

**Interopérabilité des Services Web en environnement .Net et Java**

Microsoft Corporation

Publication : Juin 2007

*Auteurs :* ***Benjamin BELLON-SERRE*** *et* ***Stève SFARTZ***

*Architecte en système d’information*

[*ssfartz@microsoft.com*](mailto:ssfartz@microsoft.com)*,* [*http://blog.sfartz.com*](http://blog.sfartz.com)

Pour obtenir de plus amples renseignements sur ce sujet, visitez le site <http://www.microsoft.com/france/msdn/architects>.

Les informations contenues dans ce document représentent l’opinion actuelle de Microsoft Corporation sur les points cités à la date de publication. Microsoft s’adapte aux conditions fluctuantes du marché et cette opinion ne doit pas être interprétée comme un engagement de la part de Microsoft. De plus, Microsoft ne peut pas garantir la véracité de toute information présentée après la date de publication.

Ce livre blanc doit être utilisé à titre informatif uniquement. MICROSOFT EXCLUT TOUTE GARANTIE EXPRESSE, IMPLICITE OU LÉGALE, RELATIVEMENT AU CONTENU DE CE DOCUMENT.

L’utilisateur est tenu d’observer la réglementation relative aux droits d’auteur applicable dans son pays. Aucune partie de ce document ne peut être reproduite, stockée ou introduite dans un système de restitution, ou transmise à quelque fin ou par quelque moyen que ce soit (électronique, mécanique, photocopie, enregistrement ou autre) sans la permission expresse et écrite de Microsoft Corporation.

Les produits mentionnés dans ce document peuvent faire l’objet de brevets, de dépôts de brevets, de marques, de droits d’auteur ou d’autres droits de propriété intellectuelle et industrielle de Microsoft. Sauf stipulation expresse contraire d’un contrat de licence écrit de Microsoft, la fourniture de ce document n’a pas pour effet de vous concéder une licence sur ces brevets, marques, droits d’auteur ou autres droits de propriété intellectuelle.

© 2007 Microsoft Corporation. Tous droits réservés.

Microsoft, Active Directory, Visual Basic, Windows, Windows NT et Windows Serversont soit des marques de Microsoft Corporation, soit des marques déposées de Microsoft Corporation aux États-Unis d’Amérique et/ou dans d’autres pays.

Les noms des produits et des entreprises mentionnés dans ce document peuvent être des marques déposées par leurs propriétaires respectifs.

SOMMAIRE

[I. Introduction 3](#_Toc169342345)

[A. Pré-requis 4](#_Toc169342346)

[B. Domaine d’expérimentation de l’interopérabilité 4](#_Toc169342347)

[C. Choix des technologies 6](#_Toc169342348)

[D. Exemple de grille de lecture 7](#_Toc169342349)

[II. Services WEB : Principes et Interopérabilite 8](#_Toc169342350)

[A. L’Architecture Orientée Service (SOA) 8](#_Toc169342351)

[1. Qu’est-ce que l’architecture orientée service ? 8](#_Toc169342352)

[2. Que sont les services web ? 11](#_Toc169342353)

[B. Comment assurer l’interopérabilité ? 12](#_Toc169342354)

[1. Qu'est-ce que l’interopérabilité ? 12](#_Toc169342355)

[2. Respect des standards 12](#_Toc169342356)

[a. http 12](#_Toc169342357)

[b. XML 12](#_Toc169342358)

[c. WSDL 12](#_Toc169342359)

[d. SOAP 13](#_Toc169342360)

[e. Les différents types d'encodage des messages 14](#_Toc169342361)

[f. WS-\* (STAR) 16](#_Toc169342362)

[3. L’organisme WS-I et les outils de vérification 17](#_Toc169342363)

[a. Les différents profils 17](#_Toc169342364)

[b. Les outils de vérification 18](#_Toc169342365)

[4. Stratégies d’implémentation : Code First versus Contract First 19](#_Toc169342366)

[a. Implémentation First 19](#_Toc169342367)

[b. WSDL First 20](#_Toc169342368)

[C. Les implémentation des services web 22](#_Toc169342369)

[1. Que propose Microsoft ? 22](#_Toc169342370)

[2. Que propose les environnements Java 23](#_Toc169342371)

[D. Les serveurs d’applications et les modes d’encodage 24](#_Toc169342372)

[E. La gestion des exceptions 25](#_Toc169342373)

[III. Environnement technique 27](#_Toc169342374)

[A. Configuration logicielle requise 27](#_Toc169342375)

[B. Les outils pour comprendre les services web 27](#_Toc169342376)

[1. Comprendre les échanges SOA 27](#_Toc169342377)

[2. Comprendre le contrat WSDL 28](#_Toc169342378)

[C. Procédure d’installation des environnements 28](#_Toc169342379)

[1. Environnement .Net 28](#_Toc169342380)

[2. Environnement Java 29](#_Toc169342381)

[IV. Java consomme des web services .Net 34](#_Toc169342382)

[A. Création d'un Service Web .Net 34](#_Toc169342383)

[1. Un service web utilisant des types primitifs 35](#_Toc169342384)

[2. Un service web utilisant des tableaux et des collections 40](#_Toc169342385)

[3. Un service web utilisant des types complexes 42](#_Toc169342386)

[4. Un service web utilisant des exceptions 46](#_Toc169342387)

[B. Java (Proxy Axis 1.4) consomme des web services .Net 51](#_Toc169342388)

[C. Java (Proxy Axis 2.1.1.1) consomme des web services .Net 56](#_Toc169342389)

[D. Java (Proxy JAX-WS 2.1) consomme des web services .Net 61](#_Toc169342390)

[V. .Net 3.0 consomme des web services Java 63](#_Toc169342391)

[A. Création d’un Web Service Java (Axis 1.4) 63](#_Toc169342392)

[B. Création d’un Web Service Java (Axis 2.1.1.1) 65](#_Toc169342393)

[C. Création d’un Web Service Java (GlassFish v2) 69](#_Toc169342394)

[D. .Net 3.0 consomme des web services Java 73](#_Toc169342395)

[E. .Net 3.0 consomme des web services GlassFish V2 78](#_Toc169342396)

[VI. Tester la conformité WS-I 80](#_Toc169342397)

[A. Monitor 80](#_Toc169342398)

[1. Utilisation de l’outil Monitor 80](#_Toc169342399)

[2. Exemple de trace SOAP avec Monitor 82](#_Toc169342400)

[B. Validator 84](#_Toc169342401)

[1. Utilisation de l’outil Validator 84](#_Toc169342402)

[2. Exemple de rapport avec Validator 86](#_Toc169342403)

[VII. SYNTHESE sur les experimentations 89](#_Toc169342404)

[a. Types échangés entre un serveur .Net et un client Java 91](#_Toc169342405)

[a. Types échangés entre un serveur Java et un client .Net 94](#_Toc169342406)

[VIII. PERSPECTIVE : JAVA et .NEt SONT-ils INTEROPERABLEs ? 97](#_Toc169342407)

[IX. POUR ALLER PLUS LOIN 99](#_Toc169342408)

1. Introduction

Les systèmes d’informations sont par nature hétérogènes en terme de technologies, si bien que la problématique d’intégration est récurrente, notamment au niveau des échanges inter-applications.

Nous nous concentrons dans ce document sur les Services Web, et l’interopérabilité dans ce contexte entre des environnements.Net et Java. Il existe bien d’autres façons d’intégrer des applications .Net et Java, depuis les technologies de bridging, ou encore les API de plus bas niveau de type COM, jusqu’à l’utilisation de middlewares orientés messages ou encore les dialogues de type CORBA. Pour plus d’informations à ce sujet, consulter le [portail interopérabilité de Microsoft France](http://www.microsoft.com/france/interop/themes/applications/20070509-interop-NET-J2EE/default.mspx).

La pile des spécifications liées aux Services Web permet de couvrir des scénarios les plus basiques :

échanges de messages au format SOAP - aux scénarios les plus complexes - garantie de livraison, sécurisation des échanges, maintien d’un contexte transactionnel...

Vous trouverez dans ce document une analyse des problématiques associées à des scénarios basiques d’interopérabilité, mettant en œuvre les frameworks orientés services de dernière génération : il s’agit de Windows Communication Foundation (WCF) du côté de Microsoft .Net et de l’implémentation JAX-WS de SUN (JAX-WS RI) côté Java. Afin d’illustrer cette interopérabilité avec des codes existants en entreprise, nous avons choisi de présenter de façon complémentaire des exemples basés sur le framework historique de mise en œuvre des services Web Java : AXIS d’Apache.

Il est difficile de couvrir de façon exhaustive le sujet de l’interopérabilité même en nous concentrant comme nous l’avons fait sur des scénarios basiques. Aussi, notre approche dans ce document a été de présenter des scénarios d’interopérabilité de façon didactique, afin de vous permettre de partir de ces exemples et les étendre à votre environnement.

Nous fournissons pour chaque scénario l’environnement technique associé, le code de notre implémentation, ainsi que nos constatations et conclusions sur la problématique couverte par chaque scénario. Nous vous invitons à revenir vers nous pour apporter vos expériences complémentaires à ce document par [email](mailto:ssfartz@microsoft.com).

Une vidéo accompagnant ce document propose une synthèse des éléments présentés dans ce document ainsi que des démonstrations en environnement Visual Studio / .Net et NetBeans / Java.

Enfin, le lecteur intéressé par le respect des standards se reportera à la section WS-I située en fin de document pour nos tests d’utilisation des outils proposés par le W3C.

Nous espérons que ce document en illustrant quelques problématiques récurrentes liées à l’utilisation des technologies Services Web, vous apportera quelques réponses et bonnes pratiques pour faciliter l’interopérabilité de vos implémentations.

1. Pré-requis

Ce document s’adresse à des architectes .Net et Java qui veulent comprendre les points à considérer dans l’interopérabilité entre ces deux environnements, ainsi qu’à des développeurs .Net et Java à la recherche d’exemples de code. Nous considérons que le lecteur connaît les principes des architectures orientées services ainsi que les technologies .Net et Java.

Nous effectuons un rappel des concepts associés aux architectures de services et aux technologies Services Web en préambule, et en nous concentrant sur les aspects liés à l’interopérabilité.

1. Domaine d’expérimentation de l’interopérabilité

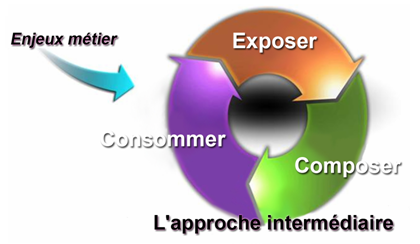
Les principes des architectures SOA permettent de distinguer **trois tiers** :

**- Consommation (Ou encore tiers d’accès)** : Le tiers de consommation représente en général une interface client sous la forme d’un client riche ou d’un client léger qui accède à des services

**- Composition (Ou encore tiers de médiation)** : Dans les environnements répartis, l’accès à un service reste complexe. En effet, le client doit gérer les différentes étapes de l’accès, il ne dispose pas de moyen standard pour connaître l’état du fournisseur de services qu’il va invoquer et il ne peut invoquer que les services dont il possède les fonctions d’accès. Dans ce contexte, le **Service de Médiation** permet de gérer, pour le compte du client les trois étapes de l’accès à un service, à savoir la **recherche** et la **localisation des fournisseurs de services**, le **choix du fournisseur** en tenant compte de l’environnement d’exécution, et l’**invocation du service**. Le Service de Médiation permet donc de déporter l’accès à un service sur une machine différente de celle du client.

**- Exposition (Ou encore tiers Métier)** : Ce sont les applications offrant des services et contenant le code métier.

Ces principes sont exposés dans le [livre blanc de l’approche SOA Microsoft](http://download.microsoft.com/download/1/5/4/154ac117-5bee-475b-aadb-302eef4e8a16/RealWorldSOA_WP.doc).



*Approche Microsoft SOA Pragmatique*

**Les scénarios présentés dans cet article traitent de l’interopérabilité entre les niveaux d’exposition et de consommation, en détaillant comment :**

* + La technologie de Services Web Microsoft est consommable depuis des clients Java
  + Les applications .Net WCF peuvent consommer des services exposés depuis des environnements Java

L’interopérabilité des échanges de type Services Web fait intervenir deux domaines :

1. **La nature des informations échangées :**

Parmi les échanges entre les environnements .Net et Java, nous allons traiter l’ensemble des types de données possibles afin de détecter les types qui passent sans difficulté et les types qui posent problèmes entre ces deux environnements.

* **Des types primitifs :** boolean, byte, double, float, int, long et short, DateTime, String,…
* **Des tableaux et des collections :** int[], String[], Long[], Byte[], Classe[]
* **Des types complexes :** Classes utilisateurs, structures et DataSet
* **Des exceptions :** Exception déclarée ou non déclarée

1. **Le format des informations échangées :**

Les standards associés aux Services Web autorisent l’utilisation de différents formats d’encodage (cf. partie II.B.2.e. Les différents types d’encodage). De plus, la norme SOAP de formatage des messages est disponible en différentes versions (cf. partie II.B.2.d. SOAP).

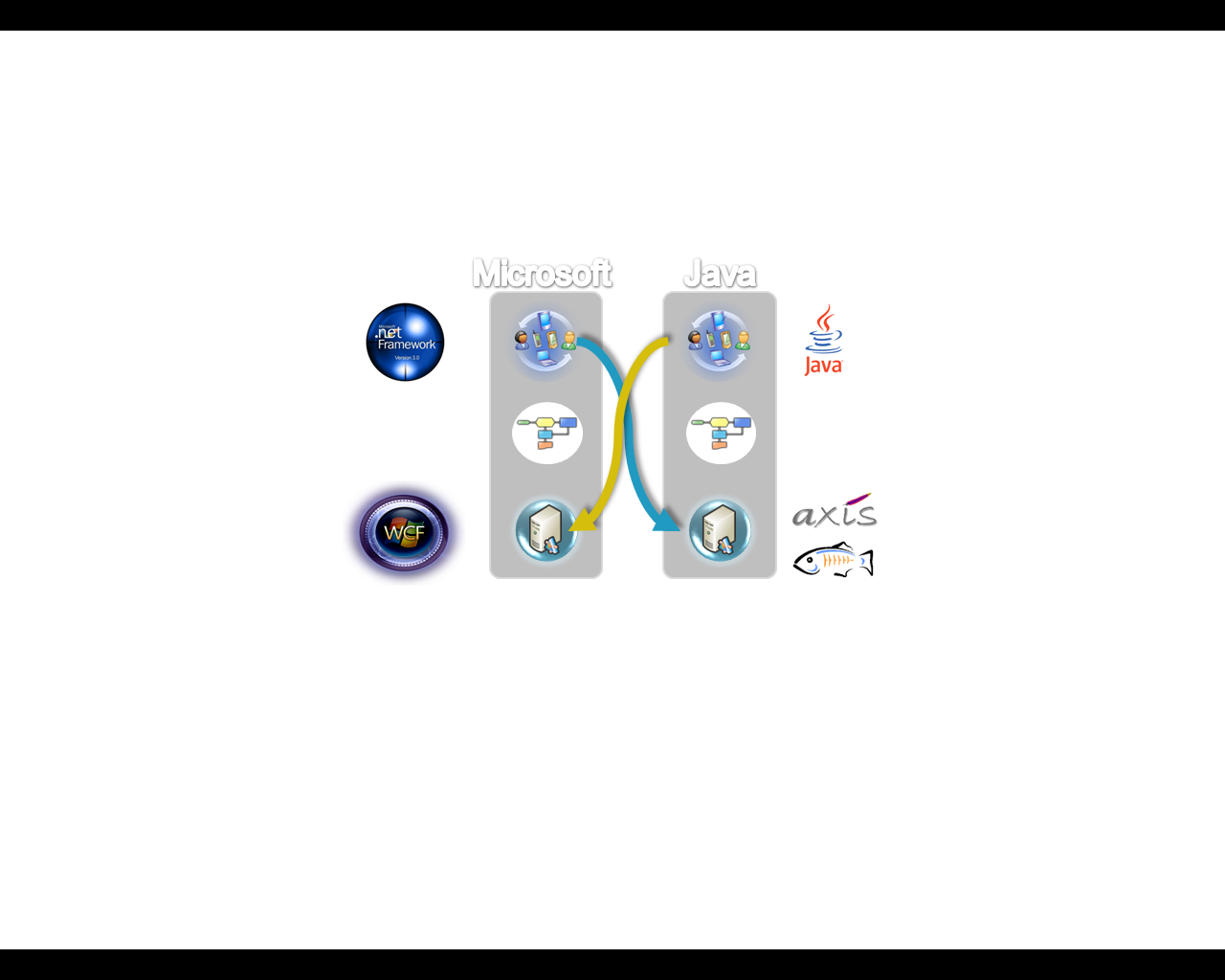
Les échanges que nous décrivons dans la suite de ce document présentent les combinaisons suivantes parmi les options disponibles pour l’implémentation de services Web :

* RPC/ Encoded en SOAP 1.1
* Document / Literal en SOAP 1.1 et 1.2
* Document / Literal Wrapped en SOAP 1.1 et 1.2

Certaines technologies ne proposent qu’un format d’encodage et/ou ne supportent qu’une seule version du protocole SOAP. Aussi, nous présenterons les options les plus pertinentes pour un consommateur donné et le producteur correspondant.

1. Choix des technologies

Les scénarios présentés traitent uniquement des cas **d’interopérabilités basiques** entre les niveaux d’**exposition** et de **consommation des stacks .Net et Java**. On ne traitera ni des protocoles WS-\*, ni d’aspects sécurité dans ces scénarios.



Nous avons utilisé les serveurs d’applications proposés par Apache et SUN dans les versions suivantes : Axis 1.4, Axis 2.1.1.1 et GlassFish V2.

Nous avons sélectionné ces serveurs d’applications J2EE pour leur notoriété, le fait que la communauté Java les considère comme des référents et dans la mesure où ils supportent des normes et encodage différents, ils nous ont permis d’effectuer une large gamme de tests pour la consommation et production depuis des applications Microsoft .Net WCF.

Axis 1.4 permet de tester WCF avec l’utilisation de **l’ancien mode d’encodage RPC / Literal**. Axis 2.1.1.1 est une réécriture complète d’Axis 1.4 et apporte le **support de SOAP 1.2 et du mode d’encodage Document/Literal**. GlassFish 2.1 permet de tester les **nouvelles spécifications JAX-WS** **2.1** qui constituent le standard en Java pour la création des Services Web.

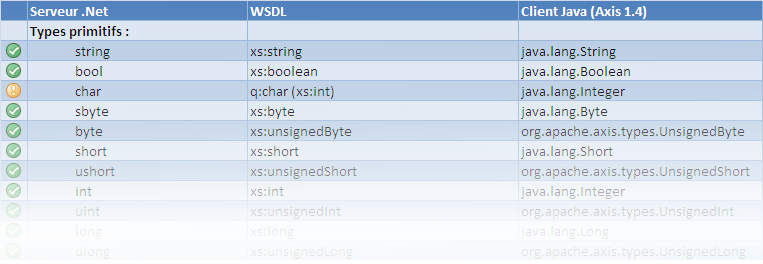


*Modes d’encodages et versions SOAP supportées par les frameworks de services web*

1. Exemple de grille de lecture

Vous trouverez en synthèse des scénarios présentés, et pour chaque scénario, une grille de lecture pour les types échangés. La première colonne présente les types au niveau du producteur, la deuxième colonne décrit les types au niveau du contrat WSDL et des messages SOAP. Enfin la troisième colonne présente les types au niveau du consommateur.

|  |  |
| --- | --- |
| *Producteur* : WCF  *Consommateur* : Java Axis 1.4 | *Version de SOAP* : SOAP 1.1  *Mode d’encodage* : Document / Literal Wrapped |





*Exemple de grilles de lecture*

Légende :

Les cas qui fonctionnent correctement (comportement attendu)

Les cas qui posent quelques problèmes mais n’empêchent pas le fonctionnement (solution de contournement, exécution dégradée)

Les cas qui ne fonctionnent pas (anomalie, comportement incorrect…)

interoWordLes cas qui ne font pas de sens (à l’encontre des spécifications ou des bonnes pratiques)

1. Services WEB : Principes et Interopérabilite
2. L’Architecture Orientée Service (SOA)
3. Qu’est-ce que l’architecture orientée service ?

L'architecture orientée service permet d'offrir une réponse aux problématiques que rencontrent les entreprises en terme de **réutilisabilité**, d'**interopérabilité** et de **réduction de couplage** entre les différents systèmes composants des systèmes d'information.

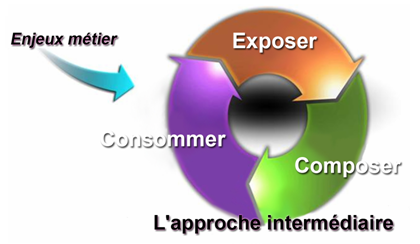
La SOA apporte une aide à l’**urbanisation du SI**: en effet, cette architecture permet de connecter et de faire collaborer des **systèmes hétérogènes** et d’offrir une **consommation multi-points** (Client Web, Client Desktop, Client Mobile,…).

Cette architecture apporte à l'entreprise une **ouverture** et une **transparence** vis-à-vis de ses partenaires, fournisseurs et clients, ce qui lui permet d’augmenter son **agilité** afin d’être plus réactif aux changements.

L’offre de services **réutilisables** améliore la **productivité** des collaborateurs humains et facilite la **maintenance** de ces services.

La mise en œuvre d’une SOA ne nécessite **pas nécessairement des investissements colossaux, et** peut être réalisée au travers d’une méthodologie **incrémentale.**

Microsoft préconise le cycle de vie SOA suivant :



*Cycle de vie d’une SOA*

L’architecture orientée service (SOA : Service Oriented Architecture) est un concept relativement récent dans le monde de l’interopérabilité. Le principe sous-jacent derrière le concept de la SOA réside dans l’intégration et la communication des systèmes informatiques, des logiciels, des équipements et des services, sans qu’ils aient été spécifiquement conçus pour cela à l’origine. Une SOA est mise en œuvre à l’aide de services Web, et les applications sont construites en tant qu’ensembles de services réutilisables et coopérant. Au sein de ces ensembles, chaque service est responsable d’un ou plusieurs processus, tâches ou services métier clairement identifiés et liés.

Souvent, ce modèle ne comprend pas de couche de présentation ; en effet, les interactions se font uniquement d’ordinateur à ordinateur et elles ne sont pas destinées à la présentation ni à l’interaction humaine. Cependant, les composants sous-jacents peuvent présenter des fonctionnalités et interfaces utilisables par les couches de présentation écrites de manière indépendante - et qui pourraient fonctionner, par exemple, via un navigateur, ou via une application cliente riche comme Microsoft Office s’exécutant sur un PC. Cela est possible en veillant à ce que les composants sous-jacents présentent leurs fonctionnalités via des standards ouverts pour un accès et une utilisation par les autres couches de présentation. Cette interface de service présente les fonctions de la couche métier dans un format que les couches de présentation sont capables de comprendre et d’exploiter.

Un point de terminaison est un ensemble de composantes qui permettent d’interroger un service web pour le consommer. Ces composantes sont les suivantes :

**Address** : l’adresse à laquelle le service se trouve,

**Binding** : le protocole qui permet de communiquer avec le service,

**Contract** : les fonctionnalités proposées par le service.

Les points de terminaison des services disponibles dans une SOA utilisent des standards tels que :

* **XML**: ou eXtensible Markup Language, qui fournit une interopérabilité des données indépendante du fournisseur entre les systèmes.
* **SOAP**: ou Simple Object Access Protocol, qui fournit la syntaxe pour l’accès aux services.
* **WSDL**: ou services Web Description Language, qui établit de manière effective le contrat des services Web, spécifiant quelles entrées sont attendues et quelles sorties seront fournies.

L’architecture orientée service (SOA) repose sur le principe de :

* fourniture d’interfaces de service,
* développement d’un riche ensemble de services,
* développement d’applications qui composent et regroupent ces services afin de fournir et de prendre en charge des systèmes et des processus de gestion plus complexes.

Le modèle SOA repose sur des standards qui permettent aux services de fonctionner sur de nombreuses plates-formes différentes et garantit qu’ils puissent toujours inter-opérer et communiquer les uns avec les autres. Les services peuvent être fournis aux interfaces utilisateur qui s’exécutent sur n’importe quelle plate-forme ou appareil capable de communiquer avec les interfaces de service sous-jacentes. La seule condition requise au niveau de l’application consommatrice est sa capacité à se conformer aux standards concernés et à les utiliser.

**.NET web service**

**.NET web service**

Data

Data

SQL

XML

file

system

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Business

Business

Data Access

Data Access

**J2EE web service**

**J2EE web service**

Data

Data

SQL

XML

file

system

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Business

Business

Data Access

Data Access

XML, SOAP,

WSDL etc

Windows

application

.NET

application

Java

application

PDA

Application

XML, SOAP,

WSDL etc

XML, SOAP,

WSDL etc

**.NET web service**

**.NET web service**

Data

Data

SQL

XML

file

system

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Business

Business

Data Access

Data Access

**.NET web service**

**Service Web .NET**

Data

Données

SQL

XML

système de

fichiers

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Interface de service

-

**sécurité commune**

-

**réprésentation commune**

-

**métalangage commun**

-

**gestion commune des erreurs**

Business

Gestion

Data Access

Accès aux données

**J2EE web service**

**J2EE web service**

Data

Data

SQL

XML

file

system

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Business

Business

Data Access

Data Access

**J2EE web service**

**Service Web J2EE**

Data

Données

SQL

XML

système de

fichiers

Service Interface

-

**common security**

-

**common representation**

-

**common meta**

**-**

**language**

-

**common error handling**

Interface de service

-

**sécurité commune**

-

**représentation commune**

-

**métalangage commun**

-

**gestion commune des erreurs**

Business

Gestion

Data Access

Accès aux données

XML, SOAP,

WSDL, etc.

Windows

Application

.NET

Application

Java

Application

Application

PDA

XML, SOAP,

WSDL, etc.

XML, SOAP,

WSDL, etc.

*Intégration interfournisseur via la SOA*

Dans le cadre de cette approche, un service Web .NET peut par exemple être utilisé par des applications clientes riches Windows, des Smartphone, un PDA, un navigateur Web ou une application Java. De même, un service Web J2EE peut être utilisé par exactement les mêmes clients - ou, en fait ces services Web sur des plates-formes différentes peuvent communiquer directement. Pour les entreprises, c’est une fonctionnalité absolument essentielle : elle permet aux services construits sur des plates-formes et technologies différentes de communiquer et d’inter-opérer à l’intérieur d’un département, entre les départements, et entre les départements et les utilisateurs externes tels que les partenaires, les fournisseurs et les clients.

1. Que sont les services web ?

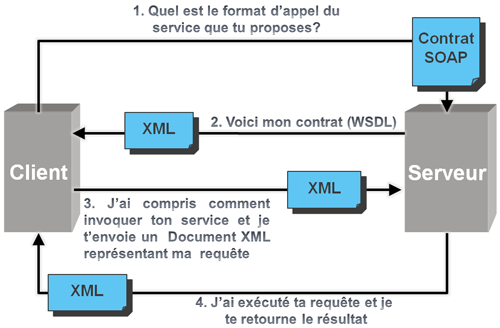
Un service Web est un ensemble de **protocoles** et de **normes** informatiques utilisé pour échanger des données entre des applications.

