Praxeme Institute

Composant PxM-45 « Modus : La méthodologie Praxeme »

Les traitements "batch" dans l'architecture de services

Objectif

La méthodologie Praxeme situe les traitements "batch" dans l'architecture de services et non au-dehors.

Ce document fixe le procédé de conception et de spécification logique de ces traitements et des services logiques qui en découlent.

Contenu

- Les notions et orientations pour la conception des traitements par lots
- Les actions du concepteur logique
- Des illustrations

Rédacteur Dominique VAUQUIER

Version 1.1, le 13 septembre 2010

Praxeme Institute ✉ 21, chemin des Sapins – 93160 NOISY-LE-GRAND - France ▪ dominique.vauquier@praxeme.org ☎ 33 (0) 6 77 62 31 75

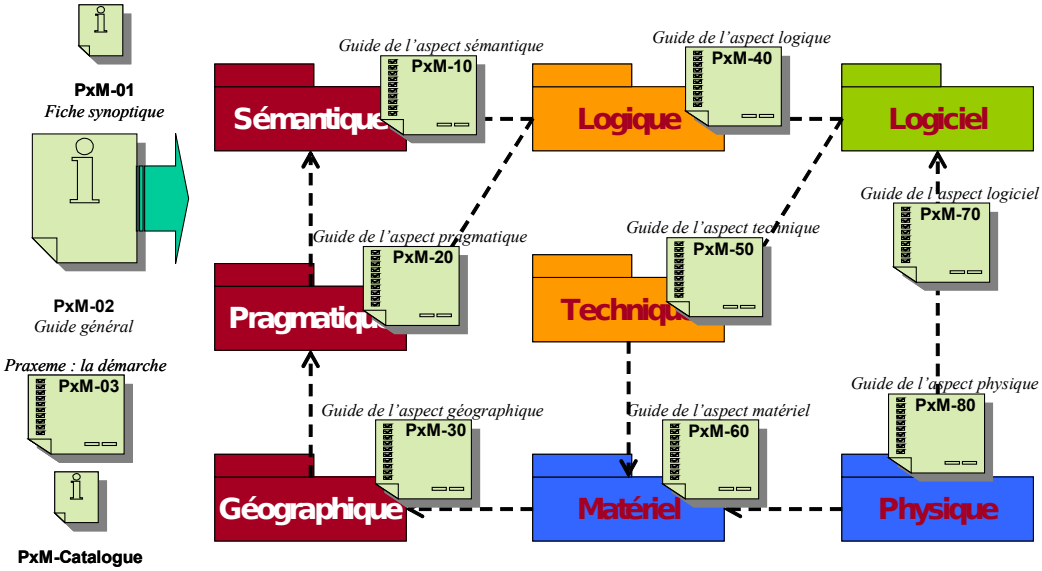
Éléments de configuration

Situation du composant

Positionnement dans la documentation

La méthodologie Praxeme est structurée selon les aspects de la Topologie du Système Entreprise. Le Guide général explique cette approche.

Le présent document, portant sur la modélisation logique, est associé au Guide de l'aspect logique, référencé « PxM-40 ».



Propriétaire

Le référentiel Praxeme a été élaboré dans le cadre de l'initiative pour une méthode publique. Les contributeurs se réunissent au sein du « Collège des contributeurs » qui oriente les travaux en fonction des préoccupations des entreprises et organismes. L'association *Praxeme Institute* fait évoluer le fonds commun.

Toute suggestion ou souhait d'évolution sont les bienvenus (à adresser à l'auteur).

Historique et planification

Indice	Date	Rédacteur	Contenu
0.0	26/10/06	DVAU	Première rédaction
0.1 **0.2** **0.3**	17/11/06 23/11/06 24/11/06	DVAU MGAR	Suite remarques du 14/11/06. Nouveau sommaire. Séance avec Pierre BONNET et Michel GARANX le 20/11/06. Complément d'informations suite entrevues avec Pascal HAUTEFEUILLE le 23/11/06 et Pierre BONNET le 24/11/06
1.0	19/06/06 (prévision)	DVAU	Après revue du document par CAT (David LAPEPINA, Hervé ROGER)
18/08/2010 1.1	18/08/10	Fabien VILLARD	Reprise pour Praxeme
18/08/2010 1.1	Version actuelle du document		

Sommaire

LES TRAITEMENTS "BATCH" DANS L'ARCHITECTURE DE SERVICES 1
Situation du composant 2
Historique et planification 2
Introduction 6
Incorporer les traitements batch à l'architecture de services 6
Les notions 7
Principes généraux : le batch dans l'architecture de services 8
La démarche 10
Positionner le traitement dans l'architecture de services 12
Les choix du concepteur 12
La possibilité de sortir de l'architecture de services 15
Les services asynchrones 16
Les services associés 18
La localisation des services dans l'architecture logique 20
Concevoir le traitement 22
La logique de communication 22
La conception par les automates 24
L'algorithme des chaînes batch 27
Paralléliser les traitements 29
Gérer la transaction 31
Gérer les incidents 34
Permettre les interruptions 35
Prendre en compte les possibilités techniques 37
Quelques conceptions typiques 38
Représenter le traitement 38
La description des services asynchrones 39
Optimiser le traitement 40
Les actions d'optimisation 40
Évaluer le comportement 40
Mettre en place le fonctionnement « au fil de l'eau » 41
Fusionner des étapes 42
Découper le traitement en étapes 42
Isoler les écritures des fichiers 43
Factoriser les traitements 43
Dénormaliser le modèle des services 44
Compléments 44
Sécuriser et suivre les traitements 45
Les dispositifs pour assurer la maîtrise de la dynamique 45
Concevoir le traitement des rejets 45
Les habilitations 46
La journalisation 47
La traçabilité 47
Le modèle des informations de sécurité et de suivi 47
Prendre les décisions sur l'aspect logiciel 48
Le déploiement et la conception du logiciel 48
La représentation de la chaîne batch 49
Annexes 50
La négociation Logique/Technique 50
Récapitulatif des possibilités d'action de l'Exploitation 51
Pseudo-code 52

Les activités et les points de vue....53
Les activités d'analyse et de conception associées au batch....54
Au fait, pourquoi différer ?....56
L'éventail des possibilités....57
Rappel sur la spécification des traitements batch....58
La Vue de l'utilisation....58
Les fonctionnalités à incorporer aux cas d'utilisation....58
Description des états....59
Représentation....60
Les possibilités de manœuvre....60
Typologie des batchs....60
Les automates à états : le méta-modèle....61
Index....62

Table des figures

Figure PxM-45_1. Exemple d'une chaîne batch....8
Figure PxM-45_2. Les modalités d'appel sur le diagramme de séquence....18
Figure PxM-45_3. Diagramme de séquence pour montrer la chaîne des appels....19
Figure PxM-45_4. Représentation informelle d'une chaîne batch....28
Figure PxM-45_5. Diagramme d'états décrivant la chaîne batch....29

Introduction

Incorporer les traitements *batch* à l'architecture de services

Objectif et périmètre

Ce document porte sur les traitements *batch* ou par lots, plus exactement sur leur conception logique. Le terme « logique » est pris dans le sens précis qu'en donne la Topologie du Système Entreprise, dans la méthodologie Praxeme.

Le procédé de conception logique des traitements *batch* forme le cœur de ce document. Néanmoins, afin de traiter complètement la question du *batch*, il a été nécessaire de déborder :

- en amont, sur la spécification fonctionnelle, pour préciser ce qui en est attendu et ce qui doit en être exclu ;
- en aval, sur la conception du logiciel pour faire la part des choses entre ce qui lui revient et ce qui est du ressort de la conception logique.

Au fil de l'exposé, nous ferons place, également, au point de vue de l'Exploitation[1], acteur essentiel pour cette partie du système couvert par les traitements *batch*.

Destinataires du document

Concepteurs logiques, avant tout (le corps du document décrit leur travail)

Mais aussi :

- Rédacteurs des spécifications fonctionnelles (cf. annexe)
- Personnel de la fonction Exploitation (Production)

Contenu du document

- Introduction : les notions
- Déroulé de la démarche : principales actions.
- Illustrations.
- Récapitulatifs : pseudo-code, négociation logique/technique, Exploitation

Synthèse des orientations

1er message

Les traitements *batch* ne sont pas hors architecture de services.

Si les traitements *batch* devaient tomber en dehors de l'architecture de services, cette échappée entraînerait des risques de sous-utilisation des services et de violation d'intégrité. C'est la situation classique qui serait reconduite. Évidemment, sur le chemin de cette orientation théorique, nous trouvons la question des performances.

2ème message

Nous devons distinguer la demande d'un traitement *batch* et sa réalisation.

La demande, elle-même interactive[2], peut s'exprimer facilement par l'intermédiaire d'un service « normal ».

Pour la réalisation du traitement lui-même, nous complétons le vocabulaire de l'aspect logique en introduisant les notions de service asynchrone et de service associé.

[1] Le terme Exploitation a été préféré à celui de Production, ce dernier étant utilisé dans un autre sens par la méthodologie.

[2] Beaucoup de traitements *batch* sont planifiés à l'avance : il faut distinguer le moment de la demande et celui du déclenchement.

Introduction (suite)

Les notions

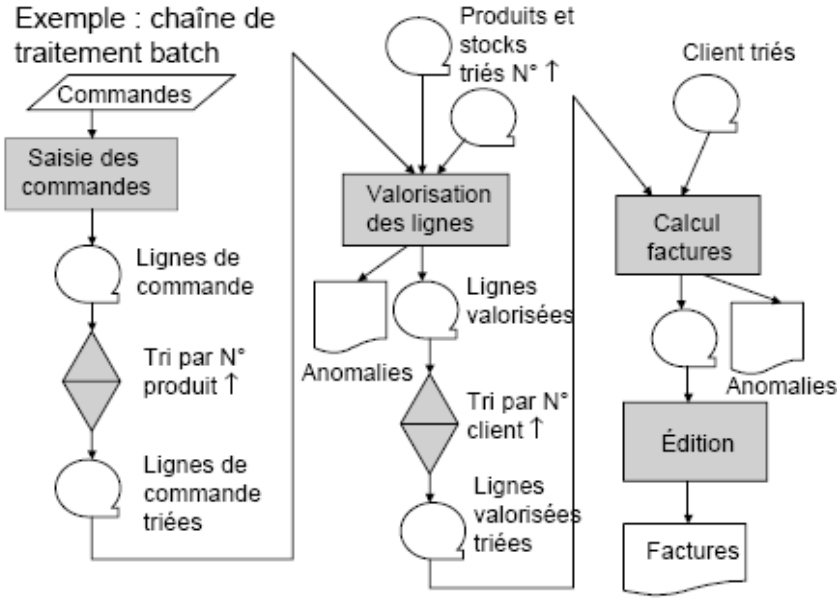
Définition

Le terme anglais « *batch*[3] » se traduit par « traitement par lots ». Il évoque, avec les notions connexes de « mode différé » et de « traitement asynchrone », des programmes « lourds », longs et se déroulant sans intervention humaine, du moins – en général – sans implication des acteurs du métier. L'acteur humain n'y est toutefois pas étranger puisque le personnel de la Production (ou Exploitation) conserve des possibilités de manœuvre sur les chaînes de traitement.

Histoire

Les premiers développements informatiques, sur *mainframe*, faisaient un usage systématique et majoritaire de ce type de programmation. Les programmes étaient organisés en chaînes d'exécution, chacun effectuant une tâche très spécialisée. Ces chaînes étaient jalonnées de stockages intermédiaires, souvent sous la forme de fichiers plats, comme dans l'exemple ci-dessous[4].

Figure PxM-45_1. Exemple d'une chaîne batch



[3] Littéralement : tas, groupe, lot…

[4] NB : Une chaîne *batch* peut se représenter en UML sous la forme d'un diagramme d'activité.

Introduction (suite)

Principes généraux : le *batch* dans l'architecture de services

Batch et SOA

Notre pétition de principe est le refus de l'isolationnisme : le *batch* n'est pas hors architecture de services, pas plus qu'il n'est hors spécification !

La spécification des traitements différés s'intègre dans la Vue de l'utilisation. Elle est examinée, ci-après, sous le titre « Point de vue externe ».

La conception de ces traitements peut conduire à identifier des services propres au *batch*. Elle fait appel à une catégorie de services ignorée jusqu'à présent : les services asynchrones. Elle doit aussi considérer la notion de programme. Les programmes se placent sur la strate « Présentation », à l'instar des IHM, et ils font appel aux services externes, i.e. les services de la strate « Organisation » (voir chapitre suivant).

La conception logique des traitements différés

À partir de la spécification fonctionnelle d'un traitement *batch*, la conception interne doit déterminer :

- les services utilisables,
- les services à développer,
- le contenu du programme *batch* qu'il faudra éventuellement développer.

Nous ne traitons ici que la partie de la conception interne qui porte sur l'aspect *logique*. Cette liste s'allonge avec les considérations de niveau logiciel et physique.

Les principes guidant la conception logique

Premier principe : Les traitements différés exploitent les mêmes services que les traitements interactifs, dans la mesure du possible.

Deuxième principe : orientation « au fil de l'eau » : l'architecture logique prévoit des services qui consolident l'information au fur et à mesure de sa modification ; de cette façon, le système dispose d'une information qui le rend mieux à même de répondre rapidement. Ce principe conduit à réduire la part des traitements différés et, donc, à augmenter la réactivité d'exécution du système.

Troisième principe : Des traitements longs, avec de longues périodes sans intervention humaine, sont inscrits dans l'architecture sous la forme de services asynchrones, déclenchés par une communication asynchrone.

« Au fil de l'eau »

Le deuxième principe vaut surtout pour les informations consolidées : totaux, dénombrements (éventuellement avec catégorisation), moyennes, extrêmes, écarts-types. Il conduit à :

- enrichir les services élémentaires[5] pour qu'ils mettent à jour ou déclenchent la mise à jour des informations consolidées ;
- étendre la structure de données des machines ensemblistes ou ajouter des services ensemblistes, ceux-là même qui fournissent l'information consolidée (et qui rendent caducs les traitements lourds) ;
- doter les machines ensemblistes des structures de données permanentes pour conserver les résultats agrégés.

Pour le traitement au fil de l'eau, il est nécessaire de prendre certaines précautions. Elles sont présentées dans le chapitre « Optimiser le traitement ».

[5] Services élémentaires : services logiques des machines logiques élémentaires.

Les traitements lourds

Ces principes et orientations réduisent la part des traitements différés dans le système, mais ne l'abolissent pas totalement. La nécessité subsiste de traitements que l'on ne peut pas réduire par le moyen d'une consolidation au fil de l'eau.

Dans ces cas, le concepteur dispose de deux options :

1. définir des services asynchrones ;
2. concevoir des traitements par lots, classiques, hors architecture de services.

Introduction (suite)

La démarche

Les tâches de la conception logique

L'aspect logique réclame l'intervention de plusieurs disciplines :

- l'urbanisation des systèmes d'information, qui fixe les orientations pour les évolutions des systèmes informatiques, en réponse aux orientations stratégiques de l'entreprise et aux exigences des métiers ;
- l'architecture logique, qui reprend la « grande carte » établie par l'urbaniste et la prolonge en soupesant les conséquences de chaque décision de structuration ;
- la conception logique, qui descend jusqu'au détail de la spécification formelle des services logiques.

