Introduction à la norme DICOM et l'extension DICOM-RT
Introduction à Geant4

Serkan Seda

14 septembre 2004

Table des matières

Table des figures

Liste des tableaux

Première partie

La norme DICOM

Chapitre 1

Le Standard DICOM et l'Extension RT

1.1 Introduction à DICOM

Les croissantes évolutions des systèmes d'acquisition d'images, des systèmes d'archivage et d'information dans le cadre médical, ont produit dans les années 80 d'important besoin en connectivité et en inter-opérabilité des équipements médicaux. Afin d'aider à la manipulation et à la visualisation d'images, les professionnels du médical (notamment les radiologues) et les fabricants d'équipements médicaux ont développé dans un effort international commun le standard DICOM, **D**igital **I**maging **C0**mmunication in **M**edicine [1]. La norme a été créé par l'ACR (**A**merican **C**ollege of **R**adiology) en association avec la NEMA (**N**ational **E**lectrical **M**anufactures **A**ssociation). Elle est régulièrement mise à jour [2] par ces deux comités auxquels se sont joints d'autres comités d'experts internationaux tels le JRIA au japon, l'ANSI aux USA, le CENTC251 en europe. La norme DICOM ne définit pas qu'un simple format d'image, elle est plutôt étoffée. Elle définit des méthodes de connection, de transfert et d'identification des données médicales. C'est à partir d'un « **Document de Conformité** »(*Conformance Statement*) émis par chacune des machines respectant cette norme qu'il est possible d'interconnecter des appareils. Elle garantie donc aux équipements de communiquer localement ou à distance à travers un média, tout en assurant la compatiblité des équipements. Dans ce premier chapitre, nous allons tenter de cerner cette norme, le point important est la compréhension de l'organisation de ce *document de conformité*. Il implémente principalement des classes **SOP** (***S**ervice **O**bject **P**air*) qui définissent des types de services à accomplir et les informations que les fichiers DICOM devront y contenir. C'est plus précisément sur le deuxième aspect de la définition d'une classe SOP que nous nous attardons dans la suite. Nous souhaitons comprendre comment la norme organise des informations assignées pour chaque image, afin qu'à terme nous sachions comment manipuler des fichier DICOM sans le corrompre. Et de ce qui est des services rendus, la norme DICOM permet la communication des programmes utilisateurs situés sur des machines distinctes et interconnectées avec l'implémentation de protocoles de communication de niveau 7 (couche application du modèle OSI : Open System Interconnection). Il y inclut aussi des méthodes de stockages de données sur média (*Media Storage Application Profiles*) et d'autres méthodes encore (voir PS 3.2).

1.2 Comment un fichier DICOM est organisé ?

Dans la suite, nous présentons progressivement la constitution d'un fichier DICOM.

1.2.1 Caractéristiques principales de DICOM

La production quotidienne massive d'images médicales ne peut être archivée dans un format commun de type JPEG au risque de perdre des données associéées à l'image tels que nom du patient, type d'examen, hopital, etc.... Le format DICOM permet de rendre unique chaque image produite et de lui associer des informations spécifiques. Cela a pour conséquence de produire des images autonomes dans la mesure où il est toujours possible d'identifier formellement leurs origines en cas de perte, de renommage ou de reproduction. Le format est de taille variable. Il contient des informations obligatoires et d'autres optionnelles. Chaque image DICOM contient obligatoirement plusieurs types de numéros d'identification unique **UID** (***Unique Identifier***) générés automatiquement par les appareils. Il ne peut exister deux UID identiques pour désigner des informations différentes, et ceci quelque soit la machine et sa localisation. Cette unicité est nécéssaire

non seulement pour des raisons médico/médico-légal, mais aussi pour permettre la formation et la gestion de bases de données.

Nous distinguons dans le tableau 1.1 suivant les IUD obligatoires :

SOP Class UID	Identifie le type de Service/Objet auquel est destiné l'image (voir §1.3 Principe du SOP)
Study Instance UID	Identifie un examen entier, en temps et lieu
Series Instance UID	Identifie une série d'images au sein de l'examen
SOP Instance UID	Identifie l'image associée au fichier

TAB. 1.1 – *Les UID obligatoires dans un fichier DICOM.*

En plus, le format utilise un vocabulaire contrôlé particulier à la médecine. On identifie les donnée de façon universelle quelque soit la machine. DICOM utilise la norme SNOMED [1] (Systemized Nomenclature for Medicine).
Et enfin, la norme se composent aujourd'hui de 14 parties standard [3] en plus des extensions et supléments de document [2]. Les parties qui nous concerne le plus dans le cadre de ce stage sont :

PS 3.2 : Conformance
PS 3.3 : Information Object Definition
PS 3.4 : Service Class Specifications
PS 3.5 : Data Structure and Semantics
PS 3.6 : Data Dictionnary
PS 3.10 : Media Storage and File Format For Media Interchange
...

Elle peut être adaptée plus spécifiquement à d'autres spécialités médicales tels que la cardiologie et pour ce qui nous importe : la radiothérapie.

1.2.2 Principes du SOP

La norme DICOM est un langage orientée objet. Chaque objet DICOM, le plus souvent une image, contient à la fois des informations (nom du patient, pixels de l'image, etc...) et des fonctions (imprimer, sauvegarder, etc ...) que doit subir ces informations.
Le traitement DICOM d'une information consiste donc à apparier un objet DICOM « *Information Object* »à une fonction spécifique « *Service Class* ». Cette combinaison est appelée « ***Service/Object Pair*** »ou « ***SOP*** ».

Information sur Objet + Classe de Service = SOP
ou par exemple :
Une Image + Imprimer = Un service DICOM

Cette parité Objet/Service est l'élément principal de la conformité à la norme. Elle est identifiée par un identifiant unique UID : SOP Class UID (voir PS 3.6/[Annexe A page 77]). Pour ce conformer à une *Classe* de *Parité Objet/Service*, une machine (*Application Entity AE*) doit pouvoir gérer un type particulier d'image et réaliser un type spécifique de traitement (ou service) correspondant à la définition de « Classe de Parité »(voir PS 3.4). Le tableau 1.2 ci-dessous présente les principales « Classes de Service »actuellement disponible dans la norme.

De plus cette classe doit spécifier si le service DICOM est employé en tant qu'utilisateur (*Service Class User : SCU*) ou en tant que fournisseur (*Service Class Provider : SCP*). Pour illustrer et présenter le contenue d'une classe SOP, en voici un exemple tableau 1.3 qui se rapporte à l'acquisition d'une image par scanner **CT** (Computer Tomography). On peut donc lui assigner la fonction couramment utiliser ; le « Storage »identifié par l'UID 1.2.840.10008.5.1.4.1.1.2 (voir PS 3.6/[Annexe A page 77]).

[1] SNOMED, nomenclature systématisée de médecine, est un concept utilisé pour indexer des parties de dossier patient. Au cours de leur pratique, les médecins créent les descriptions du patient qui font partie de l'enregistrement patient. SNOMED est une langue pour le codage et la recherche des comptes rendus

Classes de Service :	**Type de Service :**
Verification Service Class *(voir PS 3.4 /[Annexe A page 22])*	Utilisé pour les tests et permet de savoir si les machines sont connectées. Cette classe n'est pas associée à un objet DICOM, elle renvoie l'information sous la forme d'un affichage.
Storage Service Class *(voir PS 3.4 /[Annexe B page 24])*	Permet la sauvegarde et le transfert des images entre deux entités applicatives DICOM.
Media Storage Service Class *(voir PS 3.4 /[Annexe I page 239])*	La variante Media Storage Service Class spécifie les échanges entre 2 machines par l'intermédiaire d'un média (CDRom, ...).
Query/Retrieve *(voir PS 3.4 /[Annexe C page 36])*	Implémente des commandes types : FIND permet de demander une liste d'image. MOVE et GET permettent d'initier un transfert effectué via la classe Storage Service Class.
Study Contents Notification *(voir PS 3.4 /[Annexe D page 82])*	Utilisée pour notifier l'arrivée d'une nouvelle image ou série d'images.
Print Management *(voir PS 3.4 /[Annexe H page 185])*	Permet la connection avec un reprographe, spécifie le type d'image.
Patient Management *(voir PS 3.4 /[Annexe E page 85])*	Permet d'interfacer la machine au réseau hospitalier. Gestion des données des patients, ..., admission et sortie des patients.
Study Management *(voir PS 3.4 /[Annexe F page 102])*	Création, gestion de rendez-vous, suivi des examens.
Result Management *(voir PS 3.4 /[Annexe G page 168])*	Permet la gestion des résultats des examens.
Et d'autres encore	

TAB. 1.2 – *Les principales classes de service.*

Information Entity	**Modules**	**Référence à PS 3.3**
Patient	Patient	C.7.1.1
	Clinical Trial Subject	C.7.1.3
Study	General Study	C.7.2.1
	Patient Study	C.7.2.2
	Clinical Trial Study	C.7.2.3
Series	General Series	C.7.3.1
	Clinical Trial Series	C.7.3.2
Frame of Reference	Frame of Reference	C.7.4.1
Equipement	General Equipement	C.7.5.1
Image	General Image	C.7.6.1
	Image Plan	C.7.6.2
	Image Pixel	C.7.6.3
	Contrast/bolus	C.7.6.4
	CT image	C.8.2.1
	Overlay Plane	C.9.2
	VOI LUT	C.11.2
	SOP Common	C.12.1

TAB. 1.3 – *Les modules contenus dans un objet CT Image.*

Nous y trouvons des modules sur le patient, l'étude, les séries, l'image de référence, l'équipement et l'image. Chaque module, faisant référence à une section du PS 3.3 "Information Object Definition", possède des attributs spécifiques à définir (dont les valeurs sont mentionnées).

1.2.3 Le format de fichier DICOM

Le format de fichier DICOM fournie un moyen d'encapsuler l'ensemble des données representée par une « Instance SOP »relative à une définition de l'objet d'une classe SOP. Comme nous le montre le figure 1.1 suivant, les fichiers contenant une instance se succèdent et le tout constitue un ensemble de fichiers DICOM (voir PS 3.10/ [section 7 page 20]). Dans un fichier DICOM, les données sont organisées sous une forme séquentielle en commencant par une entête et suivi des données brutes de l'image.

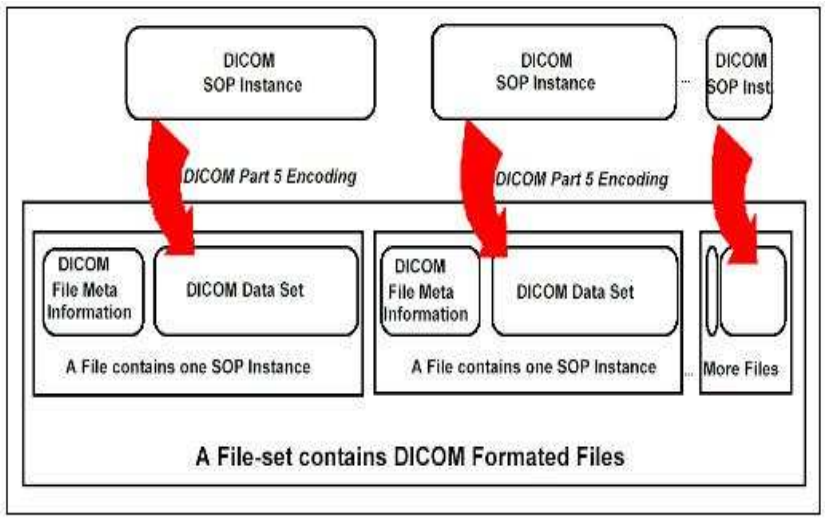


FIG. 1.1 – *Ensemble de fichiers DICOM*

L'entête

Chaque entête commence par un préambule de 128 octets généralement mis à zéro suivi de 4 octets pour y inscrire les caractères 'D'.'I'.'C'.'M' (voir PS 3.10/[section 7.1 page 20]). A la suite du préambule, toutes sortes d'informations se succèdent. Elles sont organisées en plusieurs groupes d'informations. Chaque élément de données est constituée de 3 champs de données si VR est implicite, et de 4 champs si VR est explicite (voir PS 3.5/[section 7 page 31]). L'ensemble de ces données représente un dictionnaire de données DICOM.

1. Data Tag Element :
   est une paire ordonnée de 16 bits d'entiers non signés représentant le numéro du groupe et suivi par le numéro de l'élément. Elle indique le type d'information qui va suivre. Elle est décomposée en 2 séries de 2 octets, les 2 premiers octets codent un groupe d'information (exemple « information patient » : 0010 en hexadécimal) et les 2 octets suivant précisent l'élément du groupe (exemple « âge du patient » : 0010 en hexadécimal). Il existe une liste complète des valeurs possibles pour le champs "Tag" (voir PS 3.6). Le tableau 1.4 indique les principaux groupes d'information les plus fréquement utilisés :

Groupe d'information	Signification
0008	- Méta-information (File Meta Information Version, SOP Class UID, SOP Instance UID, Transfert Syntax UID, ...), Identification du centre (Date d'examen, Type d'examen, Fabricant de la machine, Hôpital, Identification de la machine, ...).
0010	- Les informations sur le patient (Nom, Identification, Date de naissance, Sexe, ...)
0018	- Les informations sur l'acquisition de l'information (épaisseur de coupe, bolus, inclinaison du statif, paramètres kV, temps d'écho, position du patient, etc...).
0020	- Positionnement et information relatives à l'acquisition (Orientation du patient, Série, plan de référence, Nombres d'images dans l'acquisition, StudyInstanceUID, SeriesInstanceUID, ...).
0028	- Présentation de l'image (dimensions, niveaux de gris, tables de couleurs, bits alloués, bits stockés, bit le plus significatif, WindowWidth, WindowCenter...).
4000	- Texte
7FE0	- Pixels de l'image (pixel data)

TAB. 1.4 – *Quelques groupes d'informations codés en héxadécimal.*

2. Value Representation :
   est une chaîne de deux caractères de 2 octets associée à la valeur du Data Tag Element précédent. Elle indique la longueur de l'information contenue dans le dernier champs jusqu'au Tag suivant

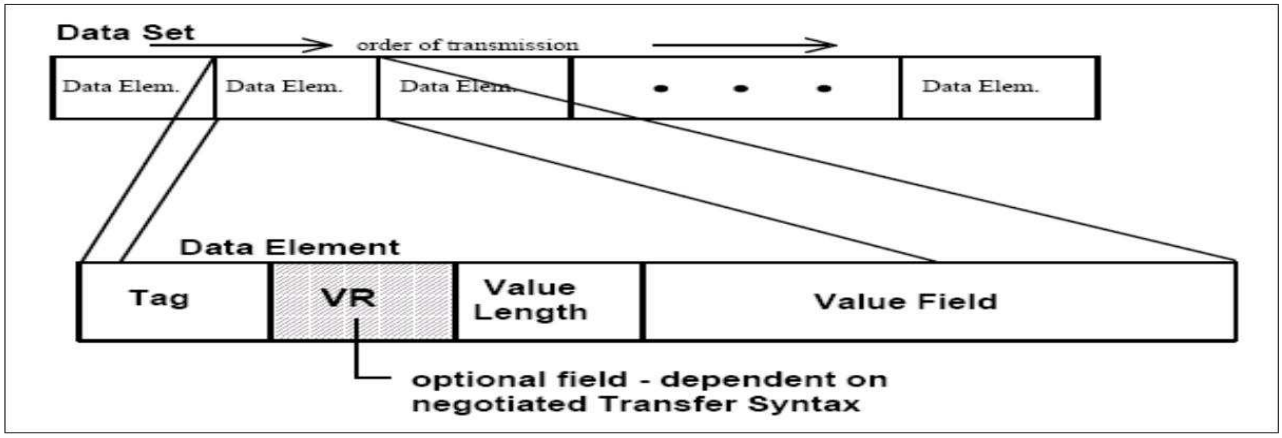


FIG. 1.2 – *Flux de données de l'entête.*

(exemple l'information sur l'âge d'un patient sera codé sur 2 octets et représenter par US) (voir PS 3.5/[section 6.2 page 21]). Le tableau 1.5 décrit les différents attributs de VR présent dans un dictionnaire de données DICOM.

3. Value Length :
   - est soit un entier non signé de 16 ou 32 bits dépedant d'un VR explicite ou implicite, et contenant la longueur explicite du champs de valeur en nombre d'octet.
   - est soit un ensemble de champs de longueur 32 bits. Les longueurs indéfiniées pourrait être utilisées pour les éléments de données ayant le VR égal à SQ (Sequence of Items) et UN (Unknown) ou autres (voir à la suite).
4. Value Field :

   est un champs de valeur de longueur variable. Il correspond à l'information identifié par le premier champs. Le type de données de valeurs enregistré dans ce champs est spécifié par le VR de l'élément de données. Le VR, pour un Data Element Tag, est donné en utilisant le dictionnaire de données de la partie 6 (voir PS 3.6).

Structure d'élément de données avec un VR explicite :

Quand en utilisant la structure d'un VR explicite, l'élément de données sera construit de 4 champs consécutifs : Data Element Tag, VR, Value Length, et Value. Selon le VR, il y deux constructions possible. Il y a donc VR égal à OB, OW, OF, SQ , UT ou Un (voir tableau 1.6). Et il y a VR égal aux autres (voir tableau 1.7).