Les services Web permettent d’échanger des données à travers des réseaux informatiques comme Internet. Cette interopérabilité est liée à l'utilisation de **normes** ouvertes regroupées au sein du terme générique de SOA (Service Oriented Architecture ou Architecture orientée services). L'**OSI**, le World Wide Web Consortium (**W3C**) et l’**OASIS** sont les comités de coordination chargés de l'architecture et de la standardisation des services Web. Pour améliorer l'**interopérabilité** entre les réalisations de services Web, l'organisation **WS-I** ([Web Services Interoperability](http://www.ws-i.org/)) a développé une série de profils d'interopérabilité pour faire évoluer les futures normes impliquées.

Les Services Web reposent sur des **standards** : [HTTP](http://fr.wikipedia.org/wiki/HTTP), [XML](http://fr.wikipedia.org/wiki/XML), [XSD](http://smrtx.com/RS/xsd.htm), [SOAP](http://fr.wikipedia.org/wiki/SOAP), [WSDL](http://fr.wikipedia.org/wiki/WSDL), [WS-\*](http://msdn2.microsoft.com/en-us/library/ms996535.aspx), …

Les services web sont une solution d’implémentation pour bâtir une architecture orientée services. De plus, ils offrent une **approche budgétaire prudente** car ils permettent une **mise en œuvre incrémentale** (bottom-up) et car le caractère standard des Services Web n’impose **pas d’investissement logiciel colossal** pour démarrer.

Quelques Services Web proposés au sein de la plateforme Microsoft sont présentés i[ci](http://msdn2.microsoft.com/en-us/library/aa480728.aspx).



*Principes de fonctionnement des Services Web*

1. Comment assurer l’interopérabilité ?
2. Qu'est-ce que l’interopérabilité ?

L’*interopérabilité* vise à permettre aux logiciels informatiques de travailler ensemble correctement et de manière efficace, quelle que soit la technologie ou l’application utilisée, et quel que soit le fabricant ou éditeur qui a fourni le système sous-jacent. Cela peut être une utilisation en interne par l’entreprise (intranet), ou une offre de services en ligne via Internet destinés à ses partenaires et clients.

Pour la suite de cet exposé, nous entendrons par *interopérabilité* la capacité pour des produits et logiciels informatiques disparates à échanger et à utiliser des données et des informations en vue de fonctionner conjointement.

1. Respect des standards

Comme vous avez pu le lire dans la partie précédente sur la présentation des services web, tous les acteurs doivent respecter les mêmes standards pour être interopérables.

Les Web services reposent sur quatre standards technologiques principaux : HTTP, XML, WSDL et SOAP.

* + 1. http

**HTTP**est le protocole déjà mis en œuvre pour le transport des formulaires sur le web (pages HTML). On peut aussi utiliser d’autres **protocoles de transport** pour échanger les messages exposés par les Services Web. Dans le cadre de l’interopérabilité, c’est HTTP qui est généralement utilisé.

* + 1. XML

**XML** est utilisé pour **décrire les messages échangés entre les Services Web**. XML assure une représentation en mode « texte » des données ce qui permet une intégration standard par les applicatifs et une consultation « humaine » à partir d’un simple éditeur de texte de type Notepad.

En complément d’XML, XSD ou encore « XML *Schema* » est mis en œuvre pour typer les documents ; il s’agit d’une standardisation, sous la forme XML, des anciennes DTD qui décrivent la structure des documents. Le *Schema* XML contient les **types de base** pour les données manipulés dans un applicatif : chaîne de caractères, entier, date… mais aussi des **types complexes** et leurs contraintes comme les plages de valeurs. XML *Schema* est la prolongation du XML « documentaire » pour la représentation et la manipulation de données structurées et typées.

Enfin, les données encapsulées dans les flux XML peuvent être cryptées afin de sécuriser les échanges.

* + 1. WSDL

**WSDL** (Web Service Description Langage) est un document XML qui utilise notamment XML *Schema* pour **décrire le Web service**. Il s’agit de sa carte d’identité au sens d’un IDL (Interface Description Language) traditionnel en DCOM, CORBA... mais exprimé de façon universelle en XML.

La description WSDL regroupe **nom du service**, le **nom de ses opérations (traitements) avec les paramètres d’entrée et de réponse** organisés sous la forme de messages. L’applicatif serveur fournit à l’applicatif client le fichier WSDL, ce qui est nécessaire et suffisant pour assurer l’intégration technique de l’appel du service. Le WSDL contient aussi des **données détaillées sur le protocole de communication utilisé pour le transport** (presque toujours HTTP mais aussi des variantes comme par exemple SMTP), des **informations sur l’encodage technique des données et l’adresse sur le réseau du Web service** sous la forme d’une URL. Le WSDL ne contient rien d’autres et évidement pas de code applicatif. Puisque le WSDL est un document XML, il est représenté en format texte et peut être consulté à l’aide un éditeur de texte.

* + 1. SOAP

Simple Object Access Protocol (SOAP) est un protocole de type RPC orienté objet bâti sur XML.

**Il permet la transmission de messages entre objets distants**, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur une autre machine. Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.

Le protocole SOAP comprend deux aspects :

* Une enveloppe, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement,
* Un modèle de données, définissant le format du message, c'est-à-dire les informations à transmettre.



*Structure d’un message SOAP*

SOAP a été initialement défini par Microsoft et IBM, mais est devenu une référence depuis une recommandation du W3C pour les Web Services.

Source : [Wikipedia.fr](http://fr.wikipedia.org/wiki/Simple_object_access_protocol)  
Voir aussi : ["Qu'est-ce que SOAP?"](http://www.soapuser.com/fr/basics1.html)

**Les différences entre SOAP 1.1 et SOAP 1.2 :**

Par rapport à [SOAP 1.1](http://www.w3.org/TR/2000/NOTE-SOAP-20000508/), [SOAP 1.2](http://www.w3.org/2002/07/soap-translation/soap12-part1.html) présente un certain nombre de **modifications dans la syntaxe** et fournit de la **sémantique supplémentaire**. Il lève selon le W3C les ambiguïtés créées par des interprétations divergentes de la spécification SOAP 1.1.

De plus, SOAP 1.2 permet l’usage du **mode de communication GET**, ce qui est intéressant pour les stratégies de gestion de cache et de sécurité (une opération GET n’est censée faire qu’une consultation sur le serveur), alors que la version précédente de SOAP n’autorisait que le mode POST.

Par ailleurs, SOAP 1.2 est **plus clair** que SOAP 1.1 car il offre des modèles plus précis et extensibles, ce qui permet une meilleure interopérabilité. SOAP 1.2 permet également une **meilleure intégration web** avec le respect des standards XML et des architectures web. Enfin, SOAP 1.2 est **plus souple** que SOAP 1.1, puisque le Framework permet l'utilisation de protocoles indépendants, et **plus rapide** puisqu’il utilise Infoset qui est basé sur XML, ce qui permet une optimisation des performances.

Si vous souhaitez en savoir plus concernant [les différences entre SOAP 1.1 et SOAP 1.2](http://www.w3.org/2002/07/soap-translation/soap12-part0.html#L4697), vous pouvez vous reporter au site du W3C.

* + 1. Les différents types d'encodage des messages

**Types de messages ( Document / RPC ):**

La spécification SOAP précise deux types de formats possibles pour le transport des données entre applications : les messages de type **RPC** et ceux de type **Document**.

Avec les **messages de type RPC**: les données encodées au format XML sont stockées dans le corps du message SOAP avant d’être envoyées à la machine destinataire. Cette dernière traduit alors les données XML dans le format de l’application cible.

Il peut aussi arriver que les données que l’on souhaite transporter soient déjà au format XML ou que l’on souhaite se passer du formalisme du modèle RPC : on parle alors de **messages de type Document** (aucune règle d’encodage ou de formatage ne s’applique alors en dehors de la conformité des données à un schéma XML).

Les types « **Document** » et « **RPC** » se réfèrent exclusivement à la façon dont l'élément est mis en forme dans le message SOAP (cela n’a pas d’impact sur le reste du message). Dans le cas du modèle "Document", il s’agit d’un message directement au format XML. Dans le cas du modèle "RPC", les arguments d'entrées-sorties sont des fonctions encodées au format XML.

**Styles d’encodage ( Encoded / Literal ) :**

Il existe deux styles différents de mise en forme des paramètres au sein d’un message SOAP : **Encoded** et **Literal**.

**Encoded** fait référence à la mise en forme des paramètres à l’aide des règles de codage définies par l’attribut ***encodingStyle***, alors que **Literal** implique que les données soient conformes à un schéma XML (présent dans le fichier WSDL) sur l’attribut ***type*** ou ***element*** des données.

**Point sur les combinaisons et l’interopérabilité**

Les différentes combinaisons possibles sont :

* + RPC / Encoded
  + RPC / Literal
  + Document / Encoded
  + Document / Literal

Sachant que l'association entre le style **Encoded** et le modèle **Document** n'est pas implémenté, et que la norme **WS-I Basic Profile 1.0** a choisi **Literal** comme standard au détriment du style **Encoded, v**ous aurez de fortes chances de rencontrer les formats SOAP suivants :

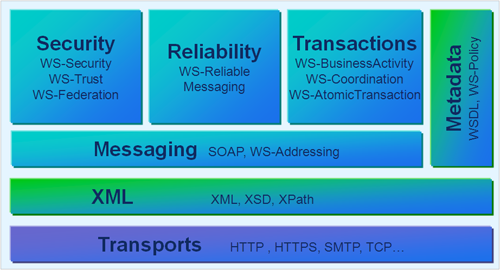
* + **RPC / Literal**
  + **Document / Literal**

Les sources mis à disposition en complément de ce document proposent des exemples de messages RPC/Literal et Document/Literal.

Pour plus d’informations sur le sujet, se reporter à l’[article de Lionel Tricon](http://lionel.tricon.free.fr/Articles/wsdl/article_wsdl.pdf)

* + 1. WS-\* (STAR)

WS-\* signifie WS-STAR pour **Secured**, **Transacted**, **Asynchronous** & **Reliable**. Ces protocols permettent d’implémenter des scenarios dits avancés entre Services Web.



*Architecture et standards des services web*

Pour des informations plus complètes concernant les protocoles WS-\*, nous vous invitons à consulter la [documentation MSDN](http://msdn2.microsoft.com/en-us/library/ms996535.aspx)

C'est sur cet ensemble de protocoles sur lesquels reposent sur les profils d'interopérabilité proposés par le WS-I.

1. L’organisme WS-I et les outils de vérification

**Web Services Interoperability Organization** est un **organisme fondé en 2002 par Microsoft, IBM, BEA et Intel** dans le but d'aboutir à **des spécifications communes à toutes les implémentations des Services Web.**

Il compte **aujourd'hui plus de 130 membres** et est, de fait, reconnu comme étant l'organisme qui régit et décide des normes associées aux Services Web.

Son but premier est **d'assurer l'interopérabilité des implémentations**. Pour ce faire, **il définit plusieurs profils** que chaque implémentation doit suivre pour pouvoir être compatible avec une autre implémentation respectant le même profil.

Ces profils correspondent en fait à un **ensemble de spécifications à utiliser**, et indiquent la manière d'interpréter ces spécifications.

1. Les différents profils

* **Basic Profile** (v1.2 depuis Mars 2007) :est le profil de base que toute implémentation se doit de respecter si elle veut être compatibles avec les autres. Les trois autres profils en dépendent.

Historique : v1.0 en Avril 2004 et v1.1 en Avril 2006

* **Attachments Profile** (v1.0 depuis Avril 2006) :est un profil qui indique les spécifications à utiliser pour transmettre des données volumineuses ou binaires entre des composants.
* **Simple SOAP Binding Profile** (SSBP depuis v1.0 Août 2004) : est un profil qui décrit la façon de transporter les messages au dessus du protocole HTTP.
* **Basic Security Profile** (v1.0 Mars 2007) : est un profile qui définit les mécanismes de sécurité à mettre en œuvre dans les communications entre les Services Web.

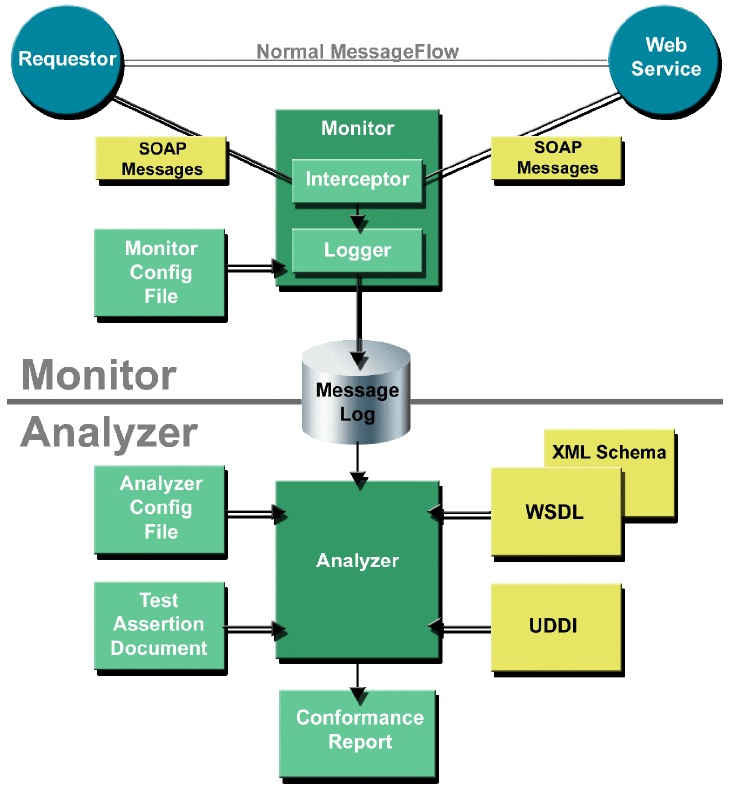
Le Basic Profile se compose de **4 modules obligatoires**, et d'un **module optionnel** :

* **Transport :** il s'agit du protocole de transport HTTP qui est utilisé pour les communications
* **Représentation :** il s'agit du protocole SOAP 1.1 qui permet de décrire les données qui sont transmises
* **Description :** il s'agit du langage de description WSDL v1.0 qui permet à un fournisseur de décrire les opérations qu’il expose.
* **Découverte :** il s'agit du protocole UDDI v2 qui permet de partager des définitions de services
* **Sécurité (optionnel) :** il s'agit du protocole HTTPS qui permet d'assurer les sécurité dans les communications et les accès.

*Les modules qui composent le Basic Profile*

* + 1. Les outils de vérification

On peut trouver sur le [site du WS-I](http://ws-i.org/deliverables/workinggroup.aspx?wg=testingtools) deux outils. Le premier, Monitor, permet de récupérer les messages SOAP échangés lors des appels entre Services Web. Le second, Analyser, génère un fichier de rapport de la conformité en regard des spécifications Basic Profile 1.1 qui lui sont transmises.



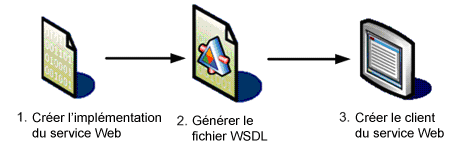
*Outils de vérification proposés par le WS-I*

1. Stratégies d’implémentation : Code First versus Contract First
2. Implémentation First

La technique de création de services Web la plus courante, la plus avérée et la mieux prise en charge par les outils de développement consiste à « inférer » une interface de service Web à partir d’une implémentation existante. Le développeur rédige le code suivant :

public int Ajouter(int x, int y) { return x+y; }

Avec WCF, il est très simple d'exposer ce code en tant que Service Web : il suffit d’ajouter l’attribut [ServiceContract] à la classe et [OperationContract] aux méthodes. Cette technique est souvent appelée « **Implementation First** » (priorité à l’implémentation) ou « **Code First** » (priorité au code) car l’interface du Service Web, décrite de façon formelle dans un document **WSDL (Web Service Description Language)**, est dérivée de l’implémentation.



*Développement d’un service Web par la méthode « Implementation First »*

La technique de développement de services Web appelée « Implementation First » consiste à écrire d’abord le code du service Web (voir étape nº 1 de la figure ci-dessus). Après compilation, l’infrastructure des services Web utilise ce code pour générer de façon dynamique un fichier WSDL (étape nº 2). Lorsque les clients demandent la définition du service Web, ils récupèrent le fichier WSDL généré et créent le **proxy client** à partir de cette définition (étape nº 3). Le proxy est créé avec l’outil **svcutil** pour un **client en WCF** et avec **wdsl2java d’axis** pour un **client java**.

Par exemple, dans WCF, le fichier WSDL peut être généré de façon dynamique à partir d’une implémentation avec URL comme celle-ci :

http://hostname:port/MyService?WSDL

Lorsque le runtime de .NET détecte le paramètre WSDL dans la requête, il effectue une réflexion sur le code doté de l’attribut [OperationContract] pour générer de façon dynamique une opération de description du service sous-jacent dans le fichier WSDL.

Cette technique d’implémentation est très simple et fonctionne très bien, mais elle génère quelques problèmes, notamment en cas de Services Web utilisés pour connecter des systèmes hétérogènes.

Par exemple, si l’on commence par l’implémentation, il n'est pas possible d’inclure dans le service Web des types spécifiques à une plate-forme. Les DataSet .NET et les Vecteurs en Java sont spécifiques à ces plate-formes et sont difficiles à représenter sur d’autres plates-formes. En effet, il n’existe pas de mappage unique et bien défini entre ces types spécifiques et les possibilités proposés par le W3C au niveau des schémas XSD.

Bref, ce n’est pas parce qu’un consommation .NET est capable de reconnaître dans un bloc XML un certain type de données, qu’un consommation de Service Web écrit en Java peut faire la même chose. Nous sommes donc confrontés à des problèmes d’interopérabilité.

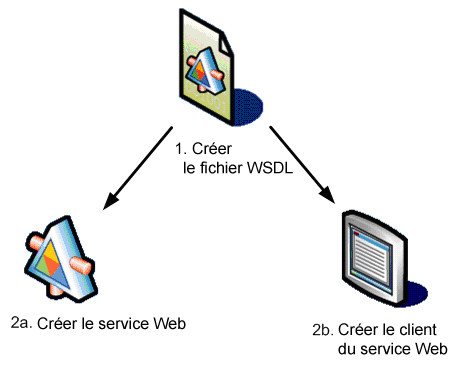
* + 1. WSDL First

Le schéma XML standard W3C définit des types de données intégrés dont, entre autres, les chaînes (string), les entiers de différentes tailles (integer), les variables booléennes (boolean), les variables représentées en virgule flottante (float) en simple et double précision, ou les données de date et d’heure (DateTime). Chaque plate-forme applicative prend également en charge son propre jeu de types de données. L’intersection entre ces jeux de types de données définit les types qui offrent la meilleure interopérabilité entre plates-formes. Si l’on commence par des types appartenant au schéma XML, il est facile de les mapper vers les types spécifiques à la plate-forme, mais si l’on commence par ces derniers, le mappage inverse n’est pas forcément possible. Par exemple le mappage des types integer, string, Boolean et float du schéma XML vers les types de données correspondants dans .NET ou Java est automatique. En revanche, les types de vecteurs (Vector) ou de tables de hachage (Hashtable) sont natifs des différentes plates-formes et ne font donc pas partie des types officiels du schéma XML. Pour plus d’informations sur les types de données pris en charge, consultez les [spécifications](http://www.w3.org/TR/xmlschema-2/) des types de données du schéma XML.

Les runtimes des stacks de Services Web sont chargés d’effectuer un mappage entre les primitives du schéma XML et les primitives spécifiques à la plate-forme. Ainsi, une chaîne du schéma XML sera mappée vers le type System.String dans .NET et vers le type java.lang.String dans Java. Avec les primitives du schéma XML et les structures et tableaux générés à partir de ces primitives, il est possible de créer des types de données plus complexes, décrits dans le schéma XML, qui pourront être mappés de façon très fidèle d’une plate-forme à l’autre. Ces types de données peuvent alors figurer dans les documents WSDL à utiliser dans le service Web.

Cela représente l’essence de la méthode appelée « **WSDL first »** (priorité au WSDL) : si l’on utilise les types du schéma XML pour définir les types de données utilisés dans le service Web, l’on a plus de chance de pouvoir mapper ces types de données entre plates-formes.

Cette technique qui consiste à créer d’abord le fichier WSDL est aussi appelée parfois « **Schema First** » (priorité au schéma) ou « **Contract First** » .



*Développement d’un service Web par la méthode « WSDL First »*

Le problème avec la technique consistant à créer d’abord le fichier WSDL, c’est que les outils produits aujourd’hui ne favorisent pas cette pratique. Elle reste néanmoins faisable, mais difficile à implémenter. Visual Studio comporte un éditeur de schéma et un éditeur XML mais pas d’éditeur WSDL. Eclipse ne propose pas non plus d’éditeur WSDL. Heureusement, ces deux environnements offrent une fonctionnalité permettant de générer le squelette de code du service Web en plus du code du proxy client à partir d'un fichier WSDL.

Vous pouvez utiliser l’outil de votre choix pour créer votre propre fichier WSDL, y compris VI ou le Bloc-notes. Plutôt que d’éditer directement le texte, vous pouvez aussi utiliser pour plus de commodité des outils spécialisés comme XmlSpy d’Altova, dotés de concepteurs WSDL. Toutefois, cette solution n’est pas forcément appropriée dans la mesure où de nombreux développeurs restent incapables de « penser en WSDL ».

Les pratiques courantes en terme d’interopérabilité consistent à travailler comme suit :

1. Concevoir rapidement un prototype d’interface de service Web en utilisant la technique « Code First »
2. Utiliser les fonctions de génération dynamique de fichier WSDL de WCF ou de JAX-WS pour créer un template WSDL.
3. Reprendre le contrat WSDL afin de peaufiner l’interface à sa guise. Ce processus s’exécute de façon itérative jusqu’à obtention du fichier WSDL final.

Comme l’illustre la figure ci-dessus, la création de services Web à partir de la méthode « WSDL First » comporte trois étapes majeures :

1. Créer le fichier WSDL.
2. Implémenter le document WSDL.
   1. Créer le producteur du Service Web (serveur)
   2. Créer le consommateur du service Web (client)

Vous pouvez effectuer les sous-étapes A et B de l’étape 2 dans l’ordre de votre choix. Comme elles dépendent toutes deux du document WSDL. Ce qui importe, c’est de créer le document WSDL avant d’entreprendre les sous-étapes A ou B.

La méthode WSDL First offre l’avantage de n’utiliser que des types définis par le W3C ce qui permet d’avoir une interopérabilité totale car aucun type propre à un langage n’est transmis. De plus cette méthode permet au développeur du service web et à celui de l’application cliente qui va consommer le service de coder leurs applications en parallèle sans que le développeur de l’application cliente soit obligé d’attendre que le service web soit implémenté. Le seul inconvénient de cette méthode est qu’actuellement les outils de manipulation des fichiers WSDL ne permettent pas de créer simplement des contrats WSDL interopérables et facilement compréhensibles à la fois par les environnements Java et .Net. Pour approfondir ces notions vous pouvez lire la suite dans cet article : [Interopérabilité des services Web de type « Contract First » entre Microsoft .NET et IBM WebSphere.](http://download.microsoft.com/download/5/8/C/58C2CE40-DE59-4725-A56B-335DC2BAFEA0/interoperabilite_services_web.doc)

1. Les implémentation des services web
2. Que propose Microsoft ?

Les différentes technologies Microsoft qui permettent d’offrir des Services Web sont :

* + **ASMX** **2.0** : Cette technologie Services Web vient avec le framework ASP.NET. Au lieu de servir des pages (extension ASPX), le serveur IIS délivre des services (extension ASMX). Ce framework supporte les profile 1.0 et 1.1, ainsi il est suffisant pour des services basiques. Par contre, si vous voulez bénéficier des spécifications évoluées des Services Web (protocoles WS-\*) ou bien si vous voulez maîtriser les échanges de messages (framework de sérialisation), ou encore si vous voulez proposer des services dans une application non Web (en dehors de IIS, comme une batch, un service NT ou encore une application Client Desktop), vous vous orienterez vers WCF.
  + **WSE 3.0** (Web Services Enhancement) : Ce framework se télécharge de façon complémentaire au framework 1.1 (version WSE 2.0) ou au framework 2.0 (version WSE 3.0). L’objectif de WSE est de proposer une implémentation des spécifications évoluées des Services Web (protocoles WS-\*). Ce framework est déprécié au profit de la technologie WCF proposée avec le framework .Net 3.0
* **WCF (**Windows Communication Foundation)est la technologie de Services Web proposée à partir du framework .Net 3.0. Elle facilite la connexion des systèmes et des applications au sein de l’entreprise et entre des sites répartis géographiquement. WCF permet de configurer le protocole de dialogue entre 2 applications (Message Queues, Dialogue binaire entre applications .Net, Services Web pour assurer l’interopérabilité). WCF implémente le profil Basic promu par le consortium WS-I mais aussi les protocoles évolués WS-\* Grâce à ce framework, les développeurs d’applications distribuées

peuvent se concentrer sur la logique métier et laisser le runtime WCF gérer la plomberie sous-jacente.

Vous trouverez à [cette adresse](http://msdn.microsoft.com/msdnmag/issues/06/01/ServiceStation/) un comparatif de ces différentes technologies.

1. Que propose les environnements Java

Les Services Web en Java reposent sur des **spécifications**. Les JSR suivantes sont notamment utilisées :

* JSR224 : JAX-WS 2.0 (mai 2006) (Nouvelle pile pour les services web)
  + La spécifications des Services Web basiques
  + JAX-WS 2.1 (février 2007 – maintenance release)
  + JSR101 : JAX-RPC (2003 – deprecated)
  + JSR109,921 : Enterprise Web Services (nov 2002 - JSR101+…)
  + JSR261 : JAX-WSA (Web Service Adressing – deprecated)
* JSR181 : Web Services Metadata for Java (juin 2005)
  + Annotations @WebService,@WebMethod, part of JSR224
* JSR222 : JAXB 2.0 (mai 2006)
  + JSR31 : JAXB (mars 2003)
* JSR311 : JAX-RS (future)
* Concerne le protocole d’échanges REST

**JAX-WS** (Java Extension for Web Service) est depuis 2002 l’héritier de Java WSDP (Web Services Developer Pack) et de JAX-RPC 1.1 qui s’occupe du mapping entre WSDL et Java ainsi que les API pour le traitement des flux SOAP. JAX-WS 2.1 date de Février 2007, il délègue à JAXB 2.0 les règles de mapping qui concernent les types XML Schema vers Java. Il supporte les versions 1.1 et 1.2 de SOAP, WSDL 1.1 et les profils Basic 1.0/1.1. JAX-WS est intégré à Java EE 5 et Java SE 6. Le support des protocoles évolués tels que WS-SecurityPolicy, WS-Trust, WS-SecureConversation, WS-ReliableMessaging et WS-CoordinationWS-AtomicTransaction doit être assuré par un framework complémentaire. Les implémentations [WSIT](https://wsit.dev.java.net/) de SUN mais aussi AXIS d’Apache proposent de telles implémentations.

