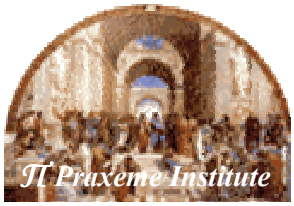


Praxeme Institute

Composant PxM-40 « Modus : La méthodologie Praxeme »

Guide de l'aspect logique

Objectif

Ce guide méthodologique expose les règles à appliquer pour identifier, concevoir et décrire les services logiques.

Ces règles comprennent les règles de dérivation à partir des modèles amont et les règles propres à l'architecture de services (SOA).

Le guide expose les procédés d'architecture logique et de conception logique.

Contenu

- Orientations de l'architecture de services
- Termes et représentations de l'aspect logique pour SOA
- Démarche de conception des services
- Thématique de la conception logique
- La dérivation du modèle sémantique
- La dérivation du modèle pragmatique
- Synthèse des règles

Rédacteur Dominique VAUQUIER

Version 1.0, le 10 août 2007

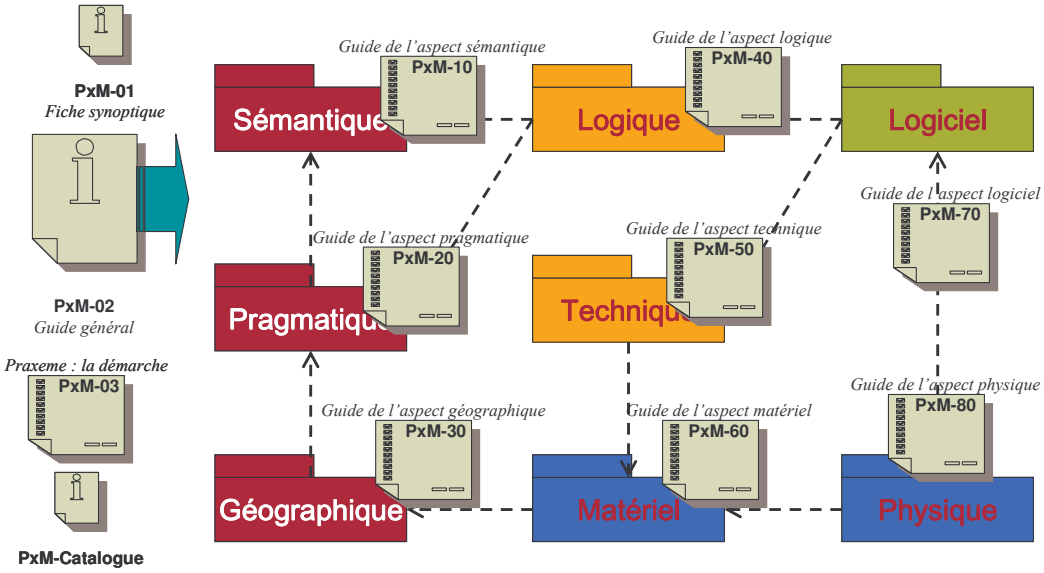
Éléments de configuration

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.

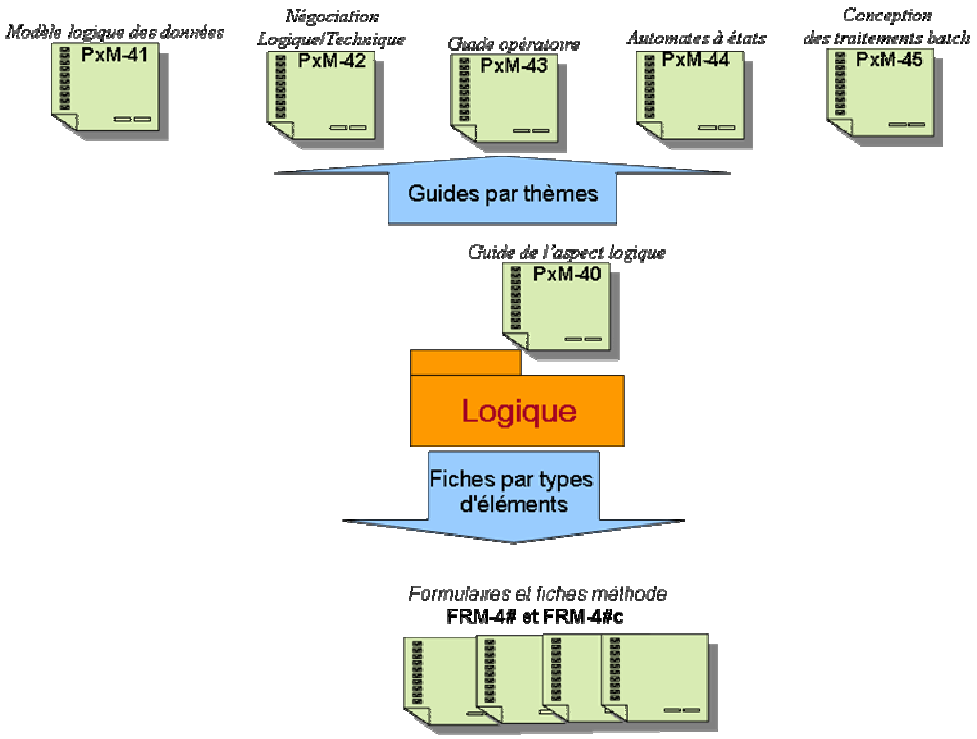
Figure PxM-40_1. Architecture du référentiel méthodologique



Documentation pour l'aspect logique

La conception logique soulève de nombreuses questions pointues. Ce guide donne une vision d'ensemble et positionne tous les thèmes. D'autres documents apportent davantage de détails sur des points précis. La figure suivante les recense.

Figure PxM-40_2. La documentation sur l'aspect logique



Éléments de configuration (suite)

L'historique

Indice	Date	Rédacteur	Contenu
	Mars 2004	DVAU	Première rédaction (Dromos : méthode Sagem pour l'urbanisation de l'informatique des systèmes de drones)
	Novembre 2005	DVAU	Version étendue (Amos : méthode de la DSI SMABTP ; approche SOA). Base de travail pour le projet SOA « Règlements ».
0.1	Décembre 2006	DVAU	Restructuration ; distinction entre éléments méthodologiques et décisions liées à un contexte d'architecture. Financement par Calyon DMC.
1.0	Août 2007	DVAU	Généralisation. Après expérimentation dans le cadre de la refonte du SI SMABTP. Réécriture financée par la SMABTP.
1.1	Septembre 2007 (prévision)	Px1	Revue par le Collège des concepteurs et architectes logiques (liste ci-dessous)
1.0			Version actuelle du document

Validation

Relecteurs : Pierre BONNET, Philippe DESFRAY, Fabien VILLARD.

Ont participé à la revue de ce document :

<< revue en cours au sein du Collèges des concepteurs et architectes logiques ; voir le site : www.praxeme.org >>

Disponibilité

Ce document est disponible sur le site Praxeme et utilisable dans les conditions définies page suivante. Les sources (documents et graphiques) peuvent être obtenues sur demande.

Propriétaire

Le référentiel Praxeme a été élaboré dans le cadre du chantier Praxime. 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. Le « Collège des architectes et concepteurs logiques (SOA) » prend en charge la validation et l'enrichissement de ce guide et des documents associés[1].

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

[1] Pour information sur le fonctionnement : mailto:college-logique@parxeme.org .

Licence

Conditions d'utilisation et de diffusion

Droits et devoirs

Ce document est protégé par une licence « Creative Commons » , résumée ci-dessous. Le terme « création » s'applique au document lui-même. L'auteur original est :

- Dominique VAUQUIER, pour le document ;
- l'association *Praxeme Institute*, pour l'ensemble de la méthodologie Praxeme.

Nous vous demandons de citer l'un et/ou l'autre, selon que vous extrayez une citation directe ou que vous vous référez aux principes généraux de la méthodologie Praxeme.

Cette page est également disponible dans les langues suivantes :
български Català Dansk Deutsch English English (CA) English (GB) Castellano Castellano (AR) Español (CL) Castellano (MX) Euskara Suomeksi français français (CA) Galego עברית hrvatski Magyar Italiano 日本語 한국어 Melayu Nederlands polski Português svenska slovenski jezik 简体中文 華語 (台灣)

creative commons

COMMONS DEED

Paternité - Partage des Conditions Initiales à l'Identique 2.0 France

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public
- de modifier cette création
- d'utiliser cette création à des fins commerciales

Selon les conditions suivantes :

Paternité. Vous devez citer le nom de l'auteur original.

Partage des Conditions Initiales à l'Identique. Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

- A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.
- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.

Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...)

Ceci est le Résumé Explicatif du Code Juridique (la version intégrale du contrat).

Avertissement

Sommaire

Table des figures

Exergue

« *La logique n'est pas une théorie, mais une image réfléchie du monde*. »

Ludwig Wittgenstein, *Tractatus logico-philosophicus*

Remerciements

Les procédés décrits ici ont été mis au point à partir d'une base théorique et dans la confrontation avec la réalité des projets. L'équipe du projet « Règlements »[2] de la SMABTP a été particulièrement sollicitée dans cette phase d'élaboration de la méthode et d'apprentissage collectif. Que ses membres en soient remerciés ainsi que Jean-Michel DETAVERNIER dont le leadership et la conviction ont rendu possible cette aventure.

Je me dois de signaler, également, la contribution indispensable de la Cellule Architecture Technique, dirigée par Pierre BONNET, d'Orchestra Networks. L'apport de ce dernier, basé sur son expertise SOA, a été essentiel. Il a permis de rendre la méthode plus réaliste et de l'arracher à la fascination que l'approche objet exerçait, au début, sur l'architecture logique.

[2] Projet de refonte de la gestion des sinistres à la SMABTP.

Introduction

Architecturer le SI pour servir les métiers de l'entreprise

L'objectif

Avec l'architecture orientée services, l'urbanisation des systèmes d'information a trouvé la technique de conception qui lui permet de passer des intentions aux actes et d'assurer la continuité entre les orientations stratégiques et le détail de la conception.

La re-conception d'un système d'information dans une approche orientée services est un investissement important. Il est motivé par la valeur apportée à l'entreprise, en la dotant d'un outil plus souple et capable d'absorber les évolutions. En l'absence de pensée architecturale, les systèmes tolèrent mal ces évolutions, tant organisationnelles que techniques. Le potentiel d'innovation est alors bloqué et la structure des coûts de la DSI trahit l'archaïsme des pratiques.

Objet et contenu du document

Le présent document, composant de la méthodologie Praxeme, s'adresse aux architectes logiques et aux concepteurs logiques, chargés de l'architecture logique et de la conception détaillée des services logiques. Il les guide pour élaborer et documenter complètement l'architecture de services. Il explique comment concevoir une architecture de services.

Il s'agit d'un guide *méthodologique* qui ambitionne de fonder et justifier de nouvelles pratiques. Le lecteur trouvera, ailleurs, les fiches méthode qui traduisent, en actes et en check-lists, la philosophie exposée ici.

Le guide aborde tous les sujets liés à l'aspect logique et les situe. D'autres documents traitent plus en détail, les thèmes :

- la discipline de l'urbanisation de SI ;
- le modèle logique des données (cf. PxM-41) ;
- le pseudo-langage utilisé pour la description des opérations et des services ;
- les conventions de nommage (à titre d'exemple, cf. OLQ-02x[3]) ;
- la conception de la strate « Présentation » et de son interaction avec la strate « Organisation » (voir p. 101) ;
- la négociation logique / technique (sous la forme d'une *check-list* ; cf. PxM-42) ;
- le processus à suivre (dynamique globale, cycle PRO2, etc. ; cf. PxM-03).

La lecture de ce document présuppose la connaissance de la méthodologie Praxeme, dans ses lignes directrices (cf. PxM-02, le Guide général). Les acteurs de la modélisation logique travaillent à partir des modèles établis pour les aspects sémantique et pragmatique. Ils n'ont pas, pour autant, besoin des compétences de modélisation propres à ces aspects. Il leur suffira de pouvoir lire dans le détail et comprendre les modèles amont.

Avertissement

Le terme « service » est compris, ici, dans un sens très précis : celui qu'il prend dans l'expression « *Service Oriented Architecture* » (architecture de services). Il s'agit d'un emprunt que fait la communauté informatique, au langage courant. Ce n'est donc pas un objet organisationnel, ni une action humaine, ni même un service que rend le système directement à son utilisateur (au sens de SLA[4]). « Service » et SOA réfèrent à la construction des systèmes informatiques. L'informaticien fait, du terme « service », un usage métaphorique. Praxeme précise encore l'usage de ce terme en l'assignant exclusivement à l'aspect logique, tel que défini à partir de la Topologie du Système Entreprise (voir plus loin, la définition p. 4).

[3] Les conventions de nommage ne sont pas dans le référentiel méthode (mode de *production*), mais sont attachées au *produit*. D'où l'identifiant OLQ, pour : « Opus – Logique ».

[4] SLA, *service level agreement*, est un moyen de spécifier les services attendus d'un système.

Qu'est-ce qu'un service logique ? – Une métaphore pour décrire le système informatique

Demander un service, c'est attendre que quelqu'un, le fournisseur, réalise une action qui produise un résultat intéressant pour le demandeur. Solliciter et fournir un service sont les deux termes de l'interaction. Le demandeur n'a pas à connaître la façon selon laquelle le fournisseur s'y prendra. Il ne s'intéresse qu'au résultat et aux conditions dans lesquelles il en bénéficiera.

SOA est une approche des systèmes informatiques qui s'est construite sur la base de cette métaphore. Pour respecter celle-ci, le « service » dénote l'action et son résultat. Or, certains discours à teneur technique ont fait glisser le terme qui désigne, aussi, le composant qui fournit le service. C'est le cas, surtout, dans la solution technique des *web services*. Un *web service* affiche plusieurs opérations. Praxeme respecte la métaphore et assigne le « service » à l'unité de l'opération, c'est-à-dire au grain le plus petit, à l'atome du système. Si on regarde le contenu d'un service, on atteint la ligne d'instruction, inutile et inutilisable d'un point de vue externe.

La caractéristique majeure de cette approche réside dans l'encapsulation (pour reprendre le terme de l'approche orientée objet). Dire que le service est le grain élémentaire du système, c'est exclure la donnée hors du champ de vision. En effet, pour obtenir une information, dans une architecture de services, il n'est d'autre moyen que de solliciter le service *ad hoc*. SOA, comme l'approche objet, interdit la manipulation à distance. Pas question que plusieurs programmes attaquent directement les mêmes supports de données : ils demandent les services pour accéder aux données dont ils ont besoin. Ces services, non seulement factorisent les manipulations, mais aussi ils garantissent que toutes les contraintes applicables seront respectées. Cette caractéristique change radicalement la façon de concevoir le logiciel. Elle augmente la qualité.

Nous disons « service *logique* » pour bien signifier que l'effort de conception et de structuration qui va tirer parti de cette métaphore se joue sur l'aspect logique. Le service n'est pas une solution technique, même s'il existe des solutions dédiées. Nous pouvons concevoir un système informatique à l'aide de cette approche, tout en utilisant des technologies classiques. L'application de la métaphore conduira à améliorer la structure du système. Tandis que l'architecture technique établit les conditions de faisabilité d'une architecture de services, l'architecture logique bâtit la structure et la conception logique trouve les services et détaille leurs spécifications.

L'usage du terme « service » soulève une interrogation : concerne-t-il l'acteur du système, « l'utilisateur » ? Il faut montrer, ici, une grande prudence. Le détournement du nom courant « service » risque de causer la confusion. On peut dire, bien sûr, que l'utilisateur attend, du système informatique, des services et on peut, d'ailleurs, en fixer le niveau de qualité (par des SLA[5]). Afficher la synthèse client, rechercher un produit répondant à certains critères, établir l'arrêté des comptes… autant de services que l'on peut attendre d'un système automatisé. Il y a de bonnes chances que l'architecture de services mette en regard de ces besoins des « services logiques » qui portent le même nom. Ce sont les opérations qui réalisent les services. Avec ces exemples, nous sommes à la périphérie du système informatique. Nous y constatons la coïncidence entre :

- d'un côté, la perception naturelle, externe – le service rendu par le système ;
- de l'autre, la construction interne, le « service logique » en tant qu'unité de construction du système, dans le style SOA.

