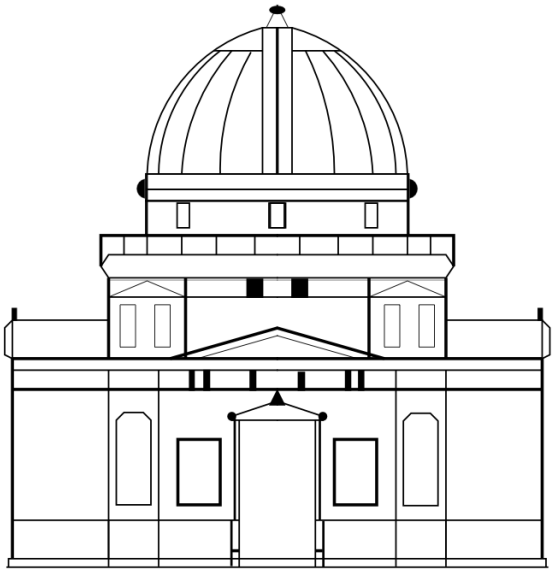
Rapport de stage

IUP Génie Mathématique et Informatique



Observatoire astronomique de Strasbourg

Outil de Supervision de WebServices

3 septembre 2004

Stagiaire : Jean-François NICOLAS

Responsable : André SCHAAFF

Table des matières

Table des figures

Remerciements

Avant toute chose, je tiens à remercier mon responsable au sein de l'Observatoire Astronomique de Strasbourg, André SCHAAFF, ainsi que toute l'équipe du Centre de Données astronomiques de Strasbourg (CDS). Je les remercie pour m'avoir offert un stage pour la deuxième fois, ainsi que pour l'aide et le suivi qu'ils m'ont accordé.

Introduction

Ce rapport concerne mon stage de deuxiéme année d'IUP Génie Mathématique et Informatique à Strasbourg. Celui-ci s'étendait sur la période du 14 juin au 31 août 2004. Il a été accompli à l'Observatoire Astronomique de Strasbourg et plus particulièrement au Centre de Données Astronomiques de Strasbourg (CDS).

Ce rapport est construit de la maniére suivante : pour commencer une présentation du centre de recherche, par la suite je présenterai le sujet du projet et enfin je décrirai plus précisément la solution développée.

Chapitre 1

Présentation du laboratoire

1.1 Structure juridique

L'Observatoire de Strasbourg est une Unité de Formation et de Recherche (UFR) de l'Université Louis Pasteur. Il est également une Unité Mixte de Recherche du CNRS et de l'Université Louis Pasteur (UMR 7550). [1]

1.2 Présentation générale

L'Observatoire est situé sur le campus de l'Esplanade ; ses bâtiments font partie du campus historique de l'université de Strasbourg.

Enseignements dispensés :

- DEA "Analyse et Traitement des Données sur les Milieux Astronomiques"
- DEUG Sciences et autres DEUG (enseignements d'ouverture).
- Licence de Géosciences.
- Maîtrise de Géosciences, Maîtrise de Physique, Maîtrise de Sciences Naturelles.
- Préparation au CAPES et à l'Agrégation.
- DESS Applications des Technologies Spatiales.
- Diffusion de la Culture

La partie publique de l'Observatoire, le Planétarium, est ouvert pour la vulgarisation de l'Astronomie.

L'Observatoire se compose de quatre équipes de recherche :

1.2.1 Hautes Énergies

L'équipe Astrophysique des Hautes énergies [2] a pour thème l'étude des astres et sites de l'univers émetteurs de photons de haute énergie. Cette thématique générale recouvre des aspects variés, comme l'étude des astres compacts en fin d'évolution, la physique de leur activité, les phénoménes de haute énergie intéressant les étoiles jeunes ou le soleil, ou l'étude de ces phénoménes à l'échelle galactique.

Ses recherches se sont largement appuyées sur les données acquises par le satellite ROSAT et s'appuieront dans l'avenir sur celles des satellites X de nouvelle génération, tout spécialement XMM.

1.2.2 Etoiles et systemes stellaires

Les recherches menées par l'équipe « Etoiles et systémes stellaires » [3] recouvrent un domaine étendu, incluant les étoiles, les milieux interstellaires, la Galaxie et les galaxies proches.

De l'étoile, objet individuel, l'intérêt s'est porté aux groupes d'étoiles, témoin de l'évolution stellaire mais aussi traceur des grandes structures de la Voie Lactée.

1.2.3 Galaxies

Les activités de l'équipe [4] sont centrées sur les problémes de la structure du Groupe Local, de ses populations stellaires et sur la dynamique gravitationnelle. De plus, l'équipe possède un savoir-faire sur les outils statistiques d'analyse de données et sur les méthodes inverses non paramétriques.

Un des objectifs consiste à combiner les informations d'évolution des populations stellaires et celles de dynamique afin de reconstituer les événements déterminants liés aux processus de formation et d'évolution galactique.

1.2.4 Le centre de données (CDS)

L'activité de recherche du CDS [5] s'est concentrée sur l'étude de la dynamique galactique et des populations d'étoiles binaires, sur une participation importante à la mission HIPPARCOS de l'Agence Spatiale Européenne, ainsi que sur le développement de méthodologies nouvelles applicables à l'analyse et au traitement de données astronomiques.

1.3 Organisation du personnel

- Directeur : Jean-Marie HAMEURY
- Responsable administratif : Sandrine LANGENBACHER
- 23 enseignants et chercheurs
- 20 personnels ingénieurs, techniciens et administratifs

1.4 Centre de données astronomiques (CDS)

J'ai effectué mon stage au sein du Centre de Données astronomiques de Strasbourg (CDS) qui est un centre de données dédié à la collection et à la distribution dans le monde entier de données astronomiques.

Le CDS héberge la base de données SIMBAD, la base de référence mondiale pour l'identification d'objets astronomiques. Le but du CDS est de :

- rassembler toutes les informations utiles, concernant les objets astronomiques, disponibles sous forme informatisée : données d'observations produites par les observatoires du monde entier, au sol ou dans l'espace ;
- mettre en valeur ces données par des évaluations et des comparaisons critiques ;
- distribuer les résultats dans la communauté astronomique ;
- conduire des recherches utilisant ces données.

Le CDS joue, ou a joué, un rôle dans d'importantes missions astronomiques spatiales : contribuant aux catalogues d'étoiles guides, aidant à identifier les sources observées ou organisant l'accés aux archives, etc. Le CDS contribue au XMM Survey Science Center, sous la responsabilité de l'équipe "Hautes-Energies" de l'Observatoire de Strasbourg.

Le CDS a signé des accords d'échanges internationaux avec les organismes suivants :
- NASA,
- National Astronomical Observatory (Tokyo, Japon),
- l'Académie des Sciences de Russie,
- le réseau PPARC Starlink au Royaume-Uni,
- l'Observatoire de Beijing (Chine),
- l'Université de Porto Allegre au Brésil,
- l'Université de La Plata en Argentine,
- InterUniversity Center for Astronomy and Astrophysics (Inde).

Le CDS est membre de la Fédération des Services d'Analyse de Données Astrophysiques et Géophysiques.

Le CDS coopère aussi avec l'Agence spatiale Européenne (transfert au CDS du service

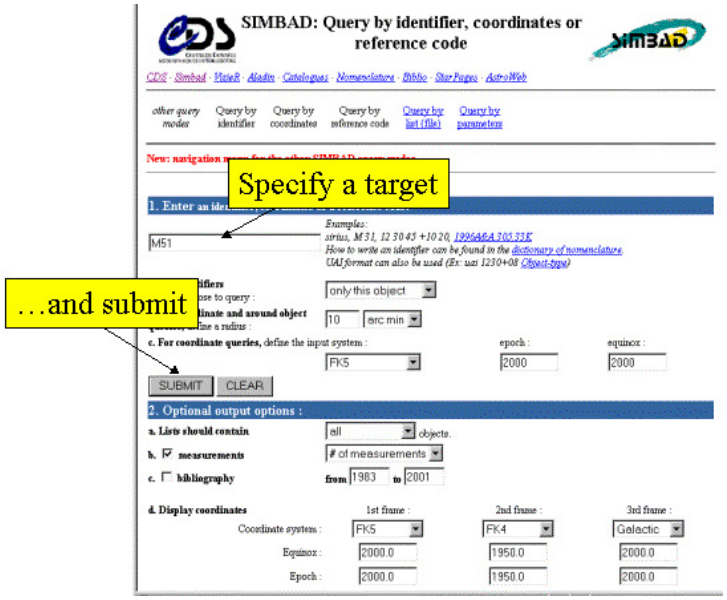


FIG. 1.1 – Page Web permettant d'effectuer une requête sur Simbad

de catalogues du projet ESIS : le projet VizieR), et avec la NASA : en particulier le CDS abrite une copie miroir du Systéme de Données Astrophysiques (ADS) et ADS abrite une copie miroir de SIMBAD. Le CDS contribue aussi au projet NASA AstroBrowse. Le CDS abrite aussi les copies miroirs Européennes des journaux de l'American Astronomical Society (AAS).

1.5 Les Services du CDS

1.5.1 Simbad

Simbad est une base de données de référence pour les identifications et la bibliographie d'objets astronomiques. [6]

Simbad contient plus 7,5 millions d'identificateurs pour plus de 2,8 millions d'objets différents. Pour chaque objet figurent dans la base quelques mesures (position, magnitude dans différents domaines de longueurs d'ondes), ainsi que les références bibliographiques où l'objet est cité (plus de 110 000 articles sont concernés).

