Université de Technologie de Troyes
DEA RACOR – Option RFCI

Programmation par aspects et services web

NASSRALLAH Rabih
Étudiant DEA, UTT
nassrallah@ensm-douai.fr
http ://nassrallah.free.fr
Département GIP
École des Mines de Douai

BOURAQADI Noury
Encadrant du stage, ENSM Douai
bouraqadi@ensm-douai.fr
http ://csl.ensm-douai.fr/noury
Département GIP
École des Mines de Douai

Dépt. G.I.P
École des Mines de Douai
941 rue Charles Bourseul - B.P. 838
59508 Douai Cedex - France

Stage de recherche
École des Mines de Douai
Mars 2003 – Juillet 2003

Table des matières

Remerciements

Durant mon stage de recherche dans le département GIP (Génie Informatique et Productique) à l'École des Mines de Douai, j'ai eu le privilège de travailler au laboratoire CSL (Computer Science Laboratory) où j'ai rencontré des personnes qui ont contribué à la réussite de mon travail.

Je commence par remercier mon encadreur Noury Bouraqadi (Maître de conférence à l'École des Mines de Douai) pour sa grande disponibilité, ses compétences, mais aussi pour son encadrement. Je remercie aussi tous les autres membres du laboratoire CSL et en particulier Houssam Fakih pour les discussions intéressantes que nous avons eu pendant la période du stage.

Je remercie sincèrement Marc Lemercier (Maître de Conférence), mon suiveur à l'UTT, pour ses remarques pertinentes. Je remercie beaucoup, Dominique Gaïti (Responsable du DEA RACOR), pour ses conseils et son aide.

Quant à vous chers parents, je ne trouverai jamais suffisamment de mots pour vous remercier. Mille mercis...

Chapitre 1

Introduction

Une des limites des paradigmes de programmation traditionnels (programmation impérative, par objets, ...) est la difficulté d'implanter des applications complexes. Le développement de telles applications implique l'implantation conjointe des fonctions (services) qu'elles proposent et des propriétés non-fonctionnelles (propriétés d'infrastructure). Par exemple dans une application de commerce électronique, les fonctions que représentent la recherche d'un produit et la passation de commande dans une application doivent être implantées en prenant en compte les propriétés non-fonctionnelles que sont la communication distante, la synchronisation, la persistance, etc...

Ces différentes parties de l'application (services, propriétés d'infrastructure) se trouvent intimement enchevêtrées dans le code de l'application. D'où une complexité de développement source d'erreurs (bugs) et coûteuse en temps de développement. Le code résultant est difficile à comprendre, ce qui constitue un obstacle à la bonne maintenance et à l'évolution des applications. A cela s'ajoute le problème d'anomalie d'héritage qui limite les possibilités de réutilisation.

La programmation par aspects [KLM+97, BL01] est un nouveau paradigme qui rend la programmation plus facile en favorisant la réutilisation du code écrit. Il est basé sur le principe de *separation of concerns* [KLM+97] (littéralement séparation des préoccupations) qui consiste à séparer le code métier de l'application des propriétés non fonctionnelles (ex. la persistance dans l'application de commerce électronique).

La réflexion [Smi84] est une des approches possible qui permet la programmation par aspects. Un des intérêts de la réflexion est qu'elle offre différents niveaux de programmation. En effet, à l'aide d'un langage réflexif, il est possible non seulement de décrire les tâches qu'un programme doit réaliser, mais également la manière de réaliser ces tâches. Ces deux descriptions sont découplées car introduites à des niveaux différents

Les Services Web [W3C03] sont une des solutions proposés qui permettent l'intéropérabilité entre différentes applications écrites dans des langages différents. Ils répondent aux besoins des entreprises qui veulent faire coopérer plusieurs applications ensemble

L'objectif visé est d'identifier les différents aspects liés aux services web et de proposer une implantation. Ces différents aspects doivent être isolés et décrits sous forme d'une bibliothèque réutilisable. Le but étant de fournir un cadre pour développer les applications en se focalisant uniquement sur les services qu'elles rendent (le côté métier). Ces services seront rendus accessibles via le web en intégrant la bibliothèque précédemment définie.

Nous commençons ce rapport par une chapitre sur l'état de l'art, ou nous présentons les concepts fondamentaux. Ensuite nous présentons l'application mono-poste que nous avons développé et comment nous la transformons en service web sans toucher son code métier. Finalement nous présentons nos résultats et les éventuels futurs travaux dans la conclusion.

Chapitre 2

État de l'art

Dans ce chapitre nous présentons *les services web* et *la programmation par aspects*, qui représentent les deux concepts fondamentaux du travail. Nous introduisons le langage *XML*, utilisé dans les "services web", ensuite nous donnons un aperçu de ce que s'est les "services web". Nous parlons par la suite de la "programmation par aspects" et nous décrirons une approche pour programmer les aspects. Il s'agit de la *réflexion*.

2.1 XML : eXtensible Markup Language

XML [W3C00b, NB03] est un langage de balisage qui ressemble à HTML[1]. La norme XML laisse à l'utilisateur la liberté de choisir ses propres balises. Elle est plus générale que HTML, car elle permet à la fois la gestion (c'est-à-dire l'organisation, l'échange et la transformation) et la représentation des données. XML dérive de SGML [W3C86] et il a l'avantage d'être moins complexe et plus universel. XML est un langage souple, il définit un ensemble de règles pour organiser les données. Plusieurs outils ont été développés autour cette norme permettant la manipulation des données décrites dans le document XML.

2.1.1 Document XML

Un document XML est un fichier texte composé d'une en-tête optionnelle et d'un corps qui a une structure d'arbre. L'en-tête contient des informations sur la version de XML et le codage utilisés, elle peut aussi référencer une DTD [W3C01b] [2] ou un XML-Schema [W3C00a]. Une DTD est une grammaire qui définit la structure d'un document XML. Elle permet de valider que chaque composant est à la bonne place. XML-Schema, définit une grammaire plus évoluée que celle des DTD. Elle permet en plus de spécifier les types des données (chaîne de caractères, décimal, structures complexes, etc...)

Le corps d'un document XML contient des éléments. Il est représenté par un couple de balises ouvrante et fermante. Il commence toujours par la balise ouvrante de l'élément racine et se termine par la balise fermante du même élément. l'élément racine encapsule plusieurs sous-éléments, qui eux encapsulent d'autres sous-éléments. Un élément peut avoir des attributs. Les attributs apparaissent dans la balise ouvrante de l'élément. La norme XML 1.0 [W3C00b, Gir01] ne précise pas les cas d'utilisations des attributs et des éléments. Mais il est conseillé d'utiliser les éléments pour exprimer le contenue du document et les attributs pour décrire les relations entre les éléments ou les propriétés des éléments.

