Utilisation de l'architecture orientée services et du développement par composant logiciel pour créer des applications de type service Web

Un livre blanc de Rational Software

Alan Brown,
Simon Johnston,
Kevin Kelly

Rational®
the software development company

Sommaire

Contents 2
Introduction 2
Qu'est-ce qu'une architecture orientée services ? 3
  Conception à base d'interfaces 5
    Comportement des interfaces 5
Développement architectural de systèmes orientés services 6
    Organisation en couches de la conception d'applications 7
    Exemple de modèle de client 7
  Une conception par composant logiciel 8
  Une conception orientée services 8
    Concept d'antémémoire dans la conception orientée services 9
Conception des services Web XML 10
  Modèles de conception et de mise en oeuvre de services Web 11
    Performances et fiabilité 11
    Évolutivité grâce au comportement asynchrone et à la mise en file d'attente 12
    La "location" d'informations revisitée 13
  Modèle de conception de services Web résultant 14
Conclusion 15

Introduction

Le développement d'un système logiciel à l'échelle de l'entreprise est une mission des plus complexes. Malgré des décennies de progrès technologiques, les exigences imposées par les systèmes d'information actuels tirent jusqu'au point de rupture sur les capacités d'une entreprise à concevoir, développer et étendre ses solutions logicielles vitales. En particulier, seule une poignée de nouveaux systèmes sont développés en partant de zéro. La tâche d'un spécialise en architecture logicielle consiste plutôt généralement à prolonger la durée de vie d'une solution existante en décrivant une nouvelle logique applicative qui manipule un référentiel de données existant, en présentant les données et les transactions existantes par le biais de nouveaux canaux, comme un navigateur Internet ou des appareils de poche, en intégrant des systèmes jusqu'alors déconnectés supportant des activités qui se chevauchent, et ainsi de suite.

Afin d'aider les développeurs de logiciels, des produits d'infrastructure logicielle du commerce sont désormais disponibles auprès de sociétés de services et d'ingénierie en informatique telles que Microsoft et IBM. Ils constituent l'élément central des approches du développement de logiciels qu'elles préconisent dans leurs lignes de produits respectives que sont .NET et WebSphere. Les deux approches sont centrées sur l'assemblage hde systèmes à partir de services répartis. Mais y a-t-il quelque chose de nouveau dans le développement de solutions d'entreprise à partir de services ? En quoi les leçons des systèmes par composant logiciel s'appliquent-elles à la construction d'architectures orientées service ? Quelles sont les meilleures approches à appliquer pour développer des systèmes de qualité pour le déploiement de cette nouvelle génération de produits d'infrastructure logicielle ? Ces questions importantes constituent le sujet de ce document.

Au cours des dernières années, une grande part de l'attention de la communauté du génie logiciel a été focalisée sur des approches de conception, des processus et des outils soutenant l'idée selon laquelle des systèmes logiciels de grande envergure peuvent être assemblés à partir d'ensembles de fonctionnalités indépendants réutilisables. Une partie des fonctionnalités peuvent déjà être disponibles et mises en oeuvre en interne ou acquises auprès d'un tiers, tandis que les fonctionnalités restantes peuvent être encore à créer. Dans ces cas, l'ensemble du système doit être pensé et conçu de manière à rassembler tous ces éléments dans un tout cohérent. Aujourd'hui, cette approche est illustrée dans le *développement par composant logiciel*, un concept qui est appliqué dans des approches technologiques telles que la plate-forme .NET de Microsoft et les standards J2EE (Java 2 Enterprise Edition) pris en charge par des produits comme WebSphere d'IBM et iPlanet de Sun.

Un autre élément à prendre en compte est le fait que les systèmes opérationnels sont généralement répartis entre de nombreuses machines pour optimiser les performances, la disponibilité et l'extensibilité. Une solution d'entreprise doit coordonner des fonctionnalités s'exécutant sur un ensemble d'équipements. Une manière de concevoir un système de ce type est de le considérer comme composé d'un ensemble de*services* interdépendants. Chaque service fournit l'accès à un groupe défini de fonctionnalités. Le système dans son ensemble est conçu et mis en oeuvre comme un ensemble d'interactions entre ces services. Le fait d'exposer les fonctionnalités comme des services est un élément essentiel de flexibilité. Cela permet à d'autres éléments de fonctionnalité (peut-être mis en oeuvre eux-mêmes sous forme de services) d'utiliser d'autres services de manière naturelle indépendamment de leur emplacement physique. Un système évolue par l'ajout de nouveaux services. L' *architecture orientée services*résultante définit les services dont le système est composé, décrit les interactions qui se produisent entre les services pour réaliser un comportement donné et établit une correspondance entre les services et une ou plusieurs implémentations de technologies spécifiques.

Si les services englobent la fonctionnalité applicative, une certaine forme d'infrastructure interservices est nécessaire pour faciliter les interactions et la communication entre les services. Différentes formes de cette infrastructure sont possibles puisque les services peuvent être mis en oeuvre sur une machine seule, répartis entre un ensemble d'ordinateurs sur un réseau local, ou répartis plus largement entre plusieurs réseaux de l'entreprise. Un cas particulièrement intéressant est celui où les services utilisent Internet comme mécanisme de communication. Les *services Web* résultants partagent les caractéristiques de services plus généraux, mais ils nécessitent une attention particulière en raison du fait qu'ils utilisent un mécanisme public peu sûr pour les interactions entre services.

Jusqu'à présent, l'industrie du logiciel s'est concentrée essentiellement sur la technologie sous-jacente pour mettre en oeuvre des services Web et leurs interactions. Néanmoins, d'autres préoccupations apparaissent quant à savoir quelle est la meilleure manière de concevoir des services Web faciles à assembler dans des solutions à l'échelle de l'entreprise. Inversement, il y a eu un manque d'attention surprenant quant aux pratiques et aux outils appropriés pour concevoir des solutions logicielles d'entreprise composées de services Web. Comme pour la conception de toute structure complexe, des solutions de qualité sont le résultat de décisions architecturales anticipées soutenues par un certain nombre de techniques de conception, de patterns structurels et de styles. Ces patterns abordent des problèmes de service courants comme l'évolutivité, la fiabilité et la sécurité.