Structure d'élément de données avec un VR implicite :

Quand en utilisant la structure d'un VR implicite, l'élément de données sera construit de 3 champs consécutifs : Data Element Tag, Value Length, et Value. Si la valeur du champs a une longueur explicite alors la longueur de la valeur contiendra une valeur égal à la valeur (en octets) d'un champs de valeur (voir tableau 1.8).

La structure avec un VR explicite ou implicite dépend du choix du Transfert Syntax identifié par un UID (voir PS 3.6/[Annexe A page 77]). L'ensemble des données de l'entête en transmission peut être représenté dans un unique tableau selon la figure 1.2 .
En général, le flux de données DICOM est en hexadécimal. Les flux d'information se succèdent à la suite les uns des autres dans un ordre de balises croissant. Pour illustrer et avec l'aide d'une table de correspondance hexadécimal/ASCIIétendu (voir Annexe D), on détaille le flux d'un élément de données i d'un ensemble de données ayant une structure avec un VR implicite. Cet élément de donnée i se décompose comme suit :

VR name	Definition	Lentgth of Value
AE : Application Entity	Chaine de caractères relatif à un appareil	16 octets maximum
AS : Age String	Chaine de caractères relatif à un âge	Fixé à 4 octets
AT : Attribute Tag	Paire ordonnée d'entiers non signés de 16 bits qui représente la valeur d'une balise d'élément de données	Fixé à 4 octets
CS : Code String	A string of characters with leading or trailing spaces being non-significant.	16 octets maximum
DA : Date	Chaine de caractères relatif à une date	Fixé à 8 octets
DS : Decimal String	A string of characters representing either a fixed point number or a floating point number.	16 octets maximum
DT : Date Time	The Date Time common data type. Indicates a concatenated date-time ASCII string in the format : YYYYMMDDHHMMSS.FFFFFF ZZZZ. The components of the string, from left to right, are YYYY=Minute, SS=Second, FFFFF=Fractional Second and ZZZZ=hours and Minutes of offser.	26 octets maximum
FL : Floating Point Single	Single precision binary floating point number represented in 32-bit Floating Point Number Format.	Fixé à 4 octets
FD : Floating Point Double	Double precision binary floating point number represented in 64-bit Floating Point Number Format.	Fixé à 8 octets
IS : Integer String	Chaine de caractères représentant un entier en décimal.	12 octetc maximum
LO : Long String	A character string that may be padded with leading and/r trailing spaces.	64 chars maximum
LT : Long Text	Chaine de caractères long relatif à un ou plusieurs paragraphes.	10240 chars maximum
OB : Other Byte String	A string of bytes where the encoding of the contents is specified by the negotiated Transfer Syntax. OB is a VR wich is insensitive to Little/Big Endian byte ordering (see PS 3.5 section 7.3).	Voir la définition de la Syntaxe de Transfert
OF : Other Float String	string of 32-bit floating point words. OF is a VR which requires byte swapping within each word when changing between Little Endian and Big Endian byte ordering.	2-4 maximum
OW : Other Word String	A string of 16-bit words where the encoding of the contents is specified by the negotiated Transfert Syntax.	Voir la définition de la Syntaxe de Transfert
PN : Person Name	A character string encoded using a 5 component convention.	64 chars maximum par composant de groupe
SH : Short String	A character string that may be padded wit leading and/or trailing spaces.	16 chars maximum
SL : Signed Long	Signed binary integer 32 bits long.	Fixé à 4 octets
SQ : Sequence of Items	Value is a Sequence of zero or more Items.	Non applicable (voir PS 3.5 section 7.5)
SS : Signed Short	Signed binary integer 16 bits long.	Fixé à 2 octets
ST : Short Text	Chaine de caractères qui peut contenir un ou plus de paragraphes.	1024 chars maximum
TM : Time	A string of characters of the format hhmmss.frac; frac contains a fractional part of a second as small as 1 millionth of a second (range "000000"-"999999").	16 octets maximum
UI : Unique Identifier (UID)	A character string containing a UID that is used to uniquely identify a wide variety of items. The UID is a series of numeric components separated by the period "." character.	64 octets maximum
UL : Unsigned Long	Entier long non signé de 32 bits.	Fixé à 4 octets
UN : Unknown	A string of bytes where the encoding of the contents is unknown (voir PS 3.5 section 6.2.2)	Any length valid for any of the other DICOM Value Representations
US : Unsigned Short	Entier long non signé de 16 bits	Fixé à 2 octets
UT : Unlimited Text	Chaine de caractères relatif à un ou plusieurs paragraphes.	232-2 chars

TAB. 1.5 – *Descriptif général des types d'attribut de VR.*

Tag		VR		Value Length	Value
Group Number (16-bit unsigned integer)	Element Number (16-bit unsigned integer)	VR (2 byte character string) of "OB", "OW", "OF", "SQ", "UT" or "UN"	Reserved (2 bytes) set to a value of 0000H	32-bit unsigned integer	Even number of bytes containing the Data Element Value(s) encoded according to the VR and negotiated Transfer Syntax. Delimited with Sequence Delimitation Item if of Undefined Length.
2 bytes	2 bytes	2 bytes	2 bytes	4 bytes	'Value Length' bytes if of Explicit Length

TAB. 1.6 – *Structure élément de données avec VR explicite égal à OB, OW, OF, SQ, UT ou UN.*

Tag		VR	Value Length	Value
Group Number (16-bit unsigned integer)	Element Number (16-bit unsigned integer)	VR (2 byte character string)	(16-bit unsigned integer)	Even number of bytes containing the Data Element Value(s) encoded according to the VR and negotiated Transfer Syntax.
2 bytes	2 bytes	2 bytes	2 bytes	'Value Length' bytes

TAB. 1.7 – *Structure élément de données avec VR explicite égal aux compléments.*

Tag		Value Length	Value
Group Number (16-bit unsigned integer)	Element Number (16-bit unsigned integer)	32-bit unsigned integer	Even number of bytes containing the Data Elements Value encoded according to the VR specified in PS 3.6 and the negotiated Transfer Syntax. Delimited with Sequence Delimitation Item if of Undefined Length.
2 bytes	2 bytes	4 bytes	'Value Length' bytes or Undefined Length

TAB. 1.8 – *Structure élément de données avec VR implicite.*

...	00 00 00 0A 53 45 52 4B 41 4E 53 45 44 41	...
Elément i-1	Elément i	Elément i+1

Tag		**Value Length**	**Value**
2 octets balise groupe	2 octets balise élément	4 octets pour code la longueur L du champs suivant	Information (sur L=10 octets), écrit ici en hexadécimal.
00 10	00 10	00 00 00 0A	53 45 52 4B 41 4E 53 45 44 41
information sur le patient.	information sera le nom du patient.	Indique la longueur du champ suivant en nombre d'octets. Ici L = 10 octets réservés dans le champ Value.	S E R K A N S E D A

Sur cette exemple tiré d'un fichier patient du CLB dont on gardera l'identité anonyme, il a été ajouté en rouge les valeurs des balises pour vérifier que celles-ci sont ordonnées dans le sens croissant des valeurs. Notons que les attributs des modules sont quelque peu éparpillés dans l'ensemble du fichier DICOM. En grand et gras, l'attribut *SOP class UID* identifie le type de service/objet qui est effectuer pour obtenir ce fichier, à savoir **CT Image Storage : une acquisition d'image au scanner pour la sauvegarde**.

```
GroupLength          584 ->(0008,0000)
SpecificCharacterSet          ISO-IR 100 -> module SOP Common (0008,0005)
ImageType          ORIGINAL -> module General Image (0008,0008)
ImageType          PRIMARY -> module General Image (0008,0008)
ImageType          AXIAL -> module General Image (0008,0008)
InstanceCreationDate          2004-Mar-17 -> module SOP Common (0008,0012)
InstanceCreationTime          121031.730 -> module SOP Common (0008,0013)
SOPClassUID          1.2.840.10008.5.1.4.1.1.2 -» CT Image Storage (0008,0016)
SOPInstanceUID          2.16.840.1.113662.2.1.4419.431059.4121031.420040317.4121032860 -> (0008,0018)
StudyDate          2004-Mar-17 -> module General Study (0008,0020)
SeriesDate          2004-Mar-17 -> module General Series (0008,0021)
AcquisitionDate          2004-Mar-17 -> module General Image (0008,0022)
Image/ContentDate          2004-Mar-17 -> module General Image (0008,0023)
StudyTime          121031.730 -> module General Study (0008,0030)
SeriesTime          121031.730 -> module General Series (0008,0031)
AcquisitionTime          121031.730 -> module General Image (0008,0032)
Image/ContentTime          121032.860 -> module General Image (0008,0033)
Modality          CT -> module General Series (0008,0060)
Manufacturer          Picker International, Inc. -> module General Equipement (0008,0070)
InstitutionName          CENTRE LEON BERARD LYON-» module General Equipement (0008,0080)
StationName          Picker CT -» module General Equipement (0008,1010)
Manufacturer'sModelName          PQ2000 -» module General Equipement (0008,1090)
GroupLength          310 ->(0010,0000)
Patient'sName          XXXX XXXXXX -» module Patient (0010,0010)
PatientID          XXXX855 -» module Patient (0010,0020)
PatientComments          -» module Patient (0010,4000)
PatientComments          -» module Patient (0010,4000)
PatientComments          -» module Patient (0010,4000)
PatientComments          -» module Patient (0010,4000)
GroupLength          188 ->(0018,0000)
Contrast/BolusAgent          DOSI -» module contrast/bolus (0018,0010)
SliceThickness          5.0 -» module Image Plan (0018,0050)
KVP          120 -> module CT Image (0018,0060)
DeviceSerialNumber          419 -» module General Equipement (0018,1000)
SoftwareVersion(s)          4.6B1 (Q -» module General Equipement (0018,1020)
ProtocolName          SMALL ABDOMEN -» module Image Plan (0018,1030)
Gantry/DetectorTilt          .0 -» module CT Image (0018,1120)
```

```
RotationDirection          CW -» module CT Image (0018,1140)
ExposureTime          4213 -» module CT Image (0018,1150)
X-rayTubeCurrent          65 -» module CT Image (0018,1151)
Exposure          273 -» module CT Image (0018,1152)
FilterType          0 -» module CT Image (0018,1160)
FocalSpot(s)          1 -» module CT Image (0018,1190)
PatientPosition          HFS-» module General Series (0018,5100)
GroupLength          404 -»(0020,0000)
StudyInstanceUID          2.16.840.1.113662.2.1.1419.131059.1040317.1115915 -» module General Study (0020,000D)
SeriesInstanceUID          2.16.840.1.113662.2.1.2419.231059.2040317.2115915.239505444.20 -» module General Series (0020,000E)
StudyID          31059 -» module General Study (0020,0010)
SeriesNumber          3647 -» module General Series (0020,0011)
AcquisitionNumber          3647 -» module General Image (0020,0012)
InstanceNumber          21 -» module General Image (0020,0013)
PatientOrientation          L -» module General Image (0020,0020)
PatientOrientation          P -» module General Image (0020,0020)
ImagePosition(Patient)          -2.245605e02 -» module Image Plan (0020,0020)
ImagePosition(Patient)          -5.115605e02 -» module Image Plan (0020,0020)
ImagePosition(Patient)          -8.265599e02 -» module Image Plan (0020,0020)
ImageOrientation(Patient)          1.000000e00 -» module Image Plan (0020,0037)
ImageOrientation(Patient)          0.000000e00 -» module Image Plan (0020,0037)
ImageOrientation(Patient)          0.000000e00 -» module Image Plan (0020,0037)
ImageOrientation(Patient)          0.000000e00 -» module Image Plan (0020,0037)
ImageOrientation(Patient)          1.000000e00 -» module Image Plan (0020,0037)
ImageOrientation(Patient)          0.000000e00 -» module Image Plan (0020,0037)
FrameofReferenceUID          2.16.840.1.113662.2.1.3419.331059.3040317.3115915 -» module Frame of Reference(0020,0052)
SliceLocation          826.56 -» module Image Plan (0020,1041)
GroupLength          176 -»(0028,0000)
SamplesperPixel          1 -» module Image Pixel (0028,0002)
PhotometricInterpretation          MONOCHROME2 -» module Image Pixel (0028,0004)
Rows          512 -» module Image Pixel (0028,0010)
Columns          512 -» module Image Pixel (0028,0011)
PixelSpacing          8.789062e-01 -» module Image Plan (0028,0030)
PixelSpacing          8.789062e-01 -» module Image Plan (0028,0030)
BitsAllocated          16 -» module Image Pixel (0028,0100)
BitsStored          16 -» module Image Pixel (0028,0101)
HighBit          15 -» module Image Pixel (0028,0102)
PixelRepresentation          1 -» module Image Pixel (0028,0103)
WindowCenter          -63 -» module VOI LUT (0028,1050)
WindowWidth          490 -» module VOI LUT (0028,1051)
RescaleIntercept          0 -» moduel CT Image (0028,1052)
RescaleSlope          1 -» moduel CT Image (0028,1052)
GroupLength          524296 -»(7FE0,0000)
PixelData          16-bit binary data :0,64640,64653,64638,64655,64641,64661,64642,64650,64654,...(262144 bytes) -» module Image
Pixel (7FE0,0010)
```

A ces informations, suivent les données brutes de l'image.

Les images

Les images succèdent à l'entête. Elles peuvent être compressées au format JPEG par l'intermédiaire d'un groupe d'élément « Transfer Syntax UID »(voir PS 3.6/[Annexe A page 77]). La façon dont les pixels sont sauvegardés peut énormément changer. Selon l'architecture de l'ordinateur, le problème de l'ordonnancement des bits entervient lors de la lecture (little/big endian).

Par exemple, un pixel qui a pour valeur 258 (en décimal) peut être représenté selon l'ordre « Little Endian »ou « Big Endian »en binaire.

- Big Endian :

15	14	13	12	11	10	9	**8**	7	6	5	4	3	2	**1**	0	**Bit n**
32768	16384	8192	4096	2048	1024	512	**256**	128	64	32	16	8	4	**2**	1	**Valeur 2^n**
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**octets**

MSB : Most Signifiant Bit **LSB** : Least Signifiant Bit

- Little Endian :

0	**1**	2	3	4	5	6	7	**8**	9	10	11	12	13	14	15
1	**2**	4	8	16	32	64	128	**256**	512	1024	2048	4096	8192	16384	32768
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0

LSB **MSB**

Le souci de ces modes little ou big endian se pose au niveau de l'ordonnancement des octets (voir annexe B). Chaque pixel d'une image est codé sur n bits et ordonnancés en mot (groupement de d'octets) (voir PS 3.5 /[Annexe D page 70][PS 3.3/section C.7.6.3 page 238]). Dans un cas où l'information des pixels est codée sur 16 bits et contenue dans des mots de 16 bits, l'ordonnancement des mots se succèdent simplement les uns derrières les autres comme dans la figure 1.3.

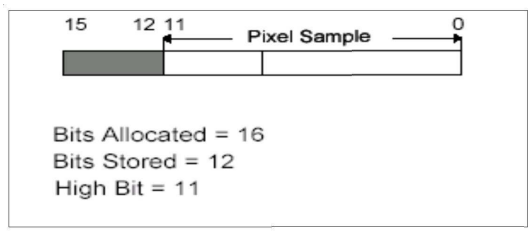


FIG. 1.3 – *Il a été alloué 16 bits pour coder un pixel. Seulement 12 bits suffisent pour stocker l'information. Le bit le plus significatif est en position 11.*

Ensuite, chaque mots se succèdent et renferme l'information d'un pixel comme sur la figure 1.4.

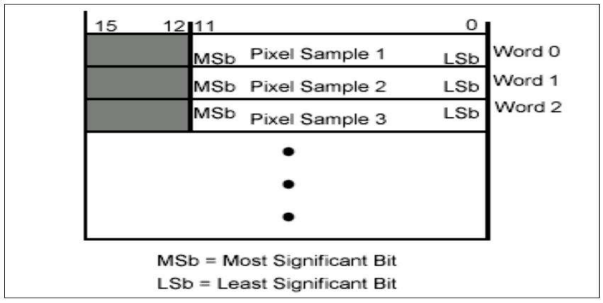


FIG. 1.4 – *Agencement des mots représentant les pixels.*

Nous pouvons aussi coder l'information sur 24 bits en ayant seulement besoin de stocker 18 bits, et choisir le bit le plus significatif à la position 19. Dans cette situation l'ordonnancement des mots se complique comme dans la figure 1.5.

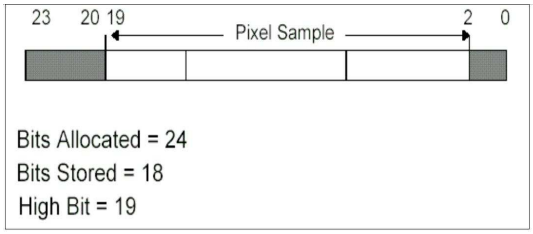


FIG. 1.5 – *Il a été alloué 24 bits pour coder un pixel. Seulement 18 bits suffisent pour stocker l'information. Le bit le plus significatif est choisi en position 19.*

Ensuite, si en plus nous souhaitons ordonnancer ces données en mots de 16 bits, chaque échantillon de pixel sera étallé sur deux mots comme sur le figure 1.6.