Le corps du document peut contenir des commentaires et des sections littérales. <!- - voici un exemple de commentaire - ->. Les sections littérales constituent un mécanisme qui permet

[1] Hyper Text Markup Language
[2] Document Type Definition

d'insérer dans un document XML des balises XML, qui seront traitées comme des chaînes de caractères.

<![CDATA[Voici le <contenu> de la section littérale]]>

XML est souvent utilisé pour l'échange de données entre applications. C'est la cas des "services web" par exemple. Les espaces de nommage [W3C99] permettent d'identifier d'une façon unique les balises d'un document XML. Nous utilisons des espaces de nommages communs aux applications, pour éviter qu'il y ait différentes interprétations du même document XML.

2.2 Services web

La définition des services web [W3C03] donnée par le W3C est la suivante :

> "A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols." [3]

Un service web [Man01, Got02, Som02, Cur02, Cha02, NB03] possède les caractéristiques suivantes :

- Il est accessible via le réseau.
- Il dispose d'une interface publique (les opérations qu'il implémente) décrite en XML.
- Ses descriptions (opérations, comment l'invoquer et où le trouver) peuvent être stockées dans un annuaire.
- Il communique en utilisant des messages XML. Ces messages sont transportés par des protocoles Internet (généralement HTTP).
- L'intégration d'applications en implémentant des services web, produit des systèmes faiblement couplés. Le demandeur du service ne connaît pas forcement le fournisseur. Ce dernier peut disparaître sans perturber l'application cliente qui trouvera un autre fournisseur en cherchant dans l'annuaire.

Nous pouvons classer les services web dans deux catégories ; les services web métier ou bien les services proposés par l'entreprise (ex. service de consultation d'un agenda distribué) et les services web techniques indispensables au bon fonctionnement des services web métier (ex. l'annuaire UDDI qui permet la découverte des services)

2.2.1 Architecture

La figure 2.1 de la page 6 donne les acteurs et les actions dans une architecture de service web [Kre01, Leb02].

Les acteurs sont :

Le demandeur du service : c'est l'utilisateur, l'application ou le service web qui invoque une ou plusieurs opération du service web.

Le fournisseur du service : c'est l'entreprise propriétaire du service ou l'hébérgeur de service.

le service d'enregistrement : c'est un service web, qui permet d'enregistrer les descriptions des services web – publiées par l'entreprise propriétaire des services – dans un annuaire distribué de service web.

Les actions sont (dans l'ordre) :

La publication du service : c'est le fait de diffuser la description du service web en invoquant l'opération "publish" du service d'enregistrement

[3] Un service web est un système logiciel identifié par son URI (Unified Resource Identificator) et dont les Interfaces publiques et les liens sont décrits en XML. Sa définition peut être découverte par d'autres systèmes logiciels. Ces systèmes peuvent interagir avec ce service web comme décrit dans sa description, en échangeant des messages XML transportés par des protocoles Internet.

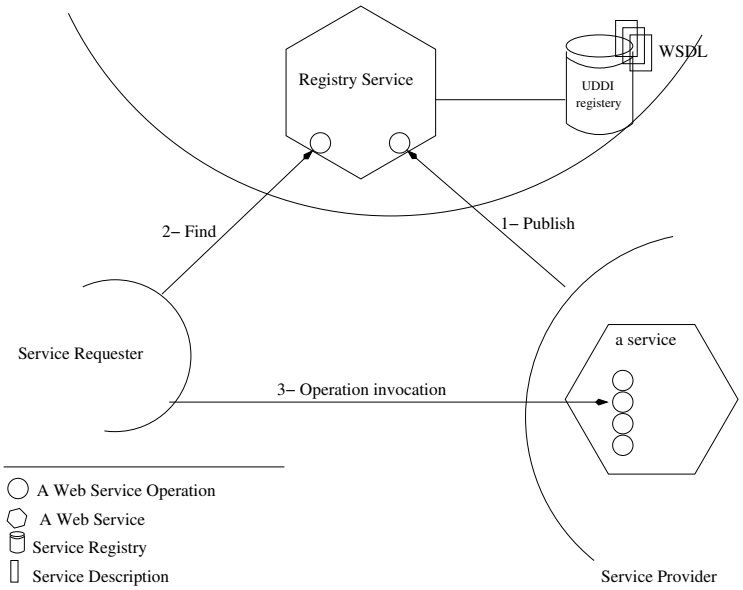

FIG. 2.1 – Architecture des services web avec publication/découverte dynamique

Le découverte d'un service : c'est le fait de trouver sa description et son URI en invoquant l'opération "find" du service d'enregistrement.

L'invocation du service : c'est l'invocation d'une ou plusieurs opérations du service web (ex. invocation de l'opération + d'un service web calculatrice).

La publication peut être *dynamique* ou *statique*. La publication est "dynamique" si le fournisseur enregistre ses services dans un annuaire distribué comme le montre la figure 2.1. Quand le fournisseur envoie les descriptions et les URI de ses services web, à ses clients (qu'il connaît), la publication est "statique" (figure 2.2).

De manière symétrique, la découverte des services se fait d'une façon *dynamique* ou *statique*. La découverte "dynamique" consiste à utiliser le service d'enregistrement (invocation de l'opération "find") pour trouver les listes des services web qui répondent aux besoins du demandeur de service (voir figure 2.1). La découverte "statique" est le résultat d'une publication "statique" comme illustré dans la figure 2.2.

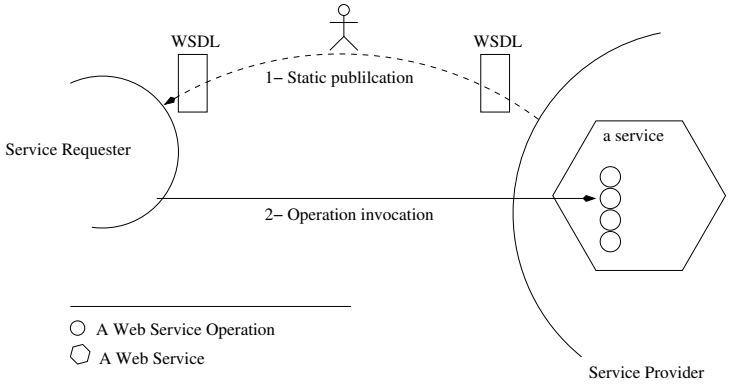

FIG. 2.2 – Architecture des services web avec publication/découverte statique

2.2.2 SOAP : Simple Object Access Protocol

SOAP [W3C00c] est définit comme un protocole d'échange de donnée dans un environnement distribué. Il est basé sur la norme XML et comporte trois parties : une enveloppe qui définit un cadre décrivant le contenu du message SOAP et comment l'analyser, un ensemble de règles de codage qui décrit les types de données (types simples, types complexes) et une convention de représentation des requêtes/réponse. SOAP est transporté via la couche applicative du le modèle OSI [ISO70]. Plusieurs protocoles peuvent être utilisés, pourtant seul l'association de SOAP avec HTTP est décrite dans le draft de la norme SOAP.

