projet se perçoit en tant que processus complet associant les partenaires dépositaires des besoins en visualisation (en l'occurrence les projets SciDAC financés par le DOE) jusqu'à la communauté de la recherche sur ce domaine précis. Ces recherches portent sur les algorithmes propres à la visualisation distante ainsi que sur les infrastructures nécessaires aux déploiements effectifs.
Les promoteurs de DIVA (le DOE) dressent le constat qu'en dépit d'années d'efforts et de

9 http://www-vis.lbl.gov/Research/DiVA/index.html

démonstrations réussies de visualisation distante, les utilisateurs utilisent de façon prédominante des outils en local sur les stations. Au mieux ce sont des connections via X11 sur des outils distants séquentiels. Pourtant il n'existe pas d'autre issue que la visualisation distante/parallèle pour gérer les données actuelles. Mais les outils offrent des fonctionnalités fractionnaires et les efforts en matière de visualisation distante ne le sont pas moins : certains outils sont Open Source (Parallel VTK, OpenDX), d'autres sont des environnements commerciaux (CEI EnSight, AVS Express) enfin certains sont totalement intégrés (VisIT, Visapult, Terascale Browser).

Les promoteurs de DIVA insistent sur le manque de généralité des outils existants et surtout sur le fait que ces architectures n'offrent pas de solution exhaustive pour les architectures parallèles et distribuées actuelles. Comment les faire inter-opérer? En tout état de cause, uniquement avec une architecture commune créée à cette fin et conçue pour susciter l'adoption des utilisateurs. Ils proposent donc de construire une architecture rigide induisant de bonnes pratiques de conception. Le système devrait fournir les moyens de décharger les développeurs des tâches domestiques pour qu'ils se concentrent sur les concepts.

DIVA devrait :

- ✔ être modulaire pour intégrer les développements communautaires ;
- ✔ supporter la découverte de ressources ;
- ✔ supporter l'analyse de performance et l'équilibrage de charge en conséquence ;
- ✔ supporter le streaming, l'out-of-core et les niveaux de détails ;
- ✔ découpler les modules de traitement et les codes de présentation/GUI ;
- ✔ intégrer des modèles internes robustes ;
- ✔ reprendre les caractéristiques essentielles des outils de la communauté comme OpenDX, AVS et VTK ;
- ✔ encoder des structures de données de base en visualisation et science (décomposition de domaine, représentation hiérarchique, LOD...) afin de mettre fin à la balkanisation des formats et des données.

Le projet par ces ambitions et son analyse ressemble à gViz. Par contre il semble plutôt en gestation et peu dynamique avec des workshops datant de 2003 puis une présentation au LBL's Distributed Systems Department en nov 2004.

8. SciRun/SciRun2

Le SCI Institute de l'université de l'Utah a pour thème de recherche général le calcul scientifique incluant les aspects visualisation et traitement d'images. Dans ce cadre, il est à l'origine du développement de SCIRun [45], PSE (Problem Solving Environment) dédié aux simulations numériques et qui intègre des outils pour la visualisation. Nous avons vu que les premiers pas vers la visualisation distante ont donné lieu à un prototype appelé Semotus Visum qui a permis un travail sur la répartition du pipeline graphique. Désormais cet aspect est abordé d'une manière plus générale avec le développement de SCIRun2 [46] qui est une approche par composants conforme au CCA.

8.1 -SciRun

SCIRun est défini comme un environnement de programmation pour le calcul scientifique et la visualisation. Il se définit comme :

- ✔ un ensemble de bibliothèques qui fournissent diverses implémentations de méthodes numériques (solvers, ...), de méthodes de pré rendu et de rendu,
- ✔ une infrastructure qui permet la création et la gestion de l'application via une interface graphique. Une application est alors conçue comme un canevas de modules connectés entre eux via leurs ports selon le modèle data flow,
- ✔ un ensemble de ponts avec des bibliothèques très utilisées en calcul scientifique (PETSc, Matlab via des scripts).

L'utilisation de SCIRun est basée sur une interface graphique qui permet de construire l'application en créant le canevas de l'application graphiquement module par module et en inter-connectant leurs ports. De plus, un module particulier peut être utilisé afin d'intégrer de la visualisation dans une application. Ce module est en fait un viewer avec seulement des ports d'entrée connectés à des modules de pré rendu et de rendu appliqués aux résultats d'une simulation comme l'illustre la figure 25 issue d'un tutoriel SCIRun[10].

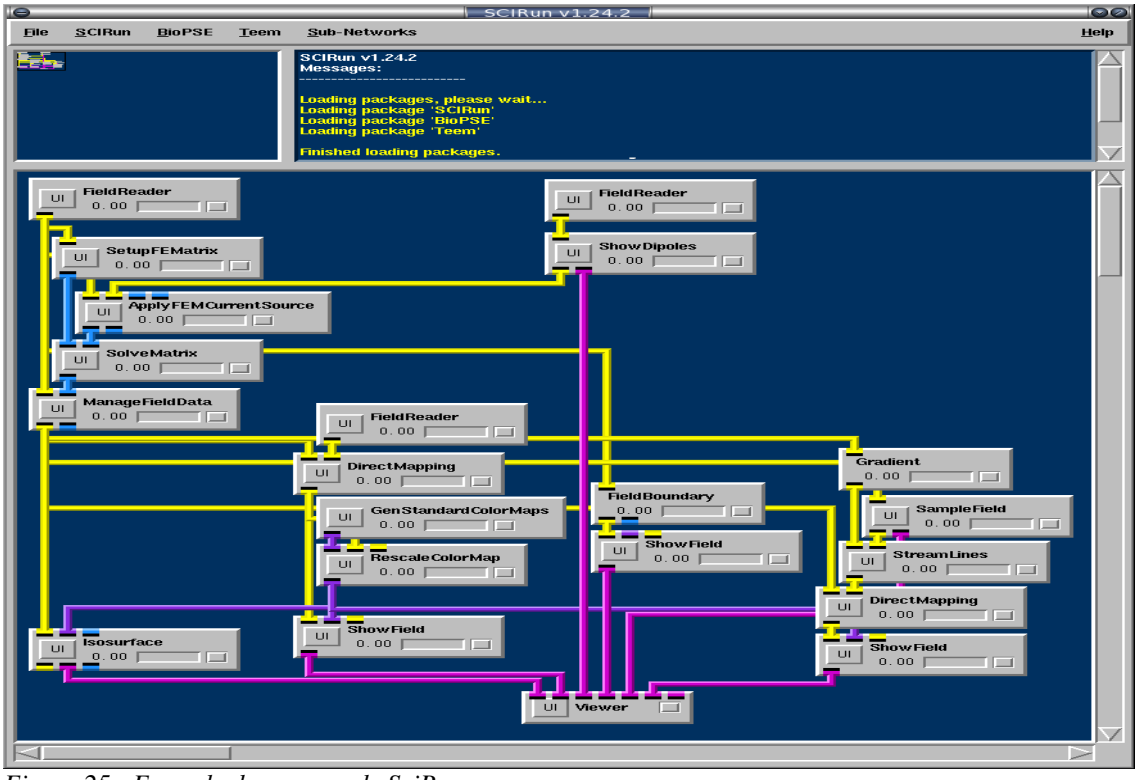

Figure 25 : Exemple de canevas de SciRun

SCIRun est destiné aux architectures à mémoire partagée et utilise donc un parallélisme par threads. En intégrant à la fois des bibliothèques de calculs et graphiques il offre un environnement de travail convivial et qui semble facile à utiliser. Il reste une question sur la facilité d'intégration d'un

10 http://software.sci.utah.edu/scirun.html

nouveau module de calcul ou graphique lorsque l'algorithme voulu n'est pas implémenté dans les bibliothèques disponibles.

8.2 -SciRun2

Dans [45], la visualisation distante est pointée comme une perspective d'amélioration de SCIRun. Cependant, au vu de la complexité des différents scénarios possibles pour la répartition du pipeline graphique (similaires à nos trois scénarios), la direction choisie est de faire évoluer SCIRun en une approche par composants conforme au modèle CCA.