Il existe différentes implémentations de JAX-WS. Tout d’abord il y a JAX-WS RI qui est l’implémentation de référence réalisée par SUN qui est utilisée par les serveurs d’applications GlassFish et BEA WebLogic Server 10.0. IBM et JBoss ont réalisé leurs propres implémentations de JAX-WS 2.0

Pour en savoir plus sur la roadmap de JAX-WS, consulter ce [lien](https://jax-ws.dev.java.net/roadmap/jaxws-roadmap.html)

1. Les serveurs d’applications et les modes d’encodage

Nous avons choisi deux versions d’Axis afin de supporter les **différentes versions de SOAP** (1.1 et 1.2) ainsi que les **différents modes d’encodages (RPC/Encoded , Document / Literal et Document / Literal Wrapped)**.

Tout d'abord, il faut expliquer ce qu’est **Axis**. C’est un projet de **Apache** Software Foundation, proposant un package Java qui 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).
    - Des outils pour déployer, tester et monitorer des Services Web.

Par rapport à Axis 1.4, Axis 2.0 est une réécriture complète qui a pour objectif d'être plus efficace, plus modulaire et plus orientée XML. Des modules sont en cours de développement en ce qui concerne la sécurité, les transactions...

En effet, **Axis 1.4** permet d’offrir un support de **SOAP 1.1** mais **pas de SOAP 1.2** (il n’est pas 100% compatible avec SOAP 1.2). Avec Axis 1.4, lors d’un **déploiement** d’un service web avec la méthode **JWS** (Java Web Services), le **format d’encodage** utilisé est **RPC/Encoded** (aussi appelé SOAP/Encoded ou Version 5/Encoded).

Avec **Axis 2.1.1.1**, lors d’un déploiement d’un service web avec la méthode **AAR** (Axis Archive), le service web est proposé en **SOAP 1.1** et en **SOAP 1.2** et le **format d’encodage** utilisé est **Document / Literal**.

**GlassFish v2** est un serveur d’application open source basé sur les implémentations de référence **JAX-WS RI 2.1** (Java Extension for Web Service) et **WSIT** (Implémentation des protocoles WS-\* proposée par SUN). Le Service Web peut être en **SOAP 1.1** (par défaut) ou en **SOAP 1.2** et le **format d’encodage** utilisé par défaut est **Document / Literal**.

WCF pour Windows Communication Foundation est l'une de quatre nouvelles briques apportées par .NET Framework 3.0. Cette brique unifie le développement des applications distribuées. Avec **WCF**, on peut **exposer un service web** en **SOAP 1.1** et **en SOAP 1.2** et le format d’encodage est **Document / Literal Wrapped**.

1. La gestion des exceptions

Il est recommandé de travailler avec des exceptions déclarées, plutôt que transmettre des exception non déclarées.

On considère qu’une exception est déclarée quand on utilise dans le code source du serveur d’application le tag :

- [FaultContract(typeof(MySpecificException))] sur une méthode .Net,

- @FaultAction(className=MySpecificException.class) sur une méthode JAX-WS.

*Remarque :* Axis 1.4 ne permet pas de déclarer une exception dans le contrat WSDL. Et Axis 2.1.1.1 déclare les exceptions par défaut dans le contrat WSDL, il n’y a pas besoin de mettre un tag au dessus du code source des méthodes.

L’ensemble des exceptions déclarées dans le code source du service web sont spécifiées dans le contrat WSDL par un élément du type : <wsdl:fault name="…" message="…"/>.

Dans l’exemple ci-dessous, on représente une exception nommée « ValeurInsuffisante » dont le champ « Detail » est formé par le message du même nom.

<definitions>

<message name="unMessage"/>

**<message name="ValeurInsuffisante">**

**<part name="solde" type="xsd:int"/>**

**</message>**

<portType name="Banque">

<operation name="virementBanque">

<input message="tns:unMessage"/>

<output message="tns:unMessage"/>

**<fault name="fault" message="tns:ValeurInsuffisante"/>**

</operation>

</portType>

</definitions>

*Exemple d’une définition de Soap Fault dans le WSDL*

Comme l’exception est déclarée dans le contrat WSDL, le client qui va consommer ce service sait par avance qu’il peut rencontrer ce type d’erreur. Ceci permet donc au développeur de l’application consommatrice de bien prévoir tous les cas d’erreur à traiter avec des try..catch. C’est donc pour cela qu’il est recommandé de travailler avec des exceptions déclarées, plutôt que transmettre des exception non déclarées.

Parmi les **exceptions déclarées** on distingue différent cas :

- **Les exceptions basiques** de type *FaultException* en .Net et les *SOAPFaultException* en Java.

- **Les exceptions propres** à l’un des deux langages *Division par zéro, Pointeur nul.*

*-* **Les exceptions plus riches** de type *FaultException<MySpecificException>* en .Net et *SOAPFaultException<MySpecificException>* en Java. Cela permet de créer sa propre classe d’erreurs qui hérite de la classe FaultCode afin de gérer ses propres codes d’erreurs.

Qu’il s’agisse d’erreurs de type fonctionnel (logique applicative) ou technique, l’exception levée au niveau du langage de programmation (java, c#) est traduite dans le message SOAP à l’aide d’une SOAP Fault.

Une SOAP fault se compose de quatre éléments :

- **Faultcode** *(obligatoire)* **:** Code d’erreur utilisé par l’applicatif.

- **Faultstring** *(obligatoire)* **:** Explication compréhensible par un humain.

- **Failactor** *(optionnel)* **:** Erreur qui apparaît lors du cheminement du message (firewall, proxy).

**-** **Detail** *(optionnel)* **:** Détail de l’erreur sous la forme XML. C’est dans cet élément que les libellés et le code erreur sont décrits.

Pour gérer ses propres codes d’erreurs, il faut définir des exceptions étendues héritant de la classe faultexception. La zone détail sert de conteneur aux informations de l’exception étendue.

Dans l’exemple ci-dessous, on représente un message SOAP qui retourne exception nommée «SOAP-ENV:Server » dont le champ « Detail » contient les données de description de l’erreur.

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

**<SOAP-ENV:Fault>**

**<faultcode>SOAP-ENV:Server</faultcode>**

**<faultstring>Erreur sur le Serveur</faultstring>**

**<detail>**

**<solde> 12003 </solde>**

**</detail>**

**</SOAP-ENV:Fault>**

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

*Exemple du flux SOAP comprenant une « Fault »*

1. Environnement technique
   1. Configuration logicielle requise

Les exemples présentés ci-après ont été réalisés avec les logiciels suivants :

* + Microsoft Windows XP Professionnel SP2
  + Microsoft Internet Information Services 5.1
  + Visual Studio 2005 SP1
  + Kit de développement (SDK) de Microsoft .NET Framework, version 3.0
  + Les extensions de WCF pour Visual Studio 2005
  + JDK de Sun Java, version 1.6
  + Eclipse 3.2.2
  + Tomcat 5.5.20
  + Axis 1.4 & Axis 2.1.1.1
  + NetBeans 5.5.1
  + GlassFish V2 b33 (JAX-WS RI 2.1)
  1. Les outils pour comprendre les services web

1. Comprendre les échanges SOA

Pour comprendre les échanges SOA, il faut savoir ce qu'il se passe coté client et côté serveur, lorsque l'on met en œuvre un Service Web. Pour ce faire, on utilise un outil qui trace les messages échangés, souvent au niveau de la stack IP ou bien pour le protocole concerné (HTTP, TCP, ...)

Voici quelques outils gratuits pour commencer vos investigations :

* + Amberpoint : <http://www.amberpoint.com/solutions/express.shtml> (version Express)
  + Fiddler  : <http://www.fiddlertool.com/fiddler/>
  + WS-I Monitor: <http://ws-i.org/>
  + Network Monitor 3 : <http://www.microsoft.com/downloads/details.aspx?familyid=AA8BE06D-4A6A-4B69-B861-2043B665CB53&displaylang=en>
  + SOAPUI  : <http://www.soapui.org/>
  + TcpTrace  : <http://www.pocketsoap.com/tcptrace/>

*Remarque :* Pour utiliser le logiciel Fiddler, il faut que le client utilise le vrai nom de la machine du serveur pour interroger les services web. Il ne faut pas utiliser localhost et 127.0.0.1 ([Informations complémentaires…](http://www.fiddlertool.com/Fiddler/help/hookup.asp)).

1. Comprendre le contrat WSDL

Pour faciliter l’analyse des contrats WSDL, vous pouvez créer des représentations graphique à partir d’outils tels que :

* + XMLSpy de chez Altova : <http://www.altova.com/>
  + IDesign MEX Explorer : <http://www.idesign.net>
  1. Procédure d’installation des environnements

1. Environnement .Net

Tout d'abord, il faut installer **Windows XP SP2** et effectuer toutes les **mises à jour de Windows**. Puis il faut installer **Visual Studio 2005** ([lien](http://www.microsoft.com/france/msdn/vstudio/default.mspx)).



Il faut installer dans l'ordre les patchs et logiciels suivants :

- Le **Service Pack 1 de Visual Studio 2005** :

* Pour la version **Standard, Professional,** ou **Team Editions** :

<http://www.microsoft.com/downloads/details.aspx?displaylang=fr&FamilyID=BB4A75AB-E2D4-4C96-B39D-37BAF6B5B1DC>

* Pour la version **Express** :

<http://www.microsoft.com/downloads/details.aspx?familyid=7B0B0339-613A-46E6-AB4D-080D4D4A8C4E&displaylang=fr>

- Le **Service Pack 2 de SQL Server Express**

<http://www.microsoft.com/downloads/details.aspx?FamilyId=31711d5d-725c-4afa-9d65-e4465cdff1e7&displaylang=fr>

- Le **Framework 3.0**

<http://www.microsoft.com/downloads/details.aspx?FamilyId=10CC340B-F857-4A14-83F5-25634C3BF043&displaylang=en>

- Le **SDK de Windows**

<http://www.microsoft.com/downloads/details.aspx?FamilyId=7614FE22-8A64-4DFB-AA0C-DB53035F40A0&displaylang=en>

- Les **extensions de Visual Studio pour WCF** (qui gèrent aussi WPF)

<http://www.microsoft.com/downloads/details.aspx?FamilyId=F54F5537-CC86-4BF5-AE44-F5A1E805680D&displaylang=en>

Pour des explications plus détaillées, consulter cet [article…](http://blog.sfartz.com/post/2007/04/25/Ma-configuration-logicielle-au-25-avril)

1. Environnement Java

Dans un premier temps, installer un JDK. A partir de la version 6, ceux-ci sont compatibles Vista.

- **JDK de Sun Java, version 1.6** : <http://java.sun.com/javase/downloads/index.jsp>

Lancer l’exécution de *jdk-6-windows-i586.exe* et choisir le chemin d’installation suivant : *C:\Program Files\Java\jdk1.6.0*

- **Eclipse 3.2.2** : <http://www.eclipse.org/downloads/>

Pour installer eclipse, il suffit de décompresser le fichier *eclipse-SDK-3.2.2-win32.zip* dans le dossier *C:\eclipse* ce qui donne le chemin *C:\eclipse\eclipse.exe* pour l'exécutable.

- Module **wtp\_wst\_sdk 1.6** pour **Eclipse** : <http://www.eclipse.org/webtools/main.php>

Pour installer ce module, il faut lancer Eclipse, puis aller dans *Help/Software Updates/Find and Install*. Après il faut choisir *Search for new features to install.* Enfin, lancer la recherche sur le site *Callisto Discovery Site* et installez *wtp\_wst\_sdk*.

- **Tomcat 5.5.20 d’Apache** : <http://tomcat.apache.org/download-55.cgi>

Pendant l’installation il faut préciser les paramètres suivants :

* Port : 8181
* Login : admin
* Mot de passe : admin
* Chemin d’installation : C:\Program Files\Apache Software Foundation\Tomcat 5.5

- **Axis 1.4** : <http://ws.apache.org/axis/>

Pour installer Axis 1.4, il faut décompresser le fichier *axis-bin-1\_4.zip* dans le répertoire C:\Program Files\Apache Software Foundation\axis-1\_4, ce qui donne C:\Program Files\Apache Software Foundation\axis-1\_4\README

Il faut ensuite copier le dossier C:\Program Files\Apache Software Foundation\axis-1\_4\webapps\axis dans le répertoire C:\Program Files\Apache Software Foundation\**Tomcat 5.5**\webapps\axis. Ce qui donne C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\axis\index.jsp.

Il manque des fichiers JAR dans le dossier C:\Program Files\Apache Software Foundation\Tomcat 5.5\common\lib. Il faut télécharger les fichiers suivants et les mettre dans ce répertoire.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fichier Jar** | **Version** | **Fichier zip** | **Adresse** |
| soap.jar | 2.3.1 | soap-bin-2.3.1.zip | <http://apache.mirrors.esat.net/ws/soap/version-2.3.1/> |
| mail.jar | 1.4 | javamail-1\_4.zip | <http://java.sun.com/products/javamail/> |
| activion.jar | 1.1 | jaf-1\_1-fr.zip | http://java.sun.com/products/javabeans/jaf/downloads/ |
| xmlsec.jar | 1.4 | xml-security-bin-1\_4\_0.zip | <http://xml.apache.org/security/> |
| xerces.jar | 1.4.4 | Xerces-J-bin.1.4.4.zip | <http://archive.apache.org/dist/xml/xerces-j/> |

Tutoriel à lire aussi : [Déploiement et Appel aux web services Java avec Axis](http://karamimed.developpez.com/j2ee/tutoriels/axis/)

- **Axis 2.1.1** : <http://ws.apache.org/axis2/>

Pour installer Axis 2.1.1.1, il faut décompresser le fichier *axis2-1.1.1.zip* dans le répertoire C:\Program Files\Apache Software Foundation\axis2-1.1.1, ce qui donne C:\Program Files\Apache Software Foundation\axis2-1.1.1\README.txt

Télécharger le fichier ***axis2.war*** à partir de <http://ws.apache.org/axis2/download/1_1_1/download.cgi>

Puis placer ce fichier dans ce répertoire : C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\.

- **Configuration des variables d’environnement**  :

Vous pouvez placer ces variables en variables système :

**JAVA\_HOME** = C:\Program Files\Java\jdk1.6.0

**CLASSPATH** = C:\Program Files\Java\jdk1.6.0\lib\tools.jar;C:\Program Files\Java\jdk1.6.0

**CATALINA\_HOME** = C:\Program Files\Apache Software Foundation\Tomcat 5.5

**AXIS\_HOME** = C:\Program Files\Apache Software Foundation\axis-1\_4

**AXIS\_LIB** = %AXIS\_HOME%\WEB-INF\lib

**AXIS2\_HOME** = C:\Program Files\Apache Software Foundation\axis2-1.1.1

**Path** **+=** C:\Program Files\Java\jdk1.6.0\bin;%AXIS2\_HOME\bin%

Redémarrer l’ordinateur une fois les variables d’environnement ajoutées ou modifiées.

Voici un site à aller complémentaire afin de vous guider dans votre installation : [Install and configure Apache Tomcat/Axis for web service development on Windows XP SP2](http://www.aniltj.com/blog/2006/02/06/InstallAndConfigureApacheTomcatAxisForWebServiceDevelopmentOnWindowsXPSP2.aspx).

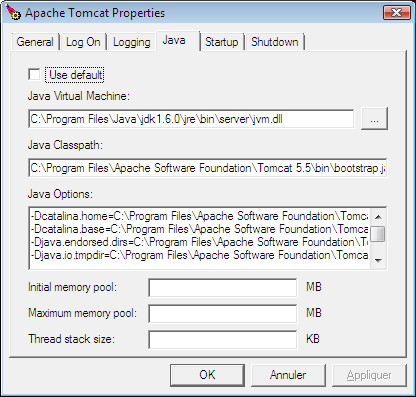
*Remarque* : Si vous mettez des **variables utilisateur,** vous n’avez qu’à **fermer votre session** et à vous reconnecter pour que les variables soient prise en compte. Si vous mettez des **variables système** vous aurez besoin de **redémarrer la machine** pour que les variables soient prise en compte.

- **Lancement de** **TomCat** : Pour lancer Tomcat il faut lancer l’exécutable C:\Program Files\Apache Software Foundation\Tomcat 5.5\bin\tomcat5.exe.

*Remarque* : si vous êtes sous **Vista**, vous devez lancer les exécutables **Tomcat5.exe**, **Eclipse.exe**, **devenv.exe** (Visual Studio 2005) avec les droits administrateurs. Pour cela il suffit de faire un clic droit sur l’exécutable et faire *Exécuter en tant qu’Administrateur*. Cette méthode de lancement vous évitera quelques erreurs dues souvent à une écriture dans le répertoire *Program Files*.

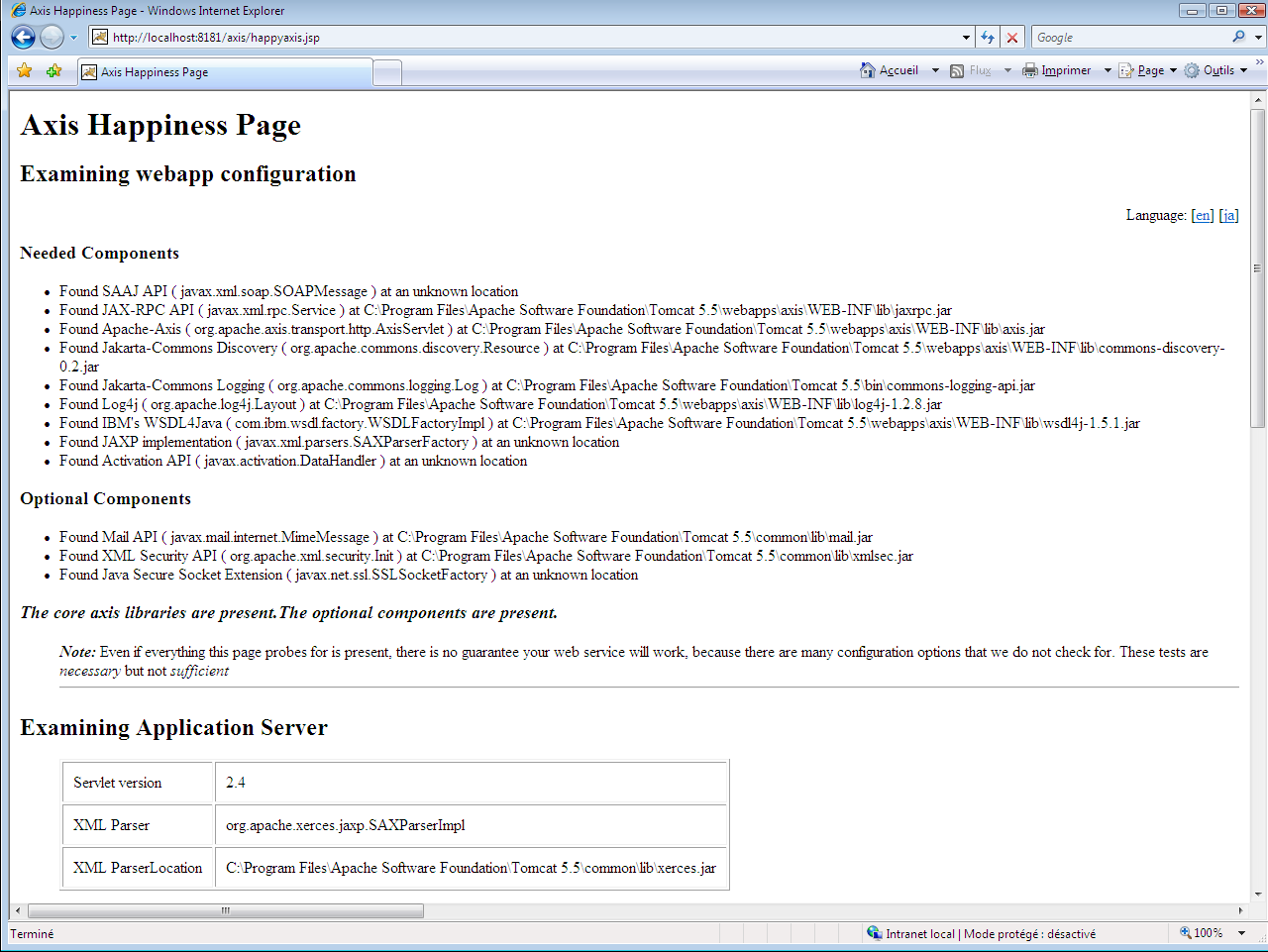
- **Configuration de** **TomCat** : Lancer le moniteur C:\Program Files\Apache Software Foundation\Tomcat 5.5\bin\tomcat5w.exe. Dans l’onglet *Java*, il faut vérifier que *Java Virtual Machine* = C:\Program Files\Java\**jdk1.6.0**\jre\bin\server\jvm.dll. Car il y a de fortes chances que cela soit par défaut : C:\Program Files\Java\jre1.6.0\bin\client\jvm.dll.

Arrêter TomCat et le redémarrer afin qu’il prenne en compte cette modification.



- **Lancement d’Axis 1.4** : Lancer le navigateur et saisir l’URL suivante :

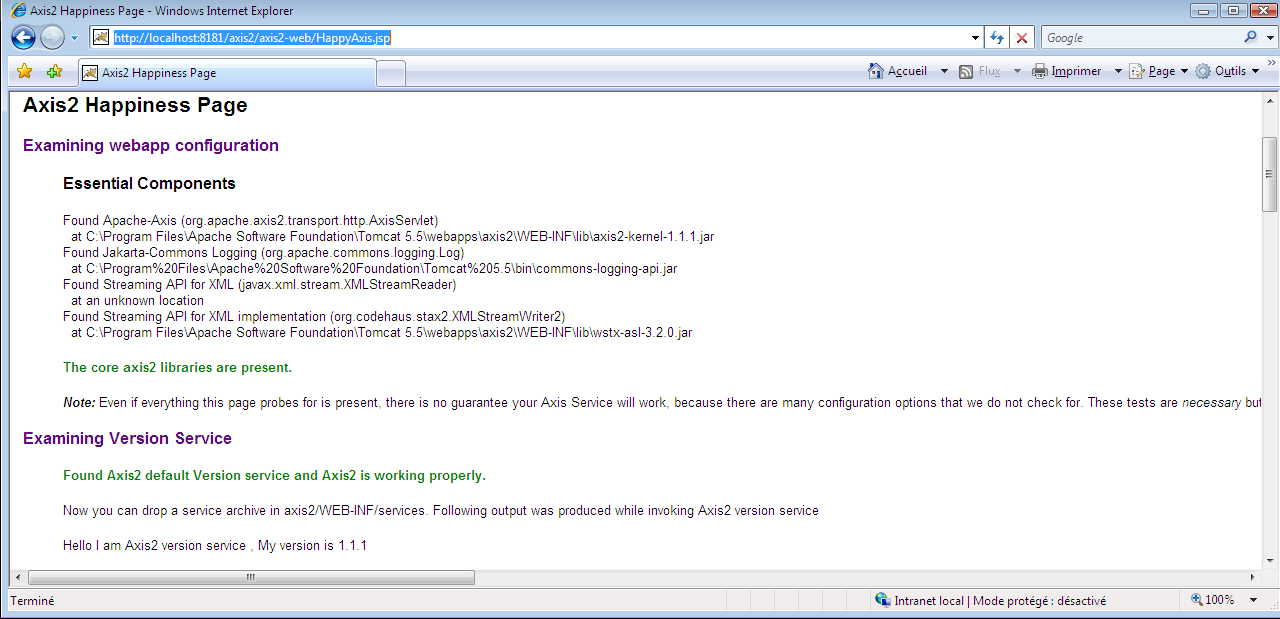
<http://localhost:8181/axis/> (le port 8181 est le port de TomCat). Puis cliquer sur le lien *Verification* (<http://localhost:8181/axis/happyaxis.jsp>). On doit obtenir ceci :



- **Déploiement d’Axis 2.1.1.1** : Nous avions copié tout à l’heure le fichier axis2.war, nous allons maintenant déployer Axis 2. Pour cela il suffit de lancer l’URL <http://localhost:8181/axis2>

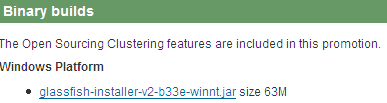
- **Lancement d’Axis 2.1.1.1** : Il suffit de lancer l’URL <http://localhost:8181/axis2>

Cliquer sur *Validate* (<http://localhost:8181/axis2/axis2-web/HappyAxis.jsp>) et vérifier que tout est bon.



- **GlassFish V2**: <https://glassfish.dev.java.net/>

Télécharger la dernière version pour la plateforme Windows :



Sous l’invite de commande dos, lancer la commande suivante :

java -Xmx256m -jar *glassfish-installer-v2-b33.jar*

Pour le chemin d’installation : C:\glassfish\

Les paramètres de déploiement du serveur GlassFish sont dans le fichier suivant :

C:\glassfish\setup.xml

Il faut juste changer le port d’écoute du serveur pour ne pas être en conflit avec le serveur IIS qui utilise le port 8080 :

<property name="admin.user" value="admin"/>

<property name="admin.password" value="adminadmin"/>

<property name="admin.port" value="4848"/>

<property name="instance.port" value="**8686**"/>

Aller ensuite dans le répertoire : C:\glassfish\lib\ant\bin

Puis lancer la commande suivante : ant -f ../../../setup.xml

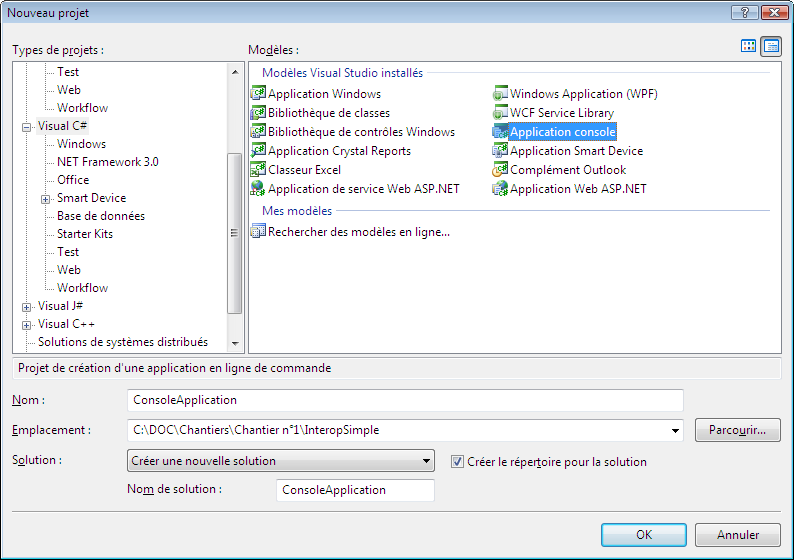
- **NetBeans 5.5.1**: <http://www.netbeans.info/downloads/dev.php>

1. Java consomme des web services .Net
   1. Création d'un Service Web .Net

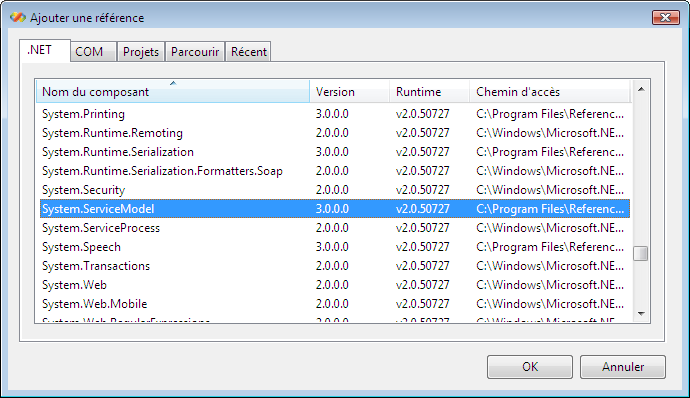
Lancer Visual Studio 2005.



