ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAITRISE EN GÉNIE ÉLECTRIQUE

PAR
LEMAY, Mathieu

ÉLABORATION D'UN INTERGICIEL POUR RELIER LES INSTRUMENTS AUX GRIDS

MONTRÉAL, LE 30 DÉCEMBRE 2007

CE RAPPORT DE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Mme Véronique François, directrice de mémoire
Département de génie électrique à l'École de technologie supérieure

Mme Christine Tremblay, présidente du jury
Département de génie électrique à l'École de technologie supérieure

M. Mohamed Cheriet, membre du jury
Département de génie de la production automatisée à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 30 NOVEMBRE 2007

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

AVANT-PROPOS

Le présent mémoire présente la version initiale de la modélisation de l'intergiciel GRIM. Puisque le projet de recherche continue et que de plus en plus de gens participent à l'amélioration du modèle et à la création d'outils, il se peut que les structures de données soient différentes de celles présentées dans ce mémoire. Afin d'obtenir la dernière version de l'intergiciel et sa spécification complète, vous devez aller sur le site suivant : http://forge.uclp.ca/projects/grim/.

Il est aussi important de noter qu'à cause de la contribution de différentes organisations, l'intergiciel résultant a été émis sous la licence de logiciel libre : Apache Software License version 2.

REMERCIEMENTS

Je tiens à remercier Matthew Arrott de CalIT2 pour sa contribution initiale et ses commentaires sur le modèle GRIM. De plus, je tiens aussi à remercier le groupe de recherche en technologies de réseaux à large bande du Centre de Recherche en Communications pour l'accès à leur équipement de laboratoire photonique. Merci à Dre. Véronique François pour son soutien et son suivi en tant que directrice de recherche. Un merci encore plus spécial à Christopher Olson qui a fait la majeure partie de l'implémentation du modèle.

ÉLABORATION D'UN INTERGICIEL POUR RELIER LES INSTRUMENTS AUX GRIDS

LEMAY, Mathieu

RÉSUMÉ

Les logiciels Grid sont en train de devenir une partie intégrale de la science électronique puisque la science moderne a besoin d'une grande capacité de calcul et une grande base de données d'information. Afin d'avoir des logiciels Grid capables de s'intégrer avec la science d'aujourd'hui, il faut que les instruments de mesure soient accessibles et représentés grâce à des intergiciels Grid de façon à ce qu'ils fassent partie de la Grid. Ce mémoire présente un résumé de la technologie des Grids, la conception du modèle et l'implémentation initiale de l'intergiciel appelé Grid Resource Instrument Model (GRIM) bâti à l'aide du WSRF pour les instruments et capteurs et inspiré par les standards IEEE1451, SensorML et TML. Le résultat de cette recherche est un intergiciel qui peut être utilisé par des applications Grid à des fins telles la planification et partage de laboratoires, le contrôle à distance d'instruments et la surveillance de capteurs.

Mot clés : Grid, WSRF, Services Web, Instruments

ELABORATION D'UN INTERGICIEL POUR RELIER LES INSTRUMENTS AUX GRIDS

LEMAY, Mathieu

ABSTRACT

Grid applications are becoming an integral part of e-Science because of science's need for intensive computing and its needs for a large knowledge base. In order to have capable applications that interact with real world science, the instruments must be accessible and represented with Grid middleware as a part of the Grid. This master's thesis presents the technology overview, model design and initial implementation of the middleware called Grid Resource Instrument Model (GRIM) built with WSRF for instruments and sensors inspired by the work done in IEEE1451, SensorML and TML. A use case will also be used to show virtualization's efficiency in a collaborative lab environment. The outcome of this research is a middleware that can be used by Grid applications for different purposes such as laboratory planning, instrument control and sensor monitoring.

Keywords : Grid, WSRF, Web Services, Instruments

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ACL	Affichage à Cristaux Liquides
CA	Certificate Authority
CIMA	Common Instrument Middleware Architecture
CN	Common Name
DN	Distinguished Name
DTE	Data Terminal Equipment
EDFA	Erbium Doped Fiber Amplifier
ÉTS	École de Technologie Supérieure
FIFO	First In, First Out
FQDN	Fully Qualified Domain Name
GML	Geographical Markup Language
GPIB	General Purpose Interface Bus
GRIM	Grid Resources for Instruments Model
GRIMML	GRIM Markup Language
GSI	Grid Security Infrastructure
GT3	Globus Toolkit version 3
GT4	Globus Toolkit version 4
HTTP	HyperText Transfer Protocol
ITU	International Telecommunication Union+
LOOKING	Laboratory for the Ocean Observatory Knowledge Integration Grid
NCAP	Network Capable Application Processor
NSF	National Science Foundation

OASIS	Organisation for the Avancement of Structured Information Standards
OGSA	Open Grid Service Architecture
OSA	Optical Spectrum Analyser
OU	Organisational Unit
PID	Proportionnel Integral Différentiel
REST	Representational State Transfer
SCPI	Standard Commands for Programmable Instruments
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SOS	Sensor Object Service
SRQ	Service ReQuest
SSL	Secure Socket Layer
STIM	Smart Transducer Interface Module
SWE	Sensor Web Enablement
TCP	Transport Control Protocol
TEDS	Transducer Electronic Data Sheet
TII	Transducer Independent Interface
TIS	Transducer Information Service
TML	Transducer Markup Language
UCLP	User Controlled LightPaths
UML	Unified Modeling Language
URL	Uniform Resource Locator

WDM	Wavelength Division Multiplexer
W3C	World Wide Web Consortium
WSRF	Web Service Resource Framework
XML	eXtended Markup Language
XML-RPC	XML Remote Procedure Call

INTRODUCTION

Les Grids ont vu le jour à partir des ensembles de grappes de serveurs « computer clusters » informatiques à la fin des années 90. Au début des années 2000, un des changements majeurs fut l'introduction de grids de données « data grids » comme le Open Science Grid, le Teragrid ou bien le Nordugrid, qui ajoutaient la mise en commun d'information aux capacités de calculs sur les serveurs. La prochaine génération des Grids sera du type « Resource Grid » où les réseaux, les instruments, la mémoire de stockage et les ordinateurs seront des ressources qui pourront être consommées par les utilisateurs des Grids.

La technologie Grid est une technologie dont on parle depuis environ 15 à 20 ans, soit depuis la création du système appellé Multics (Morgan, 1973). Le problème a toujours été l'implémentation de cette technologie. En effet, des architectures similaires au système Multics ont été tentées lors de l'arrivée des premières architectures distribuées. Par contre, les grosses difficultés qu'il fallait surmonter étaient la standardisation, la portabilité des applications et le réseau. Des applications clientes devaient être programmées spécifiquement pour contrôler les services offerts. Ainsi, la consommation de ces services était limitée par le nombre d'implémentations de clients dans une application scientifique restreinte. Dans un environnement fermé comme à l'interne d'une organisation, ces solutions étaient fantastiques; mais dans un contexte Grid géographiquement distribué où plein d'organismes, de politiques et d'environnements différents sont existantes, il fallait une solution capable d'interopérabilité. Ainsi, avec l'évolution de l'Internet et l'émergence de nouveaux standards Web, la plupart des frontières précédentes ont été brisées. Ceci est dû en majorité au fait que les technologies XML permettent d'avoir une grande compatibilité à cause, entre autres, aux métadonnées contenues avec les données.

Figure A ***Graphique des trois piliers de l'Internet de la prochaine génération.***

Ainsi naquirent les Grids telles qu'on les conçoit aujourd'hui, quelques années seulement après les premières implémentations de service Web. Bien sûr, cette technologie est encore embryonnaire, mais elle présente un potentiel inestimable. On peut voir sur la figure A les relations entre les différents aspects qui sont des piliers importants de l'Internet de la nouvelle génération. On peut voir que les trois fondements de la technologie sont les besoins de bande passante, de stockage des données et de collaboration.

Ainsi, les applications futures pourront faire usage des ressources collectives. On peut voir sur la figure B que les ressources primaires et matérielles virtualisées sont ce qu'on appelle « *Fabriques* », ou bien ressources grids. Les initiatives telles que le GRIM (Grid Resource Instrument Model), sujet du présent mémoire, ou bien l'intergiciel UCLP (User Controlled LightPaths) présenté par Recio *et al.* (2004), visent à créer la couche de fabrique nécessaire.

Figure B ***Couches du GRID***

Une grosse partie de la problématique visée par ce mémoire est que, présentement, les instruments et capteurs ne font pas partie des Grids. Les chercheurs doivent donc entrer manuellement les résultats afin d'en faire le traitement en temps différé, ce qui est un grand désavantage. Cette initiative vise la création d'un modèle standard qui permettra d'interagir avec différents types d'instruments de façon uniforme.

Au préalable, qu'est-ce qu'un **instrument** ? Un instrument est un ensemble de capteurs qui est capable de faire une mesure et un prétraitement des données avant de fournir un résultat. Un instrument peut donc aussi bien représenter un appareil physique dans un laboratoire, soit un instrument de mesure, qu'un appareil « virtuel », qui a comme base des centaines de capteurs traités selon différents paramètres d'entrée supplémentaires.

Dans le cadre de ce mémoire, le terme de **capteur** sera utilisé pour représenter les dispositifs de type « transducteur », soit un dispositif électronique capable de convertir un phénomène physique en signal électrique ou vice-versa. De plus, lors de la modélisation, les termes capteur et instrument seront souvent utilisés conjointement ou de façon interchangeable puisque, par définition, l'instrument est un capteur muni d'une unité de contrôle et de

traitement de données, ce qui peut être fait au niveau ou matériel, ou bien logiciel, grâce au modèle GRIM.

Afin de créer le GRIM, un standard "Plug-And-Play" a été utilisé comme référence. Ce standard, l'IEEE1451, permet de faire la représentation de n'importe quel capteur ou instrument en réalisant une modularisation des différentes entités de base pouvant s'y retrouver.

Le chapitre CHAPITRE 1 de ce mémoire fait un retour sur la littérature et les technologies qui sont nécessaires à la création de ressources Grid. Le chapitre 2 est consacré aux ressources Grids et aux standards qui les composent afin de pouvoir assurer l'interopérabilité et la portabilité requises. Le chapitre CHAPITRE 3, lui, est axé sur le modèle GRIM et les structures de données qui le composent. Le laboratoire photonique de l'ÉTS sera virtualisé au cours du chapitre 4 afin de pouvoir répondre à l'exemple d'utilisation du chapitre 5, qui consiste à partager l'équipement de laboratoire au sein d'une salle de classe grâce à la virtualisation.

CHAPITRE 1

RÉCENTS DÉVELOPPEMENTS AU NIVEAU DES INITIATIVES DE VIRTUALISATION DE CAPTEURS ET D'INSTRUMENTS

Les initiatives importantes de virtualisation de capteurs et d'instruments sont le Common Instrument Middleware Architecture (CIMA) et le Sensor Web Enablement, que nous allons décrire ici avant de nous concentrer sur notre projet, le Grid Resource for Instruments Model (GRIM).

1.1 Le Common Instrument Middleware Architecture (CIMA)

Le CIMA, tel que décrit par l'article de Devadithya *et al.* (2005), est une des initiatives les plus importantes de création d'intergiciels capables de représenter les instruments et capteurs dans les grilles scientifiques. Cette section vise à cerner les forces et les faiblesses de cette architecture afin de discerner pourquoi elle n'a pas été adoptée par la majorité des projets de recherche voulant virtualiser leurs instruments pour les intégrer dans les Grids. La première partie expose les motivations, l'approche et l'architecture de CIMA alors que la deuxième partie présente un cas d'usage typique d'utilisation de CIMA en cristallographie.

1.1.1 Motivations dernière le CIMA

La majorité des Grids actuelles servent à traiter et calculer les données scientifiques. Ces Grids, communément appelées « Data Grids », ne font pas l'intégration directe des sources d'informations, très souvent des capteurs et/ou des instruments.

La plupart du temps, les données sont introduites manuellement dans la grille par les utilisateurs. Ceci est acceptable seulement lorsque les données sont archivées et que l'analyse scientifique peut être faite en temps différé. Par contre, de plus en plus d'expériences scientifiques nécessitent un traitement mathématique en temps réel par les grilles informatiques afin de permettre une visualisation instantanée de ces données.

Des sciences telles la climatologie, l'astronomie et la physique quantique nécessitent des capteurs et instruments qui génèrent des pétaoctets d'informations par année. C'est pourquoi il est important que les capteurs et instruments fassent partie des Grids pour procurer le traitement en temps réel des données afin de n'avoir à archiver que les résultats pertinents. Il y aussi d'autres avantages à intégrer les instruments et capteurs dans les Grids : ces instruments ne sont pas géographiquement dépendants l'un de l'autre et peuvent être partagés par plusieurs chercheurs en même temps. Le projet CIMA vise à créer une interface commune à ces capteurs et instruments pour les Grids.

1.1.2 Objectifs de conception

Le modèle CIMA fait une distinction entre deux scénarios, similaires mais distincts, à savoir l'accès à distance « Remote Access » et les opérations distribuées « Distributed Operations ». L'*accès à distance* consiste à faire à distance les tâches qui seraient autrement effectuées localement. Ainsi le contrôle, l'acquisition des données et l'analyse sont faites de façon centralisée autour de cet instrument. Lorsqu'on parle d'opérations distribuées, ces différentes fonctionnalités sont décentralisées et un « instrument » peut être conceptuellement formé de ressources provenant de différents emplacements géographiques ou institutions. Un bon exemple est un instrument dont les capteurs de données sont différents télescopes, où les données sont ensuite envoyées et traitées par un centre de calcul sophistiqué et analysés par des scientifiques d'un laboratoire d'experts en astronomie. Le tout peut aussi offrir une interface d'interaction unique à l'utilisateur pour qu'il puisse voir cet « instrument » comme n'importe quel « appareil » conventionnel sans nécessiter les connaissances requises pour faire le traitement des données.

Lors de la création du modèle CIMA, les aspects suivants ont été ciblés comme étant les plus importants : amorçabilité, interopérabilité, transfert de données efficace et légèreté de l'intergiciel.

Afin que le CIMA soit *amorçable*, il doit pouvoir être capable de rendre un instrument opérationnel dans la Grid sans nécessairement connaître les structures de données externes spécifique à l'instrument au préalable.

En ce qui a trait à l'*interopérabilité*, les instruments créés par une organisation doivent pouvoir être utilisés par une autre organisation sans que celle-ci ne reçoive d'information sur l'opération offerte par cet instrument par un moyen externe : toute l'information nécessaire doit être véhiculée par l'intergiciel CIMA.

En ce qui a trait au *transfert efficace de données*, il faut que la vitesse de transmission de l'information par l'instrument soit moins élevée que la bande passante du lien qui le relie à la Grid. C'est spécialement important lorsqu'il y a eu une agrégation des données, car la bande passante peut devenir énorme et bouger ces flux de données d'un endroit à l'autre peut être limité par l'infrastructure de télécommunication sous-jacente à la Grid.

Finalement, lors de l'installation d'un instrument sur un site, il y a des restrictions de consommation de puissance électrique, de bande passante et de capacité de calculs. C'est pourquoi l'intergiciel doit être *léger*, afin de ne pas consommer la majorité des capacités du microcontrôleur contrôlant l'instrument.

1.1.3 Approche et architecture de CIMA.

L'approche adoptée par CIMA vise à utiliser les stratégies suivantes : une spécification à couches, des plugiciels, un couplage lâche et des modèles « Pull » et « Push ».

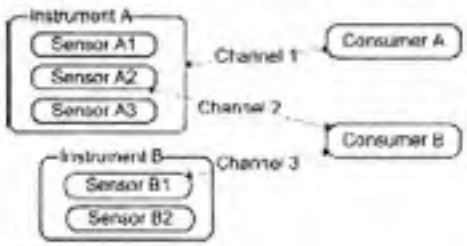

Figure 1.1 ***Architecture CIMA de représentation des instruments.***

Source : La figure est tirée de l'article, *The Common Instrument Middleware Architecture: Overview of Goals and Implementation*, Titre original Figure 2. Instrument Model. (*Voir* Devadithya, 2005).

L'*architecture à couches* vise à permettre la réutilisation des différents éléments afin de pouvoir créer rapidement de nouveaux éléments et de nouvelles fonctionnalités en ayant un minimum de changements à effectuer sur le code source. On peut voir sur la Figure 1.1 que les instruments contiennent des capteurs « Sensor » et qu'il est possible de créer des liens entre les utilisateurs « Consumer » de ces services avec l'instrument comme un tout, ou bien au contraire avec des capteurs spécifiques. Lorsque ce lien, appelle « Channel », est créé grâce à un enregistrement de l'utilisateur auprès du capteur, qu'on appelle « Publish-Subscribe », il est possible que les utilisateurs reçoivent les données au fur et à mesure qu'elles changent.

Les *plugiciels* peuvent être utilisés afin de permettre d'insérer des fonctions communément utilisées dans différents types d'instruments ou de capteurs. Ainsi les fonctions qui sont souvent communes, comme calculer des moyennes et envoyer uniquement le résultat, peuvent être insérées directement pour faire ces tâches. Ces plugiciels sont insérés entre les utilisateurs et les capteurs ; on y réfère comme à des intermédiaires de traitement. Ils utilisent

tous exactement le même format de message et qui est transparent pour les utilisateurs des capteurs.

Le *couplage lâche* sert à pouvoir décider les liens de communications uniquement lors de l'utilisation et la communication n'est pas définie par l'architecture. Il est implémenté grâce à un message qui est basé sur des services « Document Oriented » (voir Chapitre 2) ; ainsi, la dépendance des liens de communication entre l'utilisateur et le capteur est limitée. Chaque message XML peut être interprété grâce aux métadonnées incluses dans le message. Un exemple d'un tel message à la Figure 1.2 permet de voir que le modèle d'information est très simple. En effet, le <Parcel> du haut représente la requête provenant de l'utilisateur. Il procure l'information le représentant au capteur. Le capteur ajoute cet utilisateur à sa liste et envoie des messages similaires au <Parcel> du bas qui contient l'information mesurée.

```
<Parcel>
  <ID>http://<consumer-ip>/<consumer-port>
        /2005/02/25/0001</ID>
  <Type>http://www.cs.indiana.edu
        /2004/register</Type>
  <Location>inline</Location>
  <Body>
    <Time>2005-02-25T11:31:22Z</Time>
    <Consumer>
      <Host>tiger.cs.indiana.edu</Host>
      <Port>2000</Port>
    </Consumer>
    <DataInterval>5</DataInterval>
  </Body>
</Parcel>

<Parcel>
  <ID>http://<sensor-ip>/<sensor-port>
        /2005/02/25/0991</ID>
  <Type>http://www.cs.indiana.edu
        /2004/temperature</Type>
  <Location>inline</Location>
  <Body>
    <Time>2005-02-25T17:24:30Z</Time>
    <Temperature>23.453</Temperature>
  </Body>
</Parcel>
```

Figure 1.2 ***Exemple de message CIMA.***

Source : L'exemple est tirée de l'article, *The Common Instrument Middleware Architecture: Overview of Goals and Implementation.* (*Voir* Devadithya, 2005).

De plus, il est possible d'envoyer ces messages de façon « Pull » ou bien « Push ». La méthode « Pull » consiste à envoyer de façon répétitive le <Parcel> du haut et de recevoir immédiatement comme réponse le <Parcel> du bas. La méthode « Push » est un peu différente puisqu'elle permet à un capteur de générer lui-même les <Parcel> du bas sans avoir à envoyer le <Parcel> du haut à chaque fois. Il suffit de l'envoyer une seule fois pour inscrire l'utilisateur qui recevra de façon périodique les messages suivants.

1.1.4 Inconvénients de l'utilisation de CIMA

Le gros problème avec CIMA est qu'il n'a été implémenté que pour un seul appareil : un instrument de mesure en cristallisation. En effet, aucun « Toolkit » n'est encore disponible et l'implémentation n'est pas publique. De plus, le modèle n'addresse pas comment rapidement créer des services pour des appareils existants qui ont des protocoles propriétaires de communication. Standard Commands for Programmable Instruments (SCPI) permet d'avoir une syntaxe standardisée pour le port GPIB ; par contre, ce ne sont pas tous les instruments qui l'utilisent et ceux qui le font ont un ensemble de commandes qui leur est très spécifique.

Un des gros désavantages du CIMA est que, même si l'information est incluse dans les métadonnées, il est difficile de connaître exactement comment la valeur est acquise. Il n'y a aucun détail sur le type de capteur, ses limites d'utilisation, la structure des données (scalaire ou vectorielle) ou bien sur les unités physiques de ces mesures.

Même si CIMA utilise le Simple Object Access Protocol (SOAP) et les Services Web pour créer une représentation de l'instrument, il n'est pas basé entièrement sur la notion de ressources telles que spécifiées par les standards du Web Service Resource Framework (WSRF) et n'incorpore aucun aspect de sécurité à l'intérieur du modèle, les deux piliers des grids modernes que nous verront au chapitre 3. Il ne peut pas non plus créer des ressources virtuelles à partir de descriptions de capteurs utilisant les standards de représentation de capteurs SensorML et TML.

Finalement, même s'il est financé par la NSF et que les buts et l'approche semblent bons, il reste beaucoup à faire afin de faire de CIMA l'intergiciel de choix. D'autant plus que celui-ci n'est pas téléchargeable.

CIMA a été créé principalement pour interagir avec les instruments de laboratoire. Une autre initiative, le Sensor Web Enablement (SWE), a été créée afin de virtualiser les capteurs et leurs flux de données. Cette initiative sera décrite dans la prochaine section.

1.2 Sensor Web Enablement : une initiative du Open Geospacial Consortium (OGC)

Dans la section précédente, l'ensemble d'intergiciels CIMA a été présenté. Un des problèmes importants de cet ensemble d'intergiciels est que les capteurs et instruments ne sont pas bien décrits puisque transmettre uniquement la valeur mesurée sous forme de chaîne de caractère n'est pas suffisant pour faire une analyse scientifique convenable. Il est impossible de savoir quelle est l'erreur de numérisation, comment les capteurs sont calibrés ou bien quelle sont les spécifications du capteur. Les standards développés dans le cadre du programme Sensor Web Enablement (SWE) visent à adresser cette problématique. Les standards du SWE étudiés afin d'en inspirer le GRIM furent principalement le SensorML et le Transducer Markup Language (TML).

1.2.1 Introduction au SensorML

Le SensorML est un standard qui vise à décrire les capteurs grâce au XML (*Voir* Chapitre 2). En plus de modéliser les capteurs, il permet aussi de décrire le prétraitement effectué par ces capteurs. Il ne sert pas à modéliser ou à contrôler les instruments complets.

Les motivations derrière SensorML sont de pouvoir décrire de façon efficace la géométrie, la dynamique et les interfaces de communication de capteurs dynamiques distants. Il est essentiellement un « XML Schema » dont la structure de données complexe permet de faire une description adéquate pouvant s'inscrire dans l'initiative « Sensor Web Enablement ».

L'approche de SensorML est de créer des entités qui utilisent un serveur Web qui crée des flux de données « Streams » de façon continue. Ainsi, contrairement au CIMA, il ne tente pas de faire la virtualisation des capteurs pour les insérer aux Grids, mais bien vise à standardiser le modèle de données servant à décrire les capteurs. Cette architecture ressemble à la caractéristique dite « Push » offerte par CIMA, les messages n'étant pas lus par les clients mais biens poussés automatiquement par le serveur Web modifié utilisé vers le navigateur client.

Le SensorML est utilisé pour remplacer les feuilles de spécifications traditionnelles des capteurs en servant de feuille de spécification électronique. De plus, il est possible à partir d'un système de capteurs de faire la découverte des autres capteurs puisque le standard est riche en méta information et a une structure récursive. Finalement, la nature auto descriptive du SensorML permet d'ajouter de nouveaux capteurs aisément de façon prête à tourner « Plug and Play ».

1.2.2 Éléments XML du SensorML

Il y a différents types d'éléments dans une description SensorML. Voici les éléments les plus importants de cette description. On peut se référer à l'annexe IV pour un exemple de fichier SensorML d'une station météorologique simple.

1.2.2.1 Composants « Component »

Le *composant* est l'élément de base des systèmes. Lors de la création d'un modèle SensorML, il a une seule entrée et une seule sortie. Il peut être :

- un détecteur (entrée analogique, sortie numérique),
- un actionneur (entrée numérique, sortie analogique)
- un filtre (entrée et sortie du même type).

Pour créer des vecteurs, il suffit de faire des copies répétitives de ces composants.

1.2.2.2 Processus « Process »

Le *processus* est un élément complexe qui comporte une méthode et un modèle. Il sert à décrire ce qui est fait comme prétraitement de l'information par le capteur ou une entité logicielle intermédiaire.

1.2.2.3 Systèmes (« System »)

Les éléments *systèmes* de la structure de données sont formés de composants et associés à une définition de processus. Ils servent souvent à modéliser un ensemble de capteurs. Ils peuvent aussi contenir des sous-systèmes si nécessaire.

1.2.2.4 Modèle de processus (« Process Model »)

Le *modèle* de processus est l'élément de base d'un processus de traitement. Il utilise un ensemble de méthodes de processus afin de définir les différentes fonctionnalités offertes par celui-ci.

1.2.2.5 Méthode de processus (« Process Method »)

Les *méthodes* d'un processus décrivent l'interface de ce processus, soit l'ensemble de composants associés à ce processus ainsi que les différentes fonctionnalités qu'il peut accomplir.

1.2.2.6 Chaîne de processus (« Process Chain »)

Finalement une *chaîne de processus* de traitement sert à définir une séquence de différents processus et l'effet global sur les données. Un exemple de chaîne de processus pourrait être un processus de compression suivi d'un processus de cryptage qui permettraient d'avoir les données brutes disponibles sous forme d'un fichier sécurisé.

Toutefois, le SensorML ne permet pas le transfert adéquat des données d'un capteur à un autre. C'est pourquoi un autre standard créé par l'OGC et nommé Transducer Markup Language (TML) a été créé afin de gérer cet échange d'information entre les capteurs. La section suivante présente brièvement ce qu'est le TML.

1.2.3 La structure de données Transducer Markup Language (TML)

Même si le SensorML permet de modéliser les capteurs, il ne propose aucune solution pour que ceux-ci communiquent entre eux et/ou avec les applications. Le Transducer Markup Language (TML) est un nouveau standard, établi par la Défense nationale américaine (OGC, 2006), faisant lui aussi partie de l'initiative SWE (adopté en mars 2007). Il permet aux différents capteurs d'échanger de façon standardisée de l'information entre eux. Ainsi, il permet une interopérabilité sans égard au format des données, puisque toutes les données sont transférées en XML sous forme de chaines de caractères.

Le message TML ne vise pas uniquement à décrire le capteur mais aussi les caractéristiques physiques de sa réponse, la structure et le codage de ses données, ses caractéristiques temporelles et ses propriétés spatiales.

La modélisation du capteur se fait à l'initialisation. Un document similaire à celui de la Figure 1.3 est envoyé aux différents capteurs modélisés dans le système. Le même message est envoyé aux différents clients lorsque ceux-ci accèdent à un capteur pour la première fois.

```
<tml>
    <system>
        <systemClock>...period, count accy</systemClock>
        <transducers>...transducer models...</transducers>
        <process>...process models...</process>
        <relations>...transducer relationships...</relations>
    </system>
    <prodDataDesc>ID mapping,parsing, encoding and sequencing...<prodDataDesc>
<tml>
```

Figure 1.3 ***Exemple de modélisation TML.***

Source : L'exemple est tirée de la spécification OGC du TML, Transducer Markup Language Implementation Specification. URL : http://www.transducerml.org/downloads/TML.spec_100.doc Consulté le 30/12/2006.

On peut voir sur la Figure 1.4 que par la suite seules les données sont envoyées en XML simple. Ces données sont accompagnées de références (ref) et de « TimeStamp » (dk). La référence permet de savoir à quel modèle cette donnée est associée alors que le « TimeStamp » permet de savoir à quel moment la mesure a été prise.

```
<tml>
    <data ref="t001" clk="3F63B6432674">...transducer data...</ data >
    <data ref="t001" clk="3F63B64326A1">...transducer data...</ data >
    <data ref="t002" clk="3F63B6432701">... transducer data ...</ data >
    <data ref="t001" clk="3F63B6432723">... transducer data ...</ data >
    <data ref="t003" clk="3F63B643273C">... transducer data ...</ data >
    <data ref="t001" clk="3F63B6432767">... transducer data ...</ data >
    <data ref="t006" clk="3F63B6432788">... transducer data ...</ data >
    <data ref="t001" clk="3F63B64327E9">... transducer data ...</ data >
    <data ref="t001" clk="3F63B6432810">... transducer data ...</ data >
    <data ref="t001" clk="3F63B6432825">... transducer data ...</ data >
    <data ref="t008" clk="3F63B643281B">... transducer data ...</ data >
    <data ref="t001" clk="3F63B6432856">... transducer data ...</ data >
    <data ref="t002" clk="3F63B6432850">... transducer data ...</ data >
<tml>
```

Figure 1.4 ***Exemple de données dynamiques.***

Source : L'exemple est tirée de la spécification OGC du TML, Transducer Markup Language Implementation Specification. URL : http://www.transducerml.org/downloads/TML.spec_100.doc Consulté le 30/12/2006.

Le TML utilise aussi le Geography Markup Language (GML) afin de faire le positionnement géographique relatif d'un capteur par rapport à l'autre. Ainsi, il est possible de créer des systèmes complexes, ayant des dépendances relatives fixes, par rapport aux données captées par d'autres capteurs.

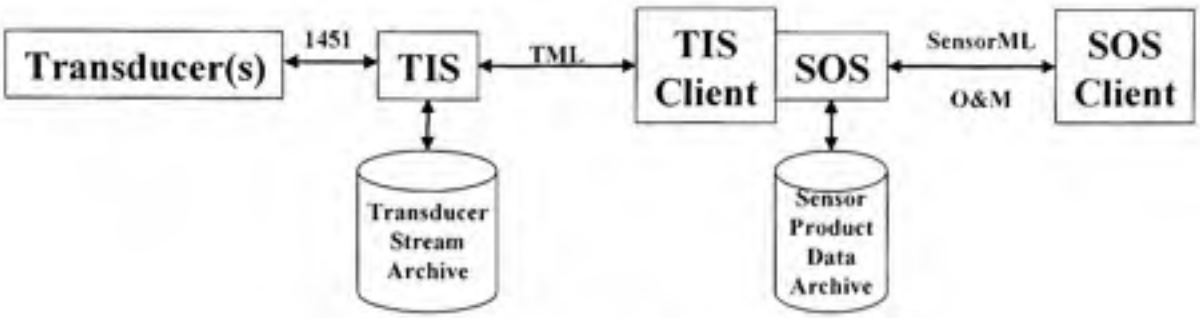

Figure 1.5 ***Système utilisant TML dans la solution.***

Source : L'image est tirée de la spécification OGC du TML, Transducer Markup Language Implementation Specification, Le titre original de la figure est « Figure 8. Example TML System ». URL : http://www.transducerml.org/downloads/TMLspec_100.doc . Consulté le 30/12/2006.

Sur la Figure 1.5, on peut voir que le TML sert à relier le « Transducer Information Service » (TIS) au client TIS. On peut aussi remarquer que le TML n'est pas créé directement par le capteur mais bien par le TIS. La communication entre le capteur et le TIS se fait à l'aide d'une famille de standards, l'IEEE1451 - Smart Transducer Interface Standards.

Contrairement aux initiatives CIMA et SWE, l'initiative IEEE1451 consiste à établir une famille de standards qui vise la création de transducteurs (capteurs et actionneurs) prêts à tourner « Plug and Play » et qui peuvent facilement avoir une représentation logicielle dans un point d'accès réseau appelé le Network Capable Access Point (NCAP).

1.3 L'initiative IEEE1451 et les objets logiciels pour transducteurs et instruments

Les transducteurs, définis comme la famille générique de capteurs et d'actionneurs, ont plusieurs champs d'applications et différents fabricants. De façon à ce qu'un certain niveau d'interopérabilité soit possible, une famille de standards, l'IEEE1451, a été créée afin d'être adoptée à l'unanimité par les fabricants de capteurs ainsi que par les créateurs de logiciels de contrôle pour ces capteurs.

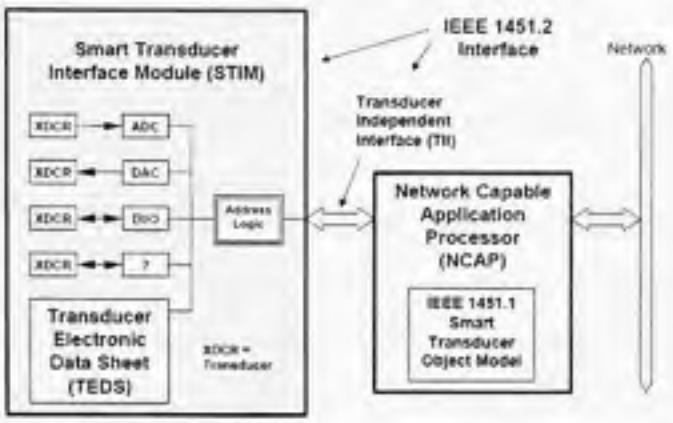

Figure 1.6 ***Architecture des composants de l'IEEE1451.***

Source : L'image est tirée de l'article de Kang Lee, IEEE 1451: A Standard in Support of Smart Transducer Networking. Le titre original de la figure est « Figure 2. Framework of IEEE 1451.1 and 1451.2 Interfaces ». URL : http://ieee1451.nist.gov/IM5340_F.pdf Consulté le 30/06/2007.

Les TEDS (Transducer Electronic Data Sheet) sont une version électronique des capacités du capteur ou actionneur suivant un format de données strict et spécifique défini par l'IEEE 1451.0 et 1451.2. Les TEDS ainsi que leur plage d'adressage sont inclus avec les différents capteurs dans ce qu'on appelle un transducteur intelligent (Smart Transducer) ; l'interface de transducteur intelligent est le Smart Transducer Interface Module (STIM), défini par les standards IEEE 1451.2, 1451.3 et 1451.4.

Le Network Capable Application Processor (NCAP) est l'élément qui a inspiré le modèle GRIM, puisqu'il permet de représenter de façon prête à tourner « Plug and Play » n'importe quel STIM qui est connecté via le Transducer Independent Interface (TII). Il a une structure de données orientée objet complexe et générique, qui permet de correctement modéliser n'importe quel instrument ou capteur. C'est d'ailleurs cet élément qui a inspiré la recherche de ce présent mémoire. Le modèle de données riche offert par le IEEE 1451.1 peut-il être traduit sous forme de ressources Grid afin d'offrir une solution aussi versatile et « Plug and Play » aux individus désirant connecter leur instruments aux Grids ?

CHAPITRE 2

THÉORIE DES GRIDS MODERNES

Afin de mieux comprendre pourquoi les Grids sont si importantes, il faut d'abord savoir ce qu'elles sont et ce qu'elles peuvent procurer à la communauté scientifique. Au début des années 90, le Web a changé la face du monde moderne en permettant la mise en commun de l'information sur le réseau Internet. Les grids de la nouvelle génération sont une extension du Web et permettent de mettre en commun des « ressources » comme de l'espace disque, du temps de calcul, des liens réseaux ou même des instruments de mesures.

Le concept de grid n'est pas aussi récent que la mise en œuvre. En effet, on peut retracer les concepts de grids aux années 60 où un projet appelé Multics mettait de l'avant un concept de service de calculs grid. Ce n'est par contre qu'en 1997, dans un atelier à l'Argonne National Lab, que les premiers concepts de grids ont été présentés par Ian Foster et Carl Kesselman dans un atelier informel intitulé « Building a computational grid » de l'organisation Globus. À la fin de '97, les résultats de cet atelier ont été publiés dans un article intitulé « Globus : a Metacomputing Infrastructure Toolkit » [8]. Les résultats de ces travaux furent par la suite l'outil fondamental de la majorité des grids : le *Globus Toolkit*. Le *Globus Toolkit* a subi depuis plusieurs itérations qui ont standardisé la façon d'échanger l'information et de créer de différentes ressources Grids de façon sécuritaire. Au début des années 2000, l'initiative *Open Grid Service Architecture* (OGSA) a décrit la majorité des besoins fonctionnels des grids pour faciliter la science électronique. De cette initiative sont nés le *Open Grid Service Infrastructure* (OGSI) et le *Globus Toolkit 3* (GT3).

En parallèle, l'évolution du Web a donné naissance à de nouvelles technologies de traitement de l'information et à de nouveaux standards d'interopérabilité. Les organismes comme l'*Organisation for the Avancement of Structured Information Standards* (OASIS) et le *World Wide Web Consortium* (W3C) ont permis l'élaboration de protocoles et formats de données universels qui n'ont aucune dépendance avec les langages de programmation ou la

plateforme d'exécution. Le format de données *eXtended Markup Language* (XML) et le protocole *Simple Object Access Protocol* (SOAP) ont permis de distribuer les différentes entités logicielles sur différents ordinateurs qu'on nomme maintenant les services Web.

Les grids allaient inévitablement rencontrer les nouvelles technologies Web et cela s'est produit au printemps 2005 lorsque Globus a lancé le *Globus Toolkit 4* (GT4). Le GT4 a permis de soumettre plusieurs nouveaux standards à OASIS qui procurent une extension aux services Web afin qu'ils puissent être réutilisés efficacement dans une grid. Cet ensemble de standards porte désormais le nom de *Web Service Resource Framework* (WSRF) et est la technologie de choix utilisée afin de virtualiser les instruments / capteurs en tant que ressources grid. Ce chapitre est un survol de toutes les technologies importantes à la compréhension du fonctionnement des ressources grids qui seront crées par le modèle proposé au chapitre 3.

2.1 Description du format XML

Il faut d'abord commencer par le XML puisqu'il est à la base de presque tous les échanges d'information sur le Web et est inspiré par le format de fichier le plus répandu sur l'Internet : le *HyperText Markup Language* (HTML).

2.1.1 Historique du XML

Le langage eXtended Markup Language (XML) est un sous-ensemble de l'effort de standardisation SGML (Standard Generalized Markup Language) fait par l'organisme W3C qui visait à généraliser le format HTML utilisé pour décrire les pages Web. Le SGML est né à la fin des années 80 avant l'expansion rapide de l'Internet. Par contre, il fallu plus de dix ans avant que le premier brouillon (Working Draft) pour XML soit déposé en juillet 1996. La dernière édition du standard XML 1.0 a été publiée le 16 Août 2006 et cette version est considérée la version courante du XML.

2.1.2 Qu'est-ce que le XML?

Le XML est une méthode pour structurer les données (Rambhia, 2002). Elle procure une façon de représenter l'information d'un document sous forme d'arborescence tout en restant lisible par les experts. Grâce aux différents standards, ce document peut être facilement manipulé par les systèmes ordinés. Pour ce faire, on utilise des caractères spéciaux qui délimitent l'information de la méta-information.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Etudiant niveau="maitrise" programme="ele">
3  <Nom>Lemay</Nom>
4  <Prenom>Mathieu</Prenom>
5  </Etudiant>
```

Figure 2.1 ***Listage d'un document XML.***

On peut voir sur la Figure 2.1 un exemple simple de document XML. L'information qui se retrouve entre les caractères '<' et '>' se nomme un ***élément***. Ainsi `Etudiant, Nom` et `Prenom` sont les éléments qui délimitent l'information. De plus, les éléments peuvent avoir des ***attributs*** (ex. niveau="") qui viennent préciser davantage les données ou les autres éléments internes.

Tableau 2.1

Caractères spéciaux XML

Caractère	Nom	Description
&	Esperluette	Sert à insérer des caractères réservés dans le document.
< et >	Crochets	Délimitent le début et la fin des balises.
" et '	Guillemets (Doubles ou Simples)	Délimitent les valeurs données aux attributs.

Le tableau précédent montre les caractères spéciaux utilisés dans les documents XML. Il est très important de ne pas utiliser ces caractères dans l'information sinon le document sera invalide.

2.1.3 Les forces et faiblesses du XML

L'utilisation du format XML fut sujette à de nombreuses controverses parce que, bien que ce format procure de nombreux avantages, il n'est pas parfait. Cette section a pour but de présenter ses forces et ses faiblesses afin de pouvoir expliquer pourquoi il est à l'origine de ce qu'on appelle les services Web.

Tableau 2.2

Forces et faiblesses du XML

Forces	Faiblesses
Basé sur des standards internationaux	Il y a beaucoup de redondance due aux balises.
Supporte l'encodage Unicode qui permet d'avoir l'information disponible dans n'importe quelle langue.	Les fichiers sont beaucoup plus gros à cause de la méta-information.
Facile à lire pour les experts ou les ordinateurs.	Les fichiers XML ne sont pas aussi faciles à traiter que les fichiers textes « plats ».
La méta-information décrit l'information dans la structure.	

2.1.4 Les espaces de nom

« Un **espace de nom** est un ensemble de ce qui est désignable dans un contexte donné par une méthode d'accès donnée faisant usage de noms symboliques (par exemple des chaînes de caractères avec ou sans restriction d'écriture »[1]

En d'autres mots, les espaces de noms servent à délimiter le champ d'action du vocabulaire XML. Ceci donne la capacité au XML de mélanger sans ambiguïtés plusieurs ontologies.

---

[1] Définition provenant de Wikipedia au URL suivant: http://fr.wikipedia.org/wiki/Espace_de_noms. Consulté le 30/05/2007.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ns1:Etudiant niveau="maitrise" programme="ele">
3  <ns1:Nom>Lemay</ns1:Nom>
4  <ns1:Prenom>Mathieu</ns1:Prenom>
5  </ns1:Etudiant>
6  <ns2:Etudiant niveau="maitrise" programme="ele">
7  <ns2:Nom>Lemay</ns2:Nom>
8  <ns2:Prenom>Mathieu</ns2:Prenom>
9  </ns2:Etudiant>
```

Figure 2.2 ***Listage d'un document XML avec espaces de nom.***

Dans l'exemple à la Figure 2.2, on peut voir deux espaces de nom : l'espace *ns1* et l'espace *ns2*. Ainsi, dans ce document il n'y a aucun élément qui est dupliqué, les éléments *ns1 :Etudiant* et *ns2 :Etudiant* sont des éléments aussi différent que *ns1 :Etudiant* et *ns1 :Nom*. Ainsi différentes applications peuvent avoir le même vocabulaire.