Ainsi l'ambition de SCIRun2 comme l'illustre la figure 26[11] est de fournir un méta modèle par composants permettant d'intégrer un ensemble d'environnements au sein d'une même application.

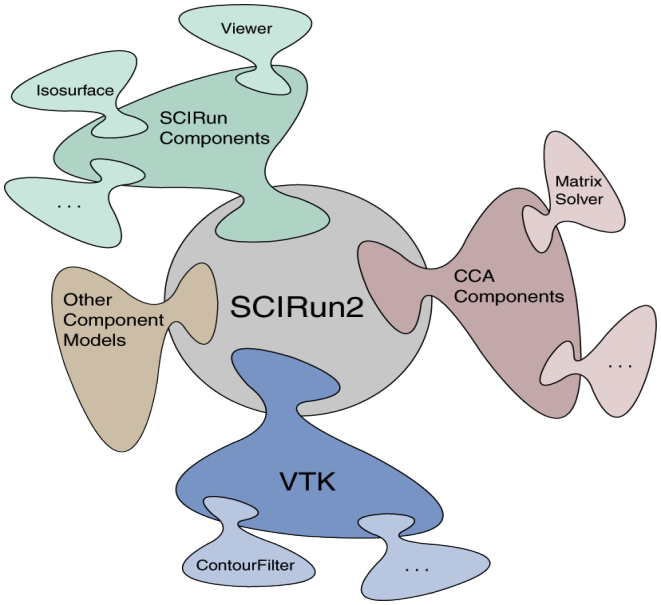

Figure 26 : SCIRun2 comme méta modéle par composants

SCIRun2 est dans un premier temps une implémentation d'une approche par composants de type CCA intégrant aussi les ports directs ou collectifs à partir d'une bibliothèque appelée MxN [36] qui formalise les méthodes collectives d'invocation et de distribution de données sur plusieurs composants. De cette manière il définit des composants parallèles constitués d'un ensemble de composants dont tous les ports participent à une même opération collective « uses » ou « provides ». Ensuite, SCIRun2 permet ou permettra d'intégrer directement des composants CORBA, Microsoft COM ou des modules VTK dans une application SCIRun2. Enfin, SCIRun2 en tant que méta modèle se base également sur des composants supplémentaires appelés « bridges » pour créer des ponts entre différents composants n'appartenant pas à la même infrastructure CCA.

Un prototype SCIRun2 semble être en cours de développement mais non disponible. Cette approche, encore une fois plus générale qu'une solution de visualisation distante est à suivre car ses perspectives pourraient en faire un environnement de déploiement d'applications très intéressant.

11 http://www.cca-forum.org

9. Mbender

Le projet Mbender[12], du Visualization Group du Lawrence Berkeley National Laboratory, s'est intéressé aux solutions de rechange aux décompositions traditionnelles sous forme de pipeline pour améliorer la visualisation 3D interactive à distance. Le but affiché est de proposer une visualisation entièrement interactive des données 3D variables dans le temps à l'aide "des outils de bureau standard". Le projet part du principe que dans beaucoup de cas, les utilisateurs exigent seulement un sous-ensemble de toutes les fonctionnalités possibles de visualisation. Ils souhaitent simplement comprendre des rapports entre la forme 3D et la profondeur, et passer en revue des données variables dans le temps présentées dans un format 3D.
L'approche adoptée est d'explorer des manières de fournir la visualisation et les résultats de rendu d'une façon qui simule l'expérience par l'exploration interactive en 3D. Le mécanisme de fourniture à distance du rendu doit trouver un équilibre entre le coût et les fonctionnalités offertes. L'envoi des images plutôt que des données est une meilleure approche en terme d'extensibilité vers des données toujours plus grandes. Cependant, l'envoi des images est intrinsèquement coûteux et l'interactivité souffre dès qu'il y a une latence même mineure due au réseau.
Le choix de Mbender est de créer un film interactif 3D. Il produit un certain nombre d'images d'une scène 4D (3 + temps) en changeant le point de vue, puis fournit à l'utilisateur la capacité de choisir facilement un point de vue ou un pas de temps. De cette façon, l'utilisateur a l'expérience de l'interaction 3D avec des données variables en temps. Cette technique, comme l'IBR, est particulièrement utile dans le cadre de la visualisation scientifique à distance puisqu'elle est basée sur le transfert des images dont la taille peut être bien inférieure à celle des données. Par contre, contrairement aux techniques traditionnelles d'IBR qui tiennent compte de la cohérence inter frame pour reconstruire des images, cette approche se contente de fournir les moyens de présenter des images préexistantes adaptées au point de vue de l'utilisateur. Plusieurs approches alternatives répondent à ces objectifs : des primitives java-script, QuickTime VR et Mbrowser, un navigateur sur des images multi-résolution.
Mbender sacrifie la généralité des autres solutions de visualisation distante mais offre de nombreux avantages :

- ✔ Le coût de transfert est borné par le nombre de vues et la résolution des images rendues qui composent le film interactif, pas par la quantité des données.
- ✔ La capacité d'exécuter de l'interaction/exploration des données 3D variables en temps en utilisant "le logiciel de bureau standard", comme un navigateur web standard, ou le module QuickTime des ordinateurs Apple.
- ✔ La capacité de fournir une expérience 3D interactive aux utilisateurs à distance indépendamment de l'algorithme, de la technique ou de l'application de visualisation produisant de l'imagerie.
- ✔ La capacité de servir de la visualisation à distance avec un simple serveur web, simplifiant par là même les problèmes de configuration des serveurs, des pare-feux.
- ✔ Une vision similaire au service web, la visualisation peut être accessible à un grand nombre d'utilisateurs potentiels, ou seulement à quelques-uns, en fonction des choix de contrôles d'accès.