Pourtant, cette coïncidence n'est pas essentielle dans l'approche SOA. L'essentiel du message réside dans la généralisation de la métaphore du « service » pour repenser le système informatique, en profondeur. Au-delà de ces services perçus par l'acteur, le système se recompose à base de services, dont beaucoup restent tapis dans ses profondeurs.

[5] SLA : *service level agreement*, un accord sur le service rendu à l'utilisateur d'un logiciel.

Les lignes directrices

Le guide s'inscrit dans les orientations de la méthodologie Praxeme. Ces orientations comprennent :

- la séparation des aspects et l'investissement sur les modèles amont, indépendants de la technologie, donc plus stables[6] ;
- l'effort de structuration du système au niveau logique ;
- le choix de l'architecture de services, pour mettre en forme le système et préparer sa réalisation que ce soit avec des technologies classiques ou plus récentes.

La motivation L'origine de ce guide réside dans la volonté très concrète d'armer les concepteurs avec des procédés précis, presque mécaniques, qui leur permettent de trouver de vrais services à valeur ajoutée et de structurer proprement, rigoureusement, le système informatique.

Les procédés se fondent sur le socle théorique de Praxeme, décrit dans le guide général (PxM-02). Ils ont été éprouvés, longuement mûris, sur de grands projets SOA[7]. Un coup d'œil sur la synthèse des règles de dérivation, p. 74 , convaincra du niveau de détail auquel est arrivé la méthode.

Les changements Les disciplines de l'architecture logique et de la conception logique sont complètement revisitées. L'urbanisation de SI et l'architecture technique sont des disciplines voisines et, en tant que voisines, subissent le contrecoup des changements de pratiques proposés par Praxeme.

Le style d'architecture L'architecture de services (SOA) est un style possible pour l'architecture logique d'un système informatique. Choisir le service logique comme unité de base, grain élémentaire du système, conditionne la structure et le détail du système que l'on veut réaliser. Ce choix laisse encore ouvert un large éventail de possibilités. Praxeme admet la diversité des styles applicables à l'aspect logique. Le présent document ne porte que sur le style SOA[8].

Dans le guide Le guide s'efforce de séparer les questions et actions de nature purement méthodologiques, d'un côté, et les options possibles pour une architecture de services, de l'autre. La méthode indique les questions qu'il faut se poser et les contraintes à respecter dans les travaux. En revanche, elle ne doit pas imposer les réponses quand celles-ci sont marquées par un contexte donné. Plusieurs réponses seront présentées, à titre indicatif et pour aider à l'application.

Cette distinction est essentielle :

- Ce qui relève de la méthodologie est l'ensemble des questions et des points à traiter, de portée universelle, à considérer quel que soit le contexte. L'application se veut la plus générale possible, étant admis le choix du style SOA.
- Ce qui dépend d'une cible technique particulière comprend les décisions issues d'une négociation logique / technique, liée à un contexte d'entreprise.

La représentation

Autant que possible, Praxeme s'appuie sur les standards. En matière de représentation, c'est le standard UML (*Unified Modeling Language*) qui s'impose. La version actuelle de ce guide se réfère à la version 1.4 d'UML. UML 2 apporte des nouveautés qui présentent un intérêt pour la modélisation logique. Leur assimilation demande un effort que nous n'avons pas fourni, à ce stade.

---

[6] Ce principe est celui fixé par le standard MDA (*Model Driven Architecture*), de l'OMG (*Object Management Group*).

[7] Citons, tout particulièrement, la SMABTP qui a largement contribué à l'élaboration de cet aspect de la méthode publique et qui l'a appliqué à grande échelle.

[8] En fait, plus que d'un style, nous devrions parler d'une « époque » SOA, comme il y a eu une « époque » fonctionnaliste ou une « époque » objet. À l'intérieur de l'époque SOA, nous identifierons des styles, en fonction des choix d'expression et de structuration qui guideront la conception logique.

Introduction (suite)

Les apports de Praxeme pour la conception SOA

L'unité « service »

La première décision forte de Praxeme en matière de SOA porte sur l'usage fait du terme « service ».

Tout d'abord, Praxeme donne un sens précis, technique, au terme « service » en l'assignant à l'aspect logique. L'architecte technique garantit les conditions de faisabilité d'une architecture de services, conditions qui reposent sur les technologies[9]. Une fois ces conditions établies, la conception d'un système dans un style SOA est une affaire de modélisation logique. Pour éviter toute confusion, nous utiliserons l'expression « service logique ».

Ensuite, Praxeme définit le service logique comme le grain élémentaire du système – le système étant vu sous son aspect logique. Un usage introduit un décalage par rapport à la perception du terme « service » dans certaines cultures techniques, notamment imprégnées de la technologie *web services*. En effet, un *web service* n'est pas un grain élémentaire : il contient des opérations, visibles de l'extérieur et demandées par les consommateurs. Dans la terminologie de Praxeme, le consommateur demande un service ; il ne voit rien d'autre, du système – rien de plus petit. En cela, nous sommes fidèle à la métaphore du service.

Le contexte de travail

La construction et la transformation du système entreprise exigent le concours de nombreuses expertises. La cohabitation de populations qui portent des expertises différentes ne va pas de soi. Praxeme répond à cette situation en établissant la cartographie des compétences et en respectant la pluralité des univers cognitifs. Un des objectifs de la clarification méthodologique est d'éviter la bouillie « technico-fonctionnelle » dont on badigeonne, trop souvent, les questions liées à la conception du système informatique. Praxeme ne connaît pas et refuse de reconnaître les architectures technico-fonctionnelles ou tout autre chimère du même acabit. Un thème donné est soit fonctionnel (dirigé alors vers l'aspect sémantique ou l'aspect pragmatique), soit logique, soit technique. Tolérer l'entre-deux, c'est ruiner toute chance de maîtriser la chaîne de production.

Certes, il arrive que l'on puisse hésiter quant à l'attribution de certains thèmes. Pour donner des exemples : la gestion des transactions, la gestion des incidents, la conversion des automates en mode procédural, etc. Pour dissiper ces hésitations, Praxeme prévoit un moment clef dans la démarche : la négociation logique / technique. Ce moment permet de distribuer les responsabilités entre les architectes logiques et techniques, à propos des thèmes aux frontières. Après quoi, chacun s'en retourne labourer son champ avec les outils qui relève de sa compétence.

La négociation logique / technique est un élément déterminant dans la démarche et, sur elle, repose en grande partie le succès des projets SOA. Une section lui est dédiée, p. 17.

Quatre préceptes pour guider la conception logique

Les apports de la méthodologie Praxeme pour la conception SOA peuvent se résumer en quatre préceptes, exposés page suivante.

Ces quatre préceptes – encapsulation, structuration, continuité et dérivation – donnent l'essentiel de la philosophie qui sous-tend les disciplines de l'architecture logique et de la conception logique. C'est aussi à cette source que s'abreuve la nouvelle urbanisation de SI. Ils inspirent le détail des procédés présentés dans ce guide et dans les documents associés.

[9] Signalons au passage que les technologies permettant de structurer le système informatique en SOA ne se limitent pas, d'ailleurs, aux nouvelles technologies ou aux solutions dévolues à SOA (SOAP, UDDI, etc.). L'aspect logique est indépendant des choix techniques. Sa description vaut pour différentes architectures techniques cibles.

Introduction (suite)

Les préceptes caractérisant l'architecture de services selon Praxeme

Le précepte d'encapsulation

Le plan des services masque le plan de données.

L'aspect logique se stratifie en deux plans. Seul le plan des services est visible : par construction, dans une SOA, on ne peut demander que des services. Les données sont protégées par ce plan. Les services logiques – plus généralement, les constituants logiques – ont besoin des données. Ce sont leurs ressources. Idéalement, chaque table, chaque donnée se place sous le contrôle exclusif d'un constituant logique.

Ce précepte de bon sens entraîne des conséquences sur la conception : l'architecture des données obéit à des contraintes topologiques ; elle s'ajuste – avec plus ou moins de bonheur – à l'architecture de services (voir p. 71).

Le précepte de structuration

L'architecture logique dispose les services en agrégats.

Le système se compose de milliers de services. Il va de soi qu'il est nécessaire d'y mettre de l'ordre. C'est l'objet de la discipline d'architecture logique. Son objectif est la maîtrise du système (limitation du couplage, économie du système, partage…). À cette fin, le procédé introduit des types d'agrégats logiques qui seront présentés dans le chapitre « Termes et représentations de l'aspect logique pour SOA ». Associées à ces termes, des contraintes topologiques s'imposent. L'architecture logique nécessite une technique de représentation : Praxeme recourt à la boîte à outil UML. L'architecte logique assume la responsabilité de décomposer le système en veillant au couplage et à la charge de communication entre les composants obtenus (appels et flux). La méthodologie le sensibilise à la question du critère de décomposition.

Le précepte de continuité

L'architecture logique prolonge les décisions d'urbanisation du SI.

Praxeme positionne les rôles d'urbaniste de SI, d'architecte logique et de concepteur logique. Tous ont à faire avec le système sous son aspect logique. Les architectes logiques et les concepteurs logiques travaillent essentiellement, si possible exclusivement, sur l'aspect logique. Les urbanistes, quant à eux, produisent la cible d'urbanisation que les architectes logiques prennent en entrée de leur travail. Ce qui les caractérise est leur rôle de passeur entre le métier, la direction général, le stratège, d'un côté, et l'informatique, de l'autre.

Le précepte de dérivation

Les services et les données se dérivent des modèles « amont ».

Une fois traitées les questions de technologies, l'approche SOA soulève la question : « comment trouver les services ? ». C'est là-dessus que le praticien attend la méthode. La réponse de Praxeme est massive : elle se fonde sur le standard MDA et réside dans la dérivation des modèles sémantique et pragmatique. Le concepteur logique applique à ces modèles des règles de dérivation qui produisent une grande partie des éléments du modèle logique.

Ce précepte repose sur le bon sens : pour bien faire quelque chose, il faut le faire en référence à une demande en amont, en préalable. Pour bien concevoir le système dans son aspect logique, il faut prendre en compte sa description dans ses aspects « métier ». Cette description ne peut pas se réduire à une expression vague, informelle : si tel était le cas, on ne pourrait rien dériver. Le point de départ de la conception logique réside dans des modèles rigoureux, exprimant le métier. Faire l'économie de ces modèles ou l'impasse sur leur qualité condamne à bricoler quand on en arrive à la conception du système informatique.

Introduction (suite)

Synoptique de la démarche

La structuration

La conception de l'architecture logique postule le principe de stratification : les composants logiques ne sont pas distribués indifféremment dans l'architecture, mais ils sont répartis en « strates[10] » et leur localisation impose des contraintes quant à leur communication. Les strates sont, en partant du centre du système : « Métier », « Organisation », « Présentation ».

Toutes les strates sont composées de la même façon, à base de « machines logiques », « ateliers logiques », etc. Les contraintes topologiques orientent la communication, de la périphérie vers le centre, c'est-à-dire des constituants de la strate « Présentation » vers ceux de la strate « Organisation », et de celle-ci vers la strate « Métier ». L'application d'autres contraintes structurelles aide à réduire le couplage et à maîtriser le système.

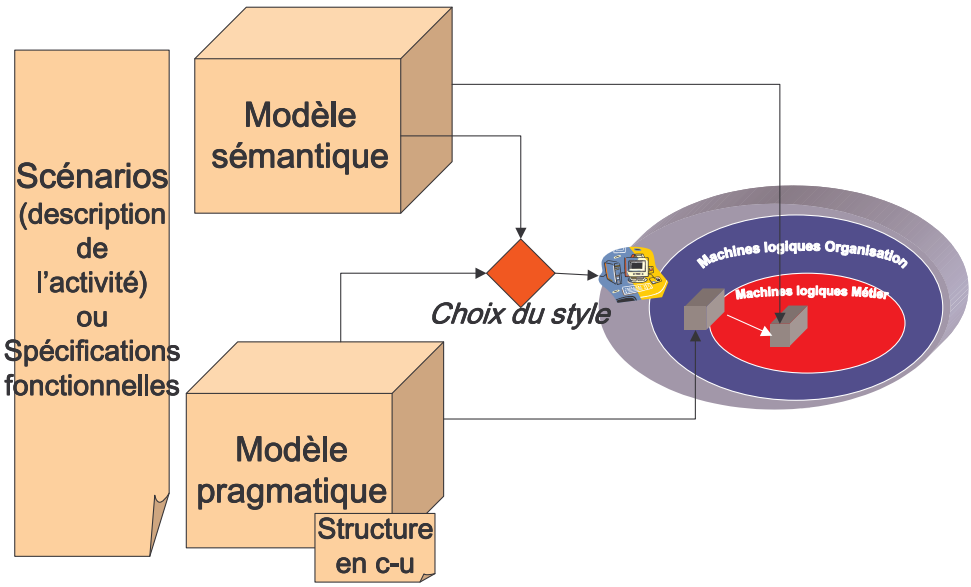
Les transformations

La chaîne d'activité respecte les dépendances entre les modèles et les décisions. En entrée, nous trouvons les éléments de pré-modélisation issus de l'élaboration stratégique (objectifs) ou de l'expression des besoins (exigences). Ce matériau reçoit une forme rigoureuse qui dissocie les fondamentaux du métier, d'un côté, et les choix d'organisation et habitudes de travail, de l'autre. Les modèles sémantique et pragmatique recueillent et formalisent cette connaissance.

En dérivant le modèle sémantique, le concepteur logique construit le noyau du système d'information, la strate dite « Métier » (en rouge sur la figure ci-dessous). Le modèle pragmatique se compose de la Vue de l'organisation (avec les processus) et de la Vue de l'utilisation (à base de cas d'utilisation). Les constituants logiques produits par la dérivation du modèle pragmatique peuplent la strate « Organisation ».

Ces deux modèles peuvent également être exploités pour concevoir la strate « Présentation » et le dialogue homme-machine.

Figure PxM-40_3. Les dépendances entre les produits



[10] Le terme « strate » a été choisi pour éviter la confusion avec les « couches » techniques.

Introduction (suite)

Synoptique de la démarche (suite)

La chaîne d'activité

Ce guide ne fournit pas de processus détaillé pour ordonner les activités liées à l'aspect sémantique. À titre d'exemple, la figure ci-dessous tire les conséquences des dépendances entre les produits pour en déduire la dynamique du projet.