L'utilisateur peut choisir le format du fichier où seront entreposés les résultats de la requête(Fig. 1.1). En effet, Simbad peut générer des fichiers HTML, XML ou xls (fichiers Excel).

Cet ensemble de données résulte d'un long travail d'identification croisée entre de

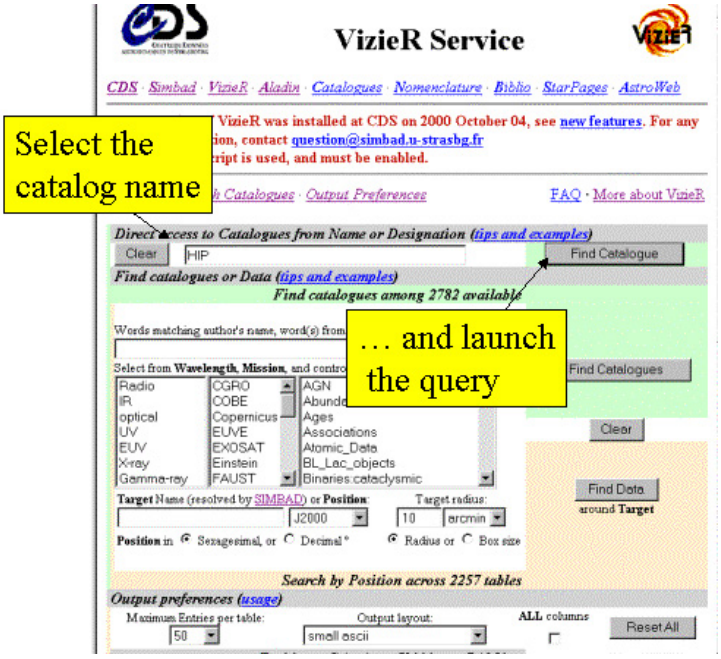


FIG. 1.2 – Page Web permettant d'effectuer une requête sur VizieR

nombreux catalogues, listes d'objets et articles de journaux, entrepris au début des années 1980, et constamment développé et mis à jour depuis.

1.5.2 VizieR

VizieR est une base de données rassemblant plusieurs milliers de catalogues astronomiques sous un format homogène. [7] Une description standardisée du contenu des catalogues permet leur inclusion dans un systéme de gestion de base de données (SGBD) relationnel. Un ensemble de liens, entre les tables de VizieR, et avec des services externes (bibliographiques, archives externes, serveurs d'images), permettent de naviguer entre les données des catalogues et d'autres données associées (Fig. 1.2). Il faut noter que les très grands catalogues (plus de 10^7 enregistrements) ne peuvent pas être gérés par un SGBD relationnel pour des raisons de performances. Des outils spécifiques doivent être utilisés.

1.5.3 Aladin

Aladin est un atlas interactif du ciel permettant d'accéder simultanément à des images numérisées du ciel, ainsi qu'à des catalogues et bases de données astronomiques. [8]

Cet outil permet de superposer, sur des images du ciel optique, les objets présents dans Simbad, des sources de catalogues contenus dans VizieR, mais aussi d'autres données,

1.5 Les Services du CDS

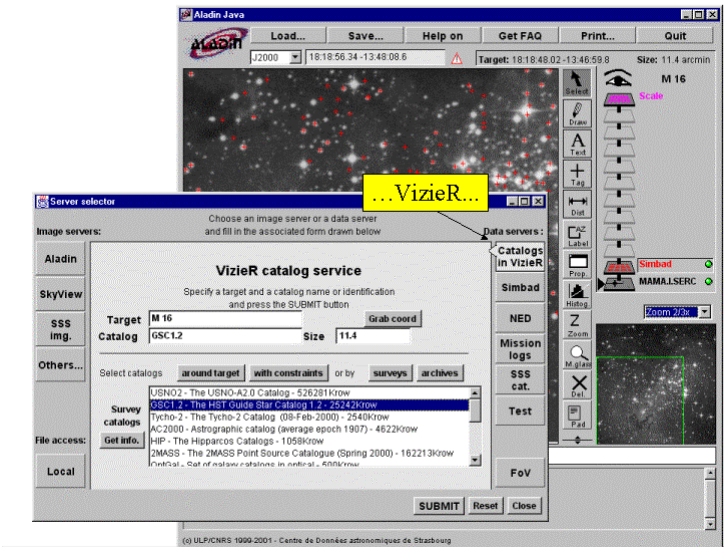


FIG. 1.3 – Exemple d'utilisation d'Aladin

locales ou situées sur des serveurs distants (archives, HST, ...).

Chapitre 2

Le projet

2.1 Le contexte

Le CDS développe des services de référence à forte valeur ajoutée, très utilisés par les astronomes du monde entier. Il joue un rôle moteur dans la mise en place de liens entre les services d'information en astronomie et dans le développement de standards, en partenariat avec les autres producteurs de services. Ces divers projets nationaux et transnationaux se sont regroupés en un consortium, l'International Virtual Observatory Alliance (IVOA)[9].

2.2 Le sujet

L'un des groupes de travail de l'IVOA se focalise sur les aspects grilles et les Web-Services. Le CDS a donc développé depuis octobre 2002, des Services Web XML en complément des services déjà existant. Ce qui était expérimental au départ commence à être utilisé réguliérement.

Cependant, aucun journal ne permettait d'évaluer et d'analyser précisément le trafic occasionné par les WebServices.

Le projet se décompose en deux parties distinctes :

1. la génération des fichiers journaux : cette partie ayant été réalisée en projet tutoré avec Julian MAGGI.
2. l'analyse et la visualisation des journaux.

2.3 Présentation de l'existant

2.3.1 Les WebServices SOAP

SOAP (Simple Object Access Protocol) offre un moyen simple et léger d'échanger des informations structurées et typées entre pairs dans un environnement décentralisé et distribué utilisant XML. Cela permet d'utiliser SOAP dans un grand nombre de systèmes allant des systèmes de messagerie aux appels de procédure à distance (RPC).

SOAP permet aux applications de communiquer entre elles d'une manière unique grâce à l'infrastructure du Web. Aujourd'hui, l'énorme majorité du trafic Web est constituée par l'envoi de pages HTML depuis des serveurs vers des navigateurs. Avec SOAP, le Web peut devenir bien plus que cela : SOAP permet aux applications de communiquer entre elles et fourni l'infrastructure pour connecter des sites Web à des applications pour créer des services Web.

Les services Web relient des sites aux applications pour apporter des fonctionnalités que des composants individuels ne pourraient pas fournir. Par exemple, imaginez une application qui relie entre elles les informations que vous recevez d'un site contenant des données sur le trafic routier grâce à une destination qui lui aurait été fournie au préalable par votre gestionnaire de calendrier. Cela pourrait donner une application qui vous rappelle de partir 15 minutes plus tôt à votre rendez-vous, et vous indique et recalcule le trajet idéal en tenant compte du trafic routier, des accidents, etc.

SOAP apporte la manière pour chacun de ces services de présenter les fonctionnalités et communiquer avec d'autres services. Grâce à SOAP, il est possible de lier les services entre-eux comme des composants et de construire ce genre d'application très rapidement. SOAP permet à des composants homogènes ou hétérogènes de communiquer par Internet d'une manière simple. Il permet une communication de service à service, de composant à service ou de composant à composant.

Même si SOAP ressemble beaucoup en termes de fonctionnalités à des protocoles comme IIOP pour CORBA, ORPC pour DCOM, ou JRMP (Java Remote Method Protocol) pour JAVA, il possède un avantage indéniable. En effet, SOAP utilise un mode texte alors que les autres fonctionnent en mode binaire, cela facilite le passage des équipements de sécurité et le traitement en cas de filtrage.

XML permet de structurer les requêtes et les réponses, indiquer les paramètres des méthodes, les valeurs de retours, et les éventuelles erreurs de traitements. SOAP n'est tenu par aucun système d'exploitation, langage de programmation ou modèle objet. Par exemple, il permet aux applications CORBA de communiquer avec des applications DCOM tournant sur n'importe quelle plate-forme. Un autre avantage est que SOAP est

relativement simple à implémenter. En partant de la spécification SOAP publiée, il est possible d'implémenter une application SOAP qui sera en mesure de tourner en quelques jours seulement.

XML joue un rôle prépondérant dans SOAP, on peut distinguer plusieurs éléments :
- une enveloppe, expliquant comment la requête doit être traitée et présentant les éléments contenus dans le message.
- un ensemble de règles de codage, permettant de différencier les types de données transmises.
- une convention permettant de représenter les appels aux procédures de traitement et les réponses.

Pour expliquer le fonctionnement des webservices, il convient de distinguer plusieurs étapes :

1. Recherche dans un annuaire UDDI : le client cherche un service particulier, il s'adresse à un annuaire qui va lui fournir la liste des prestataires habilités à satisfaire sa demande. L'annuaire UDDI peut être comparé à un moteur de recherche sauf que les documents sont remplacés par des services.
2. Recherche de l'interface du composant à contacter : une fois la réponse reçue (en XML) de l'annuaire, le client va chercher à communiquer via une interface. Cette interface décrite en langage WSDL fournit l'ensemble des services disponibles. Elle offre la possibilité de vérifier que le contrat correspond bien aux besoins demandés.
3. Invocation du service : le client doit maintenant passer les paramètres attendus par le service et assurer la communication avec le serveur. L'outil utilisé pour cela est le Proxy, c'est l'objet qui permet la communication entre le client et le serveur. Il est généré par le client en utilisant l'interface WSDL.

