Développement de services de grille standardisés : retour d'expérience

Sylvain Reynaud
CNRS – Centre de Calcul de l'IN2P3
27, Bd du 11 Novembre 1918, 69622 Villeurbanne Cedex
sylvain.reynaud@in2p3.fr

Fabio Hernandez
CNRS – Centre de Calcul de l'IN2P3
27, Bd du 11 Novembre 1918, 69622 Villeurbanne Cedex
fabio.hernandez@in2p3.fr

Résumé

Une grille de calcul se caractérise par des utilisateurs et des ressources dispersés et hétérogènes, un état des ressources matérielles (calcul, stockage…) et logicielles variable, ainsi que des groupes d'utilisateurs et une connectivité réseaux également variables au cours du temps. Il apparaît clairement dans un tel contexte que l'utilisation de standards est un point particulièrement critique pour la réussite de la mise en place d'une infrastructure de grille, notamment pour faire face aux problèmes d'hétérogénéité du matériel et des logiciels. Il ressort également de cette liste le besoin d'avoir des mécanismes permettant de gérer plus facilement la volatilité de l'état des différents composants de la grille, qu'il s'agisse des ressources matérielles ou logicielles, de la constitution des groupes et de leur connectivité, etc.

La spécification de l'infrastructure Open Grid Service Infrastructure (OGSI) [3] a pour objectif de normaliser les services composant les grilles, afin de garantir l'interopérabilité de systèmes hétérogènes pour le partage et l'accès à des ressources de calcul et de stockage distribuées. Cette spécification définit ce qu'est un Grid service ; un Web service respectant un ensemble de conventions (interfaces et comportement) adaptées aux contraintes de l'environnement de grille.

La définition de la norme OGSI a donné lieu récemment (première version stable fin juin 2003) à une implémentation en Java de cette infrastructure. Cette implémentation est intégrée dans la version 3 de Globus Toolkit [5][18]. Dans cet article, nous présentons, en l'illustrant par un cas d'utilisation, notre retour d'expérience sur le développement de Grid services et leur déploiement sur une grille de test (DataGrid [27]).

Mots clefs

OGSI, GT3, grille, Grid service, Web service

1 Les concepts de OGSI

1.1 Le contexte

La problématique à la base du concept de grille est la « coordination du partage de ressources et de la résolution de problèmes dans des organisations virtuelles dynamiques et multi-institutionnelles » [1]. Selon l'équipe du Globus Project™, les critères clefs du concept de grille apparaissent dans la proposition suivante : « coordination de ressources distribuées en utilisant des protocoles et interfaces standards, ouverts et génériques, dans le but de fournir une qualité de service non triviale » (GlobusWorld, San Diego, january 14, 2003).

L'objectif de la norme OGSI [3] est de faciliter le développement de Web services adaptés aux contraintes spécifiques à un environnement de grille. Cette norme définit un ensemble d'interfaces et comportements communs aux services susceptibles d'être proposés dans une grille. Elle est écrite en terme de descriptions de Web services (en WSDL[1] [8] + extensions GWSDL[2]) et de types XML (en XML Schema [11]).

[1] Web Services Description Language : Format XML défini par le consortium W3C, permettant la description des fonctionnalités et de la manière d'utiliser un Web service.
[2] Grid WSDL extensions : Extension au WSDL 1.1 permettant l'expression de la norme OGSI dans ce langage. Il s'agit d'une solution temporaire, qui sera à l'avenir remplacée par la version 1.2 du WSDL intégrant les notions définies par GWSDL (héritage de port-type et open-content).

Un Grid service est un Web service qui respecte les interfaces et les comportements définis dans la norme OGSI. Il implémente un certain nombre d'opérations standard, dont certaines sont obligatoires (e.g. gestion du cycle de vie, accès à l'état interne du service), et d'autres facultatives (e.g. notification, création d'instance de service, etc.).

Dans un environnement d'exécution compatible OGSI, tel que celui proposé par Globus Toolkit 3 (GT3 core), l'implémentation des opérations standard décrites dans la norme est fournie par l'environnement (OGSI Reference Implementation) et peut être intégrée à un Grid service grâce à l'héritage de port-type (GWSDL/WSDL 1.2). Le contrôle du cycle de vie des services et l'acheminement des requêtes reçues sont gérés par le Grid Service Container. L'environnement d'exécution OGSI fournit également des API et outils pour le développement de Grid services (OGSI Reference Implementation) et la sécurité des messages échangés (Security Infrastructure) [5][Figure 1].

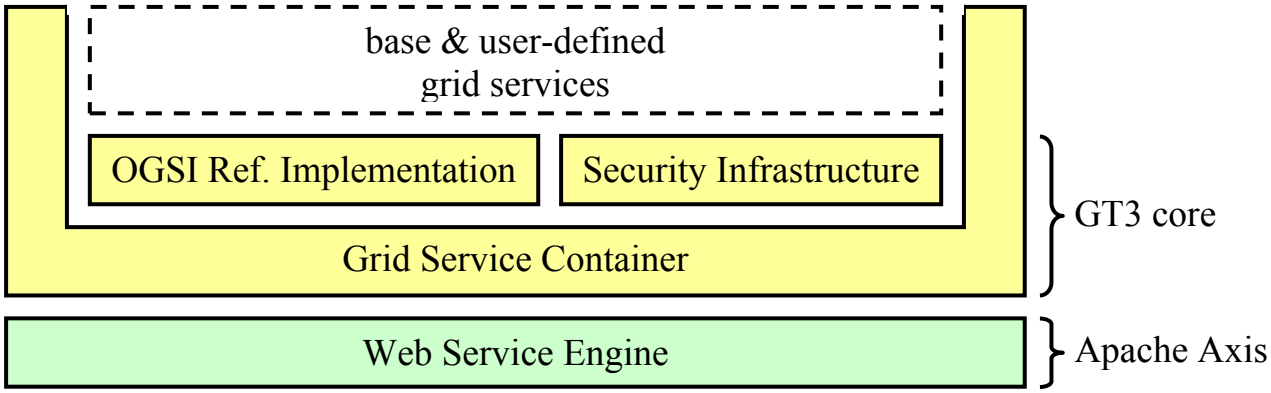

Figure 1 : *Environnement d'exécution OGSI (GT3 core)*

1.2 Instance de service volatile

Un apport important de la spécification OGSI est la possibilité de créer des instances de services volatiles, dont l'état interne et la durée de vie de chaque instance d'un même service sont indépendants. Un Grid service n'est pas obligatoirement une instance volatile, il peut être persistant comme le sont les Web services. C'est le cas par exemple des fabriques d'instances de Grid service, qui sont elles-mêmes des Grid services. Cependant, le choix d'une architecture à base d'instances de service volatiles permet notamment d'implémenter plus facilement la gestion de la reprise d'un service et la restauration de son état interne après une faute.