2.1.5 Les schémas XML

Les schémas XML sont utilisés pour définir une « classe » de documents ; c'est pourquoi lorsqu'on parle d'une « instance de document », celle-ci réfère à un document qui est conforme à un certain schéma. Le terme schéma a été repris des bases de données et de leur représentation de l'information, c'est pourquoi le schéma XML détermine la structure que doit posséder le document XML afin d'être valide.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <!--W3C Schema generated by XMLSpy v2008 sp2 U (http://www.altova.com)-->
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
4    <xs:element name="Etudiant">
5      <xs:complexType>
6        <xs:sequence>
7          <xs:element name="Nom" type="xs:string"/>
8          <xs:element name="Prenom" type="xs:string"/>
9        </xs:sequence>
10       <xs:attribute name="niveau" type="xs:string" use="required"/>
11       <xs:attribute name="programme" type="xs:string" use="required"/>
12     </xs:complexType>
13   </xs:element>
14 </xs:schema>
```

Figure 2.3 ***Listage d'un Schéma XML.***

Dans le listage de la Figure 2.3, on voit le schéma XML qui permet de décrire le document retrouvé à la Figure 2.1. On peut aussitôt remarquer que le schéma XML est beaucoup plus long que le document qu'il décrit. Il permet de valider la structure ainsi que les données. Pour qu'un document soit valide selon ce schéma, il faut que toutes les règles soient respectées. Les schémas sont primordiaux pour les services Web puisqu'ils servent à décrire l'information traitée par les interfaces de communication acceptées.

2.2 Les services Web et les standards utilisés.

Après avoir vu comment il est possible d'utiliser le XML et les schémas XML afin de décrire des structures de données, il est maintenant possible d'expliquer ce que sont les services Web et comment ils fonctionnent.

2.2.1 Introduction aux services Web

Les services Web permettent à deux applications de communiquer entre elles grâces à des protocoles standards et universels. Ils peuvent aussi être découverts et introspectés puisqu'ils ont la capacité de s'auto-décrire. Il y a plusieurs façons de créer des services Web mais les trois plus populaires sont : le *XML Remote Procedure Call* (XML-RPC), le Representational State Transfer (REST) et l'utilisation du SOAP. Ce mémoire se concentra plus particulièrement sur le SOAP puisque cette spécification est celle utilisée dans les grids.

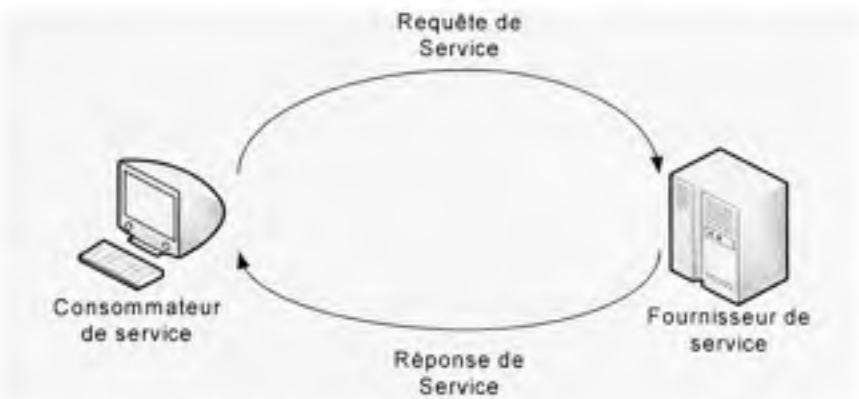

Figure 2.4 ***Échange de base pour les services Web.***

On peut voir sur la Figure 2.4 que les services Web ne sont pas très différents des requêtes client-serveur existantes. Le différentiateur est le fait que les services Web peuvent être utilisés peu importe le langage de programmation, la plateforme ou le réseau. La seule obligation est que le protocole HTTP (HyperText Transfer Protocol) doit être implémenté par la plateforme voulant utiliser un service Web. Ceci permet donc à des dispositifs mobiles tels les téléphones cellulaires dépourvus de réseau IP mais dotés de réseaux CDMA X1 de pouvoir accéder de la même façon à ces services Web. De plus, les pare-feux (firewalls) sont très contraignants dans plusieurs organisations et ne laissent passer que les pages Web (http/80), le courrier (smtp/25) et l'accès console sécuritaire (ssh/22).

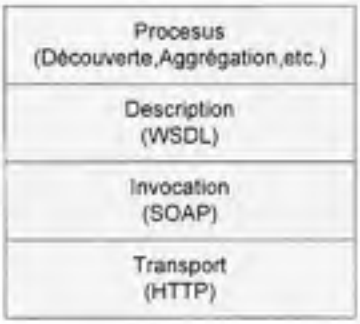

Figure 2.5 ***Pile des différents protocoles des services Web.***

La Figure 2.5 montre les différents protocoles nécessaires aux services Web; la dépendance va du haut vers le bas. La section « processus » correspond au différents WS-* : on appelle WS-* la famille de tous les différents standards de services Web. Des exemples comme WSSecurity, WS-Choreography, WS-Transactions etc. essaient de standardiser différentes opérations logicielles. Les standards étudiés plus profondément à la fin de ce chapitre seront les standards grid qui font partie de la famille WSRF.

2.2.2 Le protocole SOAP et les échanges client-serveur

Le protocole SOAP est souvent automatiquement associé aux services Web; il est utilisé pour échanger des messages XML entre différents ordinateurs. Il encapsule le document XML

dans une enveloppe qui sert à l'adressage, aux paramètres spéciaux et à la sécurité. L'avantage du SOAP est que le document suit une spécification selon une définition de schéma. Donc les éléments envoyés peuvent être complexes tandis que les protocoles REST or XML-RPC peuvent uniquement échanger des structures simples comme des chaines de caractères ou des nombres. Le listage de la Figure 2.6 montre une requête qui permettrait de pouvoir recevoir les informations concernant les étudiants définis dans le listage de la Figure 2.1.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
3    <soap:Body>
4      <getEtudiants xmlns="http://warehouse.example.com/ws">
5      </getEtudiants>
6    </soap:Body>
7  </soap:Envelope>
```

Figure 2.6 ***Listage d'une requête SOAP.***

À cette requête client, le serveur SOAP répondra par un document qui aura la section XML correspondant à la structure d'un étudiant tel que montre la Figure 2.7.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
3    <soap:Body>
4      <getEtudiantsResponse xmlns="http://warehouse.example.com/ws">
5        <getProductDetailsResult>
6          <ns1:Etudiant niveau="maitrise" programme="ele">
7            <ns1:Nom>Lemay</ns1:Nom>
8            <ns1:Prenom>Mathieu</ns1:Prenom>
9          </ns1:Etudiant>
10       </getProductDetailsResult>
11     </getEtudiantsResponse>
12   </soap:Body>
13 </soap:Envelope>
```

Figure 2.7 ***Listage d'une réponse SOAP.***

Mais comment faire pour prendre un objet d'un langage de programmation tel que Java, C++ ou bien Python et le transformer automatiquement en XML et publier les différentes opérations pouvant être effectuées ? La solution est dans ce qu'on appelle l'encodage ; il y a différents types d'encodage pour le SOAP, qu'on peut voir dans le
Tableau **2.3**.

Tableau 2.3

Types d'encodage SOAP

Encodage	Description
RPC	L'encodage RPC fait la sérialisation automatique des différents éléments XML vers les Objets du langage de programmation tout en ayant une approche synchrone (requête et réponse) envers l'opération.
RPC-Literal	L'encodage RPC-Literal ne fait pas la sérialisation automatique mais garde une approche synchrone pour l'opération.
Document-Literal	L'encodage Document-Literal est basé sur des messages asynchrones qu'il faut manipuler manuellement.

On peut voir sur la Figure 2.8 les lignes directrices lors du choix du type d'encodage à utiliser. En effet, si l'application s'occupe de la sérialisation (transformation d'objet à XML et vice-versa) des objets, elle est beaucoup plus facile à manipuler mais beaucoup moins performante que si tout le traitement XML est fait directement par le programmeur. Ainsi le RPC est souvent utilisé par les outils qui permettent de créer rapidement des services Web à partir d'applications existantes sans avoir besoin de l'intervention du programmeur tandis que le Document Literal nécessite un effort proportionnel à la taille de l'information échangée afin de pouvoir faire un traitement adéquat de cette information.

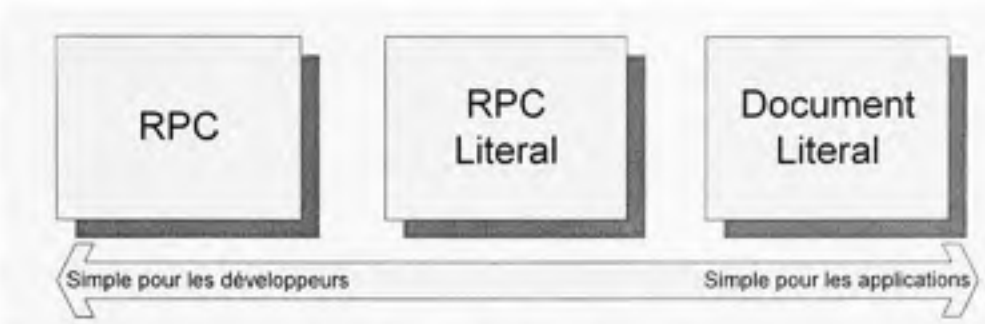

Figure 2.8 ***Choix de l'encodage.***

Il est important de noter qu'il faut idéalement faire un choix d'encodage judicieux selon l'application créée; mais les développeurs de services se soucient rarement de ce paramètre et optent habituellement pour la méthode la plus simple.

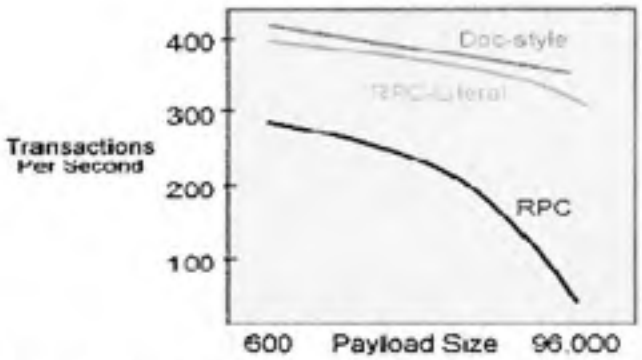

Figure 2.9 ***Analyse de performance des différents encodages.***

Source : L'image est tirée de l'article de Frank Cohen, Discover SOAP encoding's impact on Web service performance publié en 2003 sur le site d'IBM. Le titre original de la figure est « SOAP RPC-literal provides the performance benefits of SOAP document-style encoding with a little more work required to parse through the XML data ». URL : http://www-128.ibm.com/developerworks/webservices/library/ws-soapenc/ Consulté le 30/05/2007.

On peut voir sur la Figure 2.9 que le nombre de transactions par seconde du Document Literal dépasse grandement l'équivalent RPC surtout lorsque le document est gros. À 600 octets, on voit déjà une grande différence de performance puisque RPC-Literal et Document-Literal sont 33% plus rapides; mais c'est à 96 000 octets qu'on voit qu'ils sont 35 fois plus

rapides. C'est pourquoi c'est le Document-Literal qui doit être utilisé dans les Grids puisque les performances constituent un facteur très important.

2.2.3 Description des interfaces à partir de fichiers WSDL

Les Web Service Description Language (WSDL) servent à décrire le service et comment il est possible d'obtenir l'information de ce service. Cette section servira à décrire ce qu'est un WSDL et comment il fonctionne. Premièrement, un WSDL est un document XML qui permet de décrire comment les services sont situés et exécutés. Le consommateur de service de la Figure 2.4 doit premièrement trouver un document WSDL qui décrit de façon détaillée comment exécuter le service. Le fichier WSDL est téléchargé dynamiquement ou bien situé localement sur l'ordinateur et décrit au client Web comment construire sa requête SOAP à envoyer au fournisseur de service. Tel que vu dans la section 2.2.2, le service répondra avec la réponse SOAP appropriée.

Figure 2.10 ***Téléchargement d'un fichier WSDL.***

Mais comment ce document est-il construit? Pour pouvoir décrire adéquatement un service, un document WSDL doit inclure un certain nombre d'éléments importants. Ces éléments seront analysés à partir de ce fichier WSDL simple qui décrit un service qui fait l'addition de deux nombres.

```
11    <wsdl:types>
12      <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
13            xmlns="http://jws.samples.geronimo.apache.org"
14            targetNamespace="http://jws.samples.geronimo.apache.org"
15            attributeFormDefault="unqualified" elementFormDefault="qualified">
16
17        <xsd:element name="add">
18          <xsd:complexType>
19            <xsd:sequence>
20              <xsd:element name="value1" type="xsd:int" />
21              <xsd:element name="value2" type="xsd:int" />
22            </xsd:sequence>
23          </xsd:complexType>
24        </xsd:element>
25
26        <xsd:element name="addResponse">
27          <xsd:complexType>
28            <xsd:sequence>
29              <xsd:element name="return" type="xsd:int" />
30            </xsd:sequence>
31          </xsd:complexType>
32        </xsd:element>
33      </xsd:schema>
34    </wsdl:types>
```

Figure 2.11 ***Listage d'un fichier WSDL (Types et Messages).***

Les éléments les plus importants sont les <types> et les <messages>. Ces éléments décrivent l'information d'entrée et sortie au service sous forme de schémas XML. C'est cette section qui est la plus longue à écrire et à traiter.

```
51    <wsdl:binding name="CalculatorSoapBinding" type="tns:CalculatorPortType">
52      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
53
54      <wsdl:operation name="add">
55        <soap:operation soapAction="add" style="document" />
56        <wsdl:input name="add">
57          <soap:body use="literal" />
58        </wsdl:input>
59        <wsdl:output name="addResponse">
60          <soap:body use="literal" />
61        </wsdl:output>
62      </wsdl:operation>
63
64    </wsdl:binding>
```

Figure 2.12 ***Listage d'un fichier WSDL (Types et Messages).***

Une fois les différentes opérations présentées, il faut décrire comment accéder à ces opérations via le protocole SOAP. C'est ici que les notions d'encodage vues au préalable

sont importantes puisqu'elles permettent de décrire la façon de communiquer avec ce service. On peut voir que, pour le service de calculatrice montré à la Figure 2.12, c'est l'encodage de type Document-Literal qui est utilisé puisqu'à la ligne 55 on voit le style à « document » et non « rpc » et qu'à la ligne 57, l'utilisation du « literal » pour le corps du document SOAP est requise.

```
66  <wsdl:service name="Calculator">
67    <wsdl:port name="CalculatorPort" binding="tns:CalculatorSoapBinding">
68      <soap:address location="http://localhost:8080/jaxws-calculator-1.0/calculator" />
69      <wswa:UsingAddressing xmlns:wswa="http://www.w3.org/2005/08/addressing/wsdl" />
70    </wsdl:port>
71  </wsdl:service>
```

Figure 2.13 ***Listage d'un fichier WSDL (Types et Messages).***

Finalement, le service prend forme lorsqu'un <portType> est associé à un <binding> et que l'adresse de ce service est spécifiée explicitement dans un élément <Service>.

2.3 Des services Web aux ressources grâce au Web Service Resource Framework (WSRF)

Les services grid sont depuis GT4 des services Web standardisés qui définissent une ressource. Il y a deux volets importants aux services grid : ils doivent avoir des interfaces standards et assurer un niveau de sécurité. Les interfaces permettent à des applications d'acquérir l'information de la ressource sans devoir connaître le structure de données de cette ressource au préalable. Une application peut donc par la suite déduire le type de la ressource et agir en conséquence. Un exemple simple pour l'instrumentation est que les capteurs de température ou de pression peuvent être lus par n'importe quel autre service grid ou application sans que celle-ci ait besoin de savoir ce que sont température ou pression. Par la suite, seuls les nœuds d'intérêts dans le document de ressources seront évalués. Plusieurs écoles de pensées existent sur les bonnes pratiques à utiliser afin de garder l'état d'un service et c'est entre autre ce que le standard WSRF vise à standardiser. Afin de bien comprendre le WSRF il est important de bien voir la différence entre un service Web et un service Grid communément appelé ressource.

Le service Web a une adresse de style http://serveur/services/porttype. Cette adresse, appelée Uniform Resource Locator (URL), détermine où échanger les messages SOAP pour interagir avec le service. Il ne fait aucune distinction entre les différents appels et procure habituellement des fonctions sans état. Un exemple de service Web serait un service de conversion de fichier : il prend comme entrée le fichier original, le convertit, puis retourne le fichier dans le nouveau format, en procurant cette fonctionnalité à plusieurs utilisateurs simultanément.

Le service grid est une sorte de service Web qui permet d'interagir avec une ressource. Il existe donc un service commun à des ressources multiples. La ressource représente un document d'état quelconque.

Un service Web peut lui aussi garder l'état : on peut voir sur la Figure 2.14 la façon dont un service Web ordinaire de calculatrice gardera cet état. Cet exemple met en évidence les

problèmes qui sont pris en compte par les services grid. En (1), le premier appel fait par l'utilisateur 1 à cet URL dit au service d'ajouter 2 à la valeur actuelle d'état, 0, pour un total de 2. En (2), un deuxième utilisateur veut aussi utiliser le service et lui demande d'ajouter 1 à la valeur actuelle. Par contre en (3), au lieu de lire la valeur 2 tel que prévu, l'utilisateur recevra une valeur calculée de 0+2=3 puisque l'appel fait par l'utilisateur 2 a changé l'état de ce même service. Ainsi, il est impossible d'avoir sa propre calculatrice dédiée grâce aux services Web actuels puisque, lorsqu'on interagit avec le service, il n'a aucune idée des opérations précédentes faites par un utilisateur particulier.

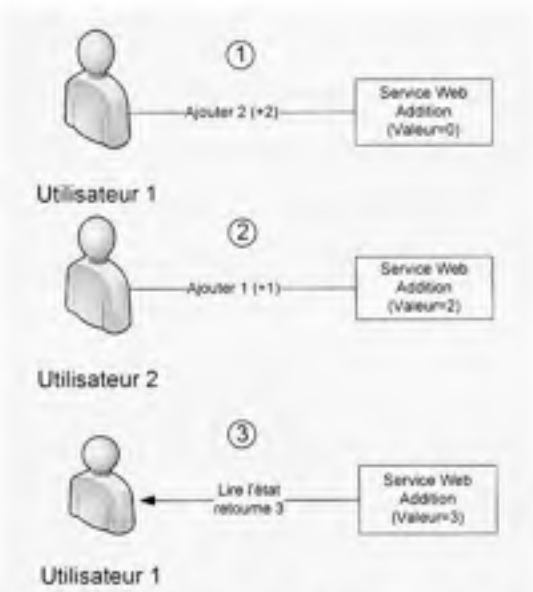

Figure 2.14 ***Les services Web et leur état.***

Pour remédier à cette situation, il faudrait idéalement qu'on puisse « créer » deux calculatrices, une pour chaque utilisateur qui aurait uniquement l'état correspondant à l'ensemble de leurs interactions avec le service. C'est ainsi que le concept de « ressource » est venu répondre à ce besoin. Elle permet d'avoir un service grid commun, qui procure l'instance d'une ressource à un ou plusieurs utilisateurs. Les prochaines sections présentent les standards et technologies qui rendent possible la création de ces ressources.

2.3.1 Introduction aux services grids et aux ressources

Le « Globus Toolkit 4 » s'est attaqué à la tâche de fournir cette notion de ressource avec état. Les ressources étaient le point commun de la spécification OGSI mise de l'avant par l'OGSA. Cette spécification, mise en œuvre dans l'implémentation du GT3, avait déjà la notion de service Web avec état. Par contre, la façon d'accéder au service était non-standardisée et rajoutait des « extensions » spécifiques au Web Service Description Language (WSDL). Ainsi les organismes de standardisation OASIS et W3C ont rejeté les modifications apportées aux WSDL puisqu'elles étaient spécifiques aux besoins des grids. Après coup, Globus a décidé de travailler avec les standards émergeant comme le WS-Addressing qui permettent l'adressage implicite, c'est-à-dire qu'une référence à la ressource concernée fait partie du message de contrôle d'une ressource grâce à un *Endpoint Reference*. Le document XML *Endpoint Reference* de la Figure 2.15, permet d'ajouter des paramètres à l'entête du message SOAP afin de permettre au service de connaître le contexte de ce message.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsa:EndpointReference>
3    <wsa:Address>http://serveur/services/porttype</wsa:Address>
4    <wsa:ReferenceParameters><Parametre1>Exemple de paramètre de la reasource</Parametre1> </wsa:ReferenceParameters>
5    <wsa:InterfaceName>xs:None de l'interface</wsa:InterfaceName>
6    <wsa:ServiceName EndpointName="xs:NCName"?>xs:NomduService</wsa:ServiceName>
7    <wsa:Policies>Pas utilisé par GT4</wsa:Policies
8    <xs:any/>*
9  </wsa:EndpointReference>
```

Figure 2.15 ***Listage d'une Endpoint Reference.***

Puisque Globus a cette fois-ci épousé les standards Web émergeant, le Web Service Resource Framework (WSRF) a pu être soumis avec succès à OASIS. Par contre, l'équipe de Globus n'a pas attendu que le standard soit finalisé avant de procéder à l'implémentation et le GT4 est le résultat de la première implémentation du WSRF. Le WSRF répond lui aussi aux besoins de l'OGSA en procurant une extension naturelle aux services comme on peut voir à la Figure 2.16. Le WSRF consiste en un ensemble de différents standards qui rendent la création de ces ressources possible. Le standard déterminant du WSRF est le WS-ResourceProperties. Ce standard décrit les opérations à utiliser pour interagir avec une

ressource. Il y a aussi l'ensemble de standards que comporte le Web Service Notifications (WS-N) qui définit les interfaces de notification, le standard WS-BaseFaults qui détermine comment traiter les erreurs de façon la plus uniforme possible et finalement le standard WS-ResourceLifetime qui permet la gestion de la destruction des ressources de façon immédiate ou bien planifiée.

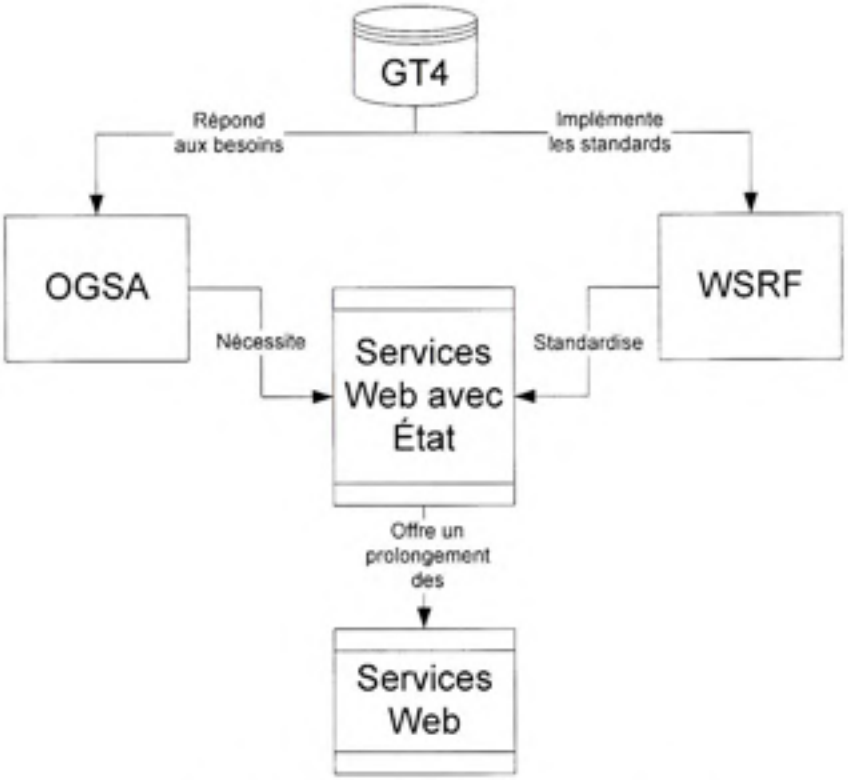

Figure 2.16 ***Diagramme de l'évolution du GT4.***

Sur la Figure 2.17, on peut voir la représentation exacte d'une ressource. La ressource a un état qui peut être volatile ou bien stocké dans une base de données. Cet état est extériorisé grâce aux différents services qui font partie du WSRF.

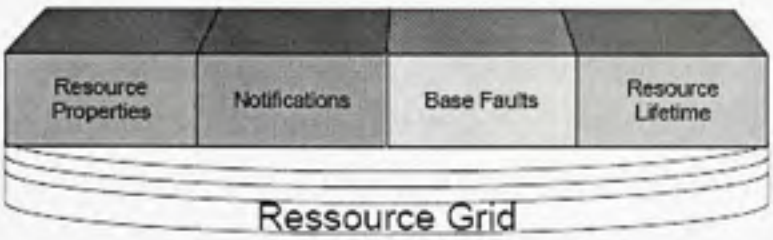

Figure 2.17 ***Ressource Grid et les standards qui la composent.***

Dans les sections suivantes, les différents standards seront analysés séparément afin de présenter comment interagir correctement avec des ressources grid.

2.3.2 Standard WS-ResourceProperties d'interaction avec l'état de la ressource

Le WS-ResourceProperties est le standard de base qui définit une ressource et permet d'interagir avec n'importe quelle autre ressource. Puisque les données sont échangées sous le type *xsd:any,* il est possible de lire les propriétés de la ressource sans connaître le type réel de cette donnée. Lorsqu'on s'adresse à une ressource grâce au WS-Addressing, on peut lire ou écrire ce qu'on appelle une « Resource Property ». La « Resource Property » correspond à un élément d'un document XML représentant l'état de cette ressource.

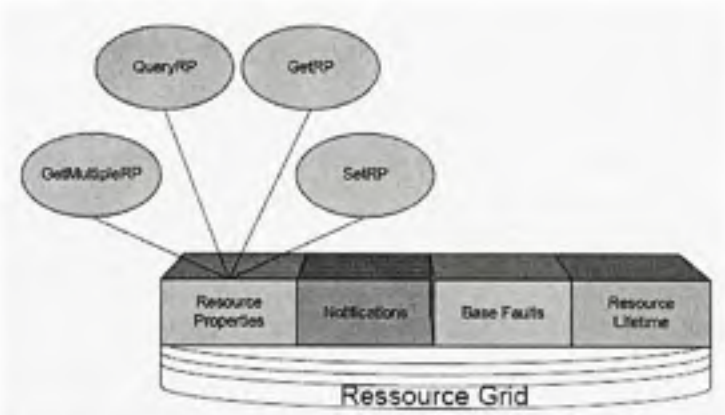

Figure 2.18 ***Les opérations du standard WS-ResourceProperties.***

On peut voir sur la Figure 2.18 que les opérations de base du WS-ResourceProperties sont : GetMultipleRP, GetRP, SetRP et QueryRP. L'opération GetMultipleRP permet de lire plusieurs propriétés à la fois. Il est même possible de lire l'état complet de la ressource grâce à cette opération. Il existe aussi une différence notoire entre QueryRP et GetRP de par la façon d'effectuer la requête. Lorsqu'on utilise QueryRP, on doit donner un chemin de recherche selon le format XPath. Le format XPath tente de replacer le Structured Query Language (SQL) qui est inefficace pour les documents XML à cause de leur structure arborescente.

```
58      <InputMessage Description="Logout Command for the Switch" Name="Logout" TermnationCode="">
59          <Verb>CANC</Verb>
60          <Modifier1>USER</Modifier1>
61          <TID Quoted="false"> <Variable>SwitchID</Variable></TID>
62          <AID Quoted="false"> <Variable>Security/Username</Variable></AID>
63          <CTAG AutoGenerate="false">1</CTAG>
64      </InputMessage>
```

Figure 2.19 ***Listage d'un document XML pour exemple XPath.***

Le chemin XPath est déterminé par la syntaxe suivante : ***/élément parent/élément enfant[@attribut='valeur']***. Grâce à cette syntaxe, il est possible de récupérer uniquement les éléments du document XML qui sont d'intérêt. Par exemple, pour le listage de la Figure 2.19, il serait possible d'aller récupérer la valeur « SwitchID » en utilisant la syntaxe suivante : ***/InputMessage[@Name='Logout']/ TID/Variable***. Les opérations GetRP et SetRP permettent de lire ou d'écrire une seule propriété de la ressource à la fois. Il est aussi important d'ajouter que SetRP peut permettre d'ajouter des propriétés qui n'existaient pas dans l'état initial de la ressource. Il est important de pouvoir interagir de façon correcte avec les propriétés de la ressource, mais que faire lorsqu'une de ces propriétés change ? Doit-on continuellement lire les ressources grâce à l'opération GetRP ? Le prochain standard WS-Notifications prend en compte ce problème.

2.3.3 Standard WS-Notifications et les sujets d'intérêts (WS-Topics).

Le WS-Notifications comprend une famille de standards qui a été « séparée » du WSRF par OASIS sous le nom WSN afin qu'il puisse être aussi utilisé sans nécessiter la notion de « ressource » par n'importe quel service Web. Il y a deux façons différentes de faire des notifications qui sont régies par deux standards différents, le WS-BaseNotifications et le WS-BrokeredNotifications.

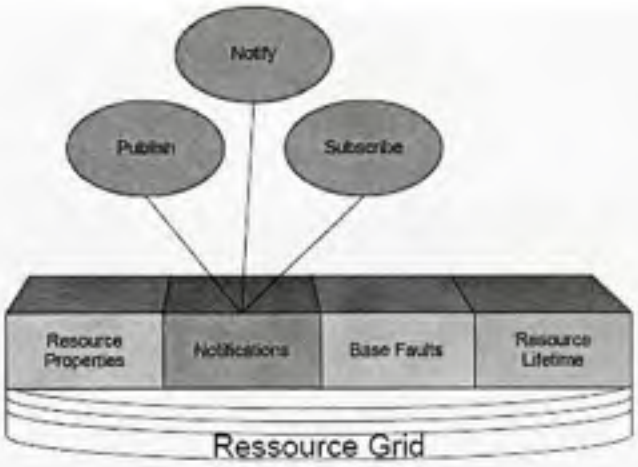

Figure 2.20 ***Les opérations du standard WS-BaseNotifications.***

Le GT4 permet par contre uniquement de faire les WS-BaseNotifications puisque le standard WS-BrokeredNotifications n'est pas implémenté. Les opérations du WS-BaseNotifications telles que vues sur la figure 2.20, permettent d'avoir un patron observateur de type « publish-subscribe ». Une ressource qui désire émettre des notifications doit faire l'annonce des sujets d'intérêts (WS-Topics) grâce à l'opération Publish. Les ressources qui désirent être notifiées lorsqu'il y a un changement au sujet d'intérêt (WS-Topic) doivent s'inscrire (Subscribe) à la ressource émettrice. Elles doivent aussi avoir une opération appelée Notify qui sera utilisée par les ressources émettrices afin de notifier les observateurs lors d'un changement d'un sujet d'intérêt.

2.3.4 Standard WS-BaseFaults et exceptions Java

Contrairement aux autres standards du WSRF, le standard WS-BaseFaults ne décrit pas d'opération mais une façon uniforme de retourner un code d'erreur à n'importe quelle application client ayant invoqué les autres services de la ressource. En effet, un des gros problèmes des services Web est qu'il n'existe pas de standard qui définit les messages d'erreur.

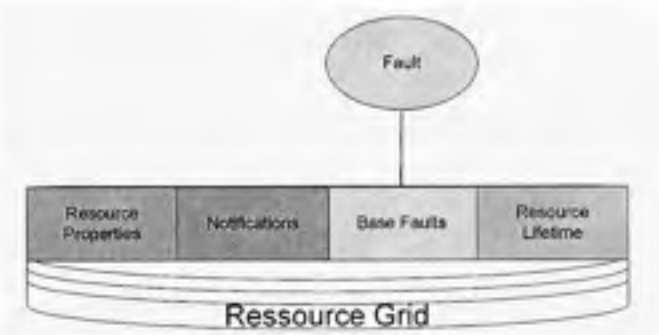

Figure 2.21 ***Le standard WS-BaseFaults.***

Puisque les services sont crées par plusieurs développeurs, la gestion des différentes exceptions qui se produisent est difficile lorsque chaque développeur a sa propre façon de représenter les messages d'erreur. On peut voir sur la figure 2.21 que le standard WS-BaseFaults décrit ce format de message d'erreur qui peut être analysé par tous les clients d'une ressource Grid. Il est donc beaucoup plus facile de faire le traitement des messages d'erreurs grâce à cette interface standard. De plus, elle est facilement « extensible » pour avoir de l'information plus spécifique à la ressource concernée.

2.3.5 Standard WS-ResourceLifetime

La durée de vie est un autre aspect important des ressources Grid. Le standard WS-ResourceLifetime procure deux façons différentes de contrôler la durée de vie de ces

ressources. La première façon s'appelle *ImmediateResourceTermination* et ajoute une opération *Destroy* de destruction à la ressource; lorsqu'on appelle cette opération via le service, la ressource est détruite ainsi que toute l'information d'état qu'elle contenait.

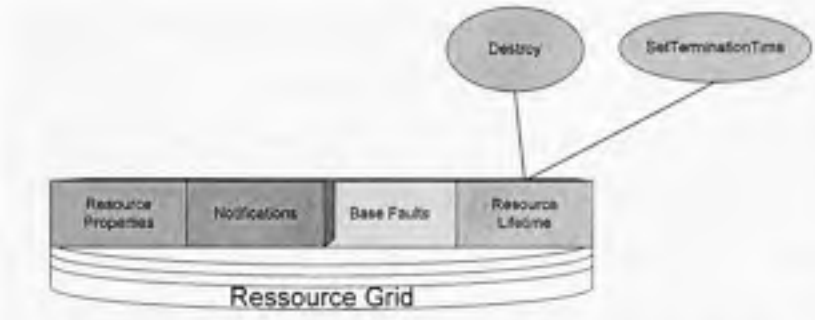

Figure 2.22 ***Les opérations du WS-ResourceLifetime.***

De plus, comme la Figure 2.22 le montre, il est aussi possible de minuter le temps de vie de la ressource grâce au *ScheduledResourceTermination* et à l'opération *SetTerminationTime.* Ceci peut être utilisé afin de détruire la ressource à une date particulière, mais en général on utilise cette technique pour faire du *Lease Management.* Ceci consiste à reporter continuellement la date de destruction afin que la ressource soit détruite si la communication avec un autre service devient impossible.

2.4 Sécurité dans les grids grâce au Globus Toolkit

Il est possible de penser qu'un système sécuritaire ne consiste qu'à faire le chiffrement de l'information. Par exemple, lorsqu'on accède à notre compte bancaire en ligne, les données sont chiffrées grâce au protocole SSL. Toutefois, la sécurité Grid englobe beaucoup plus que le chiffrement de l'information car une Grid est un environnement collaboratif où des politiques strictes doivent être appliquées différemment pour chaque utilisateur.

Il existe trois piliers fondamentaux pour une communication sécuritaire. Ces piliers sont le droit au domaine privé, l'intégrité des données et l'authentification. Idéalement, il faut avoir ces trois piliers dans tout système sécurisé, mais c'est rarement le cas.

Le droit au domaine privé consiste à faire en sorte qu'uniquement les parties qui s'échangent des données soient capables de comprendre l'information transmise. Ainsi, si une tierce partie essaie de faire l'écoute électronique de cette information, elle ne pourra percevoir aucune information intelligible. Ceci est généralement établi grâce aux algorithmes de chiffrement/déchiffrement. Un exemple simple de chiffrement : pour envoyer le message « JE SUIS MATHIEU LEMAY », on pourrait substituer chaque lettre par la prochaine de l'alphabet et considérer qu'A vient après l'espace. On obtiendrait ainsi : « KFATVJTANBUIJFVAMFNBZ ». Un observateur externe ne connaissant pas le code serait incapable de reconstituer le message, alors que les parties intéressées auraient échangé adéquatement le code au préalable.

L'intégrité des données assure que le message reçu est le même que celui qui a été envoyé. Ceci prévient qu'une tierce personne ne vienne changer les données du message de façon malicieuse. Il est évidemment nécessaire d'avoir brisé la sécurité du domaine privé afin de pouvoir atteindre à l'intégrité du message. Toutefois, il y a des conversations publiques où il n'y a pas de domaine privé (chiffrement) car l'information n'est pas confidentielle. Donc, dans certains cas, il suffit d'assurer l'intégrité du message lorsque le contenu ne dois pas être modifié mais peut rester lisible pour les observateurs externes. Afin d'assurer l'intégrité des messages, on a souvent recours à une méthode qui s'appelle le « hachage ». Elle consiste à calculer un nombre qui est unique au contenu du message. Il existe plusieurs algorithmes de hachage les plus populaires sont le « division remainer », le « folding » le « radix transform » et le « digit rearrangement ».

Le *division remainer* consiste à faire un tableau à partir du message. Ensuite, on fait l'addition des valeurs du tableau et on divise chaque valeur du tableau par cette valeur afin d'obtenir un quotient et un reste. Le reste est ce qui est utilisé comme clé de hachage. Le

folding consiste à couper les chaines en différentes parties et à en faire l'addition, ensuite les 4 derniers chiffres sont utilisés comme clé du hachage. Le *radix transform* consiste à changer la base du chiffre ainsi une chaine de caractères ayant une base hexadécimale pourrait être transformée en base64 en regroupant 4 caractères (octets) à la fois. Finalement, le *digit rearrangement* consiste simplement à remplacer les chiffres d'une position déterminée par une autre, soit par exemple changer le bit 4 pour le bit 6 de chaque octet. Il est important qu'une opération de hachage soit unidirectionnelle. Il est impossible de récupérer le message original à partir de la clé de hachage (*hash*) mais le hachage doit être reproductible puisque la clé de hachage transmise est comparée au hachage calculé à la réception du message afin d'en assurer l'intégrité.

Le dernier pilier de la sécurité est l'authentification. Elle consiste à s'assurer que les utilisateurs impliqués dans la conversation sont bien ceux qu'ils prétendent être. Ceci est important afin d'empêcher un individu ou service malicieux d'usurper l'identité d'un service. On confond parfois l'authentification à l'autorisation. L'autorisation consiste à vérifier si l'utilisateur authentifié a la permission ou non d'effectuer une action quelconque. Il est nécessaire de bien authentifier l'utilisateur avant de l'autoriser à quoi que ce soit.

2.4.1 Introduction à la cryptographie.

Comme on l'a vu dans la section précédente, on utilise la cryptographie afin de créer un domaine privé pour les membres d'une conversation sécuritaire. Par contre, l'art de la cryptographie a toujours fasciné les mathématiciens. Les algorithmes modernes vont même jusqu'à permettre d'avoir les trois piliers d'une conversation sécuritaire dans un seul système de sécurité. Il existe plusieurs façons de faire du cryptage; on a vu dans la section précédente qu'on pouvait par exemple substituer chaque lettre par une autre. Ainsi les données originales sont modifiées par un algorithme afin de les rendre inintelligibles. Ces algorithmes sont toutefois faciles à briser puisqu'il suffit de ne connaître qu'un peu d'information sur le message original (comme le nom Mathieu Lemay) et d'essayer de le récupérer grâce à des méthodes heuristiques. C'est pourquoi le type de cryptographie le plus utilisé est ce qu'on

appelle la cryptographie à clé de chiffrement. Ceci permet de chiffrer le message, tel que montré sur la Figure 2.23, avec une donnée externe appelée clé de chiffrement qui ne fait pas partie des données du message à transmettre. Il devient donc très improbable de pouvoir casser l'algorithme et d'obtenir le message transmis sans avoir la clé.