Dans ce document, centré sur les traitements batch, seules les deux dernières disciplines sont abordées. La démarche est la suivante :

Action	Acteur	Commentaire
Spécifier les traitements *batch*	Analyste métier, responsabilité de la maîtrise d'ouvrage	Hors du périmètre de ce document, mais évoqué en annexe (pour rappel sur la spécification fonctionnelle liée aux traitements *batch*) ; voir en annexe p. 58.
Positionner le traitement dans l'architecture de services	Concepteur logique, sous la responsabilité de l'architecte logique	Le concepteur logique établit une proposition de localisation des nouveaux services qu'il a déduit de l'analyse du besoin. L'architecte logique valide cette proposition et se montre particulièrement vigilant si elle conduit à modifier l'architecture (nouvelles dépendances…). Voir pp. 12 et sq..
Concevoir le traitement	Concepteur logique	Conception et documentation précise des constituants logiques ajoutés à l'architecture de services. Éventuellement, demande de services ou de modifications en dehors de la responsabilité du concepteur. Voir pp. 22 et sq.
Optimiser le traitement	Concepteur logique	Une partie de l'optimisation repose sur la connaissance du comportement réel du système. Le concepteur améliore l'algorithme. Voir pp. 40 et sq.
Sécuriser et suivre les traitements	Concepteur logique	Le concepteur complète la modélisation en prenant en compte les incidents et les exigences de sécurité (habilitations, journalisation…). Voir pp. 45 et sq.
Prendre les décisions sur l'aspect logiciel	Concepteur du logiciel et architecte technique	Hors conception logique, les questions évoquées pour cette activité permettent de compléter la chaîne de production pour les traitements batch. Il est important de faire la part des choses entre la conception logique et la conception technique. Voir pp. 48 et sq.

La spécification fonctionnelle se limite ou devrait se limiter à la description du résultat attendu. Il arrive qu'elle déborde un peu et qu'elle véhicule des présupposés sur la solution. Elle peut, même, se montrer largement organique.

Au-delà des spécifications fonctionnelles, il reste à :

1. dessiner l'algorithme du traitement (la chaîne *batch* ou la description du service) et préciser les conditions de déclenchement et d'arrêt[6] ;
2. choisir les services logiques qui interviennent dans le traitement ;
3. définir et concevoir les services à ajouter dans l'architecture logique (mêmes exigences que pour le reste de la conception logique) ;
4. anticiper le comportement (vérifier que les performances – telles que l'on peut les pressentir au niveau logique – sont compatibles avec la demande et les possibilités du système)[7].

[6] L'algorithme évoque plutôt le diagramme d'activité, alors que l'ordonnancement fin conduit à penser en termes d'automate à états. Ces deux représentations sont réversibles, la seconde étant un meilleur outil pour échapper à la linéarité de l'approche fonctionnaliste et mieux penser l'adaptation des comportements.

[7] La conception technique réexaminera ce thème, plus tard dans la chaîne de production.

Positionner le traitement dans l'architecture de services

Les choix du concepteur

Les types de traitements *batch*

Après analyse de la demande (spécification fonctionnelle), le concepteur sélectionne, dans la typologie ci-dessous, le type de *batch* qui convient :

- *batch* ensembliste : traitement de masse, reposant sur une ou plusieurs requêtes qui balayent des tables ;
- *batch* unitaire : traitement différé mais pas nécessairement lourd (par exemple, édition d'un courrier) ;
- réduction du *batch* à un traitement au fil de l'eau ;
- solution sortie de l'architecture de services.

Ces catégories de solutions sont exclusives. Le *batch* ensembliste exclut le fil de l'eau.

L'annexe « L'éventail des possibilités » propose une grille de distribution des possibilités. Ces possibilités sont examinées ci-après.

Chaque type implique un type de conception particulier. L'analyse ci-dessous permet de mieux analyser le comportement du traitement et d'en déduire les contraintes.

Les caractéristiques des traitements *batch*

Pour analyser complètement le traitement demandé et orienter la conception vers la bonne solution, il convient de considérer les caractéristiques :

- la portée,
- la durée,
- le lancement.

La portée

Le premier critère à examiner pour s'orienter parmi les solutions possibles est la portée du traitement des informations. S'agit-il d'un ensemble de données ou d'un seul objet « métier » ? Ce critère distingue le traitement ensembliste (de masse) du traitement unitaire. Le traitement ensembliste est le véritable traitement par lots (*batch*). Sa lourdeur impose certaines contraintes, de façon à assurer les performances de la solution.

La durée

Un traitement ensembliste est, le plus souvent, long – mais pas nécessairement. Certains traitements unitaires peuvent être longs. La durée présumée du traitement fixe la frontière entre le traitement interactif et le traitement différé. La charte ergonomique sort cette frontière du domaine du subjectif. Par exemple, si une recherche ou une édition impliquée dans une situation de travail (un cas d'utilisation) demande plus de trois secondes, elle met en péril l'interactivité[8].

Le lancement

En concevant la solution, il est nécessaire de s'interroger sur le mode de lancement du traitement : le lancement peut être immédiat, différé (plus tard) ou ordonnancé (à heure fixe). Le différé peut être fixe (délai à partir de la demande) ou opportuniste (dès que possible, c'est-à-dire quand l'état des ressources le permet). Le différé opportuniste entraîne la nécessité de préciser les priorités des traitements. On convient qu'entre les trois modes de lancement, les priorités vont de l'immédiat à l'ordonnancé puis au différé fixe, en terminant par le différé opportuniste.

Dans le cas de lancement immédiat, on parle de traitements synchrones, qui ne diffèrent pas de l'architecture de services définie dans PxM-40. Dans le cas du lancement différé, il s'agit de l'asynchrone. L'ordonnancement se

[8] La généralisation de l'Internet et des interfaces intranet a considérablement augmenté l'endurance des utilisateurs. Aussi, la limite des trois secondes peut-elle être révisée !

situe entre les deux : asynchrone du point de vue de l'acteur métier, il est synchrone du point de vue de l'agent d'exploitation qui exerce une surveillance sur le déroulement du traitement et qui en attend des réponses immédiates.

Pour le lancement des traitements différés et ordonnancés, l'architecte technique propose deux approches :

- l'approche classique, « fin de journée », qui réserve la nuit aux traitements lourds ;
- l'approche audacieuse, plus risquée mais plus tournée vers le confort d'utilisation, qui consiste à exécuter certains traitements batch pendant la journée de travail, en concurrence avec les traitements interactifs du métier.

Les caractéristiques décrites ici aideront à élaborer les solutions en réponse aux spécifications fonctionnelles. Par exemple, on cherchera à résorber le traitement unitaire dans une solution dite « au fil de l'eau ».

L'objet du traitement

Aux caractéristiques précédentes, s'ajoute la nature même du traitement batch, nature donnée par le type de son objectif.

Selon ce critère, on distingue :

- batch d'extraction (extraction de données de la base, façonnage d'un flux de données, puis dépôt ou envoi vers l'extérieur ; par exemple pour édition ou communication vers d'autres systèmes) ;
- batch de calcul (mise à jour des données internes, création de nouvelles informations) ;
- batch d'injection (la réciproque de l'extraction : réception de données de l'extérieur, analyse et incorporation dans les bases de données du système).

Ces trois natures se croisent avec les caractéristiques décrites ci-dessus.

Quelques règles pour décider

Le Dossier d'architecture technique fixent quelques règles à respecter pour assurer un bon fonctionnement du système. Sans avoir besoin de connaître le détail des justifications, le concepteur logique applique ces règles pour élaborer la solution. Le concepteur technique du logiciel pourra vérifier leur bonne application et apporter des précisions.

Traitement ensembliste

Le fonctionnement ensembliste impose l'approche « fin de journée ».

Quand une contrainte d'intégrité fonctionnelle porte sur plusieurs dossiers[9], un rejet entraîne l'annulation de tout le traitement[10].

Concurrence avec l'interactif

On ne peut autoriser des traitements *batch* pendant l'exécution des applications interactives qu'en l'absence de dépendance entre les informations traitées.

Il est intéressant de lancer des traitements batch pendant le travail courant des acteurs métier, à double titre :

- tout d'abord, cette option permet, parfois, d'apporter une réponse plus rapide aux acteurs métier ou aux acteurs externes ;
- ensuite, cette facilité réduit la charge des traitements différés en fin de journée.

Le concepteur doit, néanmoins, prendre quelques précautions. Les traitements *batch* ne doivent pas perturber le travail des acteurs. C'est le sens de la règle limitant le recours à la concurrence *batch*/TP[11]. En cas de fonctionnement concurrent sur les mêmes données, d'une part la priorité est accordée à l'interactif, d'autre part la conception du traitement batch doit prévoir les rejets, de la façon la plus souple (voir p. 45).

[9] Par exemple pour alimenter le *data warehouse* sur les règlements à une date donnée.

[10] Les règles de réalisation de *jobs* ensemblistes sont décrites dans un document joint au DAT.

[11] TP : temps partagé (autre nom des traitements interactifs, faisant référence à l'architecture des anciens systèmes).

Un assemblage de services

Chaque fois que cela sera possible, les traitements *batch* vont se concevoir comme des assemblages d'appel de services. Ces services seront, majoritairement, synchrones, c'est-à-dire les mêmes que ceux qui interviennent dans les traitements interactifs. Il arrivera, aussi, que la conception des traitements *batch* amène à définir des services asynchrones. Ce type de services est défini dans la section suivante (p. 16).

Le batch modélisé comme service

Les traitements *batch* eux-mêmes peuvent apparaître sous la forme de services.

Ce précepte offre deux avantages :

La description de ces traitements est, ainsi, complètement intégrée au modèle logique.

Le traitement jouit des bénéfices de l'orientation services : structuration, partage. Les possibilités de partage (donc, de réutilisation) dépendent de la localisation du service dans l'architecture (point abordé plus loin).

L'appel fait le synchronisme

Un même service pourra être appelé, à la fois, en mode synchrone et en asynchrone.

Pour qu'un service soit susceptible d'être appelé en mode asynchrone, il devra respecter certaines règles, exposées plus loin. Mais rien n'interdira qu'un tel service soit appelé, également, en mode synchrone. C'est le cas d'un service qui peut être déclenché à partir d'un traitement interactif, pour un acteur métier, et à partir d'un ordonnanceur, pour les besoins de l'exploitation. Dans le premier cas, l'appel se fait en asynchrone pour ne pas bloquer le déroulement du dialogue. Dans le second, l'agent d'exploitation attend immédiatement un retour du traitement déclenché.

Positionner le traitement dans l'architecture de services (suite)

La possibilité de sortir de l'architecture de services

Les solutions de base

Les solutions naturelles vers lesquelles doivent tendre les efforts des concepteurs logiques sont celles à base de services. Les plus simples reposent sur des assemblages de services synchrones ordinaires. La deuxième ligne de front est occupée par les services asynchrones, examinés dans la première section.

Par exception, il faut toutefois envisager des solutions qui excluent le traitement de l'architecture de services, c'est-à-dire qui en font un programme ordinaire, non un service, et même un programme qui n'appelle pas des services. Cette sortie de l'architecture de services ne peut intervenir que sur dérogation expresse de l'architecte logique, après motivation par le concepteur logique et échange avec l'architecte technique.

La dérogation

Outre les cas où les traitements n'entrent pas dans le plan d'urbanisation du SI, on peut se résigner à la solution « hors architecture » dans les cas extrêmes :

- complexité des requêtes à effectuer ;
- solution plus naturelle obtenue en exploitant par exemple le SGBDR (jointure, tri, fonctions statistiques, procédures stockées…) ;
- performances (quand la solution à base de services présente des performances intolérables par rapport à des solutions classiques).

Les conséquences

Cette possibilité de sortir de l'architecture de services devrait rester exceptionnelle et, toujours, faire l'objet d'une discussion avec l'architecte logique et l'architecte technique. Éventuellement, le recours à un expert du SGBD peut apporter un nouvel éclairage[12].

Les inconvénients de ce type de solution sont :

- le coût supplémentaire (le programme ne pouvant réutiliser les services qui encapsulent la connaissance du métier, son développement nécessite une bonne compréhension fonctionnelle et un effort de test complet[13]) ;
- le danger de perte de maîtrise lors d'évolutions (du fait de la dépendance directe par rapport à la base de données ; et aussi un risque sur l'intégrité du système parce qu'il faut reprogrammer les règles de gestion).

[12] Possibilité de recourir aux procédures cataloguées (ou procédures stockées) uniquement dans le cas des traitements *batch*.

[13] Car on ne peut pas bénéficier de l'homologation des services, testés unitairement.

Positionner le traitement dans l'architecture de services (suite)

Les services asynchrones

Définition

Un service logique est dit asynchrone s'il peut être appelé de façon asynchrone.

L'appel asynchrone est choisi pour les services dont le temps de traitement excède une durée acceptable dans une interaction entre le système et un acteur humain (on peut fixer un seuil). Il peut être choisi, aussi, indépendamment de la question du temps de réponse, pour des raisons fonctionnelles.

Un service asynchrone peut être, également, appelé de façon synchrone. Il serait plus approprié de l'appeler « service multimodal ».

Objectif

Les services asynchrones ne retournent pas de résultats vers l'appelant. Ils se limitent à traiter des données, sans interférence directe avec l'appelant, ou constituer un flux. Le flux peut, éventuellement, être déposé dans un support de stockage.

Dans tous les cas, comme tout service, le service asynchrone exerce dans le périmètre de responsabilité du constituant sollicité (c'est-à-dire de l'atelier « métier » ou de la machine logique « Organisation », selon l'endroit sur lequel a été positionné le service).

Ce flux peut ne couvrir qu'une partie de ce que le demandeur attend (par exemple, une application de comptabilité). En pareil cas, il y a un travail de collation à effectuer : ce travail est du ressort d'une machine logique « Organisation ».

Caractéristiques

Par rapport à un service « purement » synchrone, c'est-à-dire qui ne peut être appelé qu'en mode synchrone, un service asynchrone doit présenter plusieurs caractéristiques.

Signature

Pour qu'un service puisse être appelé en mode asynchrone, sa signature doit respecter certaines conditions.

- *En paramètres d'entrée* : d'éventuels paramètres de valeur fonctionnelle (par exemple, pour segmenter le traitement), le jeton établi par le service lanceur (le jeton identifie le traitement ; sa présence dans la signature est discutée plus loin).
- *En paramètres de sortie* : résultats du traitement, le plus souvent un seul paramètre fournissant le flux résultant. Il peut ne pas y avoir de paramètres de sortie (cas d'une modification interne du système ou d'un dépôt dans un stockage intermédiaire).
- *En retour* : jamais de résultat en retour du service asynchrone. C'est sa marque distinctive.
- *Nom du service* : conventions habituelles (inutile de réduire la lisibilité en s'imposant des règles de nommage ; l'absence de retour suffit à détecter la possibilité d'un appel asynchrone).

Le concepteur rendra le service asynchrone et appliquera ces règles, chaque fois que l'exécution du service dépassera le délai acceptable pour un fonctionnement synchrone.

Fonctionnement

Le lancement du traitement asynchrone et la récupération du résultat se font grâce à d'autres services, ceux-ci synchrones. Ce sont les « services associés » (section suivante).

Contenu

Le service asynchrone est la solution pour les traitements lourds de calcul ou de recherche en base, pour les traitements différés et même pour des traitements brefs, unitaires, mais qui se déroulent

sans interférence avec les appelants potentiels. Un service asynchrone peut lui-même faire appel à d'autres services, synchrones ou non.

Licence

On envisage la possibilité de développer des services spéciaux, pour améliorer les performances du *batch*, avec requêtes complexes et exploitation poussée du SGBD. De façon exceptionnelle (en cas de problèmes réels de performances), ces services peuvent violer les frontières établies par l'architecture des données[14]. C'est, déjà, se mettre en marge de l'architecture de services. Ces dérogations doivent être suivies scrupuleusement. L'architecte logique est seul habilité à les entériner.

Représentation

Puisque l'asynchronisme est lié au mode d'appel plus qu'à la nature du service, c'est sur le dessin de l'appel que l'on pourra le lire.