12 - http://www-vis.lbl.gov/Research/MBender/

9.1 -Encodeur Javascript

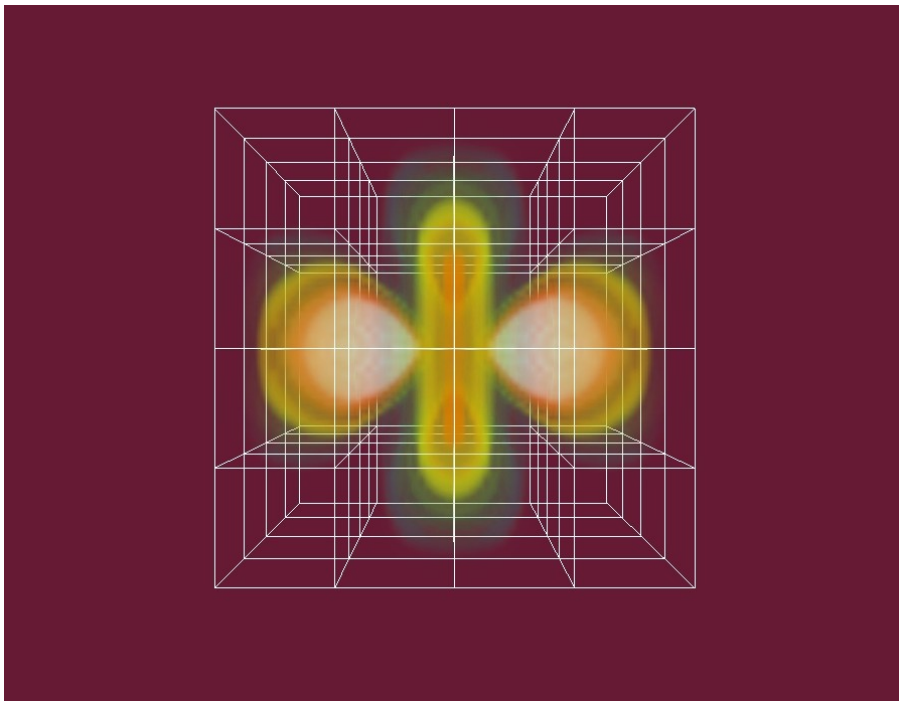

Figure 27 : Visualisation via l'encodeur Javascript

Une des premières expériences citée du projet Mbender utilise JavaScript pour exécuter du côté du client la sélection d'images dynamiques basée sur la position de la souris (Figure 27). C'est présenté en tant que solution "à faible composante technologique" utilisant n'importe laquelle parmi un certain nombre d'applications de visualisation pour produire des images selon un jeu de points de vue et stocker les images dans des fichiers de rastérisation. Ensuite, "un encodeur" crée une page web contenant les images en lien avec un client Javascript qui sélectionne les images en fonction de la position de la souris. Si les points de vue sont choisis de la bonne manière, alors l'interface de Javascript « se comporte comme » une navigation 3D interactive classique.
L'avantage principal de cette approche (via un encodeur en Javascript) est qu'un web browser standard peut être employé pour faire l'interaction 3D. Il demeure l'inconvénient que son utilisation n'est pas raisonnable pour des « grandes et complexes » interactions 3D. Le Javascript pré-charge d'abord toutes les images pour la scène dans le cache du navigateur avant qu'il soit possible d'agir sur la scène. Quand la scène contient un nombre restreint d'images, cette limitation est rarement un problème. Quand « le film interactif » contient un grand nombre d'images - peut-être représentant beaucoup de points de vue finement espacés pour fournir une bonne fidélité visuelle - alors il est possible que la saturation du cache provoque une erreur du navigateur. L' approche Javascript est intéressante pour la diffusion des résultats (peut-être pour naviguer dans les bases de résultats), mais son utilisation est relativement limitée pour la visualisation 3D interactive des données importantes en taille.

9.2 -QuickTime VR object Movies

QuickTime VR est un lecteur de média créé et maintenu par Apple. C'est une technologie inter plate-forme, pour manoeuvrer, augmenter et stocker la vidéo, le son, l'animation, les graphiques, le texte, la musique, ainsi que la visualisation à 360 degrés propre à la « réalité virtuelle». Il permet

également de lire en continu un flux vidéo numérique où le flux de données peut être stocké ou fourni en continu. Apple sur son site web précise qu'il existe deux types principaux de film QuickTime VR : les «films de panorama» et les « films d'objet». Les deux formes sont des films où l'image est dirigée par l'utilisateur. Les films peuvent être créés photographiquement ou avec un logiciel 3D. Classiquement les environnements rendus sont des interprétations photo-réalistes de la réalité. Avec un film de panorama, il est possible de tourner à 360 degrés à partir d'un point fixe afin de regarder autour de soi. Inversement, les films d'objet (voir Figure 28) sont plus apparentés au regard à l'intérieur d'un bocal depuis n'importe quelle direction. En d'autres termes, on peut placer le point de vue à n'importe quel endroit sur la surface d'une sphère tout en observant toujours le centre de la sphère. Les films d'objets sont composés d'un choix d'images discrètes : QTVR ne fait aucune interpolation. Si l'on dispose d'un grand nombre d'images prises à des angles proche, le mouvement semble lisse. Le film d'objets de QTVR est une excellente manière de présenter en 3D les résultats de visualisation et de permettre une manipulation interactive des vues. Le player QTVR autorise également les «zoom-in, zoom-out », ainsi que la capacité de rembobiner des images variables dans le temps.

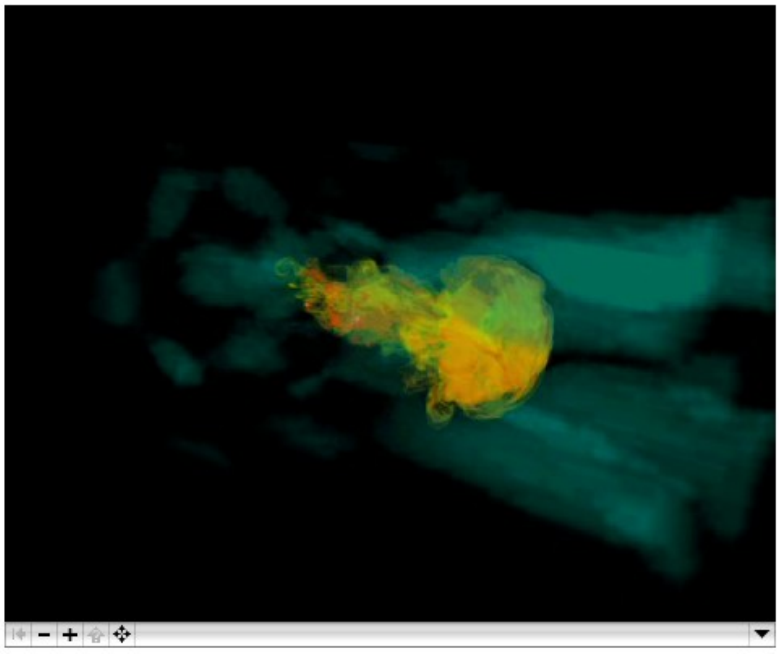

Figure 28 : Visualisation via QTVR

L'avantage important par rapport à l'encodeur javascript est que les images n'ont pas besoin d'être entièrement stockées en mémoire pour commencer le rendu. Elles peuvent apparaître au fur et à mesure en fonction de la navigation tout en amortissant la latence du réseau grâce à la capacité de QTVR d'offrir une navigation 3D à partir d'images en mémoire. Ce système a ainsi la capacité de « cacher » les résultats d'un rendu distant et de les visualiser sur n'importe quelle machine de bureau.

La contribution de LBNL sur ce sujet est la création d'un encodeur qui produit des films d'objet QuickTime VR. Le projet semble avoir rencontré des difficultés à trouver un encodeur QTVR open-source.

9.3 -Navigation interactive dans des images pré-rendues en multi-résolution.

Si l'on utilise le zoom de QTVR, on voit rapidement qu'il y a un problème avec la résolution d'image. Pendant le zoom, la résolution ne s'améliore pas. Il serait préférable d'offrir la capacité de zoomer à des résolutions aussi élevées que nécessaire. Ainsi, des images de résolution plus élevées devraient pouvoir être chargées dans un espace de cache afin d'être visualisées en cas de zoom.

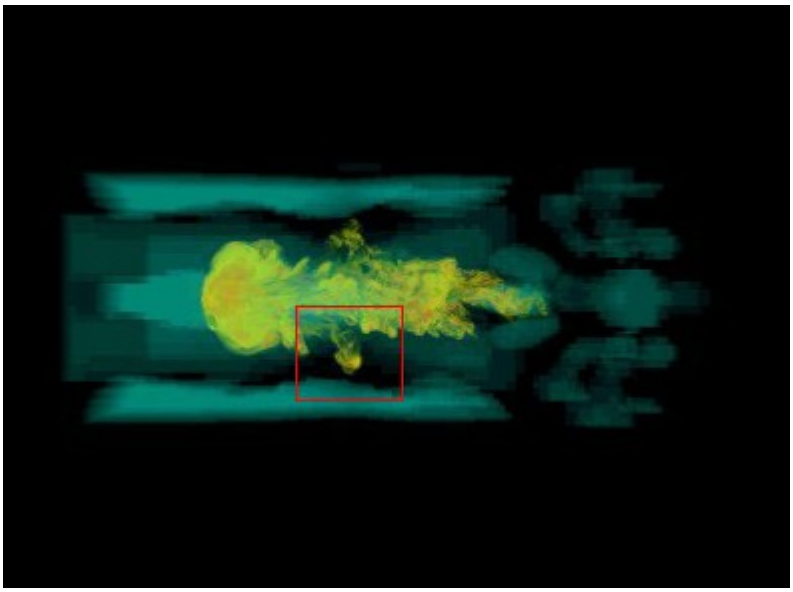

Figure 29 : Image QTVR large

Les figures 29, 30 et 31 illustrent exactement ce point. La figure 29 est la représentation la plus large de l'objet. Si l'on zoome à l'intérieur avec le player normal de QTVR, on obtient une image brouillée donnée figure 30. Avec la véritable représentation de multi-résolution, on obtient une image fortement détaillée figure 31 Le travail de multi-résolution part du besoin de maintenir la fidélité visuelle élevée pendant la navigation interactive 3D. Des technologies semblables sont naissantes sur le web, par exemple pour l'achat en ligne afin de permettre le zoom sur des détails de produit[13].