2.2.3 WSDL : Web Service Description Language

WSDL [W3C01a] est un langage de description des services web. Un document WSDL est un fichier texte au format XML. Il définit, de manière abstraite et indépendante du langage, l'ensemble des fonctionnalités offertes par le service web. Il décrit "comment" communiquer avec un service web et "ou" le trouver. Le langage WSDL, décrit les opérations qu'un service web peut exécuter, le type de messages XML (exemple : SOAP) qu'il peut traiter, les protocoles de communication qu'il supporte (exemple HTTP) et les points d'accès au service web (URI [4]).

2.2.4 UDDI : Universal Description, Discovery and Integration

UDDI [OAS02] est un annuaire distribué de services web et d'entreprises. Il se comporte en lui même comme un service web dont les opérations (publish et find) sont appelées via SOAP. C'est une architecture répartie qui permet aux fournisseurs de services web, d'enregistrer leur services et aux applications (demandeurs de service) de rechercher les services correspondant à leur besoins.

2.3 La programmation par aspects AOP

La programmation par aspects a été introduite par Kiczales [KLM+97, BL01] en 1997. Il s'agit d'un nouveau paradigme de programmation postérieure à la programmation par objets dont le but est de simplifier le développement, la maintenance et l'évolution d'applications complexes. Ce modèle de programmation, permet d'isoler dans des modules uniques et séparés les propriétés non-fonctionnelles (tous ce qui n'est pas code métier) du code métier (ou noyau) de l'application. Ces propriétés coupent d'une manière transversal le code métier ce qui rend difficile l'évolution et la maintenabilité de l'application.
Nous définissons ci-après les nouveaux termes qui sont introduits par la programmation par aspects :

aspect Un aspect représente une propriété non-fonctionnelle d'une application. La persistance, la distribution et la synchronisation sont des exemples de propriétés non-fonctionnelles.

Point de jonction Un point de jonction correspond à un moment d'exécution précis de l'application métier. C'est à ces moments précis que l'exécution du code métier de l'application s'arrête pour permettre l'intégration de l'exécution non fonctionnelle de l'application. L'appel des méthodes, l'accès à des variables, l'appel des constructeurs des classes sont des exemples des points de jonction.

tissage Le tissage correspond au mécanisme d'assemblage du code métier et du code non fonctionnel de l'application. Après le tissage, nous obtenons une application dont le code métier et le code non fonctionnel sont mélangés suivant les points de jonction déjà définis.

La figure 2.3 est un exemple d'une application développée en utilisant la programmation par aspect.

[4] Unified Resource Identifier

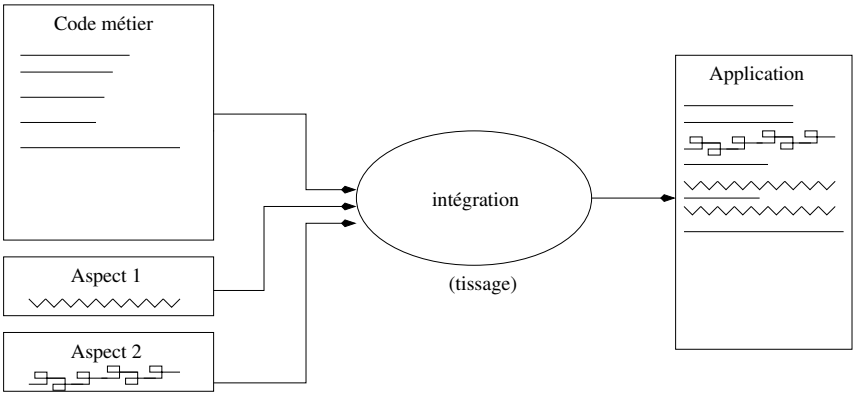

FIG. 2.3 – Construction d'une application avec la programmation par aspects

2.3.1 Réflexion

La réflexion [Smi84] est la capacité d'un système à raisonner et à agir sur lui-même. Ainsi, un système réflexif est capable de contrôler son propre fonctionnement. De la même façon, un programme réflexif peut observer et contrôler sa propre exécution.

Dans un langage réflexif nous pouvons changer la sémantique (l'interprétation) des message, accèder aux structures du programmes et contôler l'exécution. Nous identifions deux niveaux de programmation (figure 2.4) : Le niveau de base et le niveau méta. Le niveau de base correspond au côté métier de l'application. Alors que le niveau méta représente l'interprète de l'application. Les objets du niveau méta sont appelés "méta-objets".

Des protocoles appelés *Meta Object Protocols* (MOP) permettent de manipuler les méta-objets. Ils permettent l'accès à la structure du programme (classes, relations d'héritage, champs, méthodes, etc.), donnent accès aux mécanismes d'exécutions (envoi de messages, instanciation, recherche d'une méthode, etc.).

Les méta-objets peuvent être liés aux objets de base par un lien qu'on appelle *méta-lien*. Nous pouvons aussi connecter plusieurs méta-objets entre eux et les faire coopérer ensemble. La coopération se traduit par le fait qu'un méta-objet donne le contrôle au méta-objet suivant.

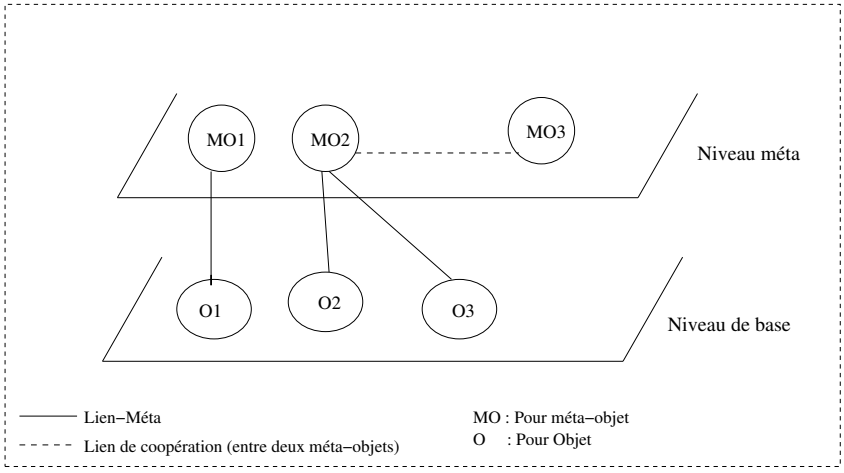

FIG. 2.4 – Les niveaux de programmation dans un langage réflexif

La réflexion nous permet de faire de la programmation par aspects [Bou99, BL02, BL03]. La

séparation entre le niveau de base et le méta-niveau permet une première séparation entre le code métier (au niveau de base) et les aspects (au niveau méta). Un aspect est vu comme une entité qui contient un ou plusieurs méta-objet qui représentent toute une seule propriété transversale (non-fonctionnelle). Un méta-objet ne doit appartenir qu'à un seul aspect, cela permettra la séparation des aspects. Les liens méta et les lien de coopération, permettent le tissage entre le code métier et le code non fonctionnel (correspondant aux aspects). Les messages envoyés à l'objet de base seront interceptés par le méta-objet. Ce dernier exécutera le code correspondant à l'aspect puis passe (ou non) le contrôle à un autre méta-objet qui exécutera son code non fonctionnel. L'exécution résultante est un mélange de l'exécution du code métier et des codes non fonctionnels. La figure 2.5 montre un cas de la programmation par aspects en utilisant la réflexion.

