1

Introduction

La première difficulté à laquelle on se heurte lorsqu'on aborde le vaste sujet des technologies de services Web est d'ordre terminologique. Un exemple, désormais bien connu, du désordre terminologique est le vrai faux acronyme SOAP, qui signifierait « Simple Object Access Protocol », alors qu'il désigne un protocole d'échange entre applications réparties – où il n'est nulle part question d'accéder à des « objets ». Le débat a finalement été tranché par le W3C, qui a d'autorité supprimé la forme développée du terme « SOAP », dont il a simplement fait un nom propre.

Les difficultés commencent, à vrai dire, avec le terme même de « service Web » (*Web service*) : George Colony, fondateur et CEO de Forrester Research Inc., dans sa conférence du 10 mars 2003 au ICT World Forum (*http://idg.net/ic_1211529_9677_1-5041.html*) dit à propos des services Web qu'il n'est absolument pas question de « services » ni de « Web », mais que la dénomination la plus appropriée serait celle de « *middleware* Internet » qui permet de connecter les applications des entreprises à celles de leurs clients et partenaires.

Il est vrai que le terme de « service » est galvaudé, que le terme « Web » évoque les *sites* Web, et que les deux termes juxtaposés font penser à des *services* pour le public et les professionnels, pourvus par des *sites* Web, ce qui est déroutant par rapport au concept de services Web. Tous ceux qui, comme les auteurs, ont animé des conférences et des présentations sur le sujet peuvent témoigner de la difficulté à articuler les messages les plus simples en raison de l'usage détourné de ces termes. Par exemple, il faut rappeler sans cesse le fait que cette technologie préside à l'échange direct des applications entre elles *sans* la participation ni l'intermédiation des utilisateurs.

Cela dit, même si la proposition de George Colony a l'avantage d'être claire, nous ne sommes pas entièrement d'accords avec lui sur deux points :

- Le terme de *middleware* doit être manié avec précaution, car il évoque le déploiement dans une architecture répartie d'un ensemble de composants technologiques cohérents, éléments du même produit. Or, il n'y a pas de produit à déployer, mais plutôt des spécifications de *langages de description* (comme WSDL) et de *protocoles d'interaction* (comme SOAP) que chacun peut

implémenter, dans son environnement technique, par des composants logiciels standards ou bien spécifiques, propriétaires ou bien ouverts. C'est la conformité aux spécifications de ces composants qui permet l'*interopérabilité* des applications, objectif primaire de la technologie des services Web, et le *middleware* en question, autant qu'on puisse l'appeler ainsi, est donc mis en œuvre par l'interaction dynamique de composants d'origines diverses et d'implémentations hétérogènes.

- À l'inverse, le terme de « service », bien que souvent employé dans des acceptions plus précises, reste pertinent et important. L'utilisation de ce terme permet de rattacher la technologie des services Web à l'*architecture orientée services*. L'architecture orientée services est un concept et une approche de mise en œuvre des architectures réparties centrée sur la notion de *relation de service* entre applications et sur la formalisation de cette relation dans un *contrat*. L'architecture orientée services est en principe un concept indépendant de la technologie des services Web, mais cette dernière représente désormais son plus important moyen d'implémentation et fournit la base technologique pour sa diffusion sur une échelle jamais expérimentée auparavant. Le langage WSDL (Web Services Description Language) en est la technologie pivot qui représente le noyau extensible d'un langage de formalisation de contrats de service entre applications.

Ces précisions faites, en conformité avec un usage désormais assez répandu, nous continuerons à appeler les technologies présentées dans cet ouvrage, *technologies de services Web* en sachant que le terme va rapidement se banaliser comme un nom propre (si ce n'est pas déjà fait). Par ailleurs, nous utiliserons aussi le terme de *service Web* pour désigner une application qui joue le rôle de prestataire dans une relation de service et est mise en œuvre sur la base de la technologie des services Web.

Cet ouvrage tente de présenter un panorama large et organisé de ces technologies et de leurs implémentations en J2EE et .Net, tout en offrant un approfondissement des problèmes fondamentaux posés par leur déploiement et leur évolution, avec à la clé des exemples d'application et une étude de cas dont l'implémentation est déclinée en plusieurs variantes.

L'ouvrage, outre cette introduction et une conclusion est organisé en vingt-cinq chapitres regroupés en cinq parties. La première partie (chapitres 2, 3 et 4) traite de l'architecture orientée services. La deuxième partie (chapitres 5, 6, 7, 8, 9, 10, 11 et 12), après un rappel des technologies Internet et XML, introduit les technologies clés SOAP, WSDL et UDDI. La troisième partie (chapitres 13, 14, 15, 16 et 17) présente les plates-formes d'implémentation J2EE et .Net, ainsi que les composants disponibles sur le poste de travail et traite les problèmes d'interopérabilité. La quatrième partie (chapitres 18, 19, 20 et 21) introduit les technologies d'infrastructure qui garantissent l'échange fiable, la gestion de la sécurité et la gestion des transactions, ainsi que la gestion des processus métier. La cinquième et dernière partie (chapitres 22, 23, 24, 25 et 26) décline une étude de cas en plusieurs architectures à configuration statique et dynamique, sur plate-forme Java et .Net, ainsi que l'application du langage de scénarios de processus métier BPEL.

Nous pensons que la matière traitée est suffisante pour donner au lecteur une vision à la fois large et approfondie de l'architecture orientée services et de la technologie des services Web. Par ailleurs, le développement de la technologie des services Web avance à grands pas et touche des domaines et des sujets qui ne sont pas traités dans cet ouvrage pour des questions d'espace et d'unité d'œuvre. Le chapitre de conclusion évoque les axes centraux de consolidation et de développement futur des services Web, et quelques idées d'exploration sur des sujets non traités.

L'architecture orientée services

Nous avons pris le parti de considérer que la déclinaison du concept d'architecture orientée services (chapitres 2, 3 et 4) était le meilleur moyen pour introduire le cadre conceptuel et la terminologie utilisé dans la suite de l'ouvrage. La technologie des services Web est donc présentée comme *le* moyen d'implémentation des architectures orientées services. La **première partie** fournit la clé de lecture qui permet de comprendre la position et le rôle fonctionnel des différents modules technologiques présentés dans la deuxième et la quatrième partie, ainsi que des implémentations présentées en troisième partie.