13 http://www.viewpoint.com/

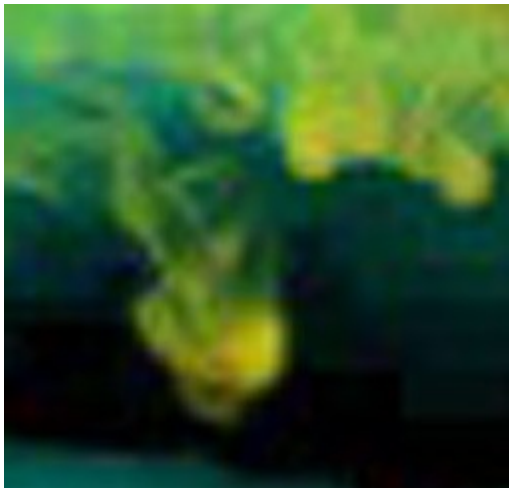

Figure 30 : Détail sans multi-résolution

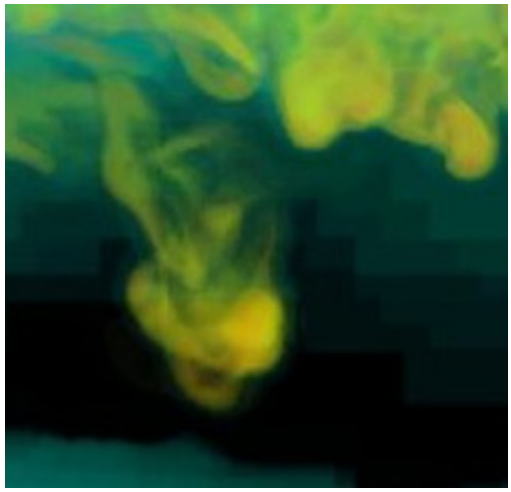

Figure 31 : Détail avec multi-résolution

L'architecture (figure 32) du système multi-résolution est basée sur des images pré-rendues, par exemple via OpenRM Scene Graph [35]. Les images, par pas de temps, sont prises de plusieurs points de vues et à différentes résolutions; par exemple : 400x300, 1600x1200 et 4000x3000. Ensuite les images sont pré-traitées pour obtenir un ensemble de fichiers avec une convention de noms en lien avec deux fichiers de méta-données : le fichier catalogue et le fichier carte. Ces fichiers sont placés sur un serveur web. Le client charge ces fichiers de méta-données puis les images en fonction de la navigation. Le client est ici un code classique écrit en C++ qui fait appel à la technologie JNI (Java Native Interface) et JNLP (Java Network Launching Protocol).

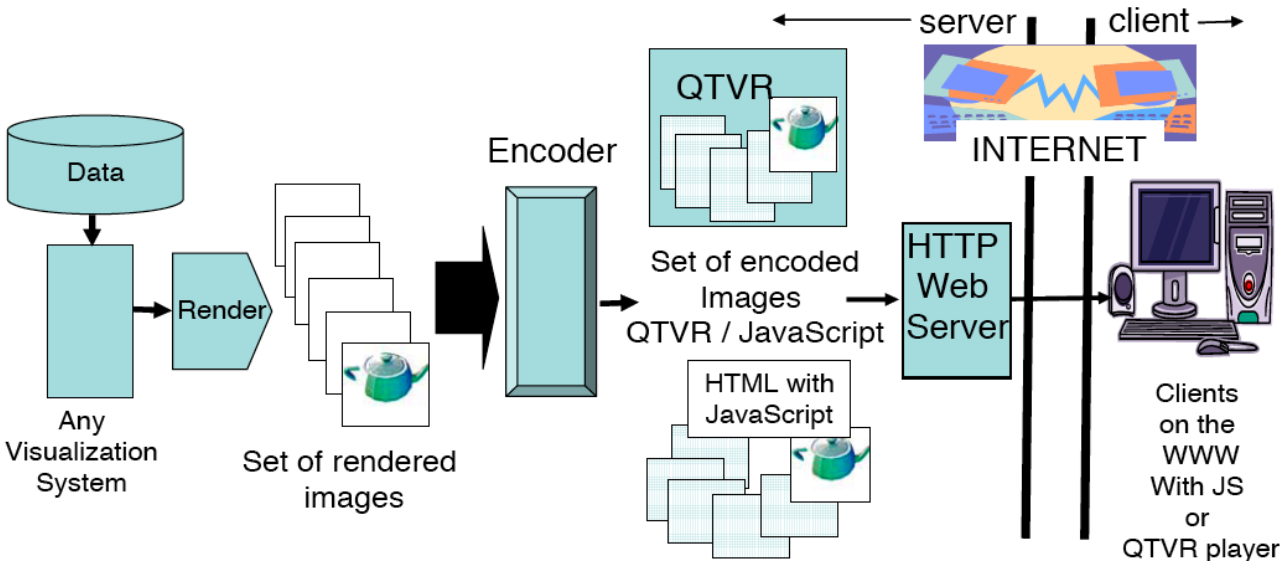

Figure 32: Chaîne de traitement Multi-résolution

Il serait intéressant d'étendre ce système à des cartes dynamiques ce qui pourrait élargir le nombre des images accessibles. De même on pourrait imaginer de générer les images à la volée en sortie de simulation.

10.CumulVS

CumulVS [13] est un système de pilotage et de visualisation de simulations lourdes, de collaboration et de tolérance aux fautes et pannes. Il est développé par Oak Ridge National Laboratory, et supporté par le U.S. Department of Energy (DOE). Il n'est pas destiné à la visualisation distante en tant que tel et peut donc être classifié comme un système tangent à notre étude. Cependant, dans le cadre du pilotage, il intègre la visualisation comme un outil pour contrôler l'évolution des données en cours de simulation. Dans ce sens, il propose aussi de la visualisation distante.

10.1 -Approche

Il s'agit de permettre à plusieurs collaborateurs de connecter/déconnecter dynamiquement un visualiseur (viewer) à une simulation parallèle en action.

10.2 -Architecture

L'architecture repose sur un viewer connecté à l'application. Ce dernier demande à visualiser une portion (ou région) du domaine de calcul à une fréquence donnée. La spécification des données concerne aussi les frontières et la taille des cellules. A partir de cette demande les données sont collectées et envoyées au viewer. Pour s'assurer que celui-ci a une vue cohérente (itération) des données, un mécanisme de synchronisation lâche (sans barrière explicite) est réalisé après chaque envoi de données.
Plusieurs collaborateurs peuvent modifier l'application parallèle. Un mécanisme à jeton permet de maintenir la cohérence des paramètres du pilotage. Des protocoles de consistance permettent de garantir que toutes les tâches modifient les paramètres à l'unisson.

CUMULVS comporte deux parties : une partie applicative et un front-end de visualisation. L'application peut être constituée de tout programme à passage de messages et le front-end de tout système de visualisation (implémentation actuelle PVM/AVS).

L'interface utilisateur est assez simple. On doit spécifier deux informations importantes soit décrire la manière dont les données sont décomposées et définir les paramètres à piloter.

Pour la collecte de champs de données, CUMULVS réalise les opérations suivantes :

- ✔ le viewer se connecte initialement à l'application pour rassembler des informations sur les différents champs de données et les paramètres,
- ✔ le viewer demande la collecte des données d'un certain nombre de champs dits VFG (View field Group),
- ✔ pour chaque VFG, le viewer spécifie la portion de données à collecter (visualization region); cette zone est définie en termes d'intervalles suivant les axes du volume et une taille de cellule.

La deuxième fonction essentielle de CUMULVS est le pilotage de simulation qui consiste à pouvoir modifier en ligne certains paramètres d'une application par des processus distants (ou externes). Le

pilotage est initialisé par un appel de fonction qui réalise l'équivalent d'une requête de champs de données en établissant une connexion synchrone lâche avec l'application. Cette synchronisation garantit que toutes les tâches modifient les paramètres au même moment.