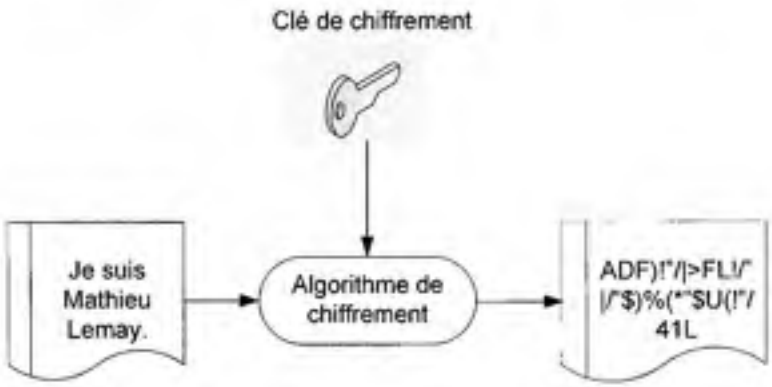

Figure 2.23 ***Chiffrement avec clé de chiffrement.***

Mais comment fait-on pour coder le message ? Si on prend par exemple un message simple : « 1 2 3 4 5 » et une clé aléatoire « 2 3 1 5 3 », si on fait l'addition du message et de la clé on obtient : « 3 5 4 9 8 ». On transmet donc ce message, la personne qui reçoit le message doit connaître la clé « 2 3 1 5 3 » et peut restituer le message original en faisant la soustraction de cette clé du message « 3 5 4 9 8 », ce qui redonnera bel et bien le message d'origine soit : « 1 2 3 4 5 ». Cet exemple présente un très simple algorithme (addition/soustraction) et les algorithmes utilisés maintenant sont beaucoup plus efficaces, mais le concept de base reste le même. La majorité des algorithmes nécessitent aussi de garder cette clé secrète entre les utilisateurs impliqués dans la conversation. Ainsi, cette clé est souvent transmise sur un canal de transmission différent de celui utilisé pour le trafic des données. Par exemple, si on utilise l'Internet pour transmettre les données, une façon sécuritaire serait de transmettre la clé par téléphone ou par fax à la personne avec laquelle on veut communiquer.

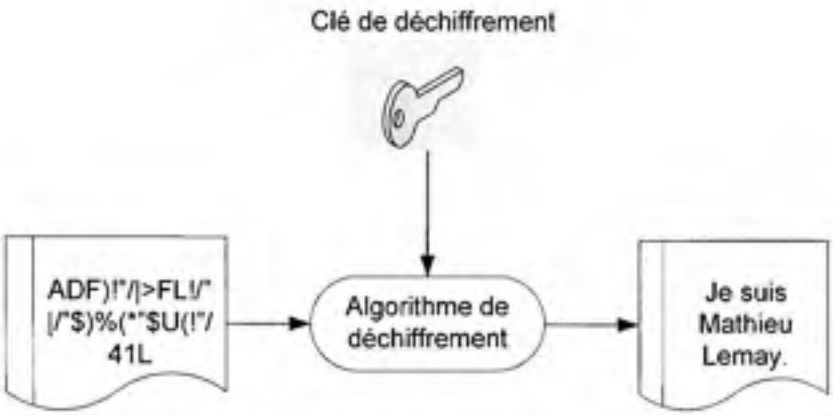

Figure 2.24 Déchiffrement *avec clé de déchiffrement.*

On peut voir sur la Figure 2.24 qu'il faut une clé de déchiffrement pour décrypter le message. Dans l'exemple précédent les clés de chiffrement et de déchiffrement étaient la même. On qualifie de symétrique ce type d'encryptions. Puisque l'échange de clé est parfois difficile à faire sur un canal séparé et surtout lorsqu'on veut sécuriser de façon rapide les données Internet, on a recours à des algorithmes asymétriques. Un algorithme est asymétrique lorsqu'on utilise une clé pour le chiffrement et une autre clé pour le déchiffrement. Le type d'algorithme le plus utilisé présentement s'appelle *Public Key* et est présenté dans la section suivante.

2.4.2 La cryptographie à clé publique

Comme nous l'avons vu dans la section précédente, la cryptographie à clé publique utilise un algorithme asymétrique à deux clés et non une seule. On a une clé qui n'est jamais transmise et qui doit être protégée qu'on appelle la *Private Key* et il y a aussi une clé qu'on distribue à tout le monde et qu'on appelle la *Public Key*. Lorsqu'on utilise une clé pour faire le chiffrement d'un message, l'autre doit être utilisée pour le déchiffrer. Ainsi, si on chiffre un message avec la *Private Key,* la *Public Key* devra être utilisée afin d'obtenir le contenu du message.

En général, c'est plutôt l'inverse qui se passe. L'émetteur chiffre le message grâce à la clé publique du récepteur puisqu'elle est connue et à la vue de tous. On envoie le message chiffré et le récepteur le déchiffre grâce à sa clé privée que lui seul connaît.

Comme nous l'avons vu précédemment, l'avantage principal que procure ce type de chiffrement est qu'on n'a pas besoin de « négocier » l'échange d'une clé commune. Un lecteur attentif se demande peut-être si cet algorithme est sécuritaire : puisqu'on connaît l'algorithme et la clé publique, ne serait-ce pas possible de récupérer la clé privée ? La façon dont l'algorithme fonctionne rend ce type d'intrusion très improbable. Il est assez facile de retrouver la clé publique à partir de la clé privée mais faire l'inverse peut prendre des décennies de calculs en continu avec les ordinateurs les plus puissants à ce jour. Plus la clé a de caractères, plus ce sera long avant de pouvoir la briser. Les clés habituelles ont maintenant au moins 128 bits. Un autre avantage des systèmes à clé publiques est qu'en plus de procurer le domaine privé, ils permettent d'assurer l'intégrité et d'authentifier les utilisateurs impliqués dans la conversation.

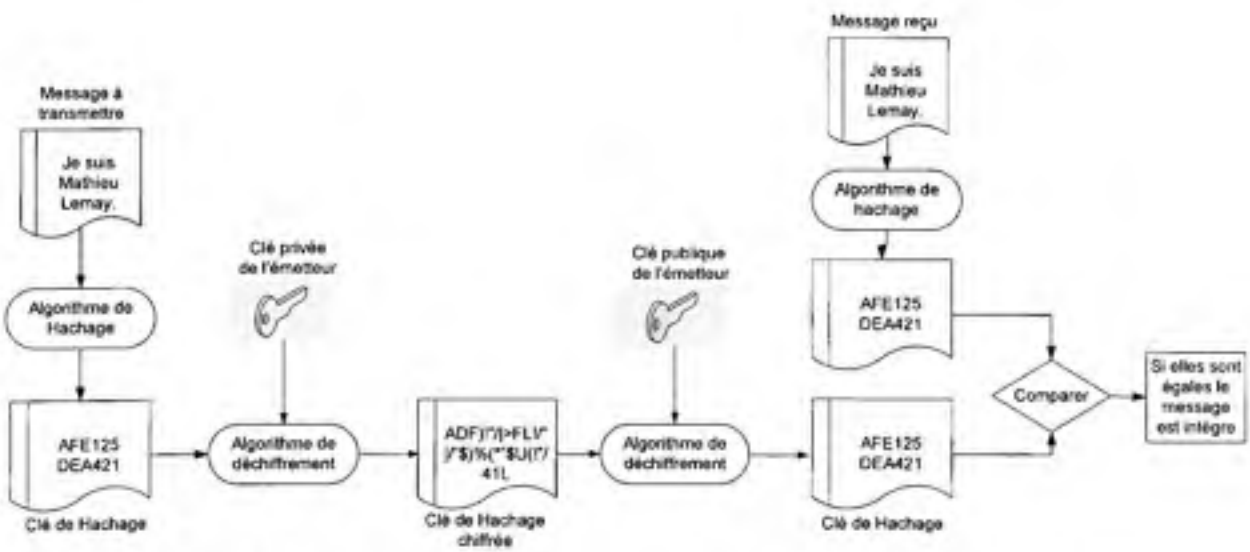

Figure 2.25 ***Assurer l'intégrité grâce à la cryptographie à clé publique.***

Sur la Figure 2.25, on peut voir comment l'intégrité est assurée grâce à la cryptographie à clé publique. Comme nous l'avons vu au début du chapitre, le hachage d'un message est fonction de son contenu et il changera si le contenu est modifié. Grâce au chiffrement à clé publique, on peut chiffrer la clé de hachage appelée *Message Digest*, et transmettre cette clé avec le message. Par contre, contrairement aux données, celle-ci est chiffrée avec la clé privée de l'émetteur et déchiffrée grâce à la clé publique. Une fois la clé de hachage déchiffrée, on la compare avec une clé de hachage générée à partir du message reçu; si les deux sont égales, l'intégrité du message a été conservée avec succès.

On a maintenant vu comment le domaine privé et l'intégrité sont bel et bien conservés par la cryptographie à clé publique et on avait préalablement mentionné que cette cryptographie procurait les trois piliers d'une conversation sécuritaire. Il reste donc à examiner comment les utilisateurs sont authentifiés. Afin de pouvoir s'authentifier lorsqu'on utilise une cryptographie à clé publique, on utilise ce qu'on appelle des certificats numériques, qui sont détaillés dans la prochaine section.

2.4.3 Les certificats et leurs autorités émettrices

Un certificat numérique est un document numérique qui certifie qu'une clé publique appartient bel et bien à un utilisateur. Ce document est « signé » par une tierce partie qu'on appelle autorité émettrice ou, en anglais, *Certificate Authority* (CA). Cette signature ne se fait pas lors de la communication sécuritaire et doit être faite séparément. Ce qu'on appelle « signature » est en fait un document numérique généré grâce à la clé privé du CA. Ainsi il est possible de vérifier l'intégrité du certificat en utilisant la clé publique du CA, tel que vu dans la section précédente.

Il y a une grande part de responsabilité humaine dans la gestion des certificats. Il faut faire « confiance » (*trust)* aux certificats émis à partir du CA. Ainsi il faut que la tierce partie, le CA soit digne de confiance pour les membres de la communication sécurisée. Il n'y a aucune automatisation pour prendre la décision d'accorder cette confiance et la décision doit être

prise manuellement. C'est pourquoi il existe une liste de CA dignes de confiance, qui a tous les certificats des CAs considérés dignes de confiance. Il y a des CA tels que VeriSign pour le commerce électronique ou GridCanada pour les grids universitaires qui sont tellement connus que leurs certificats sont parfois inclus par défaut dans les systèmes de cryptographie à clé publique.

Le format standard des certificats tel que défini par l'ITU est le X.509. Ce format contient l'information nécessaire à un bon fonctionnement des mécanismes d'authentification comme : qui est le détenteur, quelle est la durée de validité et qui est le signataire. Les champs principaux les plus importants sont donc :

- *Subject* : champ qui contient le nom distinct de l'utilisateur (*Distinguished Name*),
- *Subject's public key* : clé publique de l'utilisateur,
- *Issuer* : champ qui contient le nom distinct du CA signataire de ce certificat,
- *Digital Signature* : résultat du hachage de l'information du certificat X.509 chiffré avec la clé privée du CA.

Cette information donne l'équivalent d'un certificat tel que vu à la Figure 2.26 et les ordinateurs peuvent ainsi prendre des décisions d'autorisation adéquates vis-à-vis de l'utilisateur authentifié.

Figure 2.26 ***Exemple de certificat numérique tel que vu par l'ordinateur.***

Par contre, contrairement au nom marqué sur le certificat, les ordinateurs n'utilisent pas un nom commun comme *Certificate Authority ABC* ou bien *Mathieu Lemay* mais bien ce qu'on appelle le *Distinguished Name*. Un *Distinguished Name* est une liste de noms et de valeurs séparés par des virgules. Un exemple de nom distinct pour Mathieu Lemay serait :

O=École de technologie supérieure, OU=Département de Génie Électrique, CN=Mathieu Lemay

Dans ce nom on retrouve le champ *O*, qui signifie *Organization, OU* signifie *Organizational Unit* et finalement *CN* pour *Common Name*.

Une des caractéristiques principales des CA est qu'ils sont hiérarchisés. Ainsi, lorsqu'on ajoute un certificat dans la liste de *Trusted Certificates,* tous les certificats signés par celui-ci seront considérés fiables. Par exemple, sur la Figure 2.27, on peut voir que le certificat de *Mathieu Lemay* est signé par le CA ÉTS. Si dans ma liste de *Trusted Certificates* j'ai seulement le *Certificate Authority Suprême*, le certificat de *Mathieu Lemay* sera au départ signé par un CA inconnu. Par contre, après avoir examiné la signature du certificat du *CA ETS,* on peut voir que celui-ci est signé par le *Certificate Authority Suprême* ainsi il est digne de confiance. Parce que les CA sont hiérarchiques le certificat de *Mathieu Lemay* devient ainsi lui aussi digne de confiance.

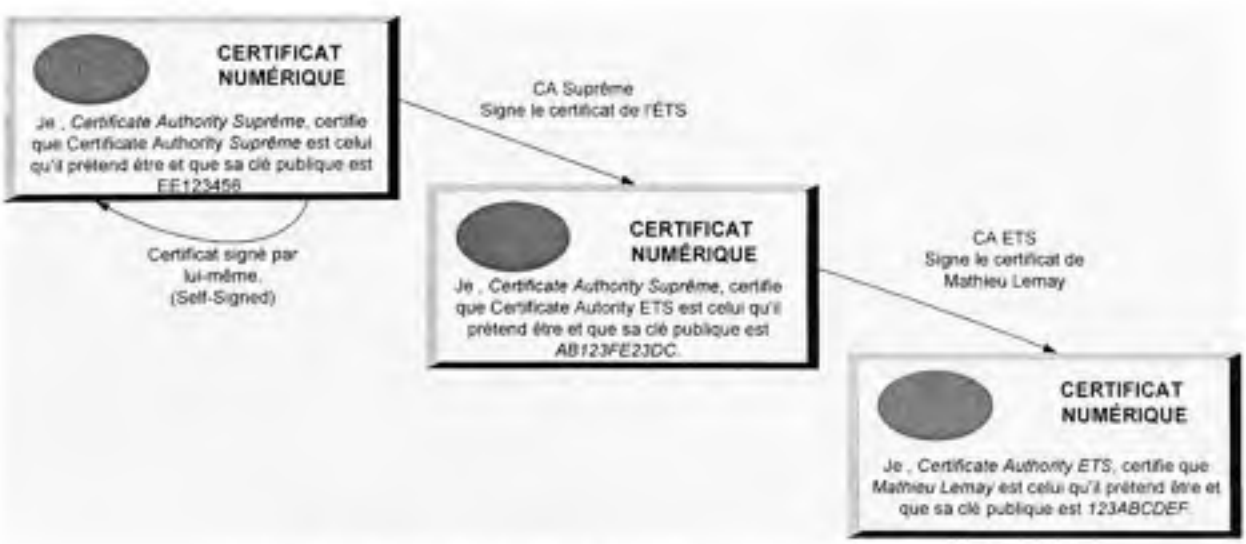

Figure 2.27 ***Exemple de signature hiérarchique.***

Par contre, qui signe le certificat de *Certificate Authority Suprême ?* En fait, il signe lui-même son propre certificat. On appelle ce type de certificat *Self-Signed* ou *Root* et, pour qu'ils soient dignes de confiance, il faut absolument qu'ils soient dans la liste de *Trusted Certificates.*

Dans les sections précédentes, on a introduit les différents concepts de sécurité et de cryptographie nécessaires à la compréhension des systèmes de sécurité utilisés dans les grids. Le *Globus Toolkit* utilise depuis le GT2 une infrastructure de sécurité basée sur les concepts et les outils de sécurité à clé publique qu'on appelle le Grid Security Infrastructure. C'est cette infrastructure qui sera présentée dans la prochaine section.

2.4.4 L'infrastructure de sécurité de Globus (GSI)

On a vu dans les sections précédentes les bases d'une conversation sécuritaire parce que dans les grids la sécurité est une des parties les plus importantes des applications Grid. Ceci est dû au fait que les grids traversent souvent les barrières organisationnelles parce que les ressources sont utilisées par plusieurs organisations différentes. Il faut donc s'assurer que seulement certaines organisations aient accès à ces ressources. Ainsi il faut utiliser les CA et les certificats pour bien authentifier les utilisateurs. De plus, selon l'application, il faudra donner le droit au domaine privé et assurer l'intégrité des données. Finalement, dans les Grids, il existe un cas spécial où un utilisateur délègue ses droits à un autre utilisateur afin qu'une tâche soit accomplie en son nom.

Le Grid Security Infrastructure (GSI) permet d'avoir différents niveaux de sécurité pour les ressources grid créées. La première chose importante à choisir est si on désire avoir une sécurité *Transport-Level* ou bien une sécurité *Message-Level*. Une sécurité au niveau du transport de type *Transport-Level* fait le chiffrement de la couche transport Transport Control Protocol (TCP) grâce au Secure Socket Layer (SSL). Ceci fait le chiffrement de toutes les données qui sont échangées entre les membres de la conversation sécuritaire et il est impossible de savoir quel service ou bien quel type d'information est échangé. Dans ce qu'on appelle le *Message-Level* Security, seulement le corps du message SOAP est chiffré, ceci laisse l'entête lisible au tiers et il devient donc possible de savoir à quelle ressource s'adresse ce message. La sécurité de type *Message-Level* offre ainsi beaucoup plus de fonctionnalités parce qu'on peut déterminer la granularité du chiffrement désiré ; par contre elle est beaucoup moins performante que la sécurité *Transport-Level*.

Globus offre deux types de sécurité *Message-Level* ainsi qu'un type de sécurité *Transport-Level*. Le *GSI Secure Message* procure une sécurité de base au niveau du message et est une implémentation du standard WS-Security. Le *GSI Secure Conversation* est un schème de sécurité *Message-Level* qui consiste à établir un contexte de sécurité entre le client et le serveur. Par la suite il est possible de réutiliser facilement ce contexte de sécurité pour les

messages subséquents. Finalement, il y a le *GSI Transport* qui procure une sécurité de niveau transport grâce au protocole SSL et constitue le schème de sécurité par défaut dans le GT4. Par contre, faire ce choix n'est pas mutuellement exclusif; même si choisir un seul schème suffit à assurer une bonne sécurité pour les ressources, il est possible d'en choisir deux ou même trois mais les performances seront affectées en conséquence.

L'authentification dans le GSI peut se faire de trois façon différentes. On peut utiliser les certificats X.509 tel que vu dans la section précédente. Si la sécurité n'est pas très importante et que le droit au domaine privé ou l'intégrité du message ne sont pas importantes, il est aussi possible d'utiliser uniquement un nom d'utilisateur et un mot de passe. Finalement, on peut aussi rester anonyme, donc non identifié. C'est important de savoir que chaque schème de sécurité vu précédemment peut avoir son type d'authentification. De plus, on peut faire l'authentification au client et au serveur, au serveur seulement ou bien choisir de ne pas avoir d'authentification.

Nous avons vu dans les sections précédentes qu'authentifier les utilisateurs est utilisé afin de pouvoir autoriser ou non l'utilisation des ressources. Afin de permettre ou non l'utilisation de ces ressources, il existe plusieurs mécanismes d'autorisation dans le GT4. Le tableau 2.4 montre les différents types d'autorisation qu'on peut avoir au serveur.

Tableau 2.4

Types d'autorisation pour les services

Type d'autorisation	**Description**
Self	Le client est autorisé s'il a la même identité que le serveur.
Host	L'autorisation est donnée pour un nom de domaine particulier.
Grimpa	Le gridmap est une liste qui fait une association entre un nom distinct de certificats et un utilisateur local sur le serveur.
Identity	Ceci est comme un Gridmap à un seul utilisateur où seul un certain nom distinct a accès au service.

SAML Callout	Ceci permet de déléguer la décision à un autre service qui utilise l'information OGSA-Authz.
None	Aucune autorisation, tout le monde a accès à la ressource.

De plus, il est aussi possible d'avoir une autorisation client, afin de sécuriser le client de services malicieux ou non autorisés. Par contre, ce type d'autorisation est beaucoup plus limité et offre uniquement un sous-ensemble des options offertes par les services.

Tableau 2.5

Types d'autorisation pour les clients

Type d'autorisation	**Description**
Self	Le client est autorisé s'il a la même identité que le serveur.
Host	L'autorisation est donnée pour un nom de domaine particulier.
Identity	Ceci est comme un Gridmap à un seul utilisateur où seul un certain nom distingué a accès au service.
None	Aucune autorisation, tout le monde a accès à la ressource.

Toutefois, si aucune des options précédentes ne convient, il est possible de faire son propre module d'autorisation sans avoir à faire des changements au GT4 puisqu'il procure une interface qui permet de spécifier la classe java responsable de faire cette autorisation.

Dans ce chapitre, nous avons vu comment créer des ressources grids et les sécuriser grâce au chiffrement à clé publique et aux schèmes de sécurité et autorisations du GSI. Dans le prochain chapitre, nous allons aborder la création de différentes ressources grid sécurisés afin de représenter les instruments ou capteurs dans une architecture orientée service appelée le Grid Resources for Instruments Model (GRIM).

CHAPITRE 3

LE GRID RESOURCES FOR INSTRUMENTS MODEL (GRIM)

L'initiative au cœur de cette recherche, le Grid Resource for Instruments Model (GRIM), consiste à créer un ensemble de ressources et de protocoles qui permettront de virtualiser les instruments et les capteurs afin de pouvoir les brancher adéquatement aux grids pour qu'ils y soient consommables par des applications scientifiques. Il faudra aussi rendre possible l'interopération entre les instruments grid et les autres types de ressources grid. Ceci permettra d'avoir des capacités de télécommunication, de calcul et de stockage de données incomparablement plus grandes que celles disponibles pour les instruments actuels.

GRIM a été créé avec comme objectif d'offrir une architecture extensible et modulaire capable de décrire et de contrôler le traitement des données et les comportements que l'on retrouve dans les instruments et capteurs. En utilisant le Web Service Resource Framework (WSRF), le GRIM fournit une plateforme embarquée permettant la réutilisation de services déjà créés pour contrôler, surveiller et partager les instruments et capteurs.

3.1 Motivations

Les principales motivations du modèle GRIM sont de permettre de répondre aux exigences requises par le Laboratory for the Ocean Observatory Knowledge Integration Grid (LOOKING) ainsi qu'à celles des salles de cours du laboratoire de photonique de l'ÉTS. Les instruments virtualisés procureront, de par leur nature distribuée, des capacités qui ne sont pas encore disponibles avec les instruments actuels. En effet, en découplant contrôle et traitement des données afin d'en faire des services indépendants, il est possible d'ajouter de nouvelles caractéristiques d'intelligence dans le pré-traitement et le post-traitement de ces informations. Les différents instruments ont une représentation logique qui peut être consommée par les autres instruments ou applications. L'orchestration consiste à relier entre elles les entrées et sorties de différents services afin d'en créer un nouveau. L'orchestration procure des capacités inégalées pour le traitement des données puisque les instruments

peuvent être surveillés, contrôlés et visualisés par n'importe quelle application logicielle qui peut n'avoir aucune connaissance des données représentées/traitées. Par exemple, un oscilloscope rapide pourrait fournir ses données en temps réel ou différé à un service de Fast Fourier Transform (FFT) ; le résultat serait ensuite envoyé pour visualisation par un service de graphisme 2D. L'ensemble de cette orchestration constituerait un service procurant l'équivalent d'un analyseur spectral, sans toutefois disposer d'un vrai analyseur branché au phénomène physique observé.

Le GRIM a de plus été élaboré afin de tenir compte des contraintes suivantes :

- Les instruments doivent être prêts à tourner, « Plug and Play »,
- Le modèle ne doit pas limiter les fonctionnalités de l'équipement,
- Le temps nécessaire à la modélisation de nouveaux instruments doit être court,
- Les instruments utilisent différents protocoles propriétaires de communication.

Le modèle fournit des solutions à ces problèmes de la façon suivante :

- Le Grid Resource Instrument Model (GRIM) s'est fortement inspiré du standard IEEE1454 afin de procurer la capacité prêt à tourner.
- Le modèle est conçu de façon à accommoder n'importe quelle fonctionnalité et peut facilement être étendu.
- Le modèle est créé comme un « Toolkit » qui pourra être utilisé afin de modéliser des instruments rapidement.
- Les protocoles sont cachés par l'implémentation.
- Le modèle utilise une version modifiée du Globus Toolkit 4 WSRF C/Java core afin d'être compatible avec les standards Grid.

Le modèle GRIM n'est pas une copie de l'IEEE1451 qui procure une interface de service Web au Network Capable Application Processor (NCAP). De tels travaux ont été entrepris entre autres par Sadok et Liscano (2005) mais n'ont pas fourni la standardisation suffisante pour procurer des fonctionnalités grids requises par l'OGSA. Ainsi, même si l'adaptation directe de l'IEEE1451 pourrait facilement être créée grâce au modèle GRIM, ce n'est pas

l'objectif recherché du projet de recherche actuel. Le standard IEEE1451 procure par contre des modèles de données et d'objets qui ont été utilisés comme inspiration au GRIM.

3.2 Notation UML utilisée

Les diagrammes utilisés dans ce chapitre sont présentés en utilisant des diagrammes statiques du Unified Modeling Language (UML). La figure 3.1 présente les objets « entities » et les relations « generalization » qui seront utilisés dans le diagramme UML statique afin de décrire adéquatement le modèle GRIM.

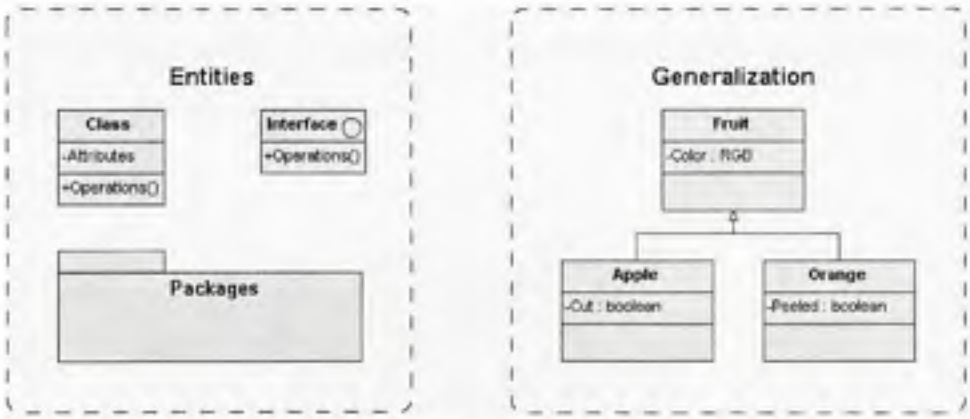

Figure 3.1 ***Les notations UML utilisées dans ce chapitre.***

Dans ce diagramme, il y a trois stéréotypes de classes UML utilisées :

a) <<Interface>> Une interface est un ensemble d'opérations qui sont supportées par des objets possédant cette interface. Une interface ne peut contenir aucun attribut.

b) <<Class>> Une classe est un ensemble d'attributs et d'opérations qui définissent le type d'un objet.

c) <<Packages>> Un package est un ensemble de classes regroupées.

d) <<Généralization>> La généralisation permet de partir d'une classe d'objets de base et d'en créer une autre plus détaillée.

Dans ce chapitre, les types de données suivants seront utilisés :

a) String – Une séquence de caractères,

b) Enumeration – Une liste de valeurs possibles.

c) TimeExpression – Une expression temporelle en millisecondes écoulées depuis le 1er janvier 1970.

d) Double – Un nombre à point flottant et à double précision.

3.3 Les couches du modèle GRIM

La première étape est d'analyser le rôle des différentes couches de services existants dans le GRIM et de voir dans quelles couches il y a aura des interactions directes avec d'autres services du GT4 ou avec les applications.

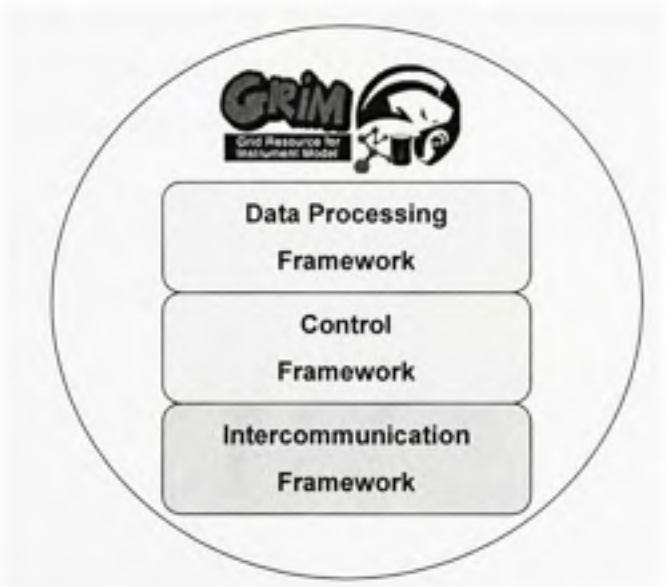

Figure 3.2 ***Les couches du GRIM.***

3.3.1 Intercommunication Framework (GT4)

La première couche, nommée *Intercommunication Framework* (GT4), est responsable de faire l'unification des différents instruments afin qu'ils parlent tous le protocole SOAP. Ceci

permet d'avoir des interactions directes entre les différents instruments. Ceci permet aussi de faire l'approvisionnement des ressources nécessaires à la communication entre les instruments en utilisant les services d'autres intergiciels comme User Controlled LightPaths (UCLP).

Les différentes caractéristiques de cette couche sont :

- protocole commun,
- service d'indexage,
- approvisionnement réseau,
- sécurité de transport.

3.3.2 Control Framework (GRIM+GT4)

La deuxième couche, le *Control Framework*, permet de gérer les appels aux différents instruments. Il permet de faire la gestion de l'autorisation d'accès aux différents services ainsi que de la durée de vie des ces ressources.

Les différentes caractéristiques de cette couche sont :

- politique et accès des utilisateurs,
- contrôle et opérations,
- gestion,
- gestion des notifications.

3.3.3 Data Processing Framework (GRIM)

Finalement, la troisième et dernière couche, le *Data Processing Framework*, sert à traiter les données recueillies grâce aux couches précédentes. Elle sert aussi à l'archivage et à la création de flux de données (streams) utiles à la visualisation en temps réel.

Cette couche possède les caractéristiques suivantes :

- analyse et introspection des services,
- prétraitement des données,
- historique and archives,
- flux de données,
- notifications des données.

3.4 Les ressources du modèle GRIM

3.4.1 Architecture du modèle GRIM

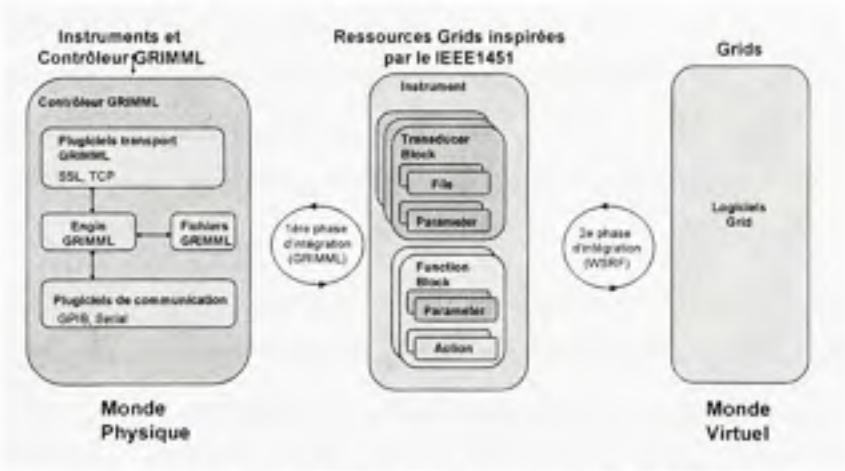

Figure 3.3 ***Les différentes ressources du GRIM.***

Dans le GRIM, les instruments et les capteurs sont modélisés en tant que ressources grid. Ces ressources sont basées sur les objets qui sont définis dans l'IEEE1451.0 mais sont adaptées aux nouvelles technologies XML et WSRF. Il y a deux types fondamentaux de ressources : **les blocs** et **les composants**. Les blocs sont des conteneurs à composants et offrent des liens d'état entre les différents composants qui le constituent. Dans les sections suivantes, nous allons voir comment les ressources sont créées. On peut voir à la figure 3.3 que le GRIM et son utilisation doivent être implémentés en deux phases d'intégration. La première phase

consiste à créer les objets virtuels représentant le GRIM à partir des contrôleurs de bas niveau, tandis que la deuxième phase d'intégration visera l'utilisation du modèle par les applications.

3.4.2 Création de ressources GRIM

La création de ressource se fait en plusieurs étapes :

1. le module d'instrument est mis en marche,
2. il fait la découverte automatique des instruments reliés,
3. il trouve les pilotes XML des ces instruments,
4. il crée les ressources pour chaque instrument en communiquant via Secure Socket Layer (SSL) avec le serveurs grid,
5. il met l'état des ressources à inactif,
6. il enregistre les ressources au service d'indexage de GT4,
7. il tente de communiquer avec le service,
8. si la communication est établie, il met l'état du service à actif.

Ces étapes de création s'assurent que la ressource est capable de communiquer adéquatement avec l'instrument. Il est important de noter qu'une vérification continue de cette connectivité est nécessaire. S'il y a un problème de communication, les ressources de cet instrument doivent être désactivées. Après un certain temps d'inactivité, cette ressource sera détruite et enlevée du service d'indexage.

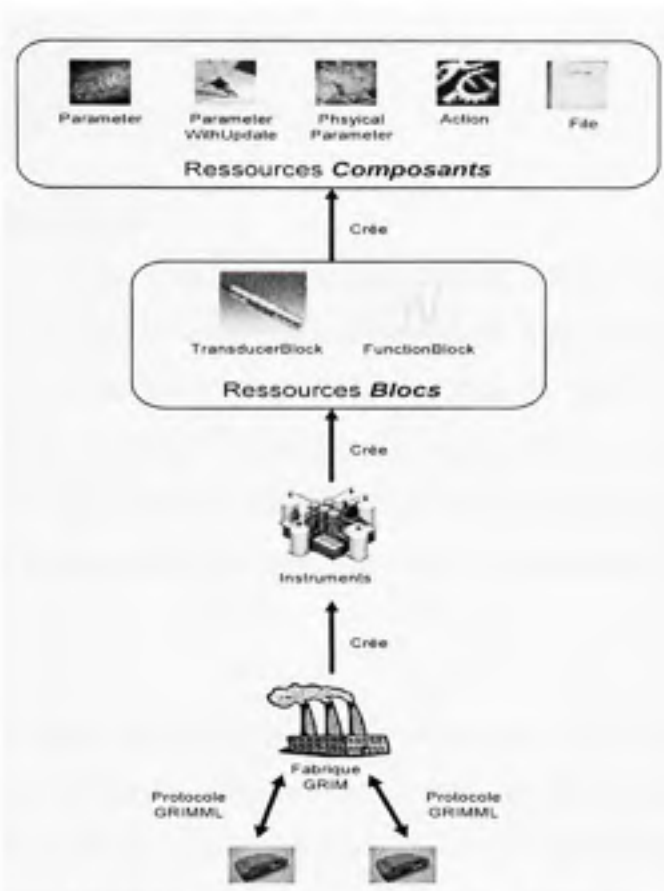

Figure 3.4 ***Fabrique GRIM et création des ressources.***

Les ressources *Component* sont les blocs de base du modèle. Elles permettent d'interagir avec les différentes fonctionnalités que procurent les capteurs ou instruments. Il existe 5 différentes sortes de ressources *Component*. Les ressources *Parameter, ParameterWithUpdate et PhysicalParameter* permettent de contenir une valeur qui peut être lue ou écrite. Elle s'assure que les limites et le type de données sont corrects. La ressource *File* permet d'avoir un transfert binaire de/vers un *Block*. Finalement, les ressources *Action* permettent d'avoir un "billet" de transaction qui permet d'exécuter des opérations longues sans avoir à attendre la valeur de retour.

On peut voir sur la figure 3.4 que la ressource *Instrument* est responsable de garder le lien de communication avec le contrôleur de la ressource. Elle sert aussi de fabrique abstraite aux ressources de type blocs.

3.5 Les ressources composants du GRIM

3.5.1 La ressource *Parameter*

La ressource *Parameter* est uniquement un "conteneur" de valeurs détaillé. Elle implémente le patron observateur et d'autres services peuvent s'inscrire afin de recevoir des notifications sur les changements de différents *Topics* tel que vu dans la section 2.3.3. Les ressources *Parameter* sont créées à partir d'une représentation XML envoyée à l'opération *createResource()* du service ParameterFactory. Il est possible de lire (*read()*) la valeur du conteneur. De plus, s'il est modifiable (*mutable=true*), on peut aussi écrire une valeur à ce paramètre.

Le paramètre peut aussi bien être un scalaire qu'un vecteur ou une liste de valeurs.
La ressource *Parameter* est généralement utilisée lorsque la valeur contenue n'a pas d'unité physique et qu'il n'est pas nécessaire de communiquer avec l'instrument de mesure avant ou après l'écriture/lecture de cette valeur dans la ressource.

La structure de données permet de décrire le type de données (vecteurs, listes, séries temporelles, etc.). La ressource *Parameter* possède aussi une courte description de la source des données. La ressource sera principalement utilisée afin d'être lue par d'autres types de ressources. Les applications de surveillance interagissent directement avec la ressource *Parameter* afin de faire l'acquisition de l'information. Puisque la ressource *Parameter* est autonome, il n'est pas nécessaire de savoir comment contrôler l'appareil pour pouvoir faire la surveillance des données.

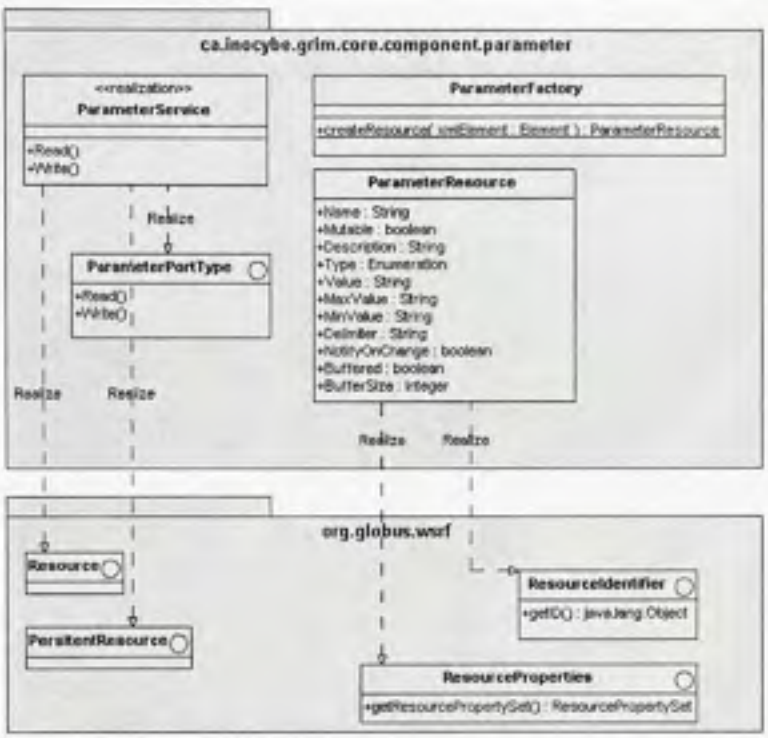

Figure 3.5 ***Diagramme de la ressource Parameter.***

Par exemple, si un *Function Block* effectue une dilatation temporelle $f(t) = \alpha t$, alors le paramètre alpha peut être modélisé en tant que ressource *Parameter* parce qu'il n'a pas d'unité physique et n'a pas besoin de communiquer avec un capteur ou un instrument lorsqu'on modifie la valeur qu'il contient. On peut voir sur la figure 3.5 les différents attributs et opérations possibles pour les ressources *Parameter*. On peut voir que les deux seules opérations possibles sont *read()* et *write()*. Ces opérations permettent de modifier la propriété *Value*. Les autres paramètres ne peuvent pas être modifiés directement et dépendent du fichier GRIMML descriptif de l'instrument.

3.5.2 La ressource ParameterWithUpdate

La ressource *ParameterWithUpdate* est un prolongement de la ressource *Parameter*. Elle n'a pas la même implémentation mais toutefois a presque la même structure de données. La

donnée additionnelle est un TimeStamp de la dernière fois où la valeur a été mise à jour. Elle sert ainsi à représenter une valeur qui doit être lue ou écrite depuis un transducteur ou un instrument.