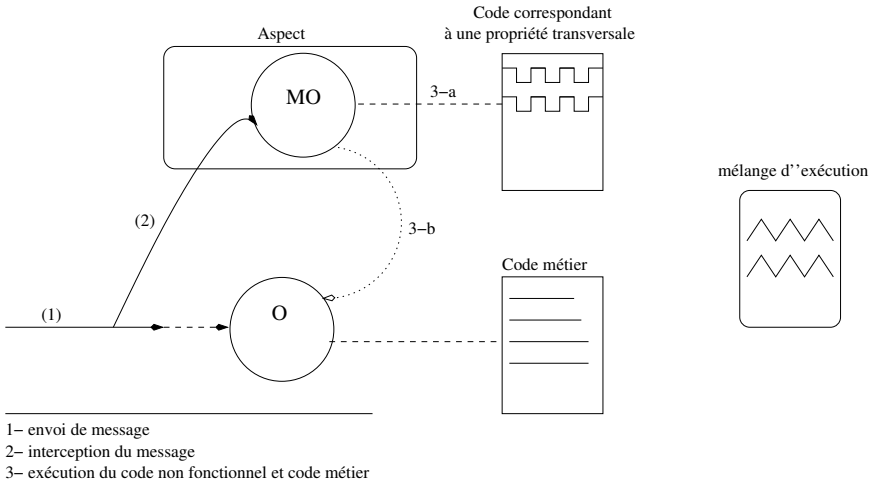

FIG. 2.5 – Programmation par aspects en utilisant la réflexion

Chapitre 3

Mariage AOP - Services Web

3.1 Introduction

"Absence-Mines" et "Agenda-Mines", sont deux applications client/serveur développées en interne à l'école des Mines de Douai. "Absence-Mines", gère les absences (congés, maladies, etc.) des employés de l'école et "Agenda-Mines" est un agenda distribué.
Pour demander un congé :

- L'employé remplit le formulaire d'absence (motif, type congé, date de début, date de fin, nombre de jours de congé) sur le site Intranet de l'école.
- "Absence-Mines" vérifie la demande (ex : compare le nombre de jours demandé au nombre de jours disponible). Ensuite, elle envoie un email au responsable hiérarchique du demandeur contenant la description de l'absence et la demande de validation.
- Le responsable valide ou refuse le congé.
- Si la demande est validée, l'application envoie un email de confirmation au demandeur et met à jour son agenda via "Agenda-Mines".
- Dans le cas échéant l'application envoie un email au demandeur pour lui informer que sa demande n'a pas été acceptée.

Nous nous sommes inspirés de ces deux applications pour définir le cahier de charge de l'application mono-poste "Employee's Vacations Manager" que nous avons développée. Nous montrons par la suite comment nous allons la transformer en un service web, sans toucher à son code fonctionnel.

3.2 Notre exemple : "Employee's Vacations Manager"

"Employee's Vacations Manager" est développée en Squeak[1] [IKM+97, BD01] qui est un Smalltalk libre (open source). Squeak est un langage pur objet, il est dynamiquement typé. Les appels de méthodes correspondent aux envoi de messages en Squeak. Commençons par une petite introduction à la syntaxe Smalltalk.

|aVariableName| est une déclaration d'une variable temporaire "aVariableName" en Smalltalk. Les variables sont séparés par un blanc (ex. |var1 var2 var3|).

m : arg est une méthode "m" qui prend un seul paramètre "arg".

add : aNumber to : anotherNumber est un exemple d'une méthode qui prend deux paramètres ("aNumber" et "anotherNumber").

^anExpression retourne le résultat de l'expression.

obj m : arg correspond à l'envoi du message "m" à un objet particulier "obj" avec "arg" en argument. Exemple : "2 + 3" , le message "+" est envoyer à l'objet "2" avec "3" en argument.

[1] http ://www.squeak.org/

Les classes “Calendar” et “Event” sont utilisées par les entités qui composent “Employee’s Vacations Manager”. “Calendar” est un agenda constitué d’un ensemble d’événements (“Event”). Un événement est un objet de la classe “Event”, il est caractérisé par sa description, sa date de début et sa durée.

Revenons à “Employee’s Vacaions Manager” qui est composé de trois briques de base :

- La classe VacationManager correspond au gestionnaire des absences “Vacation Manager”. Elle implémente les méthodes suivantes :

  requestVacation : aVacation sentBy : personName permet à un employé (personName est un String) de demander un congé (aVacation) . Le congé est un événement, son comportement est décrit dans la classe Event .

  approveTheVacation : aVacation of : personName valide la demande de congé.

  rejectTheVacation : aVacation of : personName rejette la demande de congé.

- La classe CalendarManager correspond gestionnaire des agendas “Calendar Manager”. Elle fait la correspondance entre chaque employé et son agenda. Un agenda est un objet de la classe Calendar . CalendarManager implémente les méthodes suivantes :

  addEvent : anEvent toTheCalendarOf : personName permet d’ajouter un événement à l’agenda d’une personne.

  findCalendarOf : personName trouve l’objet “agenda” d’un employé.

  associateCalendar : aCalendar to : personName associe un objet “agenda” à un employé.

- La classe Messenger représente le serveur de messagerie “Messenger”. Elle implémente une seule méthode : send : aMessage to : aReciever from : aSender qui permet comme l’indique son nom d’envoyer un message (une chaîne de caractère). **Nous précisons que notre serveur affiche tout simplement le contenu du message dans une fenêtre (Transcript).**[2].

La figure 3.1 montre les acteurs que nous venons de présenter et les interactions entre eux.