Une fois le pilotage initialisé, pour modifier un paramètre particulier, le viewer doit acquérir un jeton spécifique. Lorsqu'il a le jeton, un viewer modifie les paramètres qu'il a sélectionnés avant de les renvoyer à l'application et relâcher le jeton.

Pour certains types de simulations comme les systèmes à particules, plusieurs instances d'un même objet peuvent être contrôlées différemment. Pour ce type d'utilisation, CUMULVS propose les paramètres indexés. Avec ces derniers, un paramètre supplémentaire « index » est ajouté à ceux de l'application. Lors de la récupération des paramètres par l'application, c'est ce paramètre qui est extrait en premier pour connaître l'instance particulière d'un objet à laquelle s'appliquent les autres paramètres.

Les programmes parallèles s'exécutent de plus en plus sur des architectures éclatées et hétérogènes. La tolérance aux fautes et pannes devient nécessaire. CUMULVS fournit des outils pour un tel mécanisme. Ainsi tous les moyens nécessaires pour redémarrer une application parallèle échouée sont encapsulés dans un processus séparé, le CPD (checkpointing daemon). C'est un checkpointing dirigé par l'utilisateur dans le sens où c'est le programmeur qui spécifie les variables à sauvegarder et fournit les moyens d'indiquer si l'application démarre normalement ou à partir d'un checkpoint. CUMULVS, quant à lui, gère l'extraction du dernier checkpoint cohérent ainsi que son chargement dans les variables utilisateur. Cette approche permet la migration de processus, les checkpoints stockant suffisamment d'information.

Le CPD s'exécute sur chaque noeud de la machine virtuelle. Il fournit deux fonctions élémentaires :

- ✔ sauvegarder un checkpoint à partir d'une application
- ✔ charger un checkpoint dans une application

Deux réponses sont possibles en cas d'échec : soit on arrête tous les noeuds et on opère un redémarrage complet après chaque échec, soit on notifie aux noeuds encore actifs qu'ils doivent repartir d'un checkpoint. Dans tous les cas, c'est le CPD qui contrôle le signalement et la gestion des tâches pour redémarrer proprement une application parallèle qui a partiellement ou complètement échoué.

Le commit d'un checkpoint (écriture en mémoire non-volatile) a un surcoût important. A cause du schéma asynchrone entre les tâches, lors d'un échec, si un checkpoint est incomplet alors toutes les tâches doivent remonter collectivement au dernier checkpoint valide. Suivant la taille des données dans les checkpoints, celles-ci sont répliquées sur chaque noeud ou non.

11.Environnement pour le Pilotage de Simulations Numériques

L'environnement EPSN est également dédié au pilotage de simulations numériques parallèles et distribués. Il est développé au sein d'une ACI-GRID par le projet ScAlApplix de l'INRIA Futurs[14]. Cet environnement a en particulier fait l'objet d'une thèse [16].

14 http://www.labri.fr/projet/scalapplix/

11.1 -Approche

Dans le cadre des simulations numériques reposant sur des processus de calcul itératif selon une architecture de type SPMD, l'objectif de cet environnement est de permettre aux utilisateurs de piloter une simulation par la visualisation des résultats intermédiaires. Le principe de ce pilotage est d'insérer dans la simulation même des points d'arrêt gérés par une API qui permet l'extraction des données et leur visualisation sur le client local selon le principe illustré figure 33.

11.2 -Architecture

L'architecture d'EPSN est basée sur des modélisations de la simulation et du pilotage pour générer un modèle hiérarchique en tâches qui permettra de construire les interactions avec la simulation. Comme le montre la figure 34, l'utilisateur fournit ces modélisations sous forme d'instrumentation du code source associée à une description complémentaire XML précisant le pilotage c'est-à-dire les modes d'interaction avec la simulation. Une re-compilation de l'exécutable de la simulation, utilisant les bibliothèques EPSN, permet de fournir une nouvelle version dite « pilotable » qui se déploie sur le serveur de calculs de manière ordinaire pour l'utilisateur. Le client pour réaliser son pilotage se connecte alors dynamiquement à la simulation et peut lui soumettre différentes requêtes comme l'accès à la volée des résultats intermédiaires, la modification à distance de paramètres de la simulation. Dans ce sens, l'environnement EPSN fournit un client générique appelé *Simone* qui permet la connexion et l'interaction avec toutes simulations instrumentées par EPSN.

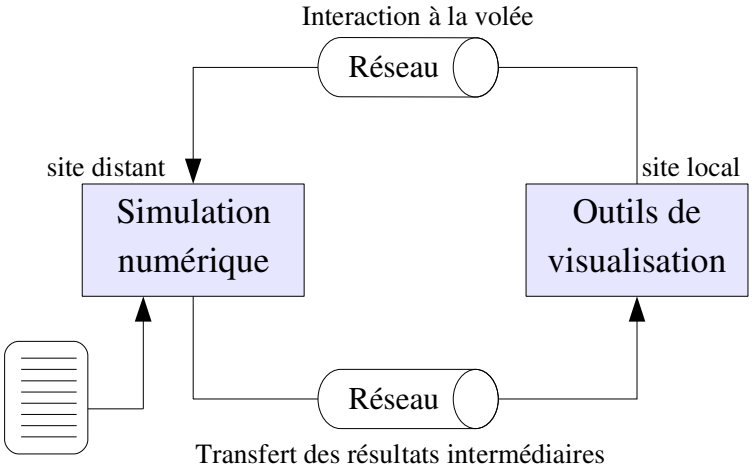

Figure 33 : Simulation interactive (schéma repris de [16])

En ce qui concerne le couplage entre la simulation et le client réalisant le pilotage, il suit un modèle par composants parallèles. Ainsi la simulation est le premier composant constitué d'un proxy EPSN et d'un ensemble de ports pour tous les processus. De même la visualisation qui peut être parallèle constitue le deuxième composant sur le même principe. EPSN fournit alors une couche communication basée sur CORBA pour assurer les transferts de données entre les deux composants. Les requêtes via le proxy du client sont envoyées au proxy du serveur de simulations qui les transmet aux ports des processus de la simulation qui enfin réalisent le transfert parallèle des

données demandées.