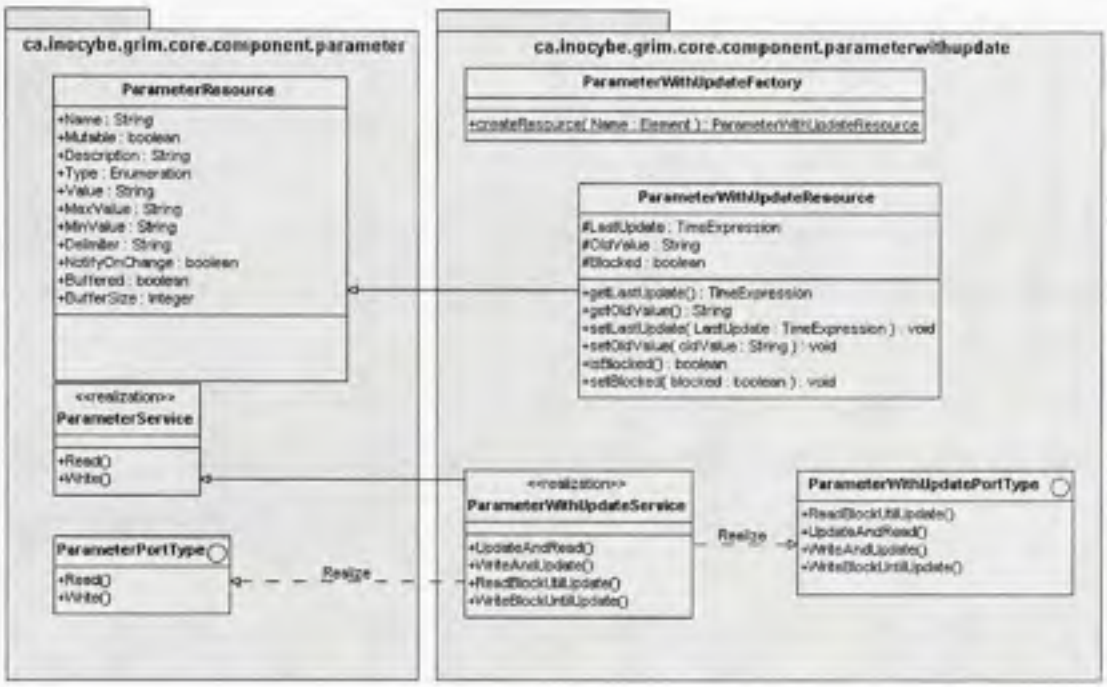

Figure 3.6 ***Diagramme de la ressource ParameterWithUpdate.***

La ressource *ParameterWithUpdate* a une capacité de blocage et attend que la valeur soit mise à jour avant de lire ou d'écrire la valeur.

Par exemple, pour un bloc *Function Block* qui contrôle un filtre matériel à réponse impulsionnelle finie (FIR), il est possible d'utiliser les composants *ParameterWithUpdate* afin de changer la matrice de coefficients du filtre. Le figure 3.6 est une extension de la ressource *Parameter* ; elle a de nouvelles propriétés comme le temps de la dernière mise à jour de la valeur (*LastUpdate*), l'ancienne valeur (*OldValue*) et un contrôle de blocage *(Blocked)* si la ressource est en train d'interagir avec l'instrument.

Les opérations *updateAndRead()* et *readAndUpdate()* diffèrent afin de spécifier exactement quelle valeur doit être retournée par l'opération. Pour l'opération *updateAndRead()* la ressource doit mettre à jour la valeur qu'elle garde en mémoire en interagissant avec l'instrument et ensuite la retourner au client. Dans le cas de *readAndUpdate(),* la valeur actuelle gardée en mémoire sera retournée et ensuite une mise à jour sera effectuée.

3.5.3 La ressource PhysicalParameter

La ressource *Physical Parameter* est le prolongement de la ressource *ParameterWithUpdate* et elle possède donc les même caractéristiques permettant de brancher un instrument de mesure. Par contre, c'est un cas spécial de paramètre puisqu'elle a une unité de mesure qui vient régir la valeur contenue par cette ressource. La plupart des paramètres d'un *Transducer Block* seront des *PhysicalParameters*, ainsi que la majorité des ressources qui manipulent les données provenant des instruments.

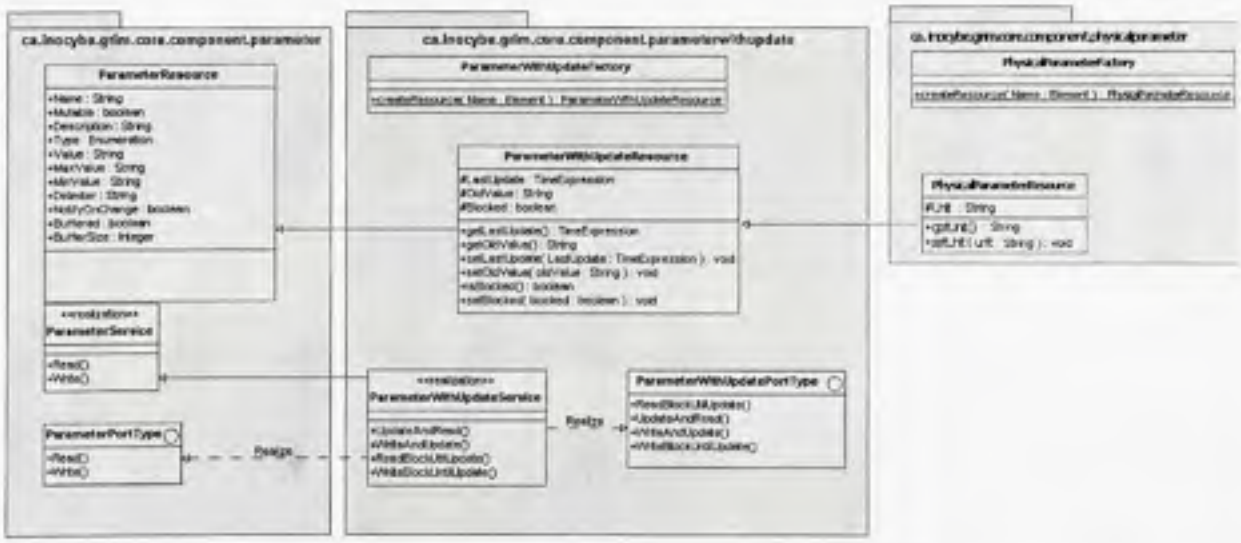

Figure 3.7 ***Diagramme de la ressource PhysicalParameter.***

Par exemple, un capteur de température utilisera le *PhysicalParameter* afin de modéliser la température mesurée par ce capteur. Sur la figure 3.7, on peut voir que le PhysicalParameter est identique au *ParameterWithUpdate* avec pour seule exception d'avoir une propriété appelée *Unit* qui correspond à l'unité de mesure physique. La différence dans l'implémentation est que les unités physiques doivent être compatibles lors de l'orchestration ou de l'utilisation dans les applications Grid.

3.5.4 La ressource *File*

La ressource File est utilisée afin de modéliser une interaction avec un fichier binaire qui est utilisé par le bloc. Cette ressource permet de lire ou écrire des fichiers dans un bloc de fonctionnalité. Si l'opération offerte par le bloc a besoin de fichiers d'entrée ou de sortie, la ressource File sera utilisée. C'est le cas, par exemple, lorsqu'on veut transférer des fichiers d'images captées par des instruments ou bien simplement lire les données brutes de l'appareil grâce au fichier de données.

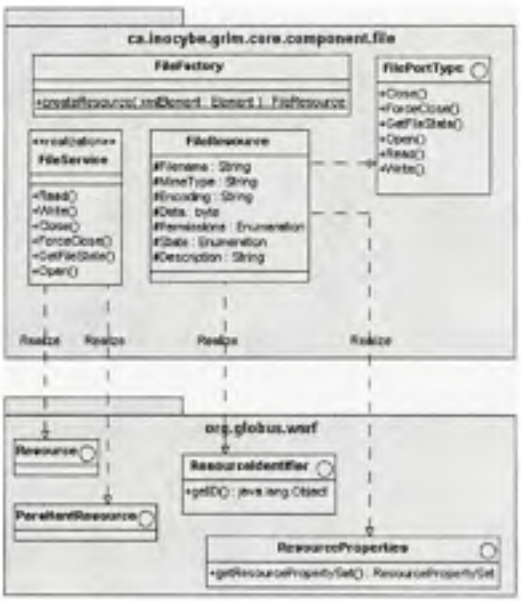

Figure 3.8 ***Diagramme de la ressource File.***

Si on modélise un appareil-photo numérique, les interactions avec les images se feraient sous forme de composants *File*.

3.5.5 La ressource *Action*

La ressource action est une ressource transactionnelle qui permet à de longues opérations d'être effectuées par un bloc. En effet, si le bloc possède une ressource action, l'utilisateur peut y faire appel et, si elle n'est pas en cours d'utilisation, le travail s'exécutera sans bloquer l'opération. En effet, une ressource sera créée pour s'assurer du bon fonctionnement et procurer l'état de l'opération.

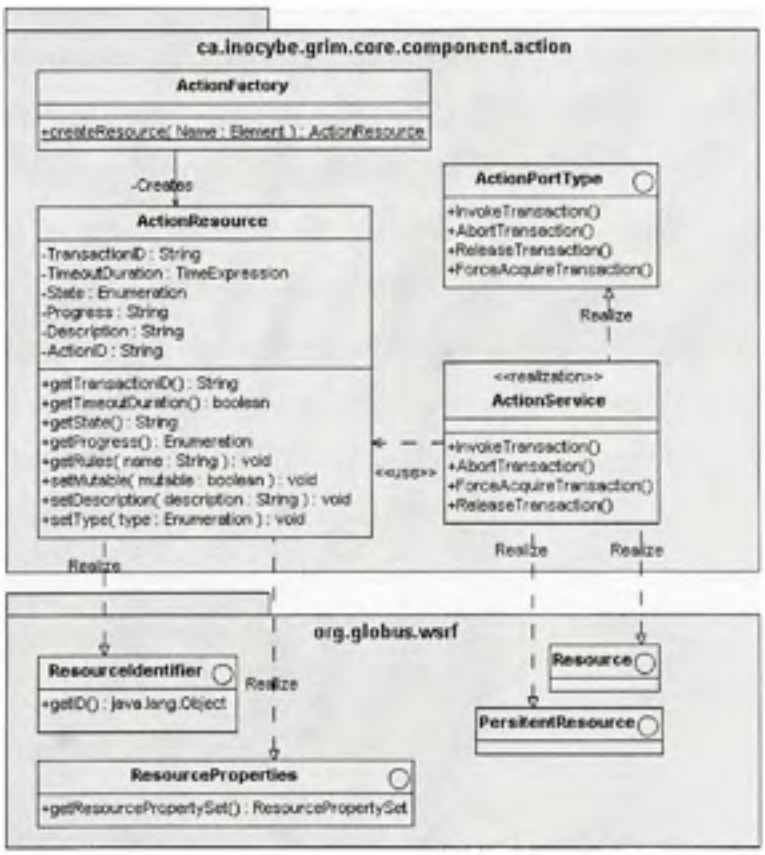

Figure 3.9 ***Diagramme de la ressource Action.***

En science, il est parfois utile d'avoir des expériences de très longue durée avant de faire le traitement des données ; les actions peuvent être utilisées dans ce but. La figure 3.9 démontre les différentes opérations que procure une ressource d'action. On peut entre autres y voir un identificateur pour la transaction (*TransactionID)*, un état (*State)* et un indicateur de progrès (*Progress)* .

3.6 Les ressources blocs du modèle GRIM

3.6.1 La base des blocs

Tous les blocs ont une base commune qui leur permet de gérer l'état de leurs composants. Le bloc doit réaliser des tâches spécifiques, qui peuvent avoir lieu soit dans le matériel situé à proximité de l'instrument (« in-situ ») ou à distance (« ex-situ »), sur le serveur GRIM. Lorsque le bloc est créé, il a un état *UNINITIALIZED* et tous les composants qui s'y rattachent ont le même état. Le contrôleur de l'instrument doit ensuite tenter une communication ; si la communication est possible via la fonction *Initialize,* le bloc tombe dans l'état *INACTIVE.* Il faut finalement appeler la fonction *Activate* (ce qui se fait de façon automatique) afin de mettre le bloc et ses composants à l'état *ACTIVE* qui rend le bloc et ses composants prêts à être utilisés dans les grids. Si une erreur de communication survient avec l'instrument, le bloc retrouve l'état *UNITIALIZED* et le processus d'initialisation doit être refait avant que la ressource ne redevienne *ACTIVE.*

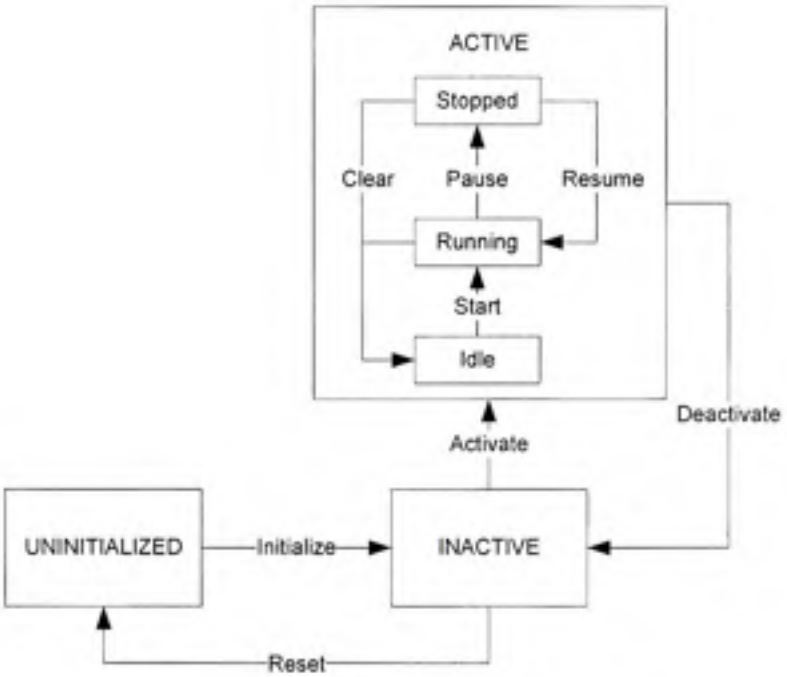

Figure 3.10 ***Machines à états des Blocks.***

Les étapes suivantes doivent être effectuées, comme le montre la machine à états de la Figure 3.10, lorsque le contrôleur est mis en marche. Il bouclera et tentera de faire l'initialisation de l'instrument jusqu'à ce que celui-ci réponde aux requêtes qui lui sont envoyées.

1. Lors de la mise sous tension, la ressource est créée selon la procédure vue dans la section 3.4.2; l'état du bloc est alors *UNINITIALIZED.*.
2. Le contrôleur tentera de faire une communication initiale et enverra des commandes d'initialisation nécessaires à l'instrument. Si cette étape est réalisée avec succès, les blocs de l'instrument tomberont dans l'état *INACTIVE.*
3. Chacun des blocs tentera alors de faire des lectures et écritures sur les différentes ressources qui composent le bloc ; si cette opération se fait avec succès, il passera à l'état *ACTIVE.*
4. À la moindre erreur de lecture ou d'écriture avec l'instrument, le bloc retournera à son état *UNINITIALIZED.*

Dans cette section nous avons vu comment les blocs de base gèrent l'état des composants. Par contre, les blocs ont aussi des fonctionnalités ; il existe deux types de blocs : les *Transducers Blocks*, qui font des lectures/écritures de façon automatiques ou manuelles, et les Function Blocks, qui ont seulement une fonction *execute*. C'est à cause de sa simplicité qu'on commence dans la section qui suit par décrire le *Function Block*.

3.6.2 Function Block

Le *Function Block* est le bloc le plus polyvalent du modèle. Il est un procureur d'opérations et permet à n'importe quel utilisateur de décrire une fonctionnalité offerte par l'instrument ou le service. Par exemple, un contrôleur Proportionnel Intégral Différentiel (PID) très simple pourrait être représenté comme un *Function Block* car il n'a pas à interagir avec les instruments ou à faire des lectures périodiques. Il aurait donc les paramètres suivants :

P – Paramètre de contrôle proportionnel – **Parameter** – Mutable, Scalar
I – Paramètre de contrôle d'intégration – **Parameter** – Mutable , Scalar
D – Paramètre de contrôle de différenciation – **Parameter** – Mutable, Scalar
x – Signal d'entrée – **PhysicalParameter** – Scalar, Buffered, Meters.
u – Sortie désirée – **PhysicalParameter** – Mutable, Scalar, Meters
z – Sortie réelle – **PhysicalParameter** – Scalar, Buffered, Meters.

Ensuite il suffit d'appeler la fonction *execute()*, *start()* ou *stop()* afin que le bloc entre en fonction et réagisse aux valeurs qui changent à l'entrée.

3.6.3 Transducer Block

Le *Transducer Block* est un sous-type de bloc fonctionnel qui est utilisé pour les transducteurs (capteurs ou actionneurs). Il réalise des opérations de lecture ou écriture périodiques de façon manuelle ou automatique des composants *ParameterWithUpdate* ou *PhysicalParameter* qui composent ce bloc.

Les blocs sont créés à partir d'une ressource Instrument qui existe seulement en tant que contenant pour les différents blocs créés. Afin de créer ces blocs de façon réutilisable, il faut avoir un fichier de configuration qui décrit l'instrument comme le ferait une feuille de spécifications. Les fichiers GRIMML que nous verrons dans la section suivante servent à décrire un instrument afin de pouvoir le virtualiser rapidement.

3.7 Les fichiers GRIMML

Afin de pouvoir facilement modéliser les instruments de toutes sortes, il faut être capable de créer tous les objets grids correspondants à un instrument à partir d'un fichier de description GRIMML. On peut voir sur la Figure 3.12 et à l'Annexe 2 la structure de données du fichier GRIMML.

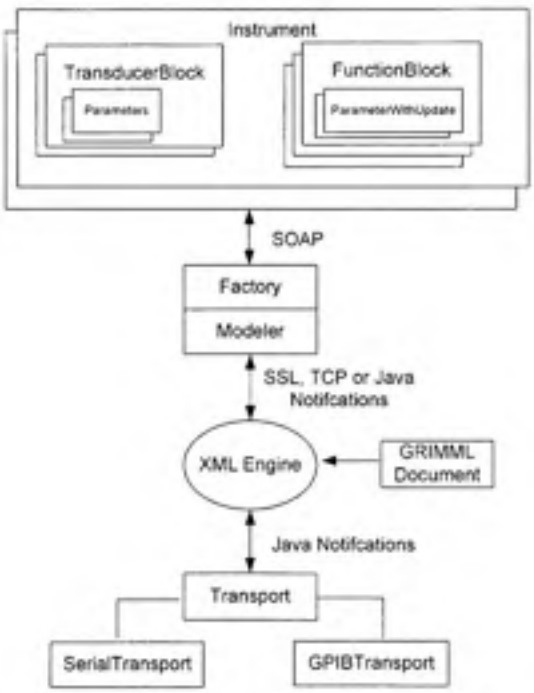

Figure 3.11 ***Le fichier GRIMML au cœur du GRIM.***

On peut voir à la Figure 3.11 que le fichier GRIMML est au centre du modèle GRIM. Grâce au moteur XML (XML Engine), il est capable de communiquer avec le composant *Modeler*

du *Factory,* afin de créer les ressources volatiles qui correspondent à cet instrument. De plus, il contient aussi l'information nécessaire pour communiquer avec les instruments via les différentes interfaces de transport.

Ce fichier peut être vu comme un fichier de configuration. Les attributs principaux sont le nom et l'identificateur universel (UUID) version 4 qui permettent de faire la correspondance entre la ressource et la structure de données de bas niveau. Le fichier GRIMML est au cœur de la modélisation GRIM. Il permet de faire le « pont » entre les commandes physiques à envoyer à l'instrument.

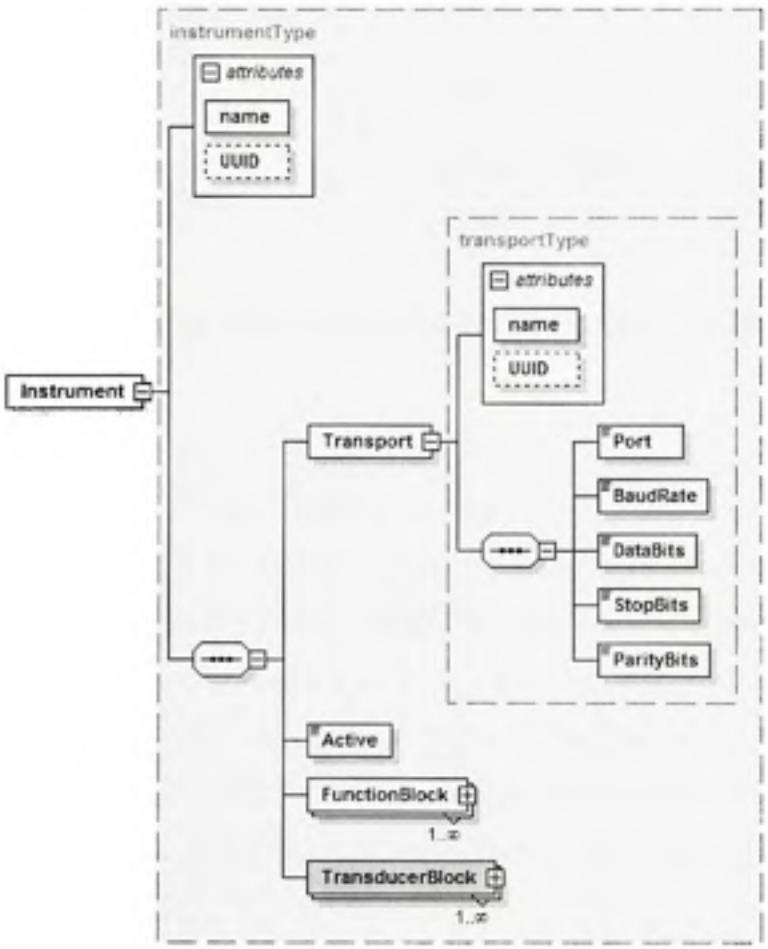

Figure 3.12 ***Schéma du format de fichier GRIMML.***

Ce document reste en mémoire dans le contrôleur GRIMML et sert de document d'interaction avec les ressources grid de haut niveau. On peut voir que l'instrument est modélisé à l'aide de *FunctionBlock* et de *TransducerBlock*. De plus, une section appelée transport est utilisée afin de décrire le type de protocole de transport à utiliser afin de communiquer avec l'instrument ou les capteurs.

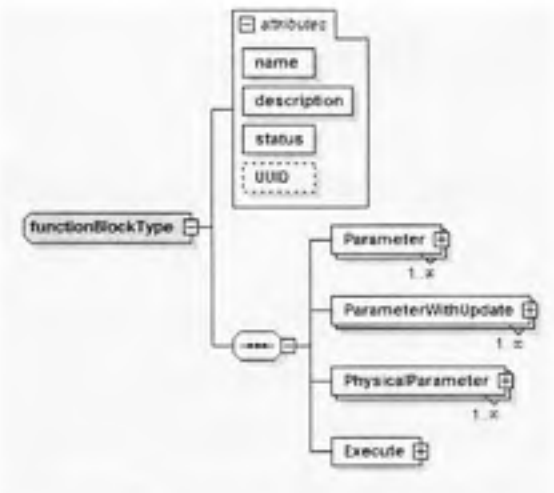

Figure 3.13 ***Schéma du format de fichier GRIMML(FunctionBlock).***

On peut voir à la Figure 3.13 que les bloc de type *FunctionBlock* peuvent contenir les différents paramètres possibles. Le format devrait être étendu afin de pouvoir aussi contenir de l'information sur des ressources *Action* et *File*. En contraste à la Figure 3.14 on peut voir qu'ils sont presque identiques, sauf que les block *FunctionBlock* ont une section intitulée *Execute*. Cette section est utilisée afin de déterminer la commande qui doit être envoyée à l'instrument pour activer le bloc. En effet, le bloc de type *FunctionBlock* peut être utilisé afin d'envoyer une commande complexe nécessitant plusieurs paramètres d'entrée et de sortie vers l'instrument d'un seul coup.

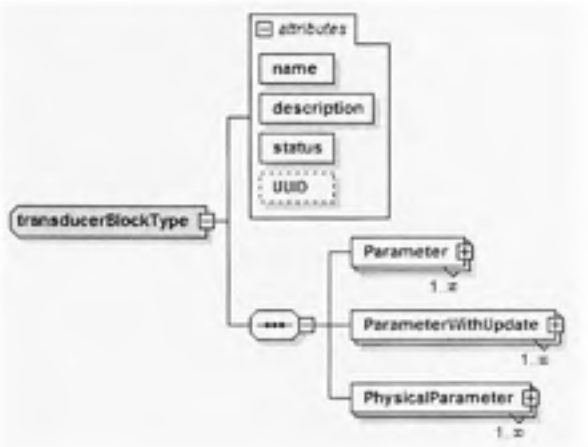

Figure 3.14 ***Schéma du format de fichier GRIMML (TransducerBlock).***

Finalement, la Figure 3.15 montre que le format de fichier GRIMML est identique aux structures de données que l'on retrouve dans les composants des sections précédentes. En effet, il y a une correspondance directe entre le document gardé en mémoire et chargé à partir du fichier GRIMML et les ressources créées pour modéliser l'instrument. Le fichier GRIMML contient toute l'information et tous les changements y sont stockés de façon permanente. On peut voir que tous les éléments du fichier ont un attribut nommé UUID. Cet identificateur unique est créé lors du chargement initial du fichier par le « Modeler ». Les éléments XML « Input » et « Ouput » sont les commandes à envoyer ou à recevoir afin de lire ce paramètre. Ces éléments utilisent les expressions régulières afin de pouvoir extraire ou ajouter les valeurs correspondantes au paramètre en question.

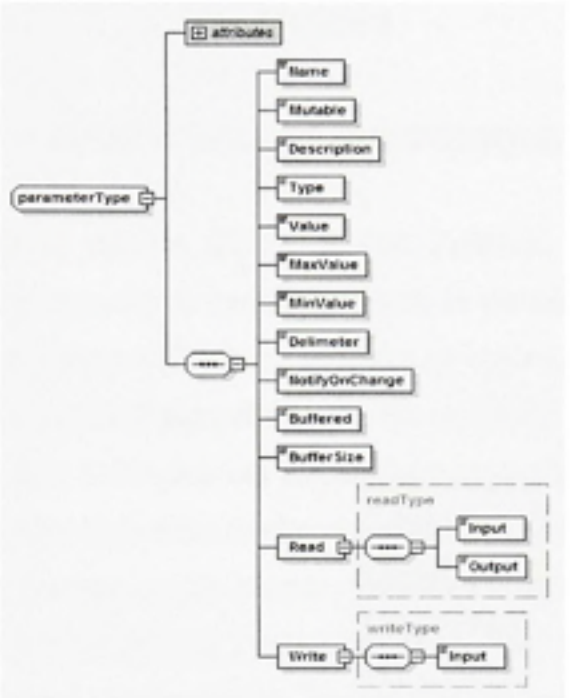

Figure 3.15 ***Schéma du format de fichier GRIMML.***

Pour terminer, nous avons vu au chapitre 3 les ressources grid et les structures de données du modèle GRIM. On a aussi vu comment il est possible de modéliser les instruments grâce au GRIMML. Dans le prochain chapitre, on verra comment on peut utiliser le GRIM et le GRIMML pour modéliser le laboratoire de photonique de l'ETS.

CHAPITRE 4

LABORATOIRE DE PHOTONIQUE

Afin de pouvoir valider le modèle GRIM, il faut l'utiliser pour modéliser quelques instruments. Il faut par contre avoir un besoin qui guide la virtualisation des instruments. À l'École de Technologie Supérieure (ÉTS), on retrouve ce cas au laboratoire de photonique qui est très récent et où le nombre d'appareils de mesure est limité. Il y est donc nécessaire de pouvoir partager et contrôler à distance ces dispendieux appareils afin de pouvoir en faire profiter une communauté plus imposante de chercheurs et ainsi rentabiliser le laboratoire.
Dans cette section, trois instruments de mesure modélisés seront détaillés et présentés. Il s'agit d'un amplificateur à fibre dopée à l'erbium (EDFA), d'un commutateur optique ainsi que d'un analyseur de spectre optique (OSA). Pour chacun de ces appareils, la base de leur principe de fonctionnement sera présentée ; suivront les différents paramètres à contrôler et le modèle GRIM correspondant à ces paramètres. Finalement, le type de transport et le protocole de communication seront présentés avant de décrire le fichier GRIMML qui fait la description virtuelle de l'instrument.

4.1 Amplificateur à fibre dopée à l'erbium, modèle EFA-P15 de MPB Communications Inc.

Les amplificateurs à fibre dopée à l'erbium sont les types d'amplificateurs les plus utilisés en communications optiques à longue portée puisqu'ils sont très efficaces dans la région d'opération à 1550 nm où les fibres de télécommunications optiques ont leur minimum d'atténuation lors des transmissions. L'amplificateur utilisé dans le laboratoire est le modèle EFA-P15 de MPB Communications Inc..

4.1.1 Principe de fonctionnement d'un EDFA

La figure 4.1 présente le diagramme des composants d'un amplificateur à fibre dopée à l'erbium. La fibre dopée à l'erbium transforme l'énergie injectée à courte longueur d'onde

(autour de 980nm ou 1480nm) en signal autour de 1550nm, qui est sa longueur d'onde naturelle d'émission spontanée. On appelle ce phénomène le pompage optique. On superpose donc un faisceau de pompage laser à 980 nm au signal entrant à 1550 nm grâce à un coupleur *Wavelength Division Multiplexer (WDM)*. Les isolateurs d'entrée et de sortie empêchent les réflexions qui pourraient causer de l'interférence avec le laser de pompage ou bien ajouter du bruit d'émission spontanée au signal d'entrée.

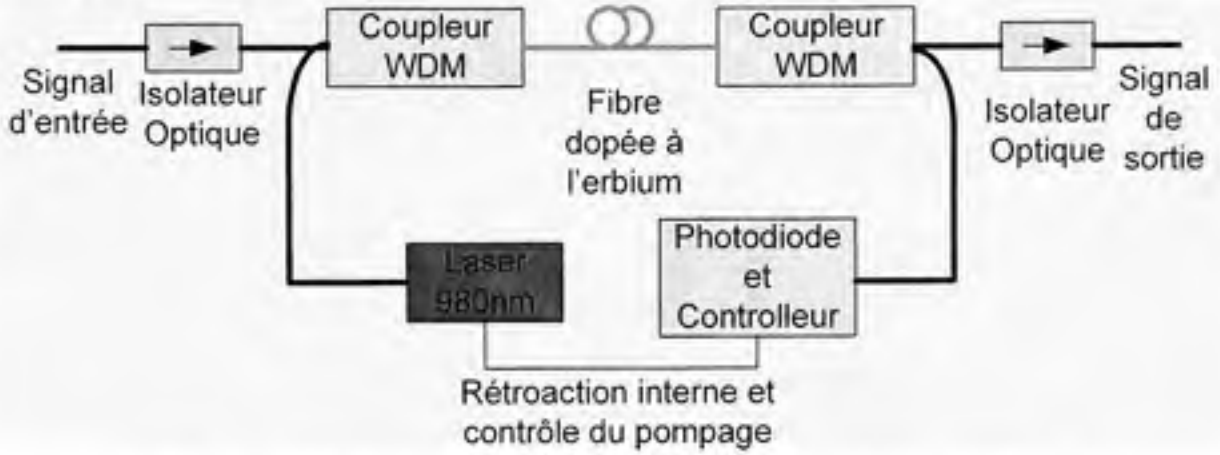

Figure 4.1 ***Diagramme d'un amplificateur à fibre dopée à l'erbium.***

4.1.2 Communication avec l' EFA-P15

L'EFA-P15 du laboratoire a un seul étage d'amplification, pompé à 980 nm, et est capable de fournir un minimum de 15 dBm de puissance de sortie lorsqu'un signal d'entrée saturant de -4 dBm est injecté. Le contrôleur du EFA-P15 est très simple et consiste en un bouton d'ajustement manuel qui permet d'augmenter la valeur du courant de la diode laser et donc le niveau de pompage optique, tout en procurant une lecture de puissance sur un afficheur à cristaux liquides (ACL) à l'aide d'un coupleur de puissance 1%.

La communication avec l'appareil se fait par un port série RS-232 opéré en mode 3 lignes seulement, soit : Tx, Rx et Gnd (voir figure 4.2). Il est nécessaire d'utiliser un câble de type « null-modem » qui relie le Tx (pin 2) du DTE (Data Terminal Equipment) au Rx (pin 3) de

l'autre DTE. Ceci permet de relier ensemble deux DTE. Les caractéristiques de ce port série sont énumérées au Tableau 4.1.

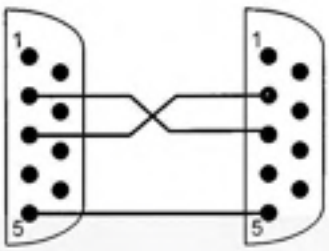

Figure 4.2 ***Diagramme de connectivité du port série de l'EFA-P15.***

Tableau 4.1

Caractéristique du port série pour l'EFA-P15

Caractéristique du port RS-232	**Valeur**
Baud Rate	9600
Data Bits	8
Stop Bit	1
Parity	None
Hardware Control	Off
Echo	None

Ainsi, il est possible d'envoyer des commandes à l'amplificateur lorsqu'il est branché au port série d'un ordinateur. Les commandes du tableau 4.2 sont celles qui sont importantes et qui seront utilisées pour la modélisation avec le GRIMML.

Tableau 4.2

Commandes pour communiquer avec l'EFA-P15 de MPB

Commandes	Description
$LDON	Mets la diode laser en marche. Il faut par contre que la clé de sécurité soit en position « ON ».
$LDOFF	Eteind la diode laser.
$GETOUTPUT	Retourne la valeur numérique de la puissance de sortie.
$GETLDCURRENT	Retourne la valeur de courant d'opération de la diode laser ; la valeur de retour n'est pas calibrée.
$SETADJUST	Permet d'ajuster la valeur du courant de 0 à 100 en pourcentage où 100 est le maximum de courant.
$GETSTATE	Retourne l'état de l'amplificateur : 1 – Problème de conditions d'interlock. 2 – Problème avec le TEC. 4 – Clé de sécurité mise à ON. 8 – Courant additionnel présent. 16 – Amplificateur en attente de la procédure de mise en marche. 32 – Amplificateur en procédure de mise en marche. 64 – Amplificateur en marche.

4.1.3 Modélisation du EFA-P15 avec GRIM

Afin de modéliser les fonctionnalités importantes de l'EFA-P15, il a fallu créer le modèle de ressources suivant en fonction des commandes vues à la section 4.1.2. On peut voir sur la figure 4.3 qu'afin de modéliser l'EFA-P15, on a créé 2 blocs, soient un *FunctionBlock* nommé EDFASettings et un *TransducerBlock* nommé EDFAStatus. Le block EDFAStatus est mis à jour toutes les 5 secondes, retournant une lecture du courant de la diode, de la puissance de sortie et de l'état de l'amplificateur.

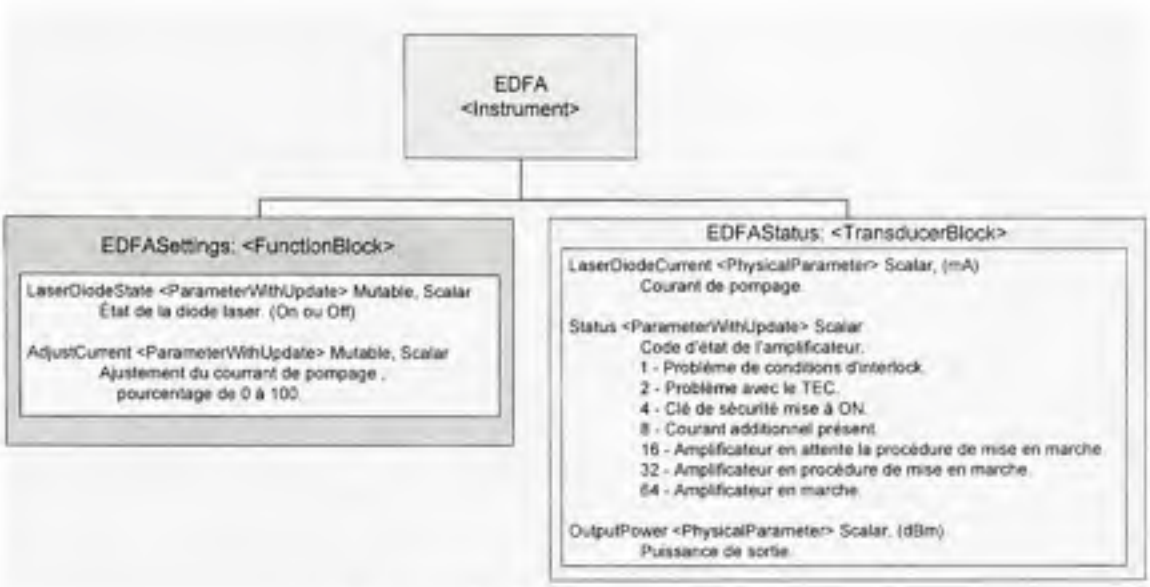

Figure 4.3 ***Modèle GRIM de l'EFA-P15.***

De plus, les notifications du bloc EDFAStatus sont mises en marche grâce à la propriété notifyFromChange du GRIMML. Donc chaque fois qu'une valeur change, tous les services qui sont inscrits afin d'observer les changements de structure de données recevront la nouvelle valeur. Le bloc EDFASettings, lui, sert simplement à modifier les valeurs de l'instrument, puisque les paramètres sont *Mutable*. Par contre, les paramètres de ce bloc auraient pu être ajoutés directement au bloc EDFAStatus. Il a été choisi d'opter pour un FunctionBlock afin de bénéficier de plus de sécurité car il faut invoquer *Execute()* afin que les changements y soient écrits. Le fichier GRIMML de cet instrument est listé à l'Annexe III.

4.2 Analyseur de Spectre Optique, modèle AQ6317B de Ando

L'analyseur de spectre optique permet de mesurer la distribution spectrale d'une one lumineuse. Les analyseurs de spectre sont utilisés afin d'étudier les comportements spectraux de différents dispositifs placés sous test. Dans le laboratoire de photonique, ils font partie des instruments les plus importants et dispendieux. L'analyseur de spectre optique qui sera virtualisé est le AQ6317B de Ando.

4.2.1 Principe de fonctionnement de l'OSA

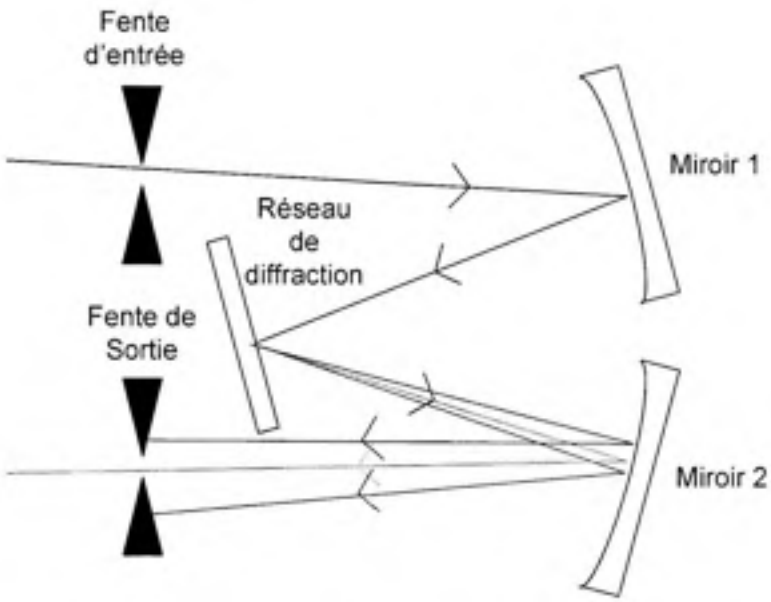

Figure 4.4 ***Fonctionnement d'un monochromateur.***

L'analyseur de spectre a comme base un monochromateur Zollner-Thurnar. Le monochromateur est un dispositif qui permet de filtrer un signal polychromatique afin de sélectionner uniquement une seule longueur d'onde à la sortie. La figure 4.4 présente son principe de fonctionnement : un signal lumineux comportant plusieurs longueurs d'onde entre par la fente d'entrée. Le premier miroir collime le faisceau et le réfléchit vers le réseau de diffraction amovible. Le réseau de diffraction sépare les différentes longueurs d'onde. Une fois cette séparation effectuée, seule une longueur d'onde peut ressortir par la fente de sortie. En déplaçant le réseau de diffraction, il est possible de sélectionner une longueur d'onde différente et c'est ainsi qu'un balayage en longueur d'onde peut être effectué.