WSDL (Web Service Definition Langage) est un format de représentation des interfaces de service Web en XML. Une analogie avec CORBA peut être faite, en effet WSDL est la représentation XML du langage IDL (description d'interfaces). WSDL a deux rôles prépondérants :
- Il sert de référence à la génération de Proxies
- Il assure le couplage entre le client et le serveur par le biais des interfaces.

2.3.2 Apache Axis

Les WebServices déployés au CDS, reposent sur une architecture Tomcat/Axis. Ceci constitue une solution gratuite, libre (licence Apache) et portable. Tomcat est un conteneur Web J2EE, hébergeant des applications Web composées de servlets et/ou JSP. Axis est une application Web déployée dans Tomcat qui implémente la spécification SOAP.

2.3 Présentation de l'existant

Apache Axis est un servlet développé et maintenu par la fondation Apache. Ce servlet fournit :

- un environnement pouvant soit fonctionner comme un serveur SOAP indépendant soit comme un plug-in de moteurs de servlet (en particulier TOMCAT),
- une API pour développer des services web SOAP RPC ou à base de messages SOAP,
- le support de différentes couches de transport : HTTP, FTP...
- la sérialisation/désérialisation automatique d'objets Java dans des messages SOAP,
- des outils pour créer automatiquement les WSDL correspondant à des classes Java ou inversement pour créer les classes Java sur la base d'un WSDL (classe proxy en quelque sorte, qui fait le lien entre l'application Java cliente et le service distant).

Chapitre 3

Création des fichiers journaux

Cette partie du projet à été réalisée dans le cadre du projet 50 heures de l'IUP2 en collaboration avec Julian MAGGI.

3.1 Récupération des informations

3.1.1 Informations sur les clients

Pour récupérer des informations sur les clients consommateurs de WebServices, il faut pouvoir intercepter chaque requête. Pour ce faire plusieurs solutions sont envisageables :

- insérer dans le code de chaque WebService un appel à une fonction en début et en fin de traitement. Cette fonction prend en paramètre l'objet `MessageContext` qui contient la plupart des informations que nous souhaitons obtenir.
- utiliser la fonction de `Handler` prévue dans Axis. Cette fonctionnalité repose sur l'utilitaire de gestion de journaux `log4j` [11] qui à l'instar de Tomcat appartient au projet Jakarta de la fondation apache.

La solution adoptée est la deuxième, car avec celle-ci il n'est pas nécessaire de modifier les WebServices déployés.

En effet à chaque requête, Axis créé un `Handler` qui est en fait un processus parallèle et asynchrone. Axis offre la possibilité de spécifier quelle classe Java sera le `Handler`, nous avons donc pu développer notre propre `LogHandler`. Ce dernier permet la récupération des informations suivantes pour chaque requête :

- la date et l'heure
- le nom du service invoqué
- le nombre de requête enregistrés pour ce service (ce paramètre est remis à 0 lors de chaque redémarrage du moteur de servlets Tomcat)

– le nom de l'opération appelée
– le temps d'exécution de la requête

D'autres informations pourront facilement être rajoutées par la suite.

La difficulté majeure est de calculer le temps d'exécution d'une requête. En effet nous avons vu précédemment que le `LogHandler` est un processus parallèle et asynchrone par rapport à l'invocation du WebService elle-même. Il est donc impossible de savoir quand la requête s'est achevée. La solution est donc de demander à Axis d'invoquer notre `LogHandler` sur le flux de requête et sur le flux de réponse. Ainsi pour le flux de requête, il suffit de lancer un chronomètre qui sera stoppé lors de l'appel du `LogHandler` pour le flux de réponse.

Pour déterminer s'il s'agit d'une requête ou d'une réponse, nous utilisons la méthode `msgContext.getPastPivot()` qui renvoie VRAI s'il s'agit d'un flux réponse. Il est possible de différencier deux requêtes concurrentes car il y a une seule instance du `LogHandler` pour un couple requête/réponse.

3.1.2 Informations sur les WebServices déployés

D'un autre coté il est intéressant de pouvoir connaître à tout moment la liste des WebServices déployés sur un serveur. Nous avons donc choisi de créer un WebService permettant de renvoyer la liste des services déployés, un peu comme le fait le servlet d'Axis accessible à l'adresse : http://localhost:8080/axis/servlet/AxisServlet/. Ce service, baptisé `WSRegistry`, renvoie un fichier XML contenant la liste de services déployés avec pour chacun, la liste des méthodes ainsi que leurs paramètres :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<AxisEngine>
        <WebService name="AladinImage">
                <EndPoint>null</EndPoint>
                <Operation name="getObservingProgram">
                        <InParams>
                                <Param type="string">position</Param
                                    >
                                <Param type="string">radius</Param>
                                <Param type="string">program_name</
                                    Param>
                        </InParams>
                        <OutParams>
                                <Param type="ObservingProgram">
```

```
                                        ObservingProgram</Param>
                        </OutParams>
                </Operation>
15              <Operation name="getFilters">
                        <InParams>
                                <Param type="string">position</Param
                                    >
                                <Param type="string">radius</Param>
                                <Param type="string">program_name</
                                    Param>
20                      </InParams>
                        <OutParams>
                                <Param type="ArrayOf_tns2_Filter">
                                    ArrayOf_tns2_Filter</Param>
                        </OutParams>
                </Operation>
25              <Operation name="getImages">
                        <InParams>
                                <Param type="
                                    ArrayOf_tns2_StoredImage">in0</
                                    Param>
                                <Param type="float">in1</Param>
                                <Param type="float">in2</Param>
30                              <Param type="string">in3</Param>
                                <Param type="string">in4</Param>
                        </InParams>
                        <OutParams>
                                <Param type="Array"/>
35                      </OutParams>
                </Operation>
        </WebService>
</AxisEngine>
```

Les balises `<InParam/>` représentent les types de paramètre d'une fonction alors que les balises `<OutParam/>` représentent les types des valeurs de retour .

Ces informations ne seront pas incorporées dans les fichiers journaux, il s'agit simplement d'une fonctionnalité supplémentaire permettant de connaître les services déployés à un instant donné.

Le choix de développer cet outil s'est fait au vue de la complexité des annuaires UDDI. En effet, notre souhait était de pouvoir disposer simplement et rapidement d'une liste des services déployés. La majeure partie des fonctionnalités d'UDDI ne sont pas utiles dans ce cas (pages blanches, pages jaunes), de plus UDDI fournit une description technique des services, redondante avec WSDL.

La difficulté majeure est qu'il n'est pas possible pour un WebService d'interroger directement le moteur Axis à travers l'objet `AxisEngine`. Pour palier ce problème il faut créer une nouvelle copie du moteur Axis à l'aide du fichier `server_config.wsdd`.

3.2 Création du journal d'activités XML

3.2.1 Pourquoi XML ?

Lors du développement de la solution, il est vite apparu qu'il faudra traiter un nombre important d'informations.

Pour chaque connexion vers un webservice, nous conservons actuellement 5 informations différentes (date, nom, méthode,...) sur le client et le webservice. Il a donc fallu définir un format pour stocker l'ensemble de ces informations. Afin d'assurer une meilleure portabilité et une meilleure organisation des informations, nous avons décidé de nous orienter vers un enregistrement des données sous un format XML. Avec ce format, les données deviennent complètement indépendantes du langage utilisé pour leur interprétation. L'utilisateur peut alors visionner ces informations avec une application dédiée ou un simple éditeur de texte.

3.2.2 Le schéma XML retenu.

Notons que ce schéma est le schéma retenu pour l'enregistrement des informations lors d'une connexion vers un webservice.

```
<xsd:complexType name='WSRequestType'>
  <xsd:sequence>
     <xsd:element name='WSName' type='xsd:string'/>
     <xsd:element name='WSRequestCounter' type='xsd:int'/>
     <xsd:element name='WSOperationName' type='xsd:string'/>
     <xsd:element name='WSClientAddress' type='xsd:string'/>
  </xsd:sequence>
  <xsd:attribute name='date' type='xsd:dateTime'/>
</xsd:complexType>
```

```
10
<xsd:element name='List'>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name='WSRequest' type='WSRequestType' minOccurs='
          0' maxOccurs='unbounded'/>
15    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

A l'instar de la plupart des serveurs Web, un fichier journal est créé chaque jour.

3.2.3 XML et Java.

Afin de pouvoir travailler avec des données XML en Java, il a fallu rechercher une librairie permettant de parser des données XML et de créer une structure XML. La librairie que nous avons retenue est JAXB (Java Architecture for XML Binding).

3.2.4 Qu'est-ce que JAXB ?

JAXB [10] définit l'architecture et les API permettant de sérialiser et désérialiser des objets Java en fragments XML. Suivant en cela des produits existants tels que Castor ou Zeus, JAXB se repose sur des schémas W3C XML Schema pour définir la structure des documents XML.

3.2.5 Optimisation par rapport XML

La structure même d'un document XML alourdie les traitements car les données doivent être encapsulées dans un élément racine. Ainsi à chaque requête, le `LogHandler` doit :

1. charger en mémoire tout le contenu du fichier XML
2. ajouter un nouveau noeud pour la requête
3. écrire le fichier sur le disque

Un tel traitement est trop lourd pour être exécuté à chaque requête. C'est pourquoi nous avons fait le choix d'archiver sur le disque des fichiers XML non valides. En effet toutes les requêtes sont stockées à la suite sans élément racine. Ainsi il suffit au `LogHandler` d'ouvrir le fichier en ajout et d'écrire la nouvelle requête.