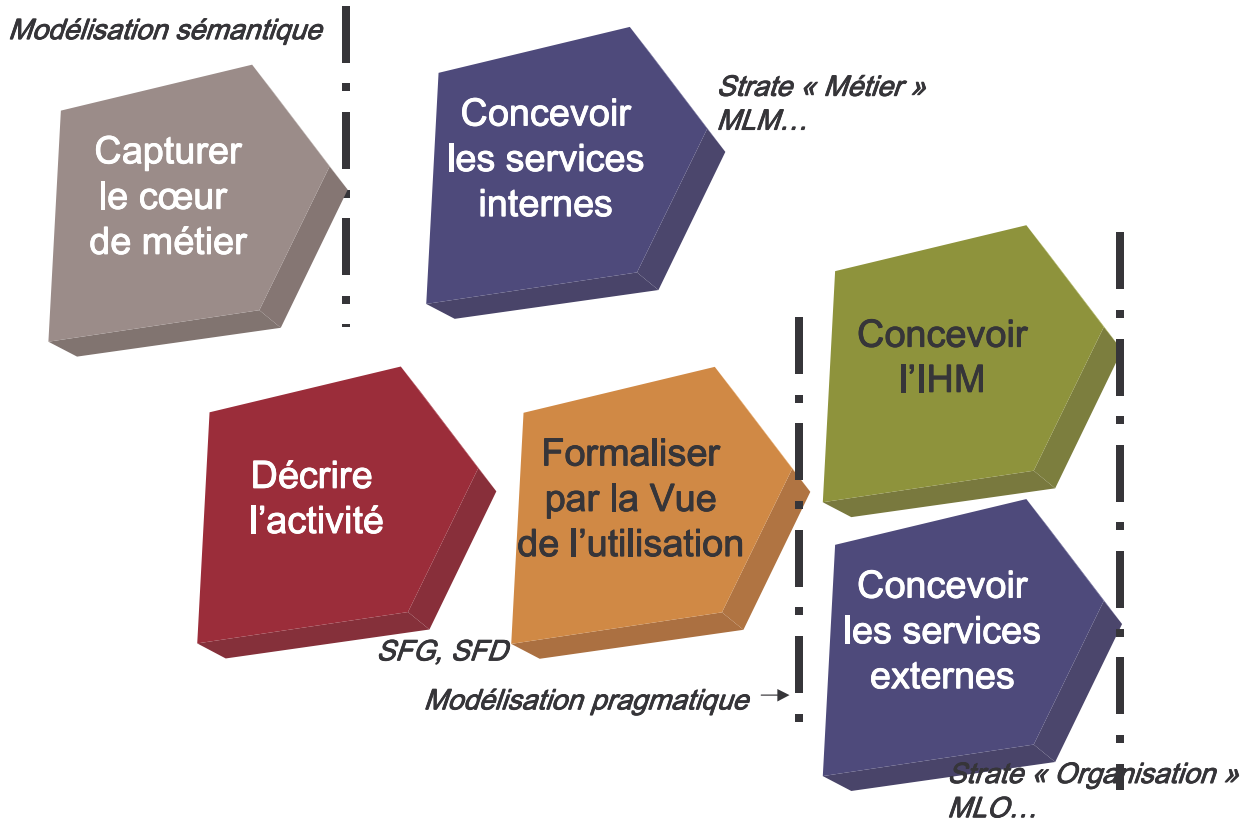
La démarche peut ménager un certain parallélisme entre l'axe sémantique qui aboutit à la strate « Métier » et l'axe pragmatique qui produit la strate « Organisation ».

Le processus peut se décrire plus finement si on considère, par exemple, que la conception des services suit plusieurs étapes :

1. grandes décisions de structuration, d'abord (c'est-à-dire : architecture logique) ;
2. identification des services et définition ;
3. description détaillée des services.

Il est possible d'identifier et de définir les services externes (ceux de la strate « Organisation ») sans connaître la structure interne du système. En revanche, la *description* de ces services, c'est-à-dire la conception de leur fonctionnement, exige de connaître les services internes (ceux de la strate « Métier »).

Figure PxM-40_4. Une partie de la chaîne d'activité (exemple de processus possible)



Orientations de l'architecture de services

Les motivations de l'architecture logique

Les objectifs de l'architecture logique

L'architecture logique est l'art de structurer le système d'information, pour le plier à des objectifs identifiés. Ces objectifs, liés à la stratégie de l'entreprise et à la vie du système, comprennent :

- L'économie : éviter la redondance et les complications inutiles.
- L'évolutivité : rendre le système assez flexible pour accueillir le plus naturellement possible les changements.
- La maîtrise du système : maintenir sa lisibilité et réduire les risques et les dépenses.

La séparation des aspects

Si nous investissons sur l'architecture *logique*, c'est que nous avons identifié un aspect nettement séparé des aspects amont (sémantique et pragmatique) ainsi que de l'aspect technique.

Le modèle logique doit donc se préserver des choix techniques. Les décisions liées aux technologies qui se glisseraient dans le modèle logique rendraient celui-ci moins pérenne. Au cas où ces décisions changeraient (ce qui ne manquera pas de se produire dans le futur), ce modèle logique entaché de technique ne servirait plus à rien. Il y aurait, alors, perte d'investissement. Au contraire, un modèle vraiment logique subsistera sur le long terme et soutiendra la démarche d'urbanisation du SI, sur la longue durée.

Néanmoins, l'architecte logique vérifie que les termes qu'il utilise pourront trouver une traduction naturelle dans les termes de la technologie. Cette vérification constitue un préalable à la conception logique. Elle fait partie de la négociation logique/technique.

La maîtrise de la complexité

La maîtrise du système s'obtient par sa documentation, par les activités d'administration et aussi, en grande partie, par la simplification de sa structure. Dans cet effort, la réduction du couplage est une préoccupation constante. Elle caractérise le métier de l'architecture logique, de tout temps.

L'architecture logique reprend les modèles amont – sémantique et pragmatique – lesquels ne montrent pas les mêmes préoccupations. Elle les remanie en cherchant à préserver, le plus possible, la structure et la logique du métier.

Dans l'état actuel des technologies, cet effort conduit à substituer une logique de contextualisation à la logique de propagation à l'œuvre dans le modèle sémantique.

Orientations de l'architecture de services (suite)

La réutilisation

Une motivation forte

La réutilisation est citée, souvent, parmi les motivations de l'architecture de services[11].

La démarche suggérée consiste à isoler des services réutilisables (comme le montre la figure PxM-40_5). La réutilisation repose, alors, sur des décisions prises *a posteriori* à partir d'une masse de services disposnibles.

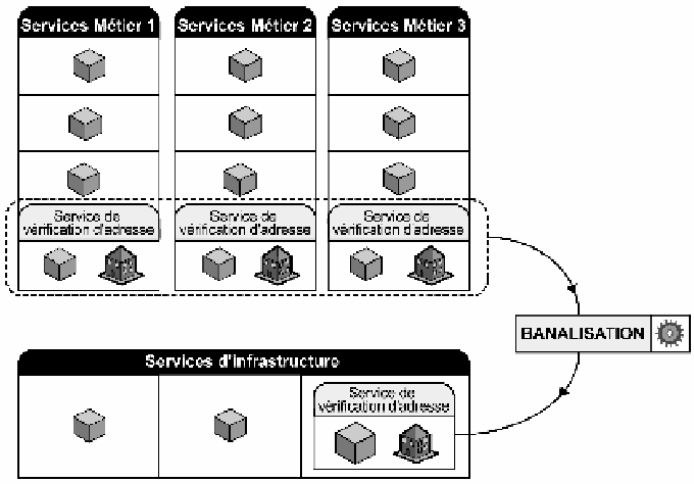


Figure PxM-40_5. « Le phénomène de banalisation des services métier », d'après Octo Technology, Livre blanc SOA.

L'approche a priori

Dans la méthodologie Praxeme, la démarche fonctionne à l'inverse. Elle assure une bonne réutilisabilité du système parce que celui-ci est d'abord structuré avec des exigences formelles et structurelles, appliquées aux aspects amont.

- D'une part, la modélisation sémantique et la conception des cas d'utilisation traquent la redondance, factorisent tout ce qui peut l'être, guidées par un principe d'économie : ne pas exprimer deux fois la même chose ! Ce faisant, elles isolent nécessairement des éléments partagés.
- D'autre part, le mouvement spontané de la modélisation sémantique, outillé par l'approche orientée objet, pousse à la généricité. Si la modélisation est bien menée, le modèle sémantique résultant a vocation à l'universel. Nous verrons plus loin que le cœur du système dérive directement du modèle sémantique. La dérivation engendre des services logiques qui conservent ce caractère universel : les services qui en sont issus sont, donc, largement réutilisable au sein du système comme au-dehors.

Ceci apparaîtra plus clairement et sera démontré dans le détail, dans la suite de ce guide.

Critique de l'approche a posteriori

La méthode proposée, ici, se place aux antipodes de la « banalisation » ou de la détection *a posteriori* de services réutilisables.

Nous ne croyons pas à cette approche pour les raisons suivantes :

1. D'abord, les services existants dont on parle ne se situent pas au niveau de la description logique mais sont des composants logiciels. Il est plus difficile de prendre des décisions fortes de conception, à ce niveau.
2. Ensuite, un service ne peut prétendre à la réutilisabilité que s'il a été conçu pour cela. On ne peut pas faire l'économie d'une conception *a priori*, motivée par la réutilisation.

L'approche a posteriori n'est certes pas inutile, mais elle se révèle insuffisante pour réformer en profondeur un système informatique.

[11] Voir, par exemple, la Livre blanc de Octo Technology, *Architecture Orientée Services, Une politique de l'interopérabilité*.

Orientations de l'architecture de services (suite)

Le choix d'un style d'architecture logique

Le style

L'important est de retenir l'idée de styles d'architecture. Il y a les grands styles (ou époques) : hier, fonctionnel ; aujourd'hui, service ; demain peut-être événement ou sémantique… Dans ces époques, il y a des variantes : par exemple, le style de l'architecture de services peut encore se positionner sur un éventail qui va de la propagation à l'orchestration…

Les styles se caractérisent par les catégories de représentation que l'on se donne dans l'aspect logique et, également, par les contraintes topologiques et les règles de structuration. Par exemple, on acceptera ou pas de mélanger des modes de communication : par appel de service et par événement. Autre exemple : on cachera certains types de constituants logiques et on interdira certaines interactions…

L'architecture de services

Il y a plusieurs façons d'élaborer une architecture logique. Le style de l'architecture se caractérise d'abord par le critère retenu pour décomposer le système. Par exemple, l'architecture fonctionnelle peut être définie comme une architecture logique construite en termes de fonctions. Elle accorde le primat à l'activité, au faire. Les limites de cette approche se sont révélées avec le temps ; elles résident surtout dans un fort taux de redondance[12].

L'architecture de services est une architecture logique qui choisit de s'exprimer avec l'unité de base du service logique. Elle renvoie à la métaphore des rapport entre clients et fournisseurs et à son corollaire : le contrat de service.

Ce style d'architecture repose sur des principes issus de l'approche client/serveur et de l'approche orientée objet :

- Le **principe d'encapsulation** stipule que le service n'est connu par ses demandeurs qu'à travers son contrat, tandis que restent masqués la solution organique et le support des données – plus généralement, les ressources.
- Le **principe de coopération** postule qu'un processus ou un traitement complexe s'obtient par le concours de plusieurs services, lesquels s'appellent les uns les autres par délégation.

Ce que devient le modèle sémantique

La méthodologie Praxeme se distingue par son insistance sur la modélisation sémantique, chargée d'isoler et de formuler les fondamentaux du métier, abstraction faite des contingences organisationnelles et techniques. Lors du passage à l'aspect logique, la manière de prendre en charge le modèle sémantique revêt donc une importance cruciale. Elle caractérise, elle aussi, le style de l'architecture et la physionomie du système résultant. Elle détermine grandement la façon selon laquelle les concepteurs logiques vont travailler.

De ce point de vue, celui de la prise en compte du modèle sémantique, on peut recenser les manœuvres :

- conservation : pas de transformation de structure (juste quelques compléments nécessaires pour implantation en "*socket*") ;
- mécanisation : passage des classes – savoir, concept – aux machines et services logiques (par dérivation) ;
- incorporation : les classes sémantiques sont conservées telles quelles mais « enrobées » dans un composant (*distributed component* – cf. Peter Herzum –, équivalent de notre atelier logique) ;

[12] L'architecture fonctionnelle utilise, comme critère de décomposition, la fonction. Elle prolonge, donc, assez naturellement, la modélisation pragmatique : l'approche du système par l'activité. Cette manière a montré ses effets pervers : redondance, difficulté à réutiliser, rigidité structurelle, dépendance de la structure informatique à l'égard de l'organisation.

- agglomération : plusieurs classes rassemblées dans un constituant logique (*business component* – pour Perter Herzum –, équivalent de la fabrique logique) avec des interfaces servant de relais vers les opérations.

Les schémas ci-dessous illustrent les transformations appliquées au modèle sémantique. Dans cette illustration, le modèle sémantique est vu, essentiellement, comme un réseau de classes. C'est effectivement l'aspect structurel qui nous occupe ici ; cela n'enlève rien à la substance du modèle sémantique avec, notamment, les comportements et les contraintes.

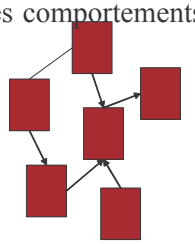


Figure PxM-40_6. Schéma du modèle sémantique de départ

La conservation Le modèle logique reprend trait pour trait le modèle sémantique. Les changements sont à la marge : les classes issues du modèle sémantique sont plongées dans un milieu qui prépare leur animation en tant que système logiciel. Ce milieu se compose de différents dispositifs.

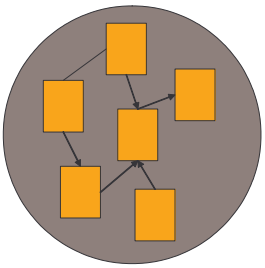


Figure PxM-40_7. Schéma de principe pour la conservation

La mécanisation Le modèle logique opte pour des moyens d'expression différents de ceux de la modélisation sémantique. La raison réside dans la nécessité de se rapprocher des possibilités technologiques, actuellement à notre disposition. Dans cette approche, la classe sémantique n'est pas transportée dans le modèle logique. On en dérive un constituant logique qui sera appelé, dans la terminologie Praxeme, une machine logique. La machine reprend la sémantique de la classe, mais en lui imposant des contraintes et des réductions propres à l'aspect logique.

La mécanisation introduit des niveaux intermédiaires pour ranger les machines : les agrégats logiques. Les deux modèles, sémantique et logique, ne se mélangent pas. Ils ne parlent pas le même langage et ne répondent pas au même besoin.

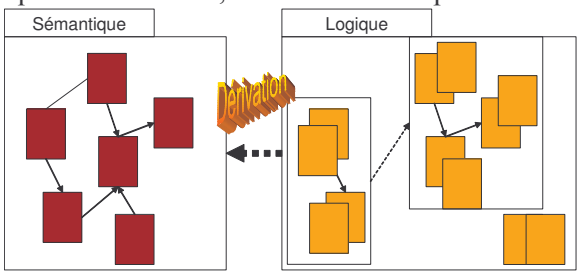


Figure PxM-40_8. Schéma de principe pour la mécanisation

L'incorporation Une autre solution consiste à reprendre telle quelle la classe sémantique et à l'embarquer dans un composant logique qui va l'enrober pour mieux l'insérer dans l'architecture. Par exemple, ce composant – que nous pouvons aussi nommer « machine logique » – se charge d'assurer la gestion des incidents ou la disparité des sources d'information, tous deux sujets qui n'ont pas été traités lors de la modélisation sémantique.

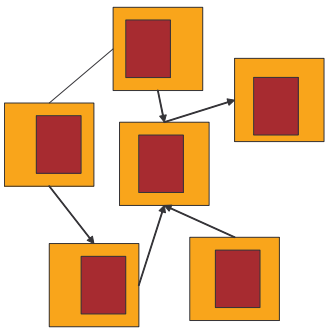


Figure PxM-40_9. Schéma de principe pour l'incorporation

L'agglomération Enfin, et de façon plus répandue, on peut bâtir le système vu sous son aspect logique, à base de grands blocs obtenus en agrégeant des classes. Ces blocs (*business components*, dans la terminologie de Peter Herzum) isole une partie des classes sémantiques et présentent, à travers leurs interfaces, une sélection des comportements de celles-ci. Ces comportements sélectionnés à partir de la sémantique constituent les services.