Ce document donne le contexte pour une meilleure compréhension des services et des architectures orientées services pour les solutions logicielles à l'échelle de l'entreprise. En particulier, il étudie les services sous l'angle de leur relation avec le concept mieux établi de composants logiciels, et il décrit comment les pratiques actuelles de développement par composant logiciel offrent un fondement testé et éprouvé pour la mise en oeuvre d'une architecture orientée services. La conception à base d'interfaces est mise en évidence comme un élément essentiel à la fois pour la conception de services et de composants, et il est indiqué que les interfaces exposées par les deux ont un certain nombre de contraintes et de critères qui les distinguent. Le langage UML (Unified Modeling Language)[1] est utilisé comme un outil pour décrire la conception logique et la conception de la mise en oeuvre, ainsi que des patterns spécifiques à la fois pour la conception de composants et de services.

Enfin, le document se concentre sur les services Web pour étudier les différents problèmes associés à la mise en oeuvre de services en général. En particulier, le document cherche à voir comment il est possible d'interpréter les directives générales pour l'architecture orientée services comme des modèles de conception spécifiques pour les services Web. Tous les patterns décrits dans ce document ont été implémentés en utilisant utilisant les capacités uniques en matière de patterns et de canevas de code de Rational® XDE™ et publiés via le Réseau de développeurs Rational[8].

Qu'est-ce qu'une architecture orientée services ?

En quoi consiste donc une architecture orientée services ? Essentiellement, il s'agit d'une méthode de conception d'un système logiciel visant à fournir des services à des applications utilisateur final ou à d'autres services par l'intermédiaire d'interfaces publiées et localisables. Dans la plupart des cas, les services offrent une manière plus efficace d'exposer des fonctions métier discrètes et, donc, un moyen idéal pour développer des applications sous-jacentes à des processus métier.

L'architecture orientée services n'est pas un nouveau concept, mais elle a pris de l'importance avec l'émergence de la technologie de services Web. Par exemple, voici un extrait d'un ouvrage publié en 2000 décrivant la valeur d'une architecture orientée services :

> *Solutions orientées services… Les applications doivent être développées sous la forme d'ensembles indépendants de services interdépendants offrant des interfaces bien définies avec leurs utilisateurs potentiels. De même, une technologie sous-jacente doit être disponible pour permettre aux développeurs d'applications de survoler des ensembles de services, de sélectionner ceux qui les intéressent et de les assembler pour créer la fonctionnalité voulue. [2]*

Pour les besoins de ce document, nous tiendrons compte de la définition suivante d'un service :

> *Un service est généralement mis en oeuvre sous la forme d'une entité logicielle localisable "à granulation grossière" qui existe en tant qu'instance unique et qui interagit avec les applications et les autres services via un modèle de communication par messages "à couplage lâche" (souvent asynchrone).*

Dans une large mesure, la terminologie applicable aux services est similaire à celle utilisée pour décrire le développement par composant logiciel. Il existe, néanmoins, des termes spécifiques utilisés pour définir les éléments de services Web, comme illustré dans la Figure 1 ci-dessous.

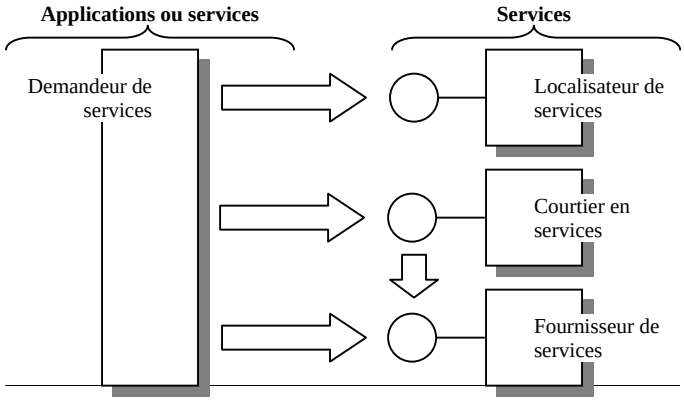

Figure 1 – Terminologie de services

- **Service** — Une entité logique ; le contrat défini par une ou plusieurs interfaces publiées.
- **Fournisseur de services** — L'entité logicielle qui met en oeuvre une spécification de service.
- **Demandeur de services** — L'entité logicielle qui appelle un fournisseur de services. En général, on parle de “client” ; toutefois, un demandeur de services peut être une application utilisateur final ou un autre service.
- **Localisateur de services** — Un genre spécifique de fournisseur de services qui agit comme un registre et permet de consulter les interfaces des fournisseurs de services et les emplacements des services.
- **Courtier en services** — Un genre spécifique de fournisseur de services spécifique qui peut transmettre des demandes de service à un ou plusieurs autres fournisseurs de services.

Cette description des services, et le contexte de leur utilisation, impose une série de contraintes. En outre, une utilisation efficace des services suggère l'application de quelques meilleures pratiques globales. Voici quelques-unes des caractéristiques clés pour une utilisation efficace des services :

- **À granulation grossière** — Les opérations sur les services sont souvent mises en oeuvre pour englober plus de fonctionnalités et fonctionner sur des ensembles de données plus volumineux, par comparaison avec une conception d'interface par composant.
- **Conception à base d'interfaces** — Les services implémentent des interfaces définies séparément. L'avantage de cette approche est que plusieurs services peuvent implémenter une interface commune et qu'un service peut implémenter plusieurs interfaces.

- **Localisable** — Les services doivent pouvoir être localisés à la fois au moment de la conception et de l'exécution, non seulement par identité spécifique, mais aussi par identité d'interface et type de service.
- **Instance unique** — À la différence du développement par composant logiciel, qui instancie les composants selon les besoins, chaque service est une instance unique à exécution permanente avec laquelle plusieurs clients communiquent.
- **À couplage lâche** — Les services sont connectés aux autres services et aux clients en utilisant des méthodes par messages découplées standard à réduction de dépendance, comme les échanges de documents XML.
- **Asynchrone** — En général, les services utilisent une approche de transmission de messages asynchrone, bien que cela ne soit pas obligatoire. En fait, de nombreux services utilisent parfois un mode de transmission asynchrone.

Une partie de ces critères, comme la conception à base d'interfaces et la localisabilité, sont également utilisés dans le développement par composant logiciel, mais c'est le total général de ces attributs qui différencient une application à base de services d'une application développée à l'aide d'architectures à base de composants logiciels comme J2EE ou .NET.