Une instance de service est créée par un service de fabrique, qui lui, est généralement un Grid service persistant. Cette instance est alors associée à une durée de vie et peut être détruite, soit de façon explicite, soit au terme de sa durée de vie (« soft-state lifetime management »). Tout Grid service doit obligatoirement implémenter l'opération '`setTerminationTime`', qui permet de modifier à distance la durée de vie de chaque instance. Dans le cas d'une gestion de la durée de vie par « soft-state lifetime management », l'instance de service est maintenue en vie grâce à l'invocation périodique de cette opération par les applications ou autres services utilisant cette instance, afin de prolonger la durée de vie de l'instance tant que celle-ci continue à être utilisée. Lorsque plus aucune application n'utilise une instance de service, celle-ci finit par mourir. Cette façon de gérer la vie des instances de service permet d'éviter au développeur une gestion complexe de l'état du service.

1.3 Désignation et liens ('bindings')

Une instance de Grid service, qu'elle soit volatile ou persistante, est accessible via un Grid Service Handle (GSH). Un GSH est une simple URI, et ne contient pas suffisamment d'information pour permettre à un client de communiquer directement avec l'instance de service. Il est donc nécessaire, avant de pouvoir utiliser une instance de service, que le GSH soit d'abord résolu en une Grid Service Reference (GSR). Un GSR contient toute l'information nécessaire au client pour communiquer avec l'instance de service, et sa forme dépend du mécanisme de 'binding' utilisé. Par exemple, un GSR est encodé sous la forme d'un IOR[3] si RMI-IIOP[4] [22] est utilisé, et sous la forme d'un document WSDL si SOAP[5] [9] est utilisé.

[3] Interoperable Object Reference : Donnés utilisées par la application CORBA pour déterminer comment se connecter à un serveur et envoyer des requêtes à un objet distant.
[4] RMI over Internet Inter-ORB Protocol : Technologie RMI basée sur le protocole standard IIOP (supporté par tous les produits CORBA inter-opérables).
[5] Simple Object Access Protocol : Protocole basé sur XML, permettant l'échange d'information entre des applications écrites dans différents langages.

L'implémentation de OGSI intégrée dans le Globus Toolkit 3 [18] est basée sur le moteur SOAP Apache Axis [19], et encode donc les GSR sous la forme de document WSDL.

Une même instance de service peut être associée à plusieurs GSR, par exemple lorsque celle-ci supporte plusieurs protocoles de communication. Un même GSH peut donc être résolu en différents GSR suivant le client, ou suivant le moment où la résolution a lieu. Afin de permettre au client de choisir le GSR (ou le type de GSR) qu'il souhaite utiliser, plusieurs GSH, chacun associé à un seul GSR (ou un seul type de GSR) par exemple, peuvent être proposés pour accéder à une même instance de service.

Quel que soit son format, un GSR est associé à une période de validité. Cette période est décorrélée de la durée de vie de l'instance qu'elle référence, et elle est également différente de la durée de validité réelle du GSR. En effet, la date de fin de validité d'un GSR ne garantit pas que celui-ci restera valide jusqu'à cette date, ni qu'il ne sera plus valide à partir de cette date, mais elle permet en revanche de mettre à jour le GSR de façon proactive, à un moment où les ressources locales sont peu sollicitées, plutôt que de le faire au moment où l'utilisateur est en attente d'une réponse à une requête qu'il vient de soumettre à l'instance de service.

1.4 Modèle d'information

Autre apport intéressant de la norme OGSI, les « service data » sont des sortes d'attributs publics[6] de services qui représentent l'état courant d'une instance de service. Un certain nombre de « service data » doivent obligatoirement être définis dans chaque Grid service (par exemple le « service data » contenant la liste des « service data » du service), le développeur pouvant ajouter à sa guise de nouveaux « service data » afin d'exposer d'autres informations sur l'état courant du service. Comme pour le GSR d'un service, tout ou partie d'un « service data » peut être associé à une période de validité. Les mécanismes mis en œuvre pour manipuler ces « service data » constituent les outils de base pour le monitoring des services, ainsi que pour la découverte des informations qu'ils exposent.

Les « service data » sont accessibles par les utilisateurs du service en lecture, soit en mode 'pull', c'est-à-dire à la demande, soit en mode 'push' grâce au mécanisme de notification (voir chapitre 1.5 ci-dessous). Ils sont également accessibles en écriture, selon les règles de modifications exprimées dans la description du service. Une requête de lecture de « service data » peut exprimer des contraintes concernant plusieurs « service data » de l'instance, et plusieurs « service data » peuvent être envoyés dans la réponse d'une requête de lecture. De même, plusieurs « service data » peuvent être modifiés de façon atomique, en une seule requête d'écriture.

Par rapport à une approche avec un « accesseur[7] » par attribut de service, au-delà d'une réduction des coûts de développement et de maintenance du service, la notion de « service data » apporte une solution au problème de l'atomicité de la lecture ou de l'écriture d'un groupe d'attributs. Par exemple, la lecture en deux temps à deux attributs complémentaires tels que les mémoires physique et virtuelle d'une machine, ne permet pas d'obtenir une vue cohérente du système. L'approche avec un « accesseur » par attribut ne permet pas non plus l'expression d'une requête avec des contraintes sur plusieurs attributs. L'autre approche permettant de répondre aux besoins d'atomicité et d'expression de requêtes sur plusieurs attributs, est l'implémentation d'un « accesseur » pour chaque combinaison d'attributs, mais cette approche devient rapidement inenvisageable lorsque le nombre d'attributs devient important (*n!* « accesseurs » pour *n* attributs) [4]. De plus, des « service data » peuvent être ajoutés dynamiquement pendant l'exécution du service.