Le **chapitre 2** introduit le concept d'architecture orientée services. Il introduit la *relation de service* et les rôles de *clients* et de *prestataires* joués par les applications participantes. Il est important de noter que nous avons choisi le terme « prestataire » pour marquer une différence avec la terminologie des architectures client/serveur, qui ne sont qu'une forme spécifique et limitée des architectures client/prestataire. Il introduit également la notion de *contrat*, lequel formalise les engagements du prestataire et éventuellement du client dans la réalisation de la prestation de services.

Un contrat est un document organisé en plusieurs parties, dont les plus importantes sont :

- la description des *fonctions* du service ;
- la description de l'*interface* du service ;
- la description de la *qualité* du service.

Le chapitre 2 présente les fonctions et l'interface dans le contrat de service. Il faut bien noter la différence entre les fonctions et l'interface du service : la description des fonctions est une description abstraite de la prestation de services, tandis que l'interface est une description des mécanismes et des protocoles de communication avec le prestataire de services. Naturellement, la compréhension du lien entre l'interface et les fonctions d'un service est capitale. Le problème de la formalisation de ce lien n'a pas encore de solution satisfaisante aujourd'hui, tout au moins à l'échelle où ce problème est posé par la diffusion des technologies des services Web.

Si la description fonctionnelle est abstraite et indépendante de l'implémentation du prestataire, la description de l'interface s'étend jusqu'aux détails concrets comme les protocoles de transport des messages et les adresses des ports de réception.

Le **chapitre 3** traite de la qualité de service, c'est-à-dire de l'ensemble des propriétés opérationnelles (non fonctionnelles) d'un service : performance, accessibilité, fiabilité, disponibilité, continuité, sécurité, exactitude, précision… La formalisation et la prise en charge explicite d'engagements de qualité de service est de façon générale encore insuffisamment, voire pas du tout, traitée dans le cadre des technologies des services Web. La qualité de service va prendre une importance croissante avec la diffusion d'architectures orientées services de plus en plus larges et dynamiques. Les engagements de qualité de service vont constituer un facteur de différentiation importante entre les prestataires fournissant le même service du point de vue fonctionnel.

Le chapitre 3 se termine par une discussion des relations entre le contrat de service et la mise en œuvre concrète des applications clientes et prestataires agissant en conformité avec le contrat. Il établit notamment la relation entre les différentes parties du contrat et les langages et protocoles des technologies de services Web. Par ailleurs, lors de la présentation (dans les chapitres 2, 3 et 4) de chaque élément du contrat, qu'il soit fonctionnel, d'interface ou opérationnel, l'ouvrage renvoie

systématiquement à la technologie de services Web censée décrire formellement l'engagement contractuel ou bien le mettre en œuvre.

Le **chapitre 4** traite des architectures orientées services à *configuration dynamique*. Pour introduire le sujet, il présente tout d'abord deux « figures » de la démarche de conception et de mise en œuvre de l'architecture orientée services :

- l'*agrégation* de services ;
- la *dissémination* de services.

L'agrégation est la réalisation d'un service qui intègre, pour réaliser sa prestation, les résultats des prestations d'autres services. La dissémination est, à l'inverse, la mise en œuvre sous forme de services modulaires des fonctions d'une application monolithique. La conception d'une architecture orientée services est en général le résultat de la combinaison de ces deux démarches.

L'aspect dynamique de la configuration de l'architecture n'est ni secondaire ni accessoire, mais bien au cœur même du concept d'architecture orientée services (ce qui n'empêche pas par ailleurs de mettre en œuvre des architectures orientées services totalement statiques). Dans une architecture dynamique, les services qui la composent, les applications prestataires qui interviennent, ainsi qu'un certain nombre de propriétés opérationnelles des prestations de services ne sont pas définis *avant* sa mise en place, mais sont composés, configurés, établis, voire négociés, au moment de l'exécution. Ce processus peut être itératif : il est possible de reconfigurer une architecture dynamique à la volée lors de son fonctionnement normal, ou bien à l'occasion d'un dysfonctionnement.

Avec les technologies de services Web disponibles actuellement, on peut notamment établir des architectures dans lesquelles les applications participantes peuvent choisir dynamiquement les services « abstraits » qu'elles consomment, les prestataires de ces services, les ports d'accès de ces prestataires. L'étude de cas présenté dans la cinquième partie articule la même application répartie en plusieurs scénarios d'architectures douées de niveaux différents de capacité de configuration dynamique.

Les technologies des services Web

La **deuxième partie** (chapitres 5, 6, 7, 8, 9, 10, 11 et 12), après un rappel des bases et des fondements (les protocoles Internet et le langage XML) présente les trois technologies clés des services Web : SOAP, WSDL et UDDI.

Il est évident que, sans Internet, l'ensemble des technologies de services Web ne serait encore qu'un autre standard de *middleware*, un nouveau concurrent de DCOM ou de CORBA. À l'inverse, certains fournisseurs qui ont un parc important de produits propriétaires installés prétendent que, sur des réseaux locaux ou propriétaires, il est possible de déployer des architectures de services Web qui n'utilisent pas de protocoles de communication Internet, mais des *middlewares* patrimoniaux. Cette « mouvance » définit un service Web comme une application dont l'interface est décrite par un document WSDL, indépendamment de la technologie de *middleware* utilisée pour interagir avec elle. En revanche, le déploiement de ces mêmes architectures sur Internet impose l'utilisation de protocoles Internet et notamment d'HTTP, qui se détache aujourd'hui comme le premier protocole de transport pour la communication avec les services Web. Le **chapitre 5** rappelle les fondamentaux des concepts

et protocoles Internet (URI et URL, HTTP, SMTP, MIME, SSL, TLS) ainsi que le modèle de référence en sept couches OSI de l'International Standard Organisation.

Le **chapitre 6** est un rappel indispensable de ce que sont XML et les technologies connexes comme XML Namespaces, Xlink, Xpath, XML Base, XML Schema et DOM. Les technologies XML constituent une véritable fondation pour les technologies de services Web : XML est à la base du format de message SOAP et du langage de description WSDL.