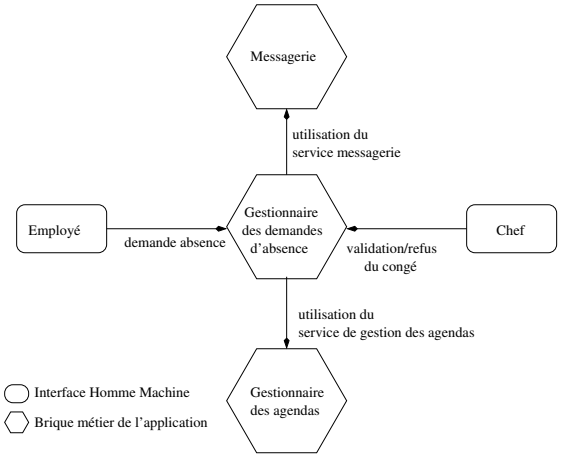

FIG. 3.1 – Schéma modélisant l’application “Employee’s Vacations Manager”

[2] c’est un objet Smalltalk qui affiche les messages quand nous appelons la méthode show : 'Le contenu du message'

3.2.1 Exemples de scénarios possibles

Notre but n'est pas le développement d'une application complète. Nous avons besoin d'un coeur fonctionnel qui intègre un minimum de fonctionnalités qui permettent d'appliquer la programmation par aspects aux services web. "Employee's Vacations Manager" ne vérifie pas si le nombre de jours de congé demandés est inférieur au nombre des jours disponibles. Aussi, elle ne gère pas les demandes en cours de validation. Elle ne gère pas le cas ou la période d'absence coïncide avec d'autres événements dans l'agenda du demandeur. D'où plusieurs scénario sont possibles.
Nous nous intéressons à deux seulement :

- Demande de congé refusé (figure 3.2) : le chef refuse le congé demandé par l'employé.
- Demande de congé accepté (figure 3.3) : l'employé demande un congé, et son chef la valide.

.

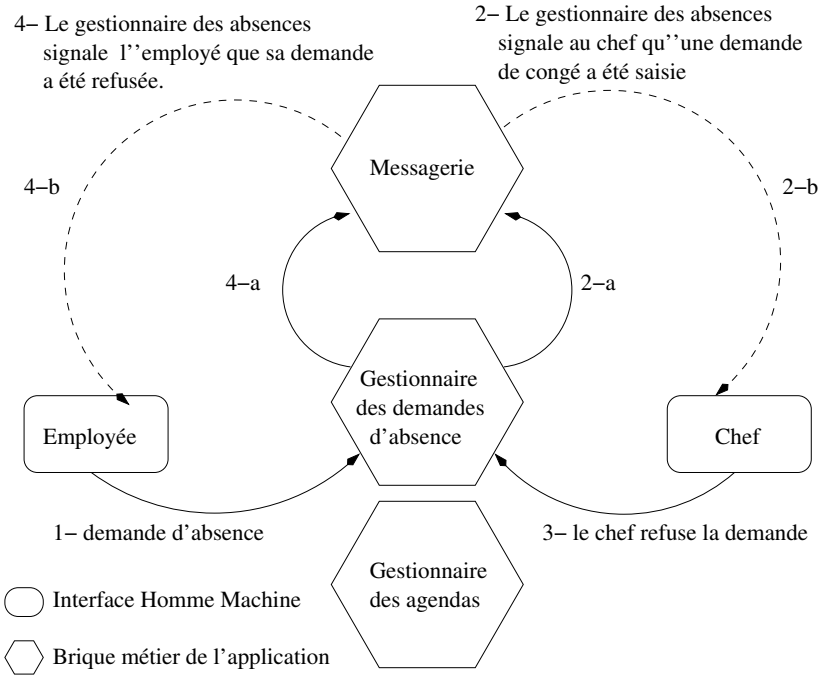

FIG. 3.2 – Exemple d'un scénario négatif

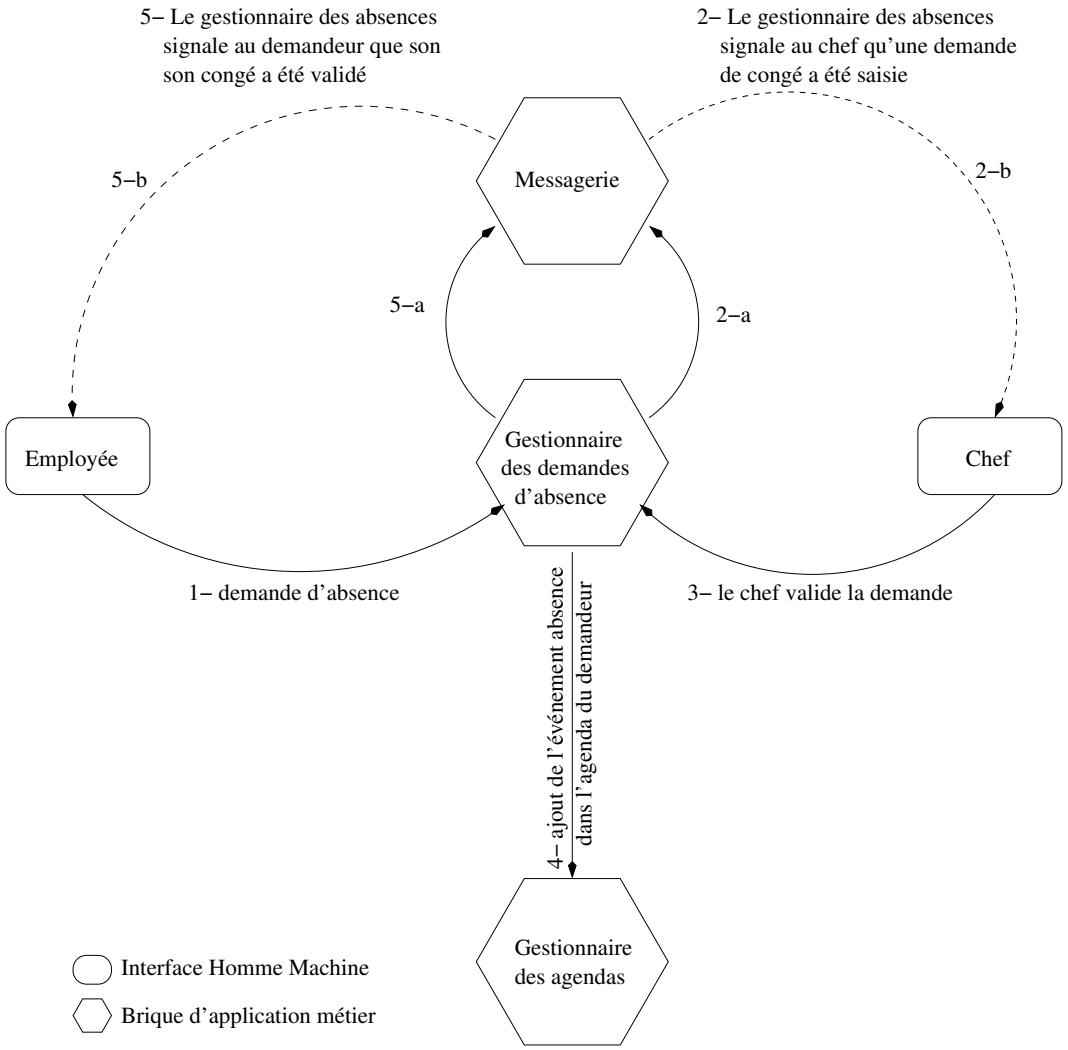

FIG. 3.3 – Exemple d'un scénario positif

3.3 "Employee's Vacations Manager" version service web

"Employee's Vacations Manager" est une application mono-poste. Les services sont invoqués par appel de méthodes (messages en Smalltalk). Maintenant nous souhaitons avoir une version service web de notre application. Dans cette nouvelle version, les briques qui composent l'application seront vues comme des services web (voir figure 3.4). Nous identifions les service web suivants :