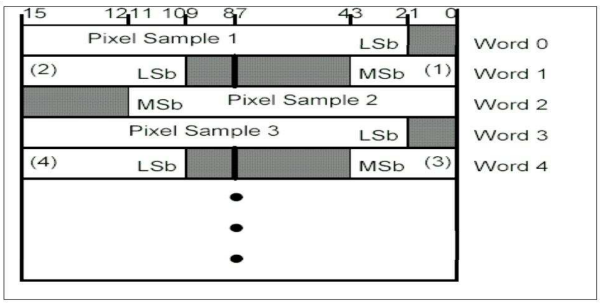


FIG. 1.6 – *Agencement de chaque pixel sur 2 mots (relatif au VR = OW).*

En général, nous retiendrons que les processeurs Intel ordonnance les octets en mode *Little Endian* et, les Motorola (MAC) et Sparc (SUN) en mode *Big Endian*.

Nous avons vu pour l'essentiel comment se structure une Ğ SOP class ğ; d'une classe de service et d'une classe d'objet. Celle-ci est composé lui même par une entête et les données images. D'autre Ğ SOP class ğ peuvent être ajouter à la suite dans un fichier DICOM. A la suite, nous poursuivons sur ce qui nous importe le plus; c'est-à-dire les aspects de l'extension RT de la norme DICOM.

1.2.4 Les aspects radiothérapie de DICOM

La radiothérapie est une des principales techniques de traitement du cancer dont le principe consiste à irradier les tumeurs à l'aide de rayon X tout en essayant d'épargner au maximum les tissus sains environnants. C'est un traitement qui repose en grande partie sur des techniques d'analyses et de traitement d'images. Toute la gestion des images en radiothérapie s'appuie sur le standard DICOM-RT. Ce protocole inclut les images 2D/3D utilisées mais également un ensemble de données, le plus souvent géométriques, associées au traitement.

En premier lieu, une classe objet Ğ CT Image ğ est à réaliser pour acquérir les données images du patient avant tout traitement comme par exemple le fichier DICOM vu précédemment au tableau 1.3. Ce n'est qu'en deuxième lieu que les objets RT se suivent. Durant l'évolution de l'étude et l'agencement successif des objets dans le temps, plusieurs acteurs (physiciens, médecins, ...) interviennent. Le flux de travail peut se décliner selon la figure 1.7.

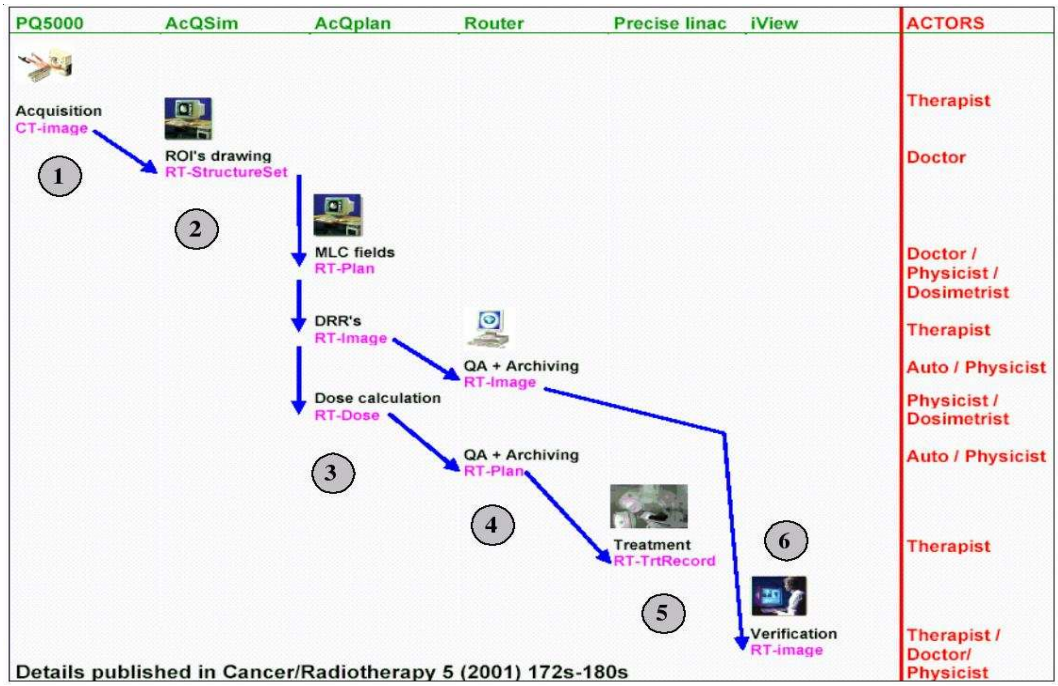


FIG. 1.7 – *Flux de travail RT. 1 : Acquisition d'image. 2 : Détermination des ROI (Tumeurs, Organes, ...). 3 : Création du plan de traitement (Dosimétrie). 4 : Stockage. 5 : Réalisation de plan de traitement. 6 : Contrôle du plan de traitement.*

Aujourd'hui, le standard DICOM-RT implémente ces objets identifiés chacun par une *SOP CLASS IUD*. De plus l'étude liée à la radiothérapie génère des dépendences qui sont réprésenter par la figure 1.8. Une étude référence plusieurs objets RT et ces derniers, pour la plupart, se dépendent réciproquement.

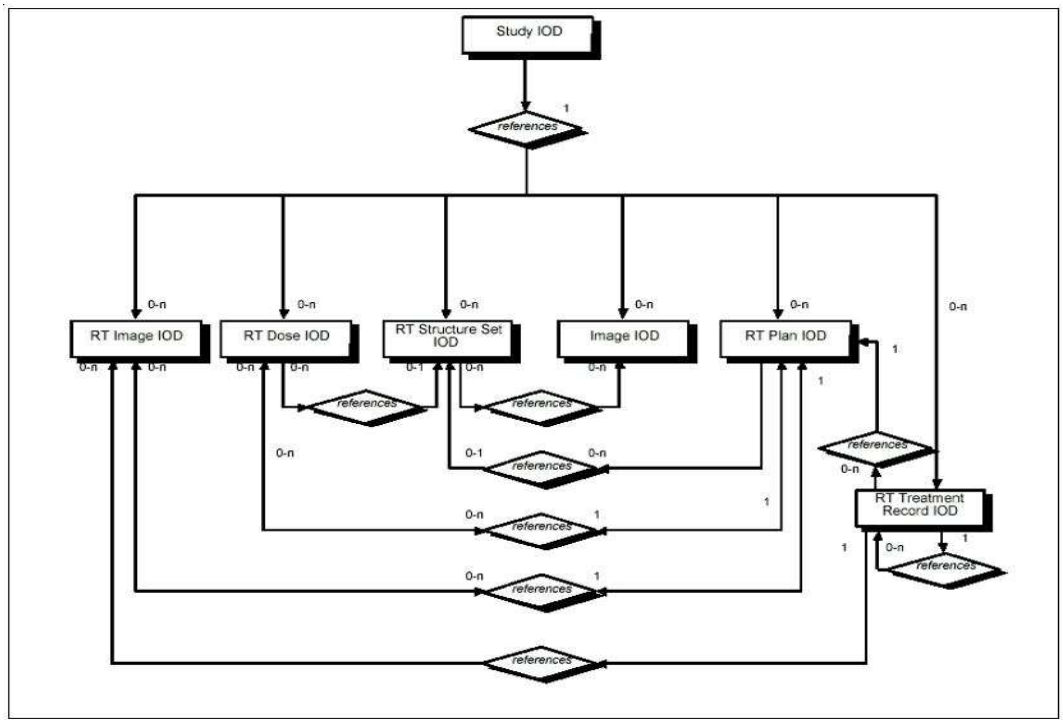


FIG. 1.8 – *Diagramme d'entité association décrviant les dépendences entre les objets RT.*

1.2.5 Les objets DICOM RT

L'extension RT est essentiellement composée de 4 objets.

RT Image

Cet objet contient des informations sur les images liées à la radiothérapie notamment des modules, RT Series,Cet objet contient des informations sur les images liées à la radiothérapie notamment des modules, RT Series, RT Image, Cine, Multi-Frame et autres (voir PS 3.3/[section A.17 page 100]). Le tableau 1.9 liste les modules utilisés dans cet objet avec la référence aux modules.

Information Entity	**Modules**	**Référence à PS 3.3**
Patient	Patient	C.7.1.1
	Clinical Trial Subject	C.7.1.3
Study	General Study	C.7.2.1
	Patient Study	C.7.2.2
	Clinical Trial Study	C.7.2.3
Series	RT Series	C.8.8.1
	Clinical Trial Series	C.7.3.2
Frame of Reference	Frame of Reference	C.7.4.1
Equipement	General Equipement	C.7.5.1
Image	General Image	C.7.6.1
	Image Pixel	C.7.6.3
	Contrast/bolus	C.7.6.4
	Cine	C.7.6.5
	Multi-Frame	C.7.6.6
	RT Image	C.8.8.2
	Modality LUT	C.11.1
	VOI LUT	C.11.2
	Approval	C.8.8.16
	Curve	C.10.2
	Audio	C.10.3
	SOP Common	C.12.1
	Curve	C.10.2
	Audio	C.10.3
	SOP Common	C.12.1

TAB. 1.9 – *Les modules contenus dans un objet RT Image.*

RT Structure Set

Cet objet contient les informations liées à l'anatomie du patient par exemple position et forme des organes ciblés. De suite, le tableau 1.10 liste les modules utilisés dans cet objet avec la référence aux modules (voir PS 3.3/[section A.19 page 105]).
Nous nous intéressons notamment à quelques attributs des trois modules suivants qui nous permet d'apporter une description des contours d'un organe d'intérêt.

1. ***Structure Set* (voir PS 3.3/[page 404]) :**
   Ce module réunit un ensemble d'attributs permettant de définir un ensemble de régions significatives. Chaque région est associée à une image de référence « Frame of Reference ». L'information qui est transférée à chaque région d'intérêt, inclue les paramètres d'affichage et géométrique et la technique de génération.

ROI Number	(3006,0022)	Nombre de Région d'Intérêt.
ROI Name	(3006,0026)	Nom associé à chaque région d'intérêt.
...	...	...

Information Entity	**Modules**	**Référence à PS 3.3**
Patient	Patient	C.7.1.1
	Clinical Trial Subject	C.7.1.3
Study	General Study	C.7.2.1
	Patient Study	C.7.2.2
	Clinical Trial Study	C.7.2.3
Series	RT Series	C.8.8.1
	Clinical Trial Series	C.7.3.2
Equipement	General Equipement	C.7.5.1
Structure Set	Structure Set	C.8.8.5
	ROI Contour	C.8.8.6
	RT ROI Contour	C.8.8.8
	Approval	C.8.8.16
	Audio	C.10.3
	SOP Common	C.12.1

TAB. 1.10 – *Les modules contenus dans un objet RT Structure Set.*

2. ***ROI Contour* (voir PS 3.3/[page 406]) :**
Ce module est utilisé pour définir la région d'intérêt comme un ensemble de coutours. Chaque ROI contient une séquence d'une ou plusieurs coutours. Le contour peut être un simple points ou plusieurs points représentant un polygone ouvert ou fermé.

ROI Display Color	(3006,002A)	Représentation en couleur (RGB) pour une région d'intérêt.
Contour Sequence	(3006,0040)	Séquence de contour définissant la région d'intérêt. Peut inclure une ou plusieurs items dans une même région d'intérêt.
Number of Contour Points	(3006,0046)	Nombre de points (triplet) dans la donnée de contour (3006,0050).
Contour Data	(3006,0050)	Séquence de triplets (x,y,z) définissant un contour dans le système de coordonnées du patient décrit au C.7.6.2.1.1 en mm.
Contour Geometric Type	(3006,0042)	Type géométrique de contour : **POINT** = simple point définissant une localisation spécifique. **OPEN-PLANAR** = contour ouvert contenant des points coplanaires. Cela signifie que le dernier sommet ne sera pas relié au premier point, et que tous les points dans Contour Data seront coplanaires. **OPEN-NONPLANAR** = contour ouvert contenant des points non-coplanaires. Cela signifie que le dernier sommet ne sera pas relié au premier point, et que les points dans Contour Data pourraient être non-coplanaires. **CLOSED-PLANAR** = contour fermé (polygone) contenant des points coplanaire. Cela signifie que le dernier point sera relié au premier point où celui-ci n'est pas repété dans Contour Data. Tous les points dans Contour Data seront coplanaires.
...	...	...

3. ***RT ROI Observation* (voir PS 3.3/ [page 409])** :
Ce module traduit l'identification et l'interprétation d'une région d'intérêt spécifiée dans les modules *Structure Set* et *ROI Contour*.

RT ROI Interpreted Type	(3006,00A4)	Type de région d'intérêt : EXTERNAL, PTV, CTV, GTV, TREATED-VOLUME, IRRAD-VOLUME, BOLUS, AVOIDANCE, ORGAN, MARKER, REGISTRATION, ISOCENTER, CONTRAST-AGENT, CAVITY, BRACHY-CHANNEL, BRACHY-SRC-APP, BRACHY-CHNL-SHLD
...	...	...

RT Plan

Cet objet contient les données géométriques et dosimétriques spécifique à un faisceau X extérieur par exemple les angles et les intensités des faisceaux d'irradiation. Une instance de l'objet RT Plan référence généralement une instance de l'objet RT Structure Set pour définir un système de coordonnées et les structures de l'ensemble du patient. Le tableau 1.11 liste les modules utilisés dans cet objet avec la référence aux modules (voir PS 3.3/[sectionA.20 page 107]).

Information Entity	**Modules**	**Référence à PS 3.3**
Patient	Patient	C.7.1.1
	Clinical Trial Subject	C.7.1.3
Study	General Study	C.7.2.1
	Patient Study	C.7.2.2
	Clinical Trial Study	C.7.2.3
Series	RT Series	C.8.8.1
	Clinical Trial Series	C.7.3.2
Frame of Reference	Frame of Reference	C.7.4.1
Equipement	General Equipement	C.7.5.1
Plan	RT General Plan	C.8.8.9
	RT Prescription	C.8.8.10
	RT Tolerance Tables	C.8.8.11
	RT Patient Setup	C.8.8.12
	RT Fraction Scheme	C.8.8.13
	RT Beams	C.8.8.14
	RT Brachy Application Setup	C.8.8.15
	Approval	C.8.8.16
	Audio	C.10.3
	SOP Common	C.12.1

TAB. 1.11 – *Les modules contenus dans un objet RT Structure Set.*

RT Dose

Cet objet contient les données des doses à préscrire selon les systèmes de planification du traitement. On y trouve par exemple les données de doses tri-dimensionnel, des courbes d'isodose, etc... Le tableau 1.12 liste les modules utilisés dans cet objet avec la référence aux modules (voir PS 3.3/[section A.18 page 103]).

Information Entity	**Modules**	**Référence à PS 3.3**
Patient	Patient	C.7.1.1
	Clinical Trial Subject	C.7.1.3
Study	General Study	C.7.2.1
	Patient Study	C.7.2.2
	Clinical Trial Study	C.7.2.3
Series	RT Series	C.8.8.1
	Clinical Trial Series	C.7.3.2
Frame of Reference	Frame of Reference	C.7.4.1
Equipement	General Equipement	C.7.5.1
Dose	General Image	C.7.6.1
	Image Pixel	C.7.6.3
	Image Plan	C.7.6.2
	Multi-Frame	C.7.6.6
	Overlay Plan	C.9.2
	Multi-Frame Overlay	C.9.3
	Modality LUT	C.11.1
	RT Dose	C.8.8.3
	RT DVH	C.8.8.4
	Structure Set	C.8.8.5
	ROI Contour	C.8.8.6
	RT Dose ROI	C.8.8.7
	Audio	C.10.3
	SOP Common	C.12.1

TAB. 1.12 – *Les modules contenus dans un objet RT Structure Set.*

1.3 Conclusion

DICOM est un standard extensif qui découle d'un besoin pressant de normalisation des données en vue de la diversifications grandissante des équipements médicaux. Il réalise de nombreux services sur les fichiers de données médicales. A partir d'une classe SOP, un ensemble de module y est affecté. Et chaque module présente une liste d'attributs à définir en fonction des besoins. Ceux-ci constituent des fichiers DICOM. C'est dans l'entête que se trouve toutes les informations relative à l'environnement d'obtention des images : méta informations, patient, acquisition, image, et d'autres informations. Un fichier DICOM est représentable comme ci-contre sur le figure 1.9. Nous ajoutons à la suite d'un premier objet d'autres objets comme ceux de l'extension RT. A terme, il est souhaité qu'on puisse manipuler ces fichiers informatiques à des fins d'études radiothérapiques. Pour ce faire, il existe des utilitaires commerciaux, mais cela nécessite un investissement financier considérable. Nous préférons choisir une librairie gratuite et fléxible ; la « dicomlib »en c++.