XML Namespaces et XML Schema sont particulièrement utilisées par les services Web. XML Namespaces est l'outil de gestion des versions et permet de gérer sans conflit l'assemblage et l'extension de technologies et d'applications d'origines différentes. Quant à XML Schema, il est spécifié d'emblée comme seul outil de définition de formats XML dans les services Web. Les DTD n'ont pas cours dans le monde des services Web : il est même explicitement interdit, par exemple, de véhiculer une DTD comme partie d'un message SOAP.

Ces rappels sont faits avec le simple objectif d'épargner au lecteur, qui a déjà une certaine familiarité avec la matière, la nécessité de quitter l'ouvrage pour un rappel rapide ou un renseignement ponctuel et ne remplacent en aucun cas les ouvrages spécialisés sur le sujet.

SOAP, qui est l'objet des chapitres 7, 8 et 9, va inévitablement devenir *le* protocole d'échange utilisé pour communiquer avec les services Web, bien qu'en principe il ne soit pas le seul protocole admis. Le **chapitre 7** introduit les fondamentaux du protocole (le format de message, le message d'erreur, le style d'échange « message à sens unique ») et présente en outre rapidement la problématique des chaînes d'acheminement (*routing*) : en fait, SOAP est basiquement conçu pour permettre d'interposer entre l'expéditeur et le destinataire une chaîne d'intermédiaires qui sont, potentiellement, des fournisseurs de services annexes comme la sécurité et la non-répudiation. L'utilisation d'une chaîne d'acheminement reste une possibilité qui peut être mise en œuvre comme une extension « propriétaire » du protocole SOAP (c'est l'option choisie par Microsoft avec la spécification WS-Routing) en attendant une spécification du mécanisme qui puisse aspirer au statut de standard.

La démarche mise en œuvre pour les chaînes d'acheminement est typique de l'approche courante du développement des spécifications des technologies de services Web :

- les spécifications de base (SOAP, WSDL) contiennent un mécanisme standard d'extension ;
- les promoteurs d'une technologie de niveau « supérieur » (par exemple la fiabilité des échanges, la sécurité, les transactions) utilisent les mécanismes standards d'extension pour proposer des spécifications : dans cette phase, on peut assister à la parution de plusieurs propositions concurrentes ;
- un acteur institutionnel (W3C, OASIS) est saisi de la tâche de bâtir une norme unifiée sur la base d'une ou plusieurs propositions concurrentes.

La troisième étape n'est évidemment pas automatique, mais résulte des négociations conduites « en coulisses » entre les acteurs technologiques majeurs.

Le **chapitre 8** présente le sujet très controversé du *codage des données* dans un message SOAP. Le sujet est complexe pour plusieurs raisons que nous analysons en détail dans ce chapitre :

- les principaux langages de programmations manipulent des structures de données partagées et circulaires (par exemple des graphes d'objets) ;

- pour pouvoir transférer ces structures, il faut un mécanisme pour les *sérialiser* dans un fragment XML, partie d'un message SOAP ;
- la représentation linéaire de ces structures ne peut pas être définie par l'utilisation standard d'XML Schema.

La spécification SOAP 1.1 propose un mécanisme de codage dont le résultat peut être validé par un analyseur syntaxique XML standard mais demande la mise en œuvre d'un mécanisme spécifique capable de reconstruire la structure partagée ou circulaire en mémoire. La discussion dans la communauté est très vive : l'organisme de validation d'interopérabilité des implémentations des technologies des services Web (WS-I) interdit, pour cause de défaut d'interopérabilité, l'utilisation du mécanisme de sérialisation (dit style de codage SOAP) car il n'est pas mis en œuvre de façon homogène, et dans la spécification SOAP 1.2 (qui n'est pas encore adoptée comme recommandation par le W3C) la mise en œuvre du style de codage est considérée comme optionnelle. Le codage permettant la sérialisation/désérialisation de structures partagées ou circulaires est cependant nécessaire pour « coller » aux applications patrimoniales des interfaces de services Web sans modifier leurs API (Application Programming Interface), car ces dernières présentent parfois des invocations de méthodes et des procédures véhiculant « par valeur » des structures de ce type.

Le chapitre 8 présente par ailleurs la spécification contenue dans la note W3C *SOAP Messages with Attachments* qui permet d'inclure dans la même requête ou réponse HTTP un message SOAP et des objets binaires (images, documents pdf, documents Word…) considérés comme des pièces jointes, tout en permettant de référencer ces pièces de l'intérieur du message. Nous ne présentons pas la spécification concurrente (DIME) d'origine Microsoft, qui est postérieure mais semble rester confinée dans le monde Microsoft.

Le **chapitre 9** décrit plus en détail les styles d'échange propres au protocole SOAP. En fait, SOAP propose deux styles d'échange : le *message à sens unique* et la *requête/réponse*. Le deuxième style ne peut être mis en œuvre que sur un protocole de transport bidirectionnel comme HTTP, à savoir sur un protocole de transport qui se charge lui-même de la corrélation entre la requête et la réponse. La corrélation entre messages transférés par des protocoles unidirectionnels (comme SMTP) peut bien entendu être réalisée, mais via des extensions, à savoir l'utilisation d'identifiants de messages contenus dans l'en-tête.

Le chapitre 9 décrit la *liaison* SOAP/HTTP, c'est-à-dire l'ensemble des règles qu'il faut respecter pour transférer correctement des messages SOAP via le protocole HTTP. La présentation de la liaison permet également d'introduire la problématique de l'asynchronisme dans l'envoi et le traitement des messages.

Le style d'échange requête/réponse en SOAP se décline en deux variantes : le style *document* et le style *rpc*. Dans le style document, la requête et la réponse SOAP n'ont pas une structure différente de celle d'un message SOAP standard. En style rpc, la requête et la réponse ont une structure particulière qui permet d'utiliser le message et le protocole SOAP pour sérialiser l'appel et le retour d'appel de procédure distante. Le style rpc est notamment indispensable pour exposer comme interface de service Web l'API d'une application patrimoniale avec un minimum d'effort.

Le **chapitre 10** présente WSDL (Web Services Description Language). WSDL est l'outil pivot de la technologie des services Web car il permet véritablement de donner une description d'un service Web indépendante de sa technologie d'implémentation. Les traits principaux du langage sont présentés via

l'exemple d'un des services Web les plus populaires : l'accès programmatique par SOAP au moteur de recherche Google (*http://www.google.com/apis*).