Pour représenter l'appel asynchrone, UML propose les outils suivants :

- Sur le diagramme de séquence : l'appel asynchrone est indiqué par une demie flèche qui débute la boîte d'exécution, laquelle se termine sans ligne de retour. Cependant le recours au diagramme de séquence n'est pas conseillé dans Praxeme car il ne décrit qu'un scénario d'exécution et non l'algorithme complet.

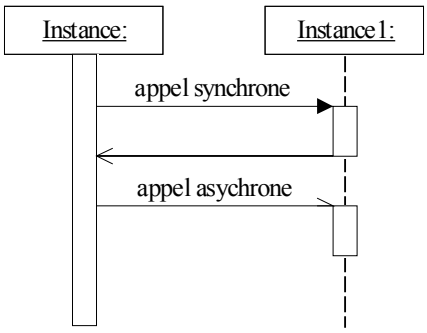


Figure PxM-45_2. Les modalités d'appel sur le diagramme de séquence

- Sur le diagramme d'activité : l'action (*actionState*), qui représente le service asynchrone, est « concurrent » . Pour que cette qualité apparaisse sur le diagramme, il est nécessaire de créer un stéréotype à placer sur la flèche de l'enchaînement (« appel asynchrone »). Le diagramme d'activité est la technique algorithmique retenue dans Praxeme pour décrire le fonctionnement des services (cf. PxM-40).

[14] On peut imaginer de charger les données par des requêtes directes, de façonner les flux à partir de ces données pour les passer aux services et bénéficier de l'encapsulation. On aura simplement court-circuité les services de recherche.

Positionner le traitement dans l'architecture de services (suite)

Les services associés

Les services associés

À chaque service asynchrone, sont associés :

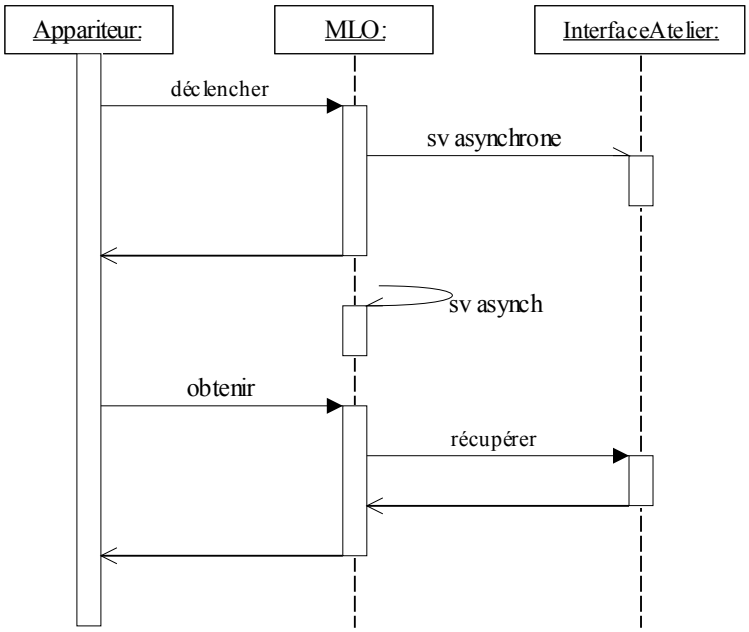
- un service de demande ou de déclenchement : le lanceur ;
- un service de récupération : le fournisseur ;
- un service d'information qui donne l'état d'avancement : l'informateur ;
- un service d'interruption qui arrête le traitement en cours en lui envoyant un événement : l'interrupteur.

Ces catégories sont détaillées dans « La logique de communication », p. 22.

Le fonctionnement

Le service de récupération reçoit, en paramètre, le jeton qui a été fourni lors du déclenchement du service asynchrone. Il renvoie le flux résultant, selon les conventions habituelles[15] (logique détaillée page suivante).

Figure PxM-45_3. Diagramme de séquence pour montrer la chaîne des appels



<< voir édition p. 21 pour corriger le diagramme >>

Le jeton est un ensemble d'informations qui permet d'identifier le traitement. Il garantit l'unité et la cohérence entre les appels des services associés et du service de traitement.

L'appariteur (machine logique de la strate « Interaction ») est soit une IHM, dans le cas où l'acteur métier a la possibilité de déclencher lui-même le traitement long, soit l'ordonnanceur, piloté par l'Exploitation.

[15] Sur le diagramme de séquence UML, l'appel asynchrone se représente par une flèche rognée. Il n'y a pas de retour à l'appelant, à la fin de la boîte d'exécution.

Le scénario se déroule de la façon suivante :

1. L'appariteur actionne le service de demande (le lanceur). Un tel service est placé, le plus souvent, sur une MLO. Il est, forcément, synchrone. Ce service crée le jeton et le transmet à l'appariteur qui le conserve.
2. Le service lanceur déclenche le service asynchrone, en lui passant le jeton. Au cours du traitement, le jeton peut servir à identifier les résultats et les stockages intermédiaires. Le service peut être localisé sur la même machine ou sur une autre, éventuellement dans la strate « Métier » si le traitement ne comporte aucune référence à l'organisation.
3. Au cours de son exécution, le service asynchrone enregistre les changements d'états correspondant aux étapes franchies.
4. À tout moment, l'appariteur peut requérir l'état du traitement. Le service informateur est logé sur la même machine que le lanceur. C'est, forcément, un service synchrone. Il accède à la variable d'état pour la retourner au demandeur. Éventuellement, la machine peut porter plusieurs variables d'états pour le même traitement, quand celui-ci se compose de plusieurs flots d'exécution.
5. Enfin, le service fournisseur donne à l'appariteur la possibilité d'obtenir le résultat du traitement. Ce service est synchrone. Les résultats stockés dans des espaces intermédiaires sont identifiés grâce au jeton.

Positionner le traitement dans l'architecture de services (suite)

La localisation des services dans l'architecture logique

Localisation des services liés au *batch*

Les services liés au *batch* peuvent se rencontrer absolument partout dans l'architecture logique. Le seul critère retenu pour les placer sur l'architecture *logique* est la signification : ils sont associés au constituant logique qui est le seul propriétaire de l'information demandée ou qui est habilité à assembler l'information par orchestration. Il n'en va pas forcément de même au niveau logiciel ou physique.

Strates

Les services asynchrones peuvent se rencontrer sur toutes les strates. Ils sont localisés en fonction de leur contenu. Par exemple, une synthèse client peut figurer sur une MLO si elle donne une vue pour un type d'acteur particulier (l'accueil, le gestionnaire…). La synthèse client illustre, d'ailleurs, le besoin d'orchestration. Un bilan de l'activité se placera sur la strate « Organisation » et aura toutes les chances de nécessiter un service asynchrone.

Mais on peut rencontrer des services asynchrones également sur la strate « Métier ». Les machines de cette strate offrent les services asynchrones qui se définissent strictement dans les termes sémantiques. Par exemple : la moyenne des coûts des sinistres par catégorie est prise en charge par la MLM mSinistresEns. Du point de vue logique, localiser les services sur la strate « Métier » s'impose comme la meilleure solution, tant que les traitements n'impliquent pas d'information ou de règle de nature organisationnelle.

Voir en annexe, l'exemple du décaissement.

Types de machines

Accordant la priorité au critère de signification, nous ne ressentons pas le besoin d'isoler les services asynchrones dans un type de constituant particulier. Ces services peuvent se rencontrer sur n'importe quel type de machine (élémentaire ou ensembliste), sur n'importe quelle strate. La forme de la signature suffit à percevoir le comportement asynchrone d'un service. Seul le critère de la pertinence fonctionnelle détermine la localisation des services.

Sans que cela soit totalement interdit, la méthode préconise de ne pas isoler les services asynchrones ou/et les services associés aux traitements batch dans des machines dédiées. En effet, le critère de structuration applicable à l'architecture logique doit rester le plus proche possible du métier. Les services asynchrones sont suffisamment repérés par leur signature pour qu'il ne soit pas nécessaire de les isoler. Le concepteur évitera, donc, de créer des machines logique *batch* et ne s'y résoudra que par exception.

Sur la strate « Présentation »

« Services de présentation » : distinguer les "vrais" services (ceux qui sont dans "Services généraux") et les fonctions générales, côté IHM (qui ne demandent pas un appel de service et qui sont embarquées).

Nous ne développons pas ce point ici.

Cas d'une MLO *batch*

Il est tout de même envisageable de créer des machines logiques dédiées aux traitements *batch* mais il ne faudrait pas en abuser. Mieux vaut donner la priorité au critère de contenu qu'à ce critère formel. Autrement dit, les services sont assemblés dans une même ML pour la raison qu'ils manipulent le même contexte informationnel. Il n'y a pas de raison de séparer les services selon leur nature synchrone ou asynchrone. En revanche, il y a un inconvénient : si ce contexte informationnel (la structure de données, le contenu sémantique) évolue, la maîtrise de l'évolution est compliquée par la dispersion des services concernés.

Il y a tout de même un cas de figure où le concepteur peut préférer la solution d'une « MLO batch » (en fait, une MLO composée essentiellement de services asynchrones correspondant aux étapes d'une chaîne *batch*). C'est le cas d'une chaîne batch lourde ou complexe que l'on ne souhaite pas exprimer sous la forme ramassée d'un service :

- soit du fait de son ampleur,
- soit parce qu'elle manipule un contexte informationnel qui excède le périmètre d'une machine existante.

En pareil cas, l'automate qui décrit cette chaîne pourrait être attaché à une MLO et les étapes de la chaîne deviennent des services asynchrones de cette MLO. Ils peuvent eux-mêmes être gouvernés par leur propre automate qui ordonnance l'appel des services « normaux ». La MLO affiche également les services associés qui sont synchrones.

Ce cas de figure se présente à partir d'un cas d'utilisation qui spécifie un besoin impliquant des traitements lourds.

Cette hypothèse théorique doit être expérimentée sur un exemple de conception logique. De toute façons, ce cas doit rester exceptionnel.

Le contexte

Le contexte n'est autre que la structure de données (il s'agit du contexte informationnel à l'exclusion du contexte véhiculé souterrainement par exemple par la VEP (Virtual Engine for Praxeme)).

Au début de la chaîne *batch*, la première action consiste à récupérer les paramètres et les variables d'environnement. À partir de là, la machine peut constituer le contexte et valoriser le jeton[16].

Le concepteur logique doit se demander quels sont les paramètres en entrée. Il faudra en déduire les écrans nécessaires. Si la conception logique n'intègre pas la strate « Présentation », c'est la conception du logiciel qui mène cette tâche.

[16] Voir plus en détail p. 47.

Concevoir le traitement

La logique de communication

La séquence type

Détail du fonctionnement.

NB : les services évoqués ci-dessous peuvent se trouver aussi bien sur la strate « Métier » que la strate « Organisation » (la figure précédente n'est qu'un exemple).

Le lanceur

Un service synchrone permet d'enregistrer la demande.

Sa signature : le service retourne le « jeton » qui sera conservé par le demandeur puis utilisé au moment de récupérer le résultat ou même pour connaître l'état d'avancement.

Son contenu : en plus de la création du jeton (selon un mécanisme à établir[17]), le service lanceur appelle le service qui réalise le traitement. Cet appel se fait sur un mode de communication asynchrone.

Négociation logique / technique : peut-on dans certains cas placer des pré-conditions de l'effecteur dans le lanceur ?

Oui, mais alors l'effecteur doit être privé. Si on est dans un atelier métier, seul le lanceur figurera dans l'interface.

NB : avant le lanceur, on peut avoir un demandeur. Cette sophistication s'impose quand le traitement différé ne doit être déclenché que sous contrainte temporelle. Pour l'exploitation, la contrainte temporelle se ramène à un événement (par exemple, l'opérateur doit agir à une heure donnée).

L'effecteur

Le service asynchrone réalise le traitement différé, qui peut être long.

Sa signature : il reçoit en paramètre le jeton, grâce auquel le composant réalisant le service pourra localiser le résultat physique.

L'informateur

Un service synchrone peut être appelé à tout instant pour renseigner sur l'état d'avancement.

Sa signature : elle fait apparaître le jeton, en paramètre d'entrée, et retourne la valeur de l'état.

Le fournisseur

Enfin, quand l'effecteur a terminé le traitement différé et que l'informateur indique la fin de la procédure, le fournisseur peut intervenir.

Sa signature : toujours le jeton en entrée ; en résultat : le flux fabriqué.

L'interrupteur

Forme

Conception des interruptions

[17] À discuter dans la **négociation logique/technique**.

Sv générique "rechercher"

requête en paramètre
classement (tri et groupement)
pagination : où en est-on ?

le sv générique renvoie une liste de flux élémentaires complexes, trié conformément à la requête demandée
voir s'il est possible de séparer :

- la requête = ce qu'il faut chercher
- le tri = comment on le restitue

c'est la strate Présentation (l'appariteur) qui gère les ruptures

Concevoir le traitement (suite)

La conception par les automates

L'intérêt des automates

La raison de l'automate à états est d'établir des points de stabilité au sein du traitement, c'est-à-dire des moments où le traitement peut s'arrêter sans mettre en péril l'intégrité ou la stabilité du système. C'est la grande question dans la conception des traitements longs et des traitements de masse.

L'apport de l'automate

Un automate à états, en soi, est un régulateur. Il gouverne le comportement de l'objet auquel il est attaché. Jusqu'à présent, nous l'avons utilisé pour gouverner le comportement des machines logiques . Pour la conception des traitements lourds, le concepteur a la possibilité d'attacher un automate à un traitement : le service logique asynchrone ou le « programme *batch* ».

Un état est un point de reprise possible.

À l'intérieur d'un état, on peut indiquer une activité, par exemple une étape d'un traitement long. La possibilité d'interruption est représentée sous la forme d'un événement (*Event*).

Les activités longues ou critiques ne peuvent pas figurer sur les transitions : elles sont forcément incluses à un état de type « étape ».

Le parallélisme possible entre des parties d'une chaîne *batch* ou même à l'intérieur d'un service s'exprime sous la forme de « régions » concurrentes dans un état complexe. Le thème du parallélisme est traité plus loin.

La localisation des automates à états

Le guide de l'aspect logique évoque les automates à états qui gouvernent le comportement des machines logiques. Soit ils reprennent l'automate attaché à la classe sémantique dont dérive la machine, soit ils proviennent des cas d'utilisation, la logique de la situation de travail pouvant elle aussi s'exprimer par un automate à états.

Pour les automates propres aux services conçus dans le cadre du batch, les choses sont un peu différentes. L'automate ici permet de coordonner des actions ou étapes du traitement. Il y a plusieurs possibilités pour inscrire ces automates, selon la portée du traitement décrit. Ces automates peuvent se placer :

1. sur les services ;
2. sur les machines ;
3. sur les ateliers.

Sur le service

La conception des traitements *batch* dans l'architecture de services conduit à placer des automates sur les services eux-mêmes (particulièrement, sur les services dont la réalisation est longue et qui sont définis comme asynchrones). UML permet, en effet, de définir un automate à états sur une opération. Dans notre contexte, un tel automate montrera l'ordonnancement du traitement en étapes. Les transitions pourront faire référence à d'autres opérations : elles correspondent aux étapes. De telles opérations internes ne font pas nécessairement sens, vues de l'extérieur. Quand leur exécution autonome n'est pas permise, elles sont privées, à l'usage exclusif des services ou opérations de la machine.

Sur la machine

Des automates peuvent, également, être ajoutés à une machine logique : ils serviront à coordonner le déclenchement ou le déroulement de plusieurs traitements *batch*… Quand la machine dispose déjà d'un automate, avant que le concepteur considère les traitements *batch*, les états et transitions liés aux traitements batch peuvent se glisser dans l'automate préexistant. Le concepteur choisit cette option dans les cas où il est nécessaire de coordonner les services qui servent le *batch* et ceux qui alimentent le fonctionnement interactif. Dans

les cas fréquents où une telle coordination n'est pas nécessaire, c'est-à-dire quand la logique des traitements *batch* est indépendante du fonctionnement interactif, la solution consiste à créer un autre automate.