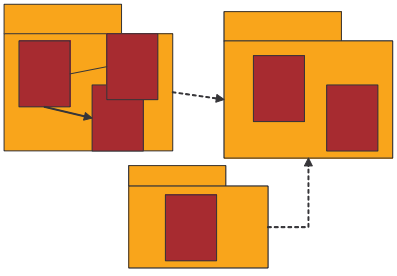


Figure PxM-40_10. Schéma de principe pour l'agglomération

Ces grands types de prise en compte du modèle sémantique par le modèle logique serviront de points de référence pour élaborer les règles de dérivation, dans la suite de ce guide.

Orientations de l'architecture de services (suite)

Le réseau de communication entre les constituants

Le style, plus précisément

Ceci étant posé, il nous faut encore préciser le genre de réseau de communication que dessinent les échanges à l'intérieur du système, toujours d'un point de vue logique. Nous distinguons trois façons de structurer l'architecture de services, rangées sur une échelle de 0 à 1. Ces valeurs indiquent le niveau de transformation imposée au modèle amont, c'est-à-dire à la logique « métier ».

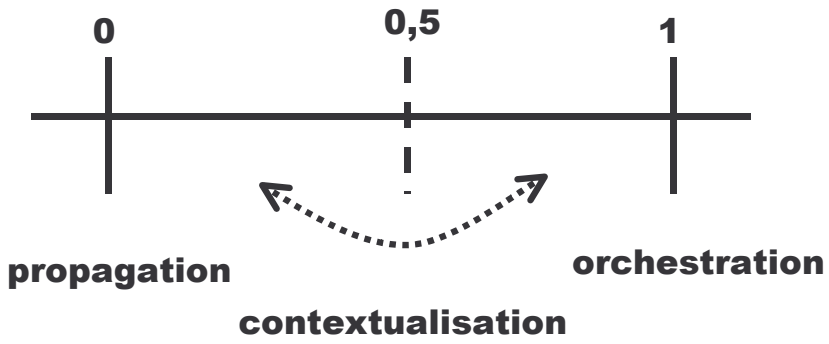
1. Position 0 : la propagation.
2. Position 1 : l'orchestration.
3. Position intermédiaire : la contextualisation.

Ces styles sont discutés ci-dessous. D'autres styles existent, notamment ceux fondés sur un couplage par « document » ou par événement, ces modes de couplage étant très faibles. Le style change du tout au tout si la négociation logique/technique introduit l'idée d'une machine virtuelle, assumant les fonctions techniques sans qu'il soit besoin de retoucher la logique.

Les trois positions

La figure ci-dessous schématise les trois styles candidats, en les repérant sur une échelle abstraite. Cette échelle mesure l'impact de la transformation infligée au modèle amont : plus on se rapproche de la valeur zéro, mieux est conservée la logique « métier ».

Figure PxM-40_11. Les trois positions repérées par l'indice de transformation

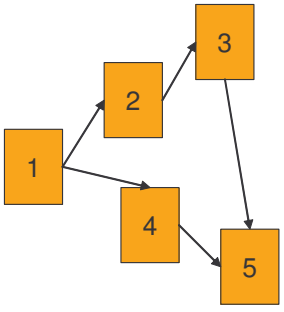


La propagation

Le modèle sémantique exprime la logique métier sur le mode de la propagation : l'information ou l'action sont demandées à un objet au moment précis où on en a besoin. C'est une conséquence directe de l'approche objet retenue dans le procédé de modélisation sémantique (cf. PxM-10). Cela tient aussi de la restitution assez naturelle du réel.

La transformation minimale du sémantique en logique conserverait ce mode de communication. L'avantage serait de bâtir un système qui adhérerait fortement à la logique métier, sans trop l'altérer ou l'appauvrir. L'inconvénient, le prix à payer, serait le niveau élevé de couplage et la fréquence des échanges lors de l'exécution.

Figure PxM-40_12. Réseau de communication dans le cas de la propagation



L'orchestration

Le mode de l'orchestration remplace l'appel direct des services par des appels médiatisés : il introduit des composants d'un nouveau type, chargés d'assembler les services, de telle manière que ceux-ci n'aient plus à se connaître.

Ce mode de communication permet de réduire le couplage entre les composants au cœur du système. Il se paye par l'introduction d'une couche supplémentaire, du moins d'un nouveau type de composant. Un autre inconvénient, plus grave, réside dans l'éclatement imposé à la logique métier. Ce mode inflige, donc, une transformation importante aux modèles amont.

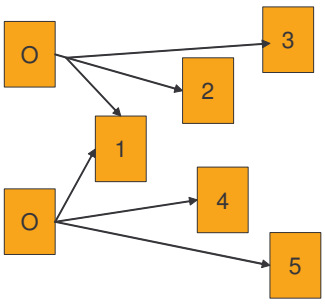


Figure PxM-40_13. Réseau de communication dans le cas de l'orchestration

À l'extrême – comme représenté sur la figure ci-contre –, les constituants de base n'entretiennent plus aucunes relations, ils deviennent complètement aveugles et sont entièrement sous la coupe des orchestrateurs. Ce mode est exagéré : l'argument en sa faveur est l'autonomie absolue des constituants, mais le prix à payer est très élevé. Ce type de réseau de communication pulvérise la logique fonctionnelle. L'intelligence des classes sémantiques est réduite à rien et reportée vers les orchestrateurs.

Où se situer ?

La propagation, position '0' de l'échelle d'appréciation des réseaux de communication, ne peut pas être retenue à l'échelle du système, du moins dans l'état actuel des technologies. Sur l'autre extrémité de l'échelle, en position '1', l'orchestration pure défigure trop la logique « métier ». On ressent, alors, le besoin d'une heuristique pour se positionner entre ces deux extrêmes, un guide pour décider dans quels cas – localement – on choisit l'un ou l'autre.

La contextualisation

La position intermédiaire est tenue par le mode de la contextualisation. Dans cette option, l'élaboration de l'architecture logique procède en deux temps :

1. D'abord, **structuration du noyau** : le cœur du système se structure par dérivation du modèle sémantique (strate « Métier »).
2. Ensuite, **optimisation** : en étudiant les contextes d'utilisation (décrits par les cas d'utilisation dont dérivent les constituants de la strate « Organisation »), on établit le bilan de l'information recueillie au moment d'appeler les services. On passe l'information utile au service, en augmentant la liste de ses paramètres. Ceci permet de réduire les appels à l'intérieur de ce service.

Ce mode de communication permet, donc, de réduire le couplage dans le noyau du système, sans introduire une nouvelle couche ni un type artificiel de service. Il restitue, aux services externes (strate « Organisation »), leur fonction d'assemblage des services internes (strate « Métier »). Le réseau de communication résultant de cette approche se situe entre les deux extrêmes de la propagation et de l'orchestration. Il teint l'équilibre entre un couplage excessif et une dégradation de la logique fonctionnelle.

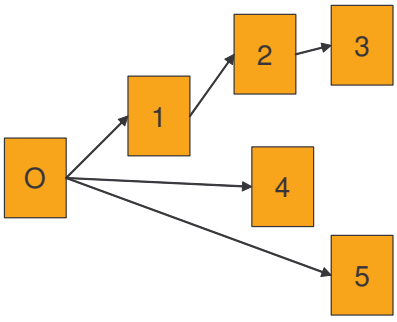


Figure PxM-40_14. Réseau de communication dans le cas de la contextualisation

L'évaluation de la structure

L'architecte logique assume la responsabilité de la structure établie dans l'aspect logique et de ses conséquences sur le système informatique. Pour évaluer l'architecture, il trouve, dans la métrologie du logiciel, des indicateurs simples qui donneront une juste idée de la qualité structurelle et fonctionnelle du système. Notamment, la profondeur et la largeur des réseaux de communication constituent de bons indices. Leur calcul est rendu possible grâce aux techniques de représentation que mobilise la discipline d'architecture logique. Ces techniques sont décrites plus loin.

Orientations de l'architecture de services (suite)

Les hypothèses concernant la technologie

Préliminaires

L'architecture de services repose sur une hypothèse de départ, la capacité de l'architecture technique à :

- traduire, en termes logiciels, tous les éléments du modèle logique ;
- assurer la communication au sein du système, sans imposer une restructuration ou déstructuration de l'architecture logique.

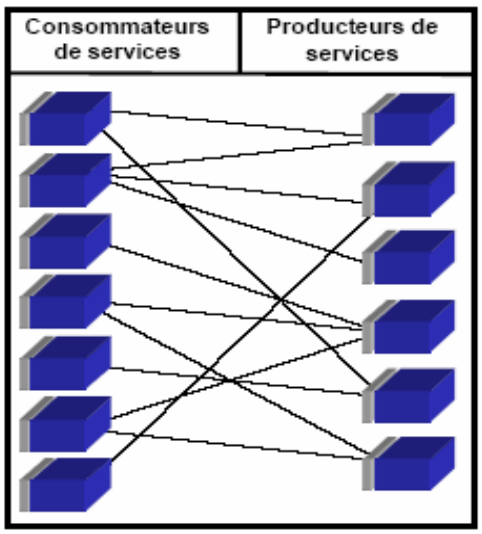
Il peut arriver qu'une correspondance rigoureuse soit impossible entre les catégories de représentation logique, d'un côté, et les éléments de solutions techniques, de l'autre. Dans ce cas, on cherche à réduire au minimum et à moindres frais les transformations nécessaires. On essaye, d'abord, de les introduire au niveau logiciel et on ne retouche l'architecture logique, qu'en dernier ressort. En effet, le modèle logique est supposé indépendant des choix de technologies, ceci afin de le conserver longtemps. Cette préoccupation répond à des motivations économiques. De plus, un même modèle logique peut déboucher, simultanément, sur plusieurs architectures techniques – notamment dans le cas d'une fédération de système. L'architecte logique veille donc à éviter le reflux de préoccupations techniques vers le modèle logique. Pour autant, il doit échanger, à certains moments, avec la partie technique, afin de s'assurer de la convertibilité du modèle logique dans les termes logiciels. Ce point crucial dans la démarche fait l'objet de cette section et de la suivante.

L'intermédiation

L'école SOA véhicule l'idée d'un dispositif d'intermédiation, qu'elle reprend d'une longue tradition. Ce dispositif, purement technique, conditionne la possibilité même de l'architecture de services. La comparaison des schémas ci-dessous illustre l'apport du « tiers d'intermédiation » (tiré de *SOA et urbanisme*, Unilog Management).

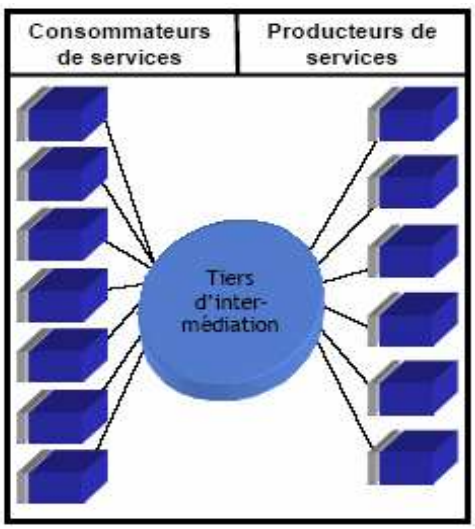
▲**Modèle sans**

Figure PxM-40_15. « Reproduction du modèle spaghetti dans l'utilisation des Services Web »



▲**Modèle avec**

Figure PxM-40_16. « Insertion d'un tiers entre producteurs et consommateurs »



Commentaire Le schéma peut être compris de deux façons, selon que les cubes représentent :

- des composants physiques ;
- des constituants logiques.

Sur le plan physique (logiciels déployés), on cherche bien sûr à éviter le plat de spaghetti, pour reprendre l'expression lancée par le Gartner Group. Sur le plan logique, en revanche, les relations peuvent se révéler irréductibles. L'intérêt de l'intermédiation est alors de transposer un schéma logique tel que celui de gauche, en une solution physique représentée à droite. Son rôle n'est pas de supprimer le couplage logique, car les consommateurs appellent des services et doivent bien les connaître pour pouvoir les appeler ! Mais, ils les connaissent logiquement : le dispositif masque la localisation physique des composants.

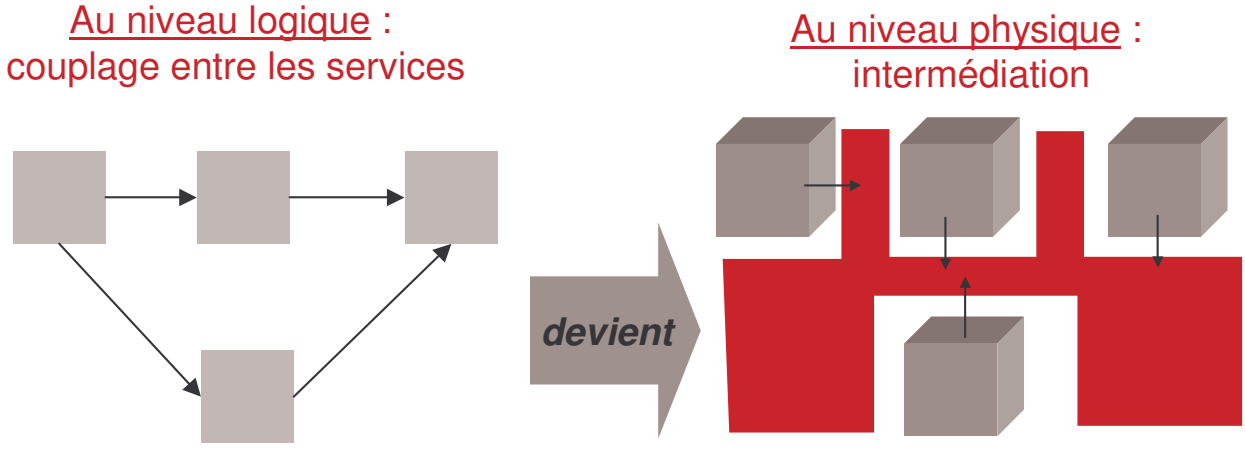
Le passage du logique au physique

Reposant sur cette hypothèse, la chaîne d'activité prévoit la dérivation du modèle logique de la façon suivante :

- À chaque constituant logique (du service à l'atelier) correspond un composant logiciel. Il peut arriver que plusieurs composants logiciels réalisent le même composant logique. Ceci se produit quand le système se déploie dans des architectures techniques différentes.
- L'architecture technique peut, éventuellement, conduire à ajouter des composants logiciels de nature purement technique, qui ne correspondent donc à aucun constituant logique.
- Dans la description logique, les constituants communiquent entre eux uniquement par appel de services et en respectant les contraintes topologiques.
- Au niveau physique, les composants logiciels, localisés sur des machines informatiques, ne se connaissent pas les uns les autres. Cette restriction offre la liberté de déplacer les composants logiciels sur l'architecture physique, sans bouleversement pour l'exécution.
- Au niveau physique, les composants logiciels accèdent à des services en utilisant leur nom logique. Le dispositif d'intermédiation se charge de localiser le composant logiciel à partir du nom logique et peut, également, convertir ou concilier les données.

Dans la figure ci-dessous, les composants logiques apparaissent sous la forme de carrés. Ils communiquent par des appels de services. Les cubes représentent les composants logiciels. Ils ne sont pas reliés directement les uns aux autres, du moins pas physiquement. Un dispositif technique d'intermédiation assure la communication au niveau physique.

Figure PxM-40_17. Le passage du logique au physique, avec un dispositif d'intermédiation



Une solution dégradée

En l'absence de dispositif d'intermédiation, il est nécessaire de développer des composants spécifiques qui vont réaliser les couplages. Ces *proxies* appartiennent à l'aspect logiciel et correspondent aux appels logiques. Cette solution ne doit pas refouler vers l'aspect logique, ni amener à déstructurer l'architecture logique. Sans offrir le confort d'une solution d'intermédiation, elle rend le système informatique plus économique et maîtrisable, et préserve les qualités attendues d'une architecture de services.