Conception à base d'interfaces

Dans le développement par composant comme par service, la conception des interfaces est telle qu'une entité logicielle met en oeuvre et expose une partie essentielle de sa définition. C'est ainsi que la notion et le concept d'"interface" ont essentiels à la réussite d'une conception, que ce soit dans les systèmes par composant logiciel ou dans les systèmes orientés services. Voici quelques définitions importantes associées aux interfaces :

- **Interface** — Définit un ensemble de signatures de méthode publiques, regroupées logiquement mais n'assurant aucune mise en oeuvre. Une interface définit un contrat entre le demandeur et le fournisseur d'un service. Toute implémentation d'une interface doit fournir l'ensemble des méthodes.
- **Interface publiée** — Interface à identification unique accessible via un registre qui permet aux clients de la localiser.[3]
- **Interface publique** — Interface accessible par les clients mais non publiée, et qui nécessite donc une connaissance statique de la part du client.
- **Double interface** — Les interfaces sont souvent développées par paires, de sorte qu'une interface donnée dépend d'une autre interface ; par exemple, un client doit mettre en oeuvre une interface pour appeler un demandeur étant donné que l'interface client offre un certain mécanisme de rappel. Ce concept a été introduit par les services Web.

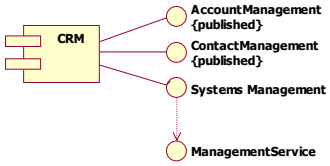

Figure 2 – Services mis en oeuvre

La Figure 2 montre la définition UML d'un service de gestion de la relation client (GRC), représenté sous la forme d'un composant UML, qui met en oeuvre les interfaces AccountManagement, ContactManagement et SystemsManagement. Seules les deux premières sont des interfaces publiées, même si la dernière est une interface publique. Note : l'interface SystemsManagement et l'interface ManagementService forment une double interface. Le service GRC peut implémenter n'importe quel nombre d'interfaces de ce type, et c'est cette capacité d'un service (ou d'un composant) à se comporter de façons multiples en fonction du client qui assure une flexibilité optimale dans la mise en oeuvre du comportement. Il est même possible de fournir de services différents ou supplémentaires à des classes spécifiques de clients. Dans certains environnements d'exécution, cette capacité permet aussi de prendre en charge différentes versions de la même interface sur un même composant ou service.

Comportement des interfaces

La définition d'une interface dans des langages tels que Java ou C#, ou comme le langage de description IDL, offre uniquement un ensemble de signatures de méthode. La définition indique le "quoi" sans aucune précision quant au

“comment.” Prenons, par exemple, l'interface Security de la Figure 3. Il est clair que les clients qui appellent une implémentation de cette interface sont capables d'appeler n'importe laquelle des trois méthodes publiques — mais est-ce bien le cas ?

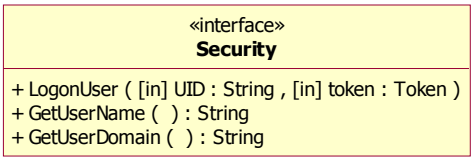

Figure 3 – Interface en langage UML

En définissant simplement le “quoi” nous ne sommes pas en mesure de représenter le fait que le client est dans l'incapacité d'appeler GetUserName( ) ou GetUserDomain( ) tant qu'il n'est pas connecté. L'automate à états ci-dessous démontre la dépendance, ou le comportement. Ce type de contrainte est souvent inclus dans la documentation sur la conception à base d'interfaces ; toutefois, comme elle n'est prise en charge par aucun langage de programmation, personne ne peut garantir que l'implémenteur d'une interface est en conformité avec une spécification comportementale quelconque.

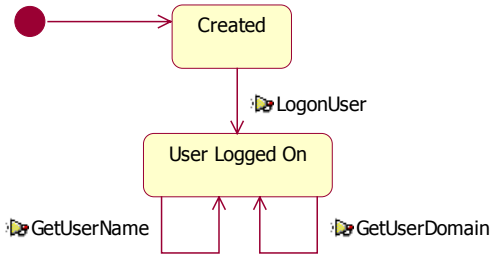

Figure 4 – Comportement d'une interface

Il n'en demeure pas moins que les entreprises évoluent de plus en plus vers des systèmes orientés services dans l'espoir qu'ils peuvent être plus faciles à intégrer et à chorégraphier pour réaliser des processus métier via des collaborations interservices. En conséquence, la notion de définition du comportement d'une interface et, surtout, du comportement d'ensembles d'interfaces associées, bénéficie d'une attention de plus en plus soutenue de la part de l'industrie. Malheureusement, il n'existe actuellement que peu d'approches standard en la matière.

Une approche pourrait consister à utiliser des modèles de conception, comme ceux présentés dans ce document, définis dans un langage standardisé tel que UMT, pour documenter les interdépendances entre les interfaces de service. Ce type de modèles peut être partagé, adapté et utilisé en soutien de standards spécifiques, le moment venu.

Par ailleurs, Rational a parrainé la spécification RAS (Reusable Asset Specification), qui fournit un mécanisme pour empaqueteret partager les ressources qui pourraient s'appliquer à ce problème. Par exemple, lorsqu'on utilise le mécanisme RAS pour distribuer les détails d'un service, il est possible d'y intégrer aussi le modèle décrivant son comportement. Dans ce modèle, un diagramme de séquence peut être utilisé ensuite pour montrer l'interaction requise entre les appels et l'interface.

Développement architectural de systèmes orientés services

Dans tout nouveau développement en matière de génie logiciel, il est très facile de supposer que l'on peut appliquer les mêmes techniques et utiliser les mêmes outils que ceux qui ont marché pour les projets précédents. Nous avons déjà indiqué que les composants et les services, même s'ils sont semblables, ne sont pas identiques ; ils ont des critères et des modèles de conception différents. Dans cette section, nous aborderons une conséquence pratique importante de cela. *Tout bon composant transformé en service ne donne pas forcément un bon service.*

Organisation en couches de la conception d'applications

Cette tendance à résoudre des problèmes inédits avec des solutions dépassées n'est pas nouvelle. De même, lorsque les développeurs ont commencé à créer des systèmes par composant logiciel, ils ont essayé de rapporter leur expérience en matière de développement orienté objets, avec des problèmes similaires. Avec plus d'expérience, on a compris que la technologie et les langages orientés objets pouvaient être très efficaces pour implémenter des composants, même s'il faut comprendre les compromis que cela représente tant en termes de décisions que d'implémentation - comme les compromis entre héritage et agrégation pour mettre en oeuvre un comportement polymorphe, ou la refonte des bibliothèques de classes pour pouvoir les utiliser dans des ensembles de composants plutôt que comme base de l'application monolithe C++.