Un document WSDL joue le rôle d'embryon de contrat de service et représente donc le document de référence pour les équipes côté « client » et côté « prestataire ». Il joue en outre un rôle technique pivot car il peut être :

- généré automatiquement à partir d'une application par des outils souvent intégrés aux environnements de développement ; dans ce cas, la formalisation du service dérive directement de la conception de l'interface d'une application ;
- ou bien être l'input de la génération de *proxies* et de *skeletons*, à savoir de code qui, intégré avec le code applicatif, permet à une application de jouer respectivement le rôle de client et de prestataire de services.

Le chapitre 10 présente quelques outils disponibles pour effectuer ces deux opérations. Ces outils sont bien sûr décrits plus avant dans les chapitres de la troisième partie de l'ouvrage et leur utilisation est montrée en détail dans l'étude de cas en cinquième partie.

Les **chapitres 11 et 12** présentent UDDI (*Universal Description, Discovery and Integration*), la spécification d'un service d'annuaire expressément dédié à la découverte et à la publication de services Web. UDDI est également réalisé comme un service Web (l'interface est décrite en WSDL et l'accès aux annuaires publics et privés est mis en œuvre en SOAP sur HTTP).

UDDI n'est pas seulement une spécification d'annuaire accompagnée de quelques implémentations (qui peuvent être utilisées pour mettre en œuvre ce que l'on appelle des *annuaires privés*, à l'intérieur d'une entreprise ou d'une communauté de partenaires) : c'est aussi le support d'un système réparti d'annuaires publics répliqués qui permettent la publication et la découverte de services sur Internet. Ce système réparti appelé UBR (UDDI Business Registry) est mis en œuvre par un groupe de fournisseurs, dont Microsoft et IBM, qui étaient parmi les promoteurs de la spécification.

La spécification UDDI distingue deux parties de l'interface d'accès : l'interface en lecture (*inquiry*) qui permet la recherche et la découverte de services Web, et l'interface de mise à jour (*publication*) qui permet la mise à jour de l'annuaire avec l'ajout de nouveaux services et la modification de services existants. Le chapitre 11 présente, via des exemples concrets d'interaction avec l'UBR réalisés en code exécutable Java, les primitives de recherche et de lecture. Le chapitre 12, illustre, toujours au moyen d'exemples d'interaction avec l'UBR, les primitives de publication. Dans l'étude de cas (cinquième partie), deux architectures dynamiques différentes (Java et .NET) sont illustrées à l'aide d'un annuaire UDDI privé. Le code source de tous les exemples des chapitres 11 et 12 est disponible en téléchargement libre sur le site d'accompagnement du livre, à l'URL *http://www.editions-eyrolles.com*.

L'annuaire UDDI offre aujourd'hui (version 2.0 et suivantes) la possibilité de définir des relations complexes entre prestataires de services (par exemple de type organisationnel ou de partenariat) ainsi que des possibilités de catégorisation et d'indexation des services et des prestataires en cohérence avec les différentes taxinomies et les divers systèmes de codification utilisés couramment par les entreprises dans les différents secteurs économiques.

Les plates-formes opérationnelles

La **troisième partie** (chapitres 13, 14, 15, 16 et 17) est consacrée à la description d'un certain nombre d'implémentations de technologies de services Web. En fait, nous présentons les différentes plates-formes Java/J2EE (chapitre 14) et la plate-forme Microsoft .Net (chapitre 15), ainsi qu'un certain nombre d'implémentations sur le poste de travail qui permettent à des applications locales, éventuellement téléchargées à la volée, de jouer le rôle de client de services Web (chapitre 16). Ces présentations sont précédées du chapitre 13 qui résume les principes de la démarche de développement des éléments d'une architecture de services Web (les clients, les prestataires), et suivies du chapitre 17, lequel traite du problème de l'interopérabilité effective entre implémentations hétérogènes.

Les principes de mise en œuvre des éléments d'une architecture de services Web (**chapitre 13**) sont indépendants des environnements de développement et d'exploitation choisis. Il est parfois surprenant de constater la fondamentale homogénéité de la démarche, que l'on soit en Java, .Net ou même sur d'autres environnements plus périphériques. Cette démarche varie selon la perspective dans laquelle on se situe : le chapitre 13 décrit les différentes méthodes de développement qui peuvent être appliquées selon que l'on se place du point de vue du prestataire d'un service ou de celui du client de ce service. La fin du chapitre présente quelques-unes de ces méthodes et montre qu'en fait la mise en œuvre des éléments d'une architecture de services se réduit à la combinaison d'un nombre restreint de tâches unitaires :

- la transformation d'un composant applicatif existant en un service, avec génération WSDL à la clé ;
- la génération d'un proxy à partir d'une description WSDL d'un service, à intégrer dans le client du service ;
- la génération d'un squelette (*skeleton*) de prestataire, toujours à partir d'une description WSDL d'un service ;
- la génération d'un client de test d'un service, toujours à partir d'une description WSDL.

De cette liste de tâches se dégage encore une fois le rôle primordial joué par la description WSDL, véritable pivot de toute action de développement. Les schémas de ces différentes tâches ne sont pas seulement décrits, mais sont mis en œuvre sur des exemples, à l'aide de différents outils de développement, en environnements .Net et J2EE.

Le **chapitre 14** présente les environnements Java/J2EE. Il débute par une description des produits de l'organisation Apache, c'est-à-dire de l'implémentation Java SOAP4J qui est considérée comme la référence *de facto*, ainsi que d'outils complémentaires tels Xerces et Tomcat, et du nouveau serveur de référence Axis.