Orientations de l'architecture de services (suite)

La négociation logique / technique dans le processus

Le principe d'indépendance

La Topologie du Système Entreprise exclut toute relation entre l'aspect logique et l'aspect technique. De ce fait, elle entérine le principe d'indépendance logique qui pose la possibilité de décrire le système informatique en des termes indépendants de la technologie. Des considérations économiques et de gouvernance motivent ce principe : le niveau de détail de la spécification logique des services exige un investissement qui n'est pas négligeable. Pour assurer la maîtrise du système, il faut bien, de toute façon, en donner une description exacte. Si cette description se donne en termes techniques, logiciels, elle durera ce que durent les techniques ! La décision rationnelle consiste à investir sur la description logique, de laquelle on pourra déduire des solutions informatiques en appliquant des choix techniques différents. Les occasions de dérivation se produiront pour le même système, dans le temps, ou, simultanément, pour plusieurs systèmes ou sous-systèmes qui cohabitent.

La nécessité d'un échange

Cette indépendance, néanmoins, est relative. La pratique démontre que cette description exacte du système en termes logiques est possible et souhaitable. Il n'en reste pas moins que le modèle logique doit se traduire en logiciel. Les pages précédentes ont montré la grande diversité des choix d'expression laissés au concepteur logique. Une imagination débridée pourrait l'égarer au point que le modèle logique ne puisse que difficilement se convertir en logiciel.

Il est, donc, indispensable de s'assurer que les termes logiques trouveront une traduction aisée dans des termes logiciels. Cette vérification doit intervenir en préalable à l'investissement lourd sur la modélisation logique. Le moment pour exercer ce contrôle est la « négociation logique / technique ». Sous cette appellation, Praxeme inscrit, dans le processus, la nécessité d'un échange entre l'architecte logique et l'architecte technique. Cette précaution évite de produire un modèle logique, incompatible avec les possibilités techniques de l'entreprise.

Le partage des rôles

Lors de ce moment critique de la négociation logique / technique, deux profils interviennent : l'architecte logique et l'architecte technique. Sous ces titres génériques, se cachent plusieurs expertises. Disons que les deux parties, logique et technique, se rencontrent. Le but commun est d'assurer les conditions de convertibilité du modèle logique en logiciel.

La technologie fournit les conditions de faisabilité d'une architecture de services, mais, en aucun cas, la compétence technique ne permet d'établir la structure complète du système[13].

Le jeu de rôles Les architectes logiques et techniques se rencontrent ; leurs échanges s'appuient sur la liste canonique qui recense les thèmes principaux à évoquer lors de la négociation logique/technique (voir plus loin). Le jeu de rôles est le suivant :

1. L'architecte logique explique ce qu'il met derrière l'expression.
2. L'architecte technique indique s'il a une solution ou s'il pourra élaborer un dispositif.
3. Le cas échéant, ils précisent les éventuelles exigences qui pèsent, en retour, sur l'expression logique.
4. Si l'architecte technique ne propose pas de solution, ni du marché, ni spécifique, alors le thème reste sous la responsabilité de la partie logique. Cette dernière devra concevoir un dispositif logique approprié[14].

[13] Sur l'aspect technique, voir le guide méthodologique PxM-50. La partie technique a bien à faire avec des questions de structure : couches techniques, limites de capacité physique imposant des contraintes à la conception (niveaux de propagation dans la communication, niveaux d'imbrication dans les flux, etc.).

Le contenu de la négociation

La négociation logique / technique produit deux types de résultats :

- d'une part, la certitude d'une correspondance entre les termes logiques et les catégories du logiciel[15] ;
- d'autre part, la distribution de thèmes précis, entre les deux parties[16].

La correspondance

Pour le premier point, l'architecte logique établit une liste précise des catégories de représentation qui seront utilisées dans le modèle logique. Le chapitre suivant, sur les termes de la modélisation logique, permet d'alimenter cette liste. Le dossier d'architecture technique la reprendra sous la forme d'un tableau et mettra, en regard de chaque catégorie, la solution technique qui convient. Ces décisions étant prises, il restera à les exposer dans le manuel de développement et, si possible, à les automatiser par un profil UML assurant la dérivation du logique vers le logiciel.

Au moment de la négociation logique/technique, il ne s'agit d'arriver à cette correspondance mais seulement de vérifier la compréhension mutuelle de la chaîne d'activité et l'adhésion de la partie technique à la vision logique. Dans cette discussion, un élément important est l'unité de déploiement. Celle-ci appartient au vocabulaire de l'aspect physique. Il est, néanmoins, utile, lors de la négociation logique/technique, de réfléchir à la question : à quel constituant logique (machine logique, atelier, logique…) correspondra l'unité de déploiement du système ?

Les thèmes

Pour plusieurs thèmes essentiels au fonctionnement des systèmes informatiques, la question se pose de savoir qui est le mieux à même de les traiter, entre la partie logique et la partie technique. Évidemment, cette question doit recevoir sa réponse avant que s'engage l'effort de conception logique détaillée.

L'automate à états fournit un bon exemple. L'architecture technique apporte-t-elle une solution simple pour faire fonctionner les automates ou pour les transformer dans le logiciel ? Si la réponse est positive, alors on laisse le concepteur logique recourir à cet outil de modélisation puissant. Même dans cette hypothèse, il peut être nécessaire de brider un peu la notation, extraordinairement riche sur ce point. Notamment, la négociation portera sur le type d'automate mis en œuvre : protocolaire ou comportemental[17].

Si la réponse est négative, c'est-à-dire si l'architecte technique ne veut pas entendre parler des automates, alors deux options se présentent :

- On exclut l'automate du modèle logique. Le concepteur devra transformer les automates des modèles amont, dans des termes logiques plus classiques : services, pré et post conditions. C'est faisable mais lourd et décevant.
- On maintient, malgré tout, l'automate parmi les moyens d'expression de la modélisation logique. La transformation devra se faire sur l'aspect logiciel. Il faudra, alors, expliquer aux développeurs le moyen de transformer les automates du modèle logique.

Cet exemple montre que, sur un point précis, la réponse issue de la négociation logique/technique aura des conséquences directes sur la façon de faire le modèle logique.

---

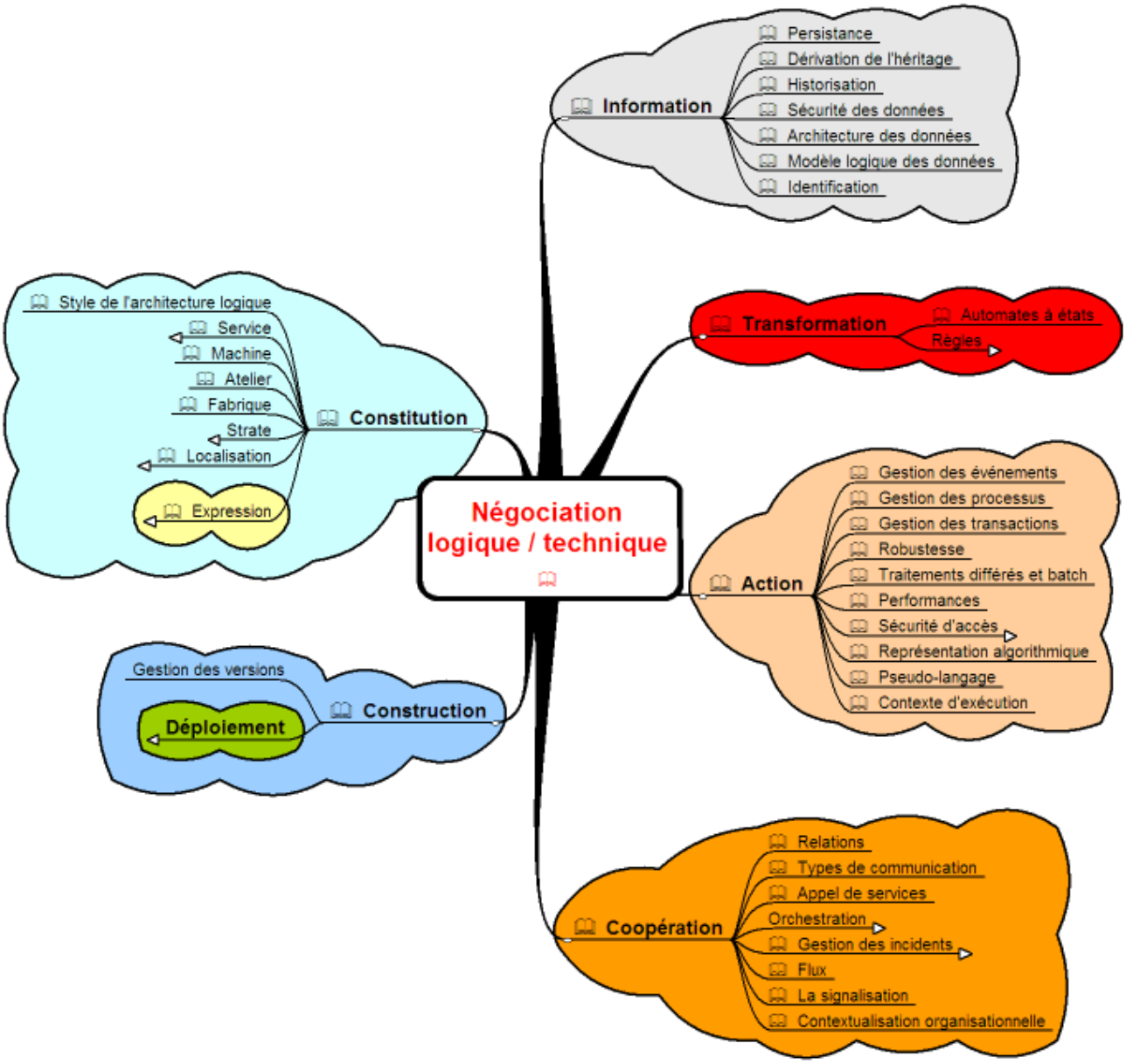
[14] Un exemple d'un tel dispositif est le traitement des codifications, bâti de toutes pièces comme une solution logique (cf. PxM-41).

[15] Par exemple, on trouvera des propositions telles que : « les machines logiques de la strate « Organisation » deviennent des composants EJB ».

[16] Par exemple, en ce qui concerne la gestion des transactions, le modèle logique désignera les services transactionnels par le moyen d'une annotation, à charge pour le framework technique d'interpréter cette information et de gérer les transactions.

[17] Ces notions sont définies dans le standard UML, version 2.

Figure PxM-40_18. Recensement des points à traiter dans la négociation logique/technique (liste simplifiée)



La liste canonique

La thématique abordée lors de la négociation logique/technique constitue un point de méthode crucial, dans le déroulement des projets SOA. Le document « PxM-42 » établit la liste canonique des questions à aborder lors de la négociation logique/technique. Cette liste, qui capitalise les expériences de projets SOA, constitue un guide essentiel dans cet exercice.

La structure se justifie de la façon suivante :

1. Information-Transformation-Action (ITA) fournit les catégories fondamentales pour l'approche du réel, à l'œuvre déjà dans la modélisation sémantique comme dans toute approche de modélisation complète. Ces trois pôles aimantent la triple approche de modélisation : structurelle (information, être), fonctionnelle (action, faire), contractuelle (transformation, devenir).Cette grille de lecture qui permet de conquérir le réel, repose sur des catégories de représentation précises. Cet outillage intellectuel doit avoir sa contrepartie dans le système informatique. C'est ce que vérifie la négociation logique/technique. De cet outillage, elle ne laissera au modélisateur logique que les instruments qui pourront être repris par des dispositifs techniques.

2. La coopération entre les objets ou les activités est implicite dans les aspects sémantique et pragmatique. Elle doit faire l'objet d'une réflexion sur l'aspect logique et d'un dispositif technique. Le principe de coopération est

à la base de l'approche objet pratiquée dans la modélisation sémantique. Sa transposition directe dans le système informatique ne va pas de soi. La technologie introduit des contraintes et des limites que le modèle logique doit prendre en compte. Le comportement du système en dépend largement. Ceci justifie une discussion entre l'architecte logique et l'architecte technique. Parmi les points abordés, on trouve le degré de coopération à établir sur l'échelle qui va de la pure propagation (approche objet) à la totale orchestration.

3. Le vocabulaire employé dans le modèle logique et les principes de structuration détermine l'architecture logique. Ils doivent faire l'objet d'une revue par l'architecte technique. C'est l'objet du chapitre « constitution ». Ce chapitre passe en revue les types de constituants logiques, c'est-à-dire les termes de la modélisation logiques. L'architecte technique se familiarise avec cette terminologie et admet l'intention de structuration qui aiguillonne l'architecture logique. Sa connaissance de la technologie informatique permet d'évaluer certaines orientations fortes de l'architecture technique. Par exemple, la localisation des descriptions de flux dépend de ce qui est admis dans l'architecture physique. Une telle décision conditionne le travail des concepteurs logiques. Elle doit être prise, si possible, lors de la négociation logique/technique.
4. Le chapitre « construction » s'intéresse à la chaîne d'activité complète. Il examine le processus, au-delà de la modélisation logique. Le processus de transformation du système – des objectifs au déploiement – peut imposer des contraintes que l'architecture et/ou la conception logiques doivent prendre en compte. Figurent, dans ce chapitre, notamment les thèmes de la traçabilité, de la gestion des versions et du déploiement. Encore une fois, il ne s'agit pas, lors de la négociation logique/technique de trouver des réponses à toutes ces questions, mais uniquement d'évaluer leur impact à prendre en compte dans le modèle logique, si besoin.

Le document référencé « PxM-42 » détaille tous ces chapitres.

Orientations de l'architecture de services (suite)

Les décisions d'architecture logique

La stratification

La règle de stratification impose de répartir les services selon des strates bien identifiées et séparées par l'action de contraintes topologiques.

Les strates s'étagent, du cœur du système à la périphérie, du plus stable au plus volatil. La communication va de l'extérieur vers l'intérieur. Elle est à sens unique. Ces règles découlent du style choisi pour l'architecture logique ; elles sont intimement liées à l'approche orientée services.

La strate « Métier »

La strate « Métier » reprend le « cœur de métier », pour autant qu'il puisse s'automatiser. Ce « cœur de métier » comprend la connaissance, les concepts, les objets du métier ou du domaine d'application, abstraction faite des façons de travailler et, bien sûr, des choix organisationnels et techniques. La strate « Métier » reprend cette substance à laquelle elle donne une forme activable. On y trouve les services internes, à fort contenu sémantique. Ce sont des services à valeur ajoutée, qui véhiculent les fondamentaux du métier. Ils manipulent et protègent l'information fondamentale de l'entreprise.