Sur l'atelier Le modèle logique doit, également, décrire la coordination des traitements *batch*, entre eux. L'exemple le plus éminent est celui de la « nuit *batch* ». Il s'agit de décrire les conditions de préséance ou de séquençage des traitements, sur un ensemble plus large que la machine logique.

C'est toujours l'automate à états qui nous fournit l'instrument pour ce travail. Cette fois-ci, le concepteur l'accroche au niveau où s'établit la coordination : l'atelier, la fabrique, la strate voire le système. UML accorde cette possibilité d'attacher un automate à états sur un paquetage. Le niveau d'agrégation à retenir dépend de la portée de la coordination souhaitée. À chaque fois que cela est suffisant, le concepteur donnera sa préférence à l'atelier. De façon générale, la bonne solution consiste à retenir la portée minimale : cela facilite la compréhension et la maintenance.

S'il est nécessaire de coordonner, par exemple, des traitements inscrits sur une MLO et d'autres fournis par les ateliers « métier », la solution est la suivante :

- l'atelier métier indique l'état des traitements sous sa responsabilité (c'est dire qu'il possède un automate et que son interface offre les services capables de renseigner les consommateurs) ;
- l'automate de la MLO mentionne les états de l'atelier impliqué (dans ses conditions d'état).

La conservation des valeurs d'états

Puisque le mécanisme repose sur les automates, il devient nécessaire d'enregistrer, au cours des traitements, les valeurs d'état qui changent. Ceci contredit le principe selon lequel les mises à jour ne devraient intervenir qu'à la fin des chaînes *batch*. Il faut, au minimum, que la machine qui supporte le service asynchrone puisse conserver la valeur de l'état courant.

Quand une machine supporte plusieurs services dotés d'un automate, elle doit conserver autant de variables d'états car elle doit pouvoir les indiquer quand un service informateur est sollicité.

Rappel : pour un même automate, plusieurs variables d'états peuvent être nécessaires. C'est le cas quand l'automate contient des états concurrents.

Le cycle de vie du traitement *batch*

L'automate à états qui décrit le traitement spécifie les valeurs et les moments pour les comptes rendus d'exécution.

L'arrivée dans un état provoque l'enregistrement d'un compte rendu d'exécution (CRE) dans le journal.

Dans le cas des états contenant des activités, ils peuvent contenir eux-mêmes des écritures de CRE. Ce thème est détaillé p. 47 (« (suite) ») et p. 47 (« Le modèle des informations de sécurité et de suivi »).

C'est en construisant l'automate à états, représenté par un ou plusieurs diagrammes d'états, que le concepteur spécifie les comptes rendus d'exécution.

Règles de conception

Si le traitement présente plusieurs étapes, le concepteur doit obligatoirement modéliser la logique du déroulement sous la forme d'un automate.

Cet automate doit révéler les possibilités de parallélisme entre les étapes.

L'algorithme exprime le déroulement sous une autre forme (diagramme d'activité). L'algorithme et l'automate doivent être strictement cohérents.

L'automate attaché à un service asynchrone comporte deux types d'états :

- des jalons (états stables, après la fin d'une étape),
- des étapes (activité en cours).

Dans les états de type « étape », on inscrit le service à réaliser pendant l'étape (normalement un seul, sinon on perd un jalon).

Cette règle facilite la représentation des interruptions et des reprises, ainsi que la conception du journal.

Un service ou une opération qui n'a pas un comportement synchrone, c'est-à-dire dont l'exécution ne peut pas être considérée comme instantanée, est nécessairement exclu des transitions de l'automate et doit être absorbé par un état de type étape.

Il est possible de définir des variables attachées à un état. Elles serviraient à spécifier les inscriptions dans le journal.

Ce point doit faire l'objet d'une négociation entre l'architecte logique et l'architecte technique.

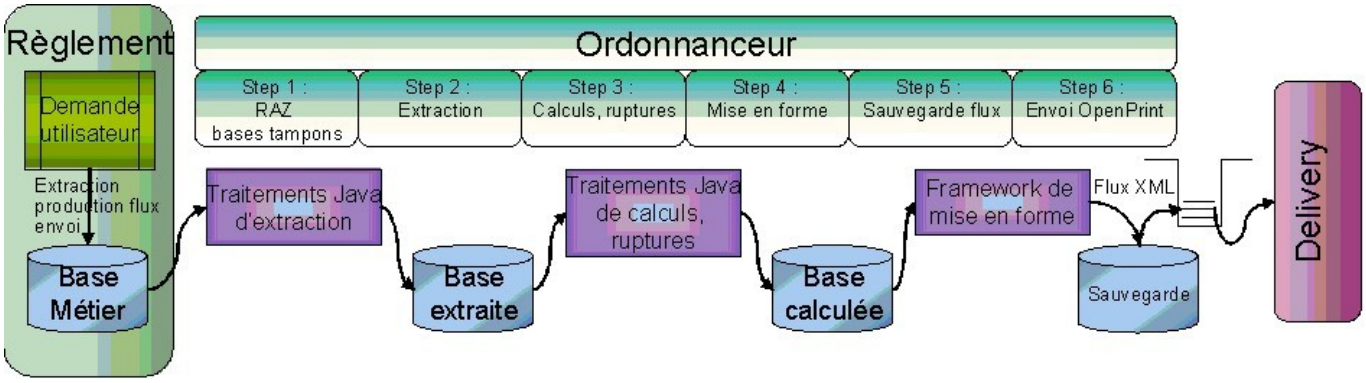
Concevoir le traitement (suite)

L'algorithme des chaînes batch

Un exemple

Le schéma ci-dessous décrit une chaîne de traitements définie pour les états dans le dossier d'architecture technique:

Figure PxM-45_4. Représentation informelle d'une chaîne batch



La chaîne batch est donc subdivisée en plusieurs étapes : préparation, extraction, calcul, etc.

L'interstice entre deux étapes définit un point de reprise. Une analyse plus poussée des conditions internes de chaque étape peut révéler d'autres points de reprise possibles.

Cet algorithme décrit une séquence parce qu'il est théorique et dispose des catégories génériques de traitements. Dans la réalité du travail de conception, le concepteur logique s'impose de n'exprimer que les conditions d'ordonnancement incontournables. De cette façon, il révèle les possibilités de parallélisme entre les étapes ou au sein des étapes. La modélisation logique doit expliciter ces possibilités. Elles sont transcrites sur l'algorithme en UML : automate et diagramme d'activité (en expulsant l'aspect logiciel).

Représentation recommandée

La figure de la page suivante restitue le même algorithme, sous la forme normalisée, c'est-à-dire à l'aide de la notation UML. Certes, cette représentation est moins esthétique, mais on peut toujours y ajouter des couleurs et même des symboles. L'important est de rester au sein du référentiel de modélisation et de pouvoir associer les éléments entre eux. Le diagramme nomme les opérations accessibles à l'endroit où il est défini.

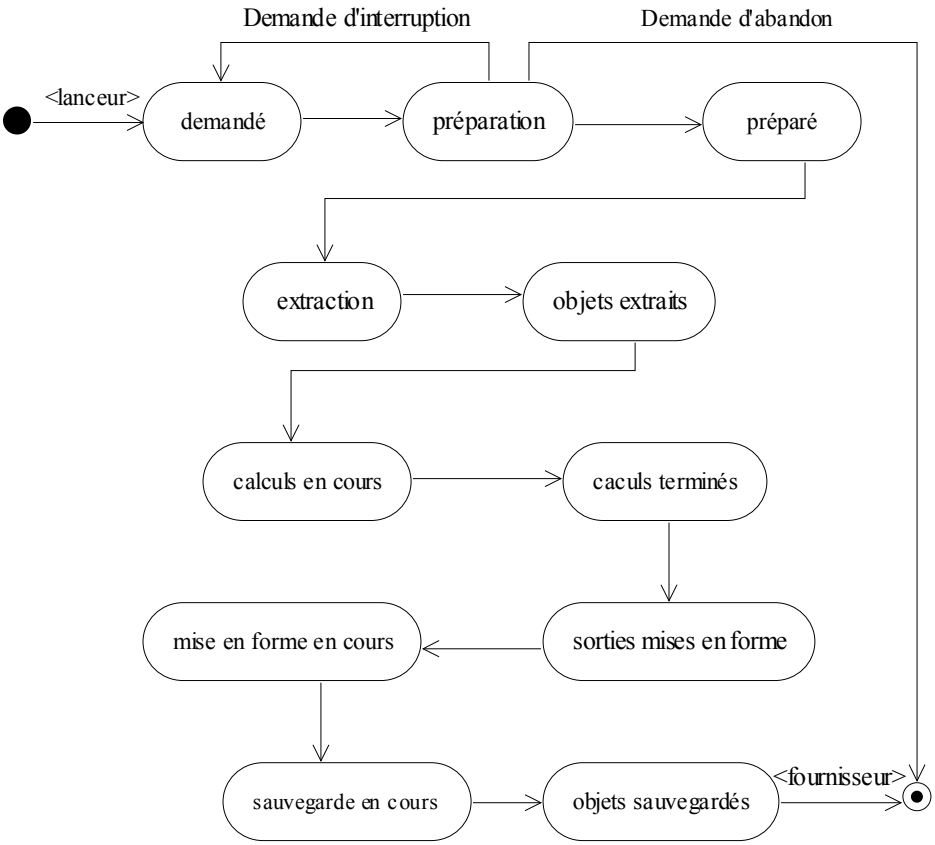
Commentaire Le service lanceur ouvre le bal. Dans l'exemple, on constate que le service fournisseur prélève et consomme les résultats : il ne sera appelé qu'une fois. On en déduit que les résultats intermédiaires s'effacent. D'autres situations sont envisageables. Si les résultats sont conservés, alors leur fourniture n'aboutit pas au point de sortie de l'automate. Si les résultats peuvent être demandés plusieurs fois, le service fournisseur est placé sur une transition réflexive : ceci indique que la fourniture ne change pas l'état.

Deux événements figurent sur cet automate :

- la demande d'interruption, qui peut survenir uniquement en cours de préparation dans cet exemple ;
- la demande d'abandon, qui force la sortie.

Les autres transactions ne portent pas de services. Ce n'est pas un état provisoire de la modélisation. Ceci s'explique par le fait que les activités, étant longues, ne peuvent pas figurer sur les transactions. Elles sont incorporées dans les états de type étape. L'outil utilisé ne permet pas de les représenter sur le diagramme mais ces activités – des services asynchrones – sont nommées dans les notes en pseudo-code attachées aux états.

Figure PxM-45_5. Diagramme d'états décrivant la chaîne batch



Concevoir le traitement (suite)

Paralléliser les traitements

L'objectif

L'Exploitation doit pouvoir exploiter les possibilités de parallélisme et les ajuster aux capacités disponibles. Elle exerce, donc, un « parallélisme variable » : elle choisit de déclencher des traitements en parallèle, si cela présente un intérêt de son point de vue, dans un contexte donné.

Le concepteur logique a pour tâche de décrire toutes les possibilités de parallélisme, pour autant que la logique fonctionnelle est respectée.

Les contraintes

La contrainte absolue à respecter dans la recherche du parallélisme est le « point de commit », c'est-à-dire le moment, dans le traitement, où la transaction est validée. Ce moment crée un point de stabilité dans le traitement.

Trois types de parallélismes envisageables :

- Le macro-parallélisme qui se joue entre des étapes d'un traitement ou d'une chaîne de traitements.
- Le parallélisme paramétrique quand le même service s'exécute plusieurs fois simultanément mais sur des données différentes.
- Le micro-parallélisme, interne à un service.

Le macro-parallélisme

Le premier est le plus intéressant dans notre contexte car il donne des marges de manœuvre à l'Exploitation. Il est examiné plus en détail dans le chapitre « Optimiser le traitement ».

Le parallélisme paramétrique

Un exemple : un service réalise le traitement des dossiers. Si on ajoute à sa signature un paramètre permettant de désigner une ou plusieurs directions régionales, on pourra paralléliser plusieurs flots d'exécution. Le même traitement s'exécutera plusieurs fois en même temps, mais chaque flot d'exécution portera sur une région ou un sous-ensemble de région, donc sur des données différentes.

Cette possibilité pourra être utilisée par l'Exploitation ou par d'autres traitements.

Le micro-parallélisme

Cette forme de parallélisme, interne à un service, peut paraître moins intéressante. Il faut en discuter tout de même. En tout cas, même s'il ne peut pas être exploité actuellement dans la programmation, le concepteur se doit d'exprimer ces possibilités, quand la logique du traitement s'y prête. Le service comporte des étapes. Si certaines de ces étapes peuvent se dérouler indépendamment les unes des autres, le concepteur l'indique. Ce parallélisme se lit sur le diagramme à états ou sur l'algorithme (diagramme d'activité).

La segmentation

Le parallélisme paramétrique permet de segmenter l'exécution d'un traitement de masse.

Le souci d'augmenter les possibilités de parallélisme peut entraîner des conséquences sur la signature des services.

Son impact

Par exemple : un service asynchrone « traiter_dossiers ». Pour rendre son exécution parallélisable et offrir cette possibilité à l'Exploitation, le concepteur ajoute un paramètre. Par exemple : la région (ou tout autre critère jugé pertinent du point de vue fonctionnel et efficace du point de vue de l'exécution). Dès lors, il devient possible de lancer en parallèle le traitement des dossiers de plusieurs régions.

Dans ce cas, le parallélisme n'est pas inclus au service. La signature a été changée pour segmenter l'exécution. Cette solution de la segmentation doit toujours prévoir le cas où l'exécution sera demandée sans parallélisme. On peut convenir que, si le paramètre est vide, le service traite les dossiers de toutes les régions.

La segmentation de l'exécution est présentée ici aux fins d'augmenter le parallélisme. Elle peut répondre à d'autres motivations. Par exemple, pour des traitements très lourds : la possibilité de n'en réaliser qu'une portion cohérente chaque nuit.

L'unité du traitement segmenté

La segmentation soulève une difficulté. Il peut être nécessaire de conserver l'unité du traitement segmenté, c'est-à-dire qu'il faut, parfois, pouvoir dire que les flots d'exécution déclenchés appartiennent au même traitement. Pour cela, plusieurs solutions sont possibles :

- Le demandeur du service passe un code qui est intégré au jeton qu'il reçoit en retour. S'il passe le même code pour chaque appel du service, il pourra le retrouver dans les jetons obtenus[18]. Le service fournisseur pourra reconstituer l'unité du traitement à l'aide de ces jetons.
- Le demandeur constitue simplement une liste des jetons reçus pour tous les appels qu'il considère comme faisant partie de la même unité d'exécution (par exemple les jetons obtenus chaque fois qu'il a déclenché « traiter_dossier » pour une région donnée). C'est à lui de conserver cette liste. Les autres services associés au traitement global, informateur et fournisseur, utiliseront cette liste.

[18] La construction des jetons est évoquée p. 47.

Concevoir le traitement (suite)

Gérer la transaction

Les attendus

Les traitements interactifs mobilisent des transactions. Leur portée est relativement brève puisqu'elle ne dépasse pas la durée d'un dialogue homme-machine ou d'un acte de gestion. La solution de l'architecture de services consiste alors à indiquer que tel service – le service de validation – est de nature transactionnelle. Avec les traitements par lots, il n'en va pas de même. La transaction assure la cohérence d'un ensemble de transformation dans le système. Un traitement de masse entraîne un grand nombre de transformations élémentaires. Dans certains cas, on considérera que rien ne s'est passé tant que la dernière étape n'a pas été franchie.