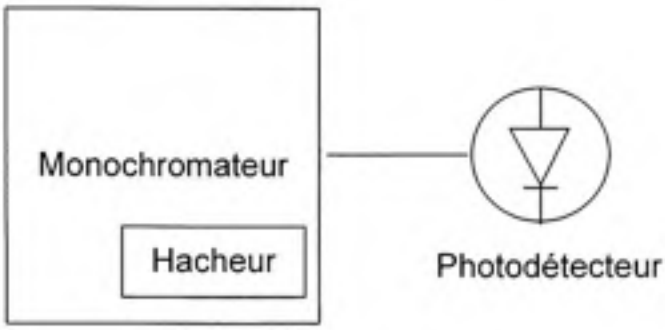

Figure 4.5 ***Photodétecteur et hacheur de l'analyseur.***

La sortie du monochromateur est incidente sur un photodétecteur qui permet de mesurer la puissance lumineuse à cette longueur d'onde particulière. Le hacheur est utilisé pour augmenter le rapport signal-sur-bruit du photodétecteur et ainsi la sensibilité des mesures. En modes Normal Auto ou Normal Hold, le hacheur est hors fonction ; à sensibilité High 3, le hacheur tourne à 270 Hz.

4.2.2 Communication avec l'Ando AQ6317B

Pour communiquer, l'AQ6317B de Ando utilise un port GPIB. Il a une adresse majeure de 0x0 et une adresse mineure de 0x12. Ces adresses sur le port sont assignées lors de la configuration de l'appareil. Puisque le protocole SCPI (Standards Commands for Programmable Instrumentation) n'existait pas lors de la conception de cet appareil, le protocole utilisé est propriétaire et ses commandes utilisées pour la modélisation GRIMML sont énumérées au tableau 4.3.

Tableau 4.3

Commandes pour communiquer avec l'AQ6317B de Ando

Commandes	Description
SPAN	Permet de changer la largeur de balayage (SPAN)
CTRWL	Permet de changer la longueur d'onde centrale du balayage.
RESLN	Permet de changer la résolution
SNHD	Change la sensibilité à Normal Hold
SNAT	Change la sensibilité à Normal Auto
SMID	Change la sensibilité à Medium
SHI1	Change la sensibilité à HIGH 1
SHI2	Change la sensibilité à HIGH 2
SHI3	Change la sensibilité à HIGH 3
SRQ	Permet de masquer les interruptions du port GPIB.
LDATA	Mets les données de l'axe des abscisses sur le bus de lecture.
WDATA	Mets les données de l'axe des ordonnées sur le bus de lecture.
SGL	Exécute un seul "Sweep"

La majorité des commandes précédentes doivent être envoyées via le bus de contrôle du port GPIB et ne permettent pas de faire l'acquisition de données. Seules les commandes LDATA et WDATA permettent de faire cette acquisition. Il est aussi important de bien configurer les interruptions grâce à SRQ, afin de recevoir une notification d'interruption lorsque les données sont prêtes à être lues après qu'une opération de balayage a été effectuée. Il s'écoule un délai considérable de quelques secondes après le balayage, lorsque l'appareil met les données sur le port de communication.

4.2.3 Modélisation de l'OSA avec GRIM

La modélisation de l'OSA avec le GRIMML est possible en principe tant avec un *FunctionBlock* qu'avec un *TransducerBlock*. Par contre, puisque le *TransducerBlock* n'est capable que de faire la mise à jour des différents paramètres, un *FunctionBlock* a été choisi pour réaliser la section *Execute*.

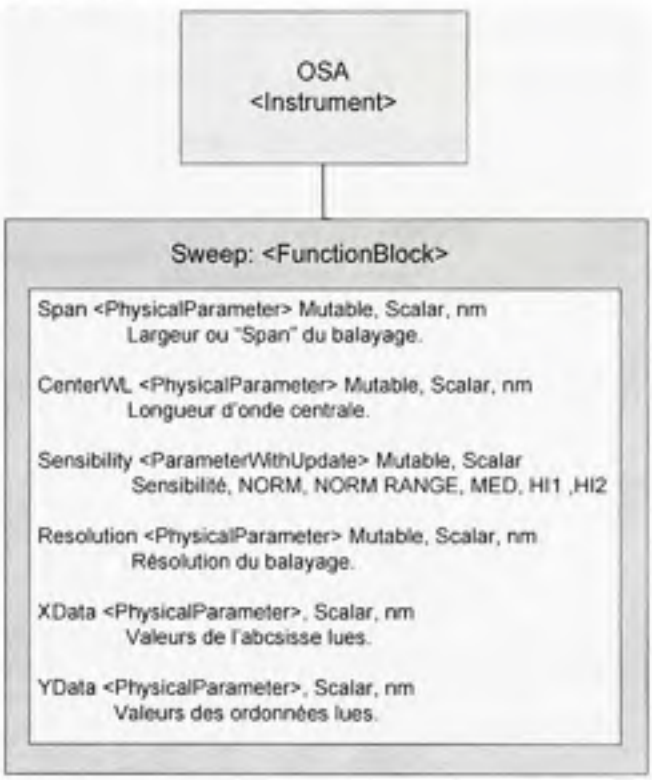

Figure 4.6 ***Modèle GRIM de l'AQ6317B.***

Les différents paramètres nécessaires afin de pouvoir envoyer une fonction SGL à l'appareil sont inclus dans le bloc et présentés à la figure 4.6. Parce que les actions à faire sont complexes et impliquent l'utilisation d'une boucle d'attente, il est nécessaire de coder manuellement ce bloc de contrôle. Le bloc a donc une classe spécifiée dans le fichier GRIMML qui permet de faire les actions requises afin d'acquérir les données lors d'un balayage. Le fichier GRIMML de cet instrument est listé à l'Annexe III.

4.3 Commutateur optique, modèle IQ-1600 de Exfo

Un commutateur optique est un dispositif qui permet de diriger les signaux lumineux vers des canaux de transmission déterminés. Il existe plusieurs types de commutateurs optiques. Certains utilisent des moyens mécaniques qui permettent de faire un réalignement des fibres entre elles ; d'autres dispositifs utilisent la technologie MEMS (Micro Electro-Mechanical Systems) et des micro-miroirs contrôlés par tension afin de faire la commutation. Finalement, certains commutateurs optiques utilisent une conversion Optique-Électrique-Optique pour router les signaux optiques. Le commutateur du laboratoire, le IQ-1600 d'EXFO, est de type mécanique en topologie 1x4, soit une entrée pour 4 sorties.

4.3.1 Principe de fonctionnement

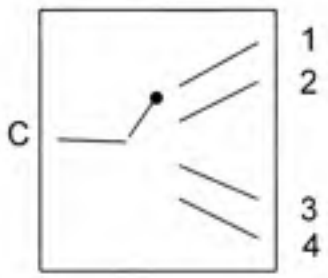

Figure 4.7 ***Principe de fonctionnement d'un commutateur 1xN.***

Le principe de fonctionnement d'un commutateur est assez simple. Il permet de modifier la connectivité entre les différentes fibres. Le type 1xN est le plus simple car il n'a qu'une seule entrée possible. On peut voir sur la figure 4.7 que la fibre d'entrée « Commune » peut-être associée aux fibres de sortie 1, 2 , 3 ou 4.

4.3.2 Communication avec l'IQ-1600

Puisque le fonctionnement du commutateur optique est simple, les commandes à envoyer le sont également. Le commutateur peut être contrôlé lui aussi à partir du port GPIB, à l'adresse

majeure 03h. Il est par contre nécessaire de mettre préalablement l'appareil en mode de contrôle externe en activant le serveur GPIB sur le châssis IQ-200 d'EXFO. L'appareil utilise le format SCPI ; ainsi il est plus facile de communiquer avec n'importe quel appareil de la série IQ-1600 de la même façon, même s'il y a plus d'une entrée / sortie.

Tableau 4.4

Commandes pour communiquer avec l'IQ-1600 d'EXFO

Commandes	Description
ROUTX :SCANX	Permet le commuter vers le canal spécifié
ROUTX :SCAN?	Permet le lire le canal présentement en utilisation

À partir de ces deux commandes, il est finalement possible de créer le modèle GRIM de cet instrument.

4.3.3 Modélisation GRIM

Pour modéliser le commutateur optique, un *FunctionBlock* a été utilisé puisque l'interaction avec le port GPIB est toujours plus facile lorsqu'il est invoqué par une librairie externe en langage C et que le *FunctionBlock* permet d'exécuter du code externe.

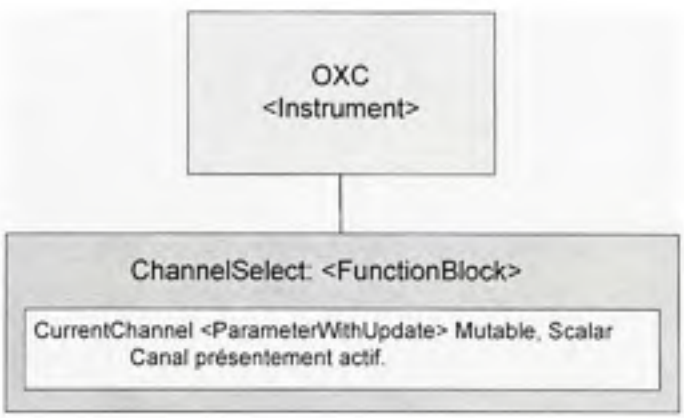

Figure 4.8 Modèle GRIM du *commutateur IQ-1600.*

Le paramètre « CurrentChannel » peut être changé de 1 à 4 selon la sortie à laquelle on veut que le commun soit associé.

Le chapitre 4 nous a permis d'exposer la virtualisation des différents instruments du laboratoire de photonique. Par contre, la virtualisation a des avantages qui sont toujours mieux illustrés lorsqu'il est possible de faire des choses qu'on ne peut pas faire autrement avec les instruments matériels. Le chapitre 5 présente un tel exemple, bien réel, où les appareils virtuels ont permis de résoudre un problème d'engorgement de l'utilisation de l'OSA lors des séances de laboratoire du cours ELE771, Dispositifs photoniques, à l'ÉTS.

CHAPITRE 5

MODÉLISATION ET PARTAGE D'INSTRUMENTS

Dans le cadre du cours ELE-771, Dispositifs photoniques, il fallait que les étudiants enregistrent plusieurs spectres optiques au cours de séances de laboratoires. Malheureusement, puisque les mesures prévues prennent environ une heure à compléter par équipe, il aurait fallu idéalement équiper chacun des 4 postes de travail du laboratoire de son propre analyseur de spectre optique (OSA). Un seul OSA était disponible au laboratoire. Or, lorsqu'un instrument est virtualisé, il peut être utilisé dans un carde de collaboration. Le partage de cet OSA unique a donc été envisagé. La virtualisation de certains des équipements du laboratoire a ainsi rendu possible de faire comme si chaque équipe d'étudiants disposait de son propre OSA à son poste de travail, alors que c'est en fait le même appareil qui est partagé en multiplexant les requêtes des différentes équipes dans le temps. Même si cette solution semble avantageuse à première vue, elle a tout de même nécessité des changements considérables dans le laboratoire, soit au niveau de la disposition physique des instruments. Il a aussi fallu faire l'ajout d'un séquenceur. Afin de voir l'impact sur le temps des laboratoires des mesures de performances ont été prises. Ce chapitre présentera la problématique détaillée, la disposition nécessaire des appareils du laboratoire, le fonctionnement du séquenceur et finalement, les résultats de performance obtenus.

5.1 Problématique

La problématique est présentée en trois partie. La première est le temps d'utilisation de l'OSA lors des séances de laboratoire ; la deuxième est la disposition physique des instruments qu'il faut adopter et finalement, la troisième est le temps requis pour prendre une mesures et le temps d'attente total.

5.1.1 Temps d'utilisation de l'OSA lors des séances de laboratoire

Le tableau 5.1 présente le temps minimum d'utilisation de l'OSA, par équipe d'étudiants, requis pour réaliser les différents laboratoires (d'une durée de 2 heures chacuns) en cours de session. Il s'agit d'un temps minimum, car c'est en fait celui mesuré par le chargé de laboratoire, très expérimenté, lors de ses tests préparatifs de vérification du bon fonctionnement des manipulations. Ce temps comprend notamment :

- le nettoyage et l'inspection des connecteurs,
- la connectorisation des dispositifs sous test à l'OSA,
- la prise de mesures,
- la sauvegarde des spectres sur disquette.

Tableau 5.1

Tableau des temps d'utilisation de l'OSA par équipe lors des laboratoires

Labo 1	**Labo 2**	**Labo 3**	**Labo4**	**Labo 5**
15-20 min	55-80 min	15-20 min	15-20 min	30-45 min

Le tableau 5.2 présente le temps total d'utilisation requis de l'OSA par les 4 équipes que le reste des équipements disponibles peut accommoder. Ce temps est, dans 2 cas sur 5, très supérieur aux 120 minutes allouées par séance de laboratoire ; dans les autres cas, il s'en approche dangereusement. On constate donc que l'accès à l'OSA représente un sérieux goulot d'étranglement pour le bon déroulement des séances de laboratoire.

Tableau 5.2

Tableau des temps d'utilisation totaux de l'OSA

Labo 1	Labo 2	Labo 3	Labo4	Labo 5
80 min	360mins	80 min	80 min	180 min

5.1.2 Disposition physique des instruments

Un des gros problèmes est lorsque les étudiants doivent connecter leurs montages à l'appareil de mesure. Souvent, les montages sont difficiles à déplacer et risquent de briser lorsqu'on les manipule. La solution idéale à ce problème consiste à limiter les déplacements en raccordant tous les postes de travail à l'OSA par l'intermédiaire d'un commutateur optique 4x1 à l'aide de câbles optiques. La figure 5.1 présente la topologie des instruments du laboratoire.

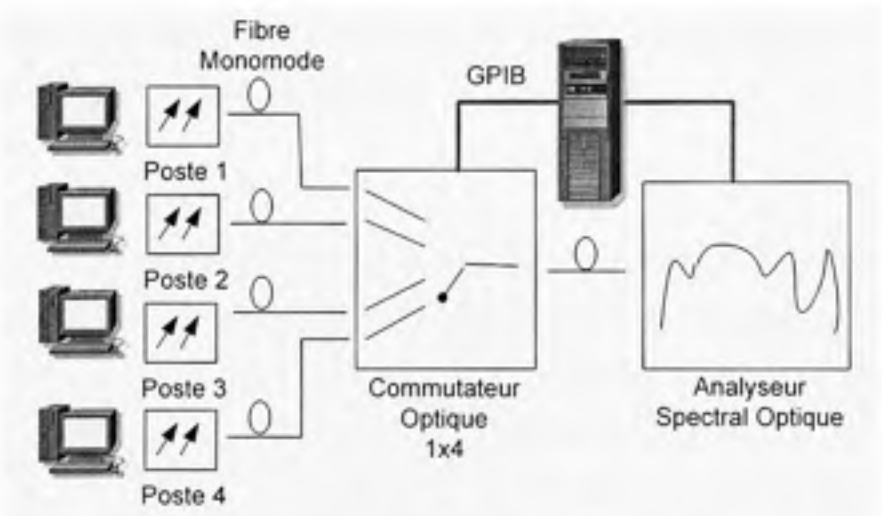

Figure 5.1 ***Disposition physique des instruments.***

Chacun des 4 postes de travail est équipé d'un connecteur identifié « OSA » auquel une fibre monomode est connectée. Chacune de ces différentes fibres est reliée au commutateur optique. Le port commun est relié à l'OSA. Les deux instruments sont reliés, via GPIB, à un

PC qui agit en tant que serveur et chaque étudiant possède un PC qui permet de charger une interface virtuelle de l'OSA.

5.1.3 Temps d'attente et de mesures

Lorsqu'à partir de l'interface virtuelle de l'OSA, l'étudiant commande un balayage, une requête de mesure est envoyée au séquenceur. Celui-ci met cette requête dans une file d'attente. Lorsque l'OSA est libre, le séquence traite la requête de mesure en commutant sur le bon poste et en activant l'OSA. La mesure est alors prise et le spectre est transmis en guise de réponse à l'utilisateur.

Le temps total d'une requête peut être calculé à partir de l'équation suivante :
Délais de commutation + Temps de balayage de l'OSA + Temps de transmission GPIB

Ainsi, si le délai de commutation est de 1 seconde, le temps de balayage de l'OSA à sensibilité Normale est d'environ 12 secondes et le temps de transmission GPIB est de 6 secondes, on obtient un temps de mesure de :
1s + 12s + 6s = 19s

Le temps de traitement minimal théorique est donc de 19 secondes. Si tout le monde est en mode « Repeat » de balayages successifs et que la queue de requête est pleine, on a dans le pire des cas un temps d'attente de 76 s entre chaque rafraîchissement. Ceci est très acceptable même s'il n'est pas aussi « temps réel » qu'on le désirerait. On pourrait toujours ajouter un deuxième OSA qui réduirait le temps d'attente maximal à 38 s entre chaque rafraîchissement. Toutefois, statistiquement, les utilisateurs font des manipulations entre les prises de mesures et donc le temps d'attente moyen correspond à peu près le temps d'attente maximal divisé par deux, soit environ de 38 s. On constate alors que ce temps correspond presque au temps de rafraichissement visuel de l'OSA ; en effet, le balayage n'a pas besoin d'afficher les traces et on obtient des performances similaires à celles qui sont obtenues lors des manipulations réelles de l'appareil, tout en ayant dans ce cas 4 utilisateurs en même temps.

5.2 Fonctionnement du séquenceur à jeton

Le séquenceur est l'aspect le plus important de cet exemple d'application d'une virtualisation puisqu'il est d'une importance cruciale lors du partage de ressources, afin de s'assurer qu'une seule personne soit capable d'accéder à la ressource à la fois.

Puisque chaque étudiant doit à tour de rôle avoir accès l'OSA, un séquenceur à jeton était nécessaire. Il existe plusieurs types de séquenceurs ayant des performances optimales et qui sont capables de procéder au blocage des ressources, comme par exemple le séquenceur à algorithmes génétiques. Par contre, le séquenceur à jeton était le plus facile à réaliser dans des délais raisonnables. Cette section couvre le fonctionnement de ce type de séquenceur.

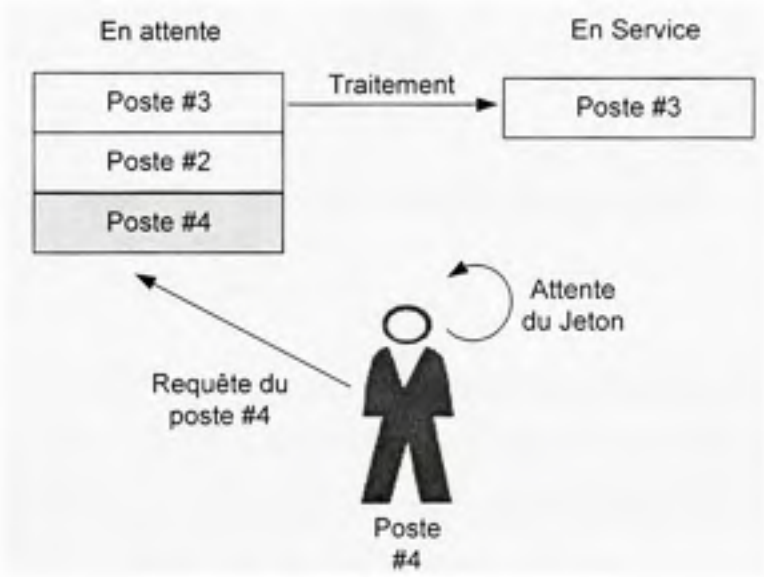

Figure 5.2 ***Requête au séquenceur.***

Lorsqu'un étudiant veut prendre une mesure, il fait une requête pour obtenir le jeton de contrôle. Cette requête est mise dans une file d'attente de type FIFO (First In First Out). La

figure 5.2 illustre la première étape : lorsqu'un étudiant du poste 4 envoie une requête, il reste en attente jusqu'à ce que toutes les requêtes antérieures à la sienne aient été traitées.

Lorsque ce poste est en traitement, il reçoit le jeton (Figure 5.3) : il a dès lors l'exclusivité d'utilisation des ressources contrôlées par le séquenceur.

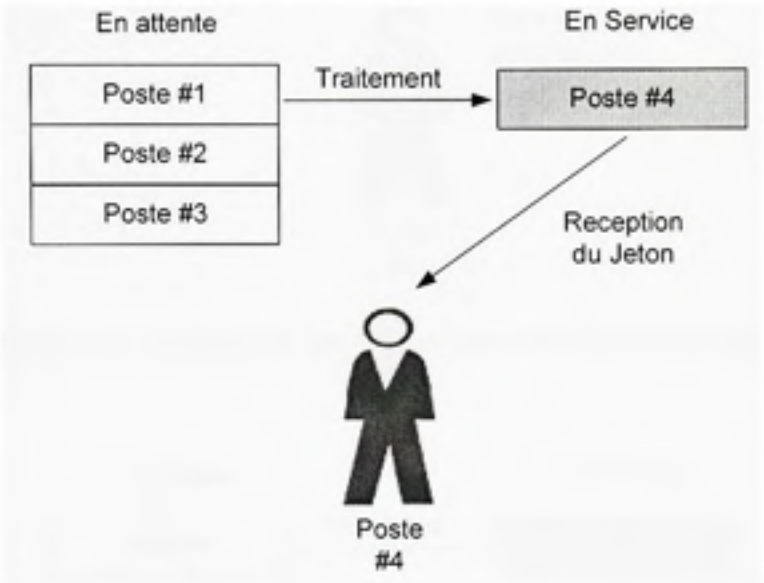

Figure 5.3 ***Réception du jeton du séquenceur.***

Ainsi, le poste 4 peut effectuer la mesure en utilisant les ressources externes (Figure 5.4) nécessaires (OSA, commutateur). Il fait ensuite l'acquisition des données de cette mesure et doit retourner le jeton (Figure 5.5) au séquenceur afin que la prochaine requête soit traitée.

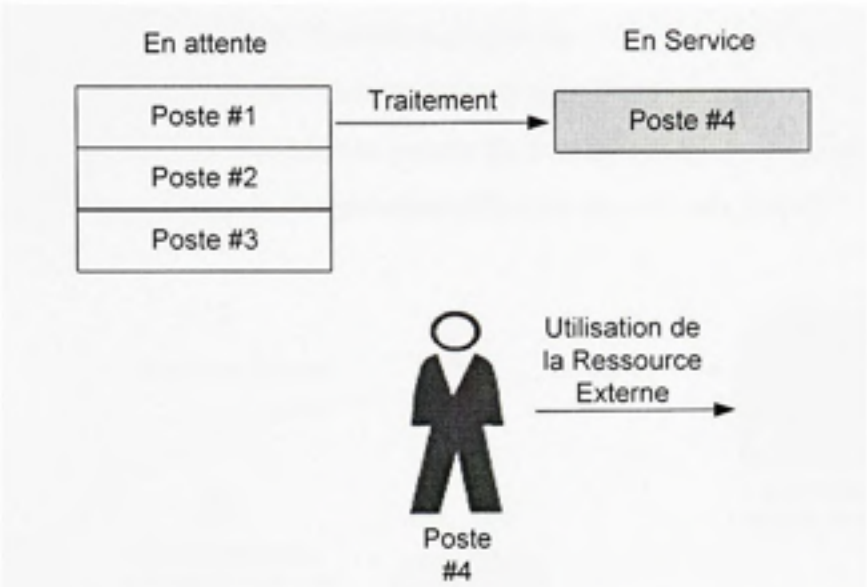

Figure 5.4 ***Utilisation des ressources externes par un poste.***

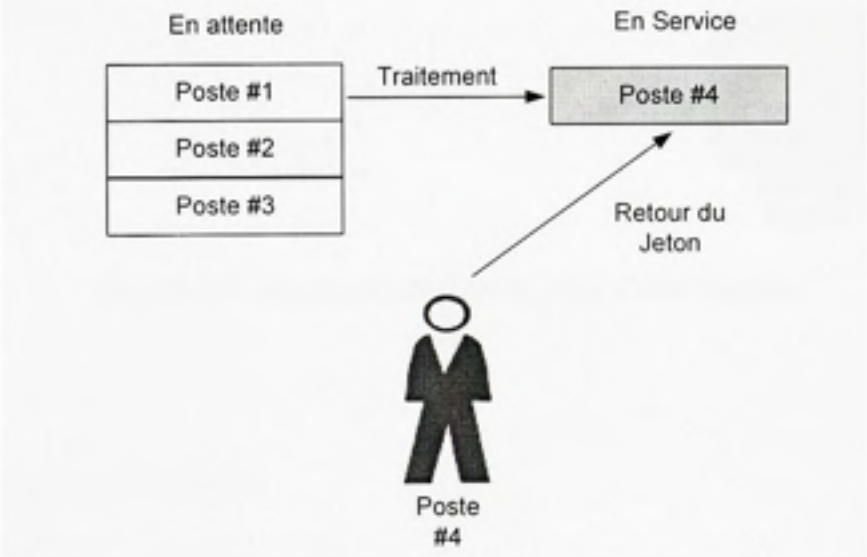

Figure 5.5 ***Retour du jeton après utilisation.***

5.3 Flux des processus

Une fois les données reçues, elles sont traitées dans l'interface graphique selon le flux de processus (*voir* Figure 5.6) et affichées sur un graphique cartésien. Même si la majorité du

travail est effectué par les services, l'interface graphique a un rôle important à jouer : elle doit représenter fidèlement l'aspect de l'instrument partagé. Dans ce cas-ci, le séquenceur et le commutateur optique doivent être cachés puisqu'ils sont nécessaires uniquement au scénario de partage et ne font pas partie de l'expérience effectuée dans le laboratoire.

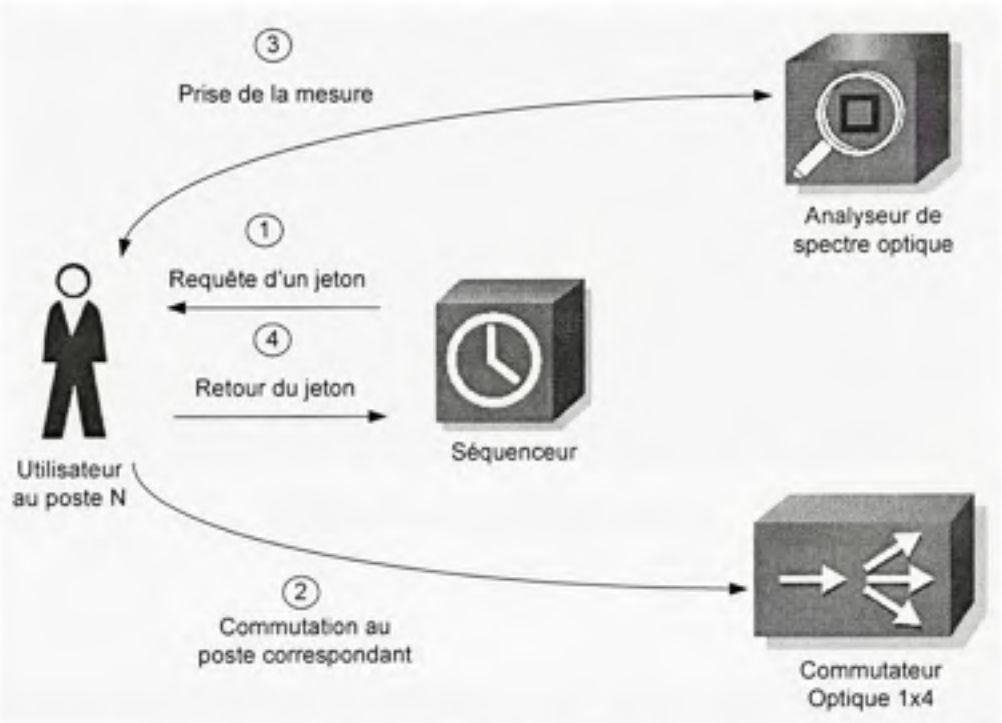

Figure 5.6 ***Processus lors de la prise d'une mesure.***

5.4 Interface graphique

L'interface graphique, présentée à la figure 5.7, permet d'interagir avec l'instrument partagé, soit l'OSA. Lors du démarrage de l'application Java, le numéro du poste auquel l'étudiant est assis est utilisé en arrière-plan afin d'envoyer les requêtes au séquenceur et aussi afin de connecter adéquatement la bonne fibre optique vers l'OSA.

L'interface d'utilisation permet de changer les paramètres fréquents (du modèle GRIM), soient la longueur d'onde centrale et la largeur de spectre. Elle peut toutefois faire le le calcul de ces valeurs à partir d'une longueur d'onde de début et de fin de balayage.

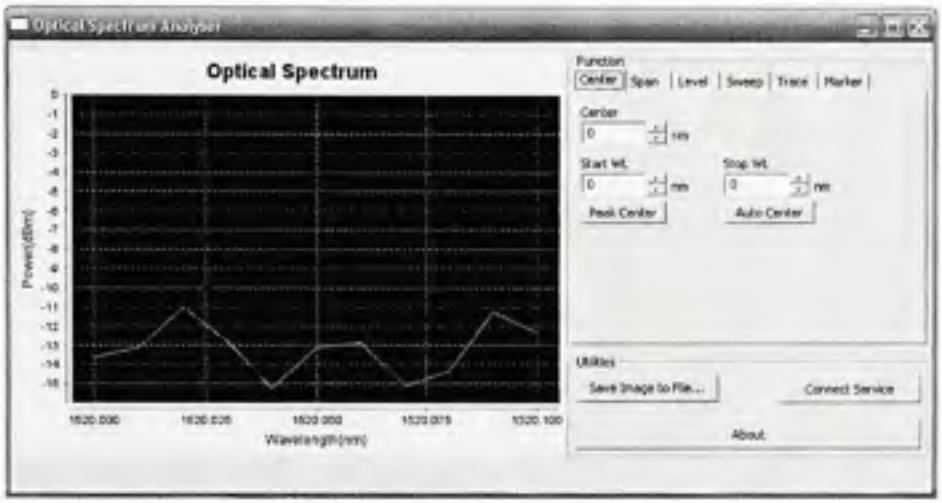

Figure 5.7 ***Interface graphique.***

Il est possible de sauvegarder les données sous format image (.JPG) ou bien en format Excel (.XLS) pour les utiliser par la suite à l'externe. Il est aussi possible d'avoir plusieurs traces différentes sur le même graphique et de faire une soustraction ou une addition des traces A et B en tant que trace C.

5.5 Résultats et performances

Afin d'évaluer les performances de la nouvelle disposition et de l'automatisation des laboratoires, il est nécessaire de faire trois versions des tests de mesures et de déterminer l'intervalle moyen de temps entre 2 mesures selon le nombre de postes actifs et dans différentes conditions. Afin d'éliminer des paramètres qui pourraient varier, on a choisi de procéder à 10 mesures avec des paramètres prédéterminés (SPAN, CENTERWL, RESOLUTION, SENSIBILITÉ). Le tableau 5.3 résume ces échantillons de mesures. Ceci a

permis de trouver le temps moyen nécessaire à la prise d'une mesure, à partir du moment où l'instrument de mesure est libre, jusqu'à ce qu'il soit libéré à nouveau.

Tableau 5.3

Tableau des séries de mesures à prendre lors de l'expérimentation

Mesure	Span	Longueur d'onde centrale	Résolution	Sensibilité
1	100nm	1550nm	0.1nm	NORM HD
2	80nm	1550nm	1nm	MED
3	50nm	1310nm	0.5nm	HIGH 1
4	5nm	980nm	2nm	NORM AT
5	40nm	1310nm	0.2nm	MED
6	20nm	980nm	1nm	HIGH 1
7	90nm	1550nm	0.5nm	HIGH 2
8	10nm	980nm	2nm	NORM HD
9	200nm	1310nm	0.1nm	MED
10	70nm	1550nm	0.2nm	HIGH 3

La première série de mesures est faite en se plaçant en file d'attente devant l'OSA et en ayant le montage directement branché à l'OSA : l'étudiant doit prendre la mesure et sauvegarder les données sur une disquette. La deuxième série est réalisée alors que les étudiants sont installés à leur à poste de mesure : à tour de rôle, chaque équipe a l'autorisation d'aller sélectionner sa fibre au commutateur optique, puis d'aller ajuster les paramètres de l'OSA pour faire sa mesure et enfin doit recueillir les données sur une disquette. Finalement, la dernière série est faite à partir du logiciel uniquement, alors que les étudiants sont à leur poste de travail respectif.

5.5.1 Mesures manuelles directement à l'OSA

Les mesures manuelles ont été effectuées avec 4 personnes placées en file d'attente. Chaque personne devait prendre une mesure avec des paramètres différents et sauvegarder ses résultats sur une « disquette ».

Tableau 5.4

Tableau des temps de mesure manuels lorsque connecté directement à l'OSA (en secondes)

Mesure	Série 1	Série 2	Série 3	Série 4
1	71	66	69	63
2	94	61	154	58
3	92	64	72	60
4	71	65	84	85
5	100	77	56	57
6	77	87	68	72
7	127	128	138	191
8	67	64	58	59
9	80	72	70	75
10	245	266	247	241
MIN	56			
1er QUARTILE	64.75			
MEDIANE	72			
3e QUARTILE	95.5			
MAX	266			

On peut voir sur la figure 5.8 que la médiane du temps de mesure est d'environ 72s pour les quatre séries de mesures effectuées. Par contre, on voit que si les séries 2 et 4 ont des résultats similaires, les séries 1 et 3 ont des durées beaucoup plus importantes. Ceci est entre autre dû à une erreur de mesure à la mesure 2 lors de la série 3 et à un manque d'expérience lors de la série 1. Les résultats ne furent pas rejetés, même s'ils ne sont pas entièrement reproductibles, puisque dans un environnement de laboratoire les étudiants inexpérimentés risquent de faire de telles erreurs de manipulation.

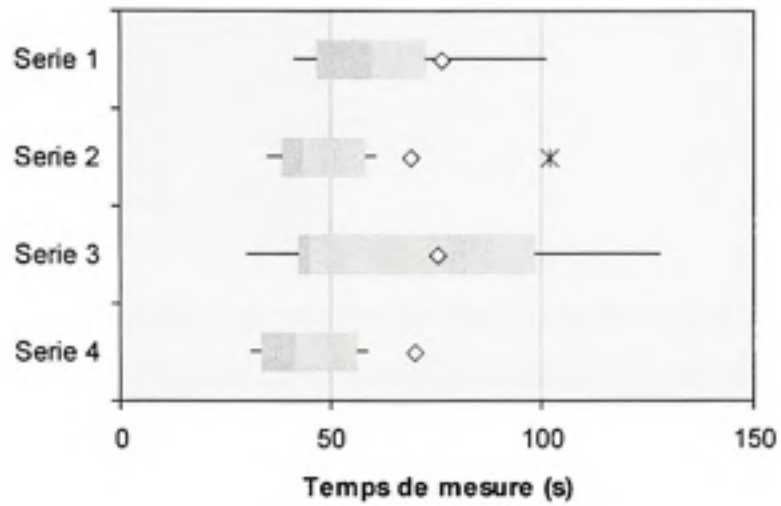

Figure 5.8 ***Boîtes à moustaches des temps de mesure de la manipulation 1.***

5.5.2 Mesures manuelles à partir du poste de travail

Les mesures manuelles du Tableau 5.5 ont été effectuées lorsque chaque étudiant devait aller sélectionner son poste au commutateur et ensuite prendre la mesure à l'OSA tout en sauvegardant le résultat sur disquette.

Tableau 5.5

Tableau des temps de mesure manuelles à partir du poste de travail (en secondes)

Mesure	Série 1	Série 2	Série 3	Série 4
1	63	80	77	70
2	70	75	68	66
3	65	71	71	68
4	72	81	72	75
5	65	72	60	63
6	80	70	83	69
7	144	147	142	148
8	89	72	68	77
9	86	102	81	77

10	257	258	243	269

MIN	60
1er QUARTILE	69.75
MEDIANE	75
3e QUARTILE	86.75
MAX	269

On peut voir sur la figure 5.9 que la médiane du temps de mesure est d'environ 75s pour les quatre séries de mesures effectuées. Contrairement à la figure 5.8, on voit ici que les résultats des différentes séries sont similaires et constants. Par contre, on peut aussi remarquer que les résultats des mesures manuelles à partir du poste de travail sont très similaires aux résultats des mesures manuelles prises directement à l'OSA et que le fait de devoir aller sélectionner le poste sur le commutateur ne représente pas une fraction significative du temps de mesure total.

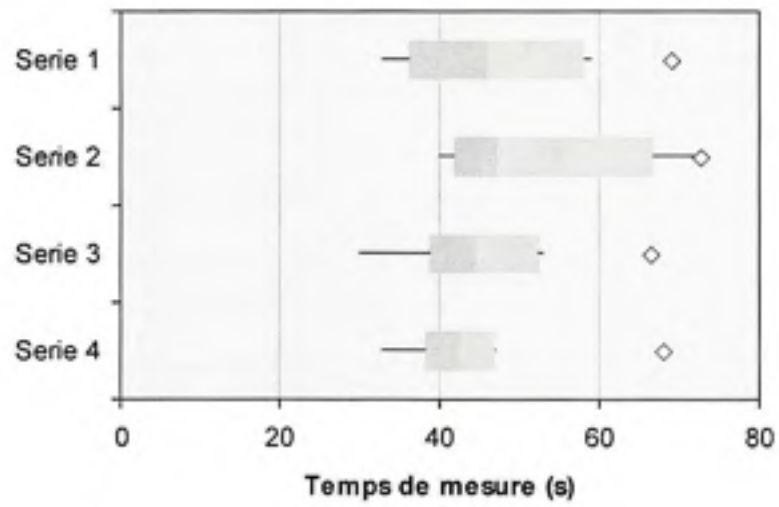

Figure 5.9 ***Boîtes à moustaches des temps de mesure de la manipulation 2.***

5.5.3 Mesures logicielles à partir du poste de travail.

Les mesures logicielle (ou automatisées) du Tableau 5.6 ont été prises avec JAMON, qui est un outil qui permet de faire des tests de performances en Java suite à des changements dans l'application d'interface graphique et reflètent le temps de prise de mesure et le temps d'attente en file d'attente.

Tableau 5.6

Tableau des temps de mesure logicielle à partir du poste de travail (en secondes)

Mesure	Série 1	Série 2	Série 3	Série 4
1	23	24	23	24
2	31	31	31	31
3	28	28	28	28
4	34	33	34	34
5	29	30	30	31
6	36	36	36	37
7	94	95	94	95
8	29	28	29	29
9	42	43	42	42
10	210	211	210	210
MIN	23			
1er QUARTILE	29			
MEDIANE	32			
3e QUARTILE	42			
MAX	211			

On peut voir sur la Figure 5.10 que la médiane du temps de mesure est d'environ 32s pour les quatre séries de mesures effectuées. Ceci est une amélioration substantielle qui diminue le temps d'attente de plus de la moitié par rapport aux prises de mesures manuelles. De plus, on peut voir qu'ici les résultats sont reproductibles et que les erreurs de manipulations dans le logiciel n'affectent pas le temps passé devant l'OSA. Ceci est dû au fait que tous les

paramètres sont envoyés en même temps et que ces paramètres sont choisis par l'utilisateur avant qu'il appuie sur le bouton commandant le balayage.

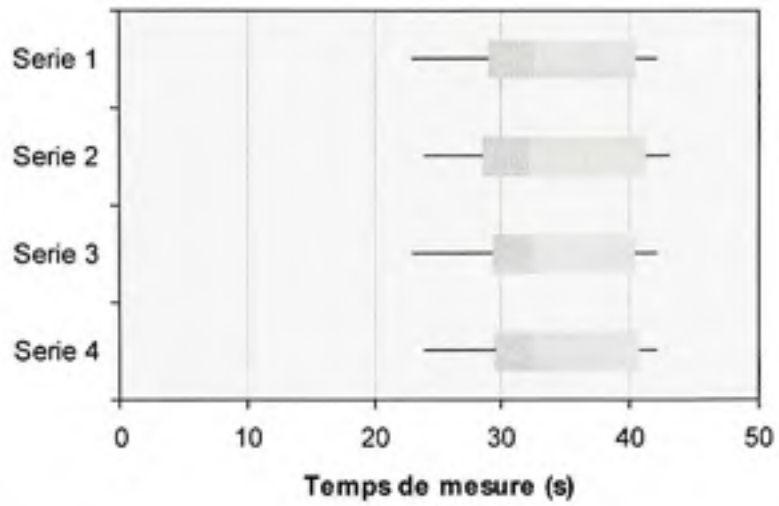

Figure 5.10 ***Boîtes à moustaches des temps de mesures de la manipulation 3.***

On observe donc un gain de temps substantiel par mesure (33 s en moyenne). Ce gain de temps doit de plus être multiplié par le nombre d'équipes d'étudiants présentes dans la file d'attente au moment de la mesure, puisque ces équipes sont toutes affectées par les délais de mesure. Avec l'exemple simple d'une équipe prenant sa mesure et d'une équipe attendant dans la file d'attente, c'est déjà plus d'une minute complète de temps d'utilisation de l'OSA qui est gagnée.