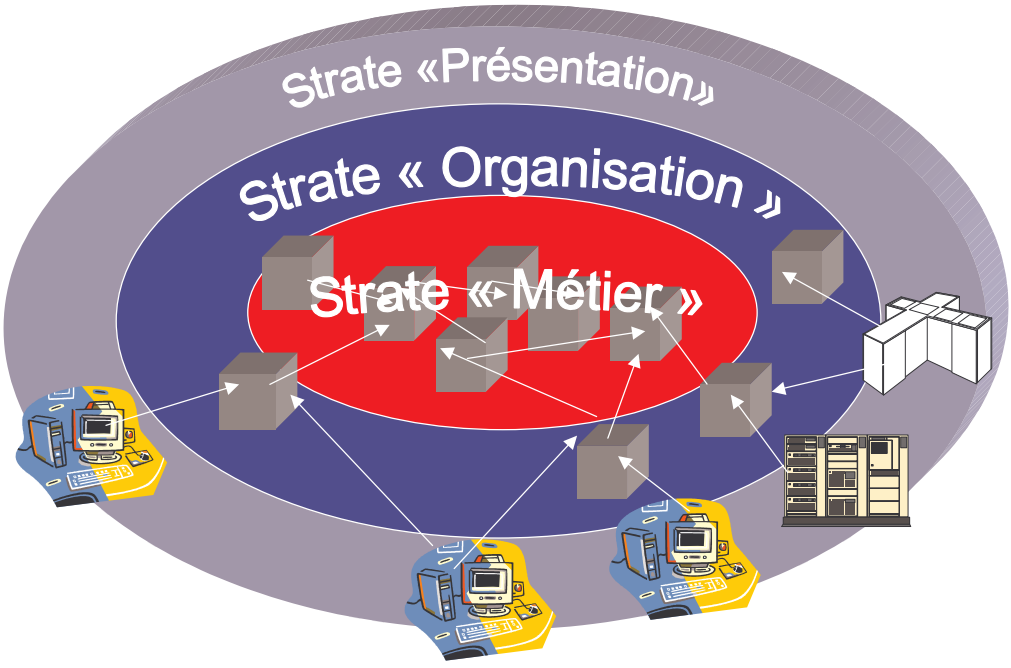
La strate « Organisation »

La strate « Organisation » isole les choix d'organisation, les variantes liées aux pratiques et aux habitudes de travail. Cet aspect est plus volatil : d'une part, les choix de cette nature varient d'un organisme à l'autre, sans que cela ait un impact sur le cœur de métier ; d'autre part, un même organisme doit pouvoir ajuster facilement son organisation. Les constituants logiques qui s'inscrivent dans cette strate reflètent les processus et les situations de travail.

La strate « Présentation »

La strate « Présentation » permet une description logique du dialogue homme-machine, c'est-à-dire indépendamment de la technologie d'interface. La notion d'interface couvre, également, les échanges avec d'autres systèmes.

Figure PxM-40_19. La stratification du système, imposée par l'architecture logique



Orientations de l'architecture de services (suite)

Les filières de dérivation vers l'aspect logique

Les aspects connexes

L'aspect logique s'abreuve à deux aspects « mont » : l'aspect sémantique et l'aspect pragmatique. Dans les deux cas, la relation qui l'unit à ces aspects résume la dérivation.

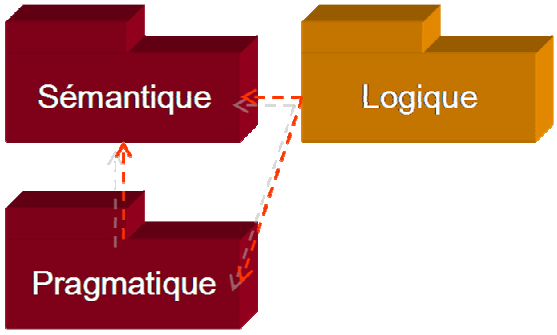


Figure PxM-40_20. Le positionnement de l'aspect logique dans la Topologie du Système Entreprise

Plus précisément, ces relations entre les aspects résument des règles de dérivation qui s'ordonnent en filières, selon leurs produits. En tout, on compte six filières de dérivation, trois pour chacun des aspects « amont ».

Les filières de dérivation

Une filière de dérivation est un ensemble de règles de dérivation produisant un résultat identifié.

À partir de l'aspect sémantique

La dérivation qui s'exerce sur le modèle sémantique produit :

- le modèle logique des données ;
- le modèle logique des services ;
- la description des flux (langage pivot).

Les classes sémantiques fournissent la matière sur laquelle s'appliquent ces trois filières. Le chapitre « Synthèse des règles » montrera que d'autres types d'éléments du modèle sémantique se soumettent également à la dérivation.

À partir de l'aspect pragmatique

Un modèle pragmatique peut embarquer des classes : celles-ci décrivent des objets de nature organisationnelle. Dans ce cas, les filières précédentes s'appliquent à la différence que leur résultat se range dans la strate « Organisation ».

Trois filières de dérivation sont propres à l'aspect pragmatique. Elles partent des cas d'utilisation pour produire :

- le modèle des services de la strate « Organisation » ;
- le modèle des données associées aux constituants de cette strate ;
- la description des flux nécessaires au fonctionnement des orchestrateurs (ces flux agrègent surtout les structures de données définies dans la strate « Métier » ; ils véhiculent, en plus, une information de nature purement organisationnelle, liée au contexte d'utilisation).

Le modèle pragmatique contient, également, la description des processus organisationnel. Ceux-ci donnent une vue globale de l'activité de l'entreprise. Cette « vue de l'organisation » ne peut, en aucun cas, entrer en contradiction avec la « vue de l'utilisation », établie à base de cas d'utilisation. La description des processus apporte des éléments d'information qui ne sont pas présents dans les cas d'utilisation. Ils concernent les règles d'organisation et la régulation de l'activité. Ces éléments doivent être eux-mêmes repris dans le modèle logique. Pour être plus précis sur ce point, il faut savoir comment les processus ont été conçus et décrits. La clef réside dans la relation entre les processus et les objets « métier ». Ce sujet n'est pas abordé dans ce document. Nous supposons, ici, que le modèle sémantique décrit parfaitement les objets « métier », avec leur cycle de vie, conformément au procédé de modélisation sémantique de Praxeme[18].

[18] Cf. le « Guide de l'aspect sémantique », référence PxM-10.

Orientations de l'architecture de services (suite)

Le rôle de la maîtrise d'ouvrage

Le propriétaire du système

La maîtrise d'ouvrage est le propriétaire naturel des modèles du métier, c'est-à-dire des descriptions portant sur les aspects « amont » : sémantique, pragmatique et géographique. Elle devrait, en conséquence, se sentir responsable de leur élaboration et de leur administration.

Une maîtrise d'ouvrage qui voudrait, de plus, restaurer son rôle de propriétaire du système[19] ressentirait le besoin de comprendre le système, de l'intérieur. Pour arbitrer entre les priorités, pour prendre part à la longue visée de restructuration du système, le maître d'ouvrage doit se faire une idée de la façon dont est construit le système. Fort heureusement, cet objectif peut être atteint sans acquérir la connaissance technique de l'informaticien. L'urbanisation du SI trouve sa motivation première dans cette intention : donner, à l'aide de la métaphore, une idée assez exacte du système et de sa structure. À partir de cette connaissance, il devient possible de participer aux décisions sur le devenir du système. Plus généralement, l'aspect logique est un aspect sous lequel le maître d'ouvrage peut percevoir le système.

L'aspect logique n'est ni métier, ni technique. C'est un aspect intermédiaire qui permet de prendre des décisions de structuration du système, dans une indépendance relative à l'égard des choix de technologies. C'est donc une zone intermédiaire que la MOA peut fréquenter.

Le modèle logique inscrit le discours de l'urbanisation du système d'information et celui de l'architecture de services (SOA). Ces deux discours et les disciplines qui leur sont attachées partagent ceci : ils fondent leur formulation sur une métaphore. Dans un cas, la planification du développement citadin ; dans l'autre, la relation entre fournisseurs et consommateurs. Ces métaphores permettent de penser le système. Elles favorisent la communication. Nous y voyons un moyen, pour la MOA, de recouvrer sa fonction de propriétaire et de se réapproprier sa responsabilité dans le devenir du système. En cela, la gouvernance du SI se renforce et trouve son vrai contenu : la transparence donnée par la DSI à ceux qu'elle sert.

Les apports du modèle logique pour la MOA

Le modèle logique, avec son vocabulaire évocateur[20], offre des atouts à la maîtrise d'ouvrage dans sa relation avec la maîtrise d'œuvre.

Pour arbitrer les priorités de développement, il est nécessaire d'avoir une compréhension minimale du système. De même, pour s'assurer de la qualité, de l'évolutivité, de l'interopérabilité du système…

Le modèle logique et, au premier niveau, l'architecture logique donnent à la MOA une vue suffisante du système informatique lui-même, au-delà des modèles « métier » mais sans pour autant se perdre dans les détails de la technologie.

---

[19] C'est bien le sens que revêt l'appellation de « maître d'ouvrage » dans son domaine d'origine : les bâtiments et travaux publics.

[20] Avec Praxeme pour SOA, on parlera de services fournis par des machines, rangées dans des ateliers, eux-mêmes disposés dans des fabriques. On peut aussi préférer la métaphore de l'urbanisme et parler de blocs, quartiers, îlots… Les termes importent peu : ce qui compte, ce sont les contraintes topologiques associées à ces termes et les décisions de structuration qu'ils aident à prendre.

Orientations de l'architecture de services (suite)

De la gouvernance à l'administration des services

Un point de vocabulaire

Le langage étant notre principal outil de travail, nous devons en prendre soin et éviter de galvauder des termes qui devraient conserver un contenu précis. Ainsi, nous éviterons d'utiliser le terme « gouvernance », là où ceux de « gestion » ou « administration » suffisent. La gouvernance a un sens très précis, technique, fabriqué par les économistes et les théoriciens du management pour caractériser certaines bonnes pratiques que devraient adopter les entreprises et leur management vis-à-vis de leurs actionnaires. Par un effet de décalage tout à fait légitime, le terme se trouve maintenant appliqué au fonctionnement de la DSI. La gouvernance du SI exprime la revendication normale de l'entreprise, de son management et de ses directions « métier », de comprendre et contrôler le fonctionnement de la DSI. L'orientation actuelle, obnubilée par les processus, focalise la gouvernance sur l'activité de la DSI. Toutefois, il est au moins aussi important que les parties prenantes se mêlent du destin du SI, lui-même : après tout, il est leur outil de travail, leur chaîne de production sur laquelle repose, en partie, leur activité.

En tout cas, on ne trouvera pas, dans Praxeme, des abus de langage comme « gouvernance des services » ou « gouvernance des données ». L'administration des données est une ancienne discipline et suffisamment respectable pour qu'il ne soit pas nécessaire de changer son nom.

Une nouvelle activité

Le développement SOA appelle une activité nouvelle : ces milliers de services composant le système, il faut bien les enregistrer, les homologuer, les publier, bref, les administrer. L'administration des services est nécessairement centralisée. Cette activité, comparable à l'administration des données, a pour but de constituer un référentiel des services et de le mettre à disposition. Elle doit stimuler la réutilisation.

Cette activité est étroitement liée à l'architecture logique. Elle se place sous la responsabilité de l'architecte logique. Elle se range parmi les activités de support aux projets.

Dans la mesure du possible, le dispositif organisationnel et technique de l'administration des services comprend le suivi des réalisations, l'estimation du taux de réutilisation et la mesure du ROI. L'administration des services prend sa place parmi les activités transverses nécessaires pour actualiser les objectifs de réutilisation des composants et de construction du système cible.

Orientations de l'architecture de services (suite)

L'exploitation du modèle logique

Que devient le modèle logique ?

Le modèle logique se construit à partir des modèles « amont ». Lui-même prépare la conception et la réalisation du logiciel. Là encore, la dérivation intervient. Elle dépend largement de l'architecture technique et des solutions rassemblées. L'existence d'un *framework* détermine grandement la dérivation et le mode de travail du développeur.

La dérivation du modèle logique produit un nouveau modèle : le modèle du logiciel. Dans les termes de MDA, ce modèle logiciel est un PSM – *platform specific model*. Il dépend largement des choix techniques. Ce modèle logiciel, complété par des décisions de conception technique, permettra de générer le logiciel lui-même.

Certaines parties du modèle logique préparent des informations pour alimenter des dispositifs logiciels :

- Les structures de données associées aux machines logiques spécifient le langage pivot qui pourra, éventuellement, déboucher sur des schémas xsd.
- Des règles ou des expressions logiques pourront être dirigées vers un moteur de règles.
- Les messages et codifications rassemblés dans le modèle alimenteront les bases.
- Le modèle est un référentiel de description : il contient la liste, partagée, des services. Cette liste prendra la forme de l'annuaire des services, clef de voûte de l'exécution dans un système SOA.
- Certaines précisions sur l'utilisation contextuelle des services sont du ressort d'une solution de MDM.
- Par ailleurs, le modèle logique contient le modèle logique des données, lequel anticipe les schémas de bases de données.

Orientations de l'architecture de services (suite)

Conclusion

Conséquences pratiques

L'élaboration de l'architecture de services applique la démarche suivante :

1. Étape 1 : dérivation du modèle sémantique pour constituer le cœur du système.
2. Étape 2 : décision de structuration logique sur le cœur du système (ces deux étapes sont intimement liées et peuvent se confondre dans la pratique).
3. Étape 3 : dérivation du modèle pragmatique pour obtenir la strate intermédiaire du système, celle des machines logiques « Organisation ».
4. Étape 4 : analyse des besoins de services des machines logiques de la strate « Organisation » (MLO) vers le noyau (les machines logiques de la strate « Métier », MLM) et connexion des deux strates.
5. Étape 5 : bilan de l'information recueillie par la MLO au moment où elle sollicite les services internes (services de MLM) ; en fonction de ce contexte, décision de modifier les interfaces des services pour réduire le couplage.

Le principe de cette démarche est d'optimiser la structure en mettant à profit les contextes d'utilisation.

Le chapitre « Démarche de conception des services » (pp. 24 et sq.) détaille la démarche.

Termes et représentations de l'aspect logique pour SOA

Le vocabulaire de l'architecture logique

Introduction

La conception logique se dote d'un vocabulaire propre, pour parler de l'aspect logique. À partir de l'idée naturelle du service, Praxeme file la métaphore de l'industrie, récupérant la notion de « machine logique », trouvée dans la méthode TACT. Ainsi, les objets de la conception logique se nomment service, machine, atelier, fabrique. Ce sont les constituants logiques, c'est-à-dire les éléments du système, perçu sous son aspect logique.

Des notions satellites complètent le paysage : strate, interface, structure de données…

Ce chapitre présente la terminologie de la modélisation logique, en partant des constituants les plus vastes et en allant vers les plus fins.

L'économie

La méthodologie cherche à réduire cet outillage terminologique au strict minimum. Ainsi, les mêmes termes seront appliqués aux différentes « strates » de l'architecture logique. Les particularités liées à ces strates font l'objet des chapitres suivants. Dans le souci d'alléger le bagage du praticien, la méthodologie évite de multiplier les notions et se méfie des typologies. Elle ne retient les termes et notions que quand ils présentent un intérêt opératoire[21].

L'usage des termes

Les termes de la modélisation logique désignent des catégories de représentation, choisies pour leur capacité à représenter et à structurer le système, d'un point de vue logique. Sous la forme de stéréotypes, ils sont injectés dans l'outil de modélisation UML. De cette façon, l'outil UML se plie à l'usage de la modélisation logique.

L'outillage

Le fait de recourir au même outillage que celui utilisé pour les aspects amont (sémantique, pragmatique) et aval (logiciel, physique) présente plusieurs avantages :

- économique, d'abord (mêmes licences ; pas de nouveaux coûts d'apprentissage ou d'administration) ;
- opératoire, ensuite (facilité de référence entre les modèles des différents aspects ; mise en œuvre facilitée de la dérivation ; conformité au standard MDA).

[21] Nous appliquons le critère du rasoir d'Occam (« *Il ne faut pas multiplier les êtres sans nécessité*. », Guillaume d'Occam).

Termes et représentations de l'aspect logique pour SOA (suite)

Les strates

Définition

Par décision d'architecture logique, le système se sédimente en trois strates qui séparent des constituants de nature différente :

- La strate « Métier » : au cœur du système, elle concentre l'essentiel du métier.
- La strate « Organisation », enrobant et protégeant la première, elle isole les choix d'organisation.
- La strate « Présentation » est la vitrine que le système présente à son environnement.

La strate est une portion du système, isolée par la nature de son contenu[22].

Origine

Les strates sont issues d'une décision générale, elle-même reprenant une pratique généralisée.

Le contenu de ces strates provient des modèles « amont » : sémantique pour la strate « Métier », pragmatique pour la strate « Organisation ».

Utilisation

En plaçant un constituant logique dans l'une ou l'autre des trois strates, le concepteur indique la nature de celui-ci. L'appartenance à une strate entraîne certaines conséquences sur la nature et le comportement du constituant logique.