Bien que le service présente ses « service data » aux utilisateurs sous forme d'éléments XML [10], les services ne sont pas tenus de manipuler effectivement du XML pour des raisons d'efficacité. Cependant, le choix d'exposer les « service data » dans un format XML permet l'utilisation de standards tels que le XML Schema [11] pour la description des types et de la structure de chaque « service data », ou bien tels que XPath[8] [12] ou XQuery[9] [13] pour l'expression des requêtes d'accès à des « service data ». De même, on peut imaginer utiliser XSLT [14] pour l'affichage des « service data » d'un service dans une page web par exemple.

---

[6] Par analogie aux attributs publics de classe en programmation orientée objet.
[7] Opération permettant l'accès en lecture ou en écriture à un attribut ('get' ou 'set').
[8] XML Path Language : Langage permettant de désigner des parties d'un document XML.
[9] XML Query Language : Langage permettant d'exprimer de requête sur un document XML.

1.5 Gestion de la souscription à des notifications

Le mécanisme de notification défini dans OGSI permet à un client de s'enregistrer pour être notifié lors de la modification d'un « service data », lorsque le service le supporte. Les messages de notification alors envoyés aux abonnés contiennent la nouvelle valeur du « service data » concerné. Cette utilisation du mécanisme de notification est donc équivalente à un accès en mode 'push' à un « service data » par son nom.

Ce principe peut être généralisé à d'autres modes de souscription à des messages de notification. Par exemple, si le service le supporte, la souscription aux notifications peut se faire avec une expression XPath, lui permettant ainsi d'obtenir plusieurs « service data » en un seul message de notification, et d'écrire des expressions booléennes complexes afin de restreindre le nombre de messages de notification envoyés à un abonné à ceux qui satisfassent la condition exprimée. Ce type de souscription permet donc de limiter le trafic en n'envoyant que les messages pertinents pour l'abonné. La liste des modes de souscription supportés par un service est accessible via le « service data » nommé « `subscribeExtensibility` ».

La gestion de la souscription à une notification est effectuée par une instance de Grid service. Cette instance est maintenue en vie par l'abonné grâce au mécanisme de « soft-state lifetime management » décrit dans le chapitre 1.2 ci-dessus. La souscription à une notification finit donc par disparaître lorsqu'un client se déconnecte brutalement du réseau sans annuler explicitement sa souscription, ce qui permet d'éviter de continuer à envoyer des messages de notification inutilement.

2 Retour d'expérience

Dans cette section nous aborderons quelques spécificités liées au développement de Grid services avec GT3 (Globus Toolkit 3), quelques difficultés rencontrées et les solutions que nous y avons apportées.

2.1 Processus de développement

Comme pour n'importe quel Web service, un Grid service s'appuyant sur les standards SOAP et WSDL se compose de plusieurs éléments :
- Un document WSDL décrivant les interfaces (section 'porttype') et les protocoles (section 'bindings') supportés par celui-ci,
- L'implémentation des opérations supportées par le service, avec le langage choisi par le développeur,
- Eventuellement des stubs permettant aux utilisateurs de communiquer avec le service sans avoir à manipuler eux-mêmes des documents SOAP pour construire leurs requêtes ou extraire l'information pertinente des réponses du service.

Comme pour Apache Axis [19] sur lequel GT3 s'appuie, des outils sont proposés pour la génération automatique de certains de ces composants, tels que la description WSDL du service, générée à partir de l'interface Java contenant la liste des opérations du service, ou les stubs en Java, générés à partir d'une description WSDL/GWSDL du service.

Plusieurs possibilités sont accessibles au développeur de service pour obtenir cette description du service :
- Utilisation d'une description WSDL existante (par exemple d'un Web service à convertir en Grid service),
- Génération automatique des sections 'binding' et du 'service' du document WSDL à partir de la section 'porttype',
- Génération automatique du document WSDL à partir de l'interface Java du service.

Le langage WSDL a un pouvoir d'expression supérieur à celui d'une interface Java, et il est donc dans certains cas préférable d'écrire le document WSDL ou seulement sa section 'porttype' à la main plutôt que de la générer à partir de l'interface Java du service. Cependant, l'écriture et la maintenance de tout ou partie (section 'porttype') d'un tel document sont assez fastidieuses, d'une part à cause de la redondance des informations, et d'autre part car pour une même description, un plus grand nombre de lignes de code est nécessaire pour une description en WSDL que pour une interface Java. A moins d'avoir une contrainte spécifique à exprimer dans le document WSDL, le développeur de service peut donc préférer s'en passer en utilisant un outil de génération automatique tel que celui qui est fourni dans Apache Axis pour les Web services. Un tel outil, dérivé de celui d'Apache Axis, est proposé pour le développement de Grid services dans GT3, et permet de générer une description du service dans le langage WSDL à partir de l'interface du service écrite dans le langage Java.

Jusqu'à la version alpha 3, cet outil était directement utilisable pour générer la description du service en WSDL, et cette description était ensuite complétée avec des éléments spécifiques aux Grid services, comme par exemple la déclaration des « service data ».

Dans la version alpha 4, GT3 a été entièrement remanié de manière à être compatible avec la dernière version de la spécification OGSI. GT3 alpha 4 apporte des changements notables par rapport aux versions précédentes, tant sur les APIs que sur le processus de build des services utilisant des fonctionnalités optionnelles des Grid services comme la notification par exemple. En effet, un tel service doit être décrit dans un document GWSDL, dans lequel la description de la fonctionnalité optionnelle, également écrite en GWSDL, est importée et héritée. Un ensemble de documents WSDL sont ensuite générés automatiquement à partir de la description GWSDL du service, et l'ensemble de ces documents est déployé avec le service sur un environnement d'exécution compatible OGSI, et utilisé pour la génération automatique des stubs en Java [Figure 2 (partie à droite)].

La notion d'héritage d'une description du 'porttype' d'un service n'existe pas dans la version 1.1 du WSDL, elle est définie par la spécification du GWSDL. Un document WSDL généré à partir de l'interface Java avec les outils actuels ne peut donc pas être directement utilisé pour les services intégrant des fonctionnalités optionnelles de la norme OGSI. Cependant, pour les Grid services implémentant les opérations obligatoires définies dans OGSI et aucune opération optionnelle, il demeure possible d'utiliser la description de service obtenue à partir du générateur de WSDL de Apache Axis, en l'enrichissant simplement de la déclaration des « service data » via des extensions GWSDL [Figure 2 (partie à gauche)].