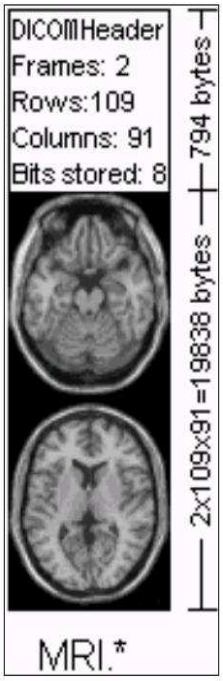


FIG. 1.9 – *Exemple d'un fichier DICOM ; une entête et les images IRM.*

Chapitre 2

Présentation et utilisation de la librairie *dicomlib*

2.1 Introduction à *dicomlib*

L'équipe d'informaticiens du département radiothérapie du cente Léon Bérard écrit les programmes et les librairies en C++. Et dans une cohérence d'opérabilité, il est naturel de choisir également une librairie en C++ pour la manipulation des fichiers DICOM. Celle qui a été choisie présente l'avantage d'être assez bien documentée [4]. Cette nouvelle librairie (version 0.8.3) a été inspiré et entièrement re-écrite à partir de la librairie ucdm99 [5] par Trevor Morgan du groupe de recherche en imagerie à Sunnybrook and Women's College Health Sciences Center, Toranto Canada. Elle implémente en grande partie la dernière version du standard DICOM. Elle utilise les mécanismes de templates pour garantir par exemple la correspondance entre les types VR de DICOM et ceux du langage C++. Elle utilise aussi les mécanismes de « serveur multithreading »en utilisant la librairie *boost : :thread*. Cela favorise une utilisation simple et rapide de la bibliothèque *dicomlib*. En plus, elle permet d'implémenter d'autres classes et fonctionnalités de DICOM selon les besoins futurs. Elle est conçu pour être portable sur des platformes Unix et Windows. En ce qui nous concerne, son installation s'est faite sous Linux SuSE 9.0 (Intel). Nous introduisons dans la suite la procédure à suivre pour l'installation de la librairie sur son poste de travail. Et puis, nous présentons quelques fonctionnalités intéréssantes au premier abord.

2.2 Installation et quelques fonctionnalités de la librairie dicomlib

2.2.1 Installation de la librairie dicomlib

La librairie dicomlib [6] dépend de trois autres librairies comme montrer sur le figure 8 suivant :

- ***socket*** [7] : librairie pour le réseau faite par le même auteur mais qui n'est pas utilisé pour l'instant.
- ***boost*** [8] : boost est un ensemble de librairie en C++ founissant diverses fonctionalités (structures de données, gestion des dates, fonctions mathématiques,...)
- **et*scons*** [9] : outils de construction d'application, un equivalent à autoconf/automake pour générer des makefiles ; écrit en python.

Dans l'ordre, on installe : python, scons, boost, socket et dicomlib.

2.2.2 Quelques fonctionnalités de la librairie *dicomlib*

Avec un fichier Makefile comme en dessous, nous compilons et executons quelques exemples d'utilisations des fonctionnalités de *dicomlib*.

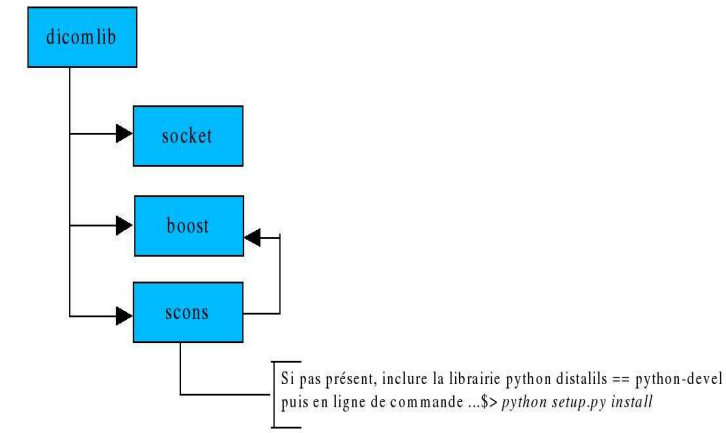


FIG. 2.1 – *Les dépendances de la librairie dicomlib.*

```
CPPFLAGS+=-I/home/stagiaireDESS/Documents/dicomlib
AGS+=-L/home/stagiaireDESS/Documents/dicomlib -L/usr/local/lib

LDFLAGS+= -ldicom -lboost_date_time-gcc -lboost_filesystem-gcc -lboost_regex-gcc -lboost_signals-gcc

CXX=g++
CC=g++

essai :essai.o
$(CC) $< -o $@ $CPPFLAGS $LDFLAGS

essai.o :essai.cpp
```

Makefile

Nous utiliserons principalement les classes de l'espace de nom (namespace) dicom et les quelques fonctions qui nous concerne.

Domaine **dicom** :
Les fonctions :
Obtenir le type de VR d'un tag : `VR GetVR(Tag tag)`
Obtenir la signification d'un tag : `std : :string GetName(Tag tag)`
Lire dans un flux d'entrée un fichier dicom : `void ReadFromStream(std : :ifstream &In, DataSet &data)`
Ecrire dans un flux de sortie un fichier dicom : `void WriteToStream(const DataSet &data, std : :ostream &Out, TS ts)`
Obtenir le groupe du tag : `UINT16 GroupeTag(Tag tag)`
Obtenir l'élément du groupe tag : `UINT16 ElementTag(Tag tag)`
Créer un nouveau tag : `Tag makeTag(UINT16 gggg, UINT16 dddd)`
...

Class **dicom : :DataSet**
Les fonctions membres :
Obtenir la valeur d'un tag : `const Value &operator()(const Tag tag) const`
Insérer une valeur à un tag : `template <VR vr, typename T>void Put(Tag tag, const T &data)`

Struct **dicom : :Value** Les fonctions membres : Constructeur : `template<typename T>Value(VR vr, const T&data)`
Fonction d'acces : `template<typename T>void Get(T &t) const`
autre fonction Get : `template<typenae T>const T &Get() const`
Opérateur équivalent à la fonction d'acces : `template<typename T>void operator»(T &t) const`
Les attributs publics :
`const VR vr_`

`struct template dicom : :TypeFromVR<vr>`, nous permet de faire la correspondance entre le type de VR et le type associé du langage C++.

A partir de ces quelques fonctions, nous présentons quelques lignes de code qui nous permettent entre autres

d'ouvrir, d'afficher et d'écrire un fichier dicom.

```
#include "dicomlib.hpp"
#include "Exceptions.hpp"
using namespace dicom;
using namespace std;

/* Programme principal incluant
* quelques lignes de programme qui
* expose l'utilisation de quelques fonctions
 * de la librairie dicomlib.
*/

int main(int argc,char** argv) {

if (argc<2) {
cout « "précisez un nom de fichier SVP" « endl;
exit(0);
}
/*
 * Permet d'afficher tous le contenu d'un fichier dicom
*/


// Charge dans un flux d'entrée In l'argument 1 de la ligne de commande en mode binaire, à savoir un fichier dicom
 ifstream In(argv[1],ios : :binary);
DataSet data;  // on déclare une variable de type DataSet
ReadFromStream(In,data);   // Lecture dans le flux d'entrée In un fichier dicom dans data
cout « data;   // Affichage du contenu de data
/*
 * Permet d'afficher le type de VR
*
*/

VR vrType = GetVR(dicom : :TAG_IMAGE_TYPE);
string nameTag = GetName(dicom : :TAG_IMAGE_TYPE);
cout«nameTag«" a pour VR : "«vrType«endl;

/*
 * Permet d'afficher la valeur d'un tag
 *
*/
 const Value &ImageType = data(dicom : :TAG_IMAGE_TYPE);
cout«"Type d'imagee est :"«ImageType«endl;
/*
 * Permet d'insérer et d'écrire dans un nouveau fichier .dcm
 * un nouvel élément avec sa valeur
* dans un dictionnaire de données data
* Exemple insérons un nom supplémentaire à data
 *
*/
cout«"Insérons un nom du patient"«endl;
// insertion d'un nouveau nom de patient dans data
data.Put<dicom : :VR_PN,string>(dicom : :TAG_PAT_NAME,newName); ofstream Out;
Out.open(argv[2],ios : :binary); // ouverture d'un flux de sortie sur fichier dans Out.
WriteToStream(data,Out); // écriture dans data de Out.
```

essai.cpp

Le résultat de l'execution de essai.cpp est affiché dans un terminal (voir annexe C).

2.3 Conclusion

Avec l'aide d'une documentation fournie sur le site [4], la librairie *dicomlib* s'installe assez facilement. En revanche sa compréhension requière tout de même une bonne connaissance en programmation C++ dont cette librairie utilise beaucoup les mécanismes de templates. De ces quelques lignes de code, il en sera inspiré à terme, pour développer des applications spécifiques à la manipulations de fichiers dicom et à la mise en réseaux avec les mécanisme de « serveur multithreading ». Un autre projet qui entre dans le cadre de la simulation de plans de traitement ; en occurrence la dosimétrie, sera développer à la suite en seconde partie. Le but serait de mieux contrôler les plans de traitement et ainsi se pourvoir d'une prestation de qualité meilleur au départhement radiothérapie du centre Léon Bérard. Pour cela nous souhaiterons utiliser les informations qu'un fichier DICOM peut contenir. Celui-ci devra nous permettre de modéliser de façon réaliste, et d'étudier la région d'intérêt d'un patient. Une librairie *Geant4* du CERN (Genève) peut nous le permettre.

Deuxième partie

GEANT4

Chapitre 3

INTRODUCTION A GEANT4

3.1 Introduction

Dans le cadre des études en radiothérapie, un des domaines qu'il est souhaité de contrôler est celui du plan de traitement ; c'est-à-dire le positionnement du patient et celui de la dose transmise que recoit un imageur portale (= détecteur de photons). En ce qui nous concerne, nous nous pencherons sur le deuxième aspect du plan de traitement ; en d'autre terme : la dosimétrie. Nous souhaitons simuler la dose reçu par un détecteur sous plusieurs angles incidents tout en considérant tous les processus physiques que sucite l'utilisation d'un faisceau de photons de haute énergie. Les phénomènes physiques se réalisant selon des lois probabilistes données, la simulation se fait donc avec des méthodes Monte Carlo (voir Physics Reference Manuel page 4).
Pour procéder à des simulations complexes appliquées à la radiothérapie, la librairie **Geant4** est écrite en C++ par des membres du CERN, Genève Suisse. Elle est constituée d'un ensemble de classes conçu pour la physique des hautes énergies et introduit toutes les connaissances de la physique des particules depuis le début du siècle jusqu'à aujourd'hui. Avec elle, nous pouvons modéliser notre simulation aussi finement que possible. Nous programmons notre espace de géométrie contenant des volumes (fantôme, détecteur, scintillateur, calorimétre, ...), nos processus physiques résultant d'interactions électromagnétiques et/ou d'interactions hadroniques, ainsi que notre source de particules primaires. Nous pouvons suivre, pister pas à pas nos particules primaires et aussi les secondaires. Il existe bien sûr des logiciels de simulation pour la radiothérapie tels que NC***, GATE incluant en partie le noyau *Geant4* et des objets prédéfinis. Mais il se trouve que leur configuration reste limiter par rapport à la librairie *Geant4*. Somme toute, il est intérèssant voir même souhaitable de faire des recoupement de résultats avec les logiciels existants afin de valider des modèles. A terme avec les résultats de simulations de dose transmise par *Geant4*, nous pourrions vérifier le positionnement du patient et remonter sur des valeurs de doses transmises que reçoit le détecteur. Et aisni, nous pouvons avoir un meilleur contrôl de qualité des plans de traitement.

3.2 Comment installer la librairie Geant4 ?

Nous pouvons effectuer des simulations avec la librairie *Geant4* écrite en C++ sur les platformes Unix et Windows. Il est présenté à la suite l'installation de la version *geant4.6.1*.

1. Dans le répertoire geant4, on installe la librairie **CLHEP** [10] :

   La librairie **CLHEP** (***C**lasse **L**ibrary for **H**igh **E**nergy **P**hysics*) écrite pour la physique des hautes énergies [11]. On y trouve une suite de classes qui y définit, des fonctions et outils mathématiques (statistique, aléatoire, distribution, trigonométrique, matrice, ...), des méthodes de calcul numérique, des constantes physiques et unités, des méthodes de constructions d'histogrammes et graphiques 2D/3D, des méthodes de manipulation de chaine de caractères, des méthodes de manipulation de fichiers, divers opérateurs, etc ...Pour plus de détails, il existe une documentation en ligne de la librairie CLHEP [12].

2. Dans le répertoire `geant4`, on installe la librairie *geant4.6.1* [13] ; une version téléchargée le 25/05/2004.

Ensuite, il faut éditer le fichier script `setup-geant4.sh` qui nous permet d'initialiser les variables environnement GEANT4. Une liste de ces variables est donnée en annexe A.A. Ou bien, on procède par la commande `./configure -install` qui nous aide à établir ce fichier script en répondant aux questions posées lors de la configuration. Nous avons en annexe A notre script. C'est dans le répertoire geant4 que doit se trouver ce `script setup-geant4.sh` .

3. Ensuite pour pouvoir compiler les sources de `geant4.6.1`, nous initialisons les variables d'environnement de *geant4* :

```
$>source setup-geant4.sh
G4INSTALL = /home/'son home'/geant4/geant4.6.1
G4SYSTEM = Linux-g++
CLHEP_BASE_DIR = /home/'son home'/geant4/CLHEP
```

Ensuite :

```
$>cd geant4.6.1/source/ $>make
```

Idem dans le répertoire `geant4.6.1/hadronics_lists/lists`.

3.3 Comment executer un exemple ?

Pour l'exemple, nous allons exécuter une application qui se trouve dans le répertoire `geant4.6.1/examples/extended/analysis/A01`.

1. Chaque fois que l'on veut faire de la simulation lors d'une nouvelle session dans un terminal, nous devons nous mettre dans le répertoire geant4 et initialiser une seule fois les variables d'environnement de `geant4` comme suit :

```
$>source setup-geant4.sh
```

Ensuite, nous nous mettons dans le repertoire A01 là où se trouve l'application A01app.cc.
Puis, on compile :

```
$>make
Making dependency for file A01app.cc...
Making dependency for file src/A01VisManager.cc ...
Making dependency for file src/A01Trajectory.cc ...
...
...
Compiling A01AnalysisManager.cc ...
Compiling A01CellParameterisation.cc ...
Compiling A01DetectorConstMessenger.cc ...
...
...
Creating/replacing object files in /afs/slac.stanford.edu/u/ey/perl/wiredwork/g4
cdtest/Geant4/geant4.6.0/tmp/Linux-g++/A01app/libA01app.a Compiling A01app.cc ...
Using granular libraries ...
Linking A01app ...
```

Remaquons que si la compilation ne fonctionne pas, on a certainement oublié d'éxecuter l'étape "`source setup-geant4.sh`".

2. Après, pour pouvoir executer l'application, on exporte une seule fois la variable d'environnement se trouvant dans le répertoire `geant4.6.1/bin/Linux-g++` :

```
$>export G4EXE=$G4INSTALL/bin/$G4SYSTEM
$>$G4EXE/A01app
```

A la suite de cette commande d'execution, un nouvel environnement apparaît dans le terminal avec comme prompt `Idle>`. Cette environnement pourrait éventuellement être celui d'un l'utilisateur physicien souhaitant uniquement entrer des paramètres à sa simulation et exploiter les résutats par la suite. Pour cela il n'a que besoin de connaître les quelques commandes utiles détaillées en [14].

`vis/open HepRepFile` -> *avec systeme de graphique HepRepFile, création d'une scène prête être dessinée.*
`Idle> /vis/scene/create` -> *création d'une scène vide.*
`Idle> /vis/scene/endOfEventAction accumulate` -> *dessine en regroupant tous les événements (hits, track, etc ...) sur la scène vide.*
`Idle> /vis/scene/add/volume` -> *additonne tous les volumes physiques sur la scène vide.*
`Idle> /vis/sceneHandler/attach` -> *créer les fichiers G4Data0.heprep et G4Data1.heprep.*
`Idle> /vis/viewer/flush` -> *permet de rafraichir l'affichage.*
`Idle> /vis/scene/add/trajectories` -> *additonne toutes les trajectoires sur la scène courante.*
`Idle> /vis/scene/add/hits` -> *additonne toutes les hits sur la scène courante.*
`Idle> /tracking/storeTrajectory 1` -> *permet de stocker l'historique des trajectoires pour le visualiseur.*
`Idle> /run/beamOn 2` -> *permet de lancer 2 simulations.*

Cette suite de commandes, nous permet de créer deux fichiers d'extension *.heprep*. Ces derniers nous permettent par la suite de visualiser la scéne dans le visualiseur WIRED dont son installation sera décrit à la suite. Bien sûr, il existe d'autres outils servant à la visualisation comme DAWN [15] et OpenGL [16] et encore d'autres outils servant à l'analyse graphique comme JAIDA et JAS3 [17]. Les précédentes commandes sont décrites à l'annexe A.

3. Installation du logiciel de visualisation WIRED.

Son installation se fait rapidement, il suffit d'avoir une version de java supérieure à 1.3 en le verifiant par :

```
$>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM (build 1.4.2-b28, mixed mode)
```

Dans son `home directory`, créer un répertoire `WIRED` dans lequel on téléchargera l'utilitaire de visualisation *WIRED* [18] [7] :

```
$>mkdir WIRED
$>cd WIRED
$>java -cp . install
```

Ensuite, il nous suffit de lancer l'exécutable `wired` du répertoire `wired/bin` avec ou sans un fichier d'extension *.heprep* :

```
$>cd WIRED/wired/bin
$>./wired [-file <chemin d'accès au fichier .heprep>]
```

3.4 Comment se créer son application ?

Nous nous inspirons des nombreux exemples fournis avec la librairie *geant4* pour développer une simulation simple (voir répertoire `geant4.6.1/examples/`). Nous trouverons sur le lien suivant la description de toutes les classes que *geant4* intègre dans sa librairie geant4.6.1 [19].