3.2 Création du journal d'activités XML

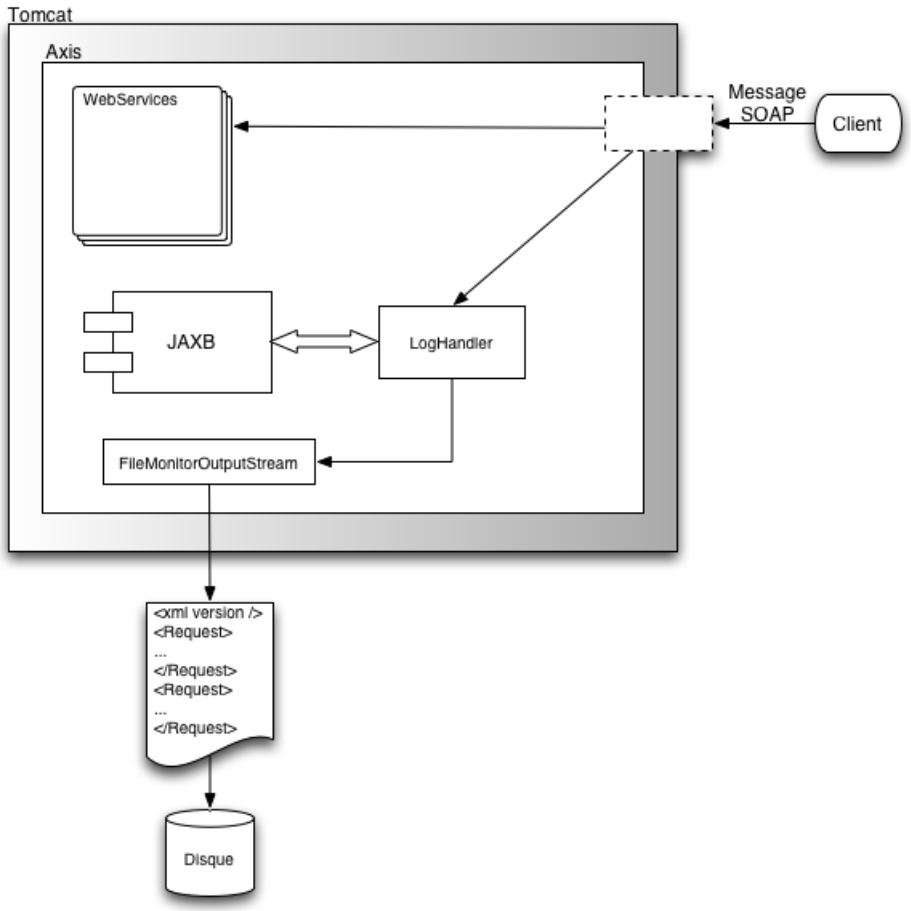


Fig. 3.1 – Fonctionnement de FileMonitorOutputStream

Pour permettre cela, nous avons surchargé les classes `java.io.FileInputStream` et `java.io.FileOutputStream`.

FileMonitorOuputStream

A chaque fois que JAXB sérialise un requête, il ajoute une entête XML. Nous avons donc un fichier journal avec autant d'entêtes que de requêtes. Ce fichier n'est pas exploitable.

Le rôle de la classe `FileMonitorOuputStream` est d'assurer qu'il n'y ai qu'une seule entête XML par fichier (voir schéma 3.1).

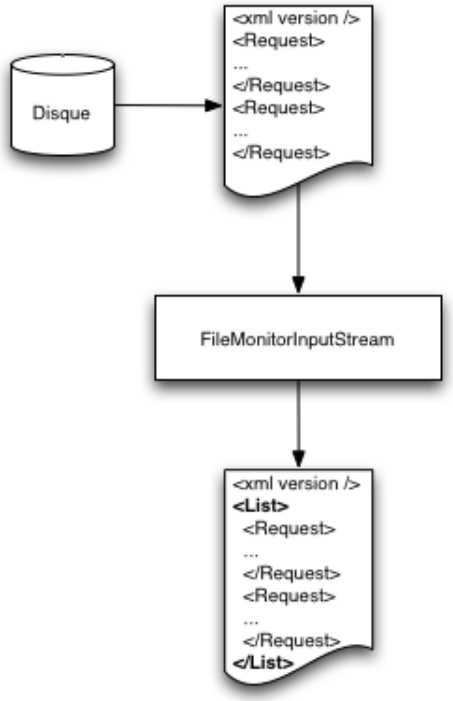


FIG. 3.2 – Fonctionnement de FileMonitorInputStream

FileMonitorOuputStream

Cette classe permet de rendre valide le fichier journal en encapsulant les données dans un élément racine : `<List></List>` (voir schéma 3.2)

3.3 Utilisation du LogHandler

Pour associer le `LogHandler` à un WebService il faut modifier son fichier de déploiement. Voici un exemple de fichier de WSDD pour le service `LogTestService` :

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/
               java">

 <handler name="track" type="java:wsmonitor.LogHandler">
5  <parameter name="filename" value="/var/log/axis"/>
 </handler>

 <service name="LogTestService" provider="java:RPC">
```

```
    <requestFlow>
10   <handler type="track"/>
    </requestFlow>
     <responseFlow>
     <handler type="track"/>
    </responseFlow>
15
    <parameter name="className" value="wsmonitor.TestService"/>
    <parameter name="allowedMethods" value="*"/>
   </service>