Créer un nouveau Projet / Visual C# / Application Console.



Faire un clic droit sur le projet, puis *Ajouter une référence*. Il faut ajouter une référence sur ***System.ServiceModel***.



Pour commencer avec WCF je vous conseille de lire ces deux articles :

-<http://www.microsoft.com/france/msdn/netframework/3/wcf/introduction-a-windows-communication-framework.mspx>

- <http://www.c2i.fr/code.aspx?IDCode=666>

Nous allons créer 4 services web en WCF :

* **Un service web utilisant des types primitifs**
* **Un service web utilisant des tableaux et des collections**
* **Un service web utilisant des types complexes**
* **Un service web utilisant des exceptions**

1. Un service web utilisant des types primitifs

Vous allez ajouter une interface *IOperationSurTypesPrimitifs.cs* à votre projet.

***Fichier : IOperationSurTypesPrimitifs.cs***

using System;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel; //WCF

namespace InteropSOA.InteropSimple

{

/\* 1. Des types simples via une fonction qui prend en paramètre tous les types primitifs autorisés et retourne leur représentation sous forme de chaîne de caractères. Vous noterez que cette fonction prend en paramètre des types qui n'existent pas en Java (Guids et types non signés).

\*/

[ServiceContract(Namespace = "http://www.TypesPrimitifs.com")]

public interface IOperationSurTypesPrimitifs

{

[OperationContract]

string ConcatNames(string in\_firstName, string in\_lastName);

[OperationContract]

string PrimitiveTypes(bool in\_bl, char in\_c, sbyte in\_sb, byte in\_b,

short in\_s, ushort in\_us,int in\_i, uint in\_ui,

long in\_l, ulong in\_ul, double in\_d, float in\_f,

decimal in\_dm, DateTime in\_dt, Guid in\_g);

}

}

Il faut maintenant implémenter cette interface. Pour cela nous allons ajouter une classe *OperationSurTypesPrimitifs.cs* qui implémente l’interface IOperationSurTypesPrimitifs.

***Fichier : OperationSurTypesPrimitifs.cs***

using System;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

using InteropSOA.InteropSimple;

namespace InteropSOA.InteropSimple.ServeurDotNet

{

public class OperationSurTypesPrimitifs : IOperationSurTypesPrimitifs

{

#region IOperationSurTypesPrimitifs

public string ConcatNames(string in\_firstName, string in\_lastName)

{

Console.WriteLine("1.1) Demande du service 'ConcatNames' : " +

in\_firstName + " " + in\_lastName);

return string.Format("Bonjour {0} {1}", in\_firstName, in\_lastName);

}

public string PrimitiveTypes(bool in\_bl, char in\_c, sbyte in\_sb, byte

in\_b, short in\_s, ushort in\_us, int in\_i,

uint in\_ui, long in\_l, ulong in\_ul,

double in\_d, float in\_f, decimal in\_dm,

DateTime in\_dt, Guid in\_g)

{

Console.WriteLine("1.2) Demande du service 'PrimitiveTypes'");

return "\nBool : " + in\_bl.ToString() + "\nChar : " + in\_c.ToString() +

"\nSByte : " + in\_sb.ToString() + "\nByte : " + in\_b.ToString() +

"\nShort : " + in\_s.ToString() + "\nUShort : " + in\_us.ToString() +

"\nInt : " + in\_i.ToString() + "\nUInt : " + in\_ui.ToString() +

"\nLong : " + in\_l.ToString() + "\nULong : " + in\_ul.ToString() +

"\nDouble : " + in\_d.ToString() + "\nFloat : " + in\_f.ToString() +

"\nDecimal : " + in\_dm.ToString() + "\nDateTime : " +

in\_dt.ToString() + "\nGuid : " + in\_g.ToString();

}

#endregion

}

}

Maintenant que l’on a créé l’interface et la classe qui l’implémente, on aimerait bien lancer notre service web. Il nous manque la définition du point d’accès, pour cela il faut faire clic droit sur le fichier *app.config* et faire *Edit WCF Configuration* (C’est l’outil Microsoft Service Configuration Editor qui provient de l’extension de Visual Studio pour WCF).

Sur le dossier service, faire un clic droit, puis *New Service*.

*Behavior Configuration* = httpGetEnabledBehaviors

*Name* = InteropSOA.InteropSimple.ServeurDotNet.OperationSurTypesPrimitifs

Sur *Host*, il faut appuyer sur le bouton *New* et mettre l’adresse où le service va être exposé.

*Base Adress* = http://localhost:5000/TypesPrimitifs

Sur *Endpoints* faire un clic droit et *New service Endpoint*, dans l’onglet *General* renseigner les attributs suivants :

*Binding* = basicHttpBinding

*Contract* = InteropSOA.InteropSimple.ServeurDotNet.**IOperationSurTypesPrimitifs**

*Remarque :* Pour le contrat il faut fournir l’interface du service web. Pour fournir le contrat, appuyer sur le bouton "…" puis sélectionner le dossier Bin/Debug/VotreExe.exe pour accéder ensuite à toutes les interfaces définies.

Il ne manque plus que de lancer le service web, pour cela il faut modifier le fichier *program.cs*.

***Fichier : program.cs***

using System;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

class Program

{

static void Main(string[] args)

{

using (ServiceHost host = new ServiceHost(typeof(InteropSOA.InteropSimple.ServeurDotNet. OperationSurTypesPrimitifs)))

{

host.Open();

Console.WriteLine("Le service est lancé...");

Console.ReadLine();

host.Close();

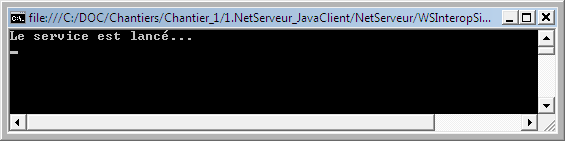
Console.WriteLine("Le service est arrêté...");

}

}

}

Compiler et exécuter maintenant le projet, pour obtenir :



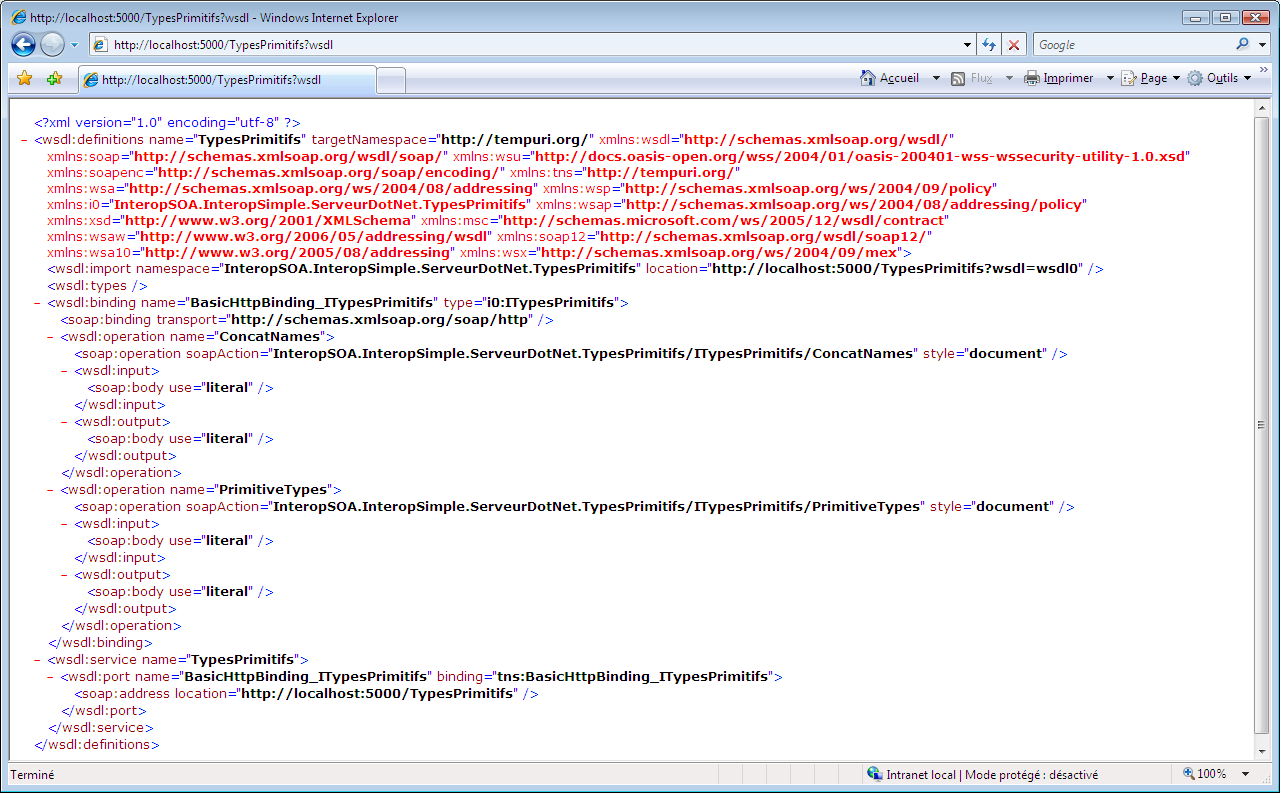
Pour tester maintenant ce web service, lancer l’url : <http://localhost:5000/TypesPrimitifs>



Pour voir le fichier WDSL, aller à l’adresse :

<http://localhost:5000/TypesPrimitifs?wsdl>

On remarque que le type d’encodage est *Document / Literal Wrapped*.



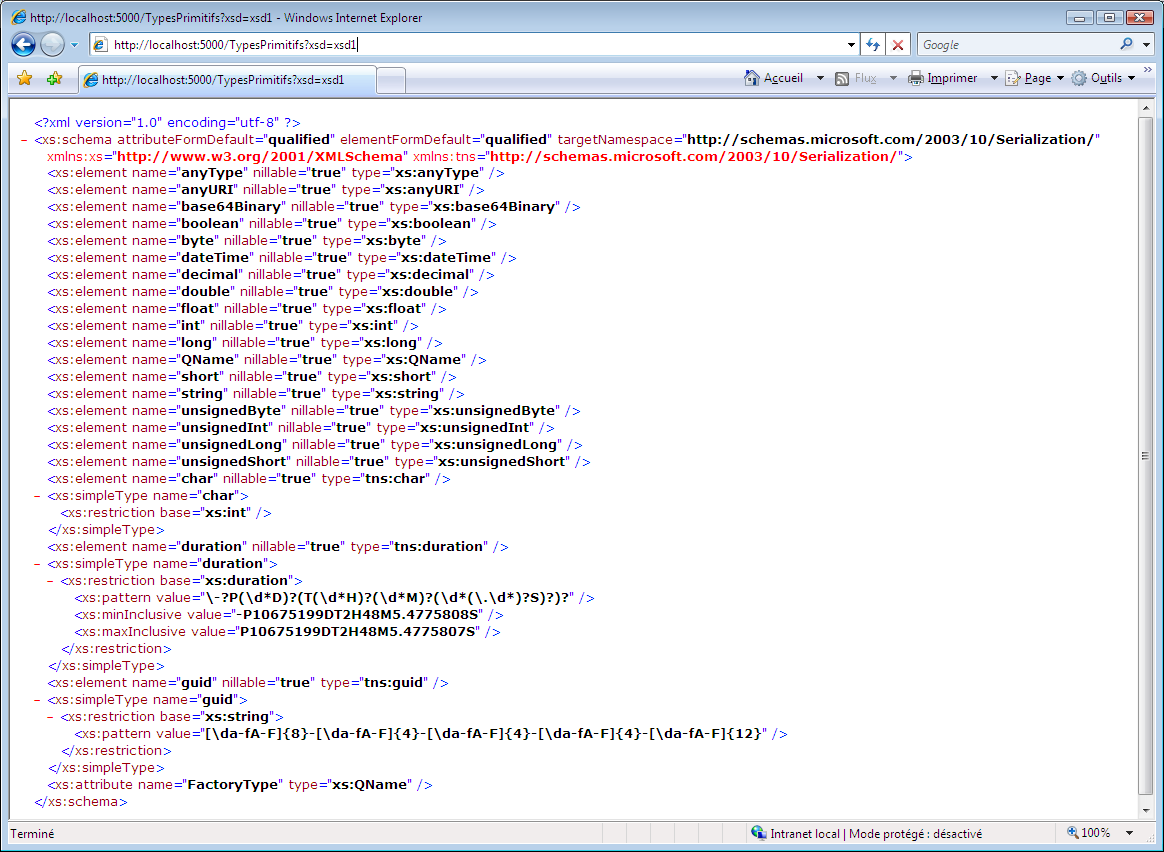
Consulter également les autres fichiers :

- <http://localhost:5000/TypesPrimitifs?wsdl=wsdl0>

- <http://localhost:5000/TypesPrimitifs?xsd=xsd0>

- <http://localhost:5000/TypesPrimitifs?xsd=xsd1>

On peut voir les types utilisés dans le WSDL :



Pour mieux contrôler la version de SOAP utilisée, il est recommandé de définir son propre **binding** dans le fichier *app.config*. Pour cela relancer l’outils Microsoft Service Configuration Editor.

Sur le répertoire *Binding*, faire *New Binding Configuration* et choisir *CustomBinding*.

*Name* = MyCustomBinding

Dans la partie *TextMessageEncoding* :

*MessageVersion* = SOAP11 ou SOAP12

Penser à revenir sur le service : dans le EndPoint, il faut mettre :

*Binding* = CustomBinding

*BindingConfiguration* = MyCustomBinding

On peut ainsi très facilement changer le binding, le endpoint et même créer plusieurs endpoints différents pour le même service afin d’être le plus interopérable avec les anciennes applications et celles de demain.

1. Un service web utilisant des tableaux et des collections

Ajouter une interface *IOperationSurTableaux.cs* au projet.

***Fichier : IOperationSurTableaux.cs***

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

using System.Runtime.Serialization; //Pour [DataContract] & [DataMember]

namespace InteropSOA.InteropSimple

{

[DataContract]

public enum EColor : short

{

[EnumMember] //Ne pas oublier de mettre le [EnumMember] au dessus de chacun !

red,

[EnumMember]

blue,

[EnumMember]

green

};

[ServiceContract(Namespace = "http://www.Tableaux.com")]

public interface IOperationSurTableaux

{

[OperationContract]

string[] StringArray();

[OperationContract]

int[] IntArray();

[OperationContract]

WSClass[] ClassArray();

[OperationContract]

byte[] ByteArray(string in\_path);

}

}

*Remarque :* Il faut ajouter une référence au projet *Systeme.Runtime.Serialization*.

Il faut maintenant implémenter cette interface. Pour cela nous allons ajouter une classe *TypesTableaux.cs* qui implémente l’interface IOperationSurTableaux.

***Fichier : OperationSurTableaux.cs***

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.IO;

using System.ServiceModel;

using InteropSOA.InteropSimple;

namespace InteropSOA.InteropSimple.ServeurDotNet

{

class OperationSurTableaux : IOperationSurTableaux

{

#region 2. Des types tableaux simples ou bien des classes représentants

des tableaux (ArrayList).

public string[] StringArray()

{

Console.WriteLine("2.1) Demande du service 'StringArray'");

return Enum.GetNames(typeof(EColor));

}

public int[] IntArray()

{

Console.WriteLine("2.2) Demande du service 'IntArray'");

return new int[] { 0, 1, 2, 3 };

}

public WSClass[] ClassArray()

{

WSClass[] out\_MyTabClass = new WSClass[2];

out\_MyTabClass[0] = new WSClass();

out\_MyTabClass[0].Name = "Titi";

out\_MyTabClass[1] = new WSClass();

out\_MyTabClass[1].Name = "Toto";

Console.WriteLine("2.4) Demande du service 'ClassArray'");

return out\_MyTabClass;

}

public byte[] ByteArray(string in\_path) {

string l\_filename = in\_path; // @"C:\LocalDatas.xml";

byte[] out\_monBuffer = null;

Console.WriteLine("2.5) Demande du service 'ByteArray' (Path : " +

in\_path +")");

if (File.Exists(l\_filename))

{

// Le fichier est placé dans le FileStream

FileStream l\_fs = new FileStream(l\_filename, FileMode.Open);

out\_monBuffer = new byte[l\_fs.Length];

// Lecture de l'intégralité du FileStream

l\_fs.Read(out\_monBuffer, 0, System.Convert.ToInt32(l\_fs.Length));

// Fermeture du FileStream

l\_fs.Close();

}

return out\_monBuffer;

}

#endregion

}

}

1. Un service web utilisant des types complexes

Ajouter une interface *IOperationSurTypesComplexes.cs* au projet.

***Fichier : IOperationSurTypesComplexes.cs***

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel; //WCF

using System.Data; //DataSet

using System.Runtime.Serialization; //Pour [DataContract] & [DataMember]

namespace InteropSOA.InteropSimple

{

[DataContract]

public class WSClass

{

private string m\_Name;

[DataMember]

public string Name

{

get { return m\_Name; }

set { m\_Name = value; }

}

}

[DataContract]

public struct WSStruct

{

private string m\_Name;

[DataMember]

public string Name

{

get { return m\_Name; }

set { m\_Name = value; }

}

}

[DataContract]

public class WSMyDataRow

{

public WSMyDataRow(int intId, string strName)

{

m\_Id = intId;

m\_Name = strName;

}

private int m\_Id;

private string m\_Name;

[DataMember]

public int Id

{

get { return m\_Id; }

set { m\_Id = value; }

}

[DataMember]

public string Name

{

get { return m\_Name; }

set { m\_Name = value; }

}

}

[ServiceContract(Namespace = "http://www.TypesComplexes.com")]

public interface IOperationSurTypesComplexes

{

/\* La fonction retourne le nom de la couleur en fonction de l'énuméré

passé en paramètre.

\*/

[OperationContract]

string EnumType(EColor in\_Color);

/\* La fonction prend un tableau de structure et retourne une classe qui

copie le nom du premier élément du tableau donné en paramètre.

\*/

[OperationContract]

WSClass[] ClassAndStruct(WSStruct[] in\_MyStruct);

/\* La fonction construit et retourne un DataSet simple.

Fonctionne avec Axis 1.4 mais pas avec Axis 2.1.1.1

\*/

[OperationContract]

DataSet RawDataSet();

[OperationContract]

WSMyDataRow[] SmartDataSet();

}

}

*Remarque :* Il ne faut pas oublier de mettre [EnumMember] au dessus de chaque membre d’un type énuméré.

Il faut maintenant implémenter cette interface. Pour cela nous allons ajouter une classe *TypesComplexes.cs* qui implémente l’interface IOperationSurTypesComplexes.

***Fichier : OperationSurTypesComplexes.cs***

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

using System.Data;

using InteropSOA.InteropSimple;

namespace InteropSOA.InteropSimple.ServeurDotNet

{

class OperationSurTypesComplexes : IOperationSurTypesComplexes

{

#region Un énuméré

public string EnumType(EColor in\_Color)

{

Console.WriteLine("3.1) Demande du service 'EnumType'");

return Enum.GetName(typeof(EColor), in\_Color);

}

#endregion

#region Des structures et des classes.

public WSClass[] ClassAndStruct(WSStruct[] in\_MyStruct)

{

WSClass[] out\_MyTabClass = new WSClass[in\_MyStruct.Length];

for (int i = 0; i < in\_MyStruct.Length; i++)

{

out\_MyTabClass[i] = new WSClass();

out\_MyTabClass[i].Name = in\_MyStruct[i].Name;

}

Console.WriteLine("4.1) Demande du service 'ClassAndStruct'");

return out\_MyTabClass;

}

#endregion

#region Un DataSet.

private DataSet GenererDataSet()

{

// Create a new DataTable

DataTable l\_myDataTable = new DataTable("MyTable");

// Create new DataColumn, set DataType, ColumnName and add to

DataTable.

DataColumn myDataColumn = new DataColumn();

myDataColumn.DataType = System.Type.GetType("System.Int32");

myDataColumn.ColumnName = "id";

l\_myDataTable.Columns.Add(myDataColumn);

// Create second column

myDataColumn = new DataColumn();

myDataColumn.DataType = System.Type.GetType("System.String");

myDataColumn.ColumnName = "Name";

myDataColumn.AutoIncrement = false;

l\_myDataTable.Columns.Add(myDataColumn);

// Instantiate DataSet

DataSet out\_myDataSet = new DataSet();

out\_myDataSet.Tables.Add(l\_myDataTable);

// Create new DataRow objects and add them to the DataTable

for (int i = 0; i < 10; ++i)

{

DataRow myDataRow = l\_myDataTable.NewRow();

myDataRow["id"] = i;

myDataRow["Name"] = "MyName is " + i.ToString();

l\_myDataTable.Rows.Add(myDataRow);

}

return out\_myDataSet;

}

public DataSet RawDataSet()

{

Console.WriteLine("5.1) Demande du service 'RawDataSet'");

return GenererDataSet();

}

public WSMyDataRow[] SmartDataSet()

{

Console.WriteLine("5.2) Demande du service 'SmartDataSet'");

DataSet l\_ds = GenererDataSet();

DataTable l\_dt = l\_ds.Tables["MyTable"];

WSMyDataRow[] out\_MyDS = new WSMyDataRow[l\_dt.Rows.Count];

int i = 0;

foreach (DataRow dr in l\_dt.Rows)

{

out\_MyDS[i] = new WSMyDataRow(int.Parse(dr["id"].ToString()), dr["Name"].ToString());

i++;

}

return out\_MyDS;

}

#endregion

}

}

1. Un service web utilisant des exceptions

Ajouter une interface *IOperationAvecExceptions.cs* au projet.

***Fichier : IOperationAvecExceptions.cs***

using System;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

using System.Runtime.Serialization; //Pour [DataContract] & [DataMember]

namespace InteropSOA.InteropSimple

{

[DataContract]

public class MySpecificException

{

private string m\_Name;

[DataMember]

public string Name

{

get { return m\_Name; }

set { m\_Name = value; }

}

}

[ServiceContract(Namespace = "http://www.Exceptions.com")]

public interface IOperationAvecExceptions

{

[OperationContract]

[FaultContract(typeof(DivideByZeroException))]

void SendException();

[OperationContract]

[FaultContract(typeof(MySpecificException))]

void SendMyException();

[OperationContract]

void SendNotFaultedException();

}

}

Il faut maintenant implémenter cette interface. Pour cela nous allons ajouter une classe *TypesExceptions.cs* qui implémente l’interface IOperationAvecExceptions.

***Fichier : OperationAvecExceptions.cs***

using System;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

using System.Globalization;

using InteropSOA.InteropSimple;

namespace InteropSOA.InteropSimple.ServeurDotNet

{

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]

class OperationAvecExceptions : IOperationAvecExceptions

{

#region 6. Retourne une exception

public void SendFaultException()

{

Console.WriteLine("6.1) Demande du service 'SendFaultException'");

throw new FaultException("Boom FaultException");

}

public void SendException()

{

Console.WriteLine("6.2) Demande du service 'SendException'");

DivideByZeroException l\_exp = new DivideByZeroException();

FaultReasonText l\_frt = new FaultReasonText("Utilisation d'une exception DotNet : Division par zéro !", CultureInfo.GetCultureInfo("fr-FR"));

FaultReason l\_fr = new FaultReason(l\_frt);

throw new FaultException<DivideByZeroException>(l\_exp, l\_fr);

}

public void SendMyException()

{

Console.WriteLine("6.3) Demande du service 'SendMyException' (Généricité)");

MySpecificException l\_exp = new MySpecificException();

l\_exp.Name = "Les données internes de mon exception ";

FaultReasonText l\_frt = new FaultReasonText("Utilisation de ma propre exception : MySpecificException.", CultureInfo.GetCultureInfo("fr-FR"));

FaultReason l\_fr = new FaultReason(l\_frt);

throw new FaultException<MySpecificException>(l\_exp, l\_fr);

}

public void SendMyFaultException()

{

Console.WriteLine("6.3 Bis ) Demande du service 'SendMyFaultException' (Héritage)");

throw new MySpecificFaultException("Les données internes de ma FaultException ");

}

public void SendNotFaultedException()

{

Console.WriteLine("6.4) Demande du service 'SendNotFaultedException'");

throw new Exception();

}

#endregion

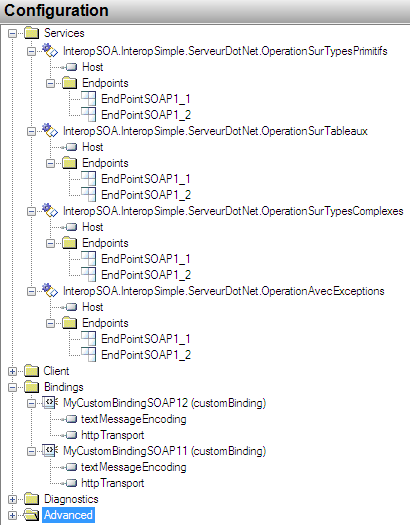
}

}

La nouvelle version du fichier app.config propose un service pour chaque contrat. Pour chaque service il y a deux endpoints. Le premier endpoint est configuré en SOAP 1.1, le deuxième est en SOAP 1.2.