3.4.1 Généralité

Pour la première fois avec *Geant4*, nous souhaitons réaliser une simulation simple d'un faisceau de photons incidents traversant un fantôme rempli d'eau au delà duquel un capteur devra enregistrer des données.

Ensuite, nous pourrions entrer dans les détails et subtilités de *Geant4* pour simuler un modèle complexe qui tiendrait compte des conditions fines : c'est-à-dire une source d'émission de particules distribués spectralement sous différents angles d'incidences, des choix optimisés de processus physiques et de valeurs d'énergie de coupure, ainsi que des volumes réalistes ; un fantôme composé de coupe d'image au format DICOM dont les valeurs d'intensité des pixels corresponderont à des configurations matériels précises (eau, graisse, os, ...) et un capteur représentant en détail un imageur portale enregistrant des doses. Cette complexité ne peut être relever dans le cadre de ce stage en temps imparti surtout quand on ne connait pas en détail les classes de la librairie *Geant4*.
Alors pour ce faire simplement, la réalisation d'une application se construit principalement en trois classes obligatoires et cinq autres classes optionnelles. Nous avons besoin de connaître un certain nombre de vocabulaire utiliser sous *Geant4*.

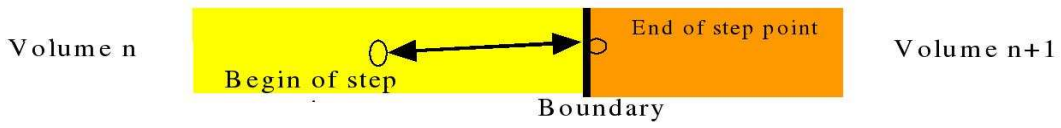
Lexiques :

- ***Run*** :
  L'execution (**run**) de *Geant4* commence avec la méthode `BeamOn()`. Durant l'execution, on ne peut changer la géométrie des détecteurs et aucun détecteurs n'est accessible. Une execution est une collection d'événements.

- ***Event*** :
  Au commencement d'un processus en cours, un événement (**event**) contient les particules primaires. Ceux ci sont mis dans une pile. Quand la pile devient vide, le processus en cours d'un événement est terminé. La classe `G4Event` réprésente un événement. Il stocke une liste de points et particules primaires, des collections de « *hits* »et des collections de « *digits* ».

- ***Track*** :
  Un ***track*** est par analogie une photo, une prise instantané d'une particule. Un ***step*** est une information « *delta* »pour un *track* (ce n'est pas une collection de *steps*). Un *track* est effacé quand il sort du volume du monde ou disparaît (par délais) ou quand son énergie cinétique tombe à zéro ou encore quand l'utilisateur décide de le tuer. Un *track* est construit de **3** classes : `G4Track` (position, volume, longueur du track, ID et « Trackmère »), `G4DynamicParticle` (vitesse, énergie, temps local, polarisation, ...), `G4ParticleDefinition` (masse, temps de vie, charge, ...).

- ***Step*** :
  Un ***step*** a deux points et aussi une information « *delta* »d'une particule (baisse d'énergie sur le *step*, temps de vol dépensé par le *step*, etc ...). Chaque point connait le volume. Un *step* est limité par le bord du volume et le point final reste sur le bord. Et il appartient logiquement au volume suivant.



- ***Trajectory*** :
  Un ***trajectory*** est un enregistrement de l'historique d'un *track*. Il enregistre des informations de tous les *steps* obtenus par le *track* (définit par `G4TrajectoryPoint`). Il est conseillé de ne pas enregistrer les particules secondaires à cause de la consommation en mémoire. L'utilisateur peut créer ces propres classe *trajectory* dérivant de `G4VTrajectory` et `G4VTrajectoryPoint`.

- ***Hit*** :
  Un ***hit*** est comme un *track*, une prise instantanée d'une interaction physique d'un *track* ou une accumulation d'interactions de *tracks* dans la région sensible de notre détecteur. Un détecteur crée des *hits*

en utilisant les informations données par le *step*. Les objets *hits* sont collectés dans un objet `G4Event` à la fin d'un événement.

- ***Digit*** :
  Un ***digit*** représente une sortie du détecteur (ADC/TDC count, trigger signal, ...). Le *digit* est créé en utilisant l'information d'une ou plusieurs *hits* et/ou d'autres *digits* par l'implémentation d'une classe dérivant de `G4VDigitizerModule`. En opposition, le *hit* qui est généré automatiquement par *tracking*, la méthode `digitize()` de chaque `G4VDigitizerModule` doit être explicitement invoqué par du code de l'utilisateur.

Dans notre *exemple1*, nous utiliserons, en plus des classes obligatoires, la classe `G4UserEventAction` pour afficher dans le terminal des informations sur un événement, et à terme construire des histogrammes et autres graphiques servant à l'analyse de résultats.

Classes Obligatoires	***G4VUserDetectorConstruction*** : Classe abstraite pour initialiser l'espace de simulation. Elle introduit une seule méthode virtuelle `Construct()` qui est invoquée par la méthode `Initialize()` de la classe `G4RunManager` inclue dans le fichier principal. La méthode `Construct()` doit obligatoirement retourner un pointeur sur `G4VPhysicalVolume` qui représente le volume de l'espace de simulation. (voir répertoire `geant4.6.1/source/run/src/`) ***G4VUserPhysicsList*** : Classe abstraite pour construire les particules et les processus physiques. L'utilisateur doit implémenter trois méthodes virtuelles dans sa propre classe concrète dérivée de cette classe : `G4VUserPhysicsList : :ConstructParticle()` (construction des particules), `G4VUserPhysicsList : :ConstructPhysics()` (construction des processus physiques), `G4VUserPhysicsList : :SetCuts()` (ensemble de valeurs d'énergie de coupure dans une gamme pour toutes les particules). (voir répertoire `geant4.6.1/source/run/src/`) ***G4VUserPrimaryGeneratorAction*** : Classe abstraite pour la génération de particule. Elle introduit une méthode virtuelle `GeneratePrimaries()` qui est invoquée par la classe `G4RunManager` durant l'execution. (voir répertoire `geant4.6.1/source/run/src/`)
Classes Optionnelles	***G4UserRunAction*** : `BeginOfRunAction(const G4Run*)` on y définit les données à représenter dans des graphiques lors d'une exécution et `EndOfRunAction(const G4Run*)` on y stocke les données mesurées dans les graphiques définis. (voir répertoire `geant4.6.1/source/run/src/`) ***G4UserEventAction*** : `BeginOfEventAction(const G4Event*)` selectionne un événement et les données à représenter dans des histogrammes, et `EndOfEventAction(const G4Event*)` analyse l'événement avec des graphiques. (voir répertoire `geant4.6.1/source/event/`) ***G4UserStackingAction*** : `Event()` , `ClassifyNewTrack(const G4Track*)` et `NewStage()` (voir répertoire `geant4.6.1/source/event/`) ***G4UserTrackingAction*** : `PreUserTrackingAction(const G4Track*)` décide si la trajectoire des particules devrait être enregistrée ou pas, créer la trajectoire définie par l'utilisateur `PostUserTrackingAction(const G4Track*)`. (voir répertoire `geant4.6.1/source/tracking/`) ***SteppingAction*** : `UserSteppingAction(const G4Step*)` tue, suspend ou reporte le track. (voir répertoire `geant4.6.1/source/tracking/`)

3.4.2 Le programme principal exemple1.cc

Pour pouvoir lancer notre simulation, nous pouvons structurer notre programme principal selon trois modes :

- « *Hard-coded Batch Mode* » : en codant directement les commandes dans le programme principal.
- « *Batch Mode with Macro File* » : en faisant appel, dans le programme principal, un fichie macro où les commandes pour la simulation y sont écrites.
- « *Interactive Mode Driven by Command Lines* » : en ouvrant une interface de commandes interactives dans le terminal, destiné à un utilisateur.

Nous choisirons de combiner les deux derniers modes pour nous donner le choix de lancer la simulation à partir d'un fichier macro « *xxxx.mac* »par la commande `exemple1 xxxx.mac` ou à partir de l'interface utilisateur par la commande `exemple1`. L'execution de la simulation consiste à une séquence d'événements. Dans une execution, l'espace de simulation et les processus physiques devront rester inchanger. Une execution est représentée par un objet de la classe `G4Run`. Elle débute avec la méthode `beamOn()` de la classe `G4RunManager`(voir *exemple1.cc*).

Nous y incluons les fichiers suivants dans *exemple1* qui nous servirons à lancer notre simulation :

- ***Ex1DetetorConstruction.hh*** contenant la géométrie de notre espace de simulation.
- ***Ex1PhysicsList.hh*** contenant les processus physiques succeptibles de se produire.
- ***Ex1PrimaryGeneratorAction.hh*** décrivant notre source d'émission de particules.
- ***Ex1EventAction.hh*** permettant d'afficher des informations sur les événements qui se produisent durant la simulation.
- ***Ex1VisManager.hh*** permettant de générer des fichiers xml afin de pouvoir visualiser notre simulation avec un logiciel de visualisation.
- ***G4RunManager.hh*** permettant de gérer le déroulement de la simulation.
- ***G4UImanager.hh*** permettant d'utiliser des commandes dans un environnement pour des utilisateurs non informaticiens.
- ***G4UIterminal.hh*** permettant de créer cette environnement de commandes interactives dans un terminal.

Nous résumons ci-dessous dans la figure 3.1 la composition de l'*exemple1* en fichiers dont une partie provient de la librairie *Geant4* ; tous ceux qui commencent par *G4****. Et les autres proviennent de l'écriture de l'utilisateur :

3.4.3 Construction de l'espace géométrique d'une simulation

Construction des volumes

Pour commencer donc, notre modèle géométrique sera composé de trois grossiers objets que l'on appelera des « volumes ». Un premier volume essentiel devra être un espace clos pour limiter l'espace de simulation nommé le « monde ». Il devra contenir tous les autres volumes. Il est la « mère »de tous. Les autres volumes « filles »seront donc placés par rapport à leur « mère ». Tous les volumes devront être composés de matières.

Sur la figure 3.2, nous nommons notre volume mère : experimentalHall. C'est un cube rempli d'air composé à 30 % d'oxygène et 70 % d'azote, et d'une densité de $1.204 * mg/cm^3$. Ces dimensions sont $2\ m * 2\ m * 2\ m$. Notre premier volume fille se nomme : fantôme. C'est un cube rempli d'eau de densité $1.000 * g/cm^3$. Ces dimensions sont $20\ cm * 20\ cm * 20\ cm$. Il est positionné par rapport au centre du volume mère selon le vecteur position suivant $(0, 0, 10\ cm)$. Notre second volume fille se nomme : captor. C'est une plaque en aluminium de densité $2.700 * g/\ cm^3$. Ces dimensions sont $20\ cm * 20\ cm * 0.639\ cm$. Il est positionné toujours par rapport au centre du volume mère selon le vecteur position suivant $(0, 0, -62.50\ cm)$. Tous les matériaux utilisés seront à pression et température standard ; c'est-à-dire $1\ atm$ et $278\ °K$.

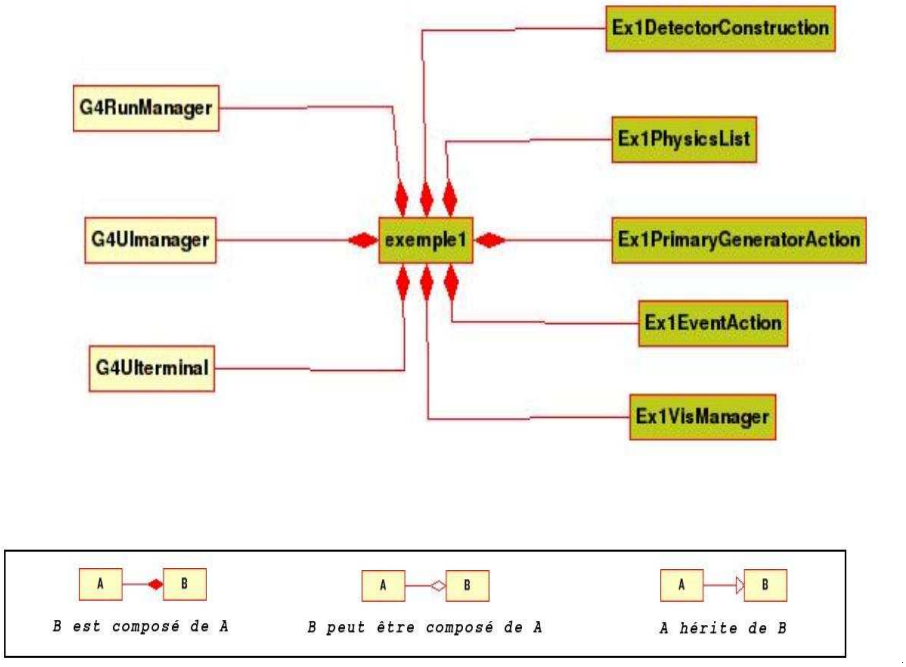


FIG. 3.1 – *Diagramme de classe de exemple1.*

A terme pour hiérarchiser la construction, quand la géométrie s'affinera certains volumes « filles »devront être des volumes « mères »pour des volumes qui la constitureront.

La traduction de ce modèle simple s'écrit par une classe qui hérite de la méthode de construction de la classe `G4VUserDetectorConstruction`. Nous la nommerons `Ex1DetectorConstruction`. Nous y construisons donc

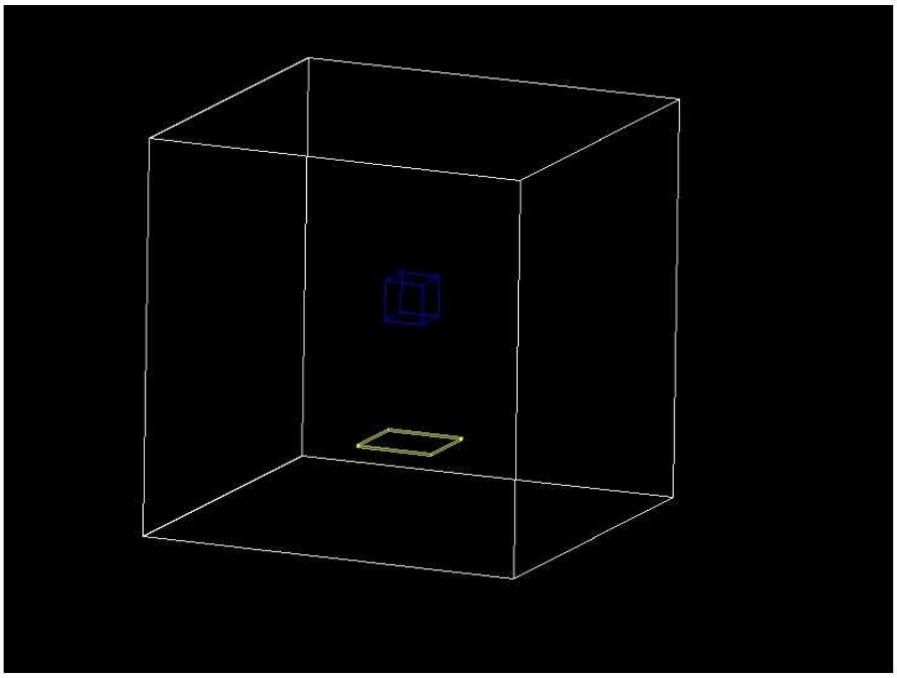


FIG. 3.2 – *Représentation de l'espace de simulation. En blanc le volume mère : experimentalHall, en bleu et jaune les volumes filles : respectivement fantôme et captor.*

tous nos objets de notre simulation. Nous définissons les formes géométriques avec leurs compositions matériels et leurs positions dans cette espace. C'est effectivement avec la méthode `Construct()` que l'on construit tous nos objets géométriques. Cette méthode doit retourner un pointeur sur `G4VPhysicalVolume` qui doit être le volume qui contient tous les autres volumes ; c'est-à-dire le « volume mère ».

La construction d'un volume se fait en trois étapes comme par exemple la création *captor* :

- Tout d'abord, nous construisons l'objet avec l'aide des classes servant à faire des boites, des tubes, des sphères, ...
- Ensuite, nous assignons à cette forme un matériau avec l'aide de la classe `G4LogicalVolume` .
- Nous finissons la création d'un objet en le positionnant par rapport à « l'objet mère »associé avec l'aide de la classe `G4PVPlacement`.

```
G4Box* captor_box = new G4Box("cap_box", 20.*cm, 20.*cm, 0.639*cm) ;
G4LogicalVolume* captor_log= new G4LogicalVolume(cap_box,Al,"cap_log",0,0,0) ;
G4PhysicalVolume* captor_phys = new G4PVPlacement(
0,
G4ThreeVector(0.*cm, 0.*cm, -62.50*cm),
captor_log,"captor",
experimentalHall_log,false,0) ;
```

Pour résumer cette étape de construction d'un espace de simulation, nous avons besoin pour le fichier `Ex1DetectorConstruction` des classes suivantes que l'on représente ci dessous à la figure 3.3 :

- ***Ex1CaptorSD.hh*** définissant les paramètres à enregistrer par notre détecteur captor.
- ***G4ThreeVector.hh*** permettant d'utiliser des vecteurs.
- ***globals.hh*** définissant toutes des unités, des constantes, des types G4, ...