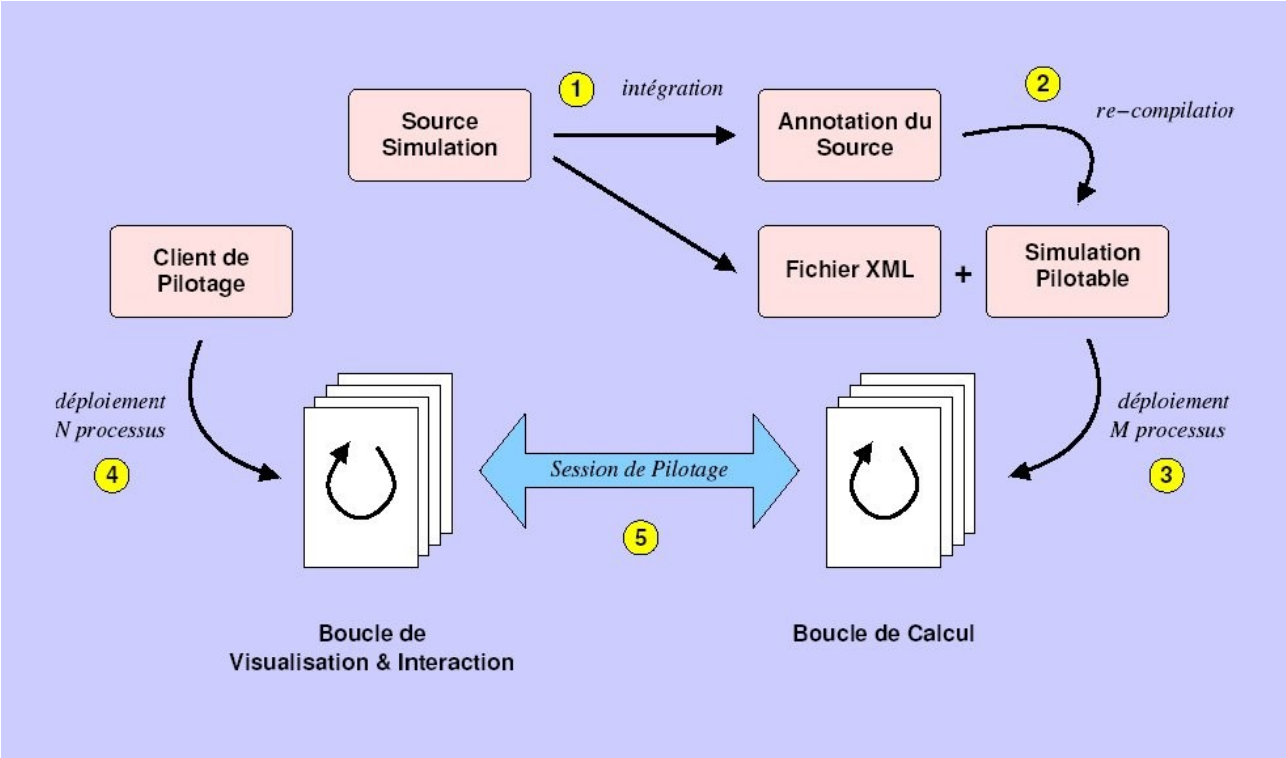

Figure 34 : Le cycle de vie d'EPSN (schéma repris de [16])

EPSN est donc plus qu'un outil de visualisation distante puisque son premier objectif est d'offrir les moyens d'interagir avec une simulation. Cependant, le pilotage étant fortement lié à de la visualisation il reste un exemple significatif dans le cadre du scénario 2 (rendu local) pour le choix effectué sur le couplage de type CORBA entre le serveur de simulation et le client qui réalise le pilotage.

12.Cactus

Cactus est un environnement libre, modulaire et portable pour le développement collaboratif de code parallèle haute performance de simulations générant des données multidimensionnelles [5]. Il est portable au sens où l'on doit pouvoir compiler et exécuter un code Cactus sur n'importe quelle plate-forme. Il est modulaire au sens où il est possible d'écrire des modules qui interagissent via des interfaces standards sans pour autant connaître le fonctionnement interne des autres modules et évidemment dans ce cas tout module peut être remplacé de façon transparente. Compte tenu du public visé (physiciens) Cactus possède des structures permettant un portage aisé de codes anciens (héritage de code Fortran 77). Cactus ne réinvente pas la roue, mais fait un large usage de technologies existantes. Il n'impose pas un modèle de programmation afin de ne pas hypothéquer l'avenir. Il propose une programmation parallèle en mode propre mais reste compatible avec les bibliothèques standards de passage de messages. Son mode d'entrée-sortie est propre mais compatible avec des systèmes standards tel que HDF. Cet environnement se veut simple à utiliser, bien documenté et maintenable.

12.1 -Modèle

Le modèle est basé sur un corps central (« flesh » littéralement « la chair » (du cactus)) auquel les modules (« thorns » littéralement les épines (du cactus)) font référence. Le corps est écrit en C ANSI à des fins de portabilité. Outre le corps, l'environnement est principalement constitué de modules pour faire du parallélisme (classiquement MPI ou sur la gille : Globus via MPICH-G), des I/O (Flex IO, HDF5, Panda), de la physique (drivers pour unigrid/ARM (GrACE), solveurs PETc) , du pilotage à distance et de la visualisation sans particulièrement privilégier cette dernière (1d, 2d, 3d Amira, AVS, Open DX). Les langages utilisés sont laissés au choix des auteurs des modules.

Le corps Cactus est indépendant des modules. Il sert de bibliothèque de service que les modules appellent pour avoir des informations ou demander des actions. Chaque module déclare l'implémentation qu'il assume et les dépendances entre modules sont exprimées non pas via des références explicites à d'autres modules (ce qui briserait la modularité du modèle) mais sous la forme de dépendances d'implémentation. Ce choix permet de rendre les modules interchangeables. C'est finalement de la bonne politique de développement assez classique pour les informaticiens et re-découverte par les physiciens avec toutefois la possibilité d'avoir un support d'abstraction de la notion d'implémentation.
Le corps Cactus est constitué ainsi de plusieurs éléments :

- un system Make : avec usage de configurations pour adapter les compilations aux architectures (T3E, IRIX, OSF, Linux, NT, etc)
- une API pour les modules afin qu'ils puissent inter-opérer avec le corps
- un scheduler qui opère sur les fonctions offertes par les modules (le scheduling des routines ou groupe de routine est de type : **avant/apres/tant que** (très utile pour les simulation avec critère de convergence) avec un système de détection d'inconsistance (figure 35).
- un langage de configuration, CCL, qui exprime ce que le corps a besoin de connaître des modules (exemple : L'implémentation fournie, l'ordre des routines que le scheduler doit respecter, variables à passer entre les routines, paramètres ...).

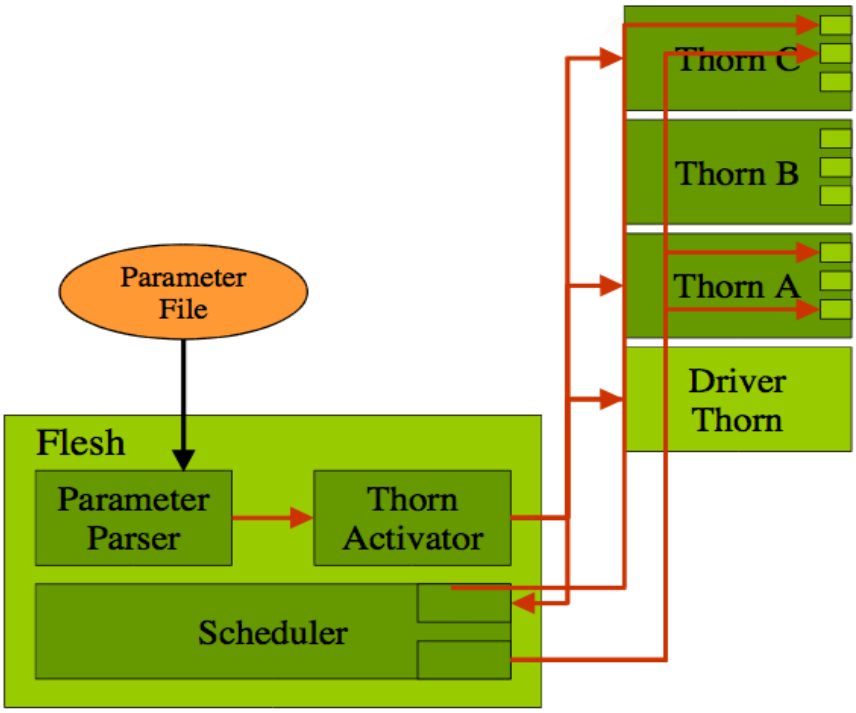

Figure 35 : flot du programme Cactus

12.2 -Parallélisme

Les « drivers » sont les seuls modules qui accèdent en propre à la notion de parallélisme. Les autres y accèdent via le corps Cactus (Flesh). Si ces modules le souhaitent, ils peuvent faire appel, via le corps, à des fonctionnalités de synchronisation, de réduction ou d'interpolation fournies, en réalité, pas le « driver ». La couche parallèle sous-jacente est transparente aux modules, elle peut être constituée de sockets ou de systèmes plus évolués. Il est possible de choisir une implémentation du parallélisme à la volée sans recompiler, juste en changeant de « driver », afin de tester la couche adéquate à l'environnement courant. Les « drivers » gèrent les problèmes de découpage des données et de répartitions de celles-ci.