De manière similaire, nous voyons *les composants comme la meilleure manière de mettre en oeuvre des services*, même s'il faut comprendre qu'une application par composant exemplaire ne fait pas forcément une application orientée services exemplaire. Nous y voyons une occasion de prendre appui sur les développeurs par composant et les composants existants de votre entreprise, une fois que le rôle joué par les services dans une architecture applicative aura été compris. La clé pour faire la transition est de réaliser qu'une approche orientée services implique une couche supplémentaire d'architecture applicative. La Figure 5 ci-dessous montre comment des couches technologiques peuvent être appliquées à une architecture applicative pour offrir des implémentations à granulation plus grossière au fur et à mesure que l'on se rapproche des utilisateurs de l'application. Le terme inventé pour cette partie du système est “application edge” ("périphérie applicative"), pour refléter le fait qu'un service est une manière efficace d'exposer une vue externe d'un système, avec réutilisation et composition internes à l'aide de la conception par composant classique.

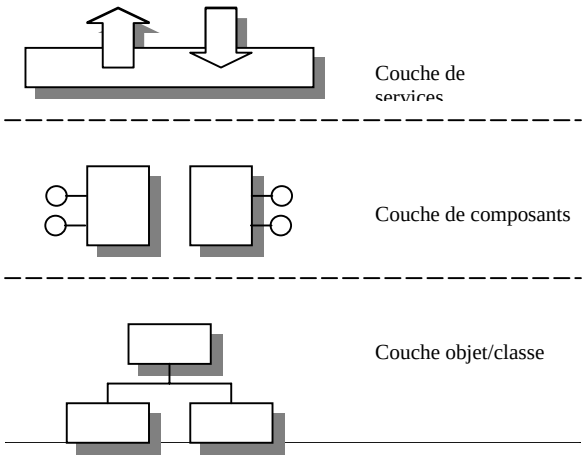

Figure 5 – Couches d'implémentation d'applications

En général, le passage du modèle orienté objets au modèle par composant logiciel prenait entre 6 et 18 mois, au cours desquels les développeurs apprenaient cette nouvelle technologie et ses nouveaux critères. Le passage aux systèmes orientés services devrait prendre moins de temps, du moins l'espère-t'on. À cette fin, les développeurs devront comprendre les défis, les compromis et les décisions de conception qui permettent de développer et de réutiliser des composants en support des applications orientées services.

Exemple de modèle de client

Pour illustrer comment les composants et les services interagissent dans la réalisation d'une application, prenons un exemple qui gère certaines informations sur la relation client, comme défini dans le diagramme de classes UML de la Figure 6 ci-dessous.

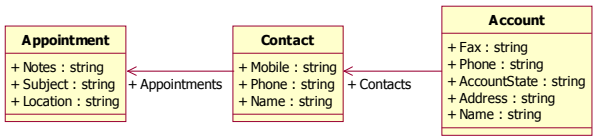

Figure 6 – Modèle de client logique

Dans les sections qui suivent, nous décrirons comment les développeurs transcriraient ce modèle logique en un modèle d'implémentation pour des applications par composant logiciel, puis pour des applications par service (en gardant à l'esprit que ce modèle ne décrit en aucune façon le comportement du système). Il apparaîtra clairement qu'une grande partie de ces étapes de traduction peuvent être générées automatiquement. Rational Software dispose d'outils pour modéliser l'architecture applicative, la récolte et l'application de ces patterns, et la gestion de ces artefacts de modèle/code tout au long du cycle de vie de développement.

Une conception par composant logiciel

Comment notre modèle logique donné en exemple serait-il traduit dans une conception par composant (du type COM ou J2EE) ? Une conception par composant logiciel pour le modèle est illustré par la Figure 7 ci-dessous.

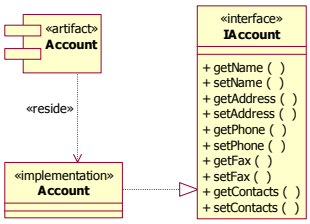

Figure 7 – Diagramme de composants générique

Il y a une caractéristique de conception *clé* associée à l'interface ci-dessus. À savoir que, pour des plate-formes de composants existantes, il existe un schéma commun ; pour chaque attribut de la classe d'analyse nous avons prévu deux opérations — une opération qui établit la valeur et une opération qui retourne la valeur. Pour les composants locaux, le temps système pour un appel de méthode est négligeable, et pour les objets distants les mécanismes RPC (appels de procédures à distance) sont optimisés pour minimiser le temps système. Dans la plupart des cas, dans une application, le client n'a besoin que d'un sous-ensemble des propriétés auquel il peut accéder selon ses besoins.

Une conception orientée services

Le modèle de conception ci-dessus illustre la bonne manière d'envisager des implémentations par composant logiciel, où chaque instance de composant représente un objet unique ; par exemple, chaque contact individuel dans notre base de données GRC devient, logiquement, un composant distinct. Ainsi, l'identité du composant est liée à l'identité du contact.

Néanmoins, pour un service, il n'y a qu'une seule instance qui gère un ensemble de ressources, et donc elles sont essentiellement sans état. Cela signifie que nous devons envisager un service comme un objet *gestionnaire* qui peut créer et gérer des instances d'un type ou d'un ensemble de types. Cela résulte dans un modèle de conception qui utilise des *objets valeur* (un schéma courant dans les systèmes répartis où l'état persiste pour les transferts entre composants) qui réprésentent l'état de l'instance - des objets qui sont en fait simplement des états sérialisés. L'un des résultats intéressants de cette approche est que, si nous pouvons définir les règles pour prendre une définition de composant, comme le modèle de la Figure 7, et la transformer en un service, nous pouvons mettre en oeuvre cette sérialisation sous la forme d'un pattern. La création et l'utilisation de ces modèles sont possibles grâce à Rational XDE.

Ce passage de l'état de fournisseur à celui de demandeur implique que, au lieu d'utiliser un grand nombre de petites opérations pour extraire l'état du composant, une seule opération est employée. Cela a un certain nombre d'implications pour l'utilisation de réseaux pour les services distants (et la plupart des services sont distants), ainsi que pour le comportement des demandeurs dans le cas d'objets valeur volumineux. Il y a aussi une autre implication : le demandeur reçoit une copie de l'état d'une entité donnée, mais est-ce que cette copie est bien d'actualité ? Nous

savons, lorsque nous récupérons le cours d'une action ou un bulletin météo, qu'il est possible que ces informations ne soient pas à jour, mais nous sommes conditionnés pour l'accepter. Nous sommes également conditionnés par le type de données — les données de cotation boursière deviennent périmées plus vite que les données météorologiques. Dans le type d'architecture décrit ici, les demandeurs doivent être conditionnés pour accepter des copies d'état. Pour des informations plus détaillées, reportez-vous à la section “Impact des interactions à granularité grossière ” plus loin dans ce document.