- ***G4Element.hh*** permettant de définir des éléments chimiques.
- ***G4Material.hh*** permettant de définir des matériaux complexes.
- ***G4Box.hh*** permettant de créer des boites.
- ***G4LogicalVolume.hh*** permettant d'assigner à un volume solide sa constitution matérielle.
- ***G4VPhysicalVolume.hh*** permettant de créer un volume physique à partir d'un volume logique.
- ***G4PVPlacement.hh*** héritant de la précédente classe, permet de placer le volume physique dans l'espace par rapport à un autre volume physique.
- ***G4SDManager.hh*** permettant de gérer des détecteurs.

- ***G4VisAttributes.hh*** permettant de donner des attributs.

Après cette étape de construction de volumes, nous avons une étape délicate à réaliser. Dans un but de quantification de variables, nous devons comprendre comment assigner la fonction spécifique de mesure. D'autres fonctions peuvent-être assigner à des volumes telles que dévier un faisceau de particules chargées par des champs magnétiques, ou générer des photons pour un scintillateur, etc ...

Designation du détecteur

Nous cherchons à mesurer le nombre d'événements, son temps de production, sa position et son énergie se produisant dans un volume. Pour cela nous devons spécifier la fonction de détection à un volume.
Nous procédons comme suit pour assigner la fonction de « détecteur »au volume *captor* dans cette même classe `Ex1DetectorConstruction`.

```
G4SDManager* SDman = G4SDManager : :GetSDMpointer() ;
G4String captorSDname="/captor";
Ex1captorSD* captorSD = new Ex1captorSD(captorSDname) ;
SDman->AddNewDetector(captorSD) ;
captor_log->SetSensitiveDetector(captorSD) ;
```

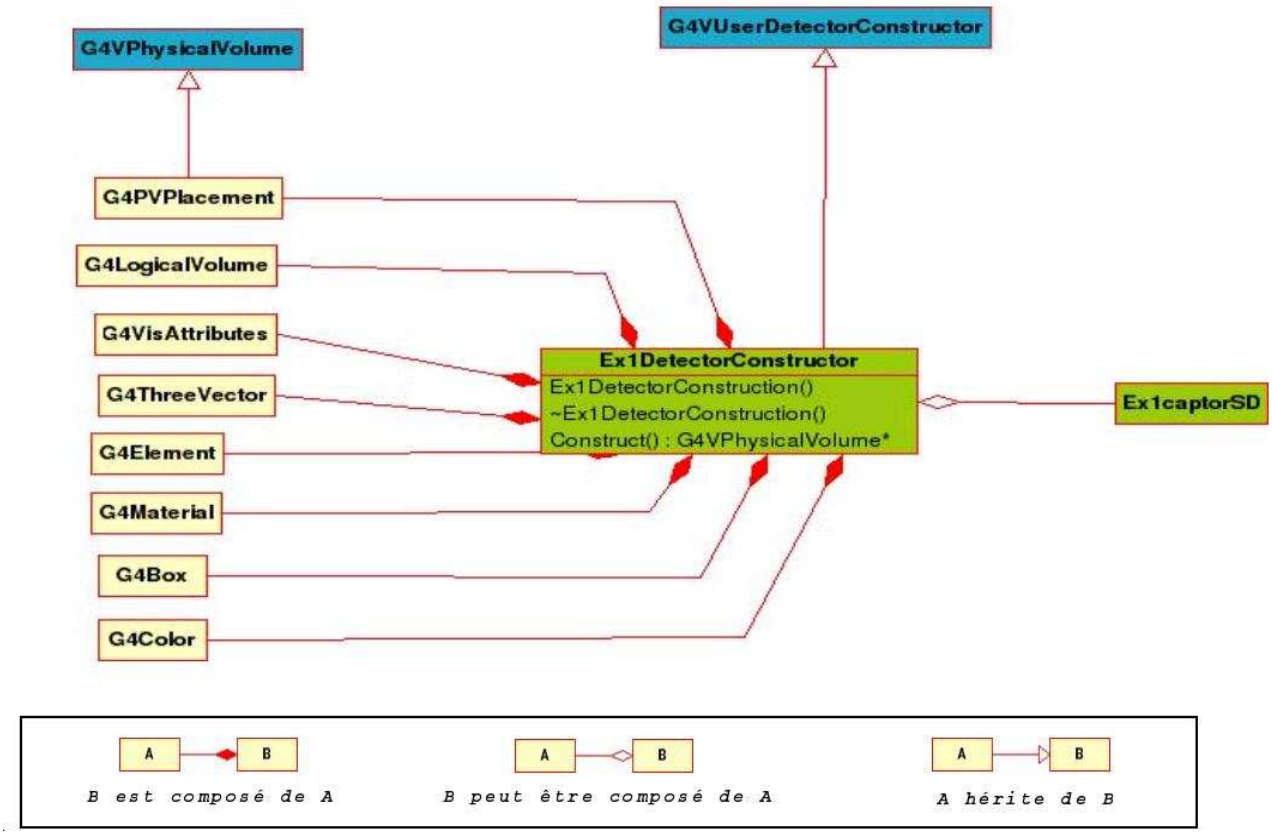


FIG. 3.3 – *Diagramme de classe de Ex1DetectorConstruction.*

Il nous faut pour ce faire utiliser la classe `G4SDmanager` et l'écriture de deux autres fichiers supplémentaires `Ex1captorHit` et `Ex1captorSD`.
Dans le premier fichier, sa construction hérite de la classe `G4VHit`. On y définit les variables à enregistrer par `Ex1captorSD`. Nous souhaitons donc enregistrer l'énergie déposée « *edep* » sur le détecteur *captor*, la position locale « *localPos* »et dans l'espace de simulation « *worldPos* », et enfin le temps de production « *time* »de chaque événement. Nous souhaitons aussi dessiner par un cercle coloré l'endroit où la production d'un événement à eu lieu. Enfin, nous souhaitons aussi afficher dans le terminal les résultats de la mesure.
Dans le second fichier, sa construction hérite de la classe `G4VSensitiveDetector`. On initialise notre détecteur, puis enregistrons la mesure des variables que l'on souhaite avec l'aide des classes `G4Step` et `G4HCofThisEvent`.

Pour résumer cette étape de fonction de détection, nous avons besoin pour le fichier `Ex1captor` des classes suivantes que l'on représente ci dessous sur la figure 3.4 :

- ***G4VSensitiveDetector*** définissant un détecteur.
- ***Ex1captorHit*** définissant les variables d'un événement à stocker dans un vecteur.
- ***G4Step*** définissant les fonctions de mesure communes lors d'une simulation en physique des particles.
- ***G4HCofThisEvent*** permettant de retourner un pointeur sur une collection d'événements
- ***G4TouchableHistory*** définissant pour un détecteur l'historique géométrique des événements.

A ce terme, nous souhaitons affichons dans le terminal les résultats des mesures des variables sur les événements produits.

Affichage des résultats dans un terminal

Nous affichons des résultats sur des événements avec une classe héritant des méthodes de la classe `G4UserEventAction`. On la nomme `Ex1EventAction`. C'est dans la méthode `EndOfEventAction()` que nous

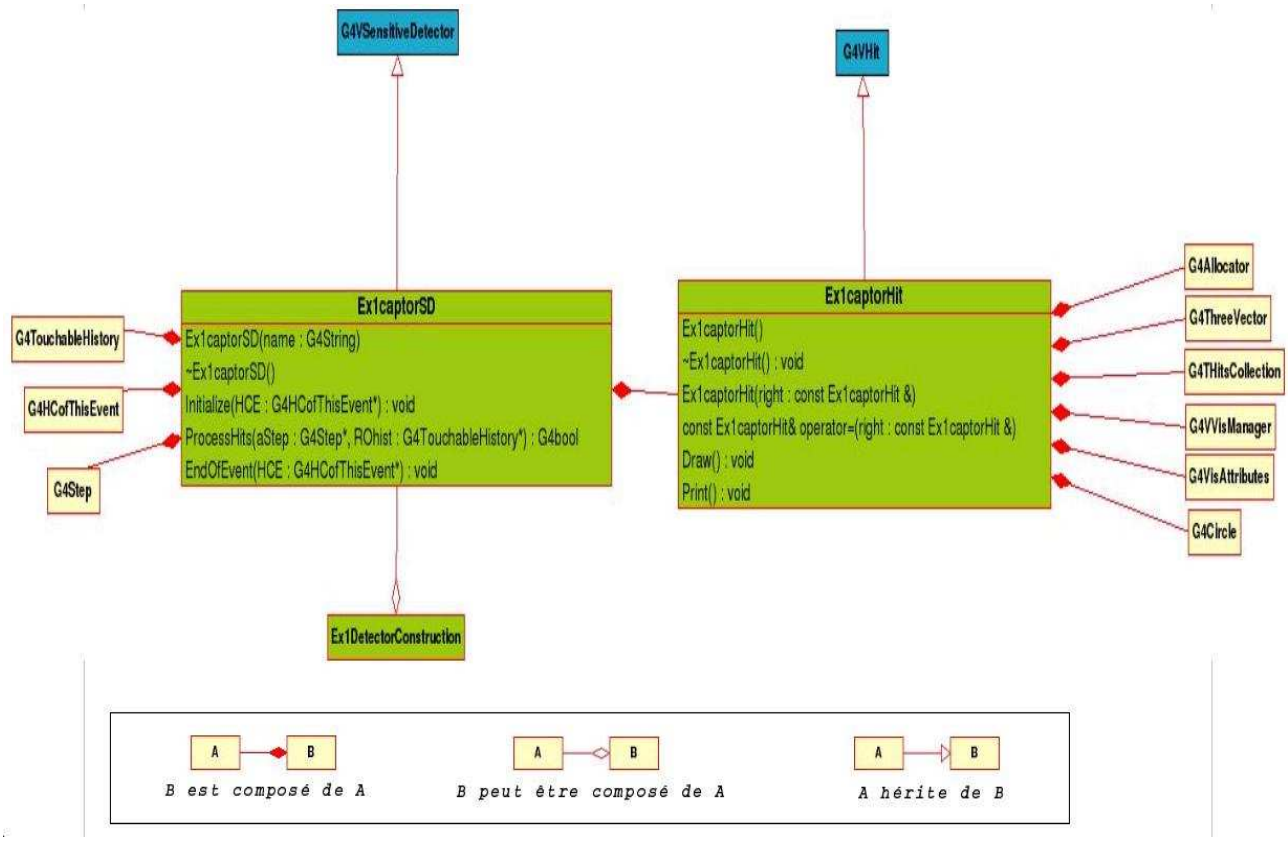


FIG. 3.4 – *Diagramme de classe de Ex1captorSD.*

manipulons les collections d'événements et y procédons à des calculs. Nous afficherons par la même occasion des résultats dans le terminal. A terme, nous construirons des histogrammes et autres graphiques pouvant servir à l'analyse et à la présentation de résultats.

Nous avons besoin pour le fichier `Ex1EventAction` des classes suivantes que l'on représente ci dessous sur la figure 3.5 :

- ***G4UserEventAction*** définissant les méthodes BeginOfEventAction() et EndOfAction().
- ***Ex1captorHit***.
- ***G4Event*** définissant un événement. Contenant toutes les entrées et sorties d'un événement simulé.
- ***G4EventManager*** permettant de gérer les événements.
- ***G4HCofThisEvent***.
- ***G4VHitsCollection***.

En conclusion dans cette partie, nous avons construit nos volumes et assignés à l'un, le volume *captor*, la fonction de détection. Puis nous avons géré l'affichage des résultats dans un terminal par exemple. Ensuite, nous devons définir les interactions physiques succeptibles de se produire lors du passage du faisceau de particules dans les volumes car le résultats de la mesure en dépend.

3.4.4 Prise en compte des processus physiques

Cette étape de modélisation relève des compétences d'un physicien spécialisé en physique des particules car la justesse des résultats de la simulation dépendra en grande partie par la prise en compte des processus physiques pouvant se produire dans de telle condition. Dans le cadre de notre étude, nous nous contenterons, pour notre compréhension globale de Geant4, de simuler des photons incidents à 6 MeV (valeur moyenne utilisé en radiothérapie). Les photons de cette énergie sont assimilés à des particules *gammas* γ sous *Geant4*.

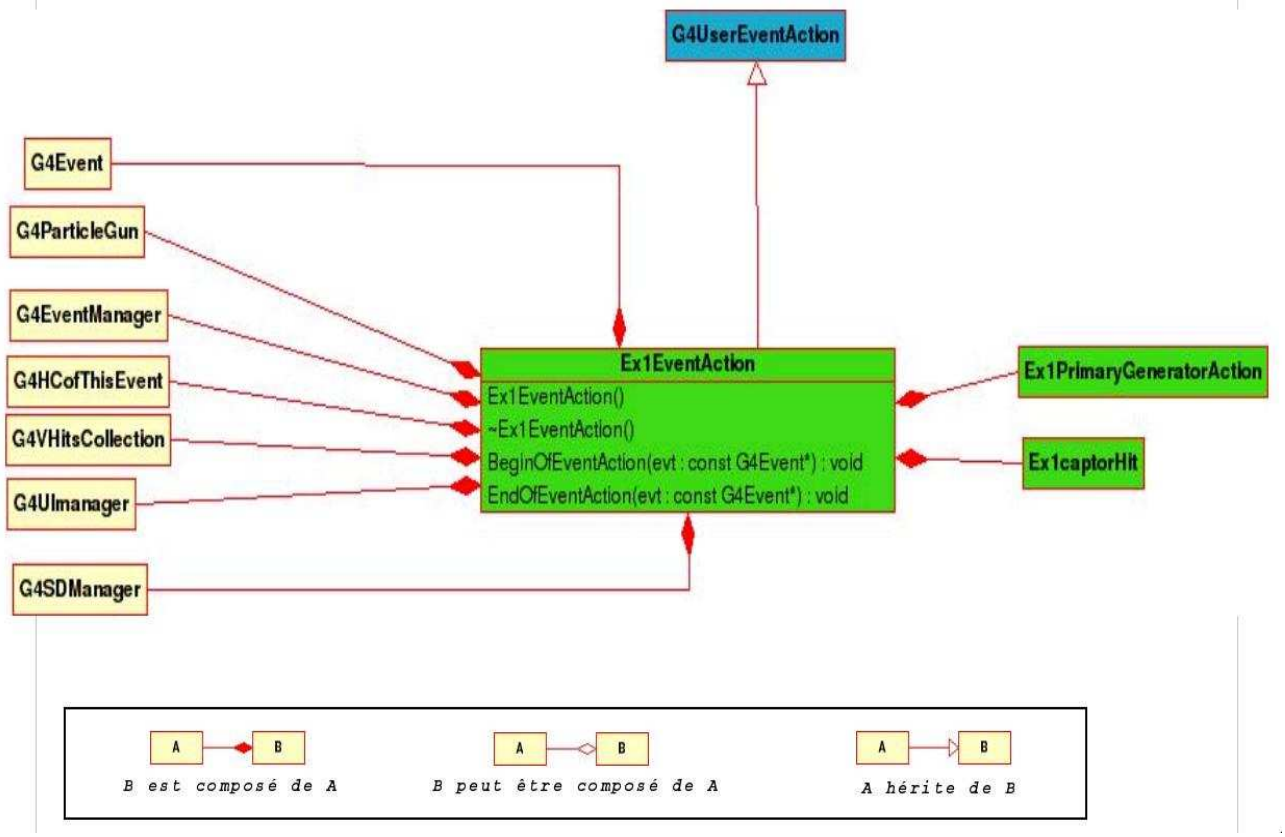


FIG. 3.5 – *Diagramme de classe de Ex1EventAction.*

En première considération, les intéractions sont d'ordre électromagnétique avec la probable génération de phénomènes secondaires.

Nous tiendrons compte des processus physiques suivants :

- ***Effets photo-électrique*** qui se manifeste par l'éjection d'un électron provenant de la matière après qu'un photon est été absorbé par la matière (voir Physics Reference Manuel page 27).
- ***Diffusion Compton*** qui se manifeste par émission d'un photon provenant d'un électron attaché à son noyau atomique.(voir Physics Reference Manuel page 30).
- ***Production de pair*** (e^+e^-) par unde haute énergie (voir Physics Reference Manuel page 34).
- ***Multi diffusions*** des particles chargés concerne les particules e^+ et e^- produites par des photons (voir Physics Reference Manuel page 65).
- ***Ionisation*** du milieu matériel produite par e^+ et e^- (voir Physics Reference Manuel page 92).
- ***Bremsstrahlung*** qui se manifeste par l'émission de photons provenant d'électrons ou de positrons en décélération (voir Physics Reference Manuel page 97).
- ***Annihilation*** (e^+e^-) produit deux (511 KeV chacun) par l'annihilation e^+ et e^- du milieu matériel (voir Physics Reference Manuel page 109).