La communication est orientée entre les strates, conformément au schéma ci-dessous. Un constituant de la strate « Présentation » peut appeler des constituants de la strate « Organisation ». À partir de celle-ci, certains constituants de la strate « Métier » sont visibles. Les mouvements inverses sont bannis. On dit que l'architecture est polarisée.

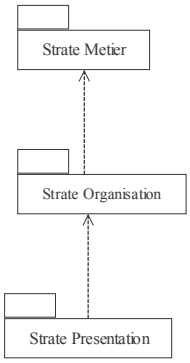


Figure PxM-40_21. Les relations permises entre les strates

Caractéristiques

Les strates n'ont d'autre substance que leur contenu. Ce sont des espaces qui structurent le modèle logique, sans que le concepteur logique ait à y intervenir.

Documentation

Il n'y a pas de documentation requise pour les strates, autre que les définitions et règles données par la méthodologie.

Formalisation La strate est représentée par un paquetage UML, inscrit directement sous le paquetage représentant l'aspect logique.

[22] La notion de strate correspond à celle de couche. Le terme strate a été préféré pour éviter toute confusion avec les couches de l'architecture technique. Les deux décompositions ne se confondent pas.

Termes et représentations de l'aspect logique pour SOA (suite)

Les fabriques logiques

Définition

Agrégat d'ateliers logiques correspondant à un domaine dans les modèles amont[23].

Origine

La fabrique logique de la strate « Métier » dérive du domaine d'objets[24].

La fabrique logique de la strate « Organisation » correspond à l'organisme (l'entreprise étudiée ou un de ses partenaires).

Typologie

Il n'y a pas de type de fabrique logique. Seule leur situation sur une des strates indique leur nature. On obtient, ainsi, des FLM (fabriques logique « Métier ») et des FLO (fabriques logiques « Organisation »).

Fabriques « transverses »

On peut faire une exception pour les fabriques « utilitaires », rassemblant des services généraux ou transverses. On en trouve dans les deux strates :

- Strate « Métier » : fabrique « Utilitaires », rassemblant des services et des ressources d'un intérêt général.
- Strate « Organisation » : fabrique « Organisation », offrant des services génériques liés à la description des organisations.

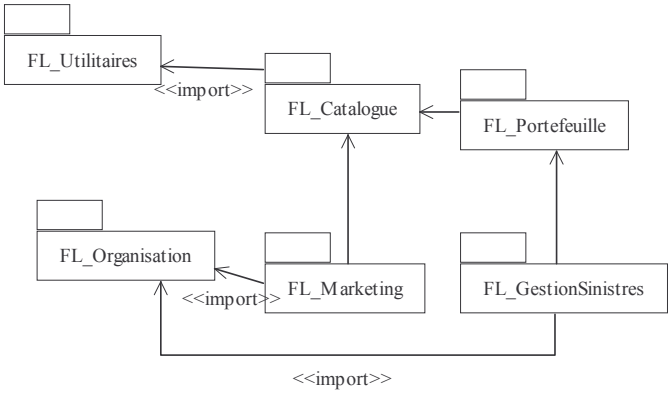
Ces fabriques transverses présentent une particularité architecturale : on peut choisir de les incorporer, en tout ou partie, dans les autres fabriques.

FLM et FLO

FLM et FLO ne se comportent pas de la même façon, statistiquement :

- Les relations entre fabriques de la strate « Métier » sont surtout dynamiques, basées principalement sur l'appel de services.
- Les relations entre fabriques de la strate « Organisation », rares, sont surtout statiques : généralisation ou importation mais pas d'échange.

Figure PxM-40_22. Exemple de graphe d'architecture



[23] Cette définition renvoie à la notion de domaine, sans préciser celle-ci. Cette position correspond au rôle de l'aspect logique, qui reçoit sa matière des aspects sémantique et pragmatique et se « contente » de la dériver et de la structurer.

[24] Rappel : le domaine d'objets est l'unité de décomposition du modèle sémantique (cf. *Guide de l'aspect sémantique*, réf. « PxM-10 »).

Utilisation

Une fabrique logique forme un tout cohérent, comparable au module d'un ERP ou au domaine d'une architecture fonctionnelle.

Aux niveaux logiciel puis physique, on ne lui fait pas correspondre de réalité très précise (ce sera le rôle de l'atelier logique).

En revanche, la fabrique constitue une unité clef pour la démarche d'urbanisation. Elle engage les notions d'organisation des ressources humaines (tant maîtrise d'ouvrage que maîtrise d'œuvre), de responsabilité et de propriété.

Caractéristiques

- La fabrique n'offre ni interface, ni service[25].
- Les relations entre fabriques sont : l'utilisation, l'importation et la généralisation.

Documentation

Formalisation La fabrique logique est représentée par un paquetage UML, stéréotypé « Fabrique Logique »[26].

Les fabriques se placent dans une strate, elle-même représentée par un paquetage. La localisation d'une fabrique sur une strate conditionne son comportement statique et dynamique.

Définition La définition de la fabrique est rédigée dans une note « commentaire » attachée au paquetage. Elle reprend les définitions tirées des modèles amont (domaines d'objets ou organismes). Dans le cas des fabriques transverses, l'architecte rédige la définition et justifie la fabrique.

Description La description d'une fabrique couvre :

- la liste des ateliers qu'elle contient ;
- les relations qu'elle entretient avec d'autres fabriques ;
- la désignation du propriétaire de la fabrique dans l'organisation humaine ;
- les décisions et prévisions qui la concernent dans le plan d'urbanisation.

Représentation Les fabriques apparaissent sur le graphe d'architecture logique. Une même fabrique peut figurer sur plusieurs versions du graphe, jalonnant la trajectoire d'urbanisation. Pour chaque étape, l'urbaniste précise l'état de construction de la fabrique.

[25] C'est un choix. Il est possible de faire autrement. Ce choix permet de simplifier la conception architecturale, en concentrant les décisions d'architecture sur l'unité intermédiaire de l'atelier logique.

[26] Il n'est pas nécessaire de préciser « métier » ou « organisation ».

Termes et représentations de l'aspect logique pour SOA (suite)

Les ateliers logiques

Définition

Ensemble de machines logiques formant un tout cohérent du point de vue fonctionnel.

NB : Le critère de cohérence s'appuie sur la proximité sémantique des machines et se mesure par les relations qu'elles entretiennent. Un atelier s'ordonne autour d'un concept central, auquel s'adjoignent les notions subordonnées.

Origine

Ateliers « Métier » Les ateliers de la strate « Métier » résultent de décision de structuration prises par l'architecte logique. Aucun élément ne leur correspond dans le modèle sémantique, sauf dans les cas suivants :

- Les domaines d'objets ont été décomposés en sous-domaines.
- Les associations ont été orientées.

Dans ces cas, évidemment, l'architecture logique s'efforce de conserver ces décisions de structuration. Celles-ci ont peu de raison de s'imposer dès le modèle sémantique. Elles le contraignent fortement et risquent de fausser la modélisation sémantique. Il est à noter que le découpage en domaines d'objets déborde du strict champ de la modélisation sémantique et inscrit, déjà, des décisions fortes d'ordre logique.

Ateliers « Organisation » Les ateliers de la strate « Organisation » reprennent les domaines fonctionnels. Un domaine fonctionnel est un champ d'activité, circonscrit dans l'organisation de l'entreprise. Il porte souvent un nom d'action, par exemple : archivage, gestion des sinistres, vente, marketing. Seule la strate périphérique reflète cette organisation. Cette approche diffère radicalement de l'architecture fonctionnelle qui utilise ce même critère pour structurer la totalité du système, noyau compris. En isolant le « cœur de métier », constitué des objets et savoirs fondamentaux et indépendants de l'organisation, l'architecte augmente les possibilités de réutilisation et facilite les reconfigurations organisationnelles.

Typologie

Ateliers « Métier » et ateliers « Organisation » sont semblables quant à leur description mais diffèrent sur les points suivants :

- Par application de la règle de stratification (cf. p. 78), les ateliers « Organisation » sollicitent les ateliers « Métier », mais l'inverse est rigoureusement interdit.
- Les ateliers « Organisation » doivent être moins couplés entre eux que ne le sont les ateliers « Métier ». Les relations entre ateliers de la strate « Organisation » doivent être exceptionnelles. Des relations entre cas d'utilisation (surtout « *include* ») peuvent relier les domaines fonctionnels. Même dans ce cas, les ateliers correspondants ne sont pas forcément couplés. Si la logique procédurale et transactionnelle le permet, la relation entre les cas d'utilisation n'est pas reportée dans la strate « Organisation », mais uniquement dans la strate « Présentation » ainsi que dans le système d'habilitation[27].

Les ateliers des strates « Métier » et « Organisation » se comportent de façon différente. Les premiers sont opaques : on ne voit que leur interface. Les seconds sont transparents : leur contenu peut être sollicité directement.

[27] Les habilitations se déduisent des droits d'accès figurés sur le diagramme des cas d'utilisation.

Utilisation

L'atelier logique fournit **l'unité de déploiement** qui sera utilisée sur le plan physique.

Les conséquences suivantes en découlent :

1. Les choix d'architecture technique s'appliquent au niveau de l'atelier[28].
2. Un type de composant logiciel parfaitement identifié doit correspondre à l'unité de l'atelier.
3. Le passage à l'architecture physique s'obtient en localisant, sur l'architecture matérielle, ces composants logiciels qui traduisent les ateliers.

Caractéristiques

La règle d'encapsulation, retenue pour notre architecture, impose de cacher les machines à l'intérieur des ateliers. En contrepartie, les ateliers offrent une interface dans laquelle ils rassemblent les services publics de leurs machines.

Cette règle se justifie par la simplification imposée à l'architecture. La structure interne des ateliers peut changer sans impact sur les composants appelants. Le concepteur et le mainteneur disposent ainsi d'une plus grande liberté pour faire évoluer le contenu des ateliers. La contrepartie réside dans la déformation plus forte imposée à l'architecture de services par rapport aux modèles amont.

Un atelier peut présenter plusieurs interfaces, chacune regroupant un ensemble cohérent de services (voir plus loin).

Les ateliers de la strate « Organisation », dérivés des domaines fonctionnels, se comportent différemment. Les services qu'ils contiennent sont mis en musique dans les dialogues homme-machine. Ces ateliers restent ouverts et ne présentent pas d'interface.

Documentation

Formalisation L'atelier logique est traduit, dans le formalisme UML, sous la forme d'un paquetage auquel on applique le stéréotype « Atelier logique ».

L'atelier logique est toujours rangé dans une fabrique logique, dont il tire sa nature par simple positionnement sur une des strates.

Il est, toutefois, commode de définir deux stéréotypes, selon la visibilité sur le contenu de l'atelier :

- Atelier fermé (ou opaque) : tous les ateliers de la strate « Métier » et quelques ateliers de la strate « Organisation » (ce point est discuté plus loin).
- Atelier ouvert (ou transparent) : la plupart des ateliers de la strate « Organisation », en tout cas ceux qui dérivent des domaines fonctionnels.

Définition La définition de l'atelier logique est rédigée sous la forme d'une note attachée au paquetage. Cette note contient la justification de création ainsi que les indications pour délimiter le périmètre et les responsabilités de l'atelier.

Description La description de l'atelier comprend, dans l'ordre décroissant d'importance :

- les interfaces de l'atelier et les missions associées ;
- pour chaque interface, les services proposés, avec leur contrat ;
- la liste des machines logiques qui le composent.

[28] On peut concevoir que des ateliers d'une même fabrique soient développés avec des technologies différentes, mais pas que les composants d'un même atelier fassent l'objet de décisions techniques séparées.

Représentation L'atelier apparaît sur les diagrammes de classes du modèle logique, comme un paquetage doté d'une ou plusieurs interfaces. Pour chaque atelier, le concepteur élabore un diagramme simplifié faisant état des besoins de l'atelier vers les autres ressources du système. Il réalise un autre diagramme, plus détaillé, montrant le contenu de l'atelier et les relations d'utilisation internes et externes.

Figure PxM-40_23. Exemple de graphe montrant l'environnement d'un atelier

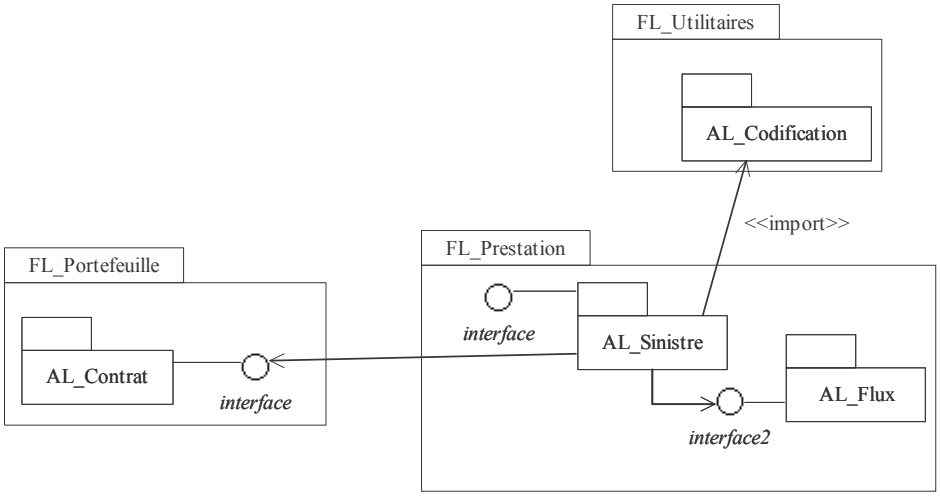
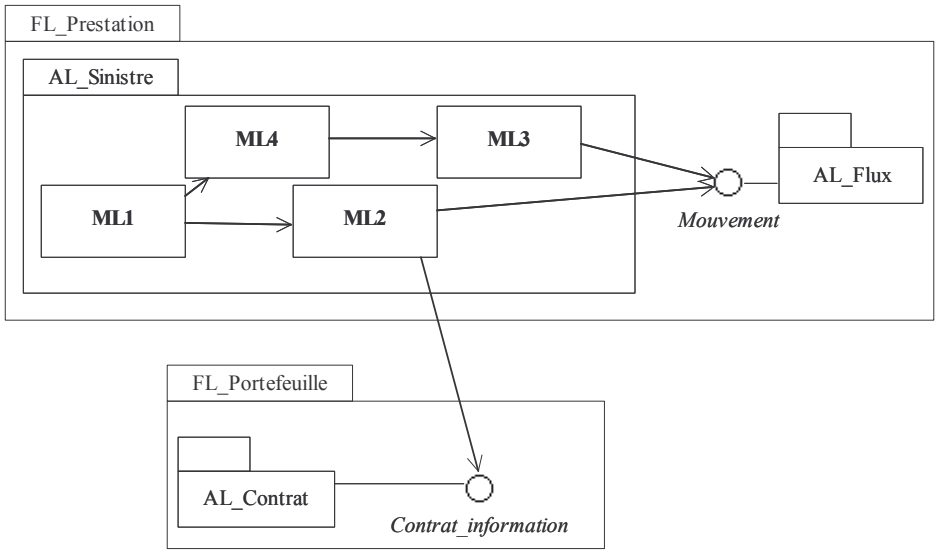


Figure PxM-40_24. Exemple de graphe pour le contenu d'un atelier



Termes et représentations de l'aspect logique pour SOA (suite)

Les machines logiques

Définition

Ensemble cohérent de services logiques, constitué autour d'une notion forte d'un modèle amont.

NB : La définition n'a pas à préciser le mode de délimitation de la machine puisque, dans Praxeme, la méthodologie fait dériver cette unité logique des unités plus significatives que la modélisation a isolées dans le métier et l'activité.