Le modèle logique doit préciser la portée de la transaction – l'empan – et clarifier les contraintes à respecter pour que l'ensemble des transformations maintienne l'intégrité du système.

Dans une chaîne batch, il faut savoir si chaque étape est susceptible d'une validation séparée ou si la transaction englobe toutes les étapes. La chaîne peut aussi couvrir plusieurs transactions, chacune assemblant quelques étapes.

Les solutions

Nous pouvons envisager deux types de solutions pour documenter les transactions, c'est-à-dire préciser l'empan de la transaction et, précisément, le point de validation (« point de commit »). La première solution reprend celle qui est utilisée pour les traitements interactifs, basée sur une annotation. La seconde repose sur un foncteur à inclure dans le pseudo-code.

La solution par annotation

Tout service peut être marqué par une annotation qui le désigne comme service transactionnel. Cette annotation est exploitée lors de la dérivation vers le logiciel. Le framework technique peut alors prendre en charge la gestion de la transaction. Ce système fonctionne pour les services synchrones impliqués dans le mode interactif. Seuls les machines logiques « Organisation » portent des services transactionnels.

Il peut se produire qu'un service transactionnel en appelle un autre de même nature. Dans ce cas, le comportement transactionnel du second est inhibé. La raison en est que l'architecture technique ne supporte pas les transactions imbriquées. Dans le modèle logique, cette restriction n'apparaît pas. Le concepteur logique se contente d'indiquer la nature transactionnelle des services. Tout ce qui peut se passer à l'intérieur et à partir du service transactionnel, y compris dans d'autres services appelés en cascade, est couvert par la transaction.

Cette solution peut s'appliquer aux services asynchrones, au moins pour les services qui s'identifient à la transaction. L'empan de la transaction est le service lui-même, dans sa totalité.

Dans le cas d'un traitement qui comporterait plusieurs étapes jouissant de l'autonomie transactionnelle, le traitement lui-même ne pourrait pas être traduit comme un service transactionnel. Les services correspondant aux étapes autonomes sont déclarés comme transactionnels.

La solution par annotation ne convient pas dans les cas où le traitement contient une transaction définie sur plusieurs étapes mais pas toutes. Une bonne solution en pareil cas, si la logique fonctionnelle l'autorise, est de définir un service transactionnel pour couvrir le sous-ensemble des étapes, chacune des étapes correspondant elle-même à un service. Quand l'exécution séparée de l'étape ne présente aucun sens, le service est privé. S'il n'est pas envisageable de définir un service pour le sous-ensemble des étapes, alors il faut se tourner vers la deuxième solution.

La solution par foncteur

La deuxième solution pour définir les transactions dans les cas les plus délicats appelle un enrichissement du pseudo-code.

Un foncteur apparaît dans le contenu du service ou du traitement : il marque le point où doit se faire le *commit*, c'est-à-dire la validation de la transaction. Ce foncteur peut se limiter à une expression, « enregistrer_transaction », sans autre précision. Le mécanisme technique auquel renvoie le foncteur absorbe à la fois la gestion de la transaction, le « pas de commit » (voir ci-après) et la journalisation.

Cette solution est plus souple puisqu'elle permet de décrire des traitements qui ne sont pas des services et qu'elle convient dans le cas de sous-ensembles transactionnels qui ne sont pas à l'échelle du service.

Le pas de validation

Le pas de validation (ou « pas de *commit* ») est, dans un traitement de masse, le nombre d'objets à traiter pour déclencher la validation d'une transaction. Le principe est de ne pas attendre le traitement de tout le volume d'objets pour reporter les transformations dans les bases de données. Cette notion est distincte de celle de l'empan de la transaction. L'empan définit le périmètre couvert par la transaction. Il doit être fixé à la conception. Cette notion sera discutée à travers l'illustration n°1. Le pas de validation est une valeur quantitative pour contrôler le volume traité par une transaction, à l'exécution.

Par exemple, on provoque la validation de la transaction après cent dossiers traités. L'unité transactionnelle est, dans cet exemple, le dossier. Le concepteur précise les limites de cette unité transactionnelle, l'empan : Qu'est-ce qu'un dossier ? Quelles sont les transformations qui sont liées logiquement et qui ne peuvent survenir qu'ensemble pour assurer la cohérence du système ? Ces points sont du ressort de la conception logique.

Ensuite vient la question du pas de validation. Cette question se pose à chaque exécution et n'est donc pas de la responsabilité du concepteur. Une transaction individuelle, au niveau du dossier, risquerait de nuire aux performances. À l'inverse, si l'on attend le traitement de tous les dossiers pour valider (transaction globale), alors on augmente les risques qu'un échec survienne, sans aucun résultat, après avoir mobilisé les ressources du système. En fixant un pas intermédiaire entre la transaction individuelle (pas = 1) et la transaction globale (pas = ∞), on améliore le comportement du système.

C'est l'Exploitation qui fixe le pas de validation, en fonction de sa connaissance du système physique et du contexte d'exécution.

Au niveau logique, le pas de *commit* est implicite. C'est dire que le foncteur couvre l'interprétation du pas de commit et la décision de valider quand le pas est atteint.

Au niveau logiciel, le pas de *commit* est paramétré : le dispositif technique de pilotage des traitements batch ou les programmes qui les lancent fixent la valeur du pas, lors de chaque exécution.

La localisation des transactions

Pour le comportement interactif du système, l'architecture de services a exclu les transactions sur la strate « Métier ». Cette décision résulte de la négociation logique / technique. Elle est liée à des préoccupations d'ordre technique.

En ce qui concerne les traitements de masse, cette restriction risque de se révéler handicapante. En effet, l'architecture logique positionne des services asynchrones et des traitements lourds au plus près des objets « métier », donc sur des machines de la strate « Métier ». La purge des objets sur des critères d'état (au sens de l'automate) fournit un exemple universel. On peut aussi évoquer des traitements statistiques. Si ces traitements font sens indépendamment de toute activité humaine et de tout choix d'organisation, alors autant les placer sur la strate « Métier ».

D'un point de vue logique, il est tout à fait envisageable de localiser un service asynchrone dans la strate « Métier »[19]. Ceci oblige à accepter des transactions définies sur la strate « Métier », également. En effet, le concepteur retrouve sur le service asynchrone toute la discussion menée ci-dessus : découpage en étapes, définition des unités transactionnelles sur des étapes simples ou sur des ensembles d'étapes… Il lui faut donc disposer, sur la strate « Métier », du vocabulaire nécessaire à la définition des transactions.

Si jamais cette solution ne peut pas être transportée au niveau logiciel, alors le mieux sera de laisser la modélisation logique se faire selon sa vocation et de donner des instructions à la conception du logiciel pour localiser les composants dérivés de ces services logiques.

À l'intérieur de la transaction

Un dernier point doit être examiné à propos de la gestion des transactions : les écritures dans les fichiers et, plus généralement, la mobilisation de ressources autres que la base de données. En effet, la gestion des transactions est étroitement associée aux SGBD mais d'autres transformations sont impliquées dans les traitements. L'unité

[19] C'est, d'ailleurs, le cas dans l'illustration n°1, plus loin.

transactionnelle peut absorber des créations de fichiers temporaires, des écritures dans différents supports, intermédiaires ou non, des émissions vers d'autres parties du système, etc.

Si la transaction échoue, il faut démonter ces transformations. Ceci est moins simple que l'annulation de la transaction du point de vue de la base de données (*rollback*). Le concepteur doit indiquer, pour toutes les ressources impliquées, quels traitements elles doivent subir en cas d'échec de la transaction. Pour un fichier, soit il sera supprimé, si le fichier est strictement associé au traitement ou à l'étape qui a échoué ; soit on ne supprimera que les lignes touchées par le traitement avorté.

Concevoir le traitement (suite)

Gérer les incidents

La solution générale

Pour la gestion des incidents et pour la communication vers le demandeur, les services et traitements liés au comportement *batch* du système adoptent le même dispositif logique que pour les services synchrones. Ce dispositif repose sur les signaux. La technique de représentation est la même[20].

Les particularités liées au *batch*

Une partie au moins des signaux émis par les traitements batch fait l'objet d'une codification à établir avec l'Exploitation.

Il devrait exister une liste de signaux établie par l'Exploitation. Ces signaux, de valeur générique, seront déclarés une fois pour toute dans le paquetage « Signalisation » de la fabrique « fUtilitaire ». Le concepteur logique y fait référence sur les algorithmes décrivant les services. La conception ne se limite pas à ces signaux génériques. Elle peut en ajouter comme sous-classes de ces signaux ou en créer de nouveaux, porteurs d'un contenu fonctionnel spécifique.

Dans le cas des traitements de masse, les événements sont enregistrés dans le journal. En tout cas, ceux de portée ensembliste.

[20] Cf. MTH-18.

Concevoir le traitement (suite)

Permettre les interruptions

L'interruptibilité des chaînes *batch*

L'ordonnanceur ou l'opérateur d'exploitation peuvent interrompre et reprendre une chaîne *batch*.

L'interruptibilité est une qualité de confort dans le fonctionnement du système. Elle intervient dans l'exploitation, au cours des traitements de masse. Elle mérite d'être assurée, aussi, pour tous les traitements un peu longs, que les acteurs « métier » ont à connaître ou à subir.

Les conditions d'interruptibilité ou de déclenchement se représentent sur l'automate à états qui décrit la chaîne.

L'interrupteur

Parmi les services associés au traitement batch, la machine logique propose un service dit « interrupteur ». En activant ce service, l'acteur du système indique son intention d'interrompre le traitement. C'est aussi un moyen de rattraper une demande déjà postée ou planifiée, dans le cas d'un traitement différé qui ne s'est pas encore déclenché.

Le concepteur élabore la signature du service interrupteur en fonction du contexte et du fonctionnement du service principal. Par exemple, dans certains cas, la demande d'interruption s'assimile à une suspension avec une intention de reprendre ultérieurement ; dans d'autres cas, elle exprime un souhait d'abandon définitif.

Dans le cas où le traitement comporte du parallélisme, il faut préciser si la demande d'interruption vaut pour l'ensemble du traitement *batch* ou pour une étape seulement ou un flot parallèle. Quand plusieurs options sont possibles et laissées à l'appréciation du demandeur, la signature du service interrupteur doit permettre de préciser le choix.

Les possibilités d'interruption

Un traitement ne s'interrompt – volontairement – que sur un état stable. Un état stable se caractérise comme une situation cohérente du système ou d'un ensemble d'objets considérés.

Ces points de stabilité se confondent avec les états tels qu'ils sont modélisés par le moyen des automates à états.

Si, à l'analyse, le concepteur perçoit des possibilités d'interruption qui ne peuvent pas se loger sur l'automate, c'est que celui-ci doit être amélioré. L'automate indique les possibilités d'interruption par des transitions qui portent le service interrupteur. La transition est réflexive si le traitement doit reprendre, plus tard, à partir du même état. L'interruption, particulièrement l'abandon, peuvent conduire à un autre état.

Sur le diagramme d'activité qui décrit le service interruptible, un signal « Demande d'interruption » se manifeste sous la forme d'un événement entrant. Il est consommé par la partie du traitement qui peut interrompre l'exécution.

De plus, le pseudo-code indique précisément l'endroit où la demande est prise en considération.

Quand une demande globale, sous la forme d'un seul événement, est supposée interrompre plusieurs traitements (flots parallèles) alors le traitement qui la traite – et consomme l'événement – doit émettre autant d'événements secondaires que nécessaire. Un événement ne peut être consommé qu'une seule fois. Une fois consommé, c'est-à-dire traité, il disparaît. Cette règle classique permet de simplifier la conception.

L'événement

L'événement d'interruption est modélisé sous la forme d'un signal. L'information qu'il véhicule comporte :

- l'acteur responsable de l'interruption ;
- si possible, sa motivation.

Cette information est enregistrée dans le journal. Rétrospectivement, elle éclairera sur le fonctionnement du système. Elle guidera, aussi, la reprise ultérieure du traitement.

Le signal est capté au sein du traitement. Si possible, l'algorithme (sous la forme d'un diagramme d'activité) montre le point où le signal est observé (*actionState* qui consomme l'événement). Sinon et dans tous les cas, la localisation de ce point se lit dans le pseudo-code.

L'arrêt demandé ne peut se produire que sur un point de stabilité du système, c'est-à-dire un « point de *commit* ». Ce point peut différer de l'endroit où le signal a été observé. En effet, une fois la demande détectée, le traitement peut poursuivre jusqu'à terminer un ensemble cohérent d'actions, délimité par l'unité transactionnelle.

La reprise

Si le traitement peut s'interrompre, il faut concevoir sa reprise. En fait, grâce aux automates à états, cette conception est déjà établie. Le traitement de masse s'est arrêté dans un certain état. Cet état exprime qu'une activité était en cours et que certains objets n'ont pas subi la transformation assurée par le traitement. Ces objets portent eux-mêmes un état. En fonction de la valeur d'état, on peut savoir quels sont les objets traités avant l'interruption et ceux qui ont été laissés sur le côté. Le traitement peux donc reprendre à l'endroit exact où il en était au moment où il a été interrompu : étape du traitement, ensemble d'objets restant à traiter dans cette étape.

Il est donc inutile d'imaginer un dispositif du genre « table de gestion des interruptions ». Cela n'apporte rien, au niveau logique, sinon une parade pour ne pas établir correctement les automates. Le journal (voir plus loin au chapitre « Sécuriser le traitement ») permet de retrouver les traitements qui ont été interrompus et les causes, éventuellement l'endroit…

Négociation logique / technique : dispositif technique d'émission et de consommation des événements[21].

Le retour arrière

Dans des cas extrêmes, l'interruption du traitement ou une condition particulière rencontrée en cours de traitement peut imposer de renverser toutes les transformations précédentes qui ont pu être entérinées. Ces transformations, validées dans le cadre de transactions précédentes, portent :

- soit sur les objets traités par l'étape interrompue,
- soit sur les étapes antérieures.

La conception doit, dans ces cas, prévoir la procédure de retour arrière, retour à la situation *ante*. Les moyens envisageables sont :

- concevoir des traitements de sauvegarde et restauration pour rétablir le système,
- ajouter, sur les automates des objets, des « états d'attente » dans lesquels les objets restent positionnés tant que le traitement n'a pas complètement abouti.

Dans cette deuxième solution, une étape supplémentaire clôt le traitement : en une transaction, elle parcourt tous les objets et les bascule dans l'état définitif.

[21] Ce peut être une table dans laquelle les demandes sont enregistrées. Les traitements interruptibles scrutent cette table. Quand ils trouvent un événement les concernant, ils effacent la ligne, c'est-à-dire qu'ils consomment l'événement.

Concevoir le traitement (suite)

Prendre en compte les possibilités techniques

L'ordonnanceur

L'ordonnanceur est un dispositif technique qui permet à l'Exploitation de piloter l'exécution des traitements *batch*. Ces traitements sont représentés dans le modèle logique, soit sous la forme d'un service asynchrone, soit sous celle d'un graphe d'activité attaché à un agrégat logique. Dans tous les cas, un algorithme décrit le fonctionnement du traitement et ses possibilités de parallélisme et d'interruption. Dans le cas d'un service, sa signature indique les paramètres à valoriser. Elle exprime aussi les possibilités de parallélisme paramétrique (cf. p. 29).

Les états de sortie

Le concepteur logique doit être averti des capacités des outils producteurs d'états[22]. En effet, ces outils offrent la possibilité de décrire non seulement la statique de l'état de sortie, mais aussi une partie de la dynamique : classements, ruptures, totaux, sous-totaux, etc. Il est donc inutile d'aller trop loin dans la conception logique.