Voici les adresses où sont situés les services :

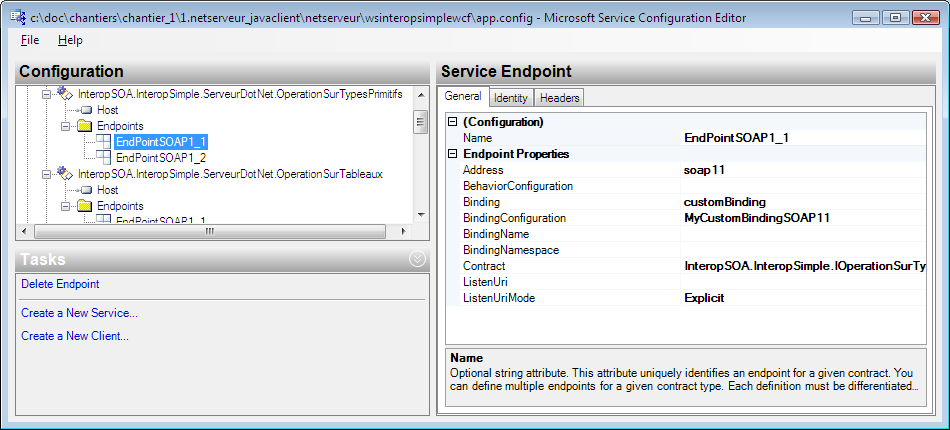
* <http://localhost:5000/TypesPrimitifs>
* <http://localhost:5000/TypesTableaux>
* <http://localhost:5000/TypesComplexes>
* <http://localhost:5000/TypesExceptions>



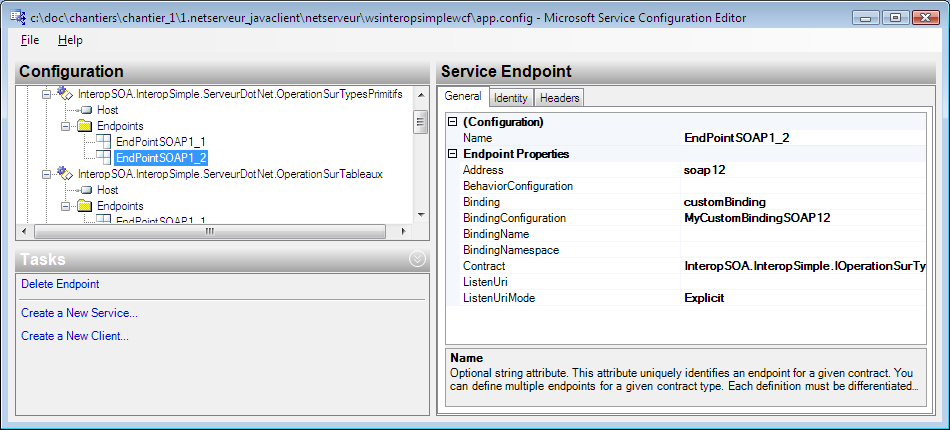
*Détail du fichier app.config*



*Détail du Host*



*Détail du EndPoint en SOAP 1.1*



*Détail du EndPoint en SOAP 1.2*

Pour lancer tous les services, on modifie le fichier *program.cs*

***Fichier : program.cs***

using System;

using System.Collections.Generic;

using System.Text;

using System.ServiceModel;

class Program

{

static void Main(string[] args)

{

ServiceHost host0 = new ServiceHost(typeof(InteropSOA.InteropSimple.ServeurDotNet.OperationBasiques));

host0.Open();

Console.WriteLine("Le service OperationBasiques est lancé...");

ServiceHost host1 = new ServiceHost(typeof(InteropSOA.InteropSimple.ServeurDotNet.OperationSurTypesPrimitifs));

host1.Open();

Console.WriteLine("Le service OperationSurTypesPrimitifs est lancé...");

ServiceHost host2 = new ServiceHost(typeof(InteropSOA.InteropSimple.ServeurDotNet.OperationSurTypesComplexes));

host2.Open();

Console.WriteLine("Le service OperationSurTypesComplexes est lancé...");

ServiceHost host3 = new ServiceHost(typeof(InteropSOA.InteropSimple.ServeurDotNet.OperationSurTableaux));

host3.Open();

Console.WriteLine("Le service OperationSurTableaux est lancé...");

ServiceHost host4 = new ServiceHost(typeof(InteropSOA.InteropSimple.ServeurDotNet.OperationAvecExceptions));

host4.Open();

Console.WriteLine("Le service OperationAvecExceptions est lancé...");

Console.ReadLine();

host0.Close();

Console.WriteLine("Le service OperationBasiques est arrêté...");

host1.Close();

Console.WriteLine("Le service OperationSurTypesPrimitifs est arrêté...");

host2.Close();

Console.WriteLine("Le service OperationSurTypesComplexes est arrêté...");

host3.Close();

Console.WriteLine("Le service OperationSurTableaux est arrêté...");

host4.Close();

Console.WriteLine("Le service OperationAvecExceptions est arrêté...");

}

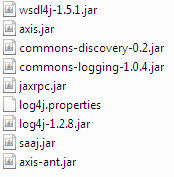
}

* 1. Java (Proxy Axis 1.4) consomme des web services .Net

On va créer un client Java avec un Proxy Axis 1.4. Avec cette version d’Axis, on ne peut pas faire du SOAP 1.2 car il n’est pas 100% compatible. On va donc se connecter à un WSDL qui est en SOAP 1.1.

On va tout d’abord **créer le proxy avec Axis 1.4** (appelé aussi stub) :

Dans un dossier on a le fichier *"genererWSDL2Java - Axis 1.4 - Soap 1.1.bat"* et un répertoire *lib* contenant les jar suivants :



***Fichier : genererWSDL2Java - Axis 1.4 - Soap 1.1.bat***

set JAVA\_HOME=C:\Program Files\Java\jdk1.6.0

set AXIS\_HOME=.\lib

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\axis.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\log4j-1.2.8.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\commons-logging-1.0.4.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\commons-discovery-0.2.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\axis-ant.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\jaxrpc.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\saaj.jar

set CLASSPATH=%CLASSPATH%;%AXIS\_HOME%\wsdl4j-1.5.1.jar

echo %CLASSPATH%

java org.apache.axis.wsdl.WSDL2Java -p InteropSOA.InteropSimple.ProxyJava1\_1.TypesPrimitifs http://localhost:5000/TypesPrimitifs?wsdl

java org.apache.axis.wsdl.WSDL2Java -p InteropSOA.InteropSimple.ProxyJava1\_1.TypesTableaux http://localhost:5000/TypesTableaux?wsdl

java org.apache.axis.wsdl.WSDL2Java -p InteropSOA.InteropSimple.ProxyJava1\_1.TypesComplexes http://localhost:5000/TypesComplexes?wsdl

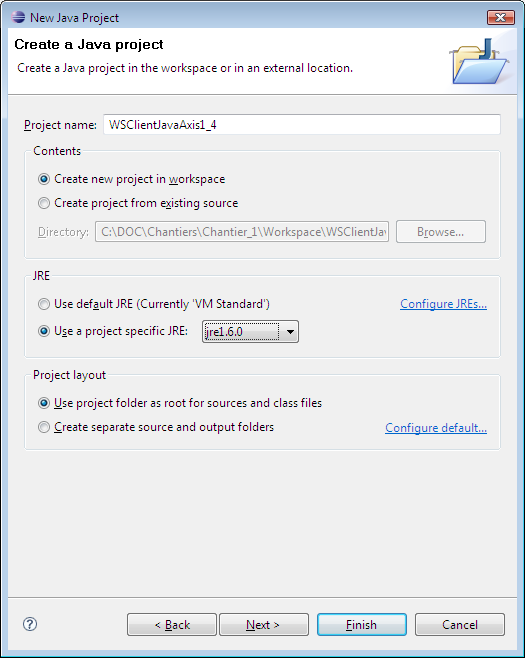
java org.apache.axis.wsdl.WSDL2Java -p InteropSOA.InteropSimple.ProxyJava1\_1.TypesExceptions http://localhost:5000/TypesExceptions?wsdl

pause

Il suffit d’ exécuter le fichier *bat* pour lancer la génération des proxy pour chaque service.

*Remarque* : Il faut supprimer le endpoint *EndPointSOAP1\_2* du service *InteropSOA.InteropSimple. ServeurDotNet.OperationAvecExceptions* dans le fichier App.Config sur serveur WCF pour pouvoir générer le Proxy Axis 1.4 sans erreur.

Une fois que les proxy ont été générés, on peut créer le client Java. Tout d'abord, il faut lancer Eclipse. Puis créer un nouveau projet de type *Java Project*

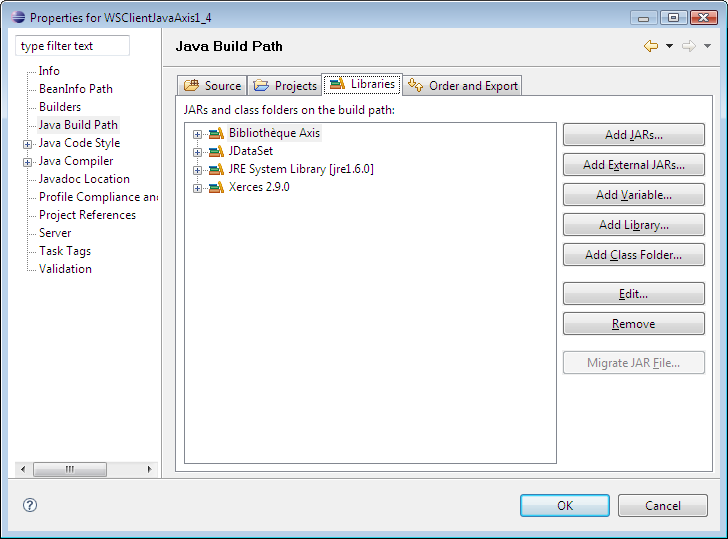


Dans la partie librairie, il faut créer une **nouvelle bibliothèque utilisateur** que l’on va appeler ***Bibliothèque Axis 1.4****.* Cette bibliothèque contient tous les *jar* présents dans le répertoire *C:\Program Files\Apache Software Foundation\axis-1\_4\lib*.

Il faudra créer aussi une **bibliothèque utilisateur *Xerces 2.9.0.*** Cette bibliothèque contient tous les *jar* présents dans le répertoire *C:\Program Files\Apache Software Foundation\xerces-2\_9\_0.* Xerces 2.9.0 peut être trouvé à cette adresse : <http://archive.apache.org/dist/xml/xerces-j/>

Il vous faudra aussi une **bibliothèque utilisateur *JDataSet.*** Cette bibliothèque contient tous les *jar* présents dans le répertoire *C:\Program Files\Apache Software Foundation\jdataset\lib.* JDataSet peut être trouvé à cette adresse : <http://jdataset.de/>.

Vous devez avoir les bibliothèques suivantes :



**On va maintenant importer les proxy :**

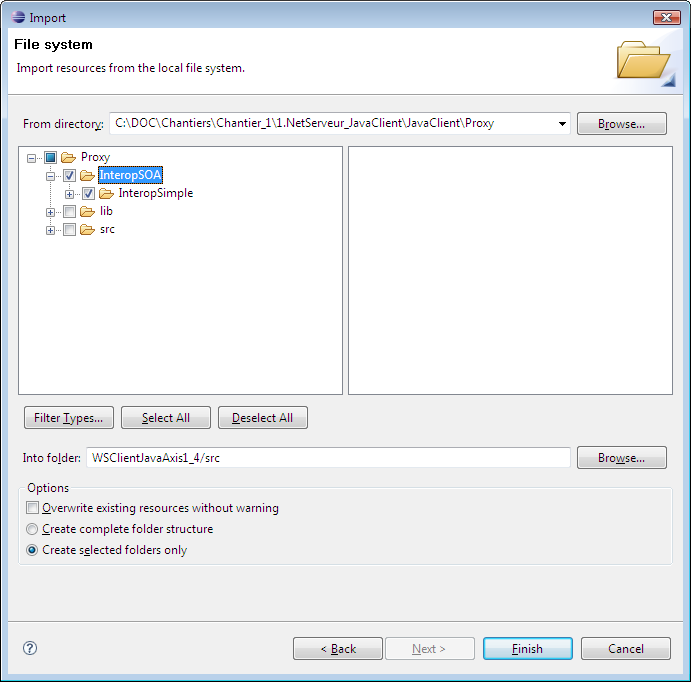
srcFolder

Vous allez ajouter un dossier source au projet :

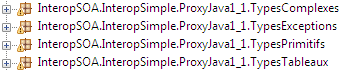
*Folder Name* = src

ImportDans le dossier *src* faire un import :

Puis choisir *File System,*



Cela va importer les proxy avec leur package :



**On va maintenant coder le client :**

class

Dans le dossier *src* ajouter une classe :

*Package* = InteropSOA.InteropSimple.ProxyJava1\_1

*Name* = JavaClient

Aller voir le code source de *JavaClient.java* dans les fichiers sources.

***Extrait du Fichier : JavaClient.java***

TypesPrimitifs locator = **new** TypesPrimitifsLocator();

ITypesPrimitifs service = locator.getCustomBinding\_ITypesPrimitifs();

// 1. Des types simples

// 1.1) Appel de concatNames

String Result = "";

Result = service.concatNames("Robert", "Dupont");

System.*out*.println("\n\n1.1) Appel de concatNames réussi : " + Result);

class

Dans le dossier *src* ajouter une classe :

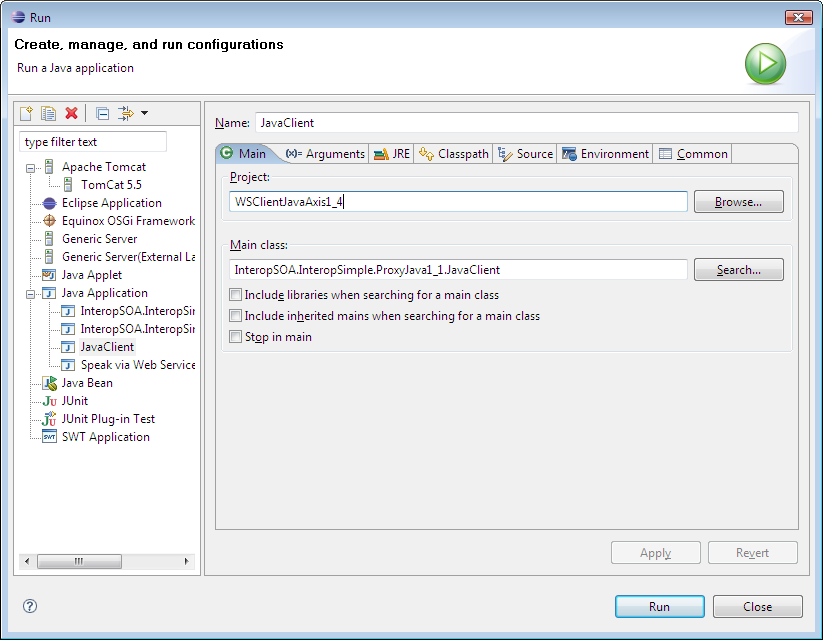
*Package* = InteropSOA.InteropSimple.ProxyJava1\_1

*Name* = NDataSet

Aller voir le code source de *NDataSet.java* dans les fichiers sources.

Pour compiler le client, il faut aller dans *Run / Run …*

Puis sur *Java Application* faire *New*.



Puis appuyer sur le bouton *Run*.

* 1. Java (Proxy Axis 2.1.1.1) consomme des web services .Net

On va créer un client Java avec un Proxy Axis 2.1.1.1. Avec cette version d’Axis, on peut faire du SOAP 1.1 & 1.2. Étant donné que l’on a déjà fait du SOAP 1.1 avec la version 1.4 d’Axis, on va plutôt faire du SOAP 1.2. Mais vous trouverez dans les sources jointes au livre blanc le code source pour vous connecter en SOAP 1.1 et en SOAP 1.2.

On va tout d’abord **créer le proxy avec Axis 2.1.1.1** :

***Fichier : genererWSDL2Java - Axis 2.1.1.1 - Soap 1.2.bat***

@echo off

set JAVA\_HOME=C:\Program Files\Java\jdk1.6.0

set AXIS2\_HOME=C:\Program Files\Apache Software Foundation\axis2-1.1.1

rem set the classes

setlocal EnableDelayedExpansion

rem loop through the libs and add them to the class path

set AXIS2\_CLASS\_PATH=%AXIS2\_HOME%

FOR %%c in ("%AXIS2\_HOME%\lib\\*.jar") DO set AXIS2\_CLASS\_PATH=!AXIS2\_CLASS\_PATH!;%%c

rem ----- Execute The Requested Command ------------------------------

echo Using AXIS2\_HOME: %AXIS2\_HOME%

echo Using JAVA\_HOME: %JAVA\_HOME%

set \_RUNJAVA="%JAVA\_HOME%\bin\java"

%\_RUNJAVA% %JAVA\_OPTS% -cp "%AXIS2\_CLASS\_PATH%" org.apache.axis2.wsdl.WSDL2Java -p interopsoa.interopsimple.proxyjava1\_2.typesprimitifs -uri http://localhost:5000/TypesPrimitifs?wsdl

%\_RUNJAVA% %JAVA\_OPTS% -cp "%AXIS2\_CLASS\_PATH%" org.apache.axis2.wsdl.WSDL2Java -p interopsoa.interopsimple.proxyjava1\_2.typestableaux -uri http://localhost:5000/TypesTableaux?wsdl

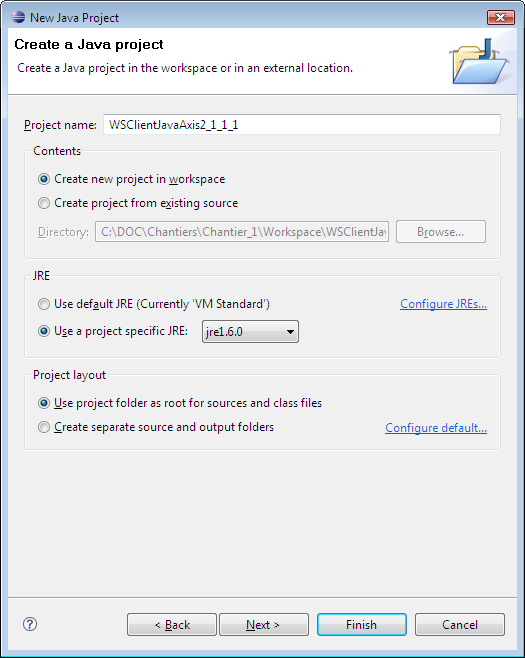
%\_RUNJAVA% %JAVA\_OPTS% -cp "%AXIS2\_CLASS\_PATH%" org.apache.axis2.wsdl.WSDL2Java -p interopsoa.interopsimple.proxyjava1\_2.typescomplexes -uri http://localhost:5000/TypesComplexes?wsdl

%\_RUNJAVA% %JAVA\_OPTS% -cp "%AXIS2\_CLASS\_PATH%" org.apache.axis2.wsdl.WSDL2Java -p interopsoa.interopsimple.proxyjava1\_2.typesexceptions -uri http://localhost:5000/TypesExceptions?wsdl

pause

Il suffit d’exécuter le fichier *bat* pour lancer la génération des proxy pour chaque service.

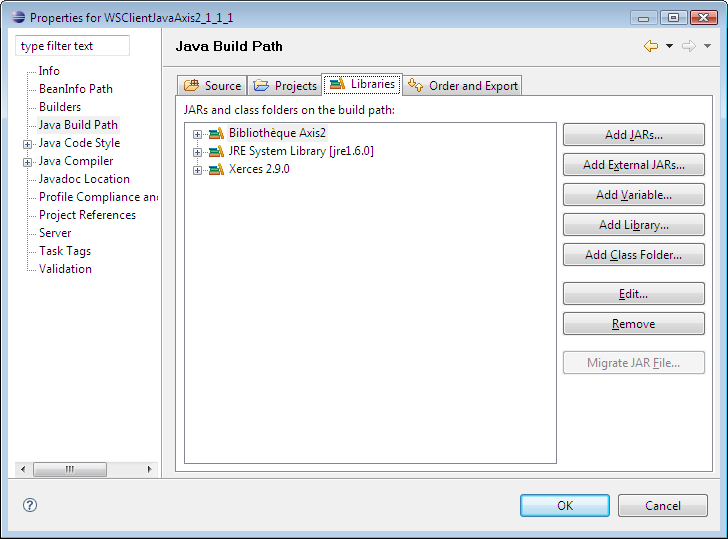
Une fois que les proxy ont été générés, on peut créer le client Java. Tout d'abord, il faut lancer Eclipse. Puis créer un nouveau projet de type *Java Project*



Dans la partie librairie, il faut créer une **nouvelle bibliothèque utilisateur** que l’on va appeler ***Bibliothèque Axis 2.1.1.1****.* Cette bibliothèque contient tous les *jar* présents dans le répertoire *C:\Program Files\Apache Software Foundation\axis2-1.1.1\lib*

Il vous faudra aussi une **bibliothèque utilisateur *Xerces 2.9.0.*** Cette bibliothèque contient tous les *jar* présents dans le répertoire *C:\Program Files\Apache Software Foundation\xerces-2\_9\_0.* Xerces 2.9.0 peut être trouvé à cette adresse : <http://archive.apache.org/dist/xml/xerces-j/>

Vous devez avoir les bibliothèques suivantes :



**On va maintenant importer les proxy :**

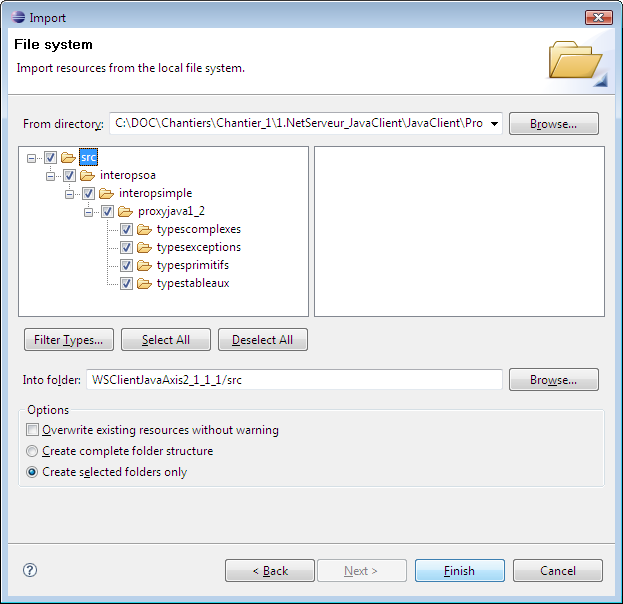
srcFolder

Ajouter un dossier source au projet :

*Folder Name* = src

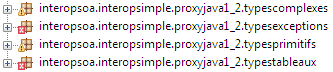
ImportDans le dossier *src* faire un import :

Puis choisir *File System,*



*Remarque* : il faut cliquer sur chaque plus en face de *typescomplexes*, *typesexceptions*, …

Cela va importer les proxy avec leur package :



Pour corriger les erreurs dans interopsoa.interopsimple.proxyjava1\_2.typesexceptions, il faut mettre java.lang.Exception à la place de Exception.

Pour corriger les erreurs dans interopsoa.interopsimple.proxyjava1\_2.typestableaux, il faut mettre java.lang.String à la place de String.

**On va maintenant coder le client :**

classDans le dossier *src* ajouter une classe :

*Package* = interopsoa.interopsimple.proxyjava1\_2

*Name* = JavaClient

Aller voir le code source de *JavaClient.java* dans les fichiers sources.

***Extrait du Fichier : JavaClient.java***

TypesPrimitifsStub stub = **new** TypesPrimitifsStub();

// 1. Des types simples

// 1.1) Appel de concatNames

ConcatNames req1\_1 = **new** ConcatNames();

req1\_1.setFirstName("Robert");

req1\_1.setLastName("Dupont");

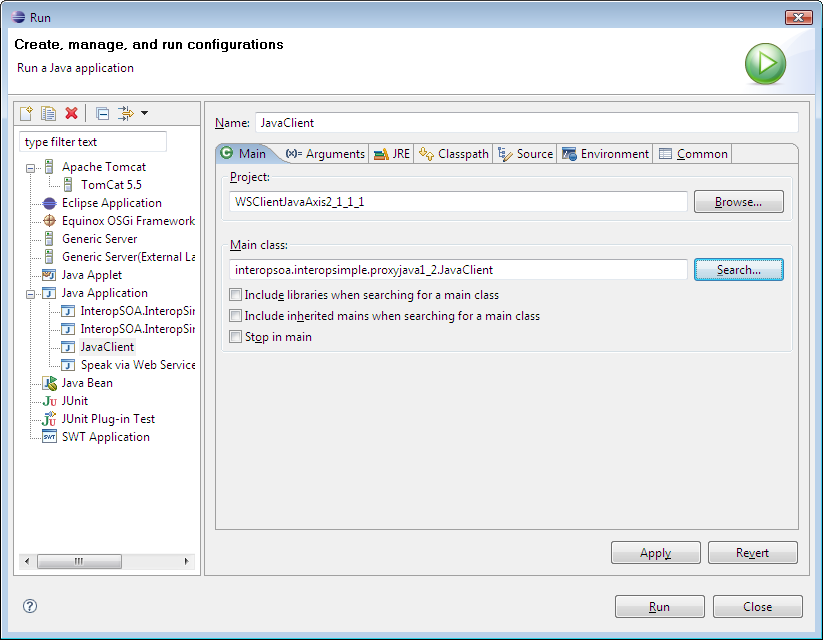
ConcatNamesResponse resp1\_1 = stub.ConcatNames(req1\_1);

System.*out*.println("\n\n1.2) Appel de primitiveTypes réussi : " +

resp1\_1.getConcatNamesResult());

Pour compiler le client, il faut aller dans *Run / Run …*

Puis sur *Java Application* faire *New*.



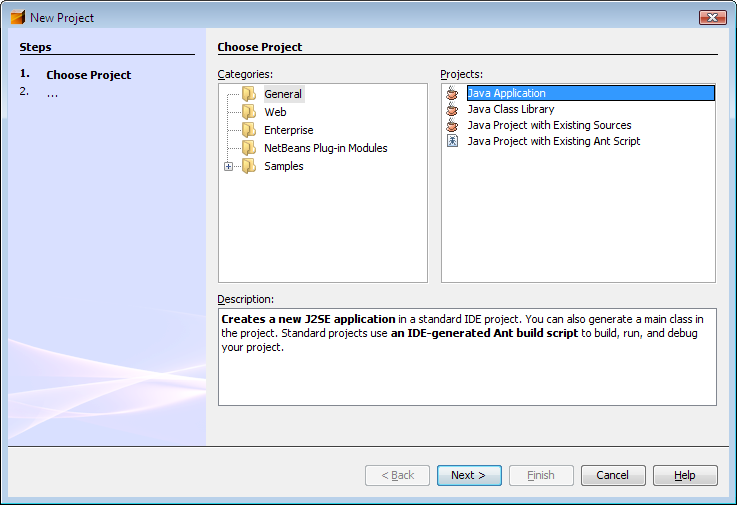
Puis cliquer sur le bouton *Run*.

*Remarque* : l’appel d’*ArrayOfAnyType* ne fonctionne pas.

* 1. Java (Proxy JAX-WS 2.1) consomme des web services .Net

On va créer un client Java avec un Proxy JAX-WS 2.1. Avec cette version de JAX-WS, on peut uniquement faire un proxy pour un service en SOAP 1.1.

On va tout d’abord **lancer NetBeans 5.5.1 DEV,** puis ajouter un nouveau projet de type Java Application.