20 </deployment>
```

La ligne 4 permet de préciser quelle classe Java joue le rôle de `Handler`. A la ligne 5 on précise le chemin vers le répertoire contenant les fichiers journaux. Ensuite de la ligne 9 à 14 on précise que cet `Handler` est appelé pour les flux de requête et de réponse.

Chapitre 4

Interprétation et visualisation

4.1 Définitions des entités entrant en jeu

Il y trois entités différentes qui entrent en jeu dans cette solution. Ce découpage a été pensé dans le but de répartir les services sur des machines différentes. Ces machines peuvent se trouver au sein du même réseau local ou accessibles depuis l'Internet.

Néanmoins rien n'empêche de regrouper toutes ces entités sur une même machine.

4.1.1 Serveur WebServices

Le couple Tomcat/Axis est installé, c'est donc cette machine qui offre tous les Web-Services à la communauté. C'est aussi cette entité qui stocke les fichiers journaux sur ses disques.

Un servlet, `WSMonitorServer`, est également déployé en tant qu'application web sur le même serveur Tomcat. Ce servlet sera interrogé grâce au protocole HTTP (méthode GET ou POST) pour récupérer les fichier journaux. Le rôle de celui-ci n'est pas simplement de rendre accessible les fichiers journaux.

En effet, nous avons vu précédemment (voir 3.2.2) que pour chaque jour, nous disposons d'un fichier journal différent. Le servlet prend en paramètre deux dates au format `dd/MM/yyyy HH:mm`. Il va donc régénérer un fichier à partir de plusieurs fichiers journaux en fonction des dates passées en paramètre.

4.1.2 Serveur WSMonitor

Cette entité se charge de télécharger le fichier journal généré par le servlet `WSMonitorServer` déployé sur chaque serveur de WebServices. Ainsi, il est possible de gérer un ou plusieurs serveurs de WebServices et comparer les résultats.

Cette entité se charge également de l'interprétation des fichiers journaux et de la mise en forme des résultats.

4.1.3 Client

Le client peut consulter et analyser les fichiers journaux de différents serveurs à l'aide de son navigateur web. Il lui faudra simplement installer un plugin pour visualiser les images générées par le serveur WSMonitor.

Toutes les pages HTML générées sont valides `XHTML 1.1`

4.1.4 Schéma récapitulatif

Voir Figure 4.1

4.2 Outils utilisés

4.2.1 Technologies J2EE

J2EE[12] (Java 2 Entreprise Edition) est la version entreprise de la plate-forme "Java 2" qui se compose de l'environnement "J2SE" ainsi que de nombreuses API et composants destinés à une utilisation "côté serveur" au sein du système d'information de l'entreprise.

Dans J2EE, nous avons utilisé plus particulièrement les technologies Servlet et JSP (Java Server Pages), qui sont des composants réseau destinés à une exécution orientée question / réponse. La technologie JSP est une extension de la notion de Servlet permettant de simplifier la génération de pages web dynamiques. JSP est un concurrent direct de l'ASP et du PHP. Un servlet est un composant coté serveur, écrit en Java, dont le rôle est de fournir une trame générale pour l'implémentation de paradigmes "requête / réponse". Ils remplacent les scripts CGI.

Ainsi, tous les développement ont été effectués en Java. Ce choix a été suggéré par le fait que ce langage est déjà utilisé pour le développement des WebServices, dans un soucis de portabilité et de simplicité par rapport à d'autres langages orientés objet comme C++.

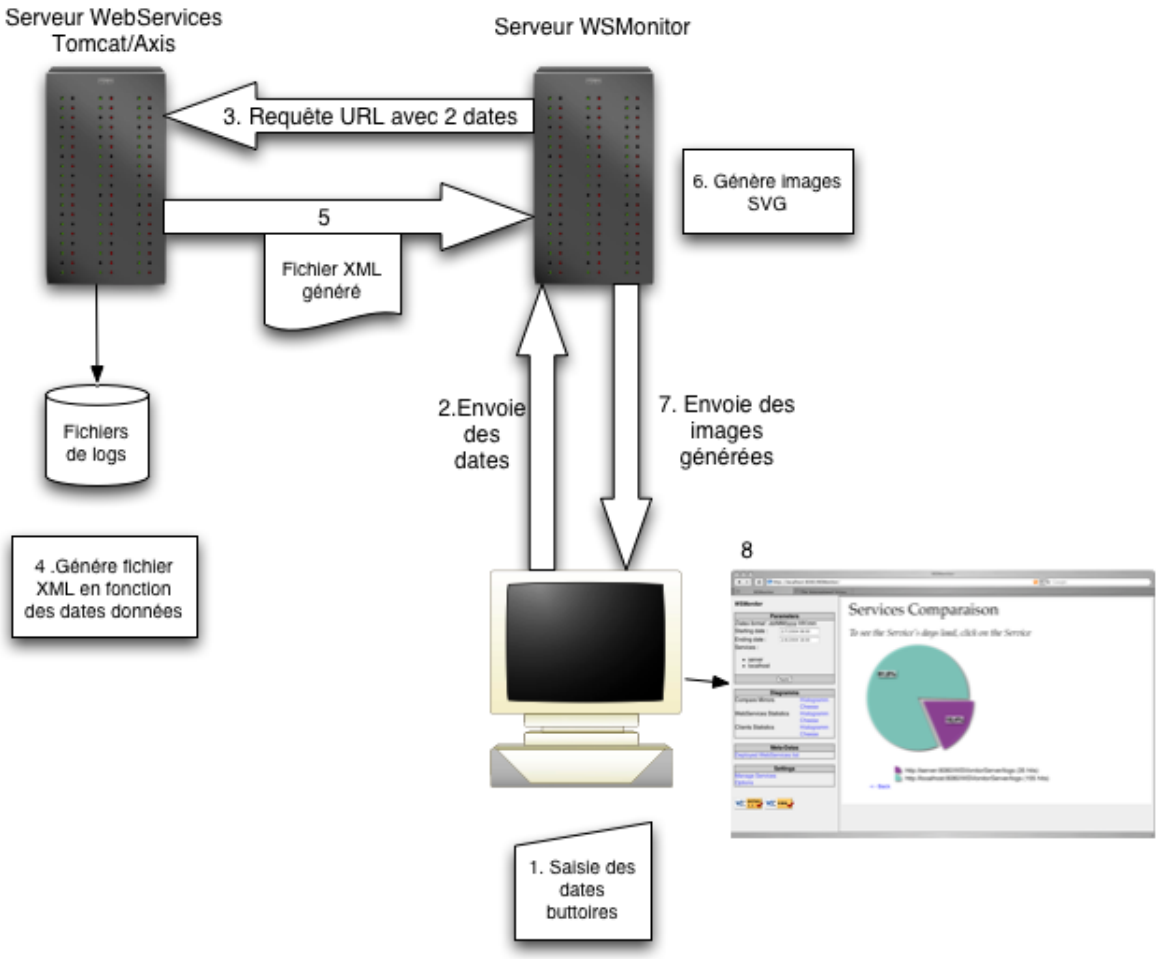


FIG. 4.1 – Fonctionnement général de WSMonitor

D'autre part, le serveur d'application J2EE utilisé est Tomcat[13]. Celui-ci étant également déjà employé pour le déploiement d'Axis.

4.2.2 Images SVG

Il est très vite apparu que l'utilisation d'images serait nécessaire pour faciliter la présentation et la compréhension des statistiques. C'est pourquoi nous avons fait le choix d'utiliser SVG (Scalable Vector Graphics).