En raison de la richesse et de l'hétérogénéité de l'offre Java, nous avons pris le parti de donner dans le chapitre 14 un large panorama des acteurs du monde Java et de l'évolution de leurs offres :

- IBM et BEA jouent dans la catégorie des acteurs ayant déjà une présence établie dans les systèmes informatiques des entreprises, avec des composants qui se situent dans le prolongement direct des offres respectives de serveurs d'applications (WebSphere, WebLogic).
- Tandis qu'IBM peut se targuer d'avoir été, avec Microsoft, à l'origine de la technologie des services Web et de rester aujourd'hui un acteur majeur avec WebSphere, Hewlett-Packard joue le

rôle surprenant du visionnaire (l'offre e-Speak, qui correspondait à des services Web avant les services Web, était remarquablement cohérente et développée) qui abandonne le marché des outils de développement et des serveurs d'applications pour se concentrer sur ses compétences en administration de systèmes répartis et les appliquer aux besoins du naissant *Web Services Management*.

- Sun Microsystems conduit des actions sur différents niveaux, comme la normalisation des implémentations du monde Java dont elle a la maîtrise via le mécanisme des JSR, qui doit tenir compte du standard de fait SOAP4J de Apache, et la mise à jour de l'offre SUN ONE.

À côté de ces acteurs historiques, et d'autres comme Oracle, Novell ou IONA Technologies qui ont un rôle pour l'instant moins marqué (sauf IONA qui voit son offre sur les services Web comme le prolongement naturel de sa maîtrise de la technologie CORBA et propose un outillage assez complet), un certain nombre d'acteurs totalement nouveaux (The Mind Electric, Cape Clear, Systinet, Bowstreet, Collaxa, PolarLake, AltoWeb, Sonic Software) se sont positionnés avec des produits intéressants. En fin de chapitre, une liste complète de sites de référence et de ressources est proposée au lecteur.

Le **chapitre 15** décrit, avec un niveau de détail important, l'environnement d'exécution Microsoft .Net, l'environnement de développement Visual Studio et la mise en œuvre des technologies des services Web dans ces environnements. Contrairement à l'environnement Java, préexistant à la parution des technologies de services Web, .Net est né en même temps, Microsoft ayant comme objectif explicite de rendre le maniement d'XML et la mise en place de services Web à la portée d'un processus de développement « sans peine ». L'intégration entre .Net, Visual Studio, le langage XML et les technologies des services Web est effectivement très poussée. Ce chapitre présente les différents éléments de l'environnement, à commencer par le CLR (Common Language Runtime) et les librairies d'objets de base, en passant par le langage C#, ASP .Net et les Web Forms pour terminer sur la génération assistée d'un service Web et d'un proxy en C#.

Le **chapitre 16** présente des technologies de différentes origines qui permettent de développer des applications sur le poste de travail. La cible de ce type d'applications est particulièrement large : plusieurs analystes, partant du constat des limitations sévères quant à la puissance et l'ergonomie que les technologies navigateur et HTML imposent aux interfaces homme/machine, font la prévision de l'arrivée d'une nouvelle génération de logiciels et d'applications sur le poste client capables de dépasser ces limitations. La possibilité d'exécuter dans le cadre d'un navigateur (Internet Explorer ou Mozilla) du code téléchargeable (en JavaScript ou Ecmascript) qui met en œuvre l'interaction avec les services Web via SOAP ouvre des perspectives très intéressantes pour une nouvelle génération d'applications.

De même, la possibilité de programmer en Visual Basic des logiciels bureautiques (tels qu'MS Word XP ou MS Excel XP), pour qu'ils puissent accéder directement à des services Web distants, change les perspectives de développement d'applications dans des domaines importants comme la gestion documentaire ou la gestion financière. L'utilisateur n'a pas à quitter son environnement de travail habituel pour interagir avec les applications et les bases de données de l'entreprise : c'est « de l'intérieur » de ses outils qu'il peut ramener des données sur le poste de travail, les visualiser sous la forme habituelle d'un texte ou d'un tableur et éventuellement sauvegarder sur les serveurs d'entreprise le résultat de son travail local simplement par un bouton d'interface. Le chapitre présente un exemple concret et détaillé de programmation d'Excel XP en Visual Basic (cette dernière application permet d'interroger le service Web Google et de produire les résultats d'une recherche dans un

tableau Excel). Le chapitre se termine par un exemple d'utilisation du composant de services Web de Macromedia Flash qui montre comment une animation locale peut se nourrir de données récupérées périodiquement auprès de services Web distants.

Le code-source de tous les exemples du chapitre 16 est disponible en téléchargement libre sur le site d'accompagnement du livre, à l'adresse *http://www.editions-eyrolles.com*.

La troisième partie est close par le **chapitre 17**, qui porte sur les moyens que la communauté de développement des services Web se donne pour tester l'interopérabilité *effective* entre implémentations hétérogènes et sur les résultats obtenus par cette démarche. Le problème de l'interopérabilité est bien le paradoxe des technologies des services Web : d'un côté elle est l'objectif principal et de l'autre un défi qu'il faut relever sans cesse. Ce qui est remarquable et nouveau (par rapport, par exemple, à la démarche de l'OMG sur CORBA et l'OMA), est que la communauté des développeurs s'est préoccupée de l'interopérabilité effective des implémentations dès le début et a mis en place des organisations, des démarches et des outils pour promouvoir, améliorer, contrôler et tester le niveau *effectif* d'interopérabilité.

Il faut noter que cette activité est non seulement bien différenciée de l'activité de spécification, mais aussi du contrôle de conformité des implémentations par rapport aux spécifications. En fait, elle ne porte pas de jugement sur la conformité aux spécifications des implémentations prises séparément, mais constate leur capacité à interopérer entre elles (en appliquant, par exemple, à chaque couple d'implémentation le même cas de test et en dressant la matrice des résultats). Du coup, cette activité produit également une critique empirique des spécifications lorsqu'un élément de ces spécifications est l'objet d'échecs répétés d'interopérabilité. La communauté de développement s'est donc dotée de plusieurs batteries de test sur les différentes technologies (SOAP, WSDL), effectue ces tests par *rounds* et en publie les résultats. Une organisation exclusivement dédiée à promouvoir, tester et contrôler l'interopérabilité a été créée par les principaux acteurs (WS-I). Le chapitre présente l'avancement de ces travaux et leurs résultats.

L'infrastructure des services Web

La **quatrième partie** de l'ouvrage reprend le tableau général de l'architecture des technologies des services Web tel que laissé à la fin de la troisième partie, où nous avons présenté la première « couche » (le protocole d'échange SOAP, le langage de description WSDL et le service d'annuaire UDDI).