Ces hypothèses sont écrites dans une classe héritant des méthodes de la classe `G4VUserPhysicsList`. Ici nous procédons par module et nous remplaçons cette dernière classe par `G4VModularPhysicsList`. Nous la nommons donc `Ex1PhysicsList`. La classe modulaire hérite à son tour des méthodes de la classe `G4VPhysicsConstructor` et se nomme `Ex1EMPhysics`.

Pour résumer cette étape des processus physiques, nous avons besoin pour le fichier `Ex1DetectorConstruction` et `Ex1EMPhysics` des classes suivantes que l'on représente ci dessous sur la figure 3.6 :

- ***G4VModularPhysicsList*** est une sous classe de G4VUserPhysicsList permettant de construire les

particules et les processus aisni que les valeurs de coupure des particules.
- ***G4ParticleWithCuts*** définissant l'énergie de coupure de chaque particule décrite dans G4ParticleDefinition ; c'est-à-dire l'énergie en dessous de laquelle la simulation ne considère plus la particule.
- ***G4ProcessManager*** permettant de collecter sous forme de vecteur et de gérer tous les processus de chaque particule.
- ***Ex1EMPhysics*** définissant la construction des particules et des processus physiques par héritage de G4VPhysicsConstructor.
- ***G4PhotoElectricEffect***, ***G4ComptonScattering***, ***G4GammaConversion***, ***G4eIonisation***, ***G4MultipleSca*** ***G4eBremsstrahlung***, ***G4eplusAnnihilationsont*** les classes définissant les processus physiques énumérés précédemment.

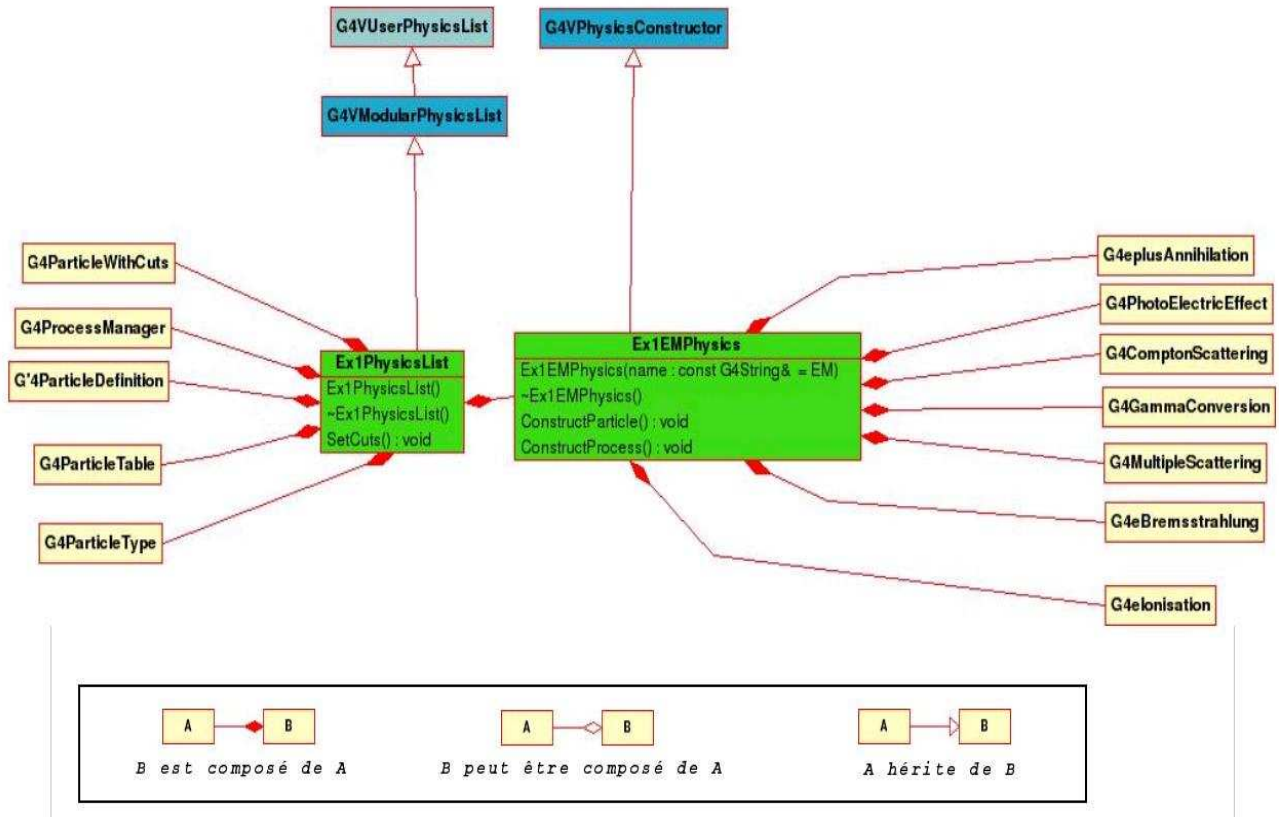


FIG. 3.6 – *Diagramme de classe de Ex1PhysicsList.*

Enfin, il nous reste plus qu'à définir notre source de particules.

3.4.5 Générateur de source de particule primaire

L'émission de particules primaires incidentes se fait par une classe héritant des méthodes de la classe `G4VUserPrimaryGeneratorAction`. On la nomme `Ex1PrimaryGeneratorAction`. Les particules sont construites avec la méthode `GeneratePrimaries()`.

Nous avons besoin pour le fichier `Ex1PrimaryGeneratorAction` des classes suivantes que l'on représente ci dessous sur la figure 3.7 :

- ***G4VUserPrimaryGeneratorAction*** définissant les propriétés de la source de particules.
- ***G4Event*** définissant un événement.
- ***G4ParticleDefinition*** définissant toutes les données des particules.
- ***G4ParticleTable*** définissant une table de pointeur vers les particules.

– ***G4ParticleGun*** définissant des méthodes pour paramétriser sa source (position, orientation,...).

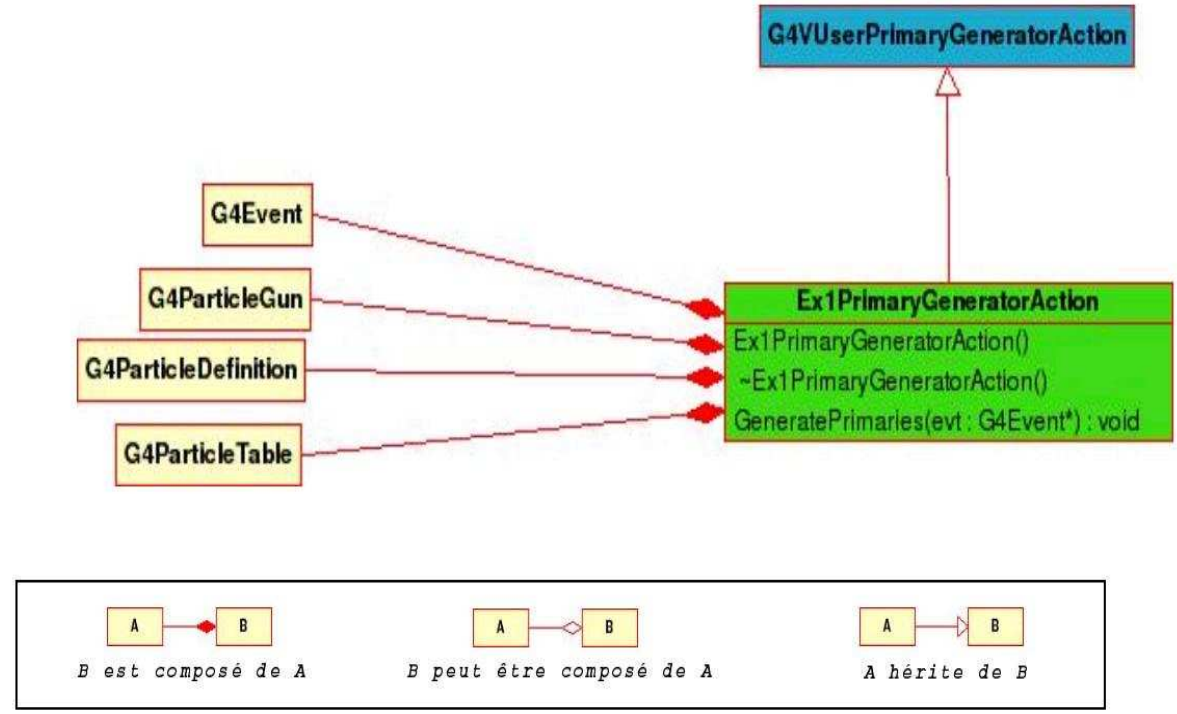


FIG. 3.7 – *Diagramme de classe de Ex1PrimaryGeneratorAction.*

A ce terme, nous pouvons compiler notre programme et lancer quelques simulations.

3.4.6 Quelques résultats

Nous avons configuré notre détecteur captor pour la mesure du nombre, du temps de production, de l'énergie déposée et de la position de chaque *hit* dans le *captor* et par événement. La réalisation d'un *hit* est soumise à des processus de calculs aléatoires. Les résultats par événement sont obtenus par simulation *Monte Carlo*.

Nous présentons deux événements avec une particule *gamma* γ en incidence verticale d'une énergie égale à 6 MeV. Puis un autre événement avec cette fois deux particules γ. Nous nous rendons compte que les résultats de deux événements d'une particule γ est égale aux résultats d'un événement de deux particules γ.

Pour 1 événement avec 1 particule γ :

```
>>> Event 0 >>> Simulation truth :  1 gamma(s)  (0,0,-6)

    0 hit (s) sont enregistrés dans captor.


Energie totale déposée dans captor : 0 (MeV)
```

Aucun *hit* n'a été enregistré par le *captor*. Comme nous le voyons sur la figure 3.8, effectivement le faisceau de γ n'atteind pas le *captor*. Sur le gros plan de la figure 3.9, le faisceau incident produit une pair (e^-, e^+). Le positron e^+ s'annihile par la suite en générant 2 γ qui partent dans des directions opposées. Nous

le vérifions par le calcul de l'énergie des deux γ devant être de chacun 511 KeV. Puis pour l'un des γ, de multiples diffusions Compton se produisent dans le fantôme avant de sortir et de finir sa course sur le bord de l'*experimentalHall*. Et pour l'autre, il va se perdre sans rencontrer le captor sur sa trajectoire et finir sur le bord de l'*experimentalHall*.

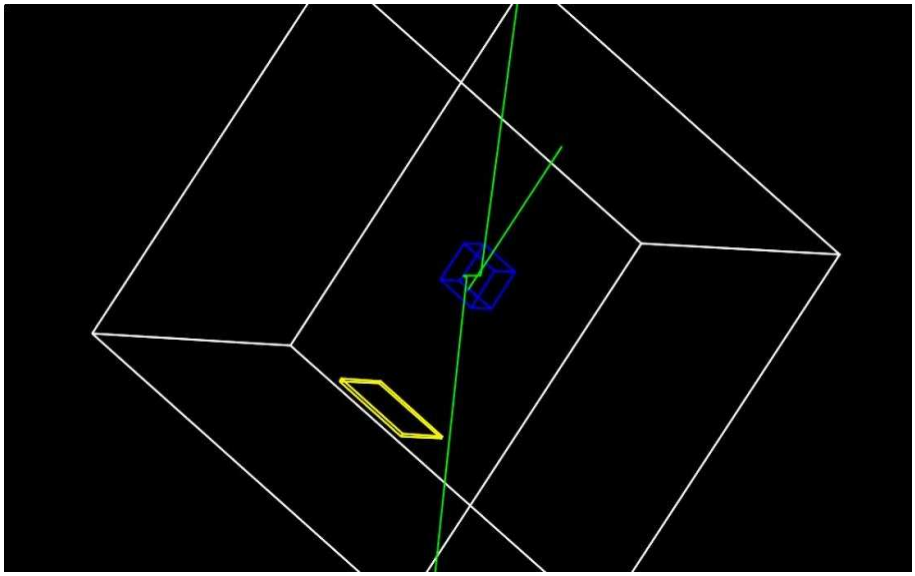


FIG. 3.8 – *Simulation d'un événement avec une particule γ.*

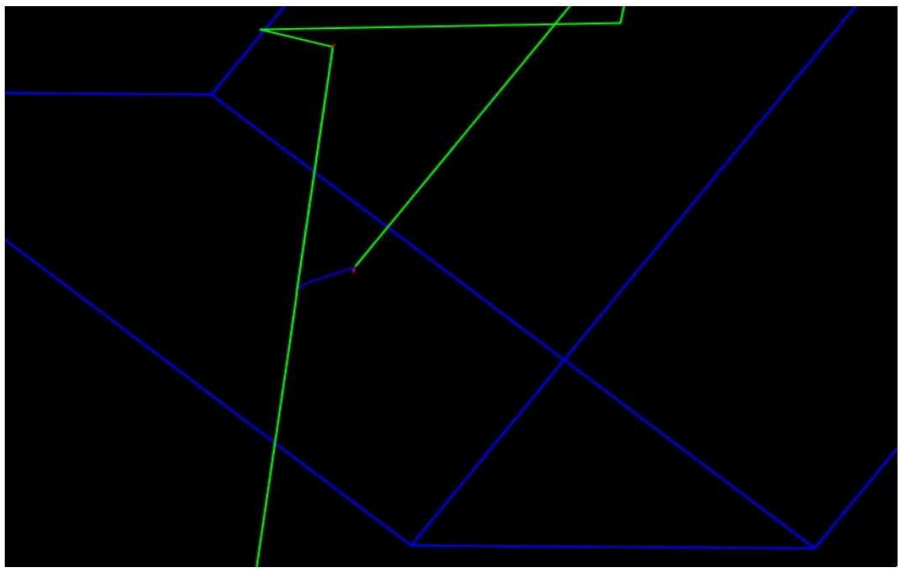


FIG. 3.9 – *Gros plan sur les processus physiques de la simulation : 1 événement avec 1 particule γ.*

Pour 2 événements avec 1 particule γ :

```
Energie totale déposée dans captor : 0 (MeV)

>>> Event 1 >>> Simulation truth :  1 gamma(s) (0,0,-6)

    9 hit (s) sont enregistrés dans captor.

n°hit: 0
        temps :4.753273 nsec
        position locale dans captor : (0,0,0.00045899009)(cm)
        énergie : 661.72228 (KeV)

n°hit: 1
        temps :4.7753117 nsec
        position locale dans captor : (-0.39764928,-0.22897462,0.40468481)(cm)
        énergie : 320.71102 (KeV)

n°hit: 2
        temps :4.753273 nsec
        position locale dans captor : (0,0,0.00045899009)(cm)
        énergie : 404.03368 (KeV)

n°hit: 3
        temps :4.75739 nsec
        position locale dans captor : (-0.0044806509,-0.0049312038,-0.12647749)(cm)
        énergie : 3785.8995 (KeV)

n°hit: 4
        temps :4.7685071 nsec
        position locale dans captor : (-0.13951705,0.11656614,-0.45681976)(cm)
        énergie : 75.752543 (KeV)

n°hit: 5
        temps :4.7685071 nsec
        position locale dans captor : (-0.13951705,0.11656614,-0.45681976)(cm)
        énergie : 1.564 (KeV)

n°hit: 6
        temps :4.7796934 nsec
        position locale dans captor : (0.14358169,0.2952063,-0.47698411)(cm)
        énergie : 19.248019 (KeV)

n°hit: 7
        temps :4.75739 nsec
        position locale dans captor : (-0.0044806509,-0.0049312038,-0.12647749)(cm)
        énergie : 1.564 (KeV)

n°hit: 8
        temps :4.7599227 nsec
        position locale dans captor : (-0.0089716772,0.013001774,-0.20012349)(cm)
        énergie : 28.217832 (KeV)


Energie totale déposée dans captor : 5.2987129 (MeV)
```

Nous relançons un second événementau premier. Cette fois-ci, nous avons un faisceau qui pénètre dans le *captor* et des *hits* se produisent. Nous en comptabilisons **9** avec des temps, des positions et des énergies pour chacun. Sur la figure 3.10, nous avons les deux événements. Nous reconnaissons le premier par les deux γ partant dans des directions opposées (voir sur la figure 3.9). Le faisceau du second événement traverse le fantôme sans interagir avec lui. Il pénètre par la suite dans le captor en générant *9 hits* comme on peut le voir sur la figure 3.11. Par ordre d'énergie décroissant, nous avons bien en premier lieu la production de pair (e^-, e^+), le positron e^+ ionise son environnement avant de s'annihiler aussitôt avec un électron e^- du milieu et produire deux γ à $511KeV$. Tandis que l'électron e^- génère deux processus Bremsstrahlung. Et les γ produits par e^+ et e^- finissent leur course sur le bord du monde en générant l'effet Compton et photo-électrique.