Ce langage permet d'écrire des graphiques vectoriels 2D en XML. Il a été inventé en 1998 par un groupe de travail (comprenant Microsoft, Autodesk, Adobe, IBM, Sun, Netscape, Xerox, Apple, Corel, HP, ILOG...) pour répondre à un besoin de graphiques légers, dynamiques et interactifs.

Ce choix fait ressortir les avantages suivants :

1. liés aux avantages du graphisme vectoriel :
   - adaptation de l'affichage à des media variés et à des tailles différentes
   - possibilité d'appliquer des styles
   - possibilité d'indexer le texte qui fait partie du graphisme
   - facilité d'édition : les éléments sont des objets, des hiérarchies, etc.
2. liés aux avantages particuliers du format SVG :
   - génération de SVG avec XSLT à partir de données XML
   - future integration totale dans XHTML, viewers SMIL etc.
   - utilisation de CSS
   - scriptable avec JavaScript via DOM.
   - modèle de couleurs sophistiqué, filtres graphiques (comme dans PhotoShop)
   - possibilité d'indexage par les moteurs de recherche (c'est un fichier texte)
   - possibilité de partager du code (format texte non propriétaire)

Ce format d'image nécessite un plugin spécifique à installer sur le client. Néanmoins de très nombreuses implémentations existent (liste complète http://www.w3.org/Graphics/SVG/SVG-Implementations.htm8) et SVG sera bientôt supporté nativement par les navigateurs.

4.2.3 Divers

Pour le développement de la solution, les outils suivants ont été utilisés :

- Eclipse http://www.eclipse.org
- plugin Tomcat pour Eclispe : Sysdeo http://www.sysdeo.com/eclipse/tomcatPluginFR.html

– plugin j2ee pour Eclipse : Lomboz http://www.objectlearn.com/index.jsp

4.3 Récupération des fichiers journaux

4.3.1 Téléchargement différentiel

Les fichiers journaux sont téléchargés à partir du servlet `WSMonitorServer` déployé sur chaque serveur de WebServices.

Comme nous avons pu voir précédemment (voir 4.1.1), ce servlet prend deux dates en paramètres. Au démarrage de la session du client, ces dates sont initialisées à la date de jour entre 8 et 18 heures. Celles-ci pourront être modifiées par le client à tout moment à l'aide de l'interface Web.

Après un premier téléchargement complet, seules les nouvelles requêtes seront transférées. Pour ce faire, après un premier téléchargement, au lieu de donner la date choisie par le client (ou celle par défaut) comme date de début, nous donnons la dernière date précédemment récupérée. La date de fin quant à elle reste inchangée.

4.3.2 Stockage

Une fois téléchargé, chaque fichier journal est désérialisé à l'aide de JAXB. Les modèles objets correspondants sont stockés dans une `Hashtable` dont les clés sont les URL des différents serveurs . Cette `Hashtable` est ensuite enregistrée dans la session du client.

4.4 Génération des images

4.4.1 Analyse des informations

Les informations précédemment téléchargées et stockées doivent être analysées. Ceci se fait grâce à la classe `wsmonitor.Statistiq`. Toutes les méthodes de cette classe renvoient un `Vector` contenant des objets `wsmonitor.Data`.

La classe `wsmonitor.Data` dispose des attributs suivants :
– `String label` : l'étiquette l'élément qui sera affichée dans l'image
– `int occurences` : le nombre d'occurrences

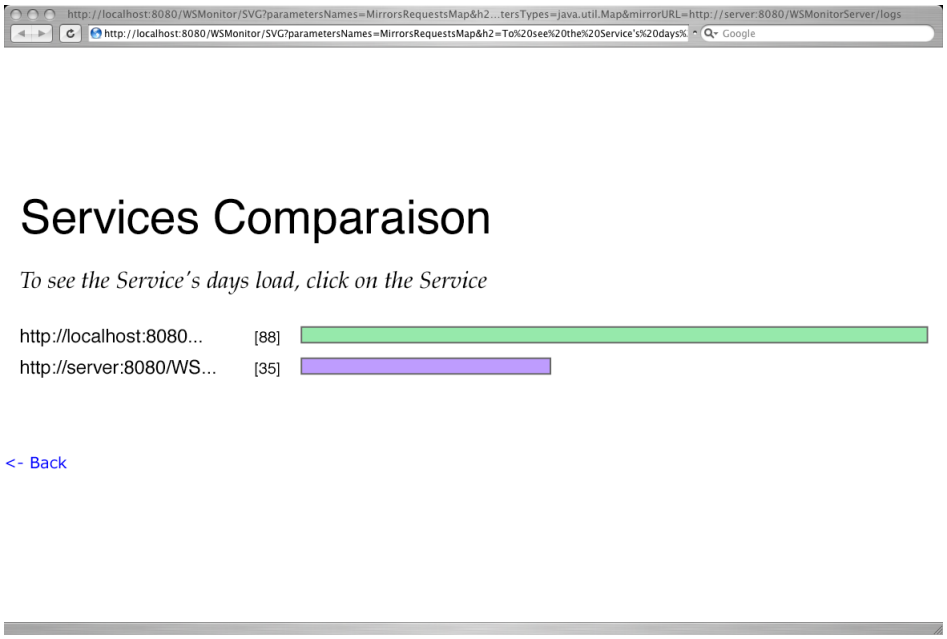


FIG. 4.2 – Exemple d'histogramme généré

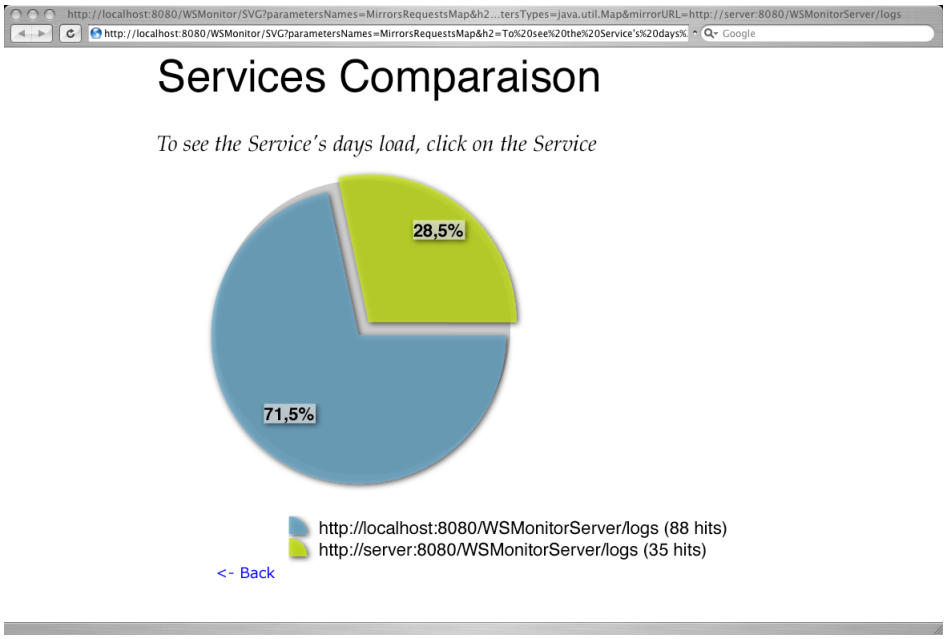


FIG. 4.3 – Exemple de camembert généré

4.4.2 Servlet SVGenerator

Ce servlet est dédié à la génération des diagrammes au format SVG. Deux types de diagrammes sont proposés : histogramme (Fig 4.2) ou camembert (Fig 4.3).

`SVGenerator` requiert les paramètres suivants :

1. `className` : nom de la classe permettant d'analyser les fichiers journaux.