La question que les utilisateurs se posent est la suivante : la disponibilité d'implémentations fiables de la première couche constitue-t-elle une condition suffisante à la mise en place d'architectures de services Web suffisamment performantes et robustes pour prendre en charge les processus métier par lesquels l'entreprise coopère avec ses clients et partenaires ? La réponse à la question n'est pas immédiate et nous conduit à nuancer nos propos.

Avec la vague Internet, l'entreprise a commencé par se *présenter* sur le Web (site institutionnel statique), puis elle a appris à *communiquer* sur le Web (site à gestion dynamique de contenu) et enfin à rendre accessible une partie de ses processus opérationnels via le Web (sites « transactionnels », sites de commerce électronique, sites B2B ou *business to business*). Ce sont évidemment ces dernières applications, surtout dans le domaine du B2B, qui sont les plus concernées par la technologie des

services Web. L'idée est simple : doubler l'accès actuel au processus de la part d'un utilisateur professionnel (appartenant à une organisation cliente ou partenaire) au moyen d'un navigateur, par un accès par programme via une interface de service Web. Cette « doublure » est-elle réalisable avec les technologies de la première couche ?

En fait, tout ce qu'un utilisateur fait manuellement à l'aide d'un navigateur peut être réalisé par un programme via une interface de service Web : il n'y a aucune dégradation de sécurité. L'utilisation de SSL/TLS se fait dans les deux cas exactement de la même manière et le danger de la saturation des appels qui conduit au *denial of service* n'est pas plus fort pour un *service* Web que pour un *site* Web. Les fonctions offertes par le service peuvent être, en première instance, exactement les mêmes que celles offertes par le site. Les outils disponibles, pour peu que l'application dispose d'une API utilisable, permettent la génération quasi-automatique du service et de sa description WSDL (qui peut être utilisée par les clients potentiels pour une génération pratiquement automatique des proxyservices). L'opération de création, pour un site Web donné, d'un service Web iso-fonctionnel, peut être effectuée (s'il n'y a pas de problèmes cachés) avec un effort quasiment nul.

La difficulté doit être cherchée plutôt du côté « client », dans la maîtrise de la « défaillance partielle » propre à toute architecture répartie, à commencer par la plus simple qui est le client/serveur traditionnel. Le « client » d'un site Web, derrière un navigateur, est un acteur humain : son intelligence est sollicitée non pas lorsqu'il remplit banalement un formulaire (un programme serait certainement plus rapide et précis) mais lorsqu'il est confronté à des situations d'erreur, d'attente indéfinie, d'incertitude. Le client d'un service Web, derrière le proxy, est un programme applicatif qui, s'il veut remplacer parfaitement l'utilisateur, doit être capable d'une performance comparable lorsque les choses ne se déroulent pas comme attendu. Une stratégie réaliste serait de soigner autant que possible la capacité de prendre en compte les défaillances du service et du réseau de la part du client, mais aussi de rendre facile l'intervention « manuelle » de l'utilisateur dans les cas, que l'on espère rares, d'erreur et de défaillance que l'on ne sait pas traiter entièrement par programme. L'utilisateur n'est plus un maillon de la chaîne de traitements, qui est automatisée, mais agit plutôt au niveau du paramétrage, de la surveillance et de la réparation du processus. Par ailleurs, dans les cas de doublure d'un site Web, une interface homme/machine avec l'application pour laquelle on a produit une interface de service Web existe déjà…

Les applications accessibles par navigateur Web sont une cible importante des technologies des services Web, mais elles ne représentent pas la seule cible. Ces technologies ont pour ambition de s'attaquer aux processus et aux applications stratégiques et, par agrégation et dissémination, de créer la possibilité de nouvelles combinaisons, de nouveaux processus automatisés, qui comprennent des dizaines, des centaines (voire plus) d'applications réparties, qui interagissent entre elles sans intervention humaine dans leur fonctionnement normal (qui inclut le contrôle et le traitement d'une dose de défaillances partielles).

Les technologies de services Web peuvent prendre en charge le véritable système nerveux de l'activité de production, de circulation, d'échange et de consommation des biens et des services. Pour mettre en œuvre des architectures en adéquation avec ce projet, les technologies sur lesquelles reposent les services Web (SOAP/WSDL/UDDI) sont nécessaires mais ne sont certainement pas suffisantes.

Il est indispensable de construire, sur la couche de base, des technologies d'infrastructure qui prennent en charge au moins trois fonctions clés :

- la fiabilité de l'échange ;
- la gestion de la sécurité ;
- la gestion des transactions.

Il faut rappeler qu'une technologie d'infrastructure, dans le cadre des services Web, est toujours bâtie selon la même méthode : par la spécification d'un *protocole*, avec sa syntaxe (le format des messages échangés et des assertions qui sont intégrées dans des documents, par exemple WSDL), et un ensemble de règles qui fixent l'interprétation et le traitement de ces messages et assertions. La mise en œuvre, par des implémentations différentes, du protocole en conformité avec les spécifications garantit en principe l'interopérabilité de ces implémentations.

La gestion de la fiabilité de l'échange (présentée **chapitre 18**) se trouve dans une situation paradoxale. Les technologies de services Web ont comme première cible la communication entre applications, garantie de l'interopérabilité : après avoir défini un protocole d'échange (SOAP), un langage de description des interfaces (WSDL) et un service d'annuaire (UDDI), on aurait pu s'attendre à un effort immédiat pour mettre, autant que possible, les applications communicantes à l'abri des défaillances du réseau et des participants à l'échange. Il n'en est rien car deux autres sujets ont retenu pratiquement toute l'attention de la communauté : la sécurité et les langages pour définir les scénarios des processus métier répartis. Les deux sujets sont certainement très importants, mais le fait est que la gestion de l'échange fiable a été étonnamment sous-estimée, voire considéré comme accessoire. Nous pensons que la sous-estimation de cette fonction d'infrastructure est une des causes du faible taux d'adoption des services Web car elle joue un rôle fondamental.

L'objectif de la gestion de l'échange fiable est pourtant simple à énoncer : donner aux applications participantes à l'échange l'assurance qu'un message est transmis une et une seule fois dans la séquence d'émission, ou que si ce n'est pas le cas l'émetteur a un compte rendu fiable de l'échec de la transmission. Il est évident que la programmation des applications qui dialoguent dans un tel contexte est facilitée car elle ne doit pas prendre en compte les situations d'incertitude sur la transmission du message.

La gestion de l'échange fiable (chapitre 18) est donc traitée, par la force des choses, de façon un peu académique puisque aucune solution n'est réellement disponible. Nous présentons la technologie HTTPR d'origine IBM, qui a le mérite d'avoir été proposée tout au début de l'essor des services Web, mais qui n'a pas encore dépassé le stade de prototype. L'idée est de « fiabiliser » HTTP et donc de rendre la gestion de la fiabilité transparente au niveau SOAP (le message SOAP ne sait pas s'il voyage sur un « canal » fiabilisé ou non).

La technologie HTTPR est une technologie élégante, mais qui reste marginale et destinée, selon l'intention même des auteurs, à des usages spécifiques. Ce n'est que depuis le début de l'année 2003 que le sujet commence à recevoir l'attention qu'il mérite, d'abord avec la proposition de spécification WS-Reliability (9 janvier 2003) par Sun Microsystems et d'autres partenaires. Nous présentons cette spécification qui a comme objet la fiabilisation du message à sens unique SOAP par une extension standard du protocole.

Le 13 mars 2003, IBM, BEA, Microsoft et TIBCO ont proposé une nouvelle spécification WS-ReliableMessaging (*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-reliablemessaging.asp*)

accompagnée d'un livre blanc et d'une *roadmap*. Cette spécification est parue trop tard pour que nous puissions la traiter dans cet ouvrage mais nous pouvons constater qu'avec l'engagement des deux acteurs historiques (IBM et Microsoft), la problématique de la fiabilité de l'échange a désormais trouvé sa place dans l'architecture des technologies des services Web.

Le **chapitre 19** présente l'infrastructure de gestion de la sécurité. C'est sans doute le sujet d'infrastructure sur lequel les travaux de spécification et d'implémentation ont fait le plus de progrès, sur la base il est vrai d'un travail préexistant assez avancé. En fait, le W3C propose des technologies essentielles pour la gestion de la sécurité des services Web (XML Signature, XML Encryption) et l'OASIS propose SAML (Security Assertion Markup Language), *framework* d'échange d'informations (assertions) de sécurité au format XML qui peuvent être encapsulées dans des messages SOAP.

La gestion de la sécurité touche les exigences classiques d'authentification et d'autorisation des acteurs et agents logiciels impliqués dans les échanges ainsi que la confidentialité, l'intégrité et la non-répudiation de ces mêmes échanges.

Le 27 juin 2002, Microsoft, IBM et VeriSign ont soumis les spécifications WS-Security à la communauté OASIS. BEA, Cisco, Intel, Iona, Novell, RSA, SAP et Sun Microsystem ont immédiatement manifesté leur disponibilité pour travailler dans le comité technique OASIS. La gestion de la sécurité est le seul des trois sujets d'infrastructure sur lequel les travaux de spécification avancent sur une unique roadmap, avec la participation des acteurs les plus importants impliqués dans le développement des technologies des services Web. L'approche choisie intègre des mécanismes patrimoniaux largement utilisés comme les certificats X.509 et les tickets Kerberos.

Le **chapitre 20** présente l'infrastructure de gestion des transactions. Sur ce sujet, deux spécifications se côtoient :

- BTP (*Business Transaction Protocol*), proposé initialement par BEA avec d'autres partenaires (dont Oracle et Hewlett-Packard) et géré, depuis mars 2001 par un comité technique OASIS ;
- le tandem WS-Coordination/WS-Transaction, (9 août 2002), proposé par IBM, Microsoft et BEA, qui n'a pas encore été confié à un organisme de standardisation.

BTP compte un certain nombre d'implémentations disponibles. WS-Coordination et WS-Transaction sont plus récentes et, de plus, coordonnées avec BPEL4WS (Business Process Execution Language for Web Services), langage destiné à spécifier des scénarios de processus métier promu par les mêmes sociétés. La présence de BEA dans cette deuxième initiative, ainsi que d'autres signes comme le fait que Collaxa, éditeur d'une des premières implémentations de BTP et aujourd'hui auteur d'une des premières implémentations de WS-Coordination/WS-Transaction, ne prend plus en charge le moteur BTP dans la nouvelle version de son serveur d'applications suggèrent que BTP n'est qu'une étape vers le standard de gestion des transactions pour les services Web, qui va se concrétiser dans l'évolution de WS-Coordination et WS-Transaction.

Le **chapitre 21** effectue un tour d'horizon des nombreuses spécifications qui traitent de la gestion des processus métier. Ce domaine est actuellement l'objet de nombreux bouleversements et plusieurs nouvelles spécifications sont apparues dans les derniers mois. Celles-ci touchent aux aspects de description des interactions entre les services Web qui participent aux processus et de l'ordre temporel de ces interactions, décrites en termes de messages et de traitements métier associés à l'émission ou à la réception de ces messages (orchestration). Ces spécifications traitent en outre de la manière de décrire l'interface publique des processus métier implémentés par les moteurs d'orchestration

(chorégraphie). Le chapitre dresse un rapide panorama des acteurs importants dans ce domaine et des principales spécifications en présence : BPEL4WS (mis en œuvre, avec WS-Coordination et WS-Transaction sur la base du moteur Collaxa, dans une variante de l'étude de cas présentée chapitre 26), BPML, WSCI et WSCL.

Les langages de définition de scénarios de processus métier, qui facilitent l'orchestration de dizaines, voire de centaines (et même plus) de services Web vont prendre de plus en plus d'importance en tant qu'outils de maîtrise de la complexité des architectures réparties de demain.

L'étude de cas

L'étude de cas est le sujet de la cinquième et dernière partie de l'ouvrage (chapitres 22, 23, 24, 25 et 26). Il s'agit de la mise en œuvre d'une architecture orientée services sur la base des technologies de services Web, qui prend en charge un processus métier d'organisation de voyages (réservation de places d'avion, de chambres d'hôtel, de voitures de location).

L'ensemble du code source des scénarios d'architecture de l'étude de cas peuvent être téléchargés librement sur le site d'accompagnement du livre, à l'adresse *http://www.editions-eyrolles.com*.