5.6 Analyse et conclusions

En présence d'un seul utilisateur de l'OSA, l'utilisation du logiciel de partage d'instrument n'a pas beaucoup d'impact. En effet, la seule raison pour laquelle le laboratoire photonique virtualisé à l'aide du GRIM est plus performant pour ce cas est que le temps requis pour

transférer les données sur une disquette est éliminé; les données sont donc disponibles plus rapidement pour un traitement externe. Par contre, le gain devient de plus en plus important à mesure que le nombre d'équipes d'étudiants en file d'attente augmente. Ceci est dû en majeure partie au fait que chaque étudiant doit, dans un environnement dépourvu de partage, brancher et débrancher son montage à l'OSA, tandis que cette opération est faite au poste de travail dans un environnement partagé. De plus, lorsque l'OSA est partagé, le temps total est directement lié au temps requis par mesure et au nombre d'utilisateurs dans la file d'attente, avec un maximum de 4 utilisateurs. Ainsi, les facteurs humains et les déplacements nécessaires lors des prises de mesures manuelles, qui ralentissent de façon considérable le processus au point où il faut presque 10 minutes avant qu'une autre équipe puisse prendre une prochaine mesure, n'affectent plus la disponibilité des mesures et le goulot d'étranglement est éliminé.

Le chapitre 5 a démontré que la virtualisation peut s'avérer un outil important pour l'enseignement, puisqu'il diminue de façon considérable les dépenses en capital requises pour équiper les laboratoires en instrument dernier cri grâce au partage d'instruments. Par contre, l'interface doit ressembler fidèlement à l'instrument, afin de représenter adéquatement chaque paramètre spécifique et d'offrir aux étudiants un apprentissage réel de l'utilisation du type d'instrument étudié.

CONCLUSION

Dans ce mémoire, on a vu comment, grâce au modèle GRIM, il était possible de raccorder les instruments aux Grids. Un résumé des notions nécessaires à la compréhension des services Web a été présenté dans le cadre du chapitre 1. Par la suite, au chapitre 2, on a vu comment les services Grid permettaient de faire l'extension du service sans état vers un service avec état appelé ressources. On a aussi vu la sécurité dans les Grids et son importance dans un environnement collaboratif. Le chapitre 3 a permis d'aller dans les détails du modèle GRIM et de voir les structures de données et de ressources de base utilisées lors de la création initiale du modèle. Ensuite, dans le chapitre 4, trois instruments du laboratoire de photonique de l'ÉTS ont été présentés et modélisés grâce au GRIMML, syntaxe XML permettant la création facile d'un modèle de ressources GRIM pour des instruments particuliers. Finalement, au chapitre 5 un exemple de cas a été présenté afin de résoudre un problème logistique lors des séances de laboratoire du cours ELE771 - Dispositifs Photoniques de l'ÉTS. Ceci a permis de voir quels sont les gains considérables de la virtualisation dans des laboratoires d'enseignement.

Les retombées principales de cette recherche ont été la création de l'intergiciel GRIM qui pourra être utilisé par de nombreux projets. Des deux phases présentées à la Figure 3.3 de l'implémentation du GRIM, seule la première a été réalisée de façon à pouvoir être réutilisée. On a aussi pu prouver que le modèle de données riche offert par le IEEE 1451.1 peut être traduit sous forme de ressources Grid afin d'offrir une solution aussi versatile et « Plug and Play » aux individus désirant connecter leur instruments aux Grids. En effet, le GRIMML permet de rapidement créer de nouveaux modèles d'instrument. Tous les instruments ont alors exactement la même représentation logicielle pour les différentes applications. Ainsi une application capable de contrôler un OSA pourrait tout aussi bien contrôler un autre type d'instrument. Par contre, même si les structures de données sont uniformes, les analyses de celles-ci ne sont pas triviales et de nouveaux services doivent être développés pour faire le pré-traitement des données.

La phase deux qui consiste à utiliser le modèle GRIM dans des applications a été abordée dans le chapitre 5, à ceci près que la solution est personnalisée au laboratoire de photonique et n'est pas réutilisable. Par contre, un nouveau projet du nom de AIRON a été mis en place au Centre de Recherche en Communications (CRC, Ottawa) afin d'utiliser le GRIM pour créer des réseaux intelligents adaptatifs. De plus, en collaboration avec l'Institut des Technologies de Université de l'Ontario (UOIT), une adaptation du GRIM pour les capteurs sans fil de type « Motes » sera élaborée dans le cadre d'une recherche financée par la Fondation Canadienne pour l'Innovation (FCI).

RECOMMANDATIONS

Il reste par contre plusieurs aspect du GRIM à améliorer.L'un de ces aspects est d'étudier davantage le séquenceur. Il existe plusieurs types de séquenceur et un meilleur séquenceur pourrait donner de meilleurs résultats. Un autre aspect serait aussi d'intégrer le GRIM à d'autre types d'intergiciels existants, comme le User Controlled LightPaths (UCLP) décrit par (Wu, 2004). Au doctorat, je compte unifier ces différents intergiciels (middleware) afin d'obtenir un environnement distribué et flexible unique. Afin de pouvoir obtenir une telle architecture orientée service, les objectifs spécifiques seront d'étendre la virtualisation aux éléments de réseaux non couverts par les intergiciels actuels comme les routeurs ou les commutateurs cablés et d'y ajouter l'interaction avec le GRIM pour couvrir la virtualisation des instruments. Il faudrait aussi améliorer la gestion des comptes d'utilisateur, afin d'avoir une interface qui permette de créer de nouveaux utilisateurs. Finalement une interface riche (Rich Client Interface) de type *LabView* sera nécessaire afin de pouvoir facilement orchestrer et réutiliser le modèle.

ANNEXE I

FICHIERS SCHEMAS DES RESSOURCES GRIM

Fichier Schema des resources *Parameter*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Mathieu Lemay (ITI) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.inocybe.ca/GRIM/2006/01/CommonData"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:import namespace="http://www.inocybe.ca/GRIM/2006/01/CommonData"
schemaLocation="C:\GRIM\gt4devel\schema\grim\core\common\CommonData.xsd"/>
 <xs:element name="read">
  <xs:annotation>
   <xs:documentation>Input Message for Read Operation </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="NumValues" type="xs:int">
     <xs:annotation>
      <xs:documentation>Number of values to read</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="write">
  <xs:annotation>
   <xs:documentation>Input Message for Write Operation </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Values"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="readResponse">
  <xs:annotation>
   <xs:documentation>Output Message for Read Operation</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="values" type="xs:string"/>
    <xs:element ref="ns1:OperationReturnCode"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="writeResponse">
```

```
  <xs:annotation>
   <xs:documentation>Output Message for Write Operation</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="ns1:OperationReturnCode"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="ParameterRP">
  <xs:annotation>
   <xs:documentation>Resource Properties for the Parameters</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Name of the Parameter Resource.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="description" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Long description of the parameter will be shown in the help section.
</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="interpretation" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Parameter interpretation / usage. </xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="buffered" type="xs:boolean">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Boolean that determines if the data is buffered or real-time.
</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="dataType" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en">XML Schema datatype for the data field. </xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="mutable" type="xs:boolean" nillable="false">
     <xs:annotation>
      <xs:documentation xml:lang="en">Boolean value that specifies if the parameter value is writable.
</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="bufferSize">
     <xs:annotation>
      <xs:documentation>Size of the buffer if the values are buffered.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="delimiter">
```

```
      <xs:annotation>
        <xs:documentation>The delimiter is the character that separates buffered values.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="value" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation xml:lang="en">Data information separated by delimiter if buffered. The number of
elements determine the dimensionality.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:attribute name="type" type="xs:anySimpleType"/>
        <xs:attribute name="formatReference" type="xs:string"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="notifyOnChange" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>This flag will trigger notifications if required</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="MaxValue" type="xs:string">
      <xs:annotation>
        <xs:documentation>The maximum value that the value field can contain.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="MinValue" type="xs:string">
      <xs:annotation>
        <xs:documentation>The minimum value taht the value field can have.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Fichier Schema des resources *ParameterWithUpdate*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Mathieu Lemay (ITI) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/ParameterWithUpdateResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="ParameterWithUpdate" type="ParameterWithUpdateType">
  <xs:annotation>
   <xs:documentation>Resource Properties for the Parameters</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:complexType name="valueType">
  <xs:annotation>
   <xs:documentation>Type for value fields.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="type" type="xs:anySimpleType"/>
  <xs:attribute name="formatReference" type="xs:string"/>
 </xs:complexType>
 <xs:complexType name="ParameterWithUpdateType">
  <xs:sequence>
   <xs:element name="name" type="xs:string">
    <xs:annotation>
     <xs:documentation xml:lang="en"> Name of the Parameter Resource.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="description" type="xs:string">
    <xs:annotation>
     <xs:documentation xml:lang="en"> Long description of the parameter will be shown in the help section.
</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="interpretation" type="xs:string">
    <xs:annotation>
     <xs:documentation xml:lang="en"> Parameter interpretation / usage. </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="buffered" type="xs:boolean">
    <xs:annotation>
     <xs:documentation xml:lang="en"> Boolean that determines if the data is buffered or real-time.
</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="dataType" type="xs:string">
    <xs:annotation>
     <xs:documentation xml:lang="en">XML Schema datatype for the data field. </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="mutable" type="xs:boolean" nillable="false">
    <xs:annotation>
     <xs:documentation xml:lang="en">Boolean value that specifies if the parameter value is writable.
</xs:documentation>
    </xs:annotation>
   </xs:element>
```

```
    <xs:element name="bufferSize">
     <xs:annotation>
      <xs:documentation>Size of the buffer if the values are buffered.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="delimiter">
     <xs:annotation>
      <xs:documentation>The delimiter is the character that separates buffered values.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="data" type="valueType" maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation xml:lang="en">Data information separated by delimiter if buffered. The number of
elements determine the dimensionality.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="notifyOnChange" type="xs:boolean">
     <xs:annotation>
      <xs:documentation>This flag will trigger notifications if required</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="maxValue" type="xs:string">
     <xs:annotation>
      <xs:documentation>The maximum value that the value field can contain.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="minValue" type="xs:string">
     <xs:annotation>
      <xs:documentation>The minimum value taht the value field can have.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="oldData" type="valueType" maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>The old value before update was performed.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="lastUpdate" type="xs:dateTime">
     <xs:annotation>
      <xs:documentation>Timestamp of the last update.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="blocked" type="xs:boolean">
     <xs:annotation>
      <xs:documentation>The resource is wating to be updated.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Fichier Schema des resources *PhysicalParameter*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Developper (IT1) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.inocybe.ca/GRIM/2006/01/CommonData"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:import namespace="http://www.inocybe.ca/GRIM/2006/01/CommonData"
schemaLocation="C:\GRIM\gt4devel\schema\grim\core\common\CommonData.xsd"/>
 <xs:element name="read">
  <xs:annotation>
   <xs:documentation>Input Message for Read Operation </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="NumValues" type="xs:int">
     <xs:annotation>
      <xs:documentation>Number of values to read</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="write">
  <xs:annotation>
   <xs:documentation>Input Message for Write Operation </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Values"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="readResponse">
  <xs:annotation>
   <xs:documentation>Output Message for Read Operation</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="values" type="xs:string"/>
    <xs:element ref="ns1:OperationReturnCode"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="writeResponse">
  <xs:annotation>
   <xs:documentation>Output Message for Write Operation</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="ns1:OperationReturnCode"/>
   </xs:sequence>
```

```
  </xs:complexType>
 </xs:element>
 <xs:element name="PhysicalParameterRP">
  <xs:annotation>
   <xs:documentation>Resource Properties for the Parameters</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Name of the Parameter Resource.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="description" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Long description of the parameter will be shown in the help section.
</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="interpretation" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Parameter interpretation / usage. </xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="buffered" type="xs:boolean">
     <xs:annotation>
      <xs:documentation xml:lang="en"> Boolean that determines if the data is buffered or real-time.
</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="dataType" type="xs:string">
     <xs:annotation>
      <xs:documentation xml:lang="en">XML Schema datatype for the data field. </xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="mutable" type="xs:boolean" nillable="false">
     <xs:annotation>
      <xs:documentation xml:lang="en">Boolean value that specifies if the parameter value is writable.
</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="bufferSize">
     <xs:annotation>
      <xs:documentation>Size of the buffer if the values are buffered.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="delimiter">
     <xs:annotation>
      <xs:documentation>The delimiter is the character that separates buffered values.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="value" maxOccurs="unbounded">
     <xs:annotation>
```

```
        <xs:documentation xml:lang="en">Data information separated by delimiter if buffered. The number of
elements determine the dimensionality.</xs:documentation>
       </xs:annotation>
       <xs:complexType>
        <xs:attribute name="type" type="xs:anySimpleType"/>
        <xs:attribute name="formatReference" type="xs:string"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="notifyOnChange" type="xs:boolean">
       <xs:annotation>
        <xs:documentation>This flag will trigger notifications if required</xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="maxValue" type="xs:string">
       <xs:annotation>
        <xs:documentation>The maximum value that the value field can contain.</xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="minValue" type="xs:string">
       <xs:annotation>
        <xs:documentation>The minimum value taht the value field can have.</xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="units">
       <xs:annotation>
        <xs:documentation>Physical units</xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="lastUpdate" type="xs:dateTime">
       <xs:annotation>
        <xs:documentation>Timestamp of the last update.</xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="blocked" type="xs:boolean">
       <xs:annotation>
        <xs:documentation>The resource is wating to be updated.</xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="oldData" maxOccurs="unbounded">
       <xs:annotation>
        <xs:documentation>The old value before update was performed.</xs:documentation>
       </xs:annotation>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:schema>
```

Fichier Schema des ressources *Actions*

```
<xs:element name="ActionRP">
  <xs:annotation>
    <xs:documentation>Action Resource Properties Document</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TransactionID" type="xs:string">
        <xs:annotation>
          <xs:documentation>This value identifies a particular invocation of the Action.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="TimeoutDuration" type="xs:time">
        <xs:annotation>
          <xs:documentation>This value is the time duration after which the Action is considered to have failed. A
value of 0 indicates that the timeout is infinite.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="State">
        <xs:annotation>
          <xs:documentation>Reflects the current state of the Action.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Idle"/>
            <xs:enumeration value="Executing"/>
            <xs:enumeration value="Completed"/>
            <xs:enumeration value="Failed"/>
            <xs:enumeration value="Other"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Progress">
        <xs:annotation>
          <xs:documentation>If the action is executing this field can return the progress of the
execution.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:float">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Rules"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fichier Schema des codes d'erreurs

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Developper (ITI) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.inocybe.ca/GRIM/2006/01/CommonData"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/CommonData" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="OperationReturnCode">
  <xs:annotation>
   <xs:documentation>Operation Return Codes based on IEEE1451</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:choice>
    <xs:element name="Complete" type="xs:string" nillable="true">
     <xs:annotation>
      <xs:documentation>Operation completed without errors</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="NOPOperation" type="xs:string" nillable="true">
     <xs:annotation>
      <xs:documentation>The operation is nulled and is only provided for compatibility purposes (Empty
Operation)</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="FailedNonSpecific" type="xs:string" nillable="true">
     <xs:annotation>
      <xs:documentation>The reason of the failure is unknown</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="CommunicationError" type="xs:string" nillable="true">
     <xs:annotation>
      <xs:documentation>A communication error has occured that prevented execution</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="Busy" type="xs:string">
     <xs:annotation>
      <xs:documentation>The resources involved are busy.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="IllegalOperation" type="xs:string">
     <xs:annotation>
      <xs:documentation>The operation tried to perform an illegal procedure</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="FailedInput" type="xs:string">
     <xs:annotation>
      <xs:documentation>The input arguments are not valid.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="FailedOutput" type="xs:string">
     <xs:annotation>
      <xs:documentation>The output arguments are not valid.</xs:documentation>
     </xs:annotation>
    </xs:element>
```

```
    <xs:element name="OperationInterrupted" type="xs:string">
     <xs:annotation>
      <xs:documentation>The operation has been interrupted.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="OperationTimeout" type="xs:string">
     <xs:annotation>
      <xs:documentation>The operation has timedout.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="InsufficientResources" type="xs:string">
     <xs:annotation>
      <xs:documentation>Inssuficient resources were found to do the operation.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Fichier Schema des ressources *Instrument*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Developper (ITI) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.inocybe.ca/GRIM/CommonData"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:import namespace="http://www.inocybe.ca/GRIM/CommonData"
schemaLocation="C:\Projects\GRIM\CommonData.xsd"/>
 <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
schemaLocation="C:\Projects\GRIM\ws\addressing\WS-Addressing.xsd"/>
 <xs:element name="InstrumentResourceProperties">
  <xs:annotation>
   <xs:documentation>Instrument Resource Properties </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="BlockState" type="ns1:BlockProperties">
     <xs:annotation>
      <xs:documentation>State for the NCAP</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="SystemInformation">
     <xs:annotation>
      <xs:documentation>Information about the System Implementation</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="ManufacturerName" type="xs:string">
        <xs:annotation>
         <xs:documentation>Hardware Manufacturer Name</xs:documentation>
        </xs:annotation>
       </xs:element>
```

```
    <xs:element name="ModelNumber" type="xs:string">
     <xs:annotation>
      <xs:documentation>Model Number for the System</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="SerialNumber" type="xs:string">
     <xs:annotation>
      <xs:documentation>Serial Number f</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="OSVersion" type="xs:string">
     <xs:annotation>
      <xs:documentation>Operating System</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="GRIMModel">
     <xs:annotation>
      <xs:documentation>Information about GRIM Model</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Date" type="xs:date">
        <xs:annotation>
         <xs:documentation>GRIM Model Date</xs:documentation>
        </xs:annotation>
       </xs:element>
       <xs:element name="Version" type="xs:string">
        <xs:annotation>
         <xs:documentation>GRIM Model Version</xs:documentation>
        </xs:annotation>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="BlockList">
  <xs:annotation>
   <xs:documentation>List of Blocks</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Instrument">
     <xs:annotation>
      <xs:documentation>Instrument Agent</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Name" type="xs:string">
        <xs:annotation>
         <xs:documentation>Name/Alias for the Instrument</xs:documentation>
        </xs:annotation>
       </xs:element>
```

```
            <xs:element name="Description" type="xs:string">
             <xs:annotation>
              <xs:documentation>Description of the instrument</xs:documentation>
             </xs:annotation>
            </xs:element>
            <xs:element name="InstrumentEPR" type="wsa:EndpointReferenceType"
maxOccurs="unbounded">
             <xs:annotation>
              <xs:documentation>Agent EPR</xs:documentation>
             </xs:annotation>
            </xs:element>
            <xs:element name="GeoLocation">
             <xs:annotation>
              <xs:documentation>Geographical Location</xs:documentation>
             </xs:annotation>
             <xs:complexType>
              <xs:sequence>
               <xs:element name="Latitude">
                <xs:simpleType>
                 <xs:restriction base="xs:float">
                  <xs:minInclusive value="-90"/>
                  <xs:maxInclusive value="90"/>
                 </xs:restriction>
                </xs:simpleType>
               </xs:element>
               <xs:element name="Longitude">
                <xs:simpleType>
                 <xs:restriction base="xs:float">
                  <xs:minInclusive value="-90"/>
                  <xs:maxInclusive value="90"/>
                 </xs:restriction>
                </xs:simpleType>
               </xs:element>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
</xs:schema>
```

Fichier Schema du protocole GRIMML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.inocybe.ca/GRIMML"
targetNamespace="http://www.inocybe.ca/GRIMML" elementFormDefault="qualified">
 <xs:complexType name="readType">
  <xs:sequence>
```

```
    <xs:element name="Input" type="xs:string"/>
    <xs:element name="Output" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="writeType">
   <xs:sequence>
    <xs:element name="Input" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="parameterType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Mutable" type="xs:boolean"/>
    <xs:element name="Description" type="xs:string"/>
    <xs:element name="Type" type="xs:string"/>
    <xs:element name="Value" type="xs:string"/>
    <xs:element name="MaxValue" type="xs:string"/>
    <xs:element name="MinValue" type="xs:string"/>
    <xs:element name="Delimeter" type="xs:string"/>
    <xs:element name="NotifyOnChange" type="xs:boolean"/>
    <xs:element name="Buffered" type="xs:boolean"/>
    <xs:element name="BufferSize" type="xs:integer"/>
    <xs:element name="Read" type="readType"/>
    <xs:element name="Write" type="writeType"/>
   </xs:sequence>
   <xs:attribute name="UUID" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="parameterWithUpdateType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Mutable" type="xs:boolean"/>
    <xs:element name="Description" type="xs:string"/>
    <xs:element name="Type" type="xs:string"/>
    <xs:element name="Value" type="xs:string"/>
    <xs:element name="MaxValue" type="xs:string"/>
    <xs:element name="MinValue" type="xs:string"/>
    <xs:element name="Delimeter" type="xs:string"/>
    <xs:element name="NotifyOnChange" type="xs:boolean"/>
    <xs:element name="Buffered" type="xs:boolean"/>
    <xs:element name="BufferSize" type="xs:integer"/>
    <xs:element name="LastUpdate" type="xs:dateTime"/>
    <xs:element name="OldValue" type="xs:string"/>
    <xs:element name="Blocked" type="xs:boolean"/>
    <xs:element name="Read" type="readType"/>
    <xs:element name="Write" type="writeType"/>
   </xs:sequence>
   <xs:attribute name="UUID" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="physicalParameterType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Mutable" type="xs:boolean"/>
    <xs:element name="Description" type="xs:string"/>
    <xs:element name="Type" type="xs:string"/>
```

```
    <xs:element name="Value" type="xs:string"/>
    <xs:element name="MaxValue" type="xs:string"/>
    <xs:element name="MinValue" type="xs:string"/>
    <xs:element name="Delimeter" type="xs:string"/>
    <xs:element name="NotifyOnChange" type="xs:boolean"/>
    <xs:element name="Buffered" type="xs:boolean"/>
    <xs:element name="BufferSize" type="xs:integer"/>
    <xs:element name="LastUpdate" type="xs:dateTime"/>
    <xs:element name="OldValue" type="xs:string"/>
    <xs:element name="Blocked" type="xs:boolean"/>
    <xs:element name="Unit" type="xs:string"/>
    <xs:element name="Read" type="readType"/>
    <xs:element name="Write" type="writeType"/>
   </xs:sequence>
   <xs:attribute name="UUID" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="executeType">
   <xs:sequence>
    <xs:element name="Read" type="readType"/>
    <xs:element name="write" type="writeType"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="transportType">
   <xs:sequence>
    <xs:element name="Port" type="xs:string"/>
    <xs:element name="BaudRate" type="xs:integer"/>
    <xs:element name="DataBits" type="xs:integer"/>
    <xs:element name="StopBits" type="xs:integer"/>
    <xs:element name="ParityBits" type="xs:integer"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="UUID" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="functionBlockType">
   <xs:sequence>
    <xs:element name="Parameter" type="parameterType" maxOccurs="unbounded"/>
    <xs:element name="ParameterWithUpdate" type="parameterWithUpdateType" maxOccurs="unbounded"/>
    <xs:element name="PhysicalParameter" type="physicalParameterType" maxOccurs="unbounded"/>
    <xs:element name="Execute" type="executeType"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="description" type="xs:string" use="required"/>
   <xs:attribute name="status" type="xs:string" use="required"/>
   <xs:attribute name="UUID" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="transducerBlockType">
   <xs:sequence>
    <xs:element name="Parameter" type="parameterType" maxOccurs="unbounded"/>
    <xs:element name="ParameterWithUpdate" type="parameterWithUpdateType" maxOccurs="unbounded"/>
    <xs:element name="PhysicalParameter" type="physicalParameterType" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="description" type="xs:string" use="required"/>
   <xs:attribute name="status" type="xs:string" use="required"/>
```

```
  <xs:attribute name="UUID" type="xs:string"/>
 </xs:complexType>
 <xs:complexType name="instrumentType">
  <xs:sequence>
   <xs:element name="Transport" type="transportType"/>
   <xs:element name="Active" type="xs:string"/>
   <xs:element name="FunctionBlock" type="functionBlockType" maxOccurs="unbounded"/>
   <xs:element name="TransducerBlock" type="transducerBlockType" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="UUID" type="xs:string"/>
 </xs:complexType>
 <xs:element name="Instrument" type="instrumentType"/>
</xs:schema>
```

ANNEXE II

FICHIERS WSDL DES RESSOURCES GRIM

Fichier WSDL d'une instance d'une ressource *Parameter*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Mathieu Lemay (ITI) -->
<!--Parameter Service WSRF Definition-->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/ParameterService_instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:rp="http://www.inocybe.ca/GRIM/2006/01/ParameterSchema"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterService_instance"
name="ParameterService">
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="../../../../wsrf/properties/WS-ResourceProperties.wsdl"/>
 <wsdl:types>
  <xsd:schema targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:include schemaLocation="Parameter.xsd"/>
  </xsd:schema>
 </wsdl:types>
 <wsdl:message name="ReadInputMessage">
  <wsdl:part name="parameters" element="prp:read"/>
 </wsdl:message>
 <wsdl:message name="WriteInputMessage">
  <wsdl:part name="parameters" element="prp:write"/>
 </wsdl:message>
 <wsdl:message name="WriteOutputMessage">
  <wsdl:part name="parameters" element="prp:writeResponse"/>
 </wsdl:message>
 <wsdl:message name="ReadOutputMessage">
  <wsdl:part name="parameters" element="prp:readResponse"/>
 </wsdl:message>
 <portType name="ParameterPortType" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="prp:ParameterRP">
  <wsdl:operation name="Read">
   <wsdl:input message="tns:ReadInputMessage"/>
   <wsdl:output message="tns:ReadOutputMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Write">
   <wsdl:input message="tns:WriteInputMessage"/>
   <wsdl:output message="tns:WriteOutputMessage"/>
  </wsdl:operation>
 </portType>
</wsdl:definitions>
```

Fichier WSDL d'une instance d'une ressource *ParameterWithUpdate*

```
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:ns2="http://www.inocybe.ca/GRIM/2006/01/CommonData"
xmlns:ns3="http://www.inocybe.ca/GRIM/2006/01/ParameterWithUpdateResource"
xmlns:ns4="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterWithUpdateResource"
name="Parameter">
        <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.wsdl" location="wsrf/properties/WS-ResourceProperties.wsdl"/>
        <wsdl:import namespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResource"
location="ParameterResource.wsdl"/>
        <wsdl:types>
                <xsd:schema
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterWithUpdateResourceProperties"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/ParameterWithUpdateResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                        <xsd:import
namespace="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
schemaLocation="grimdata/TransducerBlockResourceProperties.xsd"/>
                        <xsd:import namespace="http://www.inocybe.ca/GRIM/2006/01/CommonData"
schemaLocation="grimdata/CommonData.xsd"/>
                </xsd:schema>
        </wsdl:types>
        <wsdl:message name="UpdateAndReadResp">
                <wsdl:part name="returnCode" element="ns2:OperationReturnCode"/>
        </wsdl:message>
        <portType name="ParameterWithUpdate" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="ns3:ParameterWithUpdateResourceProperties">
                <wsdl:operation name="UpdateAndRead">
                        <wsdl:output message="ns3:UpdateAndReadResp"/>
                </wsdl:operation>
                <wsdl:operation name="WriteAndUpdate">
                        <wsdl:output message="ns3:UpdateAndReadResp"/>
                </wsdl:operation>
                <wsdl:operation name="ReadBLockUntilUpdate">
                        <wsdl:output message="ns3:UpdateAndReadResp"/>
                </wsdl:operation>
                <wsdl:operation name="WriteBlockUntilUpdate">
                        <wsdl:output message="ns3:UpdateAndReadResp"/>
                </wsdl:operation>
        </portType>
</wsdl:definitions>
```

Fichier WSDL d'une instance d'une ressource *PhysicalParameter*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by  () -->
<!--Parameter Service WSRF Definition-->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:ns2="http://www.inocybe.ca/GRIM/2006/01/CommonData"
xmlns:ns3="http://www.inocybe.ca/GRIM/2006/01/PhysicalParameterResource"
xmlns:ns4="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/PhysicalParameterResource" name="Parameter">
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="wsrf/properties/WS-ResourceProperties.wsdl"/>
 <wsdl:import namespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResource"
location="ParameterResource.wsdl"/>
 <wsdl:types>
  <xsd:schema
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/PhysicalParameterResourceProperties"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/PhysicalParameterResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
schemaLocation="grimdata/TransducerBlockResourceProperties.xsd"/>
    <xsd:import namespace="http://www.inocybe.ca/GRIM/2006/01/CommonData"
schemaLocation="grimdata/CommonData.xsd"/>
  </xsd:schema>
 </wsdl:types>
 <wsdl:message name="UpdateAndReadResp">
  <wsdl:part name="returnCode" element="ns2:OperationReturnCode"/>
 </wsdl:message>
 <portType name="PhysicalParameter" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="ns3:PhysicalParameterResourceProperties">
  <wsdl:operation name="UpdateAndRead">
   <wsdl:output message="ns3:UpdateAndReadResp"/>
  </wsdl:operation>
  <wsdl:operation name="WriteAndUpdate">
   <wsdl:output message="ns3:UpdateAndReadResp"/>
  </wsdl:operation>
  <wsdl:operation name="ReadBlockUntilUpdate">
   <wsdl:output message="ns3:UpdateAndReadResp"/>
  </wsdl:operation>
  <wsdl:operation name="WriteBlockUntilUpdate">
   <wsdl:output message="ns3:UpdateAndReadResp"/>
  </wsdl:operation>
 </portType>
</wsdl:definitions>


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

Fichier WSDL de base pour les *FunctionBlock* et *TransducerBlock*

```
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by  () -->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/FunctionResource_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/BaseBlockResource" name="BaseBlock">
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="wsrf/properties/WS-ResourceProperties.wsdl"/>
 <wsdl:types>
   <xsd:schema targetNamespace="http://www.inocybe.ca/GRIM/core/FunctionResource_Instance"
xmlns:tns="http://www.inocybe.ca/GRIM/core/FunctionResource_Instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="read" type="xsd:string"/>
    <xsd:element name="readResponse">
     <xsd:complexType/>
    </xsd:element>
    <xsd:element name="subtract" type="xsd:int"/>
    <xsd:element name="subtractResponse">
     <xsd:complexType/>
    </xsd:element>
    <xsd:element name="getValueRP">
     <xsd:complexType/>
    </xsd:element>
    <xsd:element name="getValueRPResponse" type="xsd:int"/>
    <!-- RESOURCE PROPERTIES -->
    <xsd:element name="BlockState" type="xsd:int"/>
    <xsd:element name="LastOp" type="xsd:string"/>
    <xsd:element name="FunctionResourceProperties">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element ref="tns:BlockState" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:schema>
 </wsdl:types>
 <message name="ReadInputMessage">
         </message>
 <portType name="BaseBlock" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="tns:MathResourceProperties">
  <wsdl:operation name="Initialize"/>
  <wsdl:operation name="GoActive"/>
  <wsdl:operation name="GoInactive"/>
  <wsdl:operation name="Perform"/>
 </portType>
</wsdl:definitions>
```

Fichier WSDL d'une instance d'une ressource *FunctionBlock*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by () -->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/FunctionBlock_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
targetNamespace="http://www.inocybe.ca/GRIM/core/FunctionBlock_Instance" name="MathService">
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="wsrf/properties/WS-ResourceProperties.wsdl"/>
 <wsdl:import namespace="http://www.inocybe.ca/GRIM/2006/01/BaseBlockResource"
location="BaseBlockResource.wsdl"/>
 <wsdl:types>
  <xsd:schema targetNamespace="http://www.inocybe.ca/GRIM/core/FunctionBlock_Instance"
xmlns:tns="http://www.inocybe.ca/GRIM/core/FunctionBlock_Instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!-- REQUESTS AND RESPONSES -->
   <xsd:element name="read" type="xsd:string"/>
   <xsd:element name="readResponse">
    <xsd:complexType/>
   </xsd:element>
   <xsd:element name="subtract" type="xsd:int"/>
   <xsd:element name="subtractResponse">
    <xsd:complexType/>
   </xsd:element>
   <xsd:element name="getValueRP">
    <xsd:complexType/>
   </xsd:element>
   <xsd:element name="getValueRPResponse" type="xsd:int"/>
   <!-- RESOURCE PROPERTIES -->
   <xsd:element name="State" type="xsd:int"/>
   <xsd:element name="LastOp" type="xsd:string"/>
   <xsd:element name="FunctionBlockProperties">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element ref="tns:State" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:schema>
 </wsdl:types>
 <message name="ReadInputMessage">
        </message>
 <portType name="FunctionBlock" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="tns:MathResourceProperties">
  <wsdl:operation name="Start"/>
  <wsdl:operation name="Clear"/>
  <wsdl:operation name="Pause"/>
  <wsdl:operation name="Resume"/>
```

```
</portType>
</wsdl:definitions>
```

Fichier WSDL d'une instance d'une ressource *TransducerBlock*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by  () -->
<!--Transducer Block Resource Service WSRF Definition-->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:ns2="http://www.inocybe.ca/GRIM/2006/01/CommonData"
xmlns:ns3="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
targetNamespace="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance" name="TransducerBlock">
 <wsdl:import namespace="http://www.inocybe.ca/GRIM/2006/01/BaseBlockResource"
location="BaseBlockResource.wsdl"/>
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="wsrf/properties/WS-ResourceProperties.wsdl"/>
 <wsdl:types>
  <xsd:schema targetNamespace="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:tns="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:import namespace="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
schemaLocation="grimdata/TransducerBlockResourceProperties.xsd"/>
   <xsd:import namespace="http://www.inocybe.ca/GRIM/2006/01/CommonData"
schemaLocation="grimdata/CommonData.xsd"/>
  </xsd:schema>
 </wsdl:types>
 <wsdl:message name="EnableCorrectionsResp">
  <wsdl:part name="returnCode" element="ns2:OperationReturnCode"/>
 </wsdl:message>
 <wsdl:message name="DisableCorrectionsResp">
  <wsdl:part name="returnCode" element="ns2:OperationReturnCode"/>
 </wsdl:message>
 <wsdl:message name="UpdateAllResp">
  <wsdl:part name="returnCode" element="ns2:OperationReturnCode"/>
 </wsdl:message>
 <portType name="TransducerBlock" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="ns3:TransducerBlockResourceProperties">
  <wsdl:operation name="EnableCorrections">
   <wsdl:output message="tns:EnableCorrectionsResp"/>
  </wsdl:operation>
  <wsdl:operation name="DisableCorrections">
   <wsdl:output message="tns:DisableCorrectionsResp"/>
  </wsdl:operation>
  <wsdl:operation name="UpdateAll">
   <wsdl:output message="tns:UpdateAllResp"/>
  </wsdl:operation>
 </portType>
```

```
</wsdl:definitions>
```

Fichier WSDL d'une instance d'une ressource *Action*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Developper (ITI) -->
<!--Parameter Service WSRF Definition-->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:ns2="http://www.inocybe.ca/GRIM/2006/01/CommonData"
xmlns:ns3="http://www.inocybe.ca/GRIM/2006/01/ParameterResource"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResource" name="Parameter">
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="../../../../wsrf/properties/WS-ResourceProperties.wsdl"/>
 <wsdl:types>
  <xsd:schema targetNamespace="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/ParameterResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import schemaLocation="../Common/CommonData.xsd"/>
    <xsd:import schemaLocation="Action.xsd"/>
  </xsd:schema>
 </wsdl:types>
 <wsdl:message name="operationResp">
  <wsdl:part name="returnCode" element="ns2:OperationReturnCode"/>
 </wsdl:message>
 <portType name="Action" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="ns3:ParameterResourceProperties">
  <wsdl:operation name="InvokeTransaction">
   <wsdl:output message="ns3:operationResp"/>
  </wsdl:operation>
  <wsdl:operation name="ForceAcquireTransaction">
   <wsdl:output message="ns3:operationResp"/>
  </wsdl:operation>
  <wsdl:operation name="ReleaseTransaction">
   <wsdl:output message="ns3:operationResp"/>
  </wsdl:operation>
  <wsdl:operation name="AbortTransaction">
   <wsdl:output message="ns3:operationResp"/>
  </wsdl:operation>
 </portType>
</wsdl:definitions>
```

Fichier WSDL d'une instance d'une ressource *Instrument*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by  () -->
<!--Parameter Service WSRF Definition-->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.inocybe.ca/GRIM/core/TransducerBlock_Instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-ResourceProperties-1.2-draft-01.xsd" xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
WS-BaseFaults-1.2-draft-01.xsd" xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:ns2="http://www.inocybe.ca/GRIM/2006/01/CommonData"
xmlns:ns3="http://www.inocybe.ca/GRIM/2006/01/PhysicalParameterResource"
xmlns:ns4="http://www.inocybe.ca/GRIM/2006/01/TransducerBlockResource"
targetNamespace="http://www.inocybe.ca/GRIM/2006/01/PhysicalParameterResource" name="Parameter">
 <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl" location="wsrf/properties/WS-ResourceProperties.wsdl"/>
  <wsdl:types>
   <xsd:schema targetNamespace="http://www.inocybe.ca/GRIM/2006/01/InstrumentResourceProperties"
xmlns:tns="http://www.inocybe.ca/GRIM/2006/01/InstrumentResourceProperties"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   </xsd:schema>
  </wsdl:types>
  <wsdl:message name="GetBlocksResp">
   <wsdl:part name="blocks" element="wsa:EndpointReferenceType"/>
  </wsdl:message>
  <portType name="PhysicalParameter" wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="ns3:PhysicalParameterResourceProperties">
   <wsdl:operation name="getBlocks">
    <wsdl:output message="ns3:GetBlocksResp"/>
   </wsdl:operation>
  </portType>
</wsdl:definitions>
```

ANNEXE III

FICHIERS GRIMML DES INSTRUMENTS

Fichier GRIMML du EFA-P15

```
<?xml version="1.0" encoding="utf-8"?>
<!--example xml document for an instrument -->
<Instrument name="EFA-P15" UUID="">
 <Description>MPB's Erbium Doped Fiber Amplifier (Model EFA-P15)</Description>
 <Transport name="SerialTransport" class="ca.inocybe.grim.core.SerialTransport">
  <Port>/dev/ttyS1</Port>
  <BaudRate>9600</BaudRate>
  <DataBits>8</DataBits>
  <StopBits>1</StopBits>
  <ParityBits>0</ParityBits>
 </Transport>
 <Active>false</Active>
 <!-- function block with direct read/write -->
 <FunctionBlock name="EDFASettings" UUID="">
  <ParameterWithUpdate UUID="">
   <Name>LaserDiodeState</Name>
   <Mutable>true</Mutable>
   <Description>State of the Laser Diode of the EDFA</Description>
   <Type>scalar</Type>
   <Value>0</Value>
   <MaxValue>1</MaxValue>
   <MinValue>0</MinValue>
   <Delimeter/>
   <NotifyOnChange>false</NotifyOnChange>
   <Buffered>false</Buffered>
   <BufferSize>0</BufferSize>
   <Read>
    <Input>$GETSTATE%d\r</Input>
    <Output>%d\rOK</Output>
   </Read>
   <Write>
    <Input value="0">$LDON\r</Input>
    <Input value="1">$LDOFF\r</Input>
   </Write>
  </ParameterWithUpdate>
  <ParameterWithUpdate UUID="">
   <Name>AdjustCurrent</Name>
   <Mutable>true</Mutable>
   <Description>The amount of current at the diode in percentage (0 to 1000).</Description>
   <Type>scalar</Type>
   <Value>0</Value>
   <MaxValue>1000</MaxValue>
   <MinValue>0</MinValue>
   <Delimeter/>
   <NotifyOnChange>false</NotifyOnChange>
   <Buffered>false</Buffered>
```

```
    <BufferSize>0</BufferSize>
    <Read></Read>
    <Write>
      <Input>$SETADJUST%d\r</Input>
    </Write>
  </ParameterWithUpdate>
  <Execute/>
</FunctionBlock>
<!-- function block using execute for multiple parameter read/write -->
<TransducerBlock name="EDFAStatus" UUID="">
  <PhysicalParameter UUID="">
    <Name>LaserDiodeCurrent</Name>
    <Mutable>false</Mutable>
    <Description>Current of the laser diode</Description>
    <Type>scalar</Type>
    <Value/>
    <MaxValue/>
    <MinValue>0</MinValue>
    <Delimeter/>
    <NotifyOnChange>false</NotifyOnChange>
    <Buffered>false</Buffered>
    <BufferSize>0</BufferSize>
    <Read>
      <Input>$GETLDCURRENT\r</Input>
      <Output>%d\rOK</Output>
    </Read>
    <Write/>
  </PhysicalParameter>
  <PhysicalParameter UUID="">
    <Name>OutputPower</Name>
    <Mutable>false</Mutable>
    <Description>OutputPower of EDFA</Description>
    <Type>scalar</Type>
    <Value/>
    <MaxValue/>
    <MinValue>0</MinValue>
    <Delimeter/>
    <NotifyOnChange>true</NotifyOnChange>
    <Buffered>false</Buffered>
    <BufferSize>0</BufferSize>
    <Read>
      <Input>$GETOUTPUT\r</Input>
      <Output>\d*\rOK</Output>
    </Read>
    <Write/>
  </PhysicalParameter>
  <ParameterWithUpdate UUID="">
    <Name>Status</Name>
    <Mutable>false</Mutable>
    <Description>Status CODE of EDFA</Description>
    <Type>scalar</Type>
    <Value/>
    <MaxValue>255</MaxValue>
    <MinValue>0</MinValue>
```