Ce qui reste impérativement dans le modèle logique :

- l'exécution des règles « métier » ;
- les éventuels calculs par lignes (c'est-à-dire pour un objet et non pour l'ensemble des objets traités)[23] ;
- la constitution du flux de données que l'outil exploitera.

Le moteur de règles

La présence, dans l'architecture cible, d'un moteur de règles a un impact sur la conception logique. D'une part, elle limite l'effort à fournir pour la formulation des règles. D'autre part, elle conduit à prévoir l'interaction des services avec un service ou un dispositif qui encapsule le moteur de règles. Le concepteur décide à quel moment ce service utilitaire doit être appelé.

Une réflexion doit être menée sur ce point :

- quels types de batch peuvent recourir à un moteur de règles ?
- quelles traces dans le modèle logique ?

Si les services asynchrones recourent effectivement au moteur de règles, l'algorithme doit isoler l'invocation du moteur à l'extérieur des boucles massives.

[22] Par exemple : Hyperion.

[23] Hyperion se charge des totalisations par colonnes.

Concevoir le traitement (suite)

Quelques conceptions typiques

Typologie des traitements *batch* :

- unitaire (illustration 2) / ensembliste (illustration 1)
- extraction des données (illustration 1)
- injection
- calculs

Représenter le traitement

<< Faire le point sur la notation UML >>

appel asynchrone

algorithme des sv asynchrone

Concevoir le traitement (suite)

La description des services asynchrones

La définition et l'appel

Traité plus haut (cf. p. 16).

Les services synchrones ➔ PxM-40.

Nous apportons, ici, des précisions concernant les services dits asynchrones.

L'algorithme

La technique algorithmique retenue combine le diagramme d'états et le diagramme d'activité. Dans le cas des services synchrones, seul le diagramme d'activité est utilisé pour décrire l'algorithme. C'est normal : un service synchrone est plus petit et n'implique pas des gros paquets d'objets, ni un découpage en étapes.

Les stockages

Dans les traitements de masse, il est parfois besoin de recourir à stockages intermédiaires. Ils apparaissent sur l'algorithme sous la forme de flots d'objets entre les actions ou, à l'intérieur du pseudo-code, avec un foncteur dédié (ci-dessous).

Dans le modèle logique, on ne s'intéresse pas à la réalisation physique de ces flux. Il suffit de désigner le type complexe (*data type*) qui décrit le flux. Il est possible d'indiquer l'état de l'objet, si cela apporte une précision utile pour la compréhension du traitement.

Le concepteur utilise le nom de l'objet (apparaissant devant le caractère ':') pour préciser la nature du flot. Le même nom peut être utilisé sur plusieurs flots d'objets du diagramme.

Le nom peut, aussi, être valorisé à '*' pour dire que le flux contient plusieurs objets.

Le foncteur « conserver »

Dans le cas d'un service asynchrone dont le résultat n'est pas passé en paramètre, il faut conserver ce résultat en l'associant au jeton.

Le mot réservé proposé est : « conserver ». Il s'agit d'un foncteur car il désigne un mécanisme particulier.

La syntaxe est : conserver (<< le flux >>, << le jeton >>), où :

- << le flux >> représente une variable dans laquelle ont été agrégés les résultats du service (éventuellement : « sd », tout simplement) ;
- << le jeton >> permet d'identifier l'instance de flux et de faire le lien avec les services associés (quand il s'agira de récupérer le flux).

Ce foncteur sera traduit, en termes techniques, par une solution de support de masse. La conception du logiciel déterminera l'option la mieux appropriée :

- enregistrement dans une base de travail ;
- constitution d'un fichier (par exemple en xml).

Les ouvertures et fermetures de fichiers devront retenir l'attention du concepteur du logiciel.

Optimiser le traitement

Les actions d'optimisation

Introduction

Afin d'optimiser le fonctionnement du système, le concepteur logique met en œuvre les actions suivantes :

- Évaluer le comportement.
- Mettre en place le fonctionnement « au fil de l'eau ».
- Fusionner des étapes.
- Découper le traitement en étapes.
- Isoler les écritures des fichiers.
- Factoriser les traitements.

Elles sont examinées dans les sections qui suivent.

Évaluer le comportement

Intérêt

Pour éclairer les décisions de la conception logique, le concepteur doit disposer d'information sur les conditions de fonctionnement et sur le comportement général du système. Des informations lui sont nécessaires, en provenance des aspects « amont ». Il doit posséder, également, des connaissances sur le système, dans son aspect physique. Il assume, en effet, une part des responsabilités sur le fonctionnement global du système et cherche à optimiser les performances par des choix d'ordre logique.

En amont Des informations quantitatives (nombre d'objets, volumes de données, fréquence) devraient provenir des modèles amont. À défaut, on devrait les trouver au moins dans les spécifications fonctionnelles. Ces informations quantitatives ont une valeur fonctionnelle, avant tout. Le concepteur en a besoin pour prendre certaines décisions d'optimisation. Elles servent aussi au dimensionnement de l'architecture matérielle.

En aval Le concepteur logique a, également, besoin de connaître les coûts de traitement. Il s'agit de connaissances générales sur le comportement physique du système et qui permettront d'arbitrer entre plusieurs options. Par exemple : temps moyen d'accès en base, coût d'une écriture dans un fichier, coût des recherches… Le concepteur, en effet, se trouvera dans la position de choisir entre plusieurs options : pour un même besoin, l'algorithme pourrait comporter une seule ou plusieurs recherche, etc. Le concepteur doit aussi arbitrer entre des solutions *batch* pur ou des solutions au fil de l'eau. Ces arbitrages ne peuvent se faire que sur la base d'un connaissance suffisamment objective du système réel. Cette connaissance généralise le savoir acquis par l'Exploitation, sur le fonctionnement réel du système.). Elle peut prendre la forme d'abaques ou de données chiffrées sur les comportements.

Armée de ces informations provenant de l'amont et de l'aval, le concepteur logique, situé en position intermédiaire, peut évaluer le comportement prévisible des solutions qui s'offrent à lui. Cette action est surtout utile quand le concepteur se place dans la situation d'arbitrer entre des solutions interactives ou différées (dont *batch*), avec tout l'éventail des possibilités intermédiaires (voir en annexe).

Optimiser le traitement (suite)

Mettre en place le fonctionnement « au fil de l'eau »

Définition

Le « fil de l'eau » désigne une solution de transformation de l'information, objet par objet, par opposition aux traitements de masse.

Prenons l'exemple d'une somme à calculer sur un ensemble d'objets « métier ».

1. La solution classique consiste en un traitement batch qui balaie la table et réalise le cumul.
2. Dans la solution au fil de l'eau, chaque modification de la valeur d'un objet provoque l'actualisation de la somme. La somme est conservée par la machine ensembliste. Elle peut être fournie, soit par un service ad hoc, soit par la structure de données de la machine ensembliste.

L'impact

En recourant au fil de l'eau, le concepteur réduit la part laissée aux traitements *batch*. Il transforme le système : les acteurs du systèmes recevront plus de réponses en interactif, ce qui augmente le confort d'utilisation.

Une objection

Le traitement au fil de l'eau s'impose quand il n'y a pas propagation entre plusieurs machines. C'est le cas quand les transformations restent localisées sur un objet.

En revanche, les cas où la solution « au fil de l'eau » impliquent plusieurs machines soulèvent une objection, d'un point de vue technique. Surtout quand il y a interaction entre une machine élémentaire et une machine ensembliste, comme dans l'exemple donné ci-dessus. En effet, cette configuration risque de poser des problèmes de verrous, si les services élémentaires s'adressent régulièrement aux services ensemblistes pour mettre à jour l'information de la machine. Le concepteur logique doit donc anticiper les accès concurrents à la machine ensembliste. Pour cela, il doit pouvoir évaluer la fréquence des appels par les services élémentaires. Quand la transformation qui provoque l'actualisation au fil de l'eau est plutôt rare, il n'y a pas de problème. Dans le cas contraire, quand l'actualisation est fréquente, le fonctionnement risque d'être considérablement ralenti par la pose de verrous sur la machine ensembliste.

La réponse

Tout d'abord, il convient de distinguer :

- la logique du fonctionnement ;
- le problème de performance lié à une architecture technique.

Du point de vue logique, ce n'est pas une mauvaise chose que le service élémentaire s'adresse à la machine ensembliste pour l'informer. Ce genre d'interaction est circonscrit, bien sûr, dans les limites d'un atelier.

Au moment de la conception du logiciel, la solution au fil de l'eau pourra être tempérée par des dispositions évitant de bloquer la machine ensembliste, par exemple :

Le stockage d'informations intermédiaires par la machine élémentaire ;

L'exploitation de ces informations intermédiaires ou des changements d'états du jour, par des traitements en léger différé pour actualiser les informations de la machine ensembliste.

Optimiser le traitement (suite)

Fusionner des étapes

Le principe

Il arrive que la SFD, au-delà de la spécification du résultat attendu, fixe les étapes du traitement. Elle outrepasse, alors, son objectif et devient organique. Le concepteur logique peut, parfois, décider de fusionner ces étapes pour rendre le traitement plus efficace, en évitant de balayer plusieurs fois la même table.

Les limites

Il existe des limites à la fusion des étapes.

Les écritures

Si une étape comprend une écriture dans un fichier, elle ne peut pas être absorbée par une autre étape. En effet, elle fragiliserait l'ensemble car l'accès au fichier introduit un risque technique.

La durée maximale

Le dossier d'architecture technique peut recommander que la durée d'une unité de traitements n'excède pas une durée particulière.

Une unité de traitements correspond à une séquence de *jobs* entre deux sauvegardes. Ce sont ces points de sauvegardes qui définissent les points de reprises possibles. Cette vision ne tient pas compte de l'approche de conception fondée sur les automates.

Découper le traitement en étapes

Le principe

Le mouvement inverse du précédent peut aider à optimiser le traitement. En effet, la mise en parallèle de plusieurs flots d'exécution est un moyen d'optimisation dans la mesure où elle ajuste le traitement aux capacités disponibles à un moment donné.

Pour que cela soit possible, le traitement ou la chaîne de traitement doit présenter un nombre suffisant d'étapes dont quelques-unes peuvent s'exécuter indépendamment. Le souci de l'optimisation motive le concepteur logique. Les techniques de l'automate à états et de l'algorithme, présentées plus haut, lui permettent de mener cette réflexion.

L'activation des possibilités de parallélisme reste une option à la discrétion de l'Exploitation.

Le parallélisme paramétrique offre un moyen de fractionner les étapes. C'est l'exemple, évoqué plus haut, du traitement des dossiers par directions régionales. Pour obtenir ce résultat, le concepteur doit prévoir des paramètres dans la signature des services. À charge pour l'Exploitation ou l'ordonnanceur d'appeler le services pour toutes les valeurs des paramètres (dans l'exemple, toutes les directions régionales).

Optimiser le traitement (suite)

Isoler les écritures des fichiers

Le principe

Un impératif : les écritures dans des fichiers sont séparées du reste du traitement et constituent des étapes distinctes. La raison est que ces écritures introduisent un risque technique. Il importe de les confiner.

Si on considère le fonctionnement d'une extraction, produisant un fichier, on identifie plusieurs cas de figure :

- l'extraction est documentaire : elle n'est pas liée à un changement d'état, de signification fonctionnelle ;
- l'extraction est globale : en cas d'échec, le nouveau fichier écrase l'ancien ;
- l'extraction peut être partielle : en cas d'interruption, le fichier en l'état est utilisable ; lors de la reprise, il faut reprendre au point où le premier traitement s'est interrompu, d'où la nécessité d'un dispositif pour reprendre au bon endroit (état sur les objets ou pointeur conservé sur le traitement, par la MLO).

Le concepteur logique utilise cette échelle de possibles pour déterminer le comportement du traitement qu'il décrit.

La troisième situation est la plus confortable.

Étudier la durée de vie des stockages intermédiaires (tables ou fichiers) : tables temporaires, tables existant entre deux étapes, tables permanentes mais nettoyées (selon quel rythme ? par le traitement, par un traitement global…).

Description des fichiers (« puits de données »). Cas du fichier en entrée : quand il n'existe pas, quand il est vide, quand il ne contient que les enregistrements de début et de fin…

Factoriser les traitements

Éviter d'avoir des traitements qui font plusieurs fois la même étape

sous-totaux par lignes ou par colonnes

pré-traitement

exemple : moteur de règles ➔ le moins d'appels possible

Optimiser le traitement (suite)

Dénormaliser le modèle des services

Le principe

Dans les cas extrêmes, quand des problèmes de performance sont avérés, le concepteur logique dénormalise le service. Parce que la dénormalisation repousse la solution aux marges de l'architecture de services, voire en dehors, cette option requiert une dérogation expresse de la part de l'architecte logique.

Les possibilités de dénormalisation sont :

- requêtes et exploitation poussées du SGBD, en violation des règles imposées à l'architecture de données ; particulièrement : jointure entre des tables dépendant d'ateliers différents ;
- étapes supplémentaires permettant la constitution d'un cache de données (soit au démarrage du traitement, soit entre deux étapes normales) ;
- à l'extrême, renoncement à faire un service ➔ solution sous la forme d'un programme.

La dernière option n'est pas du ressort de la conception logique. Le concepteur logique spécifie toujours le ou les services en réponse à un besoin. C'est le concepteur du logiciel qui, en concertation avec l'architecte technique, décidera de traduire le service logique non pas en appliquant les règles générales, mais sous la forme traditionnelle d'un programme.

Compléments

Autres cas

Le concepteur a la possibilité de concevoir, sur dérogation, des services pour encapsuler des requêtes optimisées, violant les règles imposées par l'architecture des données et les dépendances entre les ateliers. Par exemple : avec des jointures inter-ateliers...

On peut aussi envisager des services qui ne rapportent qu'une partie des informations fournies par les accesseurs habituels. Par exemple, un flux de plusieurs sinistres sans le contenu des dommages (alors que la structure de données infoDommage est incorporée à celle de Sinistre).

Ces déviations doivent être rares. Dans tous les cas, on préférera paramétrer le comportement du service de base, plutôt que de créer un autre service. La contextualisation des services, qui peut elle-même être prise en charge par la VEP (Virtual Engine for Praxeme), est le meilleur moyen pour absorber ces variations sans ajouter de service et en respectant la signature du service de base. Ceci est un point qui doit être traité dans la négociation entre l'architecte logique et l'architecte technique,

Recommandation

Ne pas créer autant de services asynchrones que d'états de sortie demandés (en faisant varier la composition du flux), mais :

- paramétrer les requêtes (cf. les services de recherche) ;
- utiliser les types complexes associés aux machines, sans en créer de nouveaux.

C'est l'interface ou les « services généraux » (éditique, courrier, présentation…) qui sélectionneront l'information.

Sécuriser et suivre les traitements

Les dispositifs pour assurer la maîtrise de la dynamique

Les actions

Nous recherchons la plus grande lisibilité sur le fonctionnement du système et la plus grande liberté pour le piloter. Ces préoccupations ont conduit à mettre en place la gestion des incidents sur les services synchrones. Dans le mode de fonctionnement différé et les traitements de masse, ces préoccupations se renforcent. Ce mode de fonctionnement réclame un surcroît d'attention de la part du concepteur logique :

- le traitement des rejets ;
- les habilitations pour les traitements *batch* ;
- le journalisation ;
- la traçabilité ou auditabilité.

Ce chapitre se conclut par le modèle des informations génériques, liées à la sécurité et au suivi de l'excéution.