12.3 -Visualisation distante et analyse de données

Celle-ci doit permettre, du point de vue Cactus, de se connecter à une simulation en cours d'exécution pour réduire les temps d'analyse des sorties de simulation. Les expérimentations citées pointent qu'il ne sert à rien d'avoir une visualisation trop performante si l'on est bridé par la sortie de simulation. Aussi les concepteurs ont-ils décidé d'implanter des outils directement sur les machines de simulation offrant ainsi un traitement de la visualisation dont la capacité est du même ordre de grandeur que celle de la simulation. L'autre voie explorée via Cactus est la réduction du volume des données à visualiser en n'exportant qu'un sous-domaine. Les données sont sous échantillonnées avec l'aide des modules d'I/O en particulier avec la méthode HDF5. Un dernière voie est la représentation dans des sous -dimensions.
Dans tous les cas de figure choisis pour la visualisation, Cactus n'apporte que son modèle modulaire et des modules pré-réalisés. Il ne constitue pas intrinsèquement un outil spécifique pour la visualisation distante. Par contre, de par le type de problème qu'il vise à résoudre, les concepteurs des applications Cactus ont dû proposer des solutions de visualisation distante.

V. Conclusion

Dans cet état de l'art, nous avons recensé différentes techniques permettant la visualisation distante. Ces techniques, comme pour VizServer par exemple, se rajoutent simplement à une application générale pour l'enrichir avec un déport de l'affichage ou sont intégrées pleinement à des systèmes généraux tel que ParaView. Les solutions de visualisation distante décrites dans ce rapport sont donc diverses et de maturités différentes par leurs ambitions.

Lorsqu'elles sont conçues comme des améliorations d'un système déjà développé elles se basent soit sur une approche service web développée de manière ad hoc soit sur de l'image streaming de type Vizserver. L'objectif est alors d'améliorer une visualisation existante pour permettre à l'utilisateur d'en avoir le résultat sur son propre poste de travail. Et dans ce cadre, même si la latence du réseau peut être un frein à la fluidité de l'affichage, ces solutions sont justifiées et suffisantes surtout s'il s'agit d'un affichage en résolution fixe de données ne variant pas dans le temps.

Lorsqu'elles sont conçues pour faciliter la visualisation de gros volumes de données sur un poste de travail simple, elles empruntent aux techniques de type out-of-core et sont basées sur du data streaming avec multi-résolution. Dans un contexte de rendu volumique indirect de type iso-surfaces, cette famille de solutions semble essentielle à la visualisation distante. Même si pour le moment elles sont largement dépendantes des formats de données, elles décrivent des méthodes de représentation et d'exploration qui nous paraissent pertinentes et qui de plus pourraient être enrichies par des techniques de niveaux de détail dynamiques. Cependant, de par la diversité des données manipulées dans les domaines applicatifs, il s'agit pour le moment d'un ensemble d'expériences qui répondent à des besoins scientifiques ponctuels et il est difficile de faire émerger une solution qui uniformiserait cette approche.

Enfin, lorsque l'ambition est de se donner les moyens d'analyser et de visualiser de grands volumes de données, les solutions sont plus complexes et s'intègrent dans un environnement plus général (Covise, EnSight, Visit, gViz, EPSN, VTK, ParaView ...) qui peut prendre en compte des aspects supplémentaires comme le travail collaboratif ou le pilotage de simulations.

Une exception parmi ces systèmes concerne VisaPult qui a l'avantage de proposer une architecture relativement simple basée sur un algorithme efficace qui assure un bon compromis entre répartition de la charge client/serveur et qualité d'affichage. Cependant, ce projet semble avoir deux perspectives possibles pour évoluer qui représentent bien l'ambivalence des besoins dans ce domaine. La première vise également la conception d'un environnement général avec DIVA dont l'objectif est d'éviter le recours à l'empilement plus ou moins efficace d'outils traitant un par un les problèmes de la visualisation scientifique. La deuxième, avec Mbender, est au contraire une réduction de la visualisation scientifique mais avec une réelle recherche de performance brute. En effet, Mbender a pour objectif de démocratiser la visualisation scientifique en permettant à tout scientifique de manipuler des données sur son poste de travail.

Plus le système vise la généralité, c'est-à-dire le couplage multi simulations enrichi de visualisation (collaborative, avec pilotage) plus le niveau d'abstraction des solutions proposées est élevé. Il s'agit alors de répondre à des souhaits d'exhaustivité pour définir une plate-forme intégrant de la

visualisation mais très ouverte à l'usage d'outils spécifiques à chaque domaine applicatif (VMD, NAMD, ...). Dans ce sens, l'approche par composants semble prometteuse. Si elle ne dispose pas encore de la maturité nécessaire face à la complexité des applications scientifiques, elle devrait permettre à terme de faciliter les mises en oeuvre de couplage complexes adaptées aux différentes architectures de déploiement.

Cependant si l'approche par composants paraît incontournable, elle ne devra pas gagner en terme de génie logiciel pour perdre en expressivité et en performance par rapport aux autres approches. Si les ambitions de ces dernières sont certes plus focalisées, elles ont l'avantage d'exploiter cette simplification afin d'être efficaces pour l'utilisateur final. Il est, à notre sens, tout à fait possible, qu'à l'instar de l'évolution des outils de formalisation d'applications complexes en Réalité Virtuelle, ce ne soit pas les environnements lourds qui parviennent à concevoir et à exécuter efficacement les codes ambitieux de visualisation scientifique, mais plutôt des environnements simplifiés, tels que FlowVR pour le cadre de la Réalité Virtuelle. Ces environnements simplifiés n'ont pas à conserver des compatibilités en dehors du cadre de leur champs d'exploitation, et peuvent ainsi offrir de grandes facilités en terme de conception et d'expressivité avec une implémentation recherchant la performance.

Bibliographie

[1]Site Internet, "*Amira*". http://www.amiravis.com, .

[2] C Upson, T Faulhaber, D Kamins, D Schlegel, D Laidlaw, J Vroom, R Gurwitz, A Vandam, "*The Application Visualization System: a Computational Environment for Scientific Visualization*". *IEEE Computer Graphics and Applications*, 1989.

[3] HD Lord, "*Improving the Application Development Process with Modular Visualization Environments*". *Computer Graphics*, 1995.

[4] W Bethel, "*Visualization Dot Com*". *Computer Graphics and Applications, IEEE*, 2000.

[5] T Goodale, G Allen, G Lanfermann, J Masso, T Radke, E Seidel, J Shalf, "*The Cactus Framework and Toolkit: Design and Applications*". *VECPAR 2002, 5th International Conference*, 2003.

[6]R Armstrong, D Gannon, A Geist, K Keahey, S Kohn, L McInnes, S Parker, B Smolinski, "*Toward a Common Component Architecture for High-Performance Scientific Computing*". HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, 1999.

[7]A Neeman, P Sulatycke, K Ghose, "*Fast Remote Isosurface Visualization with chessboarding*". in Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV04), 75-82 2004.

[8]G Humphreys, M Houston, R Ng, S Ahern, R Frank, P Kirchner, JT Klosowski, "*Chromium: A Stream Processing Framework for Interactive Graphics on Clusters of Workstations*". Proceedings of ACM SIGGRAPH 02, 693-702 2002.

[9]JM Geib, C Gransart, P Merle, "*Corba des concepts à la pratique*". DUNOD, LIFL, Université des sciences et technologies de Lille, 1999.

[10]A Denis, C Pérez, T Priol, "*Towards High Performance CORBA and MPI Middlewares for Grid Computing*". Proc of the 2nd International Workshop on Grid Computing, 14-25 2001.

[11]J Wood, H Wright, K Brodlie, "*Collaborative visualization*". VIS '97: Proceedings of the 8th conference on Visualization '97, 1997.

[12] D Rantzau, U Lang, "*A scalable virtual environment for large scale scientific data analysis*". *Future Generation Computer Systems*, 1998.

[13] J Kohl, P Papadopoulos, "*CUMULVS: Providing fault-tolerance, Visualization, and Steering of Parallel Applications*". *Int. J. of Supercomputer Applications and High Performance Computing*, 1997.

[14]D Schikore, R Frank, "*Parallel rendering*". VIZ'06, 2006.