```
    <Delimeter/>
    <NotifyOnChange>true</NotifyOnChange>
    <Buffered>false</Buffered>
    <BufferSize>0</BufferSize>
    <Read>
      <Input>$GETSTATE\r</Input>
      <Output>%d\rOK</Output>
    </Read>
    <Write/>
  </ParameterWithUpdate>
 </TransducerBlock>
</Instrument>
```

Fichier GRIMML du AQ6317B

```
<?xml version="1.0" encoding="utf-8"?>
<!--example xml document for an instrument -->
<Instrument name="AQ6317B" UUID="">
<Description>Ando's Optical Spectrum Analyser (Model AQ6317B)</Description>
<Transport name="GPIBTransport" class="ca.inocybe.grim.core.GPIBTransport">
 <Port>/dev/gpib0</Port>
 <PrimaryAddress>0</PrimaryAddress>
 <SecondaryAddress>18</SecondaryAddress>
</Transport>
<Active>false</Active>
<!-- function block with direct read/write -->
<FunctionBlock name="Sweep" UUID="">
 <PhysicalParameter UUID="">
  <Name>Span</Name>
  <Mutable>true</Mutable>
  <Description>Span of the sweep </Description>
  <Type>scalar</Type>
  <Value>0</Value>
  <Unit>nm</Unit>
  <MaxValue>1600</MaxValue>
  <MinValue>800</MinValue>
  <Delimeter/>
  <NotifyOnChange>false</NotifyOnChange>
  <Buffered>false</Buffered>
  <BufferSize>0</BufferSize>
  <Read>
   <Input>SPAN?\r</Input>
   <Output>%d </Output>
  </Read>
  <Write>
   <Input>SPAN%d</Input>
  </Write>
 </PhysicalParameter>
 <PhysicalParameter UUID="">
  <Name>CenterWL</Name>
  <Mutable>true</Mutable>
  <Description>Center Wavelength of the Sweep.</Description>
  <Type>scalar</Type>
  <Value>0</Value>
  <Unit>nm</Unit>
  <MaxValue>800</MaxValue>
  <MinValue>1600</MinValue>
  <Delimeter/>
  <NotifyOnChange>false</NotifyOnChange>
  <Buffered>false</Buffered>
  <BufferSize>0</BufferSize>
  <Read>
   <Input>CTRWL?\r</Input>
   <Output>%d </Output>
  </Read>
  <Write>
```

```
    <Input>CTRWL%d</Input>
   </Write>
  </PhysicalParameter>
  <PhysicalParameter UUID="">
   <Name>Resolution</Name>
   <Mutable>true</Mutable>
   <Description>Resolution of the Sweep.</Description>
   <Type>List</Type>
   <Value>0</Value>
   <Unit>nm</Unit>
   <ValidValues>0.1,0.2,0.5,1,2</ValidValues>
   <Delimeter>,</Delimeter>
   <NotifyOnChange>false</NotifyOnChange>
   <Buffered>false</Buffered>
   <BufferSize>0</BufferSize>
   <Read>
    <Input>RESLN?\r</Input>
    <Output>%d </Output>
   </Read>
   <Write>
    <Input>RESLN%d</Input>
   </Write>
  </PhysicalParameter>
   <PhysicalParameter UUID="">
   <Name>Sensibility</Name>
   <Mutable>true</Mutable>
   <Description>Resolution of the Sweep.</Description>
   <Type>List</Type>
   <Value>0</Value>
   <Unit>nm</Unit>
   <ValidValues>Normal Hold, Normal Auto, Medium, High 1, High 2, High 3</ValidValues>
   <Delimeter>,</Delimeter>
   <NotifyOnChange>false</NotifyOnChange>
   <Buffered>false</Buffered>
   <BufferSize>0</BufferSize>
   <Read>
    <Input>SN?</Input>
    <Output value="Normal Hold">SNHD</Output>
    <Output value="Normal Auto">SNAT</Output>
    <Output value="Medium">SMID</Output>
    <Output value="High 1">SHI1</Output>
    <Output value="High 2">SHI2</Output>
    <Output value="High 3">SHI3</Output>
   </Read>
   <Write>
     <Input value="Normal Hold">SNHD</Input>
    <Input value="Normal Auto">SNAT</Input>
    <Input value="Medium">SMID</Input>
    <Input value="High 1">SHI1</Input>
    <Input value="High 2">SHI2</Input>
    <Input value="High 3">SHI3</Input>
   </Write>
  </PhysicalParameter>
  <PhysicalParameter UUID="">
```

```
    <Name>XData</Name>
    <Mutable>false</Mutable>
    <Description>Wavelengths dataseries for sweep.</Description>
    <Type>Series</Type>
    <Value>0</Value>
    <Unit>nm</Unit>
    <Delimeter>,</Delimeter>
    <NotifyOnChange>false</NotifyOnChange>
    <Buffered>true</Buffered>
    <BufferSize>1024</BufferSize>
    <Read>
    </Read>
    <Write>
    </Write>
  </PhysicalParameter>
  <PhysicalParameter UUID="">
    <Name>YData</Name>
    <Mutable>false</Mutable>
    <Description>Power dataseries for sweep.</Description>
    <Type>Series</Type>
    <Value>0</Value>
    <Unit>nm</Unit>
    <Delimeter>,</Delimeter>
    <NotifyOnChange>false</NotifyOnChange>
    <Buffered>true</Buffered>
    <BufferSize>1024</BufferSize>
    <Read>
    </Read>
    <Write>
    </Write>
  </PhysicalParameter>
  <Execute class="ca.inocybe.grim.example.SweepFunction"/>
</FunctionBlock>
</Instrument>
```

Fichier GRIMML du IQ-1600

```
<?xml version="1.0" encoding="utf-8"?>
<!--example xml document for an instrument -->
<Instrument name="AQ6317B" UUID="">
<Description>Ando's Optical Spectrum Analyser (Model AQ6317B)</Description>
<Transport name="GPIBTransport" class="ca.inocybe.grim.core.GPIBTransport">
  <Port>/dev/gpib0</Port>
  <PrimaryAddress>0</PrimaryAddress>
  <SecondaryAddress>18</SecondaryAddress>
</Transport>
<Active>false</Active>
<!-- function block with direct read/write -->
<FunctionBlock name="ChannelSelect" UUID="">
  <ParameterWithUpdate UUID="">
    <Name>CurrentChannel</Name>
    <Mutable>true</Mutable>
    <Description>Current Channel connected to Common</Description>
    <Type>scalar</Type>
    <Value>0</Value>
    <MaxValue>4</MaxValue>
    <MinValue>1</MinValue>
    <Delimeter/>
    <NotifyOnChange>false</NotifyOnChange>
    <Buffered>false</Buffered>
    <BufferSize>0</BufferSize>
    <Read>
      <Input>ROUT0:SCAN?</Input>
      <Output>%d </Output>
    </Read>
    <Write>
      <Input>ROUT0:SCAN%d</Input>
    </Write>
  </ParameterWithUpdate>
  <Execute/>
</FunctionBlock>
</Instrument>
```

ANNEXE IV

FICHIER SENSORML D'UNE STATION MÉTÉO

```
<?xml version="1.0"?>
<SensorML xmlns="http://www.opengis.net/sensorML" xmlns:swe="http://www.opengis.net/swe"
xmlns:gml="http://www.opengis.net/gml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/sensorML
http://vast.uah.edu/schemas/sensorML/1.0.30/base/sensorML.xsd" version="1.0">
 <System id="DAVIS_SIMPLE_STATION">
  <!--============================================-->
  <!-- Station Discovery Metadata -->
  <!--============================================-->
  <description>
    <swe:Discussion>Simple Weather Station measuring temperature, pressure, wind and rain
fall</swe:Discussion>
  </description>
  <identification>
   <IdentifierList>
    <identifier name="longName">
     <Term qualifier="urn:ogc:def:identifier:longName">Davis Weather Monitor II Station</Term>
    </identifier>
    <identifier name="shortName">
     <Term qualifier="urn:ogc:def:identifier:shortName">Davis Weather Station</Term>
    </identifier>
    <identifier name="modelNumber">
     <Term qualifier="urn:ogc:def:identifier:modelNumber">7440</Term>
    </identifier>
    <identifier name="manufacturer">
     <Term qualifier="urn:ogc:def:identifier:manufacturer">Davis Instruments</Term>
    </identifier>
   </IdentifierList>
  </identification>
  <classification>
   <ClassifierList>
    <classifier name="intendedApplication">
     <Term qualifier="urn:ogc:def:classifier:application">weather</Term>
    </classifier>
    <classifier name="sensorType">
     <Term qualifier="urn:ogc:def:classifier:sensorType">thermometer</Term>
    </classifier>
    <classifier name="sensorType">
     <Term qualifier="urn:ogc:def:classifier:sensorType">barometer</Term>
    </classifier>
    <classifier name="sensorType">
     <Term qualifier="urn:ogc:def:classifier:sensorType">anemometer</Term>
    </classifier>
    <classifier name="sensorType">
     <Term qualifier="urn:ogc:def:classifier:sensorType">rain gauge</Term>
    </classifier>
   </ClassifierList>
```

```
</classification>
<validTime>
 <StartTime>2005-01-01</StartTime>
 <EndTime>currentTime</EndTime>
</validTime>
<contact role="urn:ogc:def:identifier:manufacturer">
 <ResponsibleParty>
  <organizationName>Davis Instruments</organizationName>
  <contactInfo>
   <phone>
    <voice>+01-510-732-9229</voice>
    <facsimile>+01-510-732-9188</facsimile>
   </phone>
   <address>
    <deliveryPoint>3465, Diablo Avenue</deliveryPoint>
    <city>Hayward</city>
    <administrativeArea>CA</administrativeArea>
    <postalCode>94545-2778</postalCode>
    <country>USA</country>
    <electronicMailAddress>sales@davisnet.com</electronicMailAddress>
   </address>
  </contactInfo>
 </ResponsibleParty>
</contact>
<documentation role="documents">
 <DocumentList>
  <member name="userManual">
   <Document>
    <description>
     <swe:Discussion>Davis Weather Monitor Manual</swe:Discussion>
    </description>
    <fileLocation xlink:href="http://www.davisnet.com/support/manuals/7440.pdf"/>
   </Document>
  </member>
 </DocumentList>
</documentation>
<!--==========================-->
<!-- Station Coordinate Frame -->
<!--==========================-->
<referenceFrame>
 <gml:EngineeringCRS gml:id="STATION_FRAME">
  <gml:srsName>Weather Station Spatial Frame</gml:srsName>
  <gml:usesCS xlink:href="urn:ogc:def:cs:xyzFrame"/>
  <gml:usesEngineeringDatum>
   <gml:EngineeringDatum gml:id="STATION_DATUM">
    <gml:datumName>Weather Station Spatial Datum</gml:datumName>
    <gml:anchorPoint>origin is at the base of the mounting. Z is along the axis of the mounting pole -
typically vertical. X and Y are orthogonal to Z, along the short and long edges of the case
respectively.</gml:anchorPoint>
   </gml:EngineeringDatum>
  </gml:usesEngineeringDatum>
 </gml:EngineeringCRS>
</referenceFrame>
<!--==================-->
```

```
<!-- Station Inputs -->
<!--==========-->
<inputs>
 <InputList>
  <input name="ambiantTemperature">
   <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"/>
  </input>
  <input name="atmosphericPressure">
   <swe:Quantity definition="urn:ogc:def:phenomenon:pressure"/>
  </input>
  <input name="windSpeed">
   <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed"/>
  </input>
  <input name="windDirection">
   <swe:Quantity definition="urn:ogc:def:phenomenon:windDirection"/>
  </input>
  <input name="rainFall">
   <swe:Quantity definition="urn:ogc:def:phenomenon:rainFall"/>
  </input>
 </InputList>
</inputs>
<!--==========-->
<!-- Station Outputs -->
<!--==========-->
<outputs>
 <OutputList>
  <output name="weatherMeasurements">
   <swe:DataGroup>
    <swe:component name="time">
     <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601"/>
    </swe:component>
    <swe:component name="temperature">
     <swe:Quantity definition="urn:ogc:def:phenomenon:temperature" uom="urn:ogc:def:unit:celsius"/>
    </swe:component>
    <swe:component name="barometricPressure">
     <swe:Quantity definition="urn:ogc:def:phenomenon:pressure" uom="urn:ogc:def:unit:bar" scale="1e-
3"/>
    </swe:component>
    <swe:component name="windSpeed">
     <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed"
uom="urn:ogc:def:unit:meterPerSecond"/>
    </swe:component>
    <swe:component name="windDirection">
     <swe:Quantity definition="urn:ogc:def:phenomenon:windDirection" uom="urn:ogc:def:unit:degree"/>
    </swe:component>
    <swe:component name="rainFall">
     <swe:Quantity definition="urn:ogc:def:phenomenon:rainfall" uom="urn:ogc:def:unit:meter" scale="1e-
3"/>
    </swe:component>
   </swe:DataGroup>
  </output>
 </OutputList>
</outputs>
<!--==========-->
```

```
<!-- Station Detector List -->
<!--==========================-->
<processes>
 <ProcessList>
  <!--=================================-->
  <!-- Ambiant Temperature Detector -->
  <!--=================================-->
  <process name="thermometer">
   <Detector id="DAVIS_THERMOMETER">
    <identification>
     <IdentifierList>
      <identifier name="longName">
       <Term qualifier="urn:ogc:def:identifier:longName">Davis Temperature Detector</Term>
      </identifier>
      <identifier name="modelNumber">
       <Term qualifier="urn:ogc:def:identifier:modelNumber">7817</Term>
      </identifier>
     </IdentifierList>
    </identification>
    <!-- REFERENCE FRAME -->
    <referenceFrame>
     <gml:EngineeringCRS gml:id="THERMOMETER_FRAME">
      <gml:srsName>Temperature Detector Spatial Frame</gml:srsName>
      <gml:usesCS xlink:href="urn:ogc:def:cs:xyzFrame"/>
      <gml:usesEngineeringDatum>
       <gml:EngineeringDatum gml:id="THERMOMETER_DATUM">
        <gml:datumName>Temperature Detector Spatial Datum</gml:datumName>
        <gml:anchorPoint>Origin is situated at the connector/case junction. X,Y,Z are undetermined since
orientation is not critical.</gml:anchorPoint>
       </gml:EngineeringDatum>
      </gml:usesEngineeringDatum>
     </gml:EngineeringCRS>
    </referenceFrame>
    <!-- INPUTS -->
    <inputs>
     <InputList>
      <input name="temperature">
       <swe:Quantity definition="urn:ogc:def:phenomenon:temperature" uom="urn:ogc:def:unit:celsius"/>
      </input>
     </InputList>
    </inputs>
    <!-- OUTPUTS -->
    <outputs>
     <OutputList>
      <output name="measuredTemperature">
       <swe:Quantity definition="urn:ogc:def:phenomenon:temperature" uom="urn:ogc:def:unit:celsius"/>
      </output>
     </OutputList>
    </outputs>
    <!-- PARAMETERS -->
    <parameters>
     <ParameterList>
      <steadyStateResponse>
       <ConditionalCurve fixed="true">
```

```
            <condition name="calibrationTime">
              <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
            </condition>
            <data>
              <swe:NormalizedCurve>
                <swe:function>
                  <swe:Curve arraySize="2">
                    <swe:definition>
                      <swe:Coordinates>
                        <swe:axis name="realTemperature">
                          <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
uom="urn:ogc:def:unit:celsius"/>
                        </swe:axis>
                        <swe:axis name="measuredTemperature">
                          <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
uom="urn:ogc:def:unit:celsius"/>
                        </swe:axis>
                      </swe:Coordinates>
                    </swe:definition>
                    <swe:tupleValues>-40,-40 60,60</swe:tupleValues>
                  </swe:Curve>
                </swe:function>
              </swe:NormalizedCurve>
            </data>
          </ConditionalCurve>
        </steadyStateResponse>
        <error>
          <ConditionalCurve>
            <condition name="cableLength">
              <swe:Quantity definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">10</swe:Quantity>
            </condition>
            <data>
              <swe:NormalizedCurve>
                <swe:function>
                  <swe:Curve arraySize="6">
                    <swe:definition>
                      <swe:Coordinates>
                        <swe:axis name="realTemperature">
                          <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
uom="urn:ogc:def:unit:celsius"/>
                        </swe:axis>
                        <swe:axis name="absoluteError">
                          <swe:Quantity definition="urn:ogc:def:phenomenon:absoluteError"
uom="urn:ogc:def:unit:celsius"/>
                        </swe:axis>
                      </swe:Coordinates>
                    </swe:definition>
                    <swe:tupleValues>-40,0 -10,0.1 15,0.3 27,0.5 44,1 60,2.1</swe:tupleValues>
                  </swe:Curve>
                </swe:function>
              </swe:NormalizedCurve>
            </data>
```

```
            </ConditionalCurve>
           </error>
           <latencyTime>
            <ConditionalValue>
             <condition name="medium">
              <swe:Category definition="urn:ogc:def:category:medium">air</swe:Category>
             </condition>
             <data>
              <swe:Quantity definition="urn:ogc:def:phenomenon:duration"
uom="urn:ogc:def:unit:second">10</swe:Quantity>
             </data>
            </ConditionalValue>
           </latencyTime>
          </ParameterList>
         </parameters>
         <!-- METHOD -->
         <method xlink:href="urn:ogc:def:process:1.0:detector"/>
        </Detector>
       </process>
       <!--==========================================-->
       <!-- Barometric Pressure Detector -->
       <!--==========================================-->
       <process name="barometer">
        <Detector id="DAVIS_BAROMETER">
         <identification>
          <IdentifierList>
           <identifier name="longName">
            <Term qualifier="urn:ogc:def:identifier:longName">Davis Internal Barometer</Term>
           </identifier>
          </IdentifierList>
         </identification>
         <!-- REFERENCE FRAME -->
         <referenceFrame>
          <gml:EngineeringCRS gml:id="BAROMETER_FRAME">
           <gml:srsName>Barometric Detector Spatial Frame</gml:srsName>
           <gml:usesCS xlink:href="urn:ogc:def:cs:xyzFrame"/>
           <gml:usesEngineeringDatum>
            <gml:EngineeringDatum gml:id="BAROMETER_DATUM">
             <gml:datumName>Barometric Detector Spatial Datum</gml:datumName>
             <gml:anchorPoint>Origin is situated at the connector/case junction. X,Y,Z are undetermined since
orientation is not critical.</gml:anchorPoint>
            </gml:EngineeringDatum>
           </gml:usesEngineeringDatum>
          </gml:EngineeringCRS>
         </referenceFrame>
         <!-- INPUTS -->
         <inputs>
          <InputList>
           <input name="barometricPressure">
            <swe:Quantity definition="urn:ogc:def:phenomenon:pressure" uom="urn:ogc:def:unit:bar"
scale="1e-3"/>
           </input>
          </InputList>
         </inputs>
```

```
        <!-- OUTPUTS -->
        <outputs>
         <OutputList>
          <output name="measuredBarometricPressure">
           <swe:Quantity definition="urn:ogc:def:phenomenon:pressure" uom="urn:ogc:def:unit:bar"
scale="1e-3"/>
          </output>
         </OutputList>
        </outputs>
        <!-- PARAMETERS -->
        <parameters>
         <ParameterList>
          <steadyStateResponse>
           <ConditionalCurve fixed="true">
            <condition name="calibrationTime">
             <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
            </condition>
            <condition name="temperature">
             <swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
uom="urn:ogc:def:unit:celsius">20</swe:Quantity>
            </condition>
            <data>
             <swe:NormalizedCurve>
              <swe:function>
               <swe:Curve arraySize="2">
                <swe:definition>
                 <swe:Coordinates>
                  <swe:axis name="atmosphericPressure">
                   <swe:Quantity definition="urn:ogc:def:phenomenon:pressure" uom="urn:ogc:def:unit:bar"
scale="1e-3"/>
                  </swe:axis>
                  <swe:axis name="measuredBarometricPressure">
                   <swe:Quantity definition="urn:ogc:def:phenomenon:pressure" uom="urn:ogc:def:unit:bar"
scale="1e-3"/>
                  </swe:axis>
                 </swe:Coordinates>
                </swe:definition>
                <swe:tupleValues>880.0,880.0 1080.0,1080.0</swe:tupleValues>
               </swe:Curve>
              </swe:function>
             </swe:NormalizedCurve>
            </data>
           </ConditionalCurve>
          </steadyStateResponse>
         </ParameterList>
        </parameters>
        <!-- METHOD -->
        <method xlink:href="urn:ogc:def:process:1.0:detector"/>
       </Detector>
      </process>
      <!--==========================-->
      <!-- Wind Speed Detector -->
      <!--==========================-->
```

```
<process name="anemometer">
 <Detector id="DAVIS_ANEMOMETER">
  <identification>
   <IdentifierList>
    <identifier name="longName">
     <Term qualifier="urn:ogc:def:identifier:longName">Davis Wind Speed Detector</Term>
    </identifier>
    <identifier name="modelNumber">
     <Term qualifier="urn:ogc:def:identifier:modelNumber">7911</Term>
    </identifier>
   </IdentifierList>
  </identification>
  <!-- REFERENCE FRAME -->
  <referenceFrame>
   <gml:EngineeringCRS gml:id="ANEMOMETER_FRAME">
    <gml:srsName>Wind Speed Detector Spatial Frame</gml:srsName>
    <gml:usesCS xlink:href="urn:ogc:def:cs:xyzFrame"/>
    <gml:usesEngineeringDatum>
     <gml:EngineeringDatum gml:id="ANEMOMETER_DATUM">
      <gml:datumName>Wind Speed Detector Spatial Datum</gml:datumName>
      <gml:anchorPoint>Origin is at the base of the rotating part. X,Y,Z axes are undefined since
orientation is not needed</gml:anchorPoint>
     </gml:EngineeringDatum>
    </gml:usesEngineeringDatum>
   </gml:EngineeringCRS>
  </referenceFrame>
  <!-- INPUTS -->
  <inputs>
   <InputList>
    <input name="windSpeed">
     <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed"
uom="urn:ogc:def:unit:meterPerSecond"/>
    </input>
   </InputList>
  </inputs>
  <!-- OUTPUTS -->
  <outputs>
   <OutputList>
    <output name="measuredWindSpeed">
     <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed"
uom="urn:ogc:def:unit:meterPerSecond"/>
    </output>
   </OutputList>
  </outputs>
  <!-- PARAMETERS -->
  <parameters>
   <ParameterList>
    <steadyStateResponse>
     <ConditionalCurve fixed="true">
      <condition name="calibrationTime">
       <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
      </condition>
      <data>
```

```
            <swe:NormalizedCurve>
              <swe:function>
                <swe:Curve arraySize="2">
                  <swe:definition>
                    <swe:Coordinates>
                      <swe:axis name="windSpeed">
                        <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed"
uom="urn:ogc:def:unit:meterPerSecond"/>
                      </swe:axis>
                      <swe:axis name="measuredWindSpeed">
                        <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed"
uom="urn:ogc:def:unit:meterPerSecond"/>
                      </swe:axis>
                    </swe:Coordinates>
                  </swe:definition>
                  <swe:tupleValues>0.9,0.9 78,78</swe:tupleValues>
                </swe:Curve>
              </swe:function>
            </swe:NormalizedCurve>
          </data>
        </ConditionalCurve>
      </steadyStateResponse>
    </ParameterList>
  </parameters>
  <!-- METHOD -->
  <method xlink:href="urn:ogc:def:process:1.0:detector"/>
 </Detector>
</process>
<!--==========================-->
<!-- Wind Direction Detector -->
<!--==========================-->
<process name="windDirectionDetector">
 <Detector id="DAVIS_WIND_DIRECTION">
  <identification>
   <IdentifierList>
    <identifier name="longName">
     <Term qualifier="urn:ogc:def:identifier:longName">Davis Wind Direction Detector</Term>
    </identifier>
    <identifier name="modelNumber">
     <Term qualifier="urn:ogc:def:identifier:modelNumber">7911</Term>
    </identifier>
   </IdentifierList>
  </identification>
  <!-- REFERENCE FRAME -->
  <referenceFrame>
   <gml:EngineeringCRS gml:id="WIND_DIRECTION_DETECTOR_FRAME">
    <gml:srsName>Wind Direction Detector Spatial Frame</gml:srsName>
    <gml:usesCS xlink:href="urn:ogc:def:cs:xyzFrame"/>
    <gml:usesEngineeringDatum>
     <gml:EngineeringDatum gml:id="WIND_DIRECTION_DETECTOR_DATUM">
      <gml:datumName>Wind Direction Detector Spatial Datum</gml:datumName>
      <gml:anchorPoint>Origin is at the base of the rotating part. Z is along the axis around which
measurement is made. X is the reference (0) direction that should point north when installed correctly. Y is
orthogonal to X and Z.</gml:anchorPoint>
```

```
            </gml:EngineeringDatum>
          </gml:usesEngineeringDatum>
        </gml:EngineeringCRS>
      </referenceFrame>
      <!-- INPUTS -->
      <inputs>
        <InputList>
          <input name="windDirection">
            <swe:Quantity definition="urn:ogc:def:phenomenon:windDirection"
uom="urn:ogc:def:unit:degree"/>
          </input>
        </InputList>
      </inputs>
      <!-- OUTPUTS -->
      <outputs>
        <OutputList>
          <output name="measuredWindDirection">
            <swe:Quantity definition="urn:ogc:def:phenomenon:windDirection"
uom="urn:ogc:def:unit:degree"/>
          </output>
        </OutputList>
      </outputs>
      <!-- PARAMETERS -->
      <parameters>
        <ParameterList>
          <steadyStateResponse>
            <ConditionalCurve fixed="true">
              <condition name="calibrationTime">
                <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
              </condition>
              <data>
                <swe:NormalizedCurve>
                  <swe:function>
                    <swe:Curve arraySize="2">
                      <swe:definition>
                        <swe:Coordinates>
                          <swe:axis name="windDirection">
                            <swe:Quantity definition="urn:ogc:phenomenon:windDirection"
uom="urn:ogc:def:unit:degree"/>
                          </swe:axis>
                          <swe:axis name="measuredWindDirection">
                            <swe:Quantity definition="urn:ogc:phenomenon:windDirection"
uom="urn:ogc:def:unit:degree"/>
                          </swe:axis>
                        </swe:Coordinates>
                      </swe:definition>
                      <swe:tupleValues>0.0,0.0 360.0,360.0</swe:tupleValues>
                    </swe:Curve>
                  </swe:function>
                </swe:NormalizedCurve>
              </data>
            </ConditionalCurve>
          </steadyStateResponse>
```

```
        </ParameterList>
       </parameters>
       <!-- METHOD -->
       <method xlink:href="urn:ogc:def:process:1.0:detector"/>
      </Detector>
     </process>
     <!--=============================-->
     <!-- Rain Fall Detector -->
     <!--=============================-->
     <process name="rainGauge">
      <Detector id="DAVIS_RAIN_GAUGE">
       <identification>
        <IdentifierList>
         <identifier name="longName">
          <Term qualifier="urn:ogc:def:identifier:longName">Davis Rain Fall Detector</Term>
         </identifier>
         <identifier name="modelNumber">
          <Term qualifier="urn:ogc:def:identifier:modelNumber">7852</Term>
         </identifier>
        </IdentifierList>
       </identification>
       <!-- REFERENCE FRAME -->
       <referenceFrame>
        <gml:EngineeringCRS gml:id="RAIN_GAUGE_FRAME">
         <gml:srsName>Rain Fall Detector Spatial Frame</gml:srsName>
         <gml:usesCS xlink:href="urn:ogc:def:cs:xyzFrame"/>
         <gml:usesEngineeringDatum>
          <gml:EngineeringDatum gml:id="RAIN_GAUGE_DATUM">
           <gml:datumName>Rain Fall Detector Spatial Datum</gml:datumName>
           <gml:anchorPoint>Origin is situated at the connector/case junction. X,Y,Z are undetermined since
orientation is not critical.</gml:anchorPoint>
          </gml:EngineeringDatum>
         </gml:usesEngineeringDatum>
        </gml:EngineeringCRS>
       </referenceFrame>
       <!-- INPUTS -->
       <inputs>
        <InputList>
         <input name="rainFall">
          <swe:Quantity definition="urn:ogc:def:phenomenon:rainFall" uom="urn:ogc:def:unit:meter"
scale="1e-3"/>
         </input>
        </InputList>
       </inputs>
       <!-- OUTPUTS -->
       <outputs>
        <OutputList>
         <output name="measuredRainFall">
          <swe:Quantity definition="urn:ogc:def:phenomenon:rainFall" uom="urn:ogc:def:unit:meter"
scale="1e-3"/>
         </output>
        </OutputList>
       </outputs>
       <!-- PARAMETERS -->
```

```
        <parameters>
         <ParameterList>
          <steadyStateResponse>
           <ConditionalCurve fixed="true">
            <condition name="calibrationTime">
             <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
            </condition>
            <data>
             <swe:NormalizedCurve>
              <swe:function>
               <swe:Curve arraySize="2">
                <swe:definition>
                 <swe:Coordinates>
                  <swe:axis name="rainFall">
                   <swe:Quantity definition="urn:ogc:def:phenomenon:rainFall"
uom="urn:ogc:def:unit:meter" scale="1e-3"/>
                  </swe:axis>
                  <swe:axis name="measuredRainFall">
                   <swe:Quantity definition="urn:ogc:def:phenomenon:rainFall"
uom="urn:ogc:def:unit:meter" scale="1e-3"/>
                  </swe:axis>
                 </swe:Coordinates>
                </swe:definition>
                <swe:tupleValues>0,0 999,999</swe:tupleValues>
               </swe:Curve>
              </swe:function>
             </swe:NormalizedCurve>
            </data>
           </ConditionalCurve>
          </steadyStateResponse>
         </ParameterList>
        </parameters>
        <!-- METHOD -->
        <method xlink:href="urn:ogc:def:process:1.0:detector"/>
       </Detector>
      </process>
      <!--=================================-->
      <!-- Station Internal Clock -->
      <!--=================================-->
      <process name="clock">
       <Detector id="INTERNAL_CLOCK">
        <identification>
         <IdentifierList>
          <identifier name="longName">
           <Term qualifier="urn:ogc:def:identifier:longName">Davis Monitor Internal Clock</Term>
          </identifier>
         </IdentifierList>
        </identification>
        <!-- INPUTS -->
        <inputs>
         <InputList>
          <input name="time">
           <swe:Time definition="urn:ogc:def:phenomenon:time"/>
```

```
        </input>
       </InputList>
      </inputs>
      <!-- OUTPUTS -->
      <outputs>
       <OutputList>
        <output name="measuredTime">
         <swe:Time definition="urn:ogc:def:phenomenon:time" referenceTimeFrame="1970-01-
01T00:00:00Z" uom="urn:ogc:def:unit:second"/>
        </output>
       </OutputList>
      </outputs>
      <!-- PARAMETERS -->
      <parameters>
       <ParameterList>
        <steadyStateResponse>
         <ConditionalCurve fixed="true">
          <condition name="calibrationTime">
           <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
          </condition>
          <data>
           <swe:NormalizedCurve>
            <swe:function>
             <swe:Curve arraySize="2">
              <swe:definition>
               <swe:Coordinates>
                <swe:axis name="realTime">
                 <swe:Quantity definition="urn:ogc:def:phenomenon:time"
uom="urn:ogc:def:unit:second"/>
                </swe:axis>
                <swe:axis name="measuredTime">
                 <swe:Quantity definition="urn:ogc:def:phenomenon:time"
uom="urn:ogc:def:unit:second"/>
                </swe:axis>
               </swe:Coordinates>
              </swe:definition>
              <swe:tupleValues>0,0 1e8,1e8</swe:tupleValues>
             </swe:Curve>
            </swe:function>
           </swe:NormalizedCurve>
          </data>
         </ConditionalCurve>
        </steadyStateResponse>
        <accuracy>
         <ConditionalValue>
          <condition name="calibrationTime">
           <swe:Time definition="urn:ogc:def:phenomenon:time" uom="urn:ogc:def:unit:iso8601">2004-01-
01T04:30:00</swe:Time>
          </condition>
          <data>
           <swe:Quantity definition="urn:ogc:def:phenomenon:duration"
uom="urn:ogc:def:unit:second">0.001</swe:Quantity>
          </data>
```

```
            </ConditionalValue>
           </accuracy>
          </ParameterList>
         </parameters>
         <!-- METHOD -->
         <method xlink:href="urn:ogc:def:process:1.0:detector"/>
        </Detector>
       </process>
      </ProcessList>
    </processes>
    <!--=================================-->
    <!-- Station Internal Connections -->
    <!--=================================-->
    <connections>
     <ConnectionList>
      <connection name="inputToThermometer">
       <Link>
        <source ref="this/inputs/ambiantTemperature"/>
        <destination ref="thermometer/inputs/temperature"/>
       </Link>
      </connection>
      <connection name="thermometerToOutput">
       <Link>
        <source ref="thermometer/outputs/measuredTemperature"/>
        <destination ref="this/outputs/weatherMeasurements/temperature"/>
       </Link>
      </connection>
      <connection name="inputToBarometer">
       <Link>
        <source ref="this/inputs/atmosphericPressure"/>
        <destination ref="barometer/inputs/barometricPressure"/>
       </Link>
      </connection>
      <connection name="barometerToOutput">
       <Link>
        <source ref="barometer/outputs/measuredBarometricPressure"/>
        <destination ref="this/outputs/weatherMeasurements/barometricPressure"/>
       </Link>
      </connection>
      <connection name="inputToAnemometer">
       <Link>
        <source ref="this/inputs/windSpeed"/>
        <destination ref="anemometer/inputs/windSpeed"/>
       </Link>
      </connection>
      <connection name="anemometerToOutput">
       <Link>
        <source ref="anemometer/outputs/measuredWindSpeed"/>
        <destination ref="this/outputs/weatherMeasurements/windSpeed"/>
       </Link>
      </connection>
      <connection name="inputToWindDirection">
       <Link>
        <source ref="this/inputs/windDirection"/>
```

```
        <destination ref="windDirectionDetector/inputs/windDirection"/>
       </Link>
      </connection>
      <connection name="windDirectionToOutput">
       <Link>
        <source ref="windDirectionDetector/outputs/measuredWindDirection"/>
        <destination ref="this/outputs/weatherMeasurements/windDirection"/>
       </Link>
      </connection>
      <connection name="inputToRainGauge">
       <Link>
        <source ref="this/inputs/rainFall"/>
        <destination ref="rainGauge/inputs/rainFall"/>
       </Link>
      </connection>
      <connection name="rainGaugeToOutput">
       <Link>
        <source ref="rainGauge/outputs/measuredRainFall"/>
        <destination ref="this/outputs/weatherMeasurements/rainFall"/>
       </Link>
      </connection>
      <connection name="clockToOutput">
       <Link>
        <source ref="clock/outputs/measuredTime"/>
        <destination ref="this/outputs/weatherMeasurements/time"/>
       </Link>
      </connection>
     </ConnectionList>
    </connections>
    <!--============================-->
    <!-- Components Positions -->
    <!--============================-->
    <positions>
     <PositionList>
      <!--=============================================-->
      <!-- Position of Station in EPSG4329 -->
      <!--=============================================-->
      <position name="stationPosition">
       <swe:Position localFrame="#STATION_FRAME" referenceFrame="urn:ogc:def:crs:EPSG:4329">
        <swe:location>
         <swe:Location definition="urn:ogc:def:phenomenon:location">
          <swe:coordinate name="latitude">
           <swe:Quantity definition="urn:ogc:def:phenomenon:latitude"
uom="urn:ogc:def:unit:degree">34.72450</swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="longitude">
           <swe:Quantity definition="urn:ogc:def:phenomenon:longitude" uom="urn:ogc:def:unit:degree">-
86.94533</swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="altitude">
           <swe:Quantity definition="urn:ogc:def:phenomenon:altitude"
uom="urn:ogc:def:unit:meter">20.1169</swe:Quantity>
          </swe:coordinate>
         </swe:Location>
```

```xml
        </swe:location>
        <swe:orientation>
          <swe:Orientation definition="urn:ogc:def:phenomenon:orientation">
            <swe:coordinate name="trueHeading">
              <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:angleToNorth"
uom="urn:ogc:def:unit:degree">87.0</swe:Quantity>
            </swe:coordinate>
          </swe:Orientation>
        </swe:orientation>
      </swe:Position>
    </position>
    <!--==============================================================-->
    <!-- Position of Barometer in Station Ref Frame -->
    <!--==============================================================-->
    <position name="barometerPosition">
      <swe:Position localFrame="#BAROMETER_FRAME" referenceFrame="#STATION_FRAME">
        <swe:location>
          <swe:Location definition="urn:ogc:def:phenomenon:location">
            <swe:coordinate name="x">
              <swe:Quantity axisCode="X" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
            </swe:coordinate>
            <swe:coordinate name="y">
              <swe:Quantity axisCode="Y" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
            </swe:coordinate>
            <swe:coordinate name="z">
              <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
            </swe:coordinate>
          </swe:Location>
        </swe:location>
      </swe:Position>
    </position>
    <!--==============================================================-->
    <!-- Position of Thermometer in Station Ref Frame -->
    <!--==============================================================-->
    <position name="thermometerPosition">
      <swe:Position localFrame="#THERMOMETER_FRAME" referenceFrame="#STATION_FRAME">
        <swe:location>
          <swe:Location definition="urn:ogc:def:phenomenon:location">
            <swe:coordinate name="x">
              <swe:Quantity axisCode="X" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
            </swe:coordinate>
            <swe:coordinate name="y">
              <swe:Quantity axisCode="Y" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.02</swe:Quantity>
            </swe:coordinate>
            <swe:coordinate name="z">
              <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">1.1</swe:Quantity>
            </swe:coordinate>
          </swe:Location>
```

```
        </swe:location>
       </swe:Position>
      </position>
      <!--=============================================================-->
      <!-- Position of Anemometer in Station Ref Frame -->
      <!--=============================================================-->
      <position name="anemometerPosition">
       <swe:Position localFrame="#ANEMOMETER_FRAME" referenceFrame="#STATION_FRAME">
        <swe:location>
         <swe:Location definition="urn:ogc:def:phenomenon:location">
          <swe:coordinate name="x">
           <swe:Quantity axisCode="X" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="y">
           <swe:Quantity axisCode="Y" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">-0.1</swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="z">
           <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">2.0</swe:Quantity>
          </swe:coordinate>
         </swe:Location>
        </swe:location>
       </swe:Position>
      </position>
      <!--=============================================================-->
      <!-- Position of Wind Vane in Station Ref Frame -->
      <!--=============================================================-->
      <position name="windDirectionDetectorPosition">
       <swe:Position localFrame="#WIND_DIRECTION_DETECTOR_FRAME"
referenceFrame="#STATION_FRAME">
        <swe:location>
         <swe:Location definition="urn:ogc:def:phenomenon:location">
          <swe:coordinate name="x">
           <swe:Quantity axisCode="X" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="y">
           <swe:Quantity axisCode="Y" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">-0.1</swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="z">
           <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">2.0</swe:Quantity>
          </swe:coordinate>
         </swe:Location>
        </swe:location>
        <swe:orientation>
         <swe:Orientation definition="urn:ogc:def:phenomenon:orientation">
          <swe:coordinate name="z">
           <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:angle"
uom="urn:ogc:def:unit:degree">-87.0</swe:Quantity>
          </swe:coordinate>
```

```
          </swe:Orientation>
         </swe:orientation>
        </swe:Position>
       </position>
       <!--=========================================================-->
       <!-- Position of Rain Gauge in Station Ref Frame -->
       <!--=========================================================-->
       <position name="rainGaugePosition">
        <swe:Position localFrame="#RAIN_GAUGE_FRAME" referenceFrame="#STATION_FRAME">
         <swe:location>
          <swe:Location definition="urn:ogc:def:phenomenon:location">
           <swe:coordinate name="x">
            <swe:Quantity axisCode="X" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
           </swe:coordinate>
           <swe:coordinate name="y">
            <swe:Quantity axisCode="Y" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.6</swe:Quantity>
           </swe:coordinate>
           <swe:coordinate name="z">
            <swe:Quantity axisCode="Z" definition="urn:ogc:def:phenomenon:distance"
uom="urn:ogc:def:unit:meter">0.0</swe:Quantity>
           </swe:coordinate>
          </swe:Location>
         </swe:location>
        </swe:Position>
       </position>
      </PositionList>
     </positions>
     <!--=========================================================-->
     <!-- System Communication Interfaces -->
     <!--=========================================================-->
     <interfaces>
      <InterfaceList>
       <interface name="serial">
        <InterfaceDefinition>
         <!-- http://www.interfacebus.com/Design_Connector_RS232.html -->
         <applicationLayer>
          <Protocol definition="urn:davis:def:protocol:weatherLink"/>
         </applicationLayer>
         <physicalLayer>
          <Protocol definition="urn:ogc:def:protocol:RS232">
           <property name="num-bits">
            <swe:Count>8</swe:Count>
           </property>
           <property name="parity">
            <swe:Boolean>false</swe:Boolean>
           </property>
          </Protocol>
         </physicalLayer>
         <mechanicalLayer>
          <Connector definition="urn:ogc:def:connector:DB9">
           <property name="pin-out">
            <swe:Category definition="urn:ogc:def:pinout">EIA574</swe:Category>
```