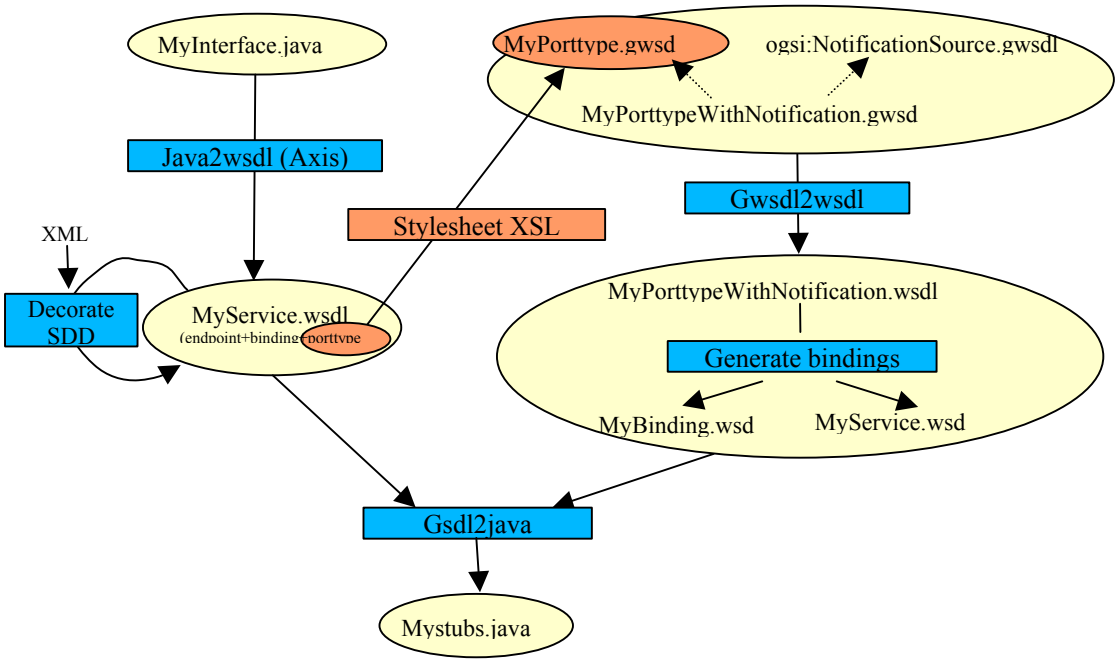

Figure 2 : *Processus de développement*

Dans le cas de Grid services utilisant des fonctionnalités OGSI optionnelles, la solution la moins fastidieuse est d'écrire la déclaration du 'porttype' du service en GWSDL. Il est cependant assez facile de récupérer la section 'porttype' du document WSDL généré à partir de l'interface Java, et de l'adapter afin de construire un document GWSDL. Afin d'automatiser ce processus et de l'intégrer au processus de build, nous avons écrit une stylesheet XSL, qui est interprétée par le processeur XSLT[10] [14] « trax » intégré dans l'outil de build Apache ANT [20] [Figure 2].

Les fonctionnalités de GWSDL vont être intégrées dans la version 1.2 du langage WSDL. Le langage GWSDL va donc disparaître au profit de la version 1.2 du WSDL, et des outils de génération adaptés au WSDL 1 .2 seront donc sans doute proposés aux développeurs dans une prochaine version de GT3.

2.2 Déploiement

Un Grid service ne peut être exécuté que dans un environnement d'exécution compatible OGSI. La dernière version de Globus Toolkit (GT3) contient un tel environnement pour l'exécution de Grid services écrits en Java. Cet environnement

[10] eXtensible Stylesheet Language Transformation : Langage XML permettant de décrire la transformation d'un documents XML en un autre document XML de façon déclarative (partie de XSL)

d'exécution OGSI est disponible séparément du paquet GT3 complet, sous l'appellation 'GT3 core'. Ce sous-ensemble de GT3 suffit pour exécuter la plupart des Grid services, et est écrit entièrement en Java, donc portable sur n'importe quelle plateforme ayant une machine virtuelle Java suffisamment récente (JDK 1.3.1 ou plus récent).

A l'instar des « Web service ARchive » ('.war') utilisées pour le déploiement de Web services, un Grid service est livré sous la forme d'un fichier « Grid service ARchive » ('.gar'). Ce fichier contient les fichiers nécessaires à l'exécution du Grid service dans un environnement d'exécution de service compatible OGSI :

- Bibliothèques du service, bibliothèques des stubs générés et bibliothèques existantes utilisées par le service ('.jar')
- Description du service ('.wsdl' + '.gwsdl')
- Déclaration des services data en XML Schéma ('.xsd')
- Descripteur de déploiement du service ('.wsdd')
- Fichiers de configuration ('.properties', '.xml', …)
- Etc…

Le déploiement d'un Grid service se fait de façon très simple, à l'aide d'une unique commande :

```
ant -Dgar.name=<fichier.gar> deploy
```

Pendant la phase de déploiement, des fichiers sont générés pour permettre la désinstallation du service, à l'aide de la commande :

```
ant -Dgar.id=<identifiant> undeploy
```

2.3 Performance - Utilisation depuis une interface en ligne de commande

Le temps d'exécution nécessaire pour analyser une requête SOAP et sa réponse est tout à fait acceptable sur une machine moderne, à condition bien sûr de l'utiliser à bon escient. En effet, la performance de Apache Axis, sur lequel s'appuie GT3, a été améliorée par rapport à celle de son prédécesseur Apache SOAP, notamment grâce à l'utilisation d'un analyseur basé sur des évènements (SAX) au lieu de l'analyseur à base de manipulation d'arbres (DOM) qui était utilisé dans Apache SOAP. Le temps passé dans l'analyse d'une requête SOAP reste bien sûr plus important que dans le cas d'un message simple dans un format propriétaire, et SOAP ne doit donc pas être utilisé si les requêtes sont fréquentes et si le temps de réponse est un élément particulièrement critique de l'application, mais ce temps est acceptable pour une utilisation normale du service [Tableau 1].