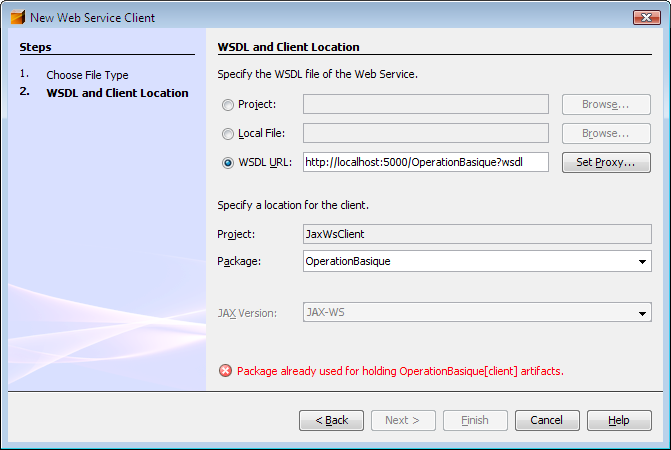


*Nouveau projet de type Java Application sous NetBeans*

Ajouter ensuite au projet un nouveau *Web Service Client …*

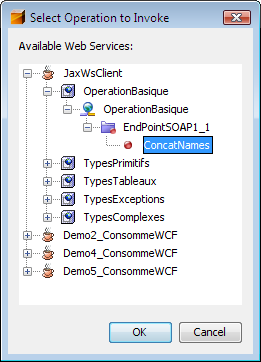
*WSDL URL* = http://localhost:5000/OperationBasique?wsdl

*Package* = OperationBasique



*Outils de génération de proxy*

Une fois que le proxy est généré, il ne reste plus qu’à coder la partie cliente. Pour cela NetBeans propose un assistant. Pour lancer cet assistant dans la page Main.java du code source, faire un clic droit puis *Web Service Client Resources / Call Web Service Operation.* Sélectionner *ConcatNames* et appuyer sur *OK.*



*Outils de génération du code client*

***Code généré par l’outils***

try { // Call Web Service Operation

OperationBasique.OperationBasique service = new OperationBasique.OperationBasique();

OperationBasique.IOperationBasique port = service.getEndPointSOAP11();

// TODO initialize WS operation arguments here

java.lang.String inFirstName = "";

java.lang.String inLastName = "";

// TODO process result here

java.lang.String result = port.concatNames(inFirstName, inLastName);

System.out.println("Result = "+result);

} catch (Exception ex) { // TODO handle custom exceptions here

}

Il ne reste plus qu’a compléter les paramètres dans le code source et à compiler l’application.

***Code généré par l’outils complété***

try {

OperationBasique.OperationBasique service = new OperationBasique.OperationBasique();

OperationBasique.IOperationBasique port = service.getEndPointSOAP11();

java.lang.String inFirstName = "Robert";

java.lang.String inLastName = "Dupont";

java.lang.String result = port.concatNames(inFirstName, inLastName);

System.out.println("\n\n1.1) Appel de concatNames réussi : " + result);

} catch (Exception ex) {

System.out.println("\n\nErreur dans OperationBasique : " + ex);

}

1. .Net 3.0 consomme des web services Java
   1. Création d’un Web Service Java (Axis 1.4)

Nous allons créer un Service Web avec Axis 1.4. Avec cette version d’Axis on va utiliser le système de fichier *JWS* pour déployer notre web service.

Le fichier *JWS* est l’équivalent des *JSP* pour les services web. Java Web Services (*JWS*) est un nouveau standard dans l’implémentation des Web Services. WebLogic Workshop est à l’origine de ce standard. De plus ce nouveau standard est supporté par la JCP (Java Community Process) à partir du JRE 1.4. Le but principal de JWS est de proposer une implémentation facile à apprendre et à utiliser. AXIS se base sur ce standard depuis la version 1.2 pour offrir ces fonctionnalités, aidant ainsi le développeur dans sa mise en place et son utilisation des Web Services.

**Avantages de JWS** :

• Facile et rapide.

**Inconvénients de JWS** :

• Impossible de packager plusieurs classes.

• Compiler à l'exécution/accès (donc moins rapide, et erreurs potentielles découvertes seulement à cet instant).

• Besoin du code pour connaître l'interface.

• Paramétrage limité sur l'accessibilité du service.

Avec le déploiement par le fichier JWS, la méthode d’encodage proposée au niveau du fichier WSDL est **RPC/Encoded** (Appelé aussi SOAP Encoded ou Version 5/Encoded).

Comme le fichier JWS empêche de packager plusieurs classes, on ne pourra pas dans cet exemple échanger un type classe au niveau du service web.

Pour déployer le service web, il faut mettre le fichier *JavaServer1\_4.jws* dans le répertoire

C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\axis

Puis pour atteindre le fichier WSDL il faut aller à l’adresse suivante :

<http://localhost:8181/axis/JavaServer1_4.jws?wsdl>

***Fichier : JavaServer1\_4.jws***

import java.lang.\*;

import java.math.\*;

import java.util.\*;

public class JavaServer1\_4

{

public JavaServer1\_4() { }

// Simples types and wrappers ; ignore operation overflows and loss of precisions if any

public boolean isItTrue(Boolean lhs, Boolean rhs) { return lhs.booleanValue() && rhs.booleanValue(); }

public byte addByte(Byte lhs, Byte rhs) { return (byte)(lhs.byteValue() + rhs.byteValue()); }

public double addDouble(Double lhs, Double rhs) { return lhs.doubleValue() + rhs.doubleValue(); }

public float addFloat(Float lhs, Float rhs) { return lhs.floatValue() + rhs.floatValue(); }

public int addInt(Integer lhs, Integer rhs) { return lhs.intValue() + rhs.intValue(); }

public long addLong(Long lhs, Long rhs) { return lhs.longValue() + rhs.longValue(); }

public short addShort(Short lhs, Short rhs) { return (short)(lhs.shortValue() + rhs.shortValue()); }

// Java standard types

public BigDecimal addBigDecimal(BigDecimal lhs, BigDecimal rhs) { return lhs.add(rhs); }

public BigInteger addBigInteger(BigInteger lhs, BigInteger rhs) { return lhs.add(rhs); }

public Date getDate(Calendar d) { return d.getTime(); }

public String getMessage(String s) { return "Message is " + s; }

// Some arrays

public int[] getIntegers() { return new int[] { 1, 2, 3 }; }

public String[] getStrings() { return new String[] { "Winwise", "THB", "Kader" }; }

public Long[] getLongs() { return new Long[] { new Long(1), new Long(2), new Long(3) }; }

// Exception propre au langage

public void SendException() throws NullPointerException

{

throw new NullPointerException();

}

// FaultException

public void generateFaultException() throws org.apache.axis.AxisFault

{

throw new org.apache.axis.AxisFault("My fault Exception");

}

}

* 1. Création d’un Web Service Java (Axis 2.1.1.1)

Nous allons créer un Service Web avec Axis 2.1.1.1. Avec cette version d’Axis on va utiliser le système de fichier *AAR* pour déployer notre Service Web.

Le fichier *AAR (Axis ARchive)* est une archive JAR spécifique pour le déploiement de Services Web sous Axis 2. Le fichier *AAR* est composé d’un fichier XML (*services.xml*) qui décrit le déploiement du Service Web et vos fichier *class.*

Avec le déploiement par le fichier *AAR*, la méthode d’encodage proposée au niveau du fichier WSDL est **Document/Literal**.

Nous avons enlevé certains types qui ont posé des problèmes au niveau de svcutil : *BigDecimal* et *BigInteger* . En effet, on ne peut pas échanger ces types au niveau du Service Web.

Pour ouvrir un fichier .*aar* vous pouvez par exemple renommer le fichier en .*rar* et l’ouvrir avec WinRar.

Le fichier *JavaServer2\_111.aar* contient un dossier *META-INF* dans lequel se trouve le fichier *services.xml*. De plus, le fichier *JavaServer2\_111.aar* contient aussi une arborescence en rapport avec le package dans lequel vous retrouverez votre classe.

Par exemple si votre classe *JavaServer2\_111* est dans le package *interopsoa.interopsimple*.

Cela donnera dans le fichier *JavaServer2\_111.aar :*

|  |
| --- |
| *3META-INF*  *149 services.xml*  *3interopsoa*  *3interopsimple*  *149 JavaServer2\_111.class*  *149 CustomType.class* |

Pour obtenir le fichier *JavaServer2\_111.class* , il faut utiliser la commande ***javac*** afin de compiler votre fichier source *JavaServer2\_111.java.*

*Remarque :* Dans le fichier *services.xml*, il faut mettre **TOUTES** les fonctions qu’il y a dans votre classe JavaServer2\_111, sinon le service web ne fonctionnera pas !

Pour déployer le service web, il faut mettre le fichier *JavaServer2\_111.aar* dans le répertoire

C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\axis2\WEB-INF\services

Puis pour atteindre le fichier WSDL il faut aller à l’adresse suivante :

<http://localhost:8181/axis2/services/JavaServer2_111?wsdl>

***Fichier : JavaServer2\_111.java***

package interopsoa.interopsimple;

import java.lang.\*;

import java.math.\*;

import java.util.\*;

public class JavaServer2\_111{

public JavaServer2\_111(){}

// Simples types and wrappers ; ignore operation overflows and loss of precisions if any

public boolean isItTrue(Boolean lhs, Boolean rhs) {

return lhs.booleanValue() && rhs.booleanValue(); }

public byte addByte(Byte lhs, Byte rhs) {

return (byte)(lhs.byteValue() + rhs.byteValue()); }

public double addDouble(Double lhs, Double rhs) {

return lhs.doubleValue() + rhs.doubleValue(); }

public float addFloat(Float lhs, Float rhs) {

return lhs.floatValue() + rhs.floatValue(); }

public int addInt(Integer lhs, Integer rhs) { return lhs.intValue() + rhs.intValue(); }

public long addLong(Long lhs, Long rhs) { return lhs.longValue() + rhs.longValue(); }

public short addShort(Short lhs, Short rhs) { return (short)(lhs.shortValue() + rhs.shortValue()); }

public Date getDate(Calendar d) { return d.getTime(); }

public String getMessage(String s) { return "Message is " + s; }

// custom type

public CustomType getCustomType() { return new CustomType(); }

// some arrays

public int[] getIntegers() { return new int[] { 1, 2, 3 }; }

public String[] getStrings() { return new String[] { "Robert", "Dupont", "Dupond" }; }

public Long[] getLongs() { return new Long[] { new Long(1), new Long(2), new Long(3) }; }

// FaultException

public void generateFaultException() throws org.apache.axis2.AxisFault

{ throw new org.apache.axis2.AxisFault("My fault Exception");}

// Exception propre au langage

public void SendException() throws NullPointerException

{ throw new NullPointerException(); }

// MySpecificException

public void SendMyException() throws MySpecificException

{ throw new MySpecificException(); }

}

***Fichier : CustomType.java***

package interopsoa.interopsimple;

public class CustomType

{

private String \_name = "TestName";

public CustomType() { }

public String getName() { return \_name; }

public void setName(String name) { \_name = name; }

}

***Fichier : MySpecificException.java***

package interopsoa.interopsimple;

public class MySpecificException extends Exception {

private String \_name = "Test My Exception : Boom";

public String getName() { return \_name; }

public void setName(String name) { \_name = name; }

}

***Fichier : services.xml***

<service name="JavaServer2\_111">

<description>

This service is JavaServer2\_111 With Axis 2.1.1.1

</description>

<parameter name="ServiceClass" locked="false">interopsoa.interopsimple.JavaServer2\_111</parameter>

<operation name="getStrings">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="isItTrue">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="addByte">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="addDouble">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="addFloat">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="addInt">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="addLong">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="addShort">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="getIntegers">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="getLongs">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="SendException">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="generateFaultException">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="SendMyException">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="getMessage">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="getDate">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

<operation name="getCustomType">

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />

</operation>

</service>

* 1. Création d’un Web Service Java (GlassFish v2)

Nous allons créer un service web avec GlassFish V2 en utilisant l’IDE Netbeans.

Pour installer le serveur GlassFish v2 dans l’IDE NetBeans consulter [ce lien](http://argentera.inria.fr/wiki/data/Main/Tp1EjbMbds.jsp) ainsi que [celui-ci](https://glassfish.dev.java.net/public/netbeans/).

Pour suivre un tutoriel sur l’utilisation de NetBeans et GlassFish consulter le labs à cette [adresse](http://www.javapassion.com/handsonlabs/3400_javaee5basics.zip).

Par défaut le serveur d’application GlassFish propose les services web au format SOAP 1.1. Pour que les services web soient au format SOAP 1.2, il faut rajouter le tag suivant au dessus de la classe :

@BindingType(value="http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP/")

***Fichier : OperationSurTypesPrimitifs.java***

package InteropSOA.InteropSimple.ServeurJava;

import javax.jws.WebMethod;

import javax.jws.WebService;

import javax.xml.ws.BindingType;

**@WebService()**

public class OperationSurTypesPrimitifs {

public String sayHello(String in\_name){

return "Hello " + in\_name + " !";

}

// Simples types and wrappers ; ignore operation overflows and loss of precisions if any

public boolean isItTrue(Boolean lhs, Boolean rhs)

{

return lhs.booleanValue() && rhs.booleanValue();

}

public byte addByte(Byte lhs, Byte rhs)

{

return (byte)(lhs.byteValue() + rhs.byteValue());

}

public double addDouble(Double lhs, Double rhs)

{

return lhs.doubleValue() + rhs.doubleValue();

}

public float addFloat(Float lhs, Float rhs)

{

return lhs.floatValue() + rhs.floatValue();

}

public int addInt(Integer lhs, Integer rhs)

{

return lhs.intValue() + rhs.intValue();

}

public long addLong(Long lhs, Long rhs)

{

return lhs.longValue() + rhs.longValue();

}

public short addShort(Short lhs, Short rhs)

{

return (short)(lhs.shortValue() + rhs.shortValue());

}

}

***Fichier : OperationSurTableaux.java***

package InteropSOA.InteropSimple.ServeurJava;

import javax.jws.WebService;

import javax.xml.ws.BindingType;

@WebService()

public class OperationSurTableaux {

// Some arrays

public int[] getIntegers()

{

return new int[] { 1, 2, 3 };

}

public String[] getStrings()

{

return new String[] { "Robert", "Dupont", "Dupond" };

}

public Long[] getLongs()

{

return new Long[] { new Long(1), new Long(2), new Long(3) };

}

}

***Fichier : OperationSurTypesComplexes.java***

package InteropSOA.InteropSimple.ServeurJava;

import java.math.BigDecimal;

import java.math.BigInteger;

import java.util.Calendar;

import java.util.Date;

import javax.jws.WebService;

import javax.xml.ws.BindingType;

@WebService()

public class OperationSurTypesComplexes {

// Java standard types

public BigDecimal addBigDecimal(BigDecimal lhs, BigDecimal rhs)

{

return lhs.add(rhs);

}

public BigInteger addBigInteger(BigInteger lhs, BigInteger rhs)

{

return lhs.add(rhs);

}

public Date getDate(Calendar d)

{

return d.getTime();

}

public String getMessage(String s)

{

return "Message is " + s;

}

// Custom type

public CustomType getCustomType()

{

return new CustomType();

}

}

***Fichier : CustomType.java***

package InteropSOA.InteropSimple.ServeurJava;

public class CustomType {

private String \_name = "TestName";

public String getName() { return \_name; }

public void setName(String name) { \_name = name; }

}

***Fichier : OperationAvecExceptions.java***

package InteropSOA.InteropSimple.ServeurJava;

import com.sun.xml.rpc.wsdl.document.soap.SOAPFault;

import javax.jws.WebService;

import javax.xml.namespace.QName;

import javax.xml.soap.SOAPException;

import javax.xml.soap.SOAPFactory;

import javax.xml.ws.WebServiceException;

import javax.xml.ws.soap.SOAPFaultException;

import javax.xml.ws.FaultAction;

import javax.xml.ws.BindingType;

@WebService()

public class OperationAvecExceptions {

@FaultAction(className=javax.xml.ws.WebServiceException.class )

public void SendFaultException() throws javax.xml.ws.WebServiceException

{

throw new WebServiceException("Une Fault Exception");

}

@FaultAction(className=NullPointerException.class )

public void SendException() throws NullPointerException

{

throw new NullPointerException();

}

@FaultAction(className=MySpecificException.class)

public void SendMyException() throws MySpecificException

{

throw new MySpecificException();

}

public void sendNotFaultedException()

{

throw new RuntimeException("Boom Boom");

}

}

***Fichier : MySpecificException.java***

package InteropSOA.InteropSimple.ServeurJava;

public class MySpecificException extends Exception {

private String \_name = "TestName";

public String getName() { return \_name; }

public void setName(String name) { \_name = name; }

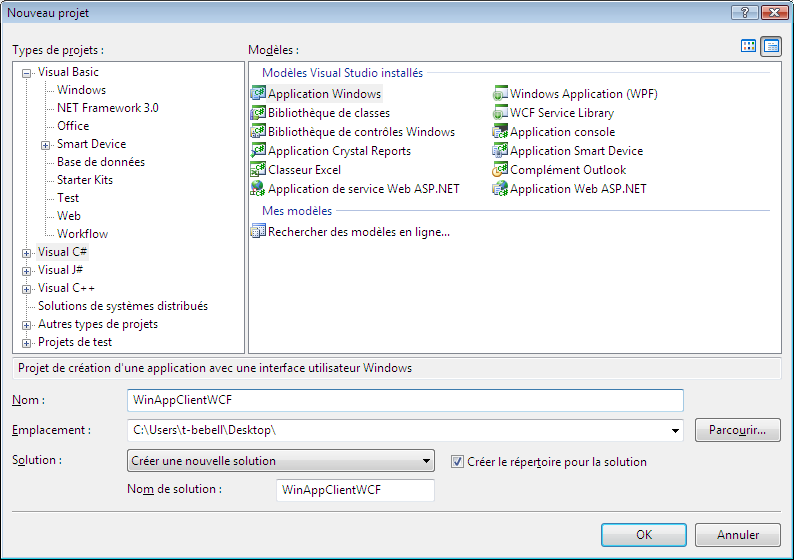
}

* 1. .Net 3.0 consomme des web services Java

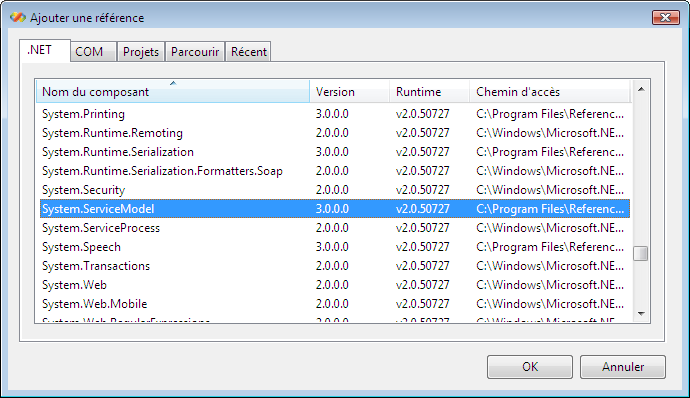
Nous allons maintenant créer un client .Net qui va se connecter à ces différents Services Web :

* + - Services Web sous Java Axis 1.4
    - Services Web sous Java Axis 2.1.1.1
    - Services Web sous GlassFish v2

Tout d’abord il faut créer un nouveau projet de type Application Windows :



Faire clic droit sur le projet, puis *Ajouter une référence*. Il faut ajouter une référence sur ***System.ServiceModel***.

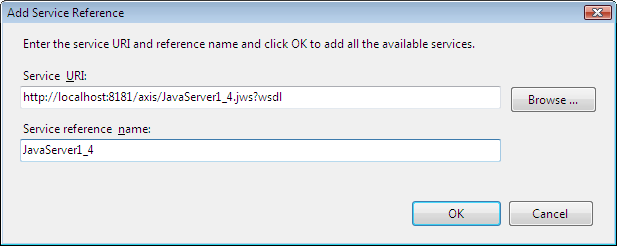


**Création du Proxy pour Axis 1.4 :**

Faire clic droit sur le projet, puis *Add Service Reference*.

*Service URL* = <http://localhost:8181/axis/JavaServer1_4.jws?wsdl>

*Service* *Reference* Name = JavaServer1\_4



**Création du Proxy pour Axis 2.1.1.1 :**

Faire clic droit sur le projet, puis *Add Service Reference*.

*Service URL* = <http://localhost:8181/axis2/services/JavaServer2_111?wsdl>

*Service* *Reference* Name = JavaServer2\_111

**Création du Proxy pour GlassFish V2 :**

Faire clic droit sur le projet, puis *Add Service Reference*.

*Service URL* = <http://localhost:8080/MyHelloWorld/OperationSurTypesPrimitifsService>

*Service* *Reference* Name = JavaServerGlassfish\_OperationSurTypesPrimitifs

*Service URL* = <http://localhost:8080/MyHelloWorld/OperationSurTypesComplexesService>

*Service* *Reference* Name = JavaServerGlassfish\_OperationSurTypesComplexes

*Service URL* = <http://localhost:8080/MyHelloWorld/OperationSurTableauxService>

*Service* *Reference* Name = JavaServerGlassfish\_OperationSurTableaux

*Service URL* = <http://t-bebell5:8080/MyHelloWorld/OperationAvecExceptionsService>

*Service* *Reference* Name = JavaServerGlassfish\_OperationAvecExceptions

Visual Studio utilise *svcutil* quand on lui demande une référence vers un service. Il lance svcutil pour générer le proxy. Il va aussi vous modifier le fichier *App.Config* en rajoutant la partie client, ainsi que le binding.

***VS 2005 : Utilisation de svcutil sur le service Axis 2.1.1.1***

C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\**svcutil**.exe /noLogo /s /ct:"System.ComponentModel.BindingList`1,System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" /d:"C:\Users\t-bebell\AppData\Local\Temp\cw1uwt5g.jur" /config:"C:\Users\t-bebell\AppData\Local\Temp\cw1uwt5g.jur\newapp.config" /mergeConfig /out:"JavaServer2\_111.cs" /language:csharp /n:\*,WinAppClientWCF.JavaServer2\_111 "http://localhost:8181/axis2/services/JavaServer2\_111?wsdl"

Attempting to download metadata from 'http://localhost:8181/axis2/services/JavaServer2\_111?wsdl' using WS-Metadata Exchange or DISCO.

Warning: L'‚l‚ment d'extension WSDL facultatif ®ÿbodyÿ¯ de l'espace de noms ®ÿhttp://schemas.xmlsoap.org/wsdl/soap/ÿ¯ n'a pas ‚t‚ trait‚.

XPathÿ: //wsdl:definitions[@targetNamespace='http://interopsimple.interopsoa']/wsdl:binding[@name='JavaServer2\_111SOAP11Binding']/wsdl:operation[@name='generateException']/wsdl:fault[@name='generateExceptionFault']

***Extrait : App.Config (Clients Axis)***

<client>

<endpoint address="http://localhost:8181/axis/JavaServer1\_4.jws"

binding="basicHttpBinding"

bindingConfiguration="JavaServer1\_4SoapBinding"

contract="WinAppClientWCF.JavaServer1\_4.JavaServer1\_4"

name="**JavaServer1\_4**" />

<endpoint address=<http://localhost:8181/axis2/services/JavaServer2_111>

binding="basicHttpBinding"

bindingConfiguration="JavaServer2\_111SOAP11Binding"

contract="WinAppClientWCF.JavaServer2\_111.JavaServer2\_111PortType"

name="JavaServer2\_111SOAP11port\_http" />

<endpoint address="http://localhost:8181/axis2/services/JavaServer2\_111"

binding="customBinding"

bindingConfiguration="JavaServer2\_111SOAP12Binding"

contract="WinAppClientWCF.JavaServer2\_111.JavaServer2\_111PortType"

name="JavaServer2\_111SOAP12port\_http" />

</client>

Vous pouvez aussi utiliser *svcutil* manuellement en lui précisant chacun de ses paramètres.

Vous trouverez l’exécutable à cet emplacement : C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin

Vous pouvez aller voir le site c2i, il y a un exemple d’utilisation manuelle de svcutil : <http://www.c2i.fr/code.aspx?IDCode=667>

Quelques paramètres importants :

* */language:c#* : Permet de choisir le type de langage qui sera généré pour le proxy.
* */config:app.config* : Permet de donner le nom du fichier *App.Config* qui sera généré.
* */dconly* : Génère le proxy uniquement sur les données (DataContract) et pas sur les services.
* */mc*: Génère le proxy uniquement sur les messages (MessageContract).

On a vu dans la page précédente que le nom du EndPoint d’Axis 1.4 dans le fichier *App.Config* est **JavaServer1\_4***.* On va maintenant créer le code du client qui va se connecter à ce EndPoint.

***Code C# : Clients WCF d’un Service Axis 1.4 (SOAP 1.1)***

using System.ServiceModel;

using WinAppClientWCF.JavaServer1\_4;

private void callAxis1\_4(object sender, EventArgs e)

{

JavaServer1\_4.JavaServer1\_4Client client2 = new

JavaServer1\_4.JavaServer1\_4Client("**JavaServer1\_4**");

// simple types and wrappers

lblRes2.Text += "\nAdding big decimal :" + client2.addBigDecimal(5, 6);

lblRes2.Text += "\nAdding big integer :" + client2.addBigInteger("5", "6");

lblRes2.Text += "\nAdding byte :" + client2.addByte(5, 6);

lblRes2.Text += "\nAdding double :" + client2.addDouble(5, 6);

lblRes2.Text += "\nAdding float :" + client2.addFloat(5, 6);

lblRes2.Text += "\nAdding int :" + client2.addInt(5, 6);

lblRes2.Text += "\nAdding long :" + client2.addLong(5, 6);

lblRes2.Text += "\nAdding short :" + client2.addShort(5, 6);

lblRes2.Text += "\nAdding boolean :" + client2.isItTrue(true, false);

// java standard types

DateTime d = client2.getDate(DateTime.Now);

lblRes2.Text += "\nDate :" + d;

lblRes2.Text += "\nString :" + client2.getMessage("INTEROP");

// some arrays

int[] iArray = client2.getIntegers();

for(int i = 0; i < iArray.Length; ++i) lblRes2.Text += "\nInteger Array [" + i + "] :" + iArray[i];

string[] sArray = client2.getStrings();

for(int i = 0; i < iArray.Length; ++i) lblRes2.Text += "\nString Array [" + i + "] :" + sArray[i];

long[] lArray = client2.getLongs();

for(int i = 0; i < lArray.Length; ++i) lblRes2.Text += "\nLong Array [" + i + "] :" + lArray[i];

// Exception

Try {client2.generateFaultException();}

catch (FaultException ex)

{lblRes2.Text += "\nFaultException (FaultException): " + ex.Message;}

catch (System.Exception ex){lblRes2.Text += "\n FaultException (Exception): " + ex.Message;}

try{client2.SendException();}

catch (FaultException ex)

{lblRes2.Text += "\nGenerated SendException (FaultException): " + ex.Message;}

catch (System.Exception ex){lblRes2.Text += "\nGenerated SendException (Exception): " + ex.Message;}

*SvcUtil* a généré deux EndPoints pour le proxy d’Axis 2.1.1.1. Dans le fichier *App.Config* on trouve deux Endpoints : JavaServer2\_111SOAP11port\_http et JavaServer2\_111SOAP12port\_http.