Origine

Les machines logiques dérivent :

- soit des classes sémantiques (machines logiques « Métier ») ;
- soit des classes du modèle pragmatique (machines logiques traitant les objets liés à l'organisation : habilitation, acteur… ou les objets de nature administrative : dossier, synthèse client, formulaire…) ;
- soit des cas d'utilisation (machines logiques de la strate « Organisation », dites MLO).

De façon marginale, apparaissent également des machines logiques qui ne dérivent pas des modèles amont, mais qui inscrivent, dans l'architecture, des dispositifs particuliers (par exemple : gestion des incidents). La fabrique logique « Utilitaires » rassemble ces machines dont les services sont transverses.

Typologie

Reprenant les origines possibles des machines, on distingue :

1. les machines « Métier » (MLM) ;
2. les machines de la strate « Organisation », dérivant les classes du modèle pragmatique ;
3. les machines « Organisation » ordinaires (MLO), chargées de la logique procédurale.

Cette distinction découle des orientations de l'architecture logique, tout particulièrement du principe de stratification. Elle est utile si on considère :

- le mode d'interrelation entre les machines, assez différent entre MLO et entre MLM (voir discussion ci-dessous) ;
- la position spéciale des ML dérivant de classes du modèle pragmatique ;
- la gestion des transactions, apanage exclusif des MLO ;
- la visibilité à l'extérieur du système, via la strate « Présentation », seule la strate « Organisation » étant exposée.

Plus important pour le développement, la distinction logique entre extension et intension[29] permet de reconnaître machines ensemblistes et élémentaires.

Machines élémentaires

Les machines élémentaires ont en charge un objet, dans toutes ses missions et transformations (ex. : contrat, sinistre, couverture…).

[29] En logique, un concept se définit « en intension » (sa définition) et « en extension » (en tant qu'ensemble des instances compatibles avec le concept).

Service	Définition	Nom réservé
Suppression d'une instance	Le destructeur est sous la responsabilité de la machine élémentaire, puisque c'est elle qui contrôle le cycle de vie de l'objet.	supprimer

Machines ensemblistes

Les machines ensemblistes sont ajoutées par la conception logique pour inscrire les opérations et comportements sur les ensembles d'objets. Les machines ensemblistes se chargent des services donnés dans le tableau ci-dessous.

Les classes sémantiques ne donnent pas toujours une machine ensembliste. Le cas peut se produire pour une classe associative : les services ensemblistes sont confiés à la machine correspondant à une des classes reliées par l'association. L'architecte logique ne s'autorise cette simplification qu'à la condition que les services ensemblistes restent dans le même atelier que la machine qui dérive la classe associative.

Service	Définition	Nom réservé
Création des instances	C'est la machine ensembliste qui contient le constructeur, ce qui lui permet de vérifier les contraintes telles que nombre maximum, compteur…	creer
Recherche d'une instance	Recherche d'une instance à partir de son identifiant (cela permet, ensuite, d'activer la machine élémentaire sur l'instance désignée).	lire
Recherche multi-critères	Ce service retourne la liste des instances trouvées, qui obéissent aux critères passés en interface.	rechercher

Utilisation

Si l'atelier constitue l'unité de déploiement, la machine logique, elle, permet de structurer cet ensemble en composants atomiques, faciles à appréhender et à suivre. C'est un niveau d'agrégation intermédiaire entre l'atelier et le service, sans lequel l'atelier se révèlerait complexe. Le fait que les machines dérivent naturellement des modèles amont réduit considérablement le travail de structuration demandé aux concepteurs logiques.

Caractéristiques

La machine logique contient trois constituants :

1. la liste des services, lesquels, d'une certaine façon, décomposent la mission de la machine ;
2. la structure de données, par laquelle elle communique avec l'extérieur ;
3. l'automate à états qui la contrôle et contraint son comportement.

Du point de vue logique, ces constituants sont irréductibles : ils comptent parmi les termes que nous avons choisis d'utiliser pour exprimer l'architecture de services[30].

Ces trois constituants font l'objet de sections particulières dans la suite de ce document.

Une autre caractéristique de la machine logique est sa visibilité.

La visibilité de la machine

La visibilité de la machine conditionne fortement l'architecture logique. Elle contribue à la maîtrise du couplage ; elle détermine ce qui peut être demandé à partir d'un point du système.

Les valeurs de la visibilité sont données dans le tableau ci-dessous. La question de la visibilité implique plusieurs notions : visibilité de la machine, visibilité du service, imbrication, interface et dépendance. Elle conditionne la conception architecturale. Un paragraphe lui est dédié (p. 63).

Valeur	Définition	Conséquence
Publique	La machine est visible de tout point qui a accès à son espace de nommage.	À utiliser pour donner accès à partir de l'interface.
Protégée	La machine est visible uniquement à l'intérieur de l'atelier et des ateliers liés par la relation de généralisation.	À utiliser à la place de « privée » pour augmenter les possibilités d'extension.
Privée	La machine n'est vue que des machines de son atelier.	On peut restreindre encore la visibilité en imbriquant la machine dans une autre (voir p. 49).

[30] On y reconnaît les trois dimensions de la modélisation : structurelle, fonctionnelle, contractuelle.

Termes et représentations de l'aspect logique pour SOA (suite)

Les services logiques

Définition

Le service logique est le grain élémentaire dans l'architecture de services.

Tout ce que l'on peut demander au système – information, action ou transformation – s'obtient par un service.

NB : Praxeme prend le terme « service » au pied de la lettre. « Calculer les coûts », « Trouver les clients respectant certains critères », « Déclarer un sinistre », « Passer le contrôle du drone »… sont des exemples de services attendus du système. Le service se nomme d'après un verbe d'action. Il correspond à quelque chose que le système doit faire pour… rendre le service. Tout naturellement, dans l'approche de modélisation orientée objets, l'action est représentée par l'opération.

Ce niveau de maille que Praxeme assigne au terme « service » peut surprendre les personnes habituées à l'utiliser au niveau du composant (le *web service*, par exemple), le composant portant lui-même des opérations. Dans Praxeme, c'est la machine logique qui porte des opérations stéréotypées « service ». Il n'y a rien de plus fin que le service (du moins, vu de l'extérieur).

Origine

L'identification des services provient d'une des sources suivantes :

- Le service peut dériver d'une opération, celle-ci ayant été trouvée soit sur une classe sémantique, soit sur une classe du modèle pragmatique.
- Le service de MLO dérive soit d'une activité élémentaire, soit d'un cas d'utilisation.
- La conception logique ajoute des services généraux, ceux inscrits sur les classes génériques ML_Ensemble et ML_Element.

Typologie

Selon le positionnement sur l'architecture, on parlera de services internes (MLM) et de services externes (MLO).

Le niveau d'agrégation conduit à opposer les services de machines – cachés par application du principe d'encapsulation – et les services d'interface. Ces derniers sont les services publiés par les ateliers. Ils ont ceci de particulier qu'ils ne font rien par eux-mêmes mais qu'ils servent de relais vers les services de machines ; éventuellement, ils peuvent les assembler.

Par rapport à des problématiques plus précises (par exemple, les services ensemblistes, les traitements *batch*…), nous rencontrerons des appellations particulières pour les services : constructeur, informateur, etc. Mais cela n'enrichit pas, pour autant, la typologie des services. Tous ces services se décrivent et se comportent de la même façon. Il n'est donc pas nécessaire d'alourdir le vocabulaire et de créer d'autres stéréotypes.

Utilisation

C'est au niveau des services que se concentre l'effort de développement et de test. Les notions présentées précédemment ressortissent à la conception architecturale seule. La conception des services est le dernier stade et précise le détail du fonctionnement des services. Elle doit vérifier les choix de structuration arrêtés par l'architecture.

Caractéristiques

Le service possède une interface riche : sa signature et son contrat.

Il peut également modifier le contenu d'information associé à la machine.

Une caractéristique importante du point de vue architectural est la visibilité du service.

Documentation

Formalisation En recourant au standard UML, le service se représente naturellement par une opération : service et opération sont des éléments de traitement, au même niveau de maille. L'opération est stéréotypée « Service logique ». Ceci oblige à l'inscrire sur une classe stéréotypée « Machine logique ».

La signature obéit à des règles de nommage : nom du service, signature, définition des paramètres. Ce sujet est développé p. 39.

Définition Chaque service reçoit une définition brève qui exprime son objectif. Cette définition est importante car c'est en en prenant connaissance que d'autres concepteurs auront l'idée d'utiliser le service[31].

Description En ce qui concerne la description algorithmique, le concepteur dispose d'une notation textuelle, le pseudo-langage, et d'une notation graphique, le diagramme d'activité. Le procédé stipule leur utilisation combinée :

1. Le diagramme d'activité, utilisé pour l'algorithme du service, fait apparaître les services appelés par le service étudié.
2. Les activités du diagramme d'activité sont décrites en pseudo-langage, quand elles sont internes au service décrit.

Le diagramme d'activité permet de visualiser les appels d'autres services. C'est un outil indispensable pour contrôler l'architecture.

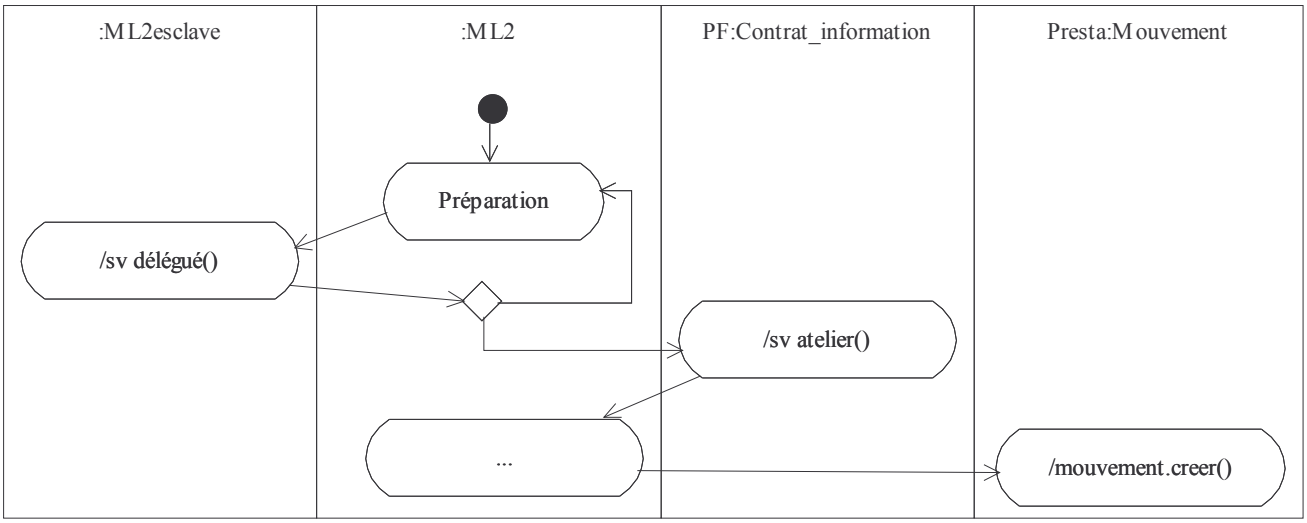
Représentation Le procédé fixé ci-dessus prépare le modèle logique à un rapprochement avec BPEL.

Pour chaque service, on réalise un diagramme d'activité construit de la façon suivante (illustrée par l'exemple de la page suivante) :

1. Un graphe d'activité (et un seul) est attaché au service logique (opération UML). Pour ce graphe, le concepteur réalise au moins un diagramme d'activité. Il peut en réaliser plusieurs pour des raisons de commodité de lecture (effet zoom, surtout).
2. Chaque machine (y compris les interfaces des ateliers sollicités) apparaît dans le diagramme sous la forme d'un couloir (partition).
3. La machine du service que l'on décrit figure elle-même sur le diagramme (plutôt au centre).
4. On pose, dans les couloirs, les activités et les éléments permettant de dessiner l'algorithme.
5. Dans les couloirs des autres machines, les symboles d'activité représentent uniquement les services (champ « Name » de la boîte de dialogue).
6. Dans le couloir de la machine décrite, peuvent figurer à la fois des appels à d'autres services de la même machine et des activités dont le contenu sera décrit en pseudo-langage.

[31] Pour plus de précision, voir le formulaire FRM-45 et son commentaire.

Figure PxM-40_25. Exemple de description de service, sous la forme d'un diagramme d'activité



La visibilité du service

La visibilité du service est une information importante pour l'architecture logique. Elle indique ce qui peut ou non être demandé à partir des différentes localisations dans le système.

Les valeurs de la visibilité sont données dans le tableau ci-dessous. La question de la visibilité implique plusieurs notions : visibilité de la machine, visibilité du service, interface et dépendance. Elle conditionne la conception architecturale. Un paragraphe lui est dédié (p. 63).

Valeur	Définition	Conséquence
Publique	Le service peut être sollicité par tout point du système ayant un accès à sa machine.	Les droits d'utilisation du service dépendent du point à partir duquel il est demandé et des relations entre ce point et la machine.
Protégée	Le service ne peut être sollicité qu'à l'intérieur de son atelier d'appartenance ou des ateliers liés par généralisation.	Un espace de nommage plus réduit peut être instauré autour d'une machine (voir p. 46).
Privée	Le service n'est connu que de sa machine.	Seuls les services de la même machine peuvent invoquer le service privé. Celui-ci joue donc un rôle subalterne, masquant des détails organiques.

La signature du service

La signature comporte les éléments suivants : nom, paramètres, résultat. Seul le nom est obligatoire.

Le nom Le nom complet du service est la forme sous laquelle il pourra être invoqué dans le système logiciel. Il est construit de la façon suivante :

<nom de la machine><séparateur><nom du service sur la machine>.

- Le séparateur est le point '.' ; les règles de génération du logiciel pourront lui substituer un autre caractère.
- Le nom court du service désigne, de la façon la plus explicite possible, l'information ou la transformation demandée au service.

Le nom logique se rapproche des conventions habituelles en programmation, tout en préservant le plus possible la lisibilité obtenue par la sémantique. D'où :

- élimination des espaces et caractères accentués ;
- formule compacte ;
- limitation du nombre de séparateurs...

Par exemple : au lieu de « Couverture.montant à présenter en recours », on préférera "Couverture.recours_calculé" ou "Couverture.recours_montant" ou "Couverture.recoursMontant"[32].

Les paramètres Pour chaque paramètre : son nom (clair), son type, la modalité (entrée et/ou sortie[33]).

La façon d'écrire la signature des services est fortement liée au style d'architecture choisi. Selon la préférence que l'on accorde aux types atomiques habituels ou aux types complexes (voir p. 70), on obtiendra :

1. Une architecture de services absolue : les signatures ne montrant que des types atomiques, réputés universels, il n'est besoin d'aucun autre dispositif pour solliciter le service.
2. Une architecture de services basées sur les flux : les signatures comportent des types complexes exprimant les structures de données ; pour interagir avec les services, il faut « connaître » ces flux, donc avoir accès aux schémas qui les décrivent.

Dans la première option, il faut se donner des règles complémentaires. Par exemple : les premiers paramètres identifient l'objet sur lequel le service doit intervenir. Dans la deuxième option, c'est la structure de donnée complète de l'objet qui pourra être passée. Ce faisant, on gagne en performances puisque l'objet, sous certaines conditions, n'aura pas à être recherché à chaque sollicitation de service.

D'autres considérations façonnent la signature des services. La communication « secondaire » ou « opératoire » avec le service porte non pas sur le flux traité par la machine mais sur les signaux qu'elle peut émettre concernant son fonctionnement. Cette communication peut s'obtenir sous la forme d'un paramètre, le code retour. C'est une façon de le rendre bien visible et de rendre le contrat du service parfaitement lisible à travers la signature du service. Le code retour ne se limite pas à l'erreur. Il contribue à la communication entre le service et son demandeur. Ses valeurs et leurs possibilités d'occurrence font partie du contrat du service et sont documