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.