Le premier permet de se connecter en SOAP 1.1 et le deuxieme en SOAP 1.2.

On va maintenant créer le code du client qui va se connecter à ce service en SOAP 1.2 :

***Extrait du Code C# : Clients WCF d’un Service Axis 2.1.1.1 (SOAP 1.2)***

using System.ServiceModel;

using WinAppClientWCF.JavaServer2\_111;

private void callAxis2\_111(object sender, EventArgs e)

{

JavaServer2\_111.JavaServer2\_111PortTypeClient client = new JavaServer2\_111.JavaServer2\_111PortTypeClient("JavaServer2\_111SOAP12port\_http");

JavaServer2\_111.addInt ai = new JavaServer2\_111.addInt();

ai.param0 = 5;

ai.param1 = 6;

JavaServer2\_111.addIntResponse air = client.addInt(ai);

lblAxis2.Text += "\nADD INT : " + air.@return.ToString();

}

*Remarque :* Quand svcutil génère le proxy, il y a une erreur de double définition dans le code généré. Pour résoudre le problème, il suffit de commenter le code suivant :

***Code C# à commenter : Proxy généré par svcutil (Services Axis 2.1.1.1)***

[System.CodeDom.Compiler.GeneratedCodeAttribute("svcutil", "3.0.4506.30")]

[System.SerializableAttribute()]

[System.Diagnostics.DebuggerStepThroughAttribute()]

[System.ComponentModel.DesignerCategoryAttribute("code")] [System.Xml.Serialization.XmlTypeAttribute(Namespace="http://interopsimple.interopsoa/xsd")]

public partial class CustomType

{

private string nameField;

[System.Xml.Serialization.XmlElementAttribute(IsNullable=true, Order=0)]

public string name

{

get

{

return this.nameField;

}

set

{

this.nameField = value;

}

}

}

\*/

* 1. .Net 3.0 consomme des web services GlassFish V2

Voici ci-dessous le code qui permet de consommer les services web d’un serveur d’application GlassFish.

***Extrait du Code C# : Clients WCF d’un Service Axis 2.1.1.1 (SOAP 1.2)***

using WinAppClientWCF.JavaServerGlassfish\_OperationSurTypesPrimitifs;

using WinAppClientWCF.JavaServerGlassfish\_OperationSurTypesComplexes;

using WinAppClientWCF.JavaServerGlassfish\_OperationSurTableaux;

using WinAppClientWCF.JavaServerGlassfish\_OperationAvecExceptions;

#region Glassfish v2

private void btGoGlassFish\_Click(object sender, EventArgs e)

{

// 1) Simples types and wrappers

lblGlassfish.Text = "OperationSurTypesPrimitifs :";

OperationSurTypesPrimitifsClient client = new OperationSurTypesPrimitifsClient("OperationSurTypesPrimitifsPort");

lblGlassfish.Text += "\nAdding byte :" + client.addByte(5, 6);

lblGlassfish.Text += "\nAdding double :" + client.addDouble(5, 6);

lblGlassfish.Text += "\nAdding float :" + client.addFloat(5, 6);

lblGlassfish.Text += "\nAdding int :" + client.addInt(5, 6);

lblGlassfish.Text += "\nAdding long :" + client.addLong(5, 6);

lblGlassfish.Text += "\nAdding short :" + client.addShort(5, 6);

lblGlassfish.Text += "\nAdding boolean :" + client.isItTrue(true, false);

// 2) Java standard types

lblGlassfish.Text += "\n\nOperationSurTypesComplexes :";

OperationSurTypesComplexesClient client2 = new OperationSurTypesComplexesClient("OperationSurTypesComplexesPort");

lblGlassfish.Text += "\nAdding big decimal :" + client2.addBigDecimal(5, 6);

lblGlassfish.Text += "\nAdding big integer :" + client2.addBigInteger("5", "6");

DateTime d = client2.getDate(DateTime.Now);

lblGlassfish.Text += "\nDate :" + d;

lblGlassfish.Text += "\nString :" + client2.getMessage("INTEROP");

// Custom type

JavaServerGlassfish\_OperationSurTypesComplexes.customType t = client2.getCustomType();

lblGlassfish.Text += "\nCustom type :" + t.name;

// 3) Some arrays

lblGlassfish.Text += "\n\nOperationSurTableaux :";

OperationSurTableauxClient client3 = new OperationSurTableauxClient("OperationSurTableauxPort");

int[] iArray = client3.getIntegers();

for (int i = 0; i < iArray.Length; ++i) lblGlassfish.Text += "\nInteger Array [" + i + "] :" + iArray[i];

string[] sArray = client3.getStrings();

for (int i = 0; i < iArray.Length; ++i) lblGlassfish.Text += "\nString Array [" + i + "] :" + sArray[i];

long[] lArray = client3.getLongs();

for (int i = 0; i < lArray.Length; ++i) lblGlassfish.Text += "\nLong Array [" + i + "] :" + lArray[i];

// 4) Exception

lblGlassfish.Text += "\n\nOperationAvecExceptions :";

OperationAvecExceptionsClient client4 = new OperationAvecExceptionsClient("OperationAvecExceptionsPort");

try

{

client4.SendFaultException();

}

catch (FaultException ex)

{

lblGlassfish.Text += "\nSendFaultException (FaultException): " + ex.Message;

}

catch (System.Exception ex)

{

lblGlassfish.Text += "\nSendFaultException (Exception): " + ex.Message;

}

try

{

client4.SendException();

}

catch (FaultException ex)

{

lblGlassfish.Text += "\nSendException (FaultException): " + ex.Message;

}

try

{

client4.SendMyException();

}

catch (FaultException ex)

{

lblGlassfish.Text += "\nSendMyException (FaultException): " + ex.Message;

}

try

{

client4.sendNotFaultedException();

}

catch (System.Exception ex)

{

lblGlassfish.Text += "\nSendNotFaultedException : " + ex.Message;

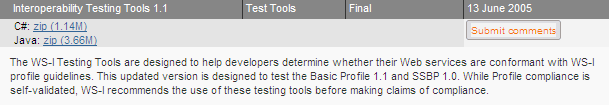
}

}

1. Tester la conformité WS-I

L’organisme WS-I propose deux outils, le premier est Monitor et permet capturer les messages SOAP entre le client et le serveur. Le deuxième Validator génère un rapport de conformité du contrat WSDL et des échanges SOAP.

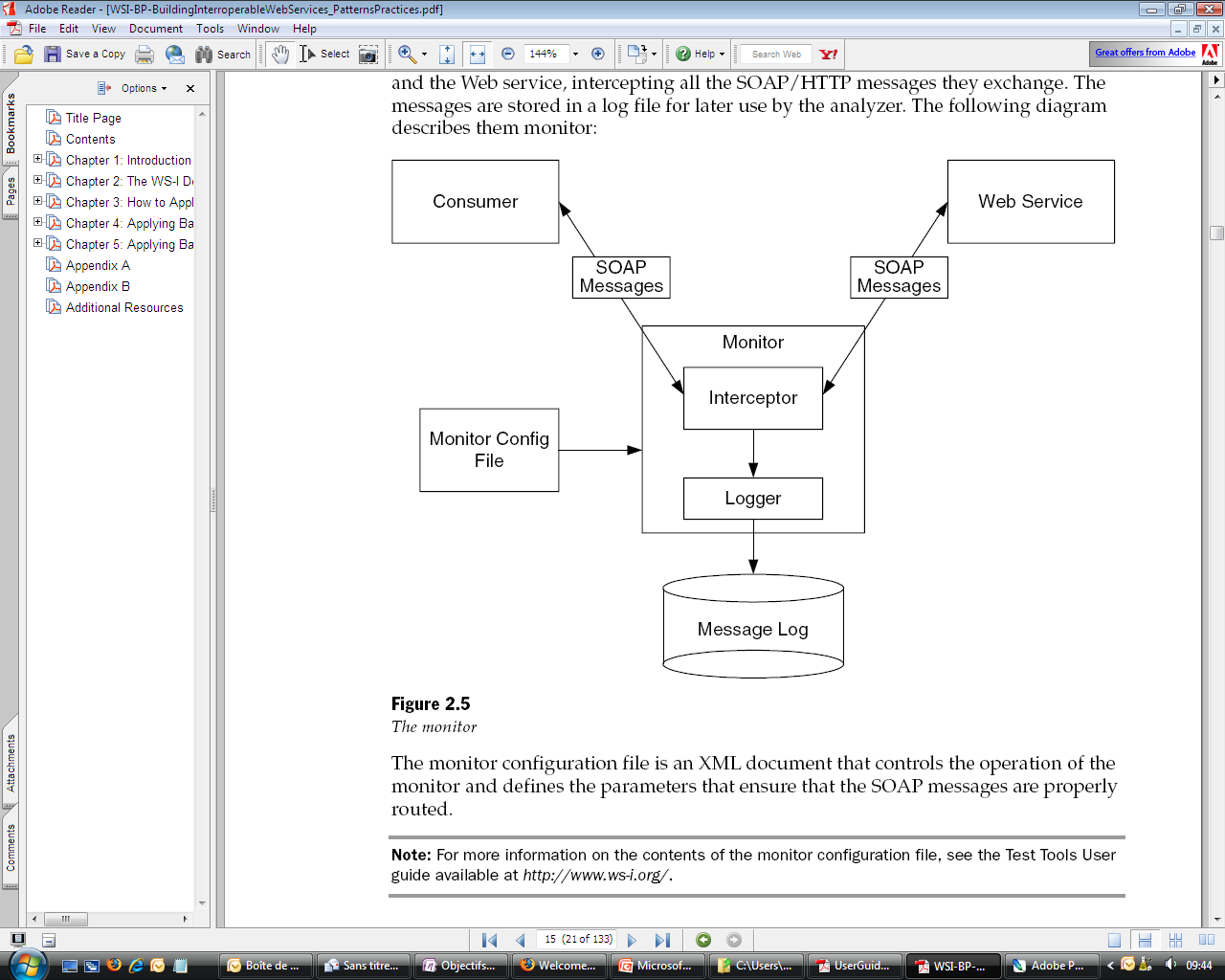
Ces outils sont disponibles sur le site du WS-I à cette [adresse](http://ws-i.org/deliverables/workinggroup.aspx?wg=testingtools).

[](http://www.ws-i.org/Testing/Tools/2005/06/WSI_Test_Java_Final_1.1.zip)

1. Monitor
2. Utilisation de l’outil Monitor

Récupérer la version Java des outils et décompresser le fichier *WSI\_Test\_Java\_Final\_1.1.zip* dans le répertoire *C:\wsi-test-tools-Java,* ce qui donne *C:\wsi-test-tools-Java\java\bin\Monitor.bat.*

L’outil Monitor est un proxy, ce qui signifie qu’il écoute sur un port d’entrée X et qu’il retransmet les demandes sur ce port X sur un autre port de sortie Y.



*Fonctionnement de l’outil Monitor*

Cet outil utilise le fichier *C:\wsi-test-tools-Java\java\bin\monitorConfig.xml* pour paramétrer les ports d’entrée et de sortie.

Dans notre exemple le client WCF va envoyer une demande sur le port **9000** et le serveur WCF écoute sur le port **5000**. L’outil va écrire l’ensemble des traces SOAP dans le fichier ***log.xml****.*

***Fichier : monitorConfig.xml***

<?xml version="1.0" encoding="utf-8" ?>

<wsi-monConfig:configuration xmlns:wsi-monConfig="http://www.ws-i.org/testing/2003/03/monitorConfig/">

<wsi-monConfig:comment>

This configuration file is used to test the WS-I sample applications running on a single system.

</wsi-monConfig:comment>

<wsi-monConfig:logFile replace="true" location="log.xml">

<wsi-monConfig:addStyleSheet href="../../common/xsl/log.xsl" type="text/xsl"/>

</wsi-monConfig:logFile>

<wsi-monConfig:logDuration>600</wsi-monConfig:logDuration>

<wsi-monConfig:cleanupTimeoutSeconds>3</wsi-monConfig:cleanupTimeoutSeconds>

<wsi-monConfig:manInTheMiddle>

<wsi-monConfig:redirect>

<wsi-monConfig:comment>This is a redirect example for local Retailer Sample App on port 8282.</wsi-monConfig:comment>

<wsi-monConfig:listenPort>9000</wsi-monConfig:listenPort>

<wsi-monConfig:schemeAndHostPort>http://localhost:5000</wsi-monConfig:schemeAndHostPort>

<wsi-monConfig:maxConnections>1000</wsi-monConfig:maxConnections>

<wsi-monConfig:readTimeoutSeconds>5</wsi-monConfig:readTimeoutSeconds>

</wsi-monConfig:redirect>

</wsi-monConfig:manInTheMiddle>

</wsi-monConfig:configuration>

Pour exécuter Monitor, il faut lancer une commande DOS :

**CD** C:\wsi-test-tools-Java\java\bin\

**Monitor.bat** -config C:\wsi-test-tools-Java\java\bin\monitorConfig.xml

Il n’y a pas besoin de modifier le serveur, il faut uniquement modifier le fichier *app.config* du client WCF.

Le endpoint du client est à l’origine :

<endpoint address=[http://localhost:**5000**/OperationBasique/soap11](http://localhost:5000/OperationBasique/soap11) … />

Il faut le modifier de la façon suivante :

<endpoint address=[http://localhost:**9000**/OperationBasique/soap11](http://localhost:9000/OperationBasique/soap11) … />

Il ne reste plus qu’à lancer le serveur puis le client. Enfin quand toutes les demandes aux services web ont été effectuées, le Monitor peut être arrêté en tapant exit dans l’invite de commande dos.

Le fichier de log se situe à cet emplacement : *C:\wsi-test-tools-Java\java\bin\log.xml*

1. Exemple de trace SOAP avec Monitor

**WS-I Testing Tools Log File**

**Monitor Configuration Information**

|  |  |  |  |
| --- | --- | --- | --- |
| **Listen Port** | **Scheme And Host/Port** | **Maximum Connections** | **Read Timeout Seconds** |
| 9000 | http://localhost:5000 | 1000 | 5 |

**Message Entry: 1 (Request)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conversation ID** | **Message Type** | **Timestamp** | **Sender Host and Port** | **Receiver Host and Port** |
| 1 | request | 2007-04-27T10:48:38.589 | 0:0:0:0:0:0:0:1:50156 | localhost:5000 |

**HTTP Headers:**

POST /OperationBasique/soap11 HTTP/1.1

Content-Type: text/xml; charset=utf-8

SOAPAction: "http://www.OperationBasiques.com/IOperationBasique/ConcatNames"

Host: localhost:9000

Content-Length: 230

Expect: 100-continue

Connection: Keep-Alive

**Message Content:**

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">

<s:Body>

<ConcatNames xmlns="http://www.OperationBasiques.com">

<in\_firstName>Robert</in\_firstName>

<in\_lastName>Dupont</in\_lastName>

</ConcatNames>

</s:Body>

</s:Envelope>

**Message Entry: 2 (Response)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conversation ID** | **Message Type** | **Timestamp** | **Sender Host and Port** | **Receiver Host and Port** |
| 1 | response | 2007-04-27T10:48:38.628 | localhost:5000 | 0:0:0:0:0:0:0:1:50156 |

**HTTP Headers:**

HTTP/1.1 200 OK

Content-Length: 443

Content-Type: text/xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Fri, 27 Apr 2007 08:48:38 GMT

**Message Content:**

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">

<s:Header>

<ActivityId CorrelationId="45ce3e90-a46a-4b24-affe-61f13a53fd30" xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">7fa90b33-3f5c-4c58-8f0f-b73c9dbb193e</ActivityId>

</s:Header>

<s:Body>

<ConcatNamesResponse xmlns="http://www.OperationBasiques.com">

<ConcatNamesResult>Bonjour Robert Dupont</ConcatNamesResult>

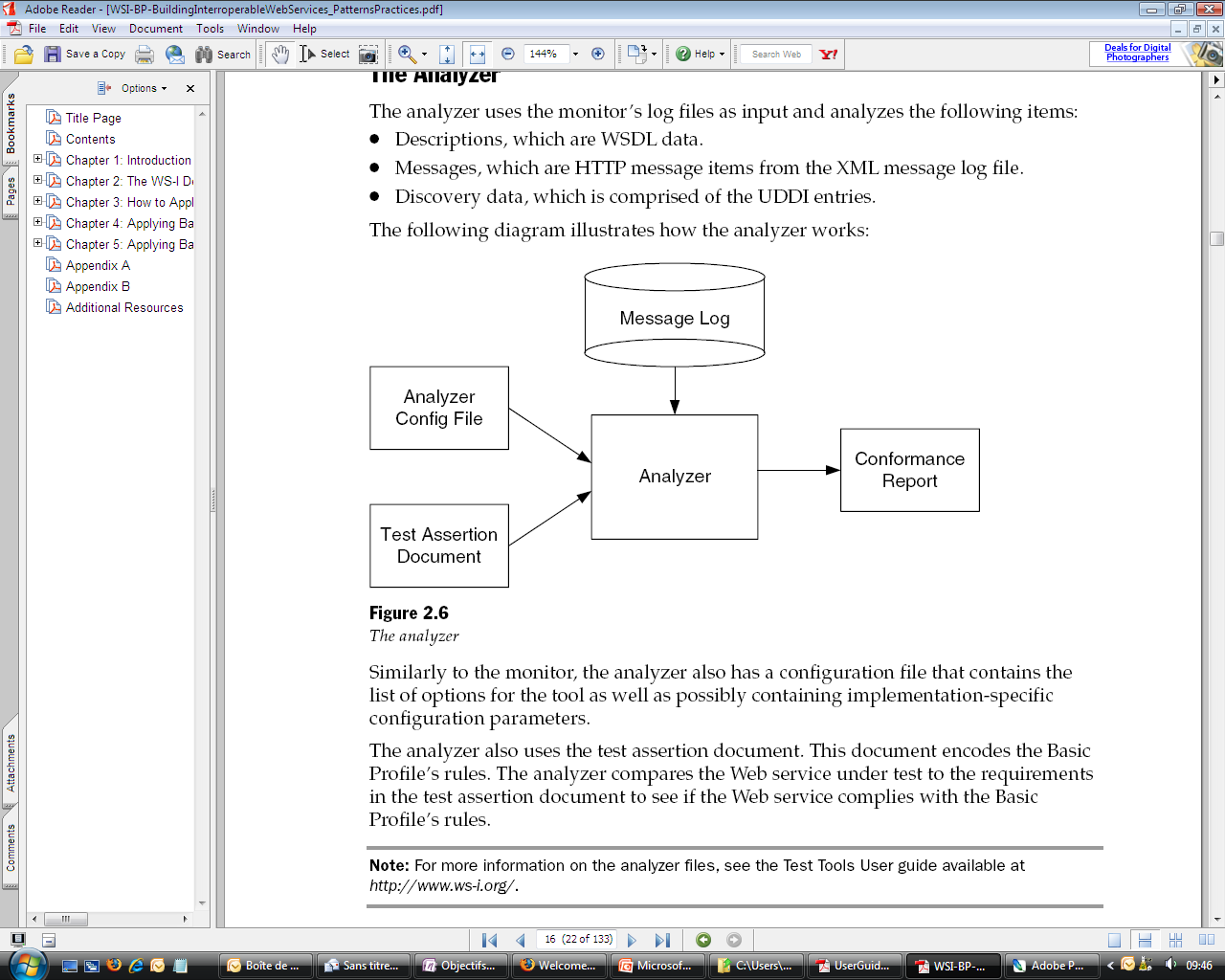
</ConcatNamesResponse>

</s:Body>

</s:Envelope>

1. Validator
2. Utilisation de l’outil Validator

L’outil Validator génère un fichier de rapport de la conformité du contrat WSDL et des échanges SOAP en regard des spécifications Basic Profile 1.1 et SSBP 1.0 qui lui sont transmis. Cet outil a besoin de connaître l’adresse du contrat WSDL et le fichier *log.xml* qui a été généré précédemment avec l’outil Monitor.



*Fonctionnement de l’outil Validator*

Cet outil utilise le fichier *C:\wsi-test-tools-Java\java\bin\analyzerConfig.xml* pour paramétrer les ports d’entrée et de sortie. Pour bien comprendre d’où proviennent les champs dans le fichier de configuration, vous trouverez ci-dessous le contrat WSDL du service.

***Fichier : http://localhost:5000/OperationBasique?wsdl***

<wsdl:definitions name="OperationBasique" targetNamespace="http://tempuri.org/">

<wsdl:import namespace="http://www.OperationBasiques.com" location="http://localhost:5000/OperationBasique?wsdl=wsdl0"/>

<wsdl:types/>

<wsdl:binding name="EndPointSOAP1\_1" type="i0:IOperationBasique">

<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="ConcatNames">

<soap:operation soapAction="http://www.OperationBasiques.com/IOperationBasique/ConcatNames" style="document"/>

<wsdl:input>

<soap:body use="literal"/>

</wsdl:input>

<wsdl:output>

<soap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

<wsdl:service name="OperationBasique">

<wsdl:port name="EndPointSOAP1\_1" binding="tns:EndPointSOAP1\_1">

<soap:address location="http://localhost:5000/OperationBasique/soap11"/>

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

***Fichier : analyzerConfig.xml***

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

<wsi-analyzerConfig:configuration name="Sample Basic Profile Analyzer Configuration"

xmlns:wsi-analyzerConfig="http://www.ws-i.org/testing/2004/07/analyzerConfig/">

<wsi-analyzerConfig:description>

This file contains a sample of the configuration file for

the Basic Profile Analyzer, which can be used with the

other sample files.

</wsi-analyzerConfig:description>

<wsi-analyzerConfig:verbose>true</wsi-analyzerConfig:verbose>

<wsi-analyzerConfig:assertionResults type="all" messageEntry="true" failureMessage="true"/>

<wsi-analyzerConfig:reportFile replace="true" location="report.xml">

<wsi-analyzerConfig:addStyleSheet href="../../common/xsl/report.xsl" type="text/xsl"/>

</wsi-analyzerConfig:reportFile>

<wsi-analyzerConfig:testAssertionsFile>../../common/profiles/SSBP10\_BP11\_TAD.xml</wsi-analyzerConfig:testAssertionsFile>

<wsi-analyzerConfig:logFile correlationType="endpoint">log.xml</wsi-analyzerConfig:logFile>

<wsi-analyzerConfig:wsdlReference>

<wsi-analyzerConfig:wsdlElement type="**binding**" namespace="http://tempuri.org/">

EndPointSOAP1\_1

</wsi-analyzerConfig:wsdlElement>

<wsi-analyzerConfig:wsdlURI>

http://localhost:5000/OperationBasique?wsdl

</wsi-analyzerConfig:wsdlURI>

</wsi-analyzerConfig:wsdlReference>

</wsi-analyzerConfig:configuration>

Il faut que le serveur fonctionne pour que le contrat WSDL puisse être consulté par le Validator.

Pour exécuter Validator, il faut lancer une commande DOS :

**CD** C:\wsi-test-tools-Java\java\bin\

**Analyzer.bat** -config analyzerConfig.xml

Le fichier de rapport se situe à cet emplacement : *C:\wsi-test-tools-Java\java\bin\report.xml*

1. Exemple de rapport avec Validator

**Analyzer Configuration Options**

|  |  |
| --- | --- |
| **Verbose** | true |
| **Type of Results to Show** | all |
| **Show Message Entry** | true |
| **Show Assertion Description** | false |
| **Show Failure Message** | true |
| **Show Failure Detail Message** | true |
| **Test Assertions Document** | ../../common/profiles/SSBP10\_BP11\_TAD.xml |
| **Message Log File** | log.xml |
| **Correlation Type** | endpoint |
| **WSDL Element** |  |
| **Name** | EndPointSOAP1\_1 |
| **Type** | binding |
| **Namespace** | http://tempuri.org/ |
| **Parent Element Name** | [Not specified] |
| **WSDL Document** | http://localhost:5000/OperationBasique?wsdl |

**Summary**

|  |  |
| --- | --- |
| **Result** | **passed** |

**Artifact Targets Analyzed:** The summary result applies to the following artifact targets which were specified in the analyzer configuration file.

|  |  |
| --- | --- |
| **Description** | binding=EndPointSOAP1\_1 |
| **Message** | log.xml |

**Assertion Result Summary**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Assertion ID** | **Passed** | **Failed** | **Prerequsite Failed** | **Warning** | **Not Applicable** | **Missing Input** |
| **BP2010** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2011** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2012** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2013** | 0 | 0 | 0 | 0 | **1** |  |
| **BP2014** | 0 | 0 | 0 | 0 | **2** |  |
| **BP2017** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2018** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2019** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2020** | 0 | 0 | 0 | 0 | **1** |  |
| **BP2021** | 0 | 0 | 0 | 0 | **1** |  |
| **BP2022** | 0 | 0 | 0 | 0 | **1** |  |
| **BP2032** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2034** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2098** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2101** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2102** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2103** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2104** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2105** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2107** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2108** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2110** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2111** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2112** | 0 | 0 | 0 | 0 | **1** |  |
| **BP2113** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2114** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2115** | **4** | 0 | 0 | 0 | 0 |  |
| **BP2116** | **4** | 0 | 0 | 0 | 0 |  |
| **BP2117** | 0 | 0 | 0 | 0 | **1** |  |
| **BP2118** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2119** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2120** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2122** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2123** | 0 | 0 | 0 | 0 | **2** |  |
| **BP2201** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2202** | **1** | 0 | 0 | 0 | **1** |  |
| **BP2208** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2402** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2404** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2406** | **1** | 0 | 0 | 0 | 0 |  |
| **BP2416** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2417** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2700** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2701** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2703** | **2** | 0 | 0 | 0 | 0 |  |
| **BP2803** | **1** | 0 | 0 | 0 | 0 |  |
| **BP4200** | **1** | 0 | 0 | 0 | **1** |  |
| **BP4201** | 0 | 0 | 0 | 0 | **2** |  |
| **BP4202** | 0 | 0 | 0 | 0 | **2** |  |
| **SSBP2209** | **1** | 0 | 0 | 0 | 0 |  |
| **SSBP2403** | **1** | 0 | 0 | 0 | 0 |  |

1. SYNTHESE sur les experimentations

Nous présentons ci-dessous une synthèse de nos expérimentations. Le détail est disponible dans les sous-parties de cette section.

**Parmi les types simples**, aucun soucis d’interopérabilité n’a été rencontré avec les types d’un langage qui sont décrits dans le contrat WSDL, à l’aide des types définis par le W3C. Par exemple, le type *String* de .Net et le type *java.lang.string* de Java correspondent en WSDL au type xs:*string* défini par le W3C.

*Remarque :* Les types *xs :…..* et *xsd :…..* font référence aux types définis par le [W3C](http://www.w3.org/TR/xmlschema-2/#built-in-datatypes), pour en savoir plus vous pouvez lire [l’article sur les schémas du W3C](http://www.dil.univ-mrs.fr/~massat/ens/xml/7-xsd.html).

*Attention***:** Le type *char* de.Net est représenté dans le fichier WSDL par un *integer*. Du côté du client Java, il faut convertir cet *integer* en *char* afin de récupérer le caractère. Le W3C n’a pas défini de type *xs :char*.