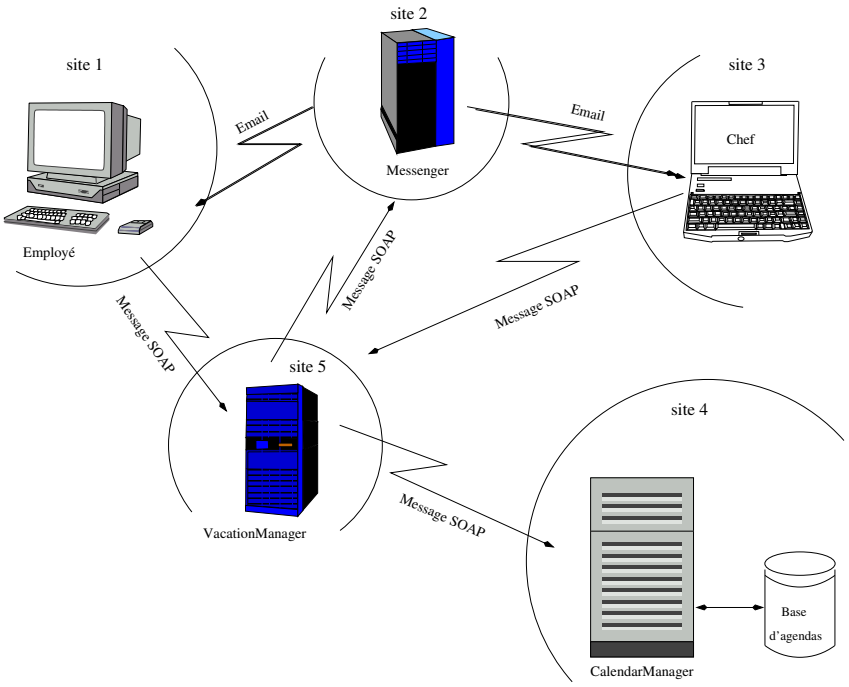

FIG. 3.4 – Employee's Vacations Manager dans sa version web service

- Le service de gestion des demandes de vacance VacationManager . Les applications clientes de ce service sont les IHM[3] des demandeurs de congé et des chefs.
- Le service de messagerie Messenger . Il est sollicité par le service web VacationManager .
- Le service de gestion des agendas CalendarManager . Il est sollicité par le service web VacationManager .

Dans ce nouveau contexte, nous avons besoin d'envoyer un message SOAP à l'objet serveur pour invoquer une opérations (ex. l'opération requestVacation : aVacation sentBy : personName du service web "VacationManager").

3.4 Développer des services web en Smalltalk avec SoapCore

Pour développer des services web en Smalltalk, nous pouvons utiliser le paquetage *SoapCore* [4] qui permet de faire du RPC en utilisant SOAP.

"SoapCore" n'utilise pas WSDL pour décrire les services web. Il décrit chacune des opérations du service web à part. Pour chaque opération nous donnons : la référence de l'objet qui va faire le traitement, le nom de la méthode (qui correpond à l'opération), la signature de la méthode, les argument et finalement l'URI du service. Les opérations sont stockées dans un dictionnaire (une sorte d'annuaire).

[3] Interface Homme Machine
[4] http ://www.mars.dti.ne.jp/ umejava/smalltalk/soapOpera/soapCore.html

L'exemple suivant (figure 3.5) montre le code Smalltalk de la méthode registerRequestVacation qui permet de décrire l'opération requestVacation : aVacation sentBy :personName du service "VacationManager". Le "handler" est l'entité installé côté serveur qui écoute le port de communication SOAP et gère le dictionnaire des opérations. Elle reçoit les messages SOAP du client, les traite et cherche dans le dictionnaire l'opération demandée. Si le service n'est pas disponible (n'est pas enregistré, ou n'existe pas) un message d'erreur est envoyé au client.

```
1- r gist rR qu stVacation
2-
3-   | impl s rv |
4-   impl := VacationManag r  n w.
5-   s rv := SoapS rvic  impl m ntor: impl s l ctor: #r qu stVacation:s ntBy:
6-   s rv signatur : (SoapS rvic Signatur  nam : 'r qu stVacation:s ntBy:'
7-                    paramNam s: #(anEv nt aString)).
8-   SoapS rvic Handl r d fault add: s rv.
```

FIG. 3.5 – Exemple d'enregistrement d'une opération d'un service web en "SoapCore"

Les lignes 4 et 7 correspondent à la description du service. La dernière ligne correspond à l'ajout de la description dans le dictionnaire.

Du côté client "SoapCore" permet la génération du message SOAP (figure 3.6). Tout d'abord nous créons un objet "SoapCall " (correspond à un objet d'appel SOAP) en précisant l'adresse IP du serveur et le port (ligne 4). Puis nous précisons l'URI, le nom de la méthode et les paramètres. La dernière ligne, génère le message SOAP à partir de l'objet "SoapCall" et l'envoi au serveur (ligne 10).

```
1 - callR qu stVacation: aVacation s ntBy: p rsonNam
2 -
3 -  | call |
4 -  call := (SoapCallEntry tcpHost: s lf host ddr ss port: s lf port) n wCall.
5 -  call transport: s lf transport.
6 -  call targ tObj ctURI: s lf targ tObj ctURI.
7 -  call m thodNam : 'r qu stVacation:s ntBy'.
8 -  call addParam t r: (SoapVariabl  nam :#anEv nt valu : aVacation).
9 -  call addParam t r: (SoapVariabl  nam :#aStringt valu : p rsonNam ).
10-  ^call invok  ndR turn
```

FIG. 3.6 – Exemple de génération d'un message SOAP en "SoapCore"

3.5 Aspect communication service web

Une opération n'est accessible que si, nous avons enregistré sa description dans le dictionnaire géré par le "handler". Pour invoquer une opération déjà enregistrée, nous devons créer le message SOAP correspondant et l'envoyer au serveur. la création et l'envoie du message SOAP est le code non fonctionnel qui correspond à notre aspect *communication service web*.

Revenons à notre application "Employee's Vacations Manager". Dans sa version mono-poste, les objets communiquent par envoi de messages (appel de méthodes). Pour basculer de cette version à une autre version service web, nous avons besoin de gérer la communication entre le client et le serveur. Cette communication consiste à générer le message SOAP correspondant à l'opération demandée, et à reçevoir ce message et le traiter du côté serveur. La communication service web est une propriété transversale que nous pouvons définir dans un "aspect communication service web". La figure 3.7 illustre cet aspect communication service web.