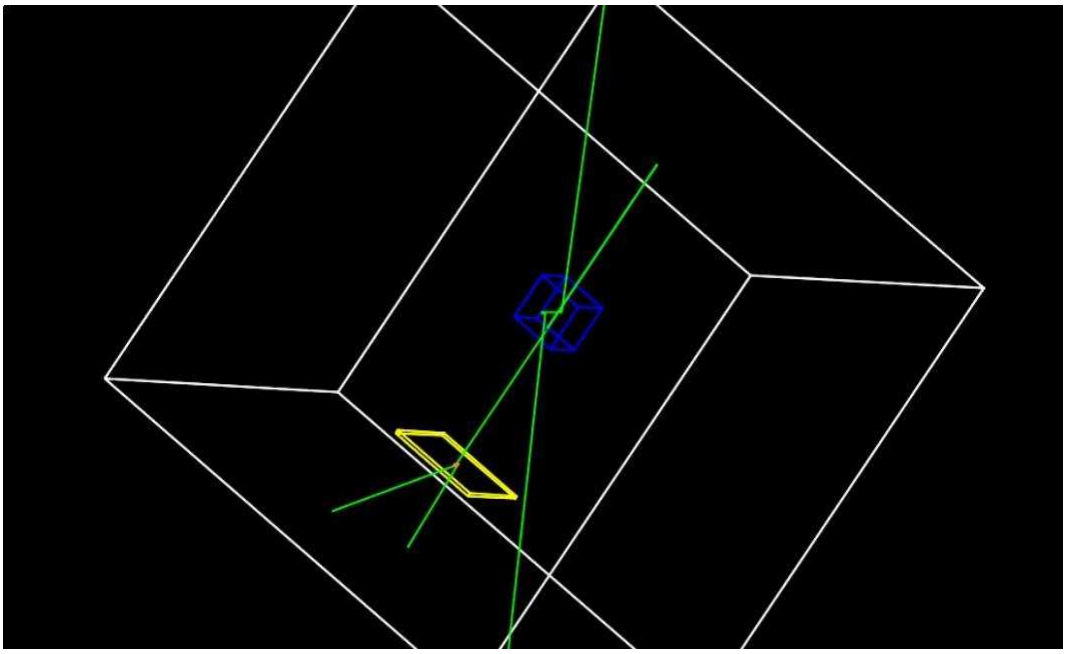


FIG. 3.10 – *Simulation de deux événements avec une particule γ.*

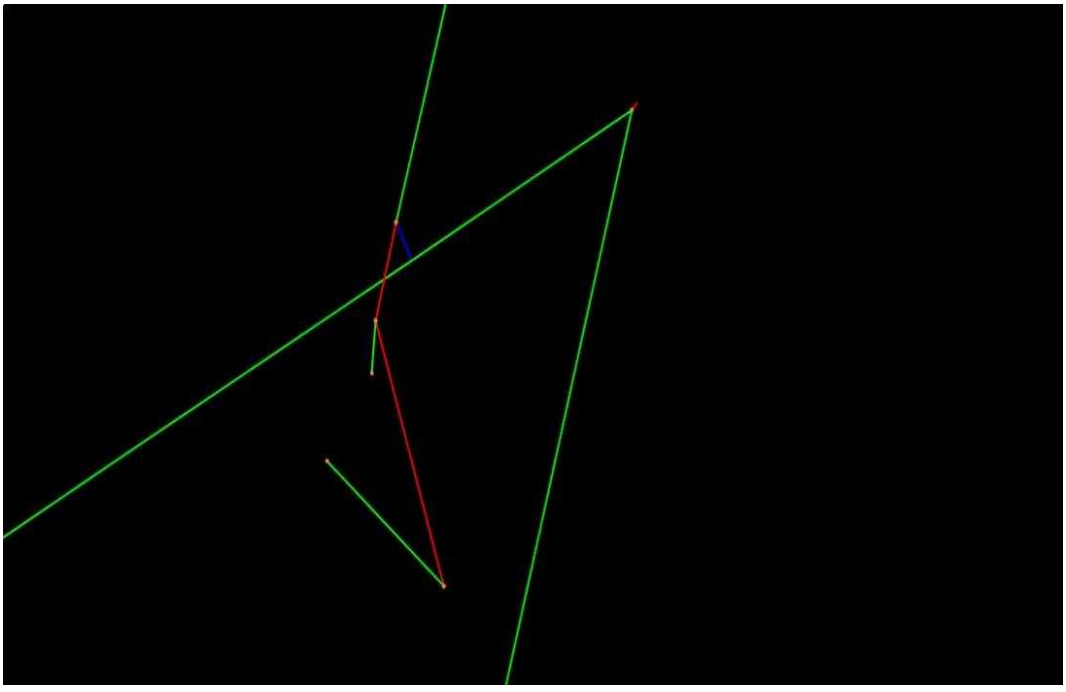


FIG. 3.11 – *Gros plan sur les processus physiques de la simulation : 2 événements avec 1 particule γ.*

Pour 1 événement avec 2 particules γ :

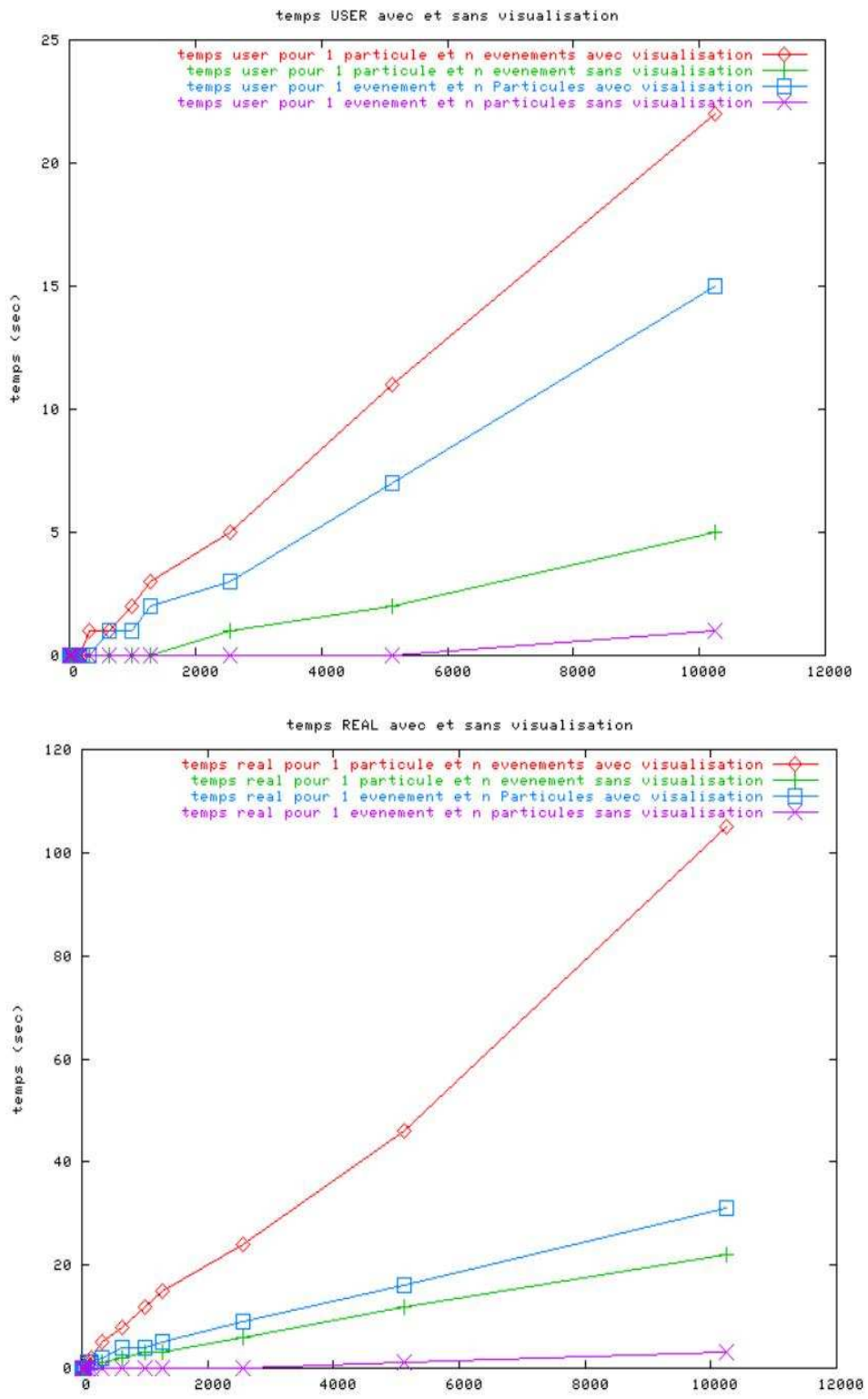
```
>>> Event 0 >>> Simulation truth :  2 gamma(s) (0,0,-6)

    9 hit (s) sont enregistrés dans captor.

n°hit: 0
        temps :4.753273 nsec
        position locale dans captor : (0,0,0.00045899009)(cm)
        énergie : 661.72228 (KeV)

n°hit: 1
        temps :4.7753117 nsec
        position locale dans captor : (-0.39764928,-0.22897462,0.40468481)(cm)
        énergie : 320.71102 (KeV)

n°hit: 2
        temps :4.753273 nsec
        position locale dans captor : (0,0,0.00045899009)(cm)
        énergie : 404.03368 (KeV)

n°hit: 3
        temps :4.75739 nsec
        position locale dans captor : (-0.0044806509,-0.0049312038,-0.12647749)(cm)
        énergie : 3785.8995 (KeV)

n°hit: 4
        temps :4.7685071 nsec
        position locale dans captor : (-0.13951705,0.11656614,-0.45681976)(cm)
        énergie : 75.752543 (KeV)

n°hit: 5
        temps :4.7685071 nsec
        position locale dans captor : (-0.13951705,0.11656614,-0.45681976)(cm)
        énergie : 1.564 (KeV)

n°hit: 6
        temps :4.7796934 nsec
        position locale dans captor : (0.14358169,0.2952063,-0.47698411)(cm)
        énergie : 19.248019 (KeV)

n°hit: 7
        temps :4.75739 nsec
        position locale dans captor : (-0.0044806509,-0.0049312038,-0.12647749)(cm)
        énergie : 1.564 (KeV)

n°hit: 8
        temps :4.7599227 nsec
        position locale dans captor : (-0.0089716772,0.013001774,-0.20012349)(cm)
        énergie : 28.217832 (KeV)


 Energie totale déposée dans captor : 5.2987129 (MeV)
```

Effectivement les valeurs obtenus entre deux événements à une particule γ et un événement à deux particules γ sont identiques. Nous le vérifions également avec n événements à une particule γ et un événement à n particules. Nous pouvons alors penser à une relation.

Soit une relation $f(nbEvent, nbParticle)$ donnant des valeurs d'une variable.

$$f(nbEvent, 1) = f(1, nbParticle) \text{ si } nbEvent = nbParticle$$

Nous choisirons l'une des deux façons de faire des simulations en fonction des performances de calculs de notre machine. Nous représentons graphiquement ci-dessous les performances en temps de calcul avec un Pentium M à 1,4 Ghz et 512 Mo de RAM.



Nous observons que dans les deux cas de temps *USER* et *REAL*, les temps de calcul pour 1 particule et n événements sont très élevés par rapport au temps de calcul pour 1 événement et n particules. Et les temps sont davantage plus court sans visualisation graphique. Pour cette démonstration, nous avons fait des simulations pour au plus 10240 particules ou événements. Pour des résultats similaires, il est donc sans conteste que nous gagnons du temps de calcul sans visualisation avec n particules et 1 événement.

En réalité, nous aurons à simuler environ un million de particules. Pour exemple avec notre géométrie simpliste, nous avons les temps de calcul suivants pour 10^6 d'événements et 1 particule, puis 1 événement et 10^6

de particules. Nous avons des temps d'attente respectifs d'environ 16 minutes contre environ 2 minutes.

```
>>> Event 999999 >>> Simulation truth :  1 gamma(s)  (0,0,-6)

    0 hit (s) sont enregistrés dans captor.


 Energie totale déposée dans captor : 0 (MeV)

Run terminated.
Run Summary
  Number of events processed : 1000000
  User=472.57s Real=976.3s Sys=13.67s
Graphics systems deleted.
Visualization Manager deleting...
G4 kernel has come to Quit state.
```

```
Start Run processing.

>>> Event 0 >>> Simulation truth :  1000000 gamma(s)  (0,0,-6)

    256425 hit (s) sont enregistrés dans captor.


 Energie totale déposée dans captor : 201099.47 (MeV)

Run terminated.
Run Summary
  Number of events processed : 1
  User=78.08s Real=143.22s Sys=1.31s
Graphics systems deleted.
Visualization Manager deleting...
G4 kernel has come to Quit state.
```

En toute évidence, nous n'aurons donc pas grand intérêt à représenter les résultats dans un terminal et voir même à chercher à les visualiser. Le mieux serait de les copier dans un fichier texte et d'exploiter ce fichier avec *gnuplot* pour faire des graphiques. Ou encore, nous pourrions utiliser les outils graphiques *AIDA* [20] ou *JAS3* [21] que la communauté *Geant4* ont développé.

3.5 Conclusion

Loin d'être une librairie évidente, Geant4 est vaste et complexe dont des mises à jour régulière se développe, incluant les toutes dernières recherches en matière de théories et de résultats expériementaux de la physique des particules. Pouvant contruire des modèles réalistes quand on en a l'expérience et le temps, à notre échelle de compréhension, nous avons construit un monde simple rempli d'air dans lequel un fantôme en eau et un capteur en aluminium sont placés. Une source de particules primaires génère des gammas d'énergie 6 MeV en incidence verticale. Il s'en suit aléatoirement des interactions d'ordre électromagnétique et en conséquence des trajectoires de particules primaires et secondaires que l'on peut, dans une certaine limite en fonction du nombre de particules, vérifier avec le logiciel de visualisation WIRED. Nous affichons également des résultats sur des temps de production, des positions et des énergies de *hits*, produit d'une simulation avec quelques particules. Très vite, l'affichage d'un nombre considérable de résultats dans un terminal devient sans intérêt car inexploitable pour l'analyse.
En perspective, il est conseillé de copier les résultats dans des fichiers textes pour une plus commode manipulation ou même encore, de façon automatique, construire une classe pour l'analyse de résultats. Ces derniers pourront être exploiter par les logiciels AIDA ou JAS3, et y procéder à de multiples traitement de données. Sur cette base, nous pouvons entrer plus finement sur les fonctionnalités de Geant4 afin d'améliorer notre prémodèle. L'introduction de coupes d'image d'un patient avec des fichiers au format DICOM pour le fantôme

serait un grand pas. Ensuite l'optimisation et la configuration de la réponse du détecteur devant se comporter comme un imageur portale permettraient de commencer à appliquer ces résultats au contrôle des plans de traitement. L'expérience et les connaissances des physiciens du médical dans le domaine expériemental de la radiothérapie pourrait être mis à contribution dans les choix d'optimisation des processus physiques d'intérèts et ainsi optimiser les temps de calcul des simulations.

Bibliographie

[1] La norme DICOM :
http ://medical.nema.org.

[2] Les mises à jour de la norme :
http ://www/dclunie.com/dicom-status.html#CorrectionProposalsByNumber.

[3] Les 14 parties du standard DICOM :
http ://medical.nema.org/dicom/2003.html

[4] La documentation de la librairie dicomlib par doxygen :
http ://dicomlib.swri.ca/dicomlib/html/index.html

[5] La librairie ucdm99 :
http ://dicomlib.swri.ca/ucdmc99.html

[6] La librairie dicomlib :
http ://dicomlib.swri.ca/dicomlib.html

[7] La librairie socket :
http ://dicomlib.swri.ca/socket.html

[8] La librairie boost :
http ://sourceforge.net/project/showfiles.php ?group_id=7586

[9] La librairie scons :
http ://scons.sourceforge.net/

[10] La librairie CHLEP version 1.8.1.0 :
http ://proj-clhep.web.cern.ch/proj-clhep/export/share/CLHEP/1.8.1.0/clhep.1.8.1.0.tgz

[11] Site sur la librairie CLHEP :
http ://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/

[12] La documentation sur la librairie CLHEP :
http ://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/doxygen/html/index.html

[13] La librairie geant4 version 4.6.1 :
http ://wwwasd.web.cerne.ch/wwwasd/geant4/source/source/geant4.6.1.gtar.gz

[14] Les commandes de l'interface utilisateur de geant4 :
http ://wwwasd.web.cern.ch/wwwasd/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/Control/UIcommands/_.html

[15] Le visualiseur DAWN :
http ://geant4.slac.stanford.edu/g4cd/March2004/Documentation/Visualization/G4DAWNTutorial/G4DAWNTutorial.html

[16] Le visualiseur OpenGL :
http ://geant4.slac.stanford.edu/g4cd/March2004/Documentation/Visualization/G4OpenGLTutorial/G4OpenGLTutorial.html

[17] Documentations sur les outils d'analyse graphique JAS3/JAIDA pour geant4 :
http ://geant4.slac.stanford.edu/g4cd/March2004/Documentation/WorkshopExercises/Exercise5.html

[18] Le visualiseur WIRED :
http ://www.slac.stanford.edu/ wiredces/install.class

[19] La documentation sur la librairie geant4.6.1 :
http ://pcitapiww.cern.ch/asdcgi/geant4/SRM/G4GenDoc.csh ?flag=1

[20] Outil graphique AIDA : *http ://aida.freehep.org/dev/geant4/index.html*

[21] Outil graphique JAS3 : *http ://jas.freehep.org/jas3/documentation.html*