C'est en revanche lors de l'utilisation depuis une interface en ligne de commande (CLI) que GT3 pose réellement des problèmes en terme de temps de réponse. En effet, l'étape pendant laquelle l'application cliente passe la plus grosse partie de son temps d'exécution est l'initialisation, notamment celle des classes chargées de gérer la sécurité [Tableau 1]. Cette étape d'initialisation charge cependant très peu le serveur (environ 3% de plus sur le temps d'exécution global du client sur un Pentium IV à 1,4 GHz si le service s'exécute sur une machine 3 fois plus lente).

	Pentium II – 450 MHz	**Athlon – 900 MHz**	**Pentium IV – 1,4 GHz**
Initialisation du client	10 s	3,2 s	2 s
Résolution du GSR	7,5 s	2,2 s	1,3 s
Aller-retour message	0,07 s	0,01 s	0,02 s
Total	17,5 s	5,4 s	3,3

Tableau 1 : *Temps d'exécution d'une CLI utilisant un Grid service*

Bien que le temps d'exécution global diminue très rapidement avec l'augmentation de la performance de l'ordinateur, il reste relativement élevé par rapport à la tâche effectivement réalisée par l'application. On ne peut pas faire la supposition que l'utilisateur ayant investi dans l'achat d'une machine récente soit prêt à accepter un tel temps d'exécution à chaque utilisation de l'application. On ne peut pas non plus supposer que les utilisateurs vont changer de machine pour pouvoir utiliser les Grid services plus confortablement. Les services ou les applications graphiques ne sont pas concernés par ce problème, puisque une fois l'initialisation effectuée, les nombreux échanges de messages qui s'en suivent prennent un temps que l'on peut considérer comme normal pour le niveau d'encapsulation et de sécurité utilisé. Le temps d'exécution global est par contre inacceptable dans le cas d'une interface en ligne de commande, où généralement un seul couple de messages (aller-retour) est envoyé au service.

Sur le Pentium II cadencé à 450 MHz, nous avons pu réduire le temps d'exécution global du programme d'environ 30% en faisant persister sur le disque dur l'URL de la Grid Service Reference (GSR). De cette façon, le programme ne résout à

nouveau le GSH que lorsque le GSR est périmé, et le comportement ainsi obtenu est alors similaire à celui implémenté dans GT3 lorsque le cache est activé et que plusieurs messages SOAP sont envoyés sans interruption de l'exécution de l'application. Si cette solution ne s'oppose pas au principe de désignation et 'bindings' définis dans la norme OGSI, le gain de temps est cependant largement insuffisant, et les autres objets pour lesquels est passé l'essentiel du temps de l'initialisation ne sont pas « sérialisables », et ne peuvent donc pas être facilement sauvegardés sur le disque dur afin de persister après chaque exécution de l'interface en ligne de commande.

La solution que nous avons adoptée pour réduire le temps d'exécution global de telles applications est donc de réutiliser les objets initialisés pour les différentes exécutions d'une application avec une interface en ligne de commande, en créant les instances de ces objets dans une exécution d'une machine virtuelle dont la durée de vie est supérieure à celle de ces applications, et en permettant une utilisation de ces objets depuis une autre exécution de la machine virtuelle. La solution mise en place ne doit cependant pas nuire à la facilité d'utilisation des services. En effet, grâce aux stubs générés automatiquement par les outils de GT3, l'appel d'une opération d'un Grid service se fait exactement de la même manière que l'invocation d'une méthode d'un objet local, et la couche intermédiaire développée ne doit changer en rien la façon d'utiliser le Grid service pour le développeur.

Notre implémentation de cette solution se compose d'un programme Java que nous appellerons l'accélérateur de CLI, gérant les instances de stubs permettant d'accéder aux instances de services, d'une API Java permettant d'accéder aux services via ce programme de la même façon qu'en utilisant directement les stubs générés par GT3, et d'un générateur de code permettant de générer automatiquement le code dépendant des opérations du service, sans nécessiter d'avoir accès au code source de ce dernier.

Afin de supporter ce mode d'utilisation de Grid service, le développeur d'une interface en ligne de commande doit générer les classes Java nécessaires, à l'aide de l'outil de génération de code fourni par l'implémentation de notre solution. L'accès aux opérations du service via l'accélérateur de CLI se fait ensuite de la même façon que l'accès sans intermédiaire, la seule différence étant la fabrique utilisée pour l'instanciation du stub du service [Figure 3].

```
// Creation du stub myService
PortTypeFactoryAbstract myFactory = new MyServicePortTypeCLIAcceleratorFactory();
MyServicePortType myService = (MyServicePortType) myFactory.createPortType(endpoint);

// Utilisation du service via le stub myService
String result = myService.operation("glop", 35);
```

Figure 3 : *Utilisation du service 'MyService' via l'accélérateur de CLI*

S'il souhaite accélérer l'exécution d'une telle interface en ligne de commande, l'utilisateur doit démarrer le programme gérant les stubs. Pour une utilisation depuis le compte de l'utilisateur, ce démarrage peut par exemple être effectué en même temps que l'initialisation du proxy du certificat x509, qui va permettre à ce denier d'être autorisé à utiliser les ressources de la grille. Dans ce cas d'utilisation, l'exécution du programme de gestion des stubs se termine soit lors de l'expiration de ce proxy (e.g. 12 heures, par défaut, après la création dans le cas de DataGrid), soit à la demande explicite de l'utilisateur.

Les stubs générés par GT3 implémentant l'interface '`java.rmi.Remote`', la solution implémentée s'appuie naturellement sur la technologie RMI[11] [21] afin de permettre une utilisation transparente pour le développeur accédant aux fonctionnalités du service. Chaque instance de stub SOAP créée par l'accélérateur de CLI est enregistrée auprès de la « RMI registry » (externe ou incorporée dans l'accélérateur de CLI), avec comme identifiant le Grid Service Handle (GSH) de l'instance de service associée, et cette instance est réutilisée lorsque la création d'une nouvelle instance de stub SOAP est demandée pour le même GSH. Ainsi, plusieurs applications peuvent utiliser une même instance de service via le même stub SOAP, et une même application peut utiliser plusieurs services ou instances de services simultanément en créant plusieurs couples de stubs (stub RMI pour l'application de l'utilisateur et stub SOAP pour l'accélérateur de CLI). De même, une instance de la classe générique '`org.gridforum.ogsi.bindings.GridServiceSOAPBindingStub`' permettant d'utiliser les opérations standards d'un Grid service (comme par exemple l'accès à un « service data » par son nom), peut-être créée pour chaque instance de service à la demande de l'application.