Sur la base de ces critères, quel devrait être l'aspect d'un service ? Le fragment de modèle de la Figure 8 illustre comment un pattern de ce type peut être décrit au niveau de la conception, en montrant les interfaces publiées par le composant et les objets valeur manipulés par l'interface.

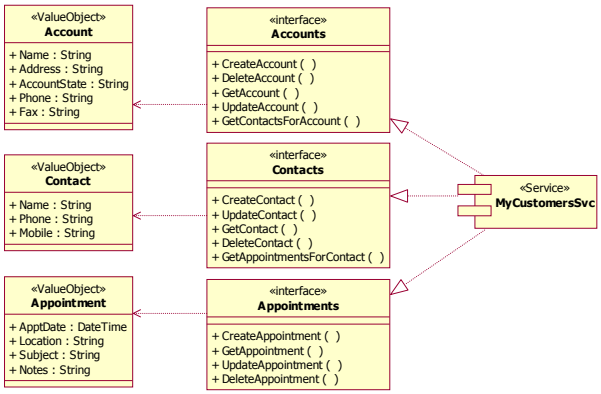

Figure 8 – Conception orientée services générique

Maintenant que nous savons comment les services peuvent être conçus, intéressons-nous à certains des attributs de cette conception particulière. Pour commencer, il est clair que beaucoup plus d'informations sont transmises dans les objets valeur qu'avec un simple “get” ou “set” transmis par le fournisseur, MyCustomerSvc, au demandeur pour une interaction donnée, et que cela peut affecter la bande passante du réseau. Toutefois, étant donnée le nature des services Web, il est clair que les protocoles utilisés dans la mise en oeuvre de services diffèrent largement de ceux utilisés dans les implémentations par composant logiciel. Une plate-forme comme celle-ci met un fardeau supplémentaire sur les épaules de l'architecte ou de l'informaticien qui doit choisir soigneusement les objets valeur et leur composition pour maximiser le contenu de chaque objet valeur sans surcharger le réseau.

Concept d'antémémoire dans la conception orientée services

Revenons sur la notion que nous avons introduite à la section précédente, soit la transmission de copies “périmées” d'information d'un fournisseur à un demandeur. Par exemple, si je suis en train de développer une application de gestion de portefeuille, je ne veux pas avoir à demander à un service Web la cote actuelle d'un titre à répétition pour chaque titre, en transmettant 3 à 5 caractères pour les données et 5 à 7 caractères pour la cote retournée. Cela peut résulter dans une charge inacceptable sur le réseau et le fournisseur de services. Au lieu de cela, le demandeur devrait demander le contenu de l'intégralité du portefeuille, soit en transmettant une liste de symboles ou l'identificateur du portefeuille au service, et en récupérant toutes les informations pour chaque titre. Maintenant, si l'utilisateur avait demandé simplement une mise à jour associée à un symbole spécifique, cela peut s'apparenter à de la surpuissance ; néanmoins, le demandeur peut maintenant mettre en antémémoire les résultats et, si l'utilisateur lui demande la mise à jour d'un autre symbole, la demande peut être satisfaite à partir de l'antémémoire. Le travail à la charge du demandeur consiste désormais à identifier la durée de “location” des données. Ainsi, dans le cas d'un portefeuille, si je sais que le service de cotation boursière a un retard de 20 minutes, je peux préférer travailler sur une marge de 25% et mettre les résultats en antémémoire pendant 5 minutes.

Ce pattern est très courant dans les systèmes d'information. Chaque fois qu'un utilisateur extrait une commande d'un système de gestion de commandes, il reçoit effectivement une copie de la commande puisqu'un autre utilisateur peut être en train de la mettre à jour au même moment (sauf si le système bloque tout autre accès à la commande). Il serait intéressant que le fournisseur de services Web puisse effectivement, dans le cadre de son interaction avec le demandeur, identifier la durée de mise en antémémoire. Ce type de problèmes a bien été compris dans des systèmes

comme Microsoft Message Queue Server (MSMQ) et IBM MQSeries, où les délais d'inactivité et d'expiration sont gérés de manière courante.

Nous reviendrons sur ce problème plus loin dans ce document et donnerons un certain nombre de directives sur le développement des demandeurs et des fournisseurs pour pallier ce problème.

Conception des services Web XML

Les services Web XML font désormais partie de notre vocabulaire et bénéficient du soutien d'un grand nombre de fournisseurs, ainsi que d'un nombre croissant de plates-formes et d'outils pour les déployer. Les éléments de la pile de service Web ont été largement décrits par ailleurs, et nous ne reviendrons pas dessus de manière exhaustive dans ce document. La définition qui suit émane du groupe de travail sur l'architecture des services Web du W3C (World Wide Web Consortium) :

> *Un service Web est une application identifiée par une adresse URL, dont les interfaces et les associations peuvent être définies, décrites et localisées par des artefacts XML, et qui assure le support d'interactions directes avec d'autres applications en utilisant des messages XML via des protocoles Internet.*[4]

Voyons brièvement comment quelques-unes des technologies utilisées aujourd'hui dans les services Web s'appliquent à l'image que nous avons dressée à la Figure 1 en décrivant la terminologie de l'architecture orientée services.

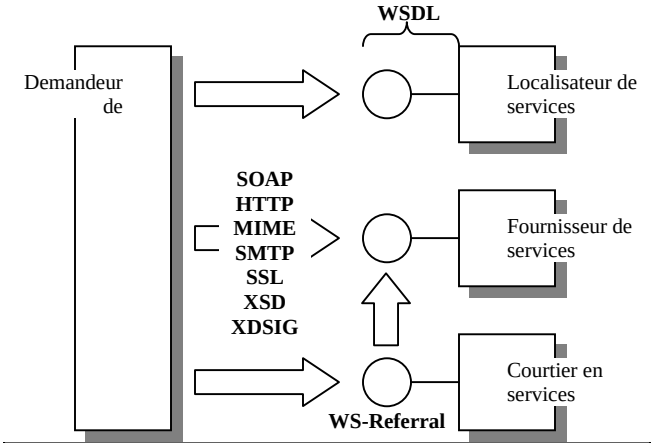

Figure 9 – Standards de service Web XML

Pour commencer, il y a une idée erronée selon laquelle tous les services Web utilisent des message XML, avec le protocole SOAP (Simple Object Access Protocol) sur HTTP — le protocole de fait pour le Web. Cela n'est pas tout à fait vrai. Un message de service Web peut utiliser le langage XML, mais il pourrait transporter des données binaires ; il peut utiliser HTTP comme transport, mais il pourrait très bien utiliser SMTP ou un autre moyen. Pour la description et la localisation de services Web, il existe deux standards bien définis : WSDL (Web Services Definition Language) et UDDI (Universal Discovery, Description and Integration).