Cet aspect comprend :

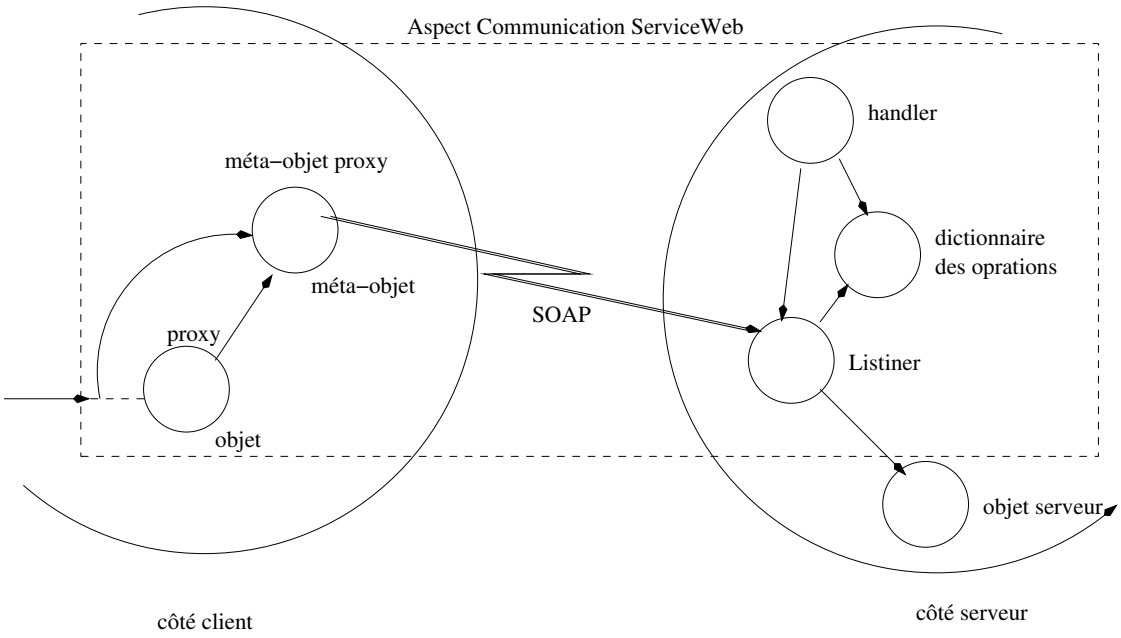

FIG. 3.7 – Aspect communication service web

Un proxy côté client : c'est un objet qui cache la distribution au client. Il reçoit les messages destinés à l'objet serveur et les renvoie à son tour à l'objet serveur. Dans le cas des service web le "proxy" doit être capable de :
- recevoir toutes les requêtes envoyées à l'objet serveur.
- générer le message SOAP correspondant à l'opération demandée.
- envoyer le message à l'objet serveur.

Un handler côté serveur : Il doit pouvoir enregistrer les opérations des services web et activer et arrêter le "listiner". Le "Listiner" est un module qui écoute le port de communication SOAP, reçois les message SOAP et envoie le message SmallTalk correspondante au message SOAP sur l'objet serveur.

Pour faire de la programmation par aspect, nous allons utilisé *MetaclassTalk* [BL02, BL03, Bou03] qui est une plate-forme réflexive. C'est une extension de Smalltalk qui permet d'expérimenter différents paradigmes de programmation. "MetaclassTalk" permet de créer des méta-objets et de les liés à des objets de base. Les protocoles des méta-objets peuvent contrôler les accès aux champs en lecture et en écriture, la réception et l'envoie de message, etc. . ..

3.5.1 Comment est codé le "proxy" côté client ?

Le proxy côté client comporte deux classes :

ProxyObject : C'est une classe vide. Les instances de cette classe servent comme objet de base.

WebServiceClientProxyMetaObject : Cette classe hérite de "LinkMetaObject", les instances de cette classe sont des méta-objets. Nous redéfinissons au niveau de cette classe la méthode receive : selector from : sender to : receiver arguments : args superSend : superFlag originClass : originCl . Cette méthode est appellée sur le méta-objet à chaque fois que l'objet de base qu'il contrôle reçoit un message quelconque. Elle génère le message SOAP correspondant au message reçu par l'objet de base et l'envoie au serveur.

Nous devons ensuite lié Un objet de la classe "ProxyObject" au méta-objet instance de la classe "WebServiceClientProxyMetaObject". Le code qui permet d'établir ce lien est le suivant.

```
|proxyObject proxyMetaObject|
proxyObject:= ProxyObject new.
proxyMetaObject:= WebServiceClientProxyMetaObject new.
proxyObject addMetaObject: proxyMetaObject.
```

Quand l'objet "proxyObject" reçoit un message, il sera intercepté par le méta-objet "proxyMetaObject". C'est ainsi que nous générons le message SOAP pour invoquer une des opérations du service web sans modifier le code du client.
La figure 3.8 illustre le mécanisme que nous venons de décrire.

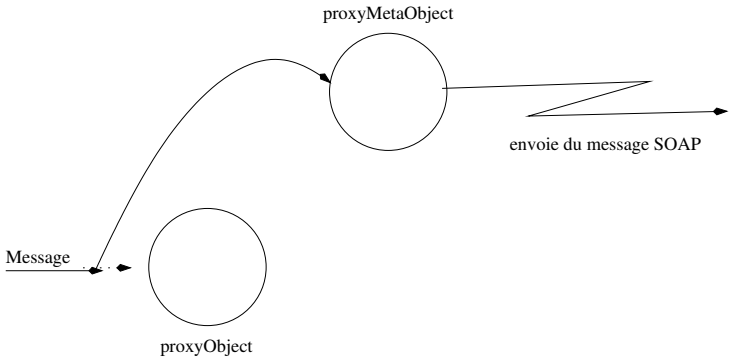

FIG. 3.8 – Le"proxy" côté client

3.5.2 Comment est codé le handler côté serveur ?

Le handler côté serveur est composé d'une unique classe WebServiceServerSideHandler . Cette classe implémente les services suivant :

- startServer , active le "listiner" qui va écouter le port de communication SOAP.
- stopServer , stope le "listiner".
- register : aServiceSelector implementor : implName arguments : argList , permet d'enregistrer les services web.