Nous avons choisi de simplifier au maximum, du point de vue fonctionnel, le processus, dont la complexité est vraiment très inférieure à celle des véritables systèmes de réservation centralisés (GDS ou Global Distribution System) comme Amadeus, Sabre, Galileo, WorldSpan. L'avantage est que le contenu fonctionnel, à ce niveau de simplification, est compréhensible de façon intuitive par toute personne ayant eu une expérience, même minimale, de ce type de voyage et le lecteur peut donc se concentrer sur l'objet de l'étude de cas, qui est l'architecture technique sous-jacente. Le contenu fonctionnel est présenté **chapitre 22**.

C'est un exemple d'*agrégation* de services de réservation de places d'avion, de chambres d'hôtel et de voitures de location, de la part d'un service d'une agence de voyages. Il s'agit donc d'une architecture à trois niveaux : un système « client final » accède à un service d'une agence de voyages qui agrège les services de réservation dans le but de constituer une réservation de voyages globale. Il est important de noter que l'application répartie, dans la variante la plus simple, comporte en fait au minimum cinq agents logiciels actifs : l'agent client (mis en œuvre comme un client SOAP dans un navigateur Internet Explorer, présenté chapitre 22), l'agent serveur de l'agence de voyages et un agent pour chaque système de réservation sectoriel (avion, hôtel, voiture).

Ce schéma est décliné d'abord dans une architecture complètement statique, à savoir totalement configurée *avant* exécution (**chapitre 23**). L'agrégation de services est mise en œuvre directement par le code applicatif Java du système de l'agence de voyages. Une telle approche est voisine de celle des architectures B2B, qui connectent de façon prédéterminée une entreprise avec ses clients et ses partenaires. Les serveurs de cette première architecture sont tous implémentés en Java par l'utilisation du *toolkit* Apache SOAP 2.3.1.

Le **chapitre 24** présente une architecture dynamique. L'idée est que d'une part les services de réservation sectoriels sont normalisés par des organismes professionnels (fictifs) dans des documents WSDL standards et que, d'autre part, une pluralité de prestataires de ces services sont accessibles en ligne. Le port d'accès au service de chacun de ces prestataires est, lui aussi, déterminé à l'exécution.

L'architecture dynamique fait donc intervenir un annuaire de services UDDI, qui permet la découverte dynamique des prestataires et de leurs ports d'accès. La mise en œuvre du processus d'organisation

du voyage est donc précédée par un processus de configuration dynamique de l'architecture (choix des prestataires et de leurs ports d'accès).

Le choix dynamique des prestataires, de leurs ports et à la limite des services est un point d'application privilégié de l'intelligence du service agrégeant, c'est-à-dire de sa capacité de prise en charge des préférences du client et de mise en œuvre de règles de gestion, qui peuvent devenir extrêmement sophistiquées (car elles représentent l'expertise métier de l'agence de voyages). On peut imaginer que les prestataires mettent en œuvre le même service minimal, décrit par le document WSDL standardisé par l'organisation interprofessionnelle. À ce service minimal, chaque prestataire peut ajouter des services annexes qui font sa valeur ajoutée. Le prestataire peut en outre se distinguer par le niveau de qualité de service sur lequel il s'engage. Nous n'avons pas poussé l'exemple aussi loin : le choix des prestataires et des points d'accès est fait au hasard (l'expertise métier de l'agence de voyages n'est pas le sujet de l'ouvrage) et il faut rappeler que les engagements de niveau de qualité de service ne font pas encore l'objet d'un langage d'assertions normalisé. La mise en œuvre de la configuration dynamique de l'architecture est, comme pour le processus métier d'organisation du voyage, le résultat de l'exécution d'un code applicatif Java intégré dans le système de l'agence.

Le **chapitre 25** met en œuvre exactement la même architecture, mais avec comme variante la ré-implémentation du service agrégeant en technologie Microsoft .Net (C#). Il s'agit d'un exercice intéressant au moins à deux titres. D'abord, cela permet de vérifier, que, dans certaines limites d'utilisation de la technologie des services Web, l'interopérabilité est effective et la technologie d'implémentation d'un service à partir de la formalisation de son contrat (le document WSDL) est interchangeable et, à la limite, banalisée. Ensuite, cela nous permet de comparer concrètement deux implémentations du même service, de tous les points de vue. Il reste entendu que le but de l'ouvrage n'est pas de trancher entre J2EE et .Net, mais au contraire de montrer que finalement, avec les services Web et, peut être, pour la première fois dans l'histoire de l'informatique, le choix de la technologie d'implémentation, qui reste un choix important et doit être pesé avec soin, n'est plus structurant ni irréversible par rapport à la mise en œuvre du service (il peut l'être, évidemment, pour d'autres raisons, surtout organisationnelles). On peut noter en passant que les microarchitectures respectives du service en Java/J2EE et en C#/.Net sont vraiment très proches et que le passage de l'une à l'autre est concrètement très simple à effectuer.

Le dernier chapitre (**chapitre 26**) reprend l'architecture statique du chapitre 23 mais la revisite en intégrant une gestion transactionnelle du processus métier de réservation et l'usage d'un langage de définition de scénario (BPEL). Cette gestion est destinée à suppléer aux faiblesses de l'architecture initiale qui ne s'appuie que sur des implémentations des spécifications de base des technologies des services Web. La nouvelle architecture fait appel à l'une des premières implémentations des spécifications BPEL4WS, WS-Coordination et WS-Transaction, matérialisée par le serveur BPEL Orchestration Server de la société Collaxa. Le fonctionnement de l'architecture qui en résulte est en mode « document » et asynchrone : les applications participantes s'échangent des documents et utilisent les interactions successives (par polling ou par callback) pour récupérer les résultats des traitements déclenchés. Cette dernière variante est plus didactique que réaliste (les quatre applications participantes doivent fonctionner sur des machines installées avec le même moteur Collaxa pour bénéficier des fonctionnalités de messagerie asynchrone, de coordination et de gestion de transactions), mais donne une très bonne idée de l'orientation adoptée ces six derniers mois par les principaux acteurs des technologies des services Web dans le domaine de l'infrastructure de gestion des processus métier et des transactions.