Cette flexibilité dans les formats et les protocoles de transport pose l'un des problèmes actuels avec les services Web — l'interopérabilité. Comment, étant donné le choix de formats SOAP, d'enveloppes, de protocoles de transport, et ainsi de suite, deux implémentations peuvent-elles échanger des informations ? Le groupe WS-I (Web Services Interoperability) a décidé de s'attaquer à ce problème en essayant de proposer à l'industrie des directives quant à l'utilisation des standards actuels et nouveaux. Les sociétés d'ingénierie et de services en informatique aident également en offrant des environnements de développement de services Web comme IBM WebSphere Studio Application Developer Integration Edition, qui crée des services Web avec des formats et des protocoles de transport multiples afin d'utiliser l'ensemble le plus rapide ou le mieux adapté selon les cas.

Modèles de conception et de mise en oeuvre de services Web

Les services Web ne changent pas grand-chose aux processus d'analyse et de conception qui entourent les critères fonctionnels d'une application — une application de traitement de sinistres doit toujours traiter des sinistres ! Ce que nous introduisons, c'est un ensemble de contraintes et de problèmes potentiels dans le domaine des critères non fonctionnels. Les sections qui suivent décrivent quelques-uns des écueils possibles.

Performances et fiabilité

La question est souvent posée de savoir si les capacités requises pour assurer les performances, la fiabilité et l'évolutivité des services Web peuvent être fournies par une architecture sur base HTTP ou SOAP, qui sont des protocoles à la fois lents et peu fiables en soi. Il convient tout d'abord de définir la double notion de “lent et peu fiable” et de réaliser que même des transports fiables reposent finalement sur des moyens peu fiables. Lors de la conception et du développement de solutions à l'échelle de l'entreprise, nous devons toujours garder à l'esprit les critères fonctionnels et non fonctionnels et nous assurer que les bons compromis sont faits et les bonnes décisions sont prises pour aider à atteindre les objectifs de l'entreprise.

Par exemple, lorsqu'on utilise SOAP sur HTTP, il est toujours possible de développer des protocoles et des interactions au niveau de l'application, qui offrent des capacités supplémentaires pour les accusés de réception de messages et la sécurité. Néanmoins, si nous considérons que certains services communiquent dans le même contexte de sécurité ou d'application, nous pouvons envisager d'utiliser un autre moyen de transport que HTTP de toute manière. Voyez l'exemple de la Figure 10.

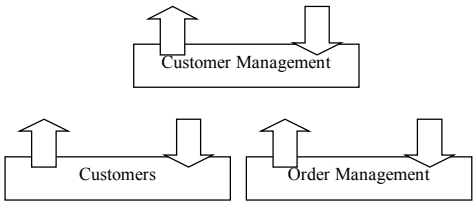

Figure 10 – Dépendances interservices

À la base, tous les clients externes interagissent avec le service Gestion de la clientèle ; mais celui-ci interagit avec deux services internes. Ici, la décision est la suivante : Pourquoi aurions-nous besoin de la flexibilité de HTTP et SOAP pour ces communications entre services internes ? Supposons que la performance soit notre critère clé pour l'interaction entre les services Gestion de la clientèle et Clients. Si c'est le cas, nous pouvons décider d'utiliser une communication RPC par composant (comme Microsoft .NET Remoting ou Java RMI) qui offre des formats de codage binaires et un plus haut niveau de performances. D'un autre côté, le critère clé pour la passation d'une commande entre les services Gestion de la clientèle et Gestion des commandes est la garantie de remise, et donc nous pourrions utiliser une technologie de file d'attente (comme IBM MQSeries ou MSMQ) pour remettre le message lorsque la fiabilité l'emporte sur les performances.

Il est très important de réaliser que même si les services Web présentent un modèle simple et un ensemble de protocoles simples et flexibles, vous n'êtes pas limités à ces choix. Tout comme WSDL a déjà des associations à la fois pour GET/PUT SOAP et HTTP, il est essentiel d'offrir aux demandeurs des choix supplémentaires. Par exemple, un service spécifique peut exposer un message en utilisant une association avec une file d'attente de messages *et* une association avec SOAP, de façon à ce que le demandeur puisse choisir l'association la plus appropriée. Dans ce cas, le fournisseur peut aussi offrir des encouragements, comme un niveau de service garanti si la file d'attente de messages est utilisée mais pas de garantie de service pour une conversation HTTP.

Une autre décision de conception à prendre le plus tôt possible est de savoir si les échanges de messages seront idempotents, commutatifs, ou les deux. Être *idempotent* implique que si le message arrive plus d'une fois, et *est traité*, dans chaque cas il n'y a pas de conséquences négatives. Être *commutatif* signifie que deux messages associés peuvent arriver dans n'importe quel ordre sans conséquences négatives. Si la conception d'un service peut être telle que les échanges de messages sont identifiés comme au moins idempotents, alors des transports peu fiables (et moins chers) paraissent la meilleure option.

De la même manière que la sécurité (non abordée dans ce document) est en fait un ensemble de choix, allant de simple et économique à complexe et onéreuse, les objectifs de conception en matière de performances, de fiabilité et d'évolutivité reviennent à un ensemble de décisions : De quoi avez-vous besoin ? Combien pouvez-vous investir ? Les services offrent autant de solutions, et tout autant de choix, que les approches de développement existantes.

Évolutivité grâce au comportement asynchrone et à la mise en file d'attente

Comme mentionné dans l'introduction à l'architecture orientée services, il est avantageux de rendre vos services Web asynchrones par nature. En raison du temps système supplémentaire associé aux services Web et de l'expectative de services distants par nature, il est essentiel de réduire le temps passé par un demandeur en attente de réponses. En rendant un appel de service asynchrone, avec un message de retour distinct, nous permettons au demandeur de continuer l'exécution pendant que le fournisseur a une chance de répondre. Cela ne signifie pas que les services synchrones sont à rejeter, mais l'expérience a montré qu'un comportement asynchrone des services était souhaitable, en particulier lorsque les coûts de communication sont élevés ou les temps d'attente imprévisibles.

Voyez l'exemple de la Figure 11.