La majorité des autres types plus complexes ne font pas référence directement aux types définis par le W3C. Une structure plus complexe permet de les décrire dans le fichier WSDL. Cette structure est décrite dans le fichier WSDL directement par les types qui sont définis par le W3C. Ce qui permet à nouveau aux types d’être interopérables car les deux environnements se basent une définition pivot des types au niveau du W3C.

**Parmi les types complexes, le type DataSet** propre au langage .Net a posé des problèmes avec Axis 2.1.1.1 et JAX-WS 2.1.

*Remarque :* Le DataSet peut être vu comme une base de données puisque le DataSet contient des « DataTables » qui sont des tables de données avec des colonnes pour chaque attribut de la table.

Dans notre cas pour simplifier l’explication, on considère qu’il n’y a qu’une table dans le DataSet. La solution à ce problème est de définir une classe à l’image des données que l’on veut échanger par le service web. Ainsi, pour une table *Personne* contenant deux attributs : *nom* et *prénom*,on crée une classe *Personne* contenant deux attributs *nom* et *prénom.* Puis on remplit un tableau de classe personne contenant autant d’élément Personne qu’il y a de ligne dans le DataTable. Enfin c’est le tableau de classe Personne qui est transmis au niveau du service web.

Par ailleurs, **le type Structure** n’est pas compatible avec JAX-WS 2.1.

**En ce qui concerne les exceptions**, il est à noter que **l’interopérabilité entre le serveur de services WCF et le client utilisant un proxy Axis (1.4 et 2.1.1.1) a bien fonctionné**. En effet, les proxy Axis ont mappé correctement les exceptions WCF (le mappage consiste à créer une classe identique à celle décrite dans le contrat WSDL).

**En revanche, le proxy JAX-WS 2.1 n’a pas mappé correctement** les classes des exceptions car il a fait hériter les classes générées de la classe *Exception* au lieu de la classe *SoapFaultException* du langage Java. On peut traiter l’exception avec la classe générée mais on ne peut pas utiliser un try…catch de type *SoapFaultException*.

**De plus, avec un producteur JAX-WS 2.1, l’outil svcutil de WCF n’a pas su mapper** correctement les classes des exceptions (pas plus qu’un consommateur JAX-WS d’ailleurs).

Vous trouverez ci-après des tableaux récapitulatifs sur les types échangés.

Nous fournissons une archive de code source qui implémente l’ensemble des scénarios présentés.

* 1. Types échangés entre un serveur .Net et un client Java

Légende :

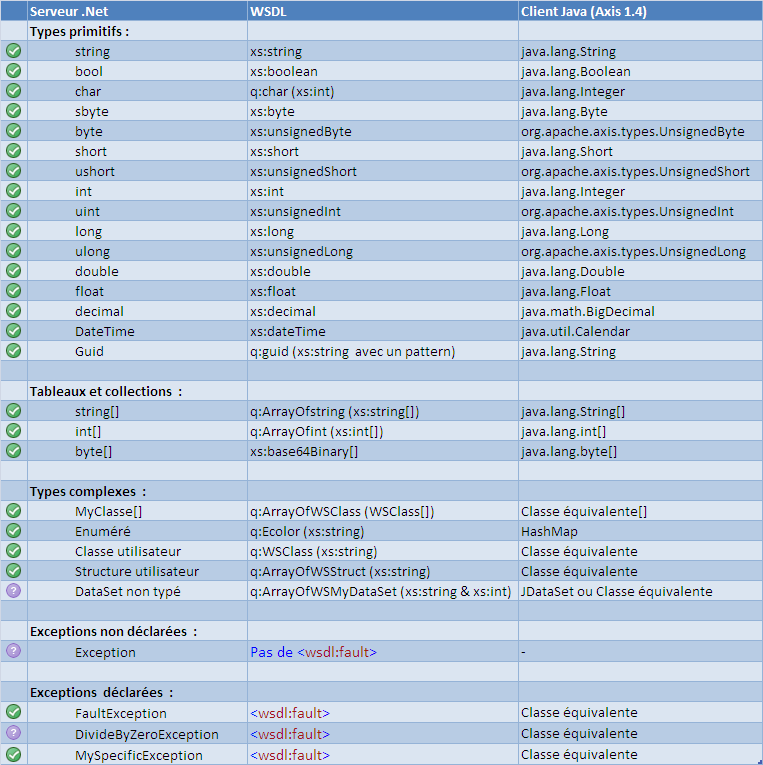
Les cas qui fonctionnent correctement (comportement attendu)

Les cas qui posent quelques problèmes mais n’empêchent pas le fonctionnement (solution de contournement, exécution dégradée)

Les cas qui ne fonctionnent pas (anomalie, comportement incorrect…)

interoWordLes cas qui ne font pas de sens (à l’encontre des spécifications ou des bonnes pratiques)

|  |  |
| --- | --- |
| *Producteur* : WCF  *Consommateur* : Java Axis 1.4 | *Version de SOAP* : SOAP 1.1  *Mode d’encodage* : Document / Literal Wrapped |

****

*Liste des correspondances entre le web service .Net et un client Java (Proxy Axis 1.4)*

Dans le cas de l’utilisation d’un proxy réalisé avec **Axis 2.1.1.1** tous les types utilisent un wrapper ([adaptateur](http://fr.wikipedia.org/wiki/Wrapper_%28informatique%29)) pour accéder aux données. Le wrapper est une classe qui peut être vue comme le message qui va être envoyé du client vers le serveur pour la demande du service web.

Tous les types présents dans le tableau de la page précédente fonctionnent avec Axis 2.1.1.1. Seuls les types suivants posent quelques problèmes. Les solutions pour le type *Char* et *DataSet* ont été expliquées précédemment.

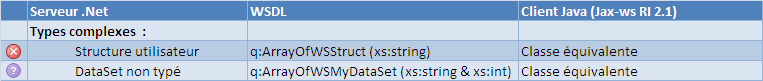
|  |  |
| --- | --- |
| *Producteur* : WCF  *Consommateur* : Java Axis 2.1.1.1 | *Version de SOAP* : SOAP 1.1 et 1.2  *Mode d’encodage* : Document / Literal Wrapped |

tab22.png

*Liste des correspondances entre le web service .Net et un client Java (Proxy Axis 2.1.1.1)*

Dans le cas de l’utilisation d’un proxy réalisé avec **JAX-WS 2.1** seuls les types suivant peuvent poser quelques problèmes.

|  |  |
| --- | --- |
| *Producteur* : WCF  *Consommateur* : Java Jax-ws RI 2.1 | *Version de SOAP* : SOAP 1.1 et 1.2  *Mode d’encodage* : Document / Literal Wrapped |



*Liste des correspondances entre le web service .Net et un client Java (Proxy Jax-ws RI 2.1)*

**Le type DataSet** défini dans le contrat WSDL empêche JAX-WS de générer le proxy. Pour générer le proxy avec succès, il faut désactiver ce type de donnés sur le serveur de Services Web WCF et republier le contrat WSDL.

**Le type Structure** ne peut pas être utilisé entre JAX-WS et WCF. En effet, si un client utilise un proxy JAX-WS, celui-ci n’arrive pas à charger correctement les données dans une structure utilisateur et il envoie alors une structure vide au serveur WCF. Notons qu’il n’y a pas vraiment de sens à utiliser un type structure dans un langage orienté objet. Le contournement sera de passer par une classe.

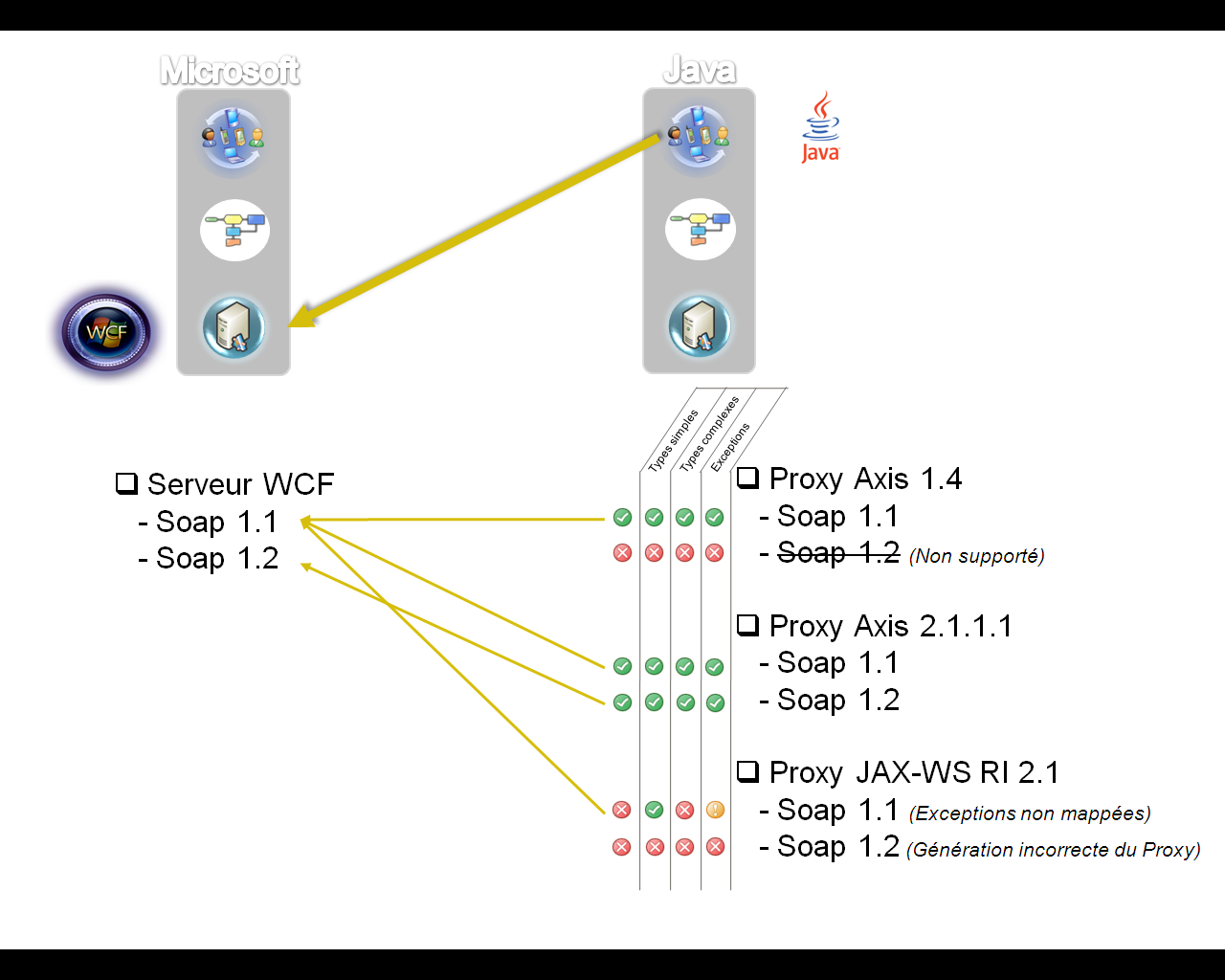
|  |  |
| --- | --- |
| *Producteur* : WCF  *Consommateur* : Java Jax-ws RI 2.1 | *Version de SOAP* : SOAP 1.1 et 1.2  *Mode d’encodage* : Document / Literal Wrapped |



*Liste des correspondances entre le web service .Net et un client Java (Proxy Jax-ws RI 2.1)*

**L’exception de type riche** est mappée par le proxy et permet d’accéder aux informations contenues dans l’exception. Cependant il n’y a pas une bonne hiérarchie d’héritage ; en effet la classe générée hérite de la classe *Exception* au lieu d’hériter de la classe *SoapFaultException*.

L’image ci-dessous représente une synthèse de cette section concentrée sur les Services Web WCF consommés par des clients Java dont le proxy a été généré avec Axis 1.4 et 2.1.1.1 et JAX-WS 2.1.



*Un serveur de services web WCF est consommé par différents clients Java*

**Le serveur WCF** a passé correctement les tests de compatibilité BasicProfile 1.1 définis par l’organisme [WS-I](http://ws-i.org/). De plus, un client WCF arrive à consommer tous les types et les exceptions du serveur WCF (plutôt normal mais bon à vérifier pour s’assurer de la bonne utilisation du framework au sein des implémentations de nos exemples).

**Le proxy Axis 1.4** ne supporte que Soap 1.1 mais il gère très bien tous les types de données. Toutefois, il ne supporte pas Soap 1.2 et on ne peut pas utiliser l’encodage Document / Literal qui est néanmoins une référence en ce qui concerne l’interopérabilité. En effet, la norme WS-I Basic Profile 1.0 a choisi Literal comme standard au détriment du style Encoded.

**Le proxy Axis 2.1.1.1** supporte plus de protocoles que la version 1.4. Ainsi, il supporte Soap 1.2 et il utilise l’encodage Document / Literal.

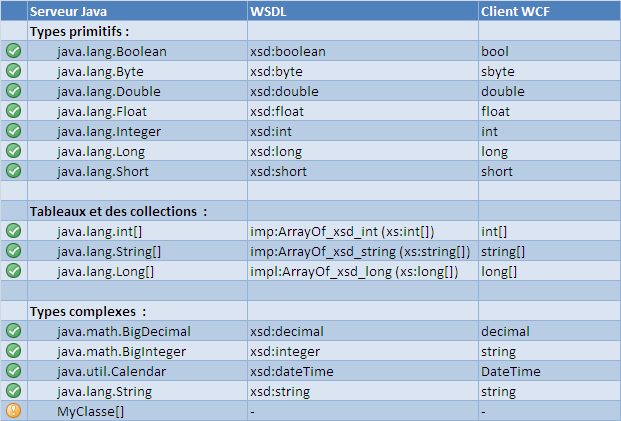
**Le proxy JAX-WS RI 2.1** n’est pas capable de générer un proxy en Soap 1.2 alors qu’il est normalement compatible Soap 1.2. De plus, il ne permet pas d’utiliser des structures comme type pour les échanges.

1. Types échangés entre un serveur Java et un client .Net

Avec les types primitifs, les tableaux, les collections et les types complexes il n’y a eu aucun problème d’interopérabilité. Par contre il n’a pas été possible de proposer une classe Java de type complexe avec Axis 1.4 car la méthode de déploiement d’Axis ne nous a pas permis de déployer plusieurs fichiers de classe.

Par contre, cette opération a été réalisée très facilement avec Axis 2.1.1.1 et JAX-WS RI 2.1, aucun problème d’interopérabilité n’a été rencontré.

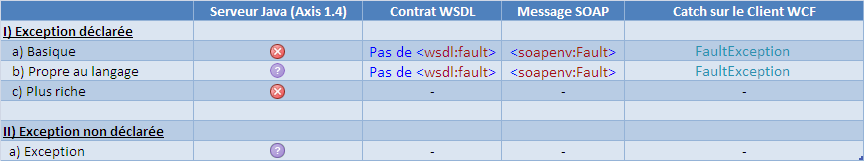
|  |  |
| --- | --- |
| *Producteur* : Java  *Consommateur* : WCF | *Version de SOAP* : SOAP 1.1 et 1.2  *Mode d’encodage* : RPC/Encoded et Document / Literal |



*Liste des correspondances entre le web service Java et un client WCF*

En ce qui concerne les exceptions, il est **obligatoire de déclarer ses exceptions dans le code source du serveur Axis 1.4**, cependant **les exceptions ne sont pas déclarées dans le contrat WSDL.**

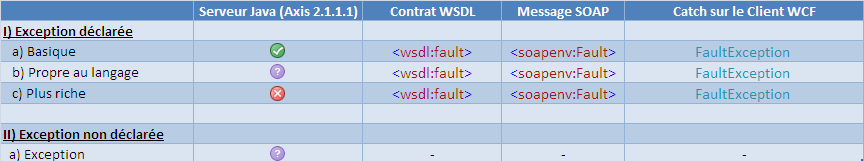
|  |  |
| --- | --- |
| *Producteur* : Java Axis 1.4  *Consommateur* : WCF | *Version de SOAP* : SOAP 1.1  *Mode d’encodage* : RPC/Encoded |



*Tableaux sur les exceptions Axis 1.4 consommées par un client WCF*

Avec **Axis 2.1.1.1 les exceptions doivent être aussi obligatoirement déclarées dans le code source du serveur et elles sont déclarées dans le contrat WSDL.** Cependant le mappage des exceptions dans le monde .Net n’est pas exécuté convenablement car les classes mappées ne permettent pas de gérer l’exception. Le développeur de l’application cliente ne peut donc pas traiter les exceptions propres au langage et plus riches dans le monde .Net qu’avec un try..catch sur *FaultException*.

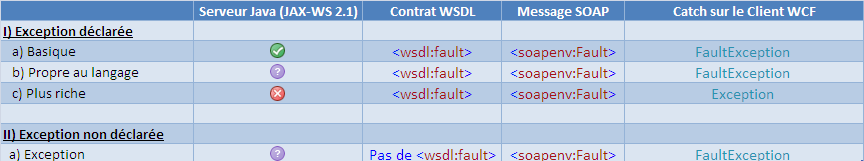
|  |  |
| --- | --- |
| *Producteur* : Java Axis 2.1.1  *Consommateur* : WCF | *Version de SOAP* : SOAP 1.1 et 1.2  *Mode d’encodage* : Document / Literal |

*Tableaux sur les exceptions Axis 2.1.1.1 consommées par un client WCF*

JAX-WS permet de proposer des exceptions déclarées et non déclarées. Cependant, dans le monde .Net, le mappage des exceptions n’est pas exécuté convenablement. Ceci empêche le développeur de l’application cliente de traiter les erreurs avec la classe mappée.

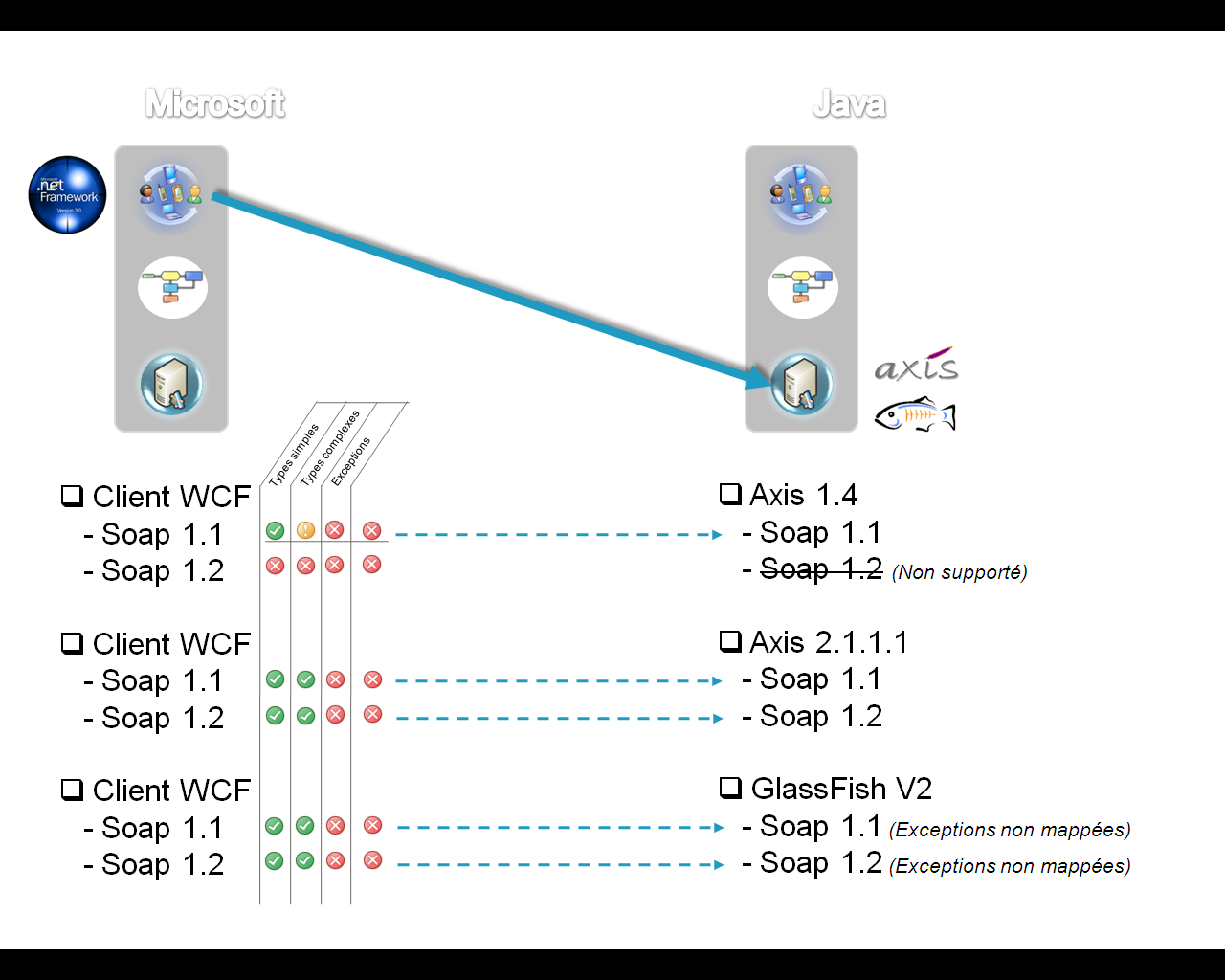
De plus, dans le cas de la création d’une exception enrichie avec JAX-WS, le consommateur WCF ne peut traiter cette exception dans le monde .Net qu’avec un try..catch sur *Exception* au lieu de FaultException.

|  |  |
| --- | --- |
| *Producteur* : Java JAX-WS 2.1  *Consommateur* : WCF | *Version de SOAP* : SOAP 1.1 et 1.2  *Mode d’encodage* : Document / Literal |



*Tableaux sur les exceptions Jax-ws 2.1 consommées par un client WCF*

L’image ci-dessous représente une synthèse de la section sur les Services Web Java consommés par un client WCF.



*Un client WCF consomme des services web de différents serveurs d’applications J2EE*

**Le serveur Axis 1.4** ne supporte que Soap 1.1 et ne propose que l’encodage RPC / Encoded. Il ne respecte donc pas la norme WS-I Basic Profile 1.0 qui a choisi Literal comme standard au détriment du style Encoded. De plus, en ce qui concerne la gestion des exceptions, on ne peut pas déclarer les exceptions dans le contrat WSDL. La méthode de déploiement avec le fichier *jws* ne permet pas d’utiliser une classe comme type échangé au niveau du service.

**Le serveur Axis 2.1.1.1** supporte des protocoles supplémentaires par rapport à la version 1.4. En effet, il supporte Soap 1.2 et utilise l’encodage Document / Literal. De même au niveau de la gestion des exceptions, Axis 2.1.1.1 permet d’avoir des exceptions déclarées dans le contrat WSDL. Axis 2.1.1.1 n’a pas passé correctement les tests de compatibilité BasicProfile 1.1 définis par l’organisme WS-I.

**Le serveur JAX-WS RI 2.1** supporte le Soap 1.1 et 1.2 et utilise un encodage Document / Literal. De plus la méthode de déploiement permet d’utiliser une classe comme type échangé au niveau du service. Ceci permet de créer sa propre classe d’exception héritant de la classe *java.lang.exception*.Jax-ws 2.1a passé correctement les tests de compatibilité BasicProfile 1.1 définis par l’organisme WS-I. Le générateur du proxy Jax-ws ne mappe pas correctement les exceptions d’un serveur jax-ws.

**Le client WCF :** le générateur de proxy de WCF ne fait pas le mappage correctement des exceptions déclarées et levées par JAX-WS RI..

1. PERSPECTIVE : JAVA et .NEt SONT-ils INTEROPERABLEs ?

Si les implémentations des Services Web proposées par APACHE, SUN et MICROSOFT respectent les spécifications proposées par le consortium WS-I, ces frameworks ne permettent pas pour autant d’assurer que vos développements soient interopérables d’un éditeur à un autre, d’un framework à un autre, que ce soit en environnement homogène ou hétérogène .Net et Java.

Les défis principaux auxquels vous vous trouverez confrontés dans le cadre d’une interopérabilité basique c’est-à-dire, sans utilisation des protocoles WS-\*) porteront sur :

* La sérialisation des données échangées au format SOAP : sur ce point, il est nécessaire de développer des compétences sur le framework proposée par votre technologie Services Web (JAX-B certainement côté Java, et DataContractSerializer côté   
   .Net), et de tester en amont la bonne adéquation des correspondances de type dans la mesure où les méthodes Contract First restent difficiles à implémenter.
* La gestion des exceptions où les frameworks mappe de façon incorrecte les exceptions lues dans le contract WSDL consommé.

Sur ces deux points, vos développements ne pourront se contenter se suivre les recommandations de l’une ou l’autre des technologies, mais devront plutôt définir des bonnes pratiques d’implémentation afin de s’assurer qu’un Service Web peut être consommé de façon identique qu’il soit implémenté dans telle ou telle version de framework.

Au terme de cette étude, il est important de souligner qu’ - alors que les spécifications des Services Web sont aujourd’hui considérées comme matures avec plusieurs années d’existence et des versions successives visant à les compléter - les implémentations des éditeurs JAVA et de Microsoft .Net restent jeunes (quelques mois à quelques semaines, et nombreuses sont celles qui ne sont pas en version finale).

De plus, ces implémentations possèdent de nombreux attributs de paramètrage (via des annotations, via des fichiers externalisés), si bien que l’interopérabilité pourra aussi être réalisé via des réglages précis associés aux aspects dynamiques de ces frameworks. Des initiatives telles que celles de Microsoft et SUN autour du projet [Tango](https://wsit.dev.java.net/) et qui visent à tester ces paramétrages et proposer des implémentations interopérables sont donc à privilégier.

Dans ce cadre, nous constatons à juste titre que l’utilisation croissante des Services Web en entreprise poussent les organisations à s’équiper de nouveaux outils et compétences afin de gérer au mieux le foisonnement des technologies Services Web.

1. POUR ALLER PLUS LOIN

Les services web présentés dans ce livre blanc ne mettent pas en œuvre un **mécanisme d’authentification.** Si vous souhaitez mettre en œuvre ce mécanisme, vous pouvez consulter le site suivant :

* Comment faire avec WCF :

<http://www.microsoft.com/france/Securite/evenements/webcastsjms_developpeurs.mspx>

* Comment faire avec Axis :

<http://ws.apache.org/axis2/modules/rampart/1_1/security-module.html>

* Comment faire avec GlassFish :

<https://wsit.dev.java.net/>

Les exemples fournis avec ce livre blanc n’utilisent pas toutes les possibilités de WCF (Security, Relaibility, Transactions, …). Pour aller plus loin, vous pouvez consulter les sites suivants :

<http://www.microsoft.com/france/msdn/aspnet/webservices/wsmsplatform.mspx>

<http://www.request-response.com/blog/PermaLink,guid,f731e5cc-9490-4f1e-bc7d-efb91f357cd1.aspx>