Concevoir le traitement des rejets

Rejeté = non traité (quelle que soit la raison)

Rejet technique : exemples = dossier verrouillé, erreur fatale, ressource indisponible…

Rejet fonctionnel : conditions non remplies (peut déboucher sur un batch de compensation)

Reprise sur interruption volontaire ou non : dans le cas normal, la cohérence du traitement reposant sur un ou plusieurs automates, il suffit de relancer le même traitement. Seuls les objets qui n'ont pas changés d'états lors de la précédente exécution seront pris en compte par la nouvelle…

Recyclage des dossiers rejetés (la conception logique exprime les conditions qui obligent à rejeter un dossier ; l'automate, dans la mesure du possible, est un bon moyen ; s'il ne convient pas, on est obligé de concevoir un stockage intermédiaire : liste des dossiers rejetés ou table de recyclage ; mais le choix du support est laissé à la conception du logiciel, sur la base des possibilités offertes par l'architecture technique)

dans le batch ensembliste : que se passe-t-il ?

Comment reprendre la table de recyclage ?

Foncteur pour enregistrer (dans les cas où le recyclage ne peut pas s'appuyer uniquement sur l'automate)

On peut envisager d'inclure la liste des rejets dans la signature du service. En fait, le service possède un paramètre par lequel il reçoit le flux des objets sur lesquels il doit s'exercer. Le flux peut être nominal, résultant d'un requête réalisée par un autre service. Dans d'autres cas, le flux se réduira aux exceptions : les objets qui n'ont pas pu être traités lors d'une précédente exécution.

À ajouter dans le modèle logique : l'arbre des signaux pour les rejets. En fait, rien ne change par rapport à la gestion des incidents établies pour les traitements interactifs.

Sécuriser et suivre les traitements (suite)

Les habilitations

Si quelqu'un déclenche un traitement, est-ce que d'autres personnes (son supérieur, ses collaborateurs…) peuvent s'en informer ou obtenir le résultat ou même interrompre ou annuler le traitement ? Sous quelles conditions ?

Qui a le droit d'interrompre un traitement ? Qui a le droit de relancer un traitement qui a été interrompu ?

Ces questions sont de nature purement fonctionnelle, plus précisément : organisationnelle.

Sécuriser et suivre les traitements (suite)

La journalisation

Comment indiquer l'écriture d'un CRE : foncteur avec, en paramètre, l'information à enregistrer.

Type info de base pour les CRE, éventuellement complété par des info spécifiques (par exemple, absorbe le type info de la MLO ; chercher à limiter la prolifération des types info).
Dans certains cas, l'écriture du CRE peut être est implicite (**issue de la négociation Lq/Tq**). Notamment, lors de la validation d'une transaction.

Que doit dire la conception logique sur ce point ? C'est elle qui détermine le besoin de contenu, en interprétant la spécification fonctionnelle et en anticipant sur le comportement du système. Renforcer sa robustesse et garantir sa lisibilité.

Les comptes rendus d'exécution (CRE) ➔ historique du batch

Le contenu du CRE est déterminé par l'automate. Le CRE reprend :

- les valeurs d'état qui ont été atteintes à l'issue de chaque étape, y compris les éventuels sous-états et reprise sur état historisé ;
- les pré et post-conditions qui ont été confirmées ou infirmée.

On y indique également le pas de commit et chaque validation ainsi que son résultat : échec ou succès.

La traçabilité

Les traitements critiques, d'un point de vue fonctionnel ou légal, incorporent une solution qui permet de tracer les mises à jour dans la base. Caractéristique d'auditabilité. Conserver la trace des acteurs qui modifient la base.

L'auditabilité s'appuie sur le système actuel. Un code « acte de gestion » est attribué à chaque traitement *batch*. Dans le cas d'un traitement segmenté, le code doit comporter une partie pour désigner l'ensemble et une autre pour l'étape ou le flot.

Existe-t-il des exigences réglementaires sur certains traitements ? Par exemple : archivage fiscal.

Le modèle des informations de sécurité et de suivi

CRE, recyclage et jeton

Le jeton identifie le traitement et les flux qui lui sont associés. Il se base sur l'identité de l'acte de gestion, véhiculée par le contexte. Quand le traitement comporte plusieurs étapes ou plusieurs flots d'exécution, le jeton qui les identifie complète l'identifiant du traitement global avec un libellé distinctif. Ce libellé mentionne :

- le nom du service qui réalise l'étape ;
- la valeur du paramètre, dans le cas où le traitement est segmenté par ensemble de données ;
- etc.

Prendre les décisions sur l'aspect logiciel

En collaboration avec un représentant de l'architecture technique. Par exemple, discuter au cas par cas du pas de commit.

Évaluation de la durée de traitement (cf. évaluer le comportement) mais ici, on est dans la conception du logiciel, donc avec davantage de connaissances sur l'architecture technique

Exemple de décision : un algorithme de Service Logique Organisation (SLO) établi pour montrer le // possible est repris, au niveau logiciel, sous la forme de l'ordonnanceur batch.

Travail à faire pour les états de sortie

Table de recyclage, à concevoir en détail (ce n'est pas du ressort de l'aspect Logique)

Que faire des fichiers créés ? Purge ? Archivage ?

Le déploiement et la conception du logiciel

L'architecture des données

Les questions essentielles sont les suivantes :

- Duplication de tablesTables ou fichiers intermédiairesLes ressources en traitement offertes par le SGBD

Décisions de l'aspect logiciel

Éventuellement établies une fois pour toutes par l'architecture technique :

- Format et localisation des éventuels fichiers intermédiaires.

- Le flux logique est constitué par agrégation dans la variable de travail du service (le plus souvent : « sd »). Ces agrégations s'obtiennent par de simples affectations dans le peudo-code. En fonction du format retenu au niveau logiciel, le programmeur peut être amené à transformer ces agrégations en écritures dans un fichier.
- De même, la reprise d'un flux physique par un autre service (asynchrone ou non) dépend de la solution retenue dans l'aspect logiciel.

Consignes d'exploitation

Pour l'ordonnancement des chaînes *batch* quand il y a des contraintes d'ordonnancement : dessin de la chaîne sous la forme d'un diagramme d'activité, par exemple.

Ce n'est pas le même que celui établi dans le modèle logique.

On se place, ici, dans l'aspect physique. Un diagramme de déploiement peut s'avérer nécessaire pour montrer les phénomènes de circulation ou de duplication à travers l'architecture physique.

Voir les possibilités de ce diagramme et les stéréotypes associés pour qualifier les phénomènes de cet aspect…

Éditique

2 possibilités :

•strate Présentation appelant des services externes

•strate Organisation : MLO incluant les moyens physiques de l'éditique

dans le 1er scénario, l'éditique est client des services (dont elle reçoit des flux)

dans le 2nd, elle est incorporée à l'architecture de services, ses moyens sont encapsulés et elle pourvoit le reste des MLO en services d'édition (parmi les "services généraux")

La représentation de la chaîne batch

Dans l'aspect logiciel

L'automate et l'algorithme ne représentent que l'ordonnancement. Si l'on veut représenter les ressources – bases et fichiers – le diagramme à utiliser est plutôt le diagramme des composants, voire le diagramme de déploiement. Ce dernier aspect sort de la modélisation logique : le diagramme de déploiement est l'outil de représentation de l'aspect physique.

Annexes

La négociation Logique/Technique

L'activité

La négociation est une activité au cours de laquelle les termes de la modélisation logique sont éprouvés au regard des contraintes et possibilités technologiques. Ils peuvent se trouver repoussés si l'architecture technique ne leur trouve pas de correspondance simple dans le logiciel ou, au contraire, si elle propose des mécanismes plus simples.

Les thèmes

Les points à négocier, autour des traitements différés :

- communication asynchrone (par événements, par messages) ;
- possibilité de services asynchrones (peut-on réaliser un traitement long, sous la forme d'un service, sans bloquer l'appelant ?) ;
- stockages intermédiaires le long d'une chaîne *batch* ;
- enregistrement transitoire des flux ;
- possibilité d'accès direct aux BD (sous dérogation) ;
- matérialité du foncteur « conserver » et mécanisme du jeton.

Les points de vigilance

- Stockage intermédiaire.
- Mécanisme du jeton.

Négociation logique / technique

Les solutions envisagées à travers les deux illustrations obligent à considérer :

Le dispositif pour l'émission et la réception du signal en cas de demande d'interruption (ce dispositif peut varier selon les cibles techniques).

Le dispositif pour l'interruption et la reprise (impact de la solution technique sur la formulation du cas d'utilisation).

La persistance des structures de données pour les machines ensemblistes et pour les machines de la strate « Organisation ».

Utilisation du système de journalisation

Point de commit dans la strate « Métier »

Parallélisme interne aux services

Construction du jeton. DL (27/12/06) : le jeton est contextualisé. Inutile de le faire figurer dans la signature.

Annexes (suite)

Récapitulatif des possibilités d'action de l'Exploitation

L'ordonnancement des traitements

Entre les batch

À l'intérieur des batch (les étapes)

ordonnancement des batch (// ou pas) en respectant les contraintes fonctionnelles (ces contraintes et possibilités doivent apparaître clairement dans la conception logique)

Le « pas de *commit* »

fixer le pas de commit

deux solutions :

- paramétrage qualitatif et explicite du service asynchrone (si critère à contenu fonctionnel ; exemple : tous les dossiers pour une direction donnée)
- paramétrage quantitatif et implicite, à la discrétion de l'Exploitation (ex. validation tous les 100 dossiers traités)

implicite dans la modélisation logique : foncteur ; pris en charge par le dispositif technique

ces deux types de segmentation du traitement, qualitatif et quantitatif, ne s'excluent pas

Annexes (suite)

Pseudo-code

Pseudo-code

L'intégration des traitements différés dans l'architecture de service demande quelques enrichissements du pseudo-code :

Des foncteurs comme abstractions des manipulations de fichiers (pour constituer les fichiers d'échanges, notamment).

Des mots réservés pour indiquer les bornes des transactions.

Ces ajouts dépendent des résultats de la négociation technique (point suivant). Ils entraînent la mise à jour du document de référence AMS-42.

tri

commit avec pas de commit implicite (assuré par le dispositif technique)

sans doute besoin de faire évoluer la syntaxe pour appel asynchrone (dans le cas des services multimodaux)

Annexes (suite)

Les activités et les points de vue

Les points de vue

Spécification Le terme « spécification » ne suffit pas à caractériser une activité. Spécifier, c'est exprimer des exigences. Les questions sont :

- Qui exprime les exigences ? Ce peut être l'utilisateur vis-à-vis de l'informaticien, mais aussi le concepteur vis-à-vis du développeur…
- Sur quoi portent les exigences ? Elles peuvent être fonctionnelles, opérationnelles ou structurelles. Dans ce dernier cas, elles ne sont pas perçues par l'utilisateur, mais peuvent concerner une MOA soucieuse de la qualité et des évolutions de son patrimoine applicatif[24].

Points de vue Nous préférons, donc, distinguer :

- le point de vue externe (utilisateur ou MOA) ;
- le point de vue interne (informaticien).

Le *point de vue externe* sur les traitements différés commence à être formulé avec la « Vue de l'utilisation », dans l'aspect pragmatique. Le point de vue externe sur le logiciel et sur les états de sortie complète cette Vue. On se trouve dans le même cas que pour une application interactive. La spécification des états peut se faire sur le même mode, par maquettage.

Le *point de vue interne* s'exprime, d'abord, dans le modèle logique. Il embrasse, également, des questions de localisation et de ressources matérielles, ainsi que de comportement du système. Ces questions se répartissent sur les aspects : matériel, technique, logiciel et physique (c'est-à-dire, les quatre aspects proprement informatiques).

Les postures

Orthogonalement aux points de vue, la méthodologie introduit la dichotomie analyse/conception, caractérisée en tant que posture plutôt qu'activité.

La posture définit une attitude par rapport à l'objet étudié.

- Dans la posture d'analyse, l'intervenant observe et décortique la réalité dans son champ d'étude. S'il s'agit d'analyser le besoin, il cherchera à le quantifier, à en détecter les motivations profondes et à en percevoir les implications.
- Dans la posture de conception, l'intervenant invente : il prend en compte les données du problème et utilise les possibilités à sa disposition pour proposer une solution.

Quel que soit le champ d'étude, quel que soit l'aspect du système, il convient d'adopter successivement ces deux attitudes.

[24] Dans le cas d'une maîtrise d'ouvrage exerçant sa fonction de propriétaire du système.

Annexes (suite)

Les activités d'analyse et de conception associées au *batch*

Les conséquences

Spécifier un *batch* revient à définir le résultat attendu par l'utilisateur. On aura intérêt à distinguer la demande initiale, d'abord exprimée puis analysée, et l'expression définitive, résultant d'une conception et requérant l'accord de la MOA.

Ceci est vrai, en général, mais plus encore en ce qui concerne les traitements différés, parce que ceux-ci sont particulièrement touchés par les anciennes conceptions de l'informatique.

Les quatre types d'activités

Le tableau ci-dessous récapitule les activités, déduites du croisement des points de vue avec les postures.

La spécification fonctionnelle (ou spécification externe), au sens strict, devrait intervenir uniquement après la conception externe. Elle décrit le résultat attendu et les modalités, du point de vue des utilisateurs.

Posture	Point de vue externe	Point de vue interne
Analyse	**Analyse du besoin** Expression du besoin par la MOA	**Analyse du contexte** Prise en compte des contraintes et du contexte par l'informaticien
Conception	**Conception externe** Définition, validée par la MOA, des résultats attendus et des modalités d'obtention	**Conception interne** Détermination des services nécessaires et description des services et programmes

Le processus

La séquence normale de ces activités est la suivante :

1. Analyse du besoin (ou analyse fonctionnelle).
2. Analyse du contexte (ou analyse organique).
3. Conception externe.
4. Conception interne.

Cette séquence est imparable quand il s'agit du détail d'une demande. Elle peut, néanmoins, être contournée : dans le respect de la Topologie du Système Entreprise, le concepteur peut déduire des services logiques, indépendamment d'un besoin exprimé. Cette possibilité conduit à inscrire, dans l'architecture logique, de nouveaux services logiques qui seront utilisés pour les traitements différés.

+ part du Lq et du Ll

Plus en détail des actions ci-dessous, qui structurent le document

Annexes (suite)

Au fait, pourquoi différer ?

Les questions

En appliquant cette démarche de conception, on s'intéresse surtout à dégager les motivations qui sous-tendent la demande. À un moment, il faut se poser les questions suivantes : pourquoi différer ? pourquoi ne pas faire en sorte de donner une réponse instantanée ?

Cela peut paraître bête, mais c'est le seul moyen de s'arracher des anciennes conceptions !

Le style retenu pour la conception logique (SOA) encourage à élaborer des solutions au fil de l'eau.

Les raisons de différer

Les raisons qui conduisent à proposer une solution *batch* :

- Le temps de traitement nécessaire excède la durée supportable dans une interaction entre un acteur et le système (calcul long, impression de masse, etc.).
- Le traitement mobilise des moyens communs qui nécessitent une planification de production (par exemple : blocage nécessaire de la base de données, imprimante spéciale, production physique nécessitant une surveillance par un opérateur humain).
- Le traitement sollicite un composant logiciel qui n'a pas été restructuré et dont la mise en œuvre ou les performances ne sont pas compatibles avec le comportement d'un service.
- Le traitement impose des échanges avec des portions d'architecture fonctionnelle non intégrées à l'architecture de services ou avec des progiciels (exemples : LOGIBAT, comptabilité).
- Les traitements se limitent, comme dans le cas précédent, à préparer des flux physiques entre systèmes ou applications (cf. document de SFD ANAEL). Ils ne nécessitent pas d'intervention humaine[25].
- Par habitude (on considère qu'il n'y a pas d'urgence à obtenir le résultat ; le résultat a toujours été donné en différé, comme dans le cas des états de gestion ou de surveillance).