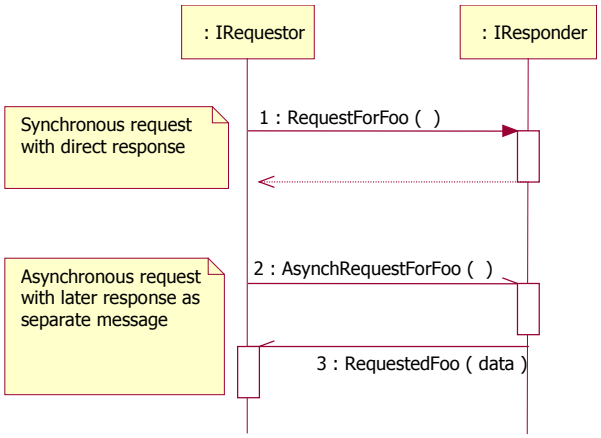

Figure 11 – Synchrone / asynchrone

Le comportement décrit à la Figure 11 est un grand pas en avant vers la mise en oeuvre de services Web hautement évolutifs. En rendant un appel de service asynchrone, le fournisseur peut utiliser plusieurs filières pour gérer plusieurs demandes client. Il y a bien d'autres choses à faire pour prendre en charge un mode d'exploitation asynchrone que simplement répondre au client rapidement. Tout d'abord, vous devez spécifier des doubles interfaces — le demandeur aura besoin de transmettre une adresse de retour à un service qui implémente une interface pouvant accepter le message retourné. Cela implique la nécessité de gérer l'état dans la conversation entre les parties. Pour en apprendre davantage sur les différentes méthodes applicables, observez la conception des sessions Web qui ne sont pas basées sur des services Web.

Toutefois, ce type de service n'est évolutif qu'à un certain degré. Pour les services qui attendent une charge très importante, il serait nécessaire de découpler la partie qui écoute le demandeur de la partie qui répond à la demande elle-même. Il s'agit d'un pattern déjà bien connu, où une file d'attente de messages est utilisée pour découpler une façade de service de l'implémentation du service. Le diagramme de la Figure 12 montre comment le fournisseur est implémenté en utilisant une file d'attente pour découpler la demande de l'implémentation.

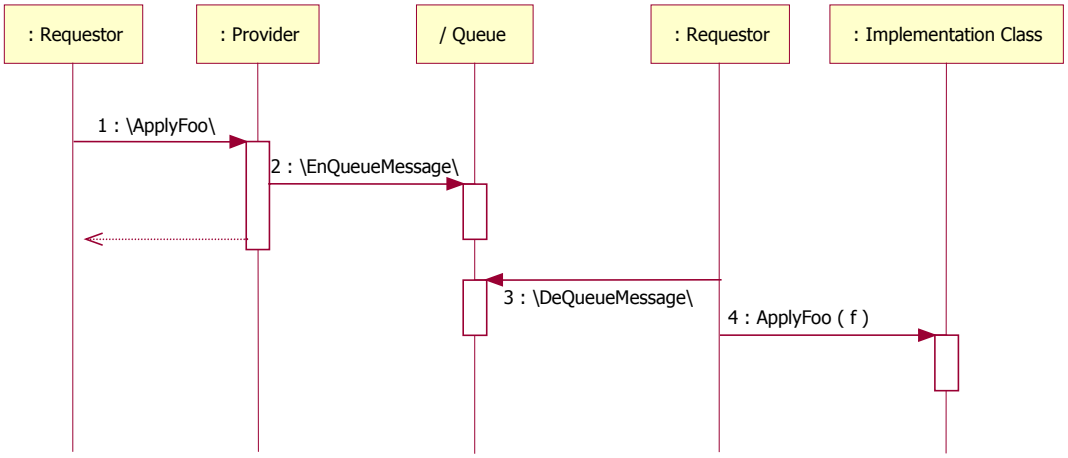

Figure 12 – Implémentation mise en file d'attente

Ce type de pattern peut être facilement mis en oeuvre à la fois dans .NET et J2EE en utilisant les services offerts par ces plates-formes : MSMQ pour .NET et Java Message Queue Service (JMS) ou les composants logiciels commandés par messages pour J2EE. Le développeur bénéficie ainsi d'un modèle d'évolutivité plus simple — au lieu de gérer un ensemble de filières avec synchronisation des demandes, l'implémentation peut simplement ajouter d'autres écouteurs de file d'attente pour extraire des messages de la queue, même sur des machines multiples.

La "location" d'informations revisitée

Lorsque nous pensons à la location d'informations, nous voyons cela plus comme s'il s'agissait d'emprunter un livre à la bibliothèque que de louer une maison ou une voiture. Implicitement, chaque fois qu'un demandeur fait une demande de service, il demande une copie de certaines informations — il reçoit toujours uniquement un instantané d'un état à un moment donné. Cela peut poser problème par manque d'explication et absence de prise en compte. Une stratégie consisterait à ce que le fournisseur inique un délai d'*expiration* en même temps qu'il communique les informations. Le demandeur pourrait aussi recevoir un “ticket” qui lui permettrait d'étendre la durée du "bail" en demandant si les informations sont toujours valables, puis en faisant en sorte que le serveur réinitialise la bail sans avoir à récupérer les données une nouvelle fois.

Ce problème peut paraître si fondamental que l'on pourrait s'attendre à ce que HTTP, SOAP ou l'un des protocoles de transport s'en charge pour nous. Nous pourrions réutiliser la sémantique d'antémémoire HTTP, qui permet aux navigateurs et aux coupe-feu de mettre des pages en antémémoire, mais cela ne se ferait pas vraiment sous le contrôle du fournisseur, et il se peut que le demandeur n'utiliser pas HTTP comme protocole de transport. Une option consisterait à intégrer ce type de support dans votre échange de documents, de sorte que les messages entre le demandeur et le fournisseur encode les informations de bail pour les clients, comme illustré Figure 13.

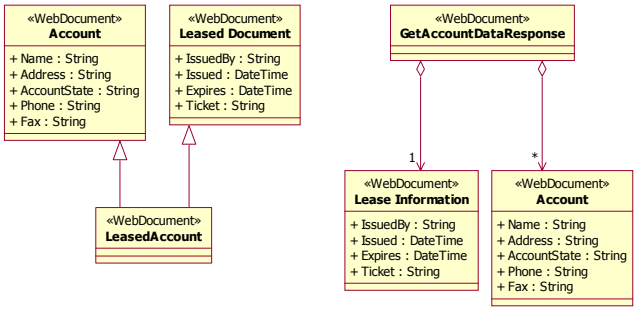

Figure 13 – Deux implémentations de la "location" d'informations

La Figure 13 montre deux choix d'implémentation du modèle de location d'informations. Le premier illustre l'utilisation de l'héritage pour spécialiser le document XML Account dans une forme spéciale qui n'est pas seulement un compte (Account) mais aussi un document loué (Leased Document) et qui, donc, inclut les informations

supplémentaires. Dans la deuxième implémentation, les informations de bail sont retournées avec le compte sous la forme d'une partie distincte du message de réponse. Ces deux approches sont tout aussi valables, même si elles résultent dans des données structurées de manière différente. Il s'agit plus d'un choix de style : héritage ou agrégation.