2. `methodName` : nom de la méthode de la classe.
3. `parametersTypes` : types des paramètres de la méthode.
4. `parameterNames` : noms des paramètres, ceux-ci devant être enregistrés dans la session du client ou passés dans la requête.
5. `type` : type de diagramme à générer (histogramme ou camembert),
6. `h1` : titre du diagramme,
7. `h2` : sous-titre du diagramme,
8. `link` : lien associé aux éléments du diagramme,
9. `linkParamN` : paramètre passé au lien.

4.5 Liste des services déployés

4.5.1 Récupération de la liste

Nous disposons d'un WebService `WSRegistry` (voir 3.1.2). Celui-ci nous renvoie un fichier XML contenant la liste des services déployés.

De plus, une page JSP permet de sélectionner le miroir à interroger. Un fois le choix effectué, le WebService `WSRegistry` est effectivement appelé à l'aide d'un client `Axis`.

4.5.2 Mise en forme

Après avoir interrogé le WebService et récupéré le fichier XML, il faut le mettre en forme pour l'affichage. Pour cela, la taglib `xtags` a été utilisée. Celle-ci permet de transformer le fichier XML en page HTML grâce à XSLT.

XSLT (Extensible Style Language Transformations) est, comme son nom l'indique, un langage destiné à transformer un fichier XML en quelque chose d'autre (HTML dans notre cas). C'est un langage déclaratif et non procédural, ce qui revient à dire qu'à la différence d'un langage de programmation classique, il ne spécifie pas "le comment" (les algorithmes) ? il se contente de déclarer "le quoi ?" : par exemple que tout ou partie des

balises `<para>` présentes dans le XML source sont à remplacer dans le HTML cible par des balises `<p>`.

4.5.3 Teste des WebServices

Au sein de l'IVOA, une interface standard pour les WebServices est en cours de développement. Une méthode devra permettre de savoir si le service est disponible et renverra un fichier XML contenant les informations suivantes :

- `Uptime` : le nombre de jours depuis lequel le service est déployé
- `ValidTo` : la date à laquelle le prochain arrêt du service est prévue
- `ContactDetails` : nom, e-mail et téléphone de la personne responsable
- `Position` : coordonnées GPS du serveur.

Au vu du statut de "Working Draft" de cette interface, cette fonctionnalité n'est que partiellement implémentée.

Bien sur, ce mécanisme n'est d'aucune utilité si le WebService lui-même est en panne. Néanmoins dans le milieu astronomique, les WebServices sont dans la plupart des cas uniquement un moyen d'accès aux ressources, lesquelles se trouvent dans des base de données, annuaires ou autre. L'intérêt est donc d'informer de la disponibilité des ressources utilisées par le WebService.

Conclusion

Comme cela avait était le cas l'année dernière pour les WebServices SOAP, ce stage m'a fait découvrir des nouvelles technologies, en l'occurrence SVG et JAXB. Ces technologies d'avenir encore assez peu répondues, sont en phase des devenir des standards incontournables. Il a donc été très enrichissant de pouvoir les apprivoiser.

De plus, ce qui au départ devait être un outil de supervision de WebServices uniquement, est maintenant également utilisé pour superviser plusieurs clusters mis en place à l'Observatoire.

Ce projet sera présenté sous forme de poster à la conférence 'Astronomical Data Analysis Software and Systems' (ADASS[14]) qui se tiendra à Pasadena (USA - Californie) du 24 au 27 octobre 2004.

Une version de ce rapport est disponible en ligne à l'adresse suivante : http://jfnicolas1.free.fr/rapport/.

Annexes : Captures d'écrans

Dans la partie gauche de chaque écran se trouve :

1. les dates permettant de spécifier la période souhaitée pour la visualisation
2. la liste des services pris en compte, cette liste est modifiable par l'administrateur uniquement.
3. les différents liens