[15]R Kares, "*EnSight Gold and the Visualization of TeraScale Simulations at the Los Alamos National Laboratory*". CEI VIZ'06 Conference, 2006.

[16]A Esnard, "*Analyse, conception et réalisation d'un environnement pour le pilotage et la visualisation en ligne de simulations numériques parallèles*". Ph.D. thesis,2005.

[17]J Allard, V Gouranton, L Lecointre, S Limet, E Melin, B Raffin, S Robert, "*FlowVR: a Middleware for Large Scale Virtual Reality Applications*". Proceedings of Euro-par 2004, 2004.

[18]H Hagen, A Ebert, R H van Lengen and G Scheuermann, "*Scientific*

Visualization - Methods and Applications -". Proceedings of the 19th spring conference on Computer graphics, 23-33 2003.
[19]K W Brodlie, D A Duce, J R Gallop, J Walton and J Wood, "*Distributed and Collaborative Visualization*". Computer Graphics Forum, 223-251 2004.
[20]S Shumilov, A Thomsen, A Cremers, B Koos, "*Management and visualization of large, complex and time-dependent 3D objects in distributed GIS*". GIS '02: Proceedings of the 10th ACM international symposium on Advances in geographic information systems, 113-118 2002.
[21]N Karonis, M Papka, J Binns, J Bresnahan, J Insley, D Jones, J Link, "*High-resolution remote rendering of large datasets in a collaborative environment*". Future Gener. Comput. Syst, 2003.
[22]S Klasky, S Ethier, Z Lin, K Martins, D McCune, R Samtaney, "*Grid-based parallel data streaming implemented for the Gyrokinetic toroidal code*". Proceedings of the ACM/IEEE SC2003 conference, 2003.
[23]W Allcock, J Bresnahan, R Kettimuthu, M Link, C Dumitrescu, I Raicu, I Foster, "*The Globus Striped GridFTP Framework and Server*". Proceedings of Super Computing 2005, 2005.
[24]Site Internet, "*gsoap*". http://www.cs.fsu.edu/ engelen/soap.html, .
[25] K Brodlie, J Wood, D Duceand, M Sagar, "*gViz: Visualization and Computational Steering on the Grid*". *UK e-Science All Hands Meeting*, 2004.
[26]R Haimes, "*pV3: A distributed system for large-scale unsteady CFD visualization*". In AIAA 32nd Aerospace Scienced Meeting and Exhibit, number AIAA 94-0321, 1994.
[27]White Paper, "*Advantages and Implementation of Remote Graphics Software*".
http://h20331.www2.hp.com/Hpsub/downloads/hp_remotegraphics.pdf, .
[28]Rendering, IBR-Assisted Volume, "*K. Mueller, N. Shareef, J. Huang, R. Crawfis*". Visualization'99, 1999.
[29]K Moreland, D Thompson, "*From Cluster to Wall with VTK*". PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2003.
[30]K Moreland, B Wylie, C Pavlakos, "*Sort-last parallel rendering for viewing extremely large data sets on tile displays*". PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics, 2001.
[31] RB Haber, D Mc Nabb, "*Visualization Idioms: A Conceptual Model for Scientific Visualization Systems*". *Visualization In Scientific Computing*, 1990.
[32]S Prohaska, A Hutanu, R KAhler, H Hege, "*Interactive Exploration of Large Remote Micro-CT Scans*". IEEE Visualization, 345-352 2004.
[33] D Foulser, "*IRIS Explorer: A Framework for Investigation*". *Computer Graphics*, 1995.
[34] R Wain, M Ashworth, "*A Java GUI and distributed CORBA client-server interface for a coastal ocean model*". CCLRC 2005.
[35] J Chen, I Yoon, W Bethel, "*Internet Delivery of Scientific Visualization via Structured, Prerendered Imagery*". *Accepted for Publication in Electronic Imaging 2006*, 2006.
[36]K Damevski, S Parker, "*Parallel Remote Method Invocation and M-by-N Data Redistribution*". Proceedings, Fourth LACSI Symposium, 2003.

[37]J Allard, V Gouranton, L Lecointre, E Melin, B Raffin, "*NetJuggler: Running VrJuggler with Multiple Displays on a Commodity Component Cluster*". IEEE Virtual reality, 2002.
[38]Site Internet, "*OpenDX*". http://www.opendx.org, .
[39] V Pascucci, "*ViSUS: Visualization Streams for Ultimate Scalability*". Lawrence Livermore National Laboratory 2005.
[40] C Pérez, T Priol, A Ribes, "*A Parallel CORBA Component Model*". INRIA, IRISA, Rennes, France 2002.
[41] C Pérez, T Priol, A Ribes, "*PACO++: A parallel object model for high performance distributed systems*". INRIA, IRISA, Rennes, France 2003.
[42]A Cedilnik, B Geveci, K Moreland, J Ahrens, J Bavre, "*Remote Large Data Visualization in the ParaView Framework*". EuroGraphics Symposium on Parallel Graphics and Visualization, 1-9 2006.
[43]Site Internet, "*PV3*". http://raphael.mit.edu/pv3/pv3.html, .
[44]W Ligon, R Ross, "*An Overview of the Parallel Virtual File System*". Proceedings of the 1999 Extreme Linux Workshop, 1999.
[45]D M Weinstein, S G Parker, J Simpson, K Zimmerman, G Jones (2005). Visualization in the SCIRun Problem-Solving Environment. In *The Visualization handbook, C. R. Johnson Editors, Elsevier*.
[46]K Zhang, K Damevski V Venkatachalapathy, S Parker, "*SCIRun2: A CCA Framework for High Performance Computing*". Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'04), 72-79 2004.
[47]E Luke, C Hansen, "*Semotus Visum: A Flexible Remote Visualization Framework*". c-VIS2002, 61-68 2002.
[48]Site Internet, "*SOAP*". http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, .
[49]site internet, "*VirtualGL*". http://www.virtualgl.org/, .
[50]G Cortelazzo, P Zanuttigh, "*Predictive image compression for interactive remote visualization*". Proceedings of the 3rd Internaional Symposium on Image and Signal Processing and Analysis (ISPA2003), 168-173 2003.
[51]W Bethel, J Shalf, CR Hansen and C R Johnson Editors, "*Consuming network bandwith with Visapult*". The Visualization handbook, 2005.
[52]K Engel, O Sommer; T Ertl, "*A Framework for Interactive Hardware Accelerated Remote 3D-Visualization*". Data Visualization 2000, Springer Computer Science, 67-177 2000.
[53]S Stegmaier, M Magallan, T Ertl, "*A Generic Solution for Hardware-Accelerated Remote Visualization*". Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '02, 2002.
[54]S Stegmaier, J Diepstraten, M Weiler, T Ertl, "*Widening the Remote Visualization Bottleneck*". Proceedings of ISPA '03, 2003.
[55]VisIt, "*User's manual*". UCRL-MA-152039, 2003.
[56]K Engel, O Sommer, C Ernst, T Ertl, "*Remote 3D Visualization using Image-Streaming Techniques*". Advances in Intelligent Computing and Multimedia Systems (ISIMADE '99), 91-96 1999.
[57]White Paper, "*SGI OpenGL Vizserver 3.5. Visualization and Collaboration*". http://www.sgi.com/pdfs/3263.pdf, .

[58]T Richardson, Q Stafford-Fraser, K Wood, A Hopper, "*Virtual Network Computing*". IEEE Internet Computing, 33-38 1998.
[59]Site Internet, "*VRML*". http://www.w3.org/MarkUp/VRML/, .
[60]W Schroeder, K Martin, W Lorensen, "*The Visualisation Toolkit: An Object-Oriented Approach to 3D Graphics*". Hall, Prentice, 1996.
[61] D Ellsworth, B Green, C Henze, P Moran, T Sandstrom, "*Concurrent Visualization in a Production Supercomputing Environment*". *IEEE Transactions on Visualization and Computer Graphics*, 2006.
[62]J Wernecke, "*The Inventor Mentor. Programming Object-Oriented Graphics with Open Inventor*". Addison-Wesley, 1994.