Modèle de conception de services Web résultant

La Figure 14 montre comment la conception générique de services de la Figure 7 peut être modélisée en utilisant un profil UML spécifique à la conception et au développement de services Web. Ce profil, très simple, introduit uniquement deux nouveaux *stéréotypes* (extensions du langage XML existant), pour "WebService" et "WebDocument". En réutilisant la sémantique d'interface déjà présente dans le langage UML, nous pouvons visualiser facilement les aspects publiés d'un service, comme dans WSDL.

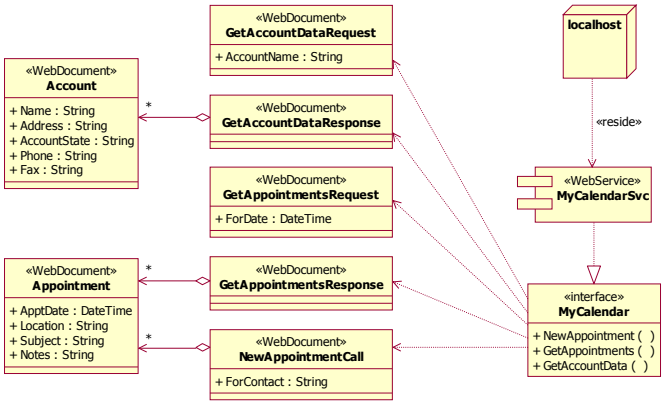

Figure 14 – Conception de service Web XML

Le tableau ci-dessous montre comment les éléments de profil de la Figure 14 sont associés aux artefacts du langage WSDL pour le service MyCalendar.

Artefact WSDL	Élément UML	Commentaire
service	"WebService"	Le service est représenté sous la forme d'un composant UML qui réalise une ou plusieurs interfaces et réside sur un emplacement particulier. La relation "reside" capturera les informations d'emplacement URL effectives.
portType	Interface	Chaque portType (type de port) est représenté sous la forme d'une interface UML par un ou plusieurs services. La relation de réalisation capturera les informations d'association.
message	"WebDocument"	Chaque message est représenté sous la forme d'une classe UML. Une correspondance doit être établie entre XML Schema et UML pour modéliser le message et la structure des parties.
part	Attribut ou fin d'association	Chaque partie du message peut être représentée sous la forme d'un attribut UML sur le "WebDocument" ou d'une association avec un autre "WebDocument".
address location	Noeud	Le noeud représente le serveur sur lequel réside le service. Le noeud peut identifier un ensemble de services résidants et un service peut résider sur plusieurs noeuds.

Il est important de noter que cette méthode de conception de services Web favorise la réutilisation à la fois des documents qui définissent l'échange de données et des interfaces supportées par ces services. Il s'agit là d'une capacité essentielle pour la conception de solutions à l'échelle de l'entreprise. Il est préférable que tous les services qui interagissent avec le document Account le fassent sur la base de la même définition de document. Nous voyons également que les interfaces opérationnelles non publiées, comme SystemsManagement (Figure 2), peuvent être

définies par des experts dans ce domaine, puis être mises à disposition de façon à être implémentées à tous les niveaux de la solution.

Conclusion

Alors que certains s'attendent à ce que les services Web soient une véritable révolution dans l'industrie du logiciel, d'autres personnes pensent que les services Web ne sont que du battage médiatique. Comme c'est souvent le cas avec les nouvelles technologies, la vérité se situe quelque-part entre les deux. Comme nous l'avons vu, il existe des différences fondamentales dans l'architecture des applications orientées services ; néanmoins, les techniques existantes de développement par composant logiciel restent valables pour la mise en oeuvre de ce type de services.

Il est possible de dériver à la fois des implémentations par composant logiciel et par service à partir d'un seul et même nom de domaine, comme illustré par le modèle "Account, Contact, Appointment" de la Figure 6. Les autres conclusions principales à tirer sont :

- Il est possible de dériver des modèles de services à partir de modèles par composant logiciel *existant*.
- Il est possible d'appliquer des structures et des modèles communs pour aider à la transformation vers des modèles orientés services.

Ces structures et modèles communs peuvent être généralisés, automatisés, puis instanciés avec un outil de transformation de modèle à modèle et de modèle à code tel que Rational XDE (voir la figure ci-dessous).

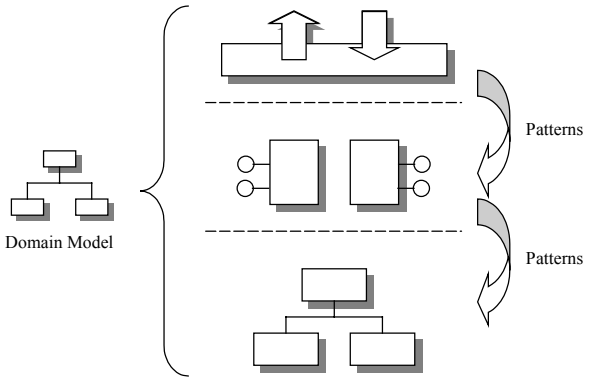

Figure 15 – Approche de la mise en oeuvre de services

Cette voie offre de nombreux avantages : elle réduit les temps de développement, augmente la prévisibilité de l'implémentation et améliore la qualité grâce à la réutilisation de modèles éprouvés. Le fait d'ajouter une pratique d'excellence telle que “ Utiliser les architectures orientées services ” à votre expertise et de codifier une partie de ces meilleures pratiques dans des ensembles d'outils d'automatisation comme Rational XDE vous permettra de faire les bons choix et les bons compromis pour mettre en oeuvre les services, de manière plus rapide et plus fiable.

Références

[1] Pour plus d'informations, voir http://www.rational.com/uml/.
[2] Large-Scale, Component-Based Development par Alan W. Brown, Prentice Hall, 2000
[3] Public versus Published Interfaces par Martin Fowler, IEEE Software, mars/avril 2002 (Vol. 19, No. 2)
[4] W3C Web Services Architecture Requirements; http://www.w3.org/TR/2002/WD-wsa-reqs-20020429
[5] Web Services Security, Version 1,0; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp
[6] Web Services Referral Protocol, Draft; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[7] XML Web Services Basics; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[8] Rational Developer Network http://www.rational.net/