Les performances obtenues avec cette solution ne sont pas particulièrement bonnes car l'initialisation de RMI occupe une bonne partie du temps d'exécution d'un programme simple, mais le gain obtenu sur le temps d'exécution global permet une

[11] Remote Method Invocation : Mécanisme d'appel de procédures à distance pour Java

utilisation beaucoup plus confortable de Grid services à partir d'interfaces en ligne de commande [Tableau 2]. Bien que le temps nécessaire pour l'aller-retour d'une requête soit forcément légèrement plus long avec un intermédiaire, le gain de temps sur l'exécution globale d'une interface en ligne de commande est forcément important puisque très peu de requêtes sont généralement envoyées par ce type d'application. Par ailleurs, la possibilité d'utiliser les Grid services via des stubs générés automatiquement est conservée, sans que cela n'induise de coût supplémentaire pour le développeur de l'application cliente.

	Pentium II – 450 MHz	**Athlon – 900 MHz**	**Pentium IV – 1,4 GHz**
Initialisation du client	1,3 s	0,6 s	0,28 s
Aller-retour message	?	0,02 s	0,02 s
Total (JVM lancée)	1,3 s	0,6 s	0,3 s

Tableau 2 : *Temps d'exécution d'une CLI avec l'accélérateur de CLI*

La version actuelle de l'accélérateur de CLI utilise comme seul mécanisme de sécurité le « SecurityManager » de Java, associé une politique de sécurité définie dans un fichier de configuration. La politique définie interdit les connections sur le port de la « RMI registry » depuis une machine distante, mais le « SecurityManager » ne permet pas de définir des restrictions en fonction des utilisateurs, ce qui limite donc l'utilisation de cette version aux plate-formes mono-utilisateur. Une amélioration de la sécurité est prévue, basée sur l'envoi du proxy du certificat x509 dans le message RMI et la vérification de ce proxy par l'accélérateur de CLI, afin de palier à cette limitation. Mais la sécurité de l'accélérateur de CLI restera limité à la restriction de son utilisation, et n'aura en aucun cas de fonctionnalité de sécurité avancée comme l'authentification du service (ici l'accélérateur de CLI), l'intégrité ou la confidentialité des messages. Il devra donc nécessairement être exécuté sur la même machine, ou éventuellement sur le même sous-réseau sécurisé que les interfaces en ligne de commande l'utilisant.

Enfin, une adaptation de l'accélérateur de CLI est envisagée afin de supporter une utilisation par des applications clientes écrites dans un langage autre que le Java, tout en conservant autant que possible la facilité d'utilisation de l'implémentation actuelle. La solution technique envisagée pour satisfaire ces deux contraintes est d'ajouter le support du protocole RMI-IIOP [22] dans l'accélérateur de CLI, les API et le générateur de code de la solution. Cette adaptation sera utile, notamment si les interfaces en ligne de commande écrites en langage C passent, comme celles écrites en Java, une partie importante de leur temps d'exécution dans la phase d'initialisation des mécanismes de sécurité. A l'heure actuelle, les tests de performance de l'accès aux opérations d'un Grid service depuis une interface en ligne de commande écrite en langage C n'ont pas encore été réalisés.

3 Grid service de logging des applications des utilisateurs d'une grille

3.1 Besoin

Les ressources de la grille sont le plus souvent utilisées pour exécuter simultanément plusieurs applications métiers. On appelle un « job » une exécution d'une application métier sur la grille. Les jobs peuvent par exemple être synchronisés et s'échanger des messages, être simplement contraints par leur ordre d'exécution, ou encore être indépendants les uns des autres.

Pour un grand nombre d'applications de bio-informatique ou de physique des hautes énergies, une utilisation commune est l'exécution d'un ensemble d'applications métiers indépendantes sur les nœuds géographiquement dispersés de la grille, pour ensuite collecter les résultats de ces exécutions afin de composer le résultat final de l'application principale. Dans ce cas, le nombre de jobs s'exécutant simultanément pour un même utilisateur peut être particulièrement élevé (parfois plusieurs centaines de jobs pour une seule collection). De plus, dans la mesure où leurs exécutions sont indépendantes, ces jobs peuvent être particulièrement dispersés sur les différents nœuds de la grille. Il est dont nécessaire d'adapter les outils classiques de l'informatique, comme par exemple le logging, aux contraintes techniques et fonctionnelles liées à l'environnement de grille, afin de permettre aux utilisateurs d'en exploiter au mieux les ressources.

Des applications des utilisateurs d'une grille de calcul (par opposition à l'infrastructure logicielle de la grille) ont besoin, comme tout autre application, de pouvoir loguer des informations à des fins de débogage, génération d'alertes, etc. Comme

ces applications s'exécutent sur des nœuds géographiquement dispersés, il est nécessaire de centraliser toute cette information afin de permettre aux utilisateurs de l'exploiter plus efficacement.

Depuis la version alpha 4, GT3 intègre un Grid service de logging, qui est utilisé par les Grid services du middleware (notification, etc.) [6]. Cependant, contrairement au logging de tels services, le logging des applications des utilisateurs implique certaines contraintes spécifiques, qui ne sont d'aucun intérêt pour le logging du middleware, mais indispensables pour celui de ces applications.

Parmi ces contraintes spécifiques figure la confidentialité des rapports de log ; ceux-ci doivent être accessibles uniquement par le soumetteur du job, et éventuellement par les autres membres de son Organisation Virtuelle. De même, le système de logging ne doit pas permettre à un utilisateur de polluer le 'repository' de rapports de log d'un autre utilisateur de la grille.

Une autre contrainte spécifique au logging dans ce contexte est la nécessité d'avoir une information plus riche et plus structurée dans chaque rapport de log. En effet, un grand nombre de jobs peut être soumis simultanément par un même utilisateur, et l'exploitation des rapports de log générés par ces nombreux jobs n'est possible qu'avec une structuration et un contenu plus riche que dans les systèmes de logging courants, dont la structure se limite généralement en une séparation du niveau de sévérité et du message du log. Cette richesse de la structure et du contenu des rapports de log permet à l'utilisateur de filtrer plus finement ces rapports afin d'en extraire les informations pertinentes pour lui permettre de résoudre son problème.