La figure 3.9 montre l'architecture du "handler" côté serveur

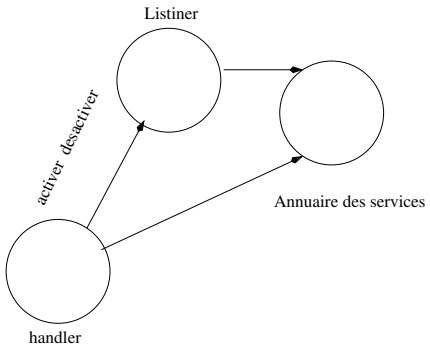

FIG. 3.9 – Le "handler" côté serveur

3.5.3 déploiement du client et du serveur

Comme nous venons de le dire, l'aspect "communication web service" est composé de deux modules. Un "proxy" par service web doit être installé du côté client et un "handler" du côté serveur. Pour invoquer des services web il faut déployer le "handler" du côté serveur, activer le "listiner" et enregistrer les services. Du côté client, il faut paramétrer le "proxy".

Exemple de déploiement

Ce qui suit, montre les étapes (par ordre), pour déployer notre application "Employee's Vacations Manager" :

1. Nous commençons par installer le code métier de chacune des trois briques de l'application. Le code de la classe VacationManager sera mis sur une machine, celui des classes CalendarManager et Calendar sur une autre et le code de la classe Messenger sur une troisième machine. Le classe métier Event sera installées sur toutes les machines, car nous échangons des objets "Event" (le congé est un événement).
2. Nous installons par la suite les "handler" sur toutes les machines qui hébergent des service web. Dans notre cas le "handler' est installé sur les trois machines qui hébergent VacationManager , CalendarManager et Messenger .
3. Nous activons les "listiner" de tous les "handler" en envoyant le message startServer à l'objet handler.
4. Nous installons des "proxy" sur les machines des employées ainsi que deux proxy sur la machine qui héberge le gestionnaire des congé.
5. Nous devons paramétrer chacun des proxy en lui indiquant l'adresse IP de la machine qui héberge le service, le numéro de port, le protocole de transport utilisé et finalement l'URI du service.

Chapitre 4

Conclusion et perspectives

Les services web est une technologie prometteuse qui est de plus en plus adoptée par les entreprises. Elle répond à leurs besoins de communication et de coopération.

Dans le cadre de nos recherches, nous avons appliqué le paradigme de la programmation par aspect sur les services web. Le résultat direct est un aspect générique "communication service web" décrit dans une bibliothèque réutilisable. Cet aspect permet la transformation de n'importe quel application mono-poste en service web (sans toucher le code de l'application).

Le déploiement de l'aspect "communication service web" est manuel pour l'instant. Une des perspectives sera d'automatiser son déploiement. Aussi, nous souhaitons identifier les autres aspects liés aux services web et les décrire dans des bibliothèques séparées et éventuellement résoudre les conflits qui peuvent apparaître entre eux.

Bibliographie

[BD01] Xavier Briffault and Stéphane Ducasse. *Squeak programmation.* Eyrolles, 2001.

[BL01] N. Bouraqadi and T. Ledoux. Le point sur la programmation par aspects (aspect-oriented programming - in french). *Technique et Science Informatique*, 20(4) :505–528, 2001.

[BL02] Noury Bouraqadi and Thomas Ledoux. Aspect oriented programming using reflection. *Technical report*, October 2002.

[BL03] Noury Bouraqadi and Thomas Ledoux. Aspect oriented software development. *Addison Wesley*, 2003. (to apear).

[Bou99] Noury Bouraqadi. Un cadre réflexif pour la programmation par aspects. *conference LMO*, 1999.

[Bou03] Noury Bouraqadi. What is metaclasstalk. *csl website*, 2003. http ://csl.ensm-douai.fr/MetaclassTalk.

[Cha02] Jean-Marie Chauvet. *Services Web avec SOAP, WSDL, UDDI, ebXML... Solutions d'entreprise.* Eyrolles, 2002.

[Cur02] Francisco Curbera. Unraveling the web services web : An introduction to soap, wsdl, and uddi. *Computer Society website*, March/April 2002. http ://www.computer.org/-internet/v6n2/w2spot.htm.

[Gir01] Didier Girard. Xml pour l'entreprise. *Application Servers*, juin 2001. http ://-www.application-servers.com/pagePublications.jsp.

[Got02] K. Gottschalk. Introduction to web services architecture. *IBM System Journal, Volume 41, Number 2, 2002 New Developments in Web Services and E-commerce*, 2002. http :/-/www.research.ibm.com/journal/sj/412/gottschalk.html.

[IKM+97] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future : the story of squeak, a usable smalltalk written in itsel. *Proc. OOPSLA*, 1997.

[ISO70] ISO. International organization for standardization website. *ISO website*, 1970. http :/-/www.iso.ch.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[Kre01] Heather Kreger. Web services conceptuale architecture. *IBM Website*, May 2001. http ://dwdemos.alphaworks.ibm.com/wstk/common/wstkdoc/doc/-WebServicesArchitecture.pdf.

[Leb02] Gérard Leblanc. *C Sharp et .Net deuxième édition d'après la version finale de visual studio .net.* Eyrolles, 2002.

[Man01] Anne Thomas Manes. enabling open, interoperable, and smart web services. *W3C workshop on Web services*, 11-12 April 2001. http ://www.w3.org/2001/03/WSWS-popa/paper29.

[NB03] Rabih Nassrallah and Noury Bouraqadi. Les services web, un condensé. *Technical report*, May 2003. http ://csl.ensm-douai.fr/research#publications.

[OAS02] OASIS. Uddi : Universal description, discovery and integration. *OASIS website*, 2002. http ://www.uddi.org.

[Smi84] B. Smith. Reflection and semantics in lisp. *POPL conference*, 1984.

[Som02] Frank Sommers. A birds-eye view of web services. *Java World*, 25 January 2002. http ://www.javaworld.com/javaworld/jw-01-2002/jw-0125-webservices.html.

[W3C86] W3C. Overview of sgml resources. *W3C website*, 1986. http ://www.w3.org/MarkUp/-SGML/.

[W3C99] W3C. Namespaces in xml. *W3C website*, January 1999. http ://www.w3.org/TR/-1999/REC-xml-names-19990114/.

[W3C00a] W3C. Dtd ; document type definition. *W3C website*, October 2000. http ://-www.w3.org/TR/2000/CR-SVG-20001102/svgdtd.

[W3C00b] W3C. Extensible markup language (xml) 1.0 (second edition). *W3C website*, October 2000. http ://www.w3.org/TR/REC-xml.

[W3C00c] W3C. Simple object access protocol (soap) 1.1 (w3c note). *W3C website*, May 2000. http ://www.w3.org/TR/SOAP/.

[W3C01a] W3C. Web services description language (wsdl) 1.1 (w3c note). *W3C website*, March 2001. http ://www.w3.org/TR/wsdl.

[W3C01b] W3C. Xml schema part 0 : Primer. *W3C website*, May 2001. http ://www.w3.org/-TR/2001/REC-xmlschema-0-20010502/.

[W3C03] W3C. Web services glossary. *W3C website*, May 2003. http ://www.w3.org/TR/ws-gloss/.