La figure 4.4 présente une image permettant de comparer le taux d'utilisation des différents miroirs.

La figure 4.5 est générée suite à un click sur la partie `localhost` de l'image 4.4. Elle présente donc la répartition par date de l'utilisation des WebServices du serveur local.

La figure 4.6 présente une page permettant de visualiser pour chaque miroir, le taux d'utilisation des différents WebServices.

La figure 4.7 est générée suite à un click sur le WebService `LogTestService1` du miroir local sur l'image 4.6. Elle présente donc la répartition par méthodes de l'utilisation du WebService `LogTestService1` du serveur local.

Les figure 4.8 et 4.9 présentent une page permettant de visualiser pour chaque miroir, la répartition des requêtes par clients.

La figure 4.10 est générée suite à un click sur le client `127.0.0.1` du miroir local sur l'image 4.9. Elle présente l'historique des connexion du client sur le serveur local.

La figure 4.11 présente une page permettant de visualiser pour chaque miroir, la répartition des requêtes par temps d'exécution.

La figure 4.12 présente une page permettant de visualiser la liste des services déployés comme cela a été développé en 4.5.

La figure 4.13 présente une page permettant de gérer la liste des services (miroirs). Cette page requiert l'authentification en tant que manager et permet d'ajouter ou de supprimer des services.

Enfin, la figure 4.14 présente une page permettant de modifier l'intervalle de rafraîchissement automatique des pages.

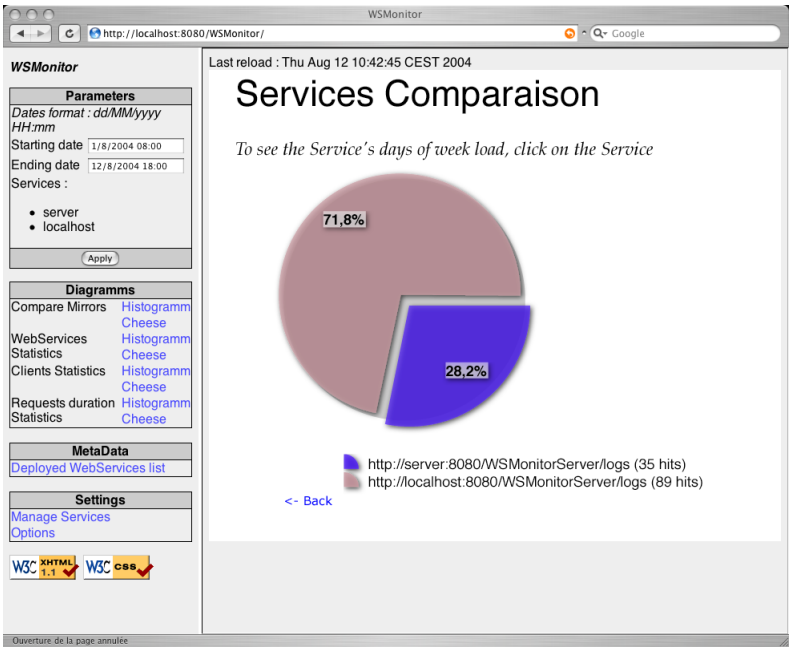


FIG. 4.4 – Comparaison des services entre eux

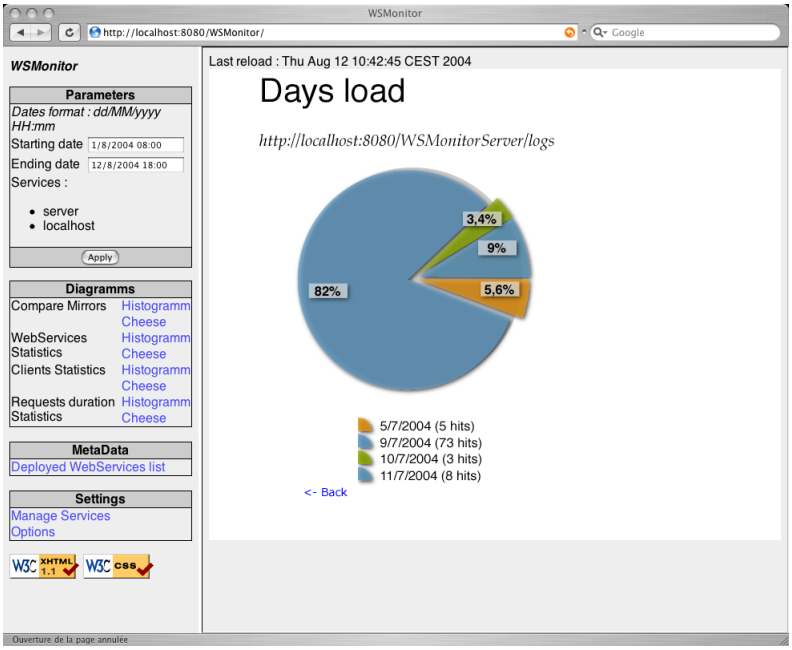


FIG. 4.5 – Charge d'un service par jour

4.5 Liste des services déployés

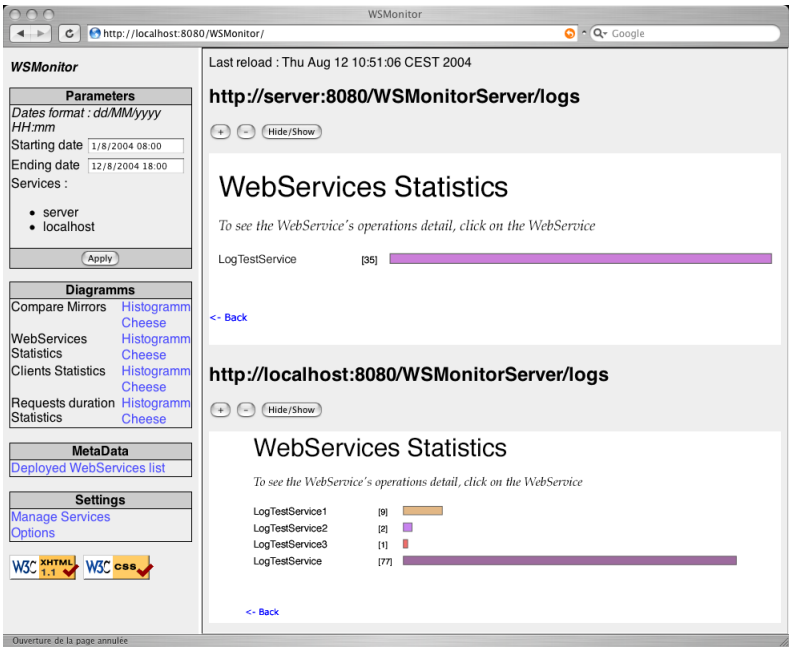


FIG. 4.6 – Statistiques par WebService

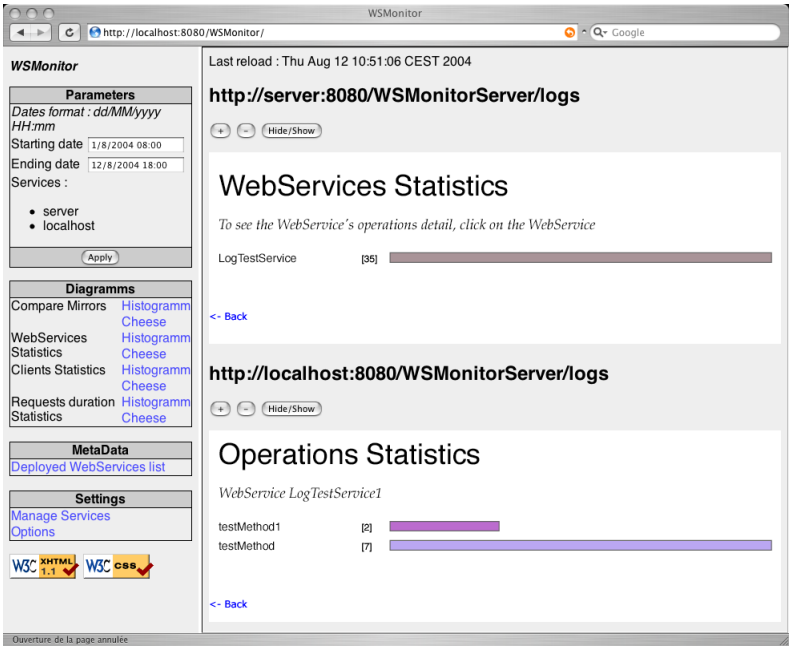


FIG. 4.7 – Statistiques par opérations pour un WebService donné

4.5 Liste des services déployés

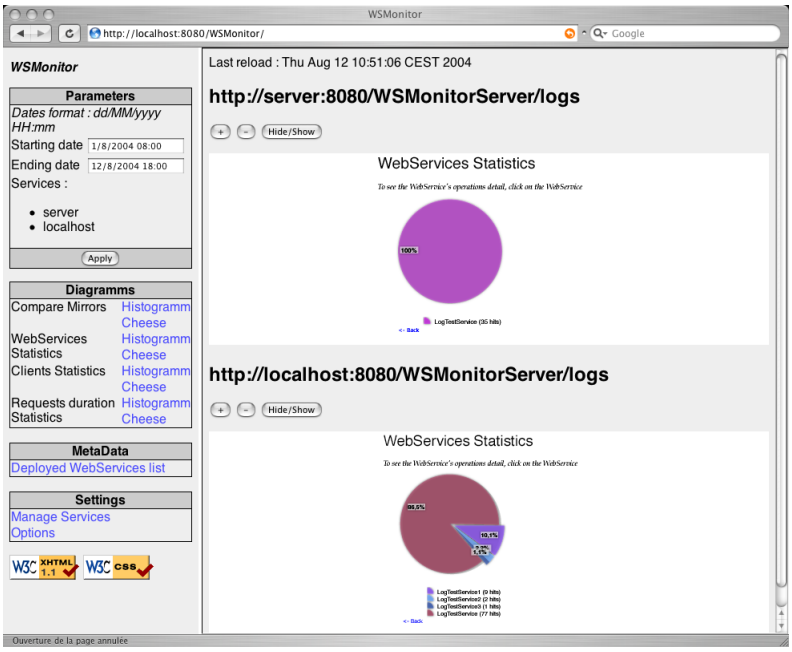


FIG. 4.8 – Statistiques par WebService (format camembert)

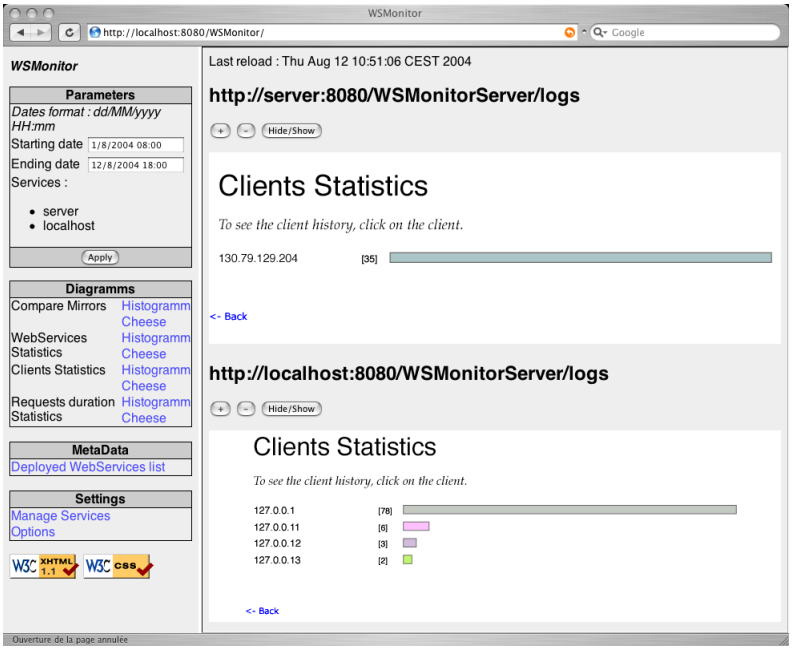


FIG. 4.9 – Statistiques par clients consommateurs de WebServices

4.5 Liste des services déployés

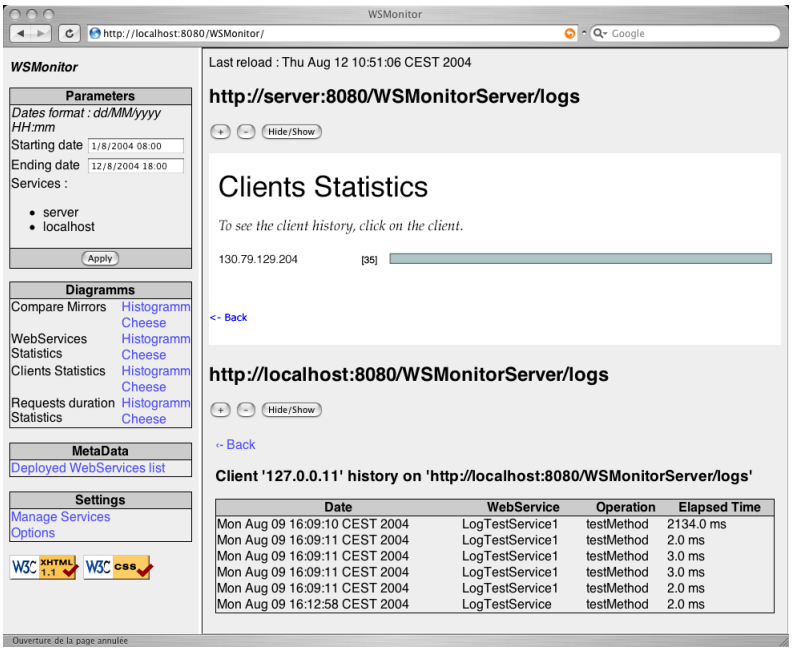


FIG. 4.10 – Détails des connexions d'un client

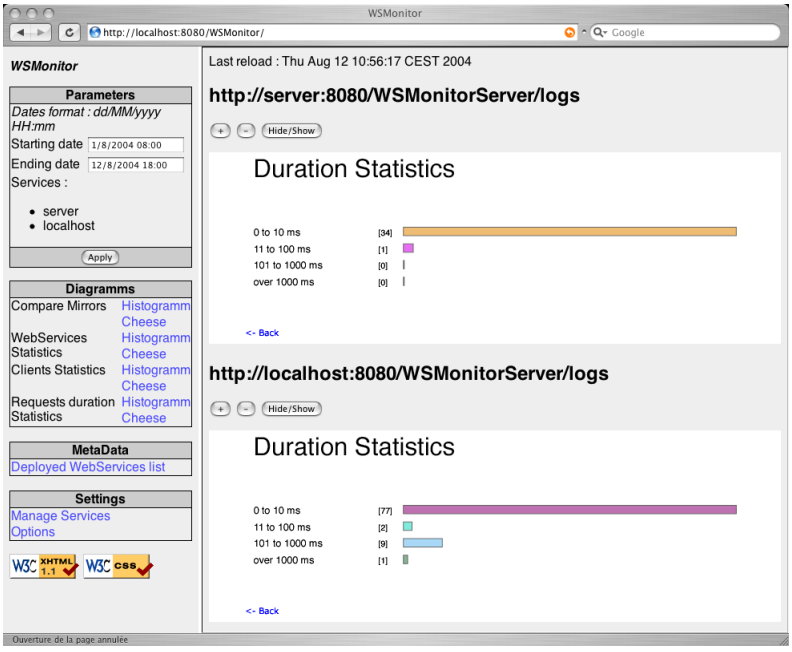


FIG. 4.11 – Statistiques sur le temps d'exécution des requêtes

4.5 Liste des services déployés

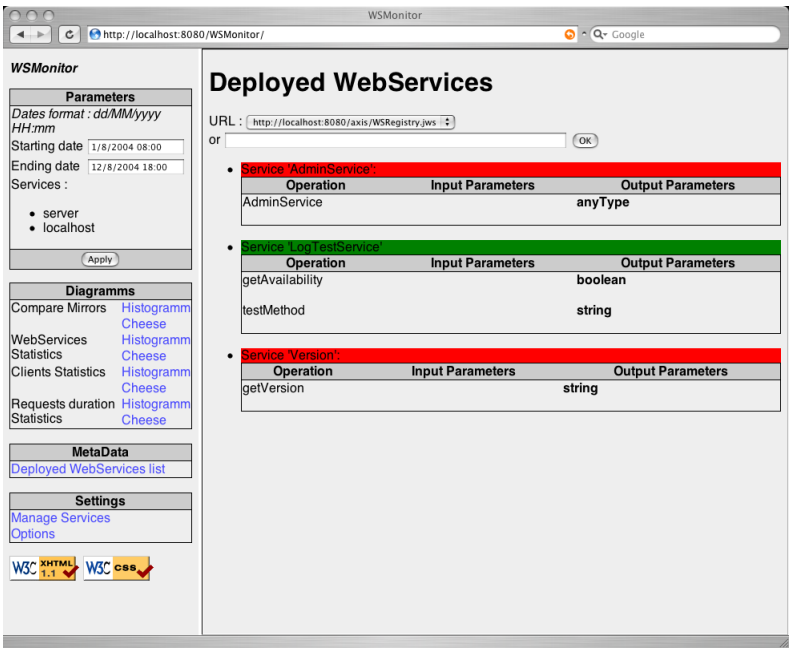


FIG. 4.12 – Liste des services déployés

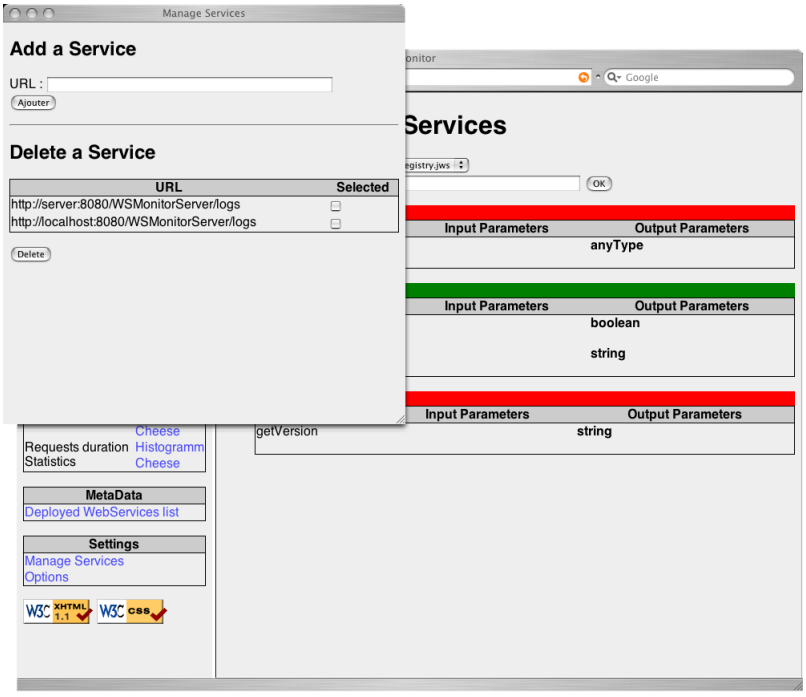


FIG. 4.13 – Gestion de la liste des services (administrateur seulement)

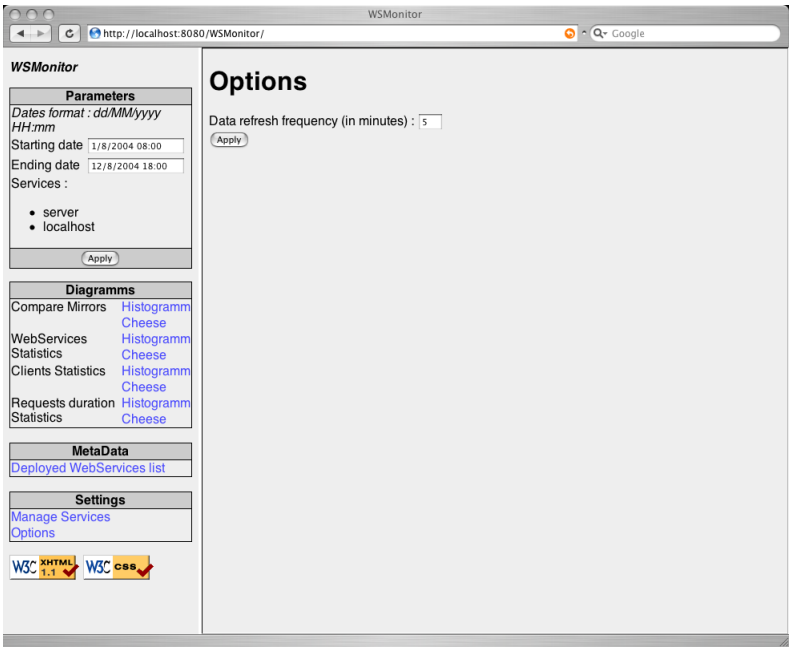


FIG. 4.14 – Options

Références

[1] Site internet de l'Observatoire astronomique de Strasbourg : http://astro.u-strasbg.fr/Obs.html

[2] Site internet de l'équipe Hautes Energies : http://astro.u-strasbg.fr/Obs/ENERGIE/ENERGIE.html

[3] Site internet de l'équipe Etoiles et système stellaire : http://astro.u-strasbg.fr/Obs/PHYSTEL/PHYSTEL.html

[4] Site internet de l'équipe Galaxies : http://astro.u-strasbg.fr/Obs/GALAXIES/

[5] Site internet du CDS : http://astro.u-strasbg.fr/Obs/CDS/CDS.html

[6] Site internet du service Simbad : http://simbad.u-strasbg.fr/Simbad

[7] Site internet du service VizieR : http://vizier.u-strasbg.fr/

[8] Site internet du service Aladin : http://aladin.u-strasbg.fr/aladin.gml

[9] Site internet du consortium IVOA : http://www.ivoa.net/

[10] Site internet de l'API JAXB : http://java.sun.com/xml/jaxb/

[11] Site internet de l'utilitaire log4j : http://logging.apache.org/log4j/

[12] Site internet des technologies j2ee : http://java.sun.com/j2ee/

[13] Site internet du serveur d'application J2EE http://jakarta.apache.org/tomcat/

[14] Site internet de la conférence ADASS http://adass.org