Dans le prototype que nous avons implémenté, les rapports de log sont structurés dans un document XML, et les filtres sont décrits à l'aide d'expressions XPath. L'exploitation des rapports de log peut se faire à la demande en accédant à l'historique des rapports de log, ou par envoi de messages par le service en mode « push ». Le mode d'accès à la demande sera plutôt utilisé dans le cas d'une recherche de l'origine d'un problème dans l'exécution de certains jobs, comme par exemple l'obtention d'un résultat incohérent ou un échec de l'exécution de ces jobs, alors que le mode d'accès par notifications sera plutôt utilisé comme moyen de surveillance du déroulement de l'exécution d'un ensemble de jobs.

Dans un cas comme dans l'autre, la définition du filtre peut se faire de manière itérative, en ajoutant ou enlevant des contraintes dans l'expression XPath en fonction des rapports de log obtenus à chaque étape de cette définition. Par exemple, un utilisateur recherchant l'origine d'un problème survenu pendant l'exécution de certains jobs peut affiner le filtre jusqu'à ce qu'il soit en mesure de repérer les messages de log pertinents. A condition que les rapports de log contiennent ces informations, les contraintes de ce filtre peuvent porter sur des critères tels que l'identifiant de l'expérience (i.e. une exécution d'un ensemble de jobs servant à produire un résultat cohérent), l'URL du nœud de calcul de la grille sur lequel s'exécutent les jobs fautifs, la période pendant laquelle le problème est survenu, le nom de l'application ou du module suspecté, le niveau de sévérité minimum des rapports de log souhaités. Le même filtre peut être utilisé pour s'enregistrer auprès du service, afin de recevoir par messages de notification les rapports de log respectant les contraintes exprimées dans l'expression XPath du filtre.

3.2 Développement du service

Le respect de la norme OGSI rend possible la surveillance et la gestion du cycle de vie du service en utilisant les outils standards de OGSI. Ainsi, sans qu'aucun développement spécifique de notre part n'ait été nécessaire pour cela, notre service de logging est capable de donner certaines informations concernant son état interne, ou répondre à certaines opérations comme la mise à jour de la date de fin de son exécution. De même, la possibilité de créer une instance de service volatile par utilisateur permet de résoudre très facilement les problématiques de gestion de session, de gestion des ressources allouées à chaque instance, de persistance de l'état interne des instances afin de permettre la restauration de cet état en cas de redémarrage du service, etc.

L'infrastructure OGSI apporte également une réponse standard et facile à mettre en œuvre pour l'implémentation de certaines fonctionnalités du service de logging, comme par exemple l'envoi de rapports de log sous la forme de notification. Dans ce cas comme dans le cas d'un accès à la demande, le rapport de log est envoyé sous la forme d'un document XML et est automatiquement sérialisé ou dé-sérialisé en Java Bean.

L'existence d'une norme définissant clairement les interfaces permet de remplacer une implémentation par une autre proposant les mêmes fonctionnalités ou les complétant. Par exemple, notre prototype de service de logging utilise les mécanismes de notification de la contribution d'IBM « OGSA Messaging JMS » [7] à la place de celui de GT3. Basé sur

l'implémentation OpenJMS [24] du standard JMS[12] [23], « OGSA Messaging JMS » permet aux clients de Grid services supportant la notification de s'enregistrer pour recevoir des notifications en fonction d'une expression XPath. Les mêmes expressions XPath peuvent alors être utilisées pour l'accès en mode « pull » et en mode « push » aux rapports de log, qui sont temporairement exposés par le service de logging en tant qu'attribut de l'instance service (« service data »).

Par ailleurs, la persistance des rapports de log est faite de façon asynchrone dans une base de données XML (Xindice [25]). Le choix d'une base de données XML pour stocker les rapports de log permet de filtrer ceux-ci avec la même expression XPath lors d'un accès à l'historique des rapports stockés, que pour les accès aux rapports de log en mode « pull » ou en mode « push » via les service data. Par ailleurs la performance de l'accès à l'historique des rapports de log n'est pas un élément critique pour l'utilisation du service.

L'authentification des parties, ainsi que le cryptage et la signature des rapports de log envoyés au service de logging, sont effectués grâce à l'un des mécanismes de sécurité de GT3, GSI-SecureConversation, basé sur les standards WS-Security [15], XML Encryption [16] et XML Signature [17], et sur l'infrastructure de sécurité de Globus : GSI[13] [26]. GSI-SecureConversation assure la sécurité au niveau de la couche application (selon le modèle OSI), en établissant un contexte de sécurité entre chaque client et le service avant de l'utiliser pour le cryptage/décryptage ou la signature/vérification des rapports de log envoyés au service de logging par les jobs.

Ce mécanisme de sécurité permet, lors du déploiement du service, de configurer le type de sécurité (authentification seule, confidentialité ou intégrité) pour l'ensemble ou pour chacune des opérations du service. Le type de sécurité souhaité dépend de la criticité sur le plan stratégique de l'information contenue dans les rapports de log envoyés par les jobs de l'utilisateur. Il est donc utile de proposer plusieurs déploiements différents de ce service avec différentes configurations. Il est également intéressant de proposer plusieurs opérations avec des niveaux de sécurité différents dans un même déploiement du service (e.g. log, logWithConfidentiality, logWithIntegrity, etc.), afin par exemple de ne pas pénaliser le temps de traitement des rapports de log qui n'ont pas besoin d'être confidentiels.

3.3 Déploiement et utilisation du service

Le déploiement du service de logging est peu contraignant. Tous ces composants, de l'environnement d'exécution « GT3 Core » au service lui-même, en passant par les serveurs OpenJMS et Xindice, sont entièrement écrits en Java, et il est donc possible de déployer ce service sur n'importe quelle plate-forme supportant une machine virtuelle Java suffisamment récente. Une fois l'environnement d'exécution OGSI installé, le déploiement du service de logging proprement dit se fait avec l'archive au format GAR, comme décrit dans le chapitre 2.2. Le numéro de port utilisé pour la communication entre le service et les clients est celui du serveur web utilisé (par exemple Apache Tomcat).

Dans son état actuel, le service de logging ne propose qu'un client Java pour l'envoi de rapports de log, limitant ainsi son utilisation aux nœuds de grille offrant aux utilisateurs une machine virtuelle Java. Pour pouvoir utiliser ce service sur le testbed de DataGrid [27], une dizaine d'archives Java ('.jar') doivent être indiquées dans la section « InputSandbox » de la description JDL[14] du job, afin d'être envoyées sur la machine qui va exécuter le job.

L'accélérateur de CLI, exécuté sur la même machine que le job, permet de réduire de façon non négligeable le temps d'exécution global du job si ce dernier prend par exemple la forme d'un script shell qui envoie de nombreux rapports de log au service via l'interface de logging en mode ligne de commande, actuellement écrit en Java.

4 Conclusion

L'infrastructure OGSI est le standard émergent pour le développement de services de grille. Elle constitue l'infrastructure de base pour la définition des services de grille composant l'architecture Open Grid Service Architecture (OGSA) [2], ainsi que, potentiellement, pour tout service de grille développé en réponse à des besoins spécifiques.

La spécification OGSI s'appuie elle-même sur de nombreux standards, au niveau des protocoles afin de favoriser l'interopérabilité, et également au niveau des API utilisées afin de faciliter le portage entre les environnements d'exécution de différents vendeurs.

[12] Java Message Service : API Java pour l'échange asynchrone de messages.
[13] Grid Security Infrastructure: Basé sur PKI, x509, SSL. Permet notamment à l'utilisateur de s'authentifier une seule fois pour l'ensemble des ressources.
[14] Job Description Language : Langage permettant de décrire les caractéristiques et contraintes du job, spécifique au middleware de DataGrid.

Certains des standards sur lesquels s'appuie l'implémentation GT3 subissent actuellement d'importantes évolutions (WSDL, WS-Security…). C'est pourquoi GT3 propose des solutions provisoires, comme par exemple le GWSDL ou le mécanisme de sécurité GSI-SecureConversation, afin de permettre le développement et l'exécution de Grid services. Pour cette même raison, certains outils et API ne peuvent pas encore être utilisés dans toutes les situations. C'est le cas, par exemple, du générateur automatique de description WSDL depuis une interface Java, dont la sortie ne peut pas être directement utilisée pour un service implémentant des fonctionnalités OGSI optionnelles.

Cependant, une fois initié à ses concepts et ses API, GT3 permet un gain de temps considérable pour le développement d'un Grid service, ainsi que pour le développement des outils clients permettant de l'utiliser grâce à la standardisation d'une partie des interfaces et des comportements des services.

La plupart des facilités offertes par cette infrastructure aux services respectant cet ensemble de conventions, ont également un intérêt certain en dehors d'un contexte de grille, et permettent de construire assez facilement toute une classe de Web services avec des fonctionnalités évoluées de notification, sécurité, gestion du cycle de vie, etc.

Le futur de GT3 est étroitement lié aux évolutions des Web services et autres standards de grille. Ainsi, le GWSDL, défini par le groupe de travail OGSI pour permettre l'expression de la spécification, va disparaître au profit du WSDL 1.2 qui intègrera les possibilités de composition de services exploitées par la spécification OGSI. Le mécanisme de sécurité de la couche transport, basé sur le protocole HTTPG, va également disparaître, tandis que les mécanismes de sécurité de la couche application vont continuer leur évolution en suivant celle des standards de sécurité pour XML et les Web services.

Dans cet article, nous avons présenté un retour d'expérience sur le développement de Grid services, en l'illustrant par quelques unes des difficultés rencontrées et les solutions que nous avons dû mettre en place pour les contourner. Malgré le temps nécessaire pour être opérationnel sur l'utilisation de cette technologie, lié à son immaturité, et malgré les changements déjà prévus dans les prochaines versions, nous sommes convaincus du rôle fondamental de OGSI dans la construction des infrastructures informatiques autour des grilles.

Références

[1] I. Foster, C. Kesselman et S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Intl J. Supercomputer Applications*, 2001.
[2] I. Foster, C. Kesselman, J. M. Nick et S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Globus Project*, 2002.
[3] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling et P. Vanderbilt, Open Grid Services Infrastructure (OGSI) Version 1.0. *Global Grid Forum*, 27 Juin 2003.
[4] S. Tuecke, K. Czajkowski, S. Graham, M. Nally, F. Leymann, E. Stokes, T. Storey et S. Weerawarana, Describing ServiceData in Web Services, 2002.
[5] T. Sandholm et J. Gawor, Globus Toolkit 3 Core – A Grid Service Container Framework, 2003.
[6] M. Williams et J. Wiley, OGSA Hosting Environment Logging Service, 2003.
[7] JP. Antony, JMS Notification Framework, 2003.

Spécifications

[8] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, 2001.
[9] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/, 2000.
[10] Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml, 2000
[11] XML Schema 1.0, http://www.w3.org/XML/Schema, 2001.
[12] XML Path Language (XPath) 1.0, http://www.w3.org/TR/xpath, 1999.
[13] XML Query Language (XQuery) 1.0, http://www.w3.org/TR/xquery/, 2003.
[14] XSL Transformations (XSLT) 1.0, http://www.w3.org/TR/xslt, 1999.
[15] Web Services Security (WS-Security), http://www.oasis-open.org/committees/wss/, 2002.
[16] XML Encryption, http://www.w3.org/TR/xmlenc-core/, 2002.
[17] XML Signature, http://www.w3.org/TR/xmldsig-core/, 2002.

Produits et API

[18] Globus Toolkit 3 (GT3), http://www-unix.globus.org/toolkit/download.html
[19] Apache Axis, http://ws.apache.org/axis/
[20] Apache Ant, http://ant.apache.org/
[21] Remote Method Invocation (RMI), http://java.sun.com/products/jdk/rmi/
[22] RMI over Internet Inter-ORB Protocol (RMI-IIOP), http://java.sun.com/products/rmi-iiop/
[23] Java Message Service (JMS), http://java.sun.com/products/jms/
[24] OpenJMS, http://openjms.sourceforge.net/
[25] Xindice, http://xml.apache.org/xindice/
[26] Grid Security Infrastructure (GSI), http://www.globus.org/security/
[27] European DataGrid (EDG), http://www.eu-datagrid.org/