L'attitude du concepteur

Parmi ces raisons possibles, quelques-unes sont mauvaises ! En tout cas, même quand la demande est exprimée en tant que traitement différé, il faut se poser la question. Chaque fois que c'est possible, on respectera l'adage : « ne pas repousser au lendemain ce que l'on peut faire le jour même ». Cette attitude amène des progrès considérables dans les processus « métier », le genre de progrès à forte visibilité.

[25] Ce cas ne devrait pas concerner l'utilisateur. Il est hors du champ de la spécification *fonctionnelle* en tant qu'elle fixe le point de vue externe. En effet, il s'agit seulement de compenser les limites d'une architecture logique ou logiciel insuffisamment intégrée.

Annexes (suite)

L'éventail des possibilités

Les composantes de la solution

Pour répondre à une demande de traitement lourd, le concepteur logique dispose d'un éventail de solutions résumées dans le tableau ci-dessous.

Les constituants à sa disposition et qui entrent dans la conception d'une solution de traitement lourd sont :

- les services synchrones ;
- les services asynchrones ;
- les programmes (hors architecture de services).

Le concepteur logique identifie l'ensemble des services que le programme pourra utiliser.

Position →	Instantané	Assimilable	Décalé	Différé
Qualité	**Interactif pur**	**Délai tolérable**	**Léger différé**	**Traitement lourd**
Architecture de services	Grâce aux agrégations « Au fil de l'eau »	Tâche en arrière plan (avec services asynchrones rapides)	Une IHM pilote directement des services asynchrones	Le batch est un service asynchrone ou un programme appelant les services.
Hors architecture de services				Le batch est un programme qui attaque directement les bases de données.

La troisième dimension

Pour que le recensement des possibilités soit complet, il faut introduire une troisième dimension : celle de l'intervention de l'utilisation. Les valeurs dans cette dimension sont :

- initiative de l'utilisateur pour le déclenchement ;
- initiative du déclenchement et suivi direct par l'utilisateur ;
- initiative indépendante de l'utilisateur (ordonnanceur, Exploitation) mais suivi par l'utilisateur ;
- aucune intervention de l'utilisateur (traitement entièrement sous le contrôle de l'Exploitation).

Le mode d'emploi

Le concepteur choisit, pour le traitement demandé, une des positions de la ligne « Architecture de services », en fonction de la nature et de la complexité du traitement. La solution doit tendre vers la gauche du tableau.

Sortir de l'architecture de services n'est possible que sur dérogation, dans le dernier cas (différé).

Rappel sur la spécification des traitements *batch*

La Vue de l'utilisation

Rappel

Vue externe

Point de vue local (un type d'utilisateur, une direction « métier »…) sur l'activité de l'entreprise. Une des deux vues de l'aspect pragmatique.

Le *batch* dans la vue de l'utilisation

Des traitements *batch* peuvent également être spécifiés par le biais des cas d'utilisation. Le libellé du cas d'utilisation doit exprimer la motivation réelle de l'utilisateur, plutôt que la solution ou le moyen.

Par exemple : éviter « Éditer les statistiques » ; préférer : « Surveiller l'activité ».

En procédant de la sorte, les avantages sont :

- laisser plus de liberté pour la conception de la solution (il y a peut-être moyen d'élaborer une solution en interactif ou de remplacer des kilos de papier par une présentation à l'écran) ;
- regrouper, dans le cas d'utilisation, différents scénarios relatifs à la même responsabilité (demander le traitement, surveiller son déroulement, obtenir les résultats…).

Les cas d'utilisation

Dans l'exemple précédent, le cas d'utilisation est construit à partir des motivations ou de la responsabilité de l'acteur. On peut identifier plusieurs types de besoin qui peuvent être exprimés sous la forme d'un cas d'utilisation ou d'une partie d'un cas :

- les fonctions courantes que tout cas d'utilisation devrait offrir ;
- les demandes pour déclencher des traitements différés ;
- la prise d'information sur les traitements demandés.

Les fonctionnalités à incorporer aux cas d'utilisation

Les fonctions courantes

Notamment :

- impression du dossier en cours de traitement ;
- production de pdf...
- copie d'écran ;
- émission : message, courrier...

Ces émissions de l'organisme sont suivies : information de la strate « Organisation ».

Différence avec les batchs

Différence entre ces fonctions et les batch : uniquement le temps de réponse (gradation entre instantané, quelques minutes et différé). Passé un certain seuil, ce ne peut être qu'un traitement différé.

Dans la majorité des cas, l'appariteur[26] dispose de toute l'information nécessaire (dans le contexte d'utilisation) : il suffit de la passer à des services support ("services généraux" dont il sera parlé plus loin).

Principe d'ergonomie : quel que soit le contenu de l'IHM, possibilité d'impression et d'émission.
Limite dans l'application de ce principe : cela ne vaut que pour l'information disponible sur le poste de travail (donc, à partir de la structure de données de la MLO).

Les demandes

Soit un cas d'utilisation à part entière, soit un scénario au sein d'un cas d'utilisation qui suit tout le déroulement de la demande au résultat. Préférence pour la 2ème option.

L'utilisateur peut choisir les modalités de production ou de déclenchement. Par exemple, sélectionner : canal, imprimante, document, message, message avec PJ, production (instantanée ou différée).

Distinguer entre la demande et la réception du résultat (ci-dessous).

Les résultats

Un scénario du cas d'utilisation (sous condition d'état précisant l'avancement du traitement différé).

Suivi des traitements => automate à états attaché au cas d'utilisation ; cet automate se synchronise lui-même avec l'automate du traitement lui-même (cf. p.Erreur : source de la référence non trouvée) ou de l'objet traité (demandé, planifié, en cours, traité...).

(conception interne : ce suivi est installé dans la strate "Organisation")

Pas très différent du suivi d'une procédure (quand l'intervention d'autres acteurs ne permet pas la réponse immédiate de l'organisme).

Description des états

Flux

les données à montrer

entête et lignes/colonnes

description : c'est une sélection sur un type Info (remarque pour la conception interne : ne pas créer de types supplémentaires). Il suffit d'indiquer les données que ne prend pas l'état (ou le contraire, selon commodité).

Requête

ce que l'on recherche

critères

exemple : "Sinistres catastrophe naturel"

Classement et rupture

exemple : "par catégories DAB"

rupture sur la société (cela revient à définir des regroupements)

Documentation SFD

bien distinguer expression du besoin et conception de la solution

référence au mdl Sq pour la signification des données et leur structuration

la correspondance avec les SD ne devrait pas se faire dans les SFD mais lors de la conception logique

c'est à ce moment-là que l'on décide de l'opportunité de créer un service pour produire le flux

Documentation SFD

Il sera possible de détailler :

le dessin des éditions. Préciser pour chaque zone de l'édition : son format d'impression, les règles de calcul, les règles de cumul, les règles de mise à jour des bases de données …

les règles de gestion associées au batch étudié,

[26] L'appariteur est un composant de la strate « Présentation ».

les moyens de suivi du traitement batch.

Il peut être important de mettre en place des mécanismes d'auto-contrôle (exemple : vérification des contrôle d'enchaînements entre fonctions, interprétation des codes retours …). Ceci doit permettre de déterminer les points de reprise pertinents dans l'application (sur survenance d'incident), et à l'application de contrôler elle-même sa logique de traitement.

Représentation

diag. de collaboration ou de séquence pour expliquer les règles pour rassembler les objets impliqués dans le traitement

Les possibilités de manœuvre

L'expression des besoins fonctionnels ne doit pas verrouiller les hypothèses et ilposer des présupposés inutiles à la conception de la solution.

Son devoir est de formuler les contraintes réelles, toutes les contraintes, rien que les contraintes – sans ajouter de fausses contraintes.

Cette remarque vaut particulièrement pour :

les stokages intermédiaires ;

l'ordonnancement des traitements.

La solution : plutôt que de représenter le déroulement du traitement sous la forme d'un algorithme (diagramme d'activité) ➔ conduit à une certaine linéarité ;

Exprimer les contraintes d'ordonnancement sous la forme d'un automate à états

Attaché au cas d'utilisation

Sur cet automate :

- Les états marquent des points de stabilité dans le traitement (le concepteur les reprendra dans la spécification des transactions).
- Les possibilités de parallélisme et les besoins de synchronisation sont clairement exprimés.

Le concepteur logique transformera cet automate (comme pour les traitements interactifs). Les possibilités de parallélisme seront préservées au maximum et laissées à la discrétion de l'Exploitation.

Typologie des batchs

batch d'édition (passif => consultation des données seulement)

batch de calcul (actif => création ou modification d'information)

Les catégories différés/fil de l'eau/long ne sont pas exclusives.

On a trois catégories chacune subdivisées:

Mode de traitement des dossiers : Ensembliste/Unitaire (fil de l'eau)

Mode de lancement du batch : Immédiat/différé/ordannacé (heure fixe)/dès

que possible (asynchrone).

Durée du batch : long/court

A cela s'ajoute la nature même du batch :

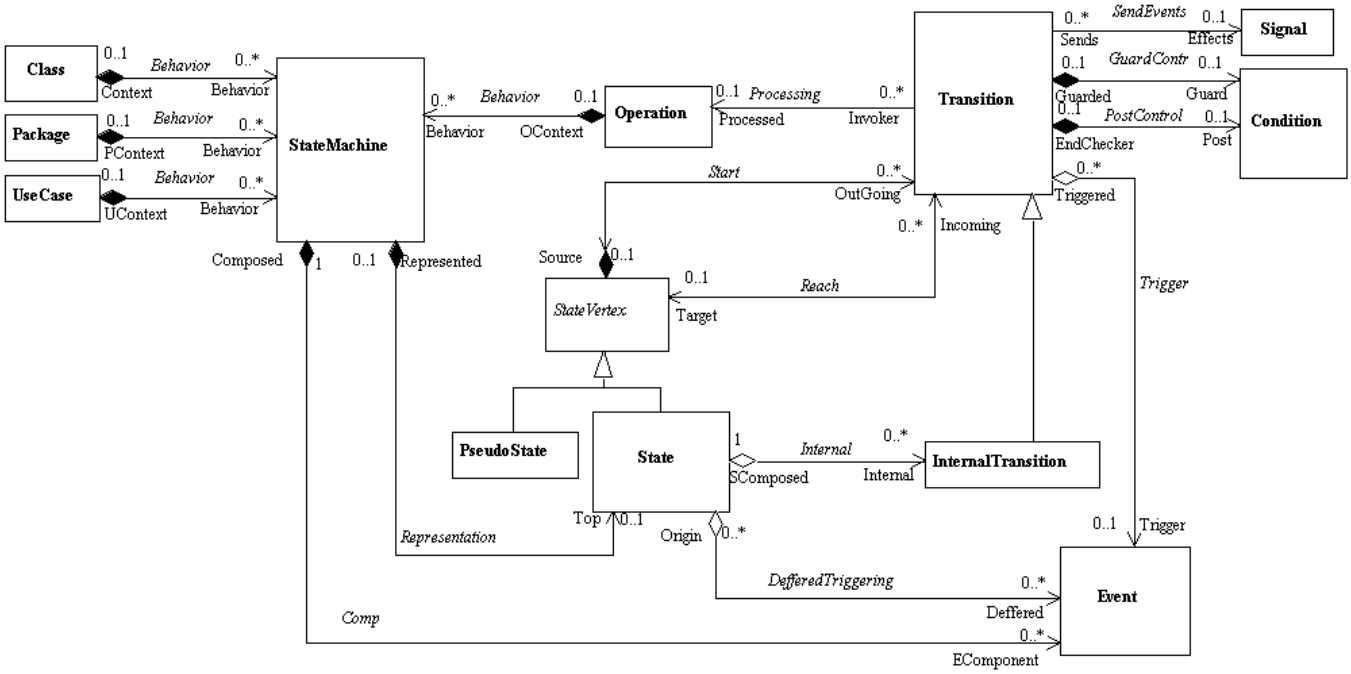
- Batch d'extraction (envoi vers l'extérieur de flux de données, d'édition,...)
- Batch de calcul (mise à jour des données internes à Règlements sur la base
- Règlements uniquement)
- Batch d'injection (réception de données de l'extérieur pour mise à jour)

Ces trois natures peuvent avoir une caractéristique de chacune des catégories citées plus haut (qui donc ne sont pas exclusives).

Les automates à états : le méta-modèle

Le méta-modèle UML permet d'attacher un automate à états (*StateMachine*) aux éléments de types :

- Classe (pour nous, dans la modélisation logique : les machines logiques) ;
- Paquetage (par exemple, pour représenter les processus à la façon de SPEM) ;
- Cas d'utilisation ;
- Opération.



Index

A
algorithme · 10, 11, 17, 25, 27, 29, 36, 37, 38, 39, 40, 42, 49, 60
architecture logique · 8, 10, 11, 20, 32, 54, 56
architecture technique · 13, 27, 31, 41, 42, 45, 48, 50
arrêt · 11, 36
asynchrone · 7, 8, 12, 14, 16, 17, 18, 20, 22, 24, 32, 38, 48, 50, 52, 60
au fil de l'eau · 8, 9, 12, 40, 41, 56
automate · 11, 21, 24, 25, 26, 27, 32, 35, 42, 45, 47, 49, 59, 60, 61

C
compte rendu d'exécution · 25
conception du logiciel · 6, 21, 32, 39, 41, 45, 48
consultation · 60
coût · 15, 40
CRE · 25, 47
cycle de vie · 25

D
dénormalisation · 44
diagramme d'activité · 7, 11, 17, 25, 27, 29, 35, 36, 39, 48, 60
diagramme d'états · 39
durée · 12, 16, 31, 42, 43, 48, 56

E
empan · 31, 32
ensembliste · 12, 13, 20, 34, 38, 41, 45
étape · 24, 25, 26, 27, 31, 33, 35, 36, 42, 43, 47
événement · 18, 22, 24, 35, 36
extraction · 13, 27, 38, 43, 61

F
fin de journée · 13
flux · 13, 16, 17, 18, 22, 23, 37, 39, 44, 45, 47, 48, 50, 56, 59, 61
fréquence · 40, 41

I
incident · 60
injection · 13, 38, 61
interruption · 18, 24, 27, 35, 36, 37, 43, 45, 50

J
jeton · 16, 18, 19, 21, 22, 30, 39, 47, 50
journalisation · 10, 25, 31, 45, 47, 50

L
lancement · 12, 13, 16, 60

M
machine logique · 16, 24, 25, 35
mise à jour · 8, 13, 52, 59, 61

N
nuit batch · 25

O
optimisation · 10, 40, 42
ordonnancement · 11, 12, 24, 27, 48, 49, 51, 60
ordonnanceur · 14, 18, 35, 37, 42, 48, 57

P
parallélisme · 24, 25, 27, 29, 30, 35, 37, 42, 60
paramètre · 16, 18, 22, 23, 29, 39, 45, 47
pas de commit · 31, 32, 47, 48, 51, 52
portée · 12, 24, 25, 31, 34
Praxeme Institute · 2
pré-traitement · 43
Production · 6, 7
programme · 8, 15, 24, 44, 57

R
recyclage · 45, 47
rejet · 13
reprise · 24, 27, 36, 43, 45, 47, 48, 50, 60

S
sécurité · 10, 45, 47
service asynchrone · 6, 16, 17, 18, 19, 20, 22, 25, 29, 32, 37, 39, 51, 57
service synchrone · 19, 22, 39
signal · 35, 36, 50
signaux · 34, 45
spécification fonctionnelle · 6, 8, 10, 12, 47, 54, 56
strate « Métier » · 19, 20, 22, 32, 50
strate « Organisation » · 8, 20, 22, 50, 58
strate « Présentation » · 8, 20, 21, 59
synchrone · 13, 14, 16, 19, 20, 26

T
table de recyclage · 45, 48
transaction · 29, 31, 32, 33, 36, 47
type complexe · 39
types complexes · 44
typologie · 12

U
unitaire · 12, 13, 38