```
            </property>
          </Connector>
        </mechanicalLayer>
      </InterfaceDefinition>
    </interface>
    <connection>
      <Link>
        <source ref="this/outputs/weatherMeasurements"/>
        <destination ref="serial"/>
      </Link>
    </connection>
  </InterfaceList>
 </interfaces>
</System>
</SensorML>
```

ANNEXE V

ARTICLES SUR GRIM

The Grid Resource Model for Instruments and Sensors

M. Lemay*, M. Arroll†, V. François‡
*Inocybe Technologies Inc., Montréal, QC, Canada
†California Institute for Telecommunications and Information Technology, San Diego, CA, US
‡École de Technologie Supérieure, Montréal, QC, Canada

***Abstract*— Grid applications are becoming an integral part of e-Science because of science's need for intensive computing and large knowledge base. In order to have capable applications that interact with real world science, the instruments must be accessible and represented in the grid middleware. In this paper, we present the overview, design and initial implementation of a standard model for instruments and sensors inspired by the work done for IEEE1451 and built on WSRF. This model can be used by grid applications for different purposes such as laboratory planning, instrument control and sensor monitoring.**

I. Introduction

Grid computing as been introduced as the "The next big thing in computing" [1] but even if grid technology evolves it is yet far from delivering everything it promised. The grid first started by focusing on processing and storage needs for e-Science applications. However, despite these advances, researchers quickly realized they needed network resources and instruments as an integral part of the grid middleware. These resources are crucial for e-Science because current bottlenecks of the Grid are automatic data acquisition and the need for new network management services. To solve network management issues efforts have been made by the grid community. The Global Lambda Integrated Facility (GLIF), an international virtual organization promoting optical networking has created a working group to address Control Plane issues for grid and global e-science collaboration [2]. In Canada, CANARIE has also started an initiative to solve this problem by founding the User Controlled LightPaths (UCLP) [3] project. This project provides shared network resources in the form of an Articulated Private Network (APN) given to a Virtual Organization. This APN can be used to establish end to end network connections between other grid resources providing the necessary high bandwidth needed by these grid applications. Since effort is being done to solve the network resource issue we focused on the instrument and automatic data acquisition. We are providing a model based on IEEE1451 data model [4] and point to point instrument signaling [5]. We decided to adopt the IEEE1451 standards because it provides a complete object data model that provides plug and play instrument capabilities and can adapt to any type of instrument[1] or sensors. The paper is organized as follows. Section II discusses current related work. Section III presents the use cases that motivated the model creation. Section IV gives the design goals of the Grid Resource Instrument Model (GRIM), while sections V and VI will present the approach and architecture respectively. Section VII briefly describes the implementation. Section VIII concludes with recommendations for future work.

[1] In this paper we make a distinction between instrument and sensors. The instrument contains sensors/actuators with integrated functionality while sensors's functionalities must be created externally.

II. Related Work

We will briefly describe other projects proposes an alternative solution to the same problem. For example, the Common Instrument Middleware Architecture (CIMA) project [6] has a very similar approach to ours. However, this approach lacks modularity and data level completeness. It provides a good model for instruments but the different components are not as flexible. The main problem with this approach is that the data definition is not standard for all the different instruments and the different functionalities are advertised through the use of ontologies. CIMA provides good functionality but both the service consumer and provided need to know the type of data being exchanged through the *readf()* and *writef()* commands. We will see that GRIM completes the design while providing a plug and play intercommunication environment between instruments. At the National Center for Microscopy and Imaging Research (NCMIR) in California another project created a model for grid instruments. The Generalized Service-Oriented Architecture for Remote Control of Scientific Imaging Instruments [7] project provides a basic model for instrument but all the low level is abstracted and this project mainly focuses on the graphical interface that can be used to orchestration, control and monitor these instruments. We also have to note that other projects [8] [9] also exist but they provide a solution for grid-enabled instruments without trying to generalize their solution to a standard. Finally, the myGrid project [10] focuses on extending the Grid framework and and serves the life sciences community. The main distinction between GRIM and other related work is the fact that GRIM is based on the Implicit Resource Model (WSRF) which is a subclass of normal Web Services that provides information about state in a standard way.

III. Motivation

The first motivation behind GRIM was to enable use cases needed by the Laboratory for the Ocean Observatory Knowledge Integration Grid (LOOKING) project [11]

instruments and sensors. However, seeing all the possible applications of this model, we tried to define the most flexible interfaces possible. Grid instruments will provide, because of their distributed nature capabilities that are not yet available with our current instruments. Indeed, by decoupling control and data and bringing them to service layers new levels of intelligence can be reached because all the different instruments will have a logical representation consumable by other instruments or applications. Orchestration of these instruments will provide unrivaled data flow management capabilities where instrument data can be treated , manipulated and visualized by any software application that might have no knowledge about the data it is representing or treating.(e.g. A high bandwidth oscilloscope could output its time domain data to a Fast Fourier Transform service. The resulting data could be visualized by an XY plot service thus creating a custom made Spectrum Analyser.)

Fig. 1. Grid Instrument Use Cases for the based on the LOOKING project [11] requirements.

A. Instrument Network Operations and Resource Management

The first use case and initial goal shown on Fig 1a is to be able to create an instrument network that can be operated and managed by many different applications. By using Service Oriented Architecture (SOA) applications for control and monitoring an application can easily be composed by harvesting many services of interest. Indeed, a researcher interested in visualizing the temperature of a specific region of the world could harvest all temperature services found within this region and gather the data automatically. He could then use other services like Google^{TM} Maps API to visualize the temperature data previously acquired.

B. Adaptive Instrument

The use case shown on Fig 1b allows the creation of adaptive instruments capable of enriching measurements of another instrument or even trigger actuators to react immediately to changes in the environment coming from one or many sensors/instruments. These adaptive instruments will have applications ranging from military to mass production chains because they will provide operator free advanced decision making capabilities.

C. Remote Multi-Mission Laboratory

Another interesting use case shown in Fig 1c is the remote multi-mission laboratory. Indeed, organizations who want to have a better return on investment (ROI) therefore are willing to share expensive instruments between different groups of e-science researchers. Because multiple researchers are going to use the same resources to do their experiments they will be capable of saving capital expenses. An advanced functionality that grid instruments enable, is to be provide the ability of doing multiple experiments at the same time by effectively planning instrument control using high level intelligent application proxies to the laboratories that have advanced scheduling algorithms. These virtual laboratories will become a new breed of laboratories enabling researchers world wide to have access to areas not accessible before like seabed and space. For example, oceanographers from around the world will be able to access an undersea laboratory to run their experiments. [11]

D. Coordinated Mobile Instrument Platform

Instruments that belong to these remote laboratories can also be grouped in new services that provides different functionalities. These functionalities parameters are applied to the instrument in order to achieve external choreography as shown on Fig 1d along with the initial internal coordination provided by the remote laboratory.

IV. Design Goals

A. Autonomous Instrument Architecture

This work has been inspired by the work done on the Mars Rover (ROV) designed by the Monterey Bay Aquarium Research Institute (MBARI) and the National Aeronautics and Space Administration (NASA) to provide an autonomous instrument architecture. We integrated most of the functional requirements from the rover architecture in the GRIM. MBARI's architecture2 describes accurately everything needed to have intelligent instrumentation that can perform multiple testing scenarios in an isolated environment. Since GRIM will support the same concepts it will be capable of working in harsh environments like Antarctic Research at the Davis Station [10] or more likely for our own underwater facility in the LOOKING project. This autonomous instrument architecture shown in Fig. 2 provides three layers that need to be preserved in our design.

These layers are the planning, agent and control layers. The planning layer creates experiment plans based on discovered resources, policies and required actions. The

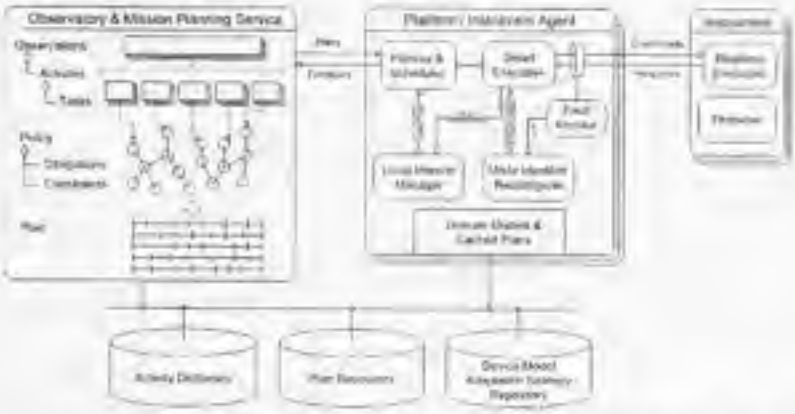

Fig. 2. Graphical representation of the Autonomous Instrument Architecture used on the Mars Rover by MBARI - courtesy of Kanna Rajan of MBARI

resulting plan is then sent to the Instrument Agent. This agent's role is to schedule the plan, supervise the execution, archive results and enforce additional policies if needed. The last but not least layer is the instrument representation and control layer responsible of acting on the physical instrument.

B. Data Flows and Orchestration

In the GRIM design we also had to take in consideration the data flow model and the orchestration possibilities between these different resources. Indeed, the data flow must be kept separate from the control flow to provide easy orchestration between different kinds of data streams. There is also a need to do buffering and some data stream verification to insure that the required rates are met by both the provider and the consumer and that there are no buffer overflow situations or incompatible data rate links. To do this orchestration, grid resources interacting with instruments must not need the knowledge of observation and control and can be generic resources reusable by many instruments.

C. Instrument Link

Since the main goal of the GRIM is to provide a plug and play interface for instruments and sensors that enables automatic data acquisition and instrument control, the relationship that exists between the software representation and the physical instrument must be reliable and verified periodically for consistency.

D. Modularity

Since the data model is decoupled from the control part, modular grid resources can be used to perform special operations on data. For example, a grid resource can be implemented to do a Fast Fourier Transform (FFT) using high performance computing on the data stream before visualization and this resource can be used to view the spectral information of any instrument or sensor that uses the GRIM model.

E. Hardware Friendliness

The GRIM must be capable of running on platforms with limited resources therefore it must have only a small mandatory framework that can fit in the resource budget of small embedded devices. It must also have telecommunication grade quality because if it is used in remote locations, it must be stable and not fail/hang because of a bad operation. Finally, since accessibility is expensive for most sensors it must provide a way to do resource service updates without restarting.

V. Approach

Indeed, the first step is to provide the communication framework between the instruments. This layer is responsible bringing all the different instruments to the same protocol which consists of the Web Service Resource Framework (WSRF) representation. Doing this allows direct ,seamless interaction between the instruments. It also consumes the network resources needed between the instruments whether these are IP tunnels or dedicated lightpaths these resources need to be accessible by the instruments. In order to establish these communication links, interaction with specialized network provisioning services such User Controlled Lightpaths (UCLP) [3] is necessary. It is also at this layer that basic security will be enforced, using plain Grid Security Infrastructure (GSI) with or without Virtual Organization Management System (VOMS), other services may at least discover available resources for which they have full or limited access.

A. Layered Frameworks

To create this new generation of instruments the technology must answer the needs expressed by these use cases. The following frameworks (Fig. 3) are needed to achieve intelligent instruments.

- Intercommunication Framework
  - Common protocol
  - Service Discovery
  - Network Provisioning
  - Security
- Control Framework

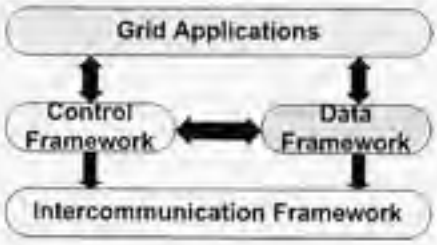

Fig. 3. Frameworks needed by grid instruments

- Policy and User Access
- Control and Operation
- Management
- Notification Handling

- Data Processing Framework
  - Introspection Analysis
  - Data Preprocessing
  - History and Archives
  - Data Streams
  - Notification Data

Because the instruments all have the same communication protocol (SOAP), they are capable of communicating with each other.

B. IEEE1451

The GRIM model is not a direct implementation of IEEE1451's to provide a web service interface to the Network Capable Access Point (NCAP) [4]. This effort has been investigated by others [12], and is not standard enough to provide grid-like functionalities required by grid middleware. Therefore, even if adaptation to IEEE1451 could easily be implemented it is not the goal we are trying to achieve in this project. The IEEE1451 framework provides many data models and classes that were used to inspire the data model provided in the following subsections.

C. Web Service Resource Framework (WSRF)

The web service resource framework offers advantages to traditional services not only by providing stateful resources but also a set of standard operations that allows different services or applications to have a basic set of interaction with the service. [13] Because the data type for these operation specified as *xsd:any*, introspection is possible even without knowledge about the actual data type.

D. WS-Notification

WSRF contributes to the creation of the Intercommunication framework. Indeed, through it's implicit resource pattern it provides the necessary introspection and discovery mechanisms. It also created the basis of a good notification framework WS-Notification / WS-Topics which was given to and reviewed by OASIS as the WSN [14]. Based on the publish/subscribe notification model, resources can be used to provide automatic event reaction when instrument informational data changes. The only thing that is needed at the intercommunication framework that is not provided by WSRF is usable network resources that can be harvested and used at runtime.

E. Data Ontology

In order to provide loose coupling between data and the grid interface that provides it an ontology model based on OWL-DL has been used. Using this ontology, services can always use an implemented model even if the XML data representation is changing. It also provides preprocessing capabilities that can be used to do intelligent introspection of instruments at the Graphical User Interface (GUI) level.

F. Semantic Web

It has been shown that the grid provides a basic communication layer of intercommunication between the services. While this is enough to provide a good interaction with services it is insufficient to understand or process the information received by the services. This is where the semantic web will be able to fill the gap left by grid middleware. Indeed, the semantics web will allow to have an accurate representation of the data models involved in the service exchange. The discovery will therefore be based on axiomatic reasoning techniques and can be optimized through multi-objective discovery algorithms. Data preprocessing will be done at all the different layers. WSMO is currently working on WSRF-S that will be a semantic extension to WSRF. The semantic web ontologies like OWL-P can also be used to describe the relationship between services (process) and policies needing to be enforced. While these approaches will help implement the full solution for grid instruments, the implementations are not mature enough yet to be incorporated in the initial model provided below. Semantic technologies will eventually be used in upcoming versions.

VI. Architecture

The Grid instrument model was inspired by IEEE1451 Software Architecture [4] and point to point transducer model [5]. This is the reason why terminology will greatly adhere to the blocks used by IEEE1451. In this model is has been implied that alike to the IEEE1451 model, the instruments might share a network connectivity. (e.g. A set of 2-3 instruments might share a GPIB bus, and therefore would share only one net-capable control point).

Fig. 4. General Model of a Grid Instrument Resource Architecture

There are three types of resources that are needed in the model (Fig. 4).

A. GRIM Entities

1) Channel Resource: The channel resource represent a data channel that resides in the transducer. Indeed, the channel resource is describe by a datastructure shown in Fig. 5 and 6 which is similar to the TEDS (Transducer Electronic DataSheet). It describes the type of data (e.g. Vector, scalar, time serie, etc) as well as a small description of the datasource. The channel resource will usually be accessed mainly for read or notification capabilities. Indeed, only some channels will have enabled write access that provide to set specific fields or actuators for the instrument. Therefore monitoring applications will directly interact with the Channel Resource for information gathering. Since the channel is a resource on its own the monitoring applications are decoupled from the control applications because there is no need to have knowledge about how to control an instrument to be able to harvest the information from it.

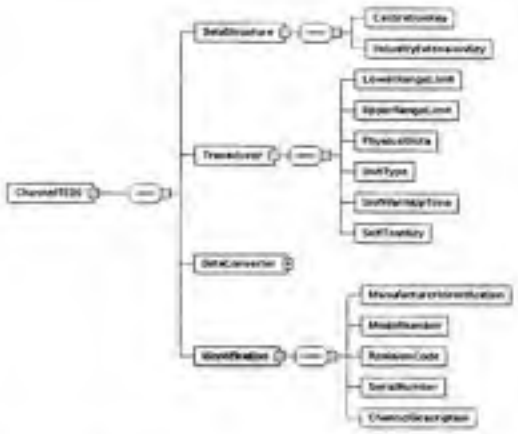

Fig. 5. Sample Channel Resource Schema based on the Transducer Electronic Datasheet (TED) of IEEE1451. PART 1

2) Transducer Resource: The transducer resource is a Channel Resource owner. Apart from ownership it also provides policy enforcement for channels as management and control operations that helps synchronize the data and refresh it at different intervals. Functional behaviors will usually interact through transducer resources providing manipulation.

3) Functional Resource: The functional resources are our normal way to interact with the instrument. Indeed, it is through these interfaces that one can control and operate the instrument. Having a seperate resources allows the same instrument to have multiple control interfaces. One might even want to have context based view if the instrument sampling is fast enough to allow it. The plan creator from higher levels will decompose the jobs for the experiment's description and will send it to the job scheduler. This schedule will overlook the execution queue of these experiments based on multi-objective optimization to prevent bottlenecks and obtain high instrument efficiency.

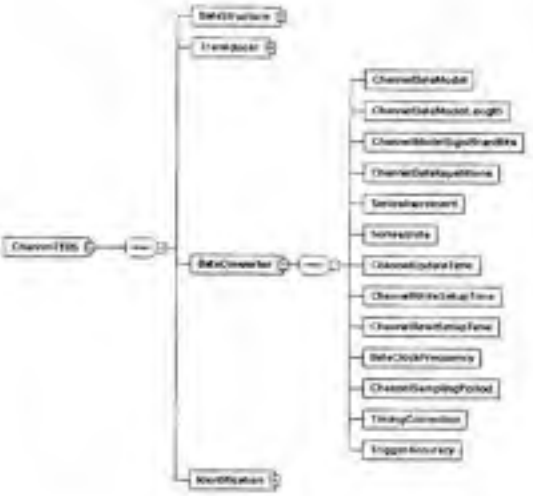

Fig. 6. Sample Channel Resource Schema based on the Transducer Electronic Datasheet (TED) of IEEE1451. PART 2

4) Access Point Resource: The access point resource mainly act as the container. It also serves as a registry for the instruments. Indeed, it manages the state, verifies that communication with the instrument is valid and applies low-level policies. This resources can also register to external registries to provide the informations about the instrument it carries.

B. Model Management

Model management is an important issues for instruments. Indeed, you want the logical representation of your instrument to always be capable of

1) Resource Creation: The creation of resources is done in a multiple stage bootstrap.

① Grid Instrument Service Platform is powered up
② Automatically discovers instruments on buses
③ Tries to find corresponding drivers
④ Loads manual instrument list
⑤ Create the resources for each instrument models
⑥ Sets resource status to inactive.
⑦ Registers the resources' WS-Addresses in platform's service group.
⑧ Attempts to initiate communication with instrument
⑨ Sets resource status to active.

These creation steps will insure the fact that a resource can communicate with the instrument. It must be noted that a continuous verification of communication state between the platform and the instrument is needed. If there is a communication problem the corresponding instrument resource and it's owned resources will be set as inactive.

2) Registration and Discovery: The registries are handled by having service groups that adhere to the ownership model that exists between the different types of resources.

3) State and Ownership: The ownership model is described by Resource Description Format (RDF) inside the service group structure. It denotes the relationship that

exists between the different resources in a grid instrument model. This notion of ownership is important because it allows easy management of state and policies for a set of resources. Indeed, if a high level resource goes down it must also disable all the owned resources that depends on this high level functionality.

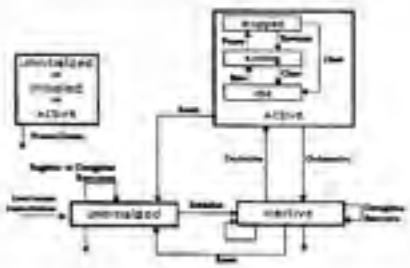

Fig. 7 State Model from IEEE1451-1

The Fig. 7 from IEEE1451-1 [4] shows the state machine at the base of all resources created with the GRIM architecture. The uninitialized state is reached when the logical resource exists but the communication with the instrument is not yet established. Once it is established it will go in an inactive mode and will register with the parent resource. The parent resource will put the resource in active mode after performing a brief validity check.

4) Workflow and Orchestration:

VII. Implementation

This section briefly describes how the model is being implemented and what type of hardware it supports. We promote the adoption of GRIM as a primary instrument model that can be extended as needed to provide all required functionalities to model instruments/sensors or actuators.

A. Desktop Instrument Controllers

To validate the model and to do a proof of concept, the GRIM is being implemented in Java to provide rapid desktop instrument application development as well as visualization components and graphical user interfaces. This version has no resource limitations and could hold many instruments on the same machine.

B. Embedded Linux Devices

In real world applications the sensors/instruments will most likely be controlled by microcontroller or mobile microprocessors. It is the reason why GRIM is also being ported to the uCLibC and uCLinux (Linux for non MMU microcontrollers such as the ARM7TDMI processor). Of course, if advanced functional blocks are required and preprocessing is complex or controls multiple instruments we are also planning to port GRIM to the new ARM926EJ-S family of microprocessors with ARM Linux.

VIII. Conclusion and Future Work

The proposed model is being implemented without semantic web functionalities. The next step is to have these capabilities to be able to discover and harvest grid instruments. Also, a planning mechanism should be exploited to do job dispatch and preprocessing before sending them to the functional resources of the instruments. For more information about the GRIM implementation and its applications you may visit http://www.inocybe.ca/grim/.

Acknowledgment

The use case pictures were graciously provided by Alan Chave from Woods Hole Oceanographic Institution member of the LOOKING project which is the biggest motivator for this model. We acknowledge Joe Bester's work from Argonne National Laboratory and a *Globus Toolkit* developper for it's help in porting Globus Toolkit 4 C WS Runtime to an ARM7 Microcontroller. Finally, we also would like to thank Tony Fountain of San Diego Supercomputing Center UCSD for its participation and initiative in the project.

References

[1] R. Deimer, "Unlocking the power of the Grid," *IEE Review*, vol. 48, no. 3, pp. 9–12, 2002.

[2] G. Kamous-Edwards, "Global e-science collaboration," *Computing in Science & Engineering [see also IEEE Computational Science and Engineering]*, vol. 7, no. 2, pp. 67–74, 2005.

[3] J. Wu, H. Zhang, S. Campbell, M. Savoie, G. Bochmann, and B. St Arnaud, "A grid oriented lightpath provisioning system," in *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, 2004, pp. 395–399.

[4] "IEEE Standard for a Smart Transducer Interface for Sensors andActuators-Network Capable Application Processor (NCAP) Information Model," 2000.

[5] "IEEE standard for a smart transducer interface for sensors andactuators - transducer to microprocessor communication protocols andTransducer Electronic Data Sheet (TEDS) formats," 1998.

[6] T. Devadithya, K. Chiu, K. Huffman, and D. McMullen, "The Common Instrument Middleware Architecture: Overview of Goals and Implementation," in *e-Science and Grid Computing, First International Conference on*, 2005, pp. 578–585.

[7] T. Molina, G. Yang, A. Lin, S. Peltier, and M. Ellisman, "A Generalized Service-Oriented Architecture for Remote Control of Scientific Imaging Instruments," in *e-Science and Grid Computing, First International Conference on*, 2005, pp. 550–556.

[8] J. Frey, S. Peppe, M. Surridge, K. Meacham, S. Coles, M. Horsfall, M. Light, H. Mills, D. De Roure, G. Smith, and E. Zaluska, "Grid-Enabling an Existing Instrument-Based National Service," in *e-Science and Grid Computing, First International Conference on*, 2005, pp. 570–577.

[9] F. Villeneuve-Seguier, "Reprocessing D216: Data with SAMGrid," in *e-Science and Grid Computing, First International Conference on*, 2005, pp. 557–561.

[10] M. Radenkovic and C. Greenhalgh, "Emerging infrastructures and platforms: science and collaboration," in *Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003. 6th International Conference on*, vol. 1, 2003, pp. 395–398 vol.1.

[11] A. Chave, M. Arrott, L. Smarr, J. Orcutt, E. Lazowska, J. Delaney, and M. Abbott, "LOOKING: Cyberinfrastructure for Ocean Observatories," in *SSC06*, 2006.

[12] F. E. Sadok and R. Liscano, "A Web-Services Framework for 1451 Sensor Networks," in *IEEE Instrumentation and Measurement*, 2005.

[13] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, "Modeling and managing State in distributed systems: the role of OGSI and WSRF," *Proceedings of the IEEE*, vol. 93, no. 3, pp. 604–612, 2005.

[14] "WS-Notification Specification," 2004.

Virtualization of Photonics Devices in Optical Lab Environment

Mathieu Lemay[1], Véronique François[1], Alex Vukovic[2]

[1] *École de Technologies Supérieure, 1100 Notre-Dame St. W, Montréal, QC, Canada, H3C 1K3*
[2] *Communications Research Centre, 3701 Carling Ave., Ottawa ON, Canada, K2H 8S2*
mlemay@inocybe.ca, vfrancois@ele.etsmtl.ca, alex.vukovic@crc.ca

Abstract: This paper presents the GRIM, an instrumentation middleware, developped to create self-adaptive networks using conventionnal photonic- devices. The GRIM architecture is outlined and implemented in optical lab environment for validation and verification.

OCIS codes: (060.1155) All-optical networks; (060.4250) Networks

1. Introduction

Grid computing techniques and Web technologies have been evolving over the latest years with the emergence of the Web Services. The trend is at using virtualized versions of the physical resources through appropriate middleware. The increasing demand in bandwidth created by the new way the Internet is now being used, where users are as much creators of the content and traffic because of peer-to-peer or user-hosted services, causes network architectures to require reconfigurability. Different middleware intiatives, including UCLP and GENI [1], aim at controlling optical network resources. However, all these middleware control Network Elements, e.g. control Layer-2 or -3 integrated devices, thus only providing a limited view of the Layer-1 instruments functionalities. Conversely, GRIM, the short form for Grid Resources for Instruments Models, is an instrument middleware whose purpose is to provide a uniform representation for *any* type of physical device, instrument or sensor in a network. Using GRIM, complex devices like Reconfigurable Optical Add-Drop Multiplexers (ROADM), Optical Channel Performance Monitors (OCPM) and Optical Spectrum Analysers (OSA), etc., latest or dated, can be virtualized and used in a central management interface, adding monitoring and instrumentation capabilities to the network. This paper explains values of applying GRID techniques in optical network, following by explanation of GRIM model architecture. Finally, the paper demonstrate practical application of GRIM model in sharing devices / instruments in optical lab environment.

2. Grid Techniques in the Optical Network

Most novel photonic instruments come with limited control capabilities and are used in the laboratories until mainstream telecom integrators provide them with integrated hardware platforms and controllers. However, network operators may want to control some instruments by themselves for greater flexibility and to keep using previous-generation, perfectly working instruments. They also might want to have their monitoring instruments available remotely to prevent a truck-roll to each site where a problem occurs. Policies can be put in place to properly react to different situations, firstly, to minimize human interactions and to allow putting the staff to a better use and, secondly, to lower the time needed for restoration. Photonic Instruments Virtualization brings such benefits to network operators. Following are three reasons why it is important to take the steps to virtualize photonic instruments and how it will bring the operational costs down: 1) virtualization makes the equipment vendor-independent: instruments from different vendors can be incorporated into the same network architecture; 2) software representations can be managed with more ease than their hardware counterparts; 3) virtualized equipment may interact with any other middleware, regardless of protocol or type of equipment, thus creating *Network Enabled Platforms*.

3. GRIM Middleware Model

The GRIM is a middleware created to quickly and easily virtualize devices, instruments and sensors and to allow them to interact via the World Wide Web. It aims at being as versatile and easy-to-use as a tool like Labview, yet providing a global-scale range of interactions with resource sharing capabilities and the possibility of being consumed by any applications or other forms of middleware.

A regular personal computer or a simple Advanced RISC Machine (ARM) based microcontroller are necessary to connect an instrument to a grid. The instrument side protocol is a lightweight protocol called GRIMML, which

can be transported over plain TCP or TLS(SSL) sockets. All the virtual representation is done on the middleware servers, which runs the factory and modelers objects. However, the deployment of the services is only a suggestion and everything could run on the instrument's controller as long as it has enough processing power and memory, thus a ARM9 microprocessor with 128Mb of RAM would be the minimal requirements of such a deployment. On the implementation side, the GRIM builds virtual instruments using blocks, allowing intuitive assemblies and re-use of previously developed blocks. Yet, it follows the object structure of the IEEE 1451.0 Standard for a Smart Transducer Interface for Sensors and Actuators[2], creating Web Service Resource Framework (WSRF) resources that represent the instrument in a uniformized manner. Moreover, the GRIM implementation is open-source and thus can be used for a variety of different purposes and adapted at will, based on a project's requirements.

Fig. 1 shows the GRIM architecture. The XML Engine is at the heart of the implementation and it is the glue between the transport, which talks to the instrument via different protocols, and the GRIMML document , which is the actual description of the virtualized instrument. Therefore, only a basic knowledge of XML is needed to add new instruments to a GRIM grid.

Figure 1. GRIM Middleware Architecture

Physical- devices are virtualized in GRIMML and sent to the modeler component via a TCP or SSL socket; the modeler is tied to the Factory and creates the objects defined by the GRIMML document. It does so using two types of building blocks : a *TransducerBlock*, is a block with an *update()* operation that can be called manually or set to be invoked periodically ; a *FunctionBlock* which executes a set of functions via an *execute()* operation. Blocks may have a number of components, namely the *Parameters*, *ParameterWithUpdate* and *PhysicalParameters*, which are used to describe the various input and output parameters of the block. While *Parameters* are just value container and don't actually talk to the instrument, the *ParameterWithUpdate* and *PhysicalParameters* issue read and write requests to the device's values using their appropriate protocol.

The GRIM middleware allows to build an event driven architecture: once the device / instrument has its blocks and parameters, grid utilities and tools must be able to listen to these resources and to harvest any change in parameter values, thus allowing appropriate reactions within a reasonable timeframe. Each parameter is both a notification producer and a consumer. It is thus possible to create *links* by selecting which resources subscribe to what. Notifications made with WS-Notifications always flow from the producer to the consumer, thus making a unidirectional relationship between two parameters with compatible units. If a bidirectional relationship is needed, both resources must register to each other's notification. However, round-trip notifications are dangerous as instruments reaction may cause hysteresis or oscillations and such scenarios must be handled carefully, for example by adding a few external blocks to supply the notification loop with intelligence.

As GRIM was created to control and monitor devices, instruments and sensors, it aims at replacing the human response times, not at providing a dynamic network framework or protocol for network protection or adaptive feedback to transient response. However, with monitoring reaction times from 500 ms to 5 s, it is fast enough to be used for normal human-network interactions such as wavelength add/drops. For instance, a torn-down link could be brought back up within 5 s using ROADMs; while this is not good enough to replace standard protection and restoration techniques, it may complement them in case of a more fatal failure, for example when both the working and the protection fibers are cut at the same time and a new path or dark fiber needs to be lit up.

As all instruments are just groups of *Blocks* and *Parameters*, they look the same on a grid and there is no way of identifying exactly which instrument/device is being interacted with. Therefore, to provide more application level capabilities, standardized ways to model such devices need to be applied, to provide "interdomain" capabilities. For example, if the equipment performs a wavelength switching functionality, then a "standard" wavelength switching service will have to be created on top, to be used in a logical platform.

4. Optical-Lab Use Case

To validate the GRIM model, a use case had to be identified, that would, firstly, justify and guide, and secondly, require the virtualization of meaningful photonic equipment. This initial use case was laboratory equipment sharing

in a classroom. Because up-to-date optical equipment is expensive in general, it is highly desirable to share the few high-priced instruments amongst as many students as possible, allowing planning of more experiments at an acceptable cost. An EDFA, an OSA and an optical switch that needed to be shared were modeled using GRIMML. The EDFA has a single *FunctionBlock* that is used to change the settings as required, while a single *TransducerBlock* automatically pulls it. Both the OSA and the optical switch are modeled using a single *FunctionBlock* because they are only activated on demand. The resources were then orchestrated in the scenario shown in Fig. 2. In order to *book* the resources, a Scheduler *FunctionBlock* was externally created to provide a simple, Token-based scheduler. The users would simply invoke the orchestration to retrieve the token, switch the optical switch to their user fiber, take the measurement and return the token.

Figure 2. GRIM Resource Orchestration.

Systematic experiments were performed to investigate the impact of instrument virtualization on the queue duration. A series of ten OSA measurements with various, predetermined parameters (span, center wavelength, resolution and sensitivity) was chosen in order to eliminate uncontrolled parameters. In a first set of experiments, users had to manually select their fiber with the optical switch, manually enter the OSA settings, wait for the OSA to perform the scan and save the data on a floppy disc. In a second set of experiments, all these steps were performed automatically using the GRIM Virtual OSA interface on each student's work bench computer. Each set of experiments was reproduces 4 times, for statistics. The measured median times were 75 s and 32 s for the manual and the automated measurements, respectively. The time to perform each measurement was thus reduced by more than 50 %, which is appreciable. Yet, one could argue that it only corresponds to an economy of 30 s of actual time; which is true if there is a single user. However, remembering the context, there might be 2 or 3 students in queue, waiting for the instruments to be available. The saved time has to, therefore, be multiplied by the number of request in the queue. This equipment sharing configuration proved to very effectively reduce students frustration (and chatting) during the lab hours. It would not have been possible using LabView, as it can't schedule individual grid resources; thus the sharing would have been external only. Also because of security reasons the Labview remote control ports are not allowed through university's firewall while HTTP and HTTPS (respectively ports 80 and 443) traffic is allowed.

5. Future Work

While the described use case is simple, the adaptive nature of the GRIM middleware makes it possible to create more complex orchestrations using the event-driven architecture capabilities. We are currently creating ROADM GRIMML models to control and route specific wavelengths to the various instruments. Also, as a collaboration to the AIRON program at the Communications Research Center[3], we are pushing GRIM's middleware adaptive capabilities to have the network react automatically to different network statuses, with the goal of reducing the operational costs. For instance, we aim at complementing the action of agile optical amplifiers by remotely changing their nominal gain setting as the span loss increases due to wear out. The GRIM middleware is expected to provide administrators with another tool to help taking appropriate reactions based on the global status of their network, rather than on localized measurement.

References

1. Vassilis Prevelakis and Admela Jukan. How to buy a network: trading of resources in the physical layer. *Communications Magazine, IEEE*, 44(12):94–102, 2006.
2. IEEE1451, IEEE Standard for a Smart Transducer Interface for Sensors andActuators-Network Capable Application Processor (NCAP) Information Model, 2000.
3. A. Vukovic, M. Savoie, H. Hua, and M. Lemay. Development of an autonomous intelligent reconfigurable optical network. *Photons Journal*, 5(1):p.48–50, May 2007.

LISTE DE REFERENCES

Chave, Alan D., Matthew Arrott et Larry Smarr. 2006. « LOOKING: Cyberinfrastructure for Ocean Observatories ». In *Scientific Submarine Cable 2006 (SSC'06)*. (Dublin, Feb. 8-10 2006), p.39-45.

Corbató, F. J. et V. A. Vyssotsky, 1965. « Introduction and overview of the Multics system », In *1965 Fall Joint Computer Conference*, Spartan Books. (Washington D.C. ,1965), p.185-196.

Cristaldi, Loredana, Alessandro Ferrero et Antonello Monti. 2005, « A virtual environment for remote testing of complex systems ». In *Instrumentation and Measurement, IEEE Transactions on* **54**(1), p.123-133.

Dettmer, Randy. 2002. « Unlocking the power of the Grid ». In *IEE Review* **48**(3), p. 9-12.

Devadithya, T. *et al.*, « The Common Instrument Middleware Architecture: Overview of Goals and Implementation », In *Proceedings of the First IEEE International Conference on e-Science and Grid Computing (e-Science 2005)*, Melbourne, Australia, Dec. 5-8, 2005. Consulté http://doi.ieeecomputersociety.org/10.1109/E-SCIENCE.2005.77, le 7 Septembre 2005.

Foster, Ian, *et al.* 2005. « An analysis of notification related specifications for Web/grid applications », In *Information Technology: Coding and Computing 2005 (ITCC 2005)*, p. 762-763 Vol. 2.

Foster, Ian *et al.* 2005. « Modeling and managing State in distributed systems: the role of OGSI and WSRF », In *Proceedings of the IEEE* **93**(3), p.604-612.

Foster, Ian *et al.* 2002. « A low-cost Internet-enabled smart sensor », In *Sensors, 2002. Proceedings of IEEE*, p. 1549-1554 vol.2.

Foster, Ian *et al.* 2000, « The design of distributed measurement systems based on IEEE1451 standards and distributed time services », In *Instrumentation and Measurement Technology Conference 2000 (IMTC 2000)*, p. 529-534 vol.2.

Foster, Ian et Carl Kesselman. 1997. « Globus: A Metacomputing Infrastructure Toolkit », In *The International Journal of Supercomputer Applications and High Performance Computing* **11**(2), p. 115-128.

IEEE. 2000. « IEEE Standard for a Smart Transducer Interface for Sensors andActuators-Network Capable Application Processor (NCAP) Information Model », *IEEE Std 1451.1-1999* .

IEEE. 1998. « Standard for a smart transducer interface for sensors andactuators - transducer to microprocessor communication protocols andTransducer Electronic Data Sheet (TEDS) formats », *IEEE Std 1451.2-1997* .

Lee, Kang. 2000. « IEEE 1451: A Standard in Support of Smart Transducer Networking », In *Instrumentation and Measurement Technology Conference, 2000 (IMTC 2000), p.* 525-528 vol.2.

.

Lee, Kang. 1999. « A Synopsis of the IEEE P1451- Standards for Smart Transducer Communication », National Institute of Standards and Technology, 1999. En ligne. <http://ieee1451.nist.gov/1451synosis-599F.pdf>. Consulté le 12/02/2006.

Lemay, Mathieu , Matthew Arrott et Véronique François, 2006. « The Grid Resource Model for Instruments and Sensors » In *Industrial Electronics, 2006 IEEE International Symposium*, Montréal, July 2006, p. 2824 – 2829, vol.4.

Morgan, D., « The Multics System », *Communications, IEEE Transactions on [legacy, pre - 1988]* , Oct 1973, p. 1166-1167, vol.21, no.10

OASIS. 2006. « Web Service Resource 1.2 (WS-Resource) », *OASIS Standard, 1 April 2006*. En ligne. <http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf>. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Resource Properties 1.2 (WS-ResourceProperties) », *OASIS Standard, 1 April 2006*. En ligne. <http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf>. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Resource Properties 1.2 (WS-ResourceProperties) », *OASIS Standard, 1 April 2006*. En ligne. <http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf>. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Resource Lifetime 1.2 (WS-ResourceLifetime) », *OASIS Standard, 1 April 2006*. En ligne. < http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf>. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Base Faults 1.2 (WS-BaseFaults) », *OASIS Standard, 1 April 2006*. En ligne. < http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf>. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Base Notifications 1.3 (WS-BaseNotifications) », *OASIS Standard, 1 October 2006*. En ligne. < http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf >. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Topics 1.3 (WS-Topics) », *OASIS Standard, 1 October 2006*. En ligne. < http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf >. Consulté le 7 juin 2007.

OASIS. 2006. « Web Service Resource 1.2 (WS-Resource) », *OASIS Standard, 1 April 2006*. En ligne. <http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf>. Consulté le 7 juin 2007.

OGC. 2006. « OpenGIS® Transducer Markup Language Implementation Specification », *Open Geospatial Consortium, 22 December 2006*. En ligne. < http://www.transducerml.org/downloads/TMLspec_100.doc >. Consulté le 6 mars 2007.

OGC. 2006. « OpenGIS® Sensor Model Language (SensorML) », *Open Geospatial Consortium, 5 May 2006*. En ligne. < http://portal.opengeospatial.org/files/?artifact_id=13879&version=2&format=doc >. Consulté le 10 Octobre 2006.

Rambhia, Ajay M. (2002), *XML Distributed Systems Design*, 1ère édition. Indianapolis : Sams, 404 p.

Recio, Joaquim *et al.* , Evolution of the user controlled lightpath provisioning system, Transparent Optical Networks, 2005, Proceedings of 2005 7th International Conference,Volume 1, July 2005, p.263 – 266.

Sadok, Emil et Ramiro Liscano, 2005, « A Web-Services Framework for 1451 Sensor Networks », In *Instrumentation and Measurement Technology Conference, 200 (IMTC 2005)*, p.554-559.

Wu, Jing et al. 2004. « A grid oriented lightpath provisioning system », In *Global Telecommunications Conference Workshops, 2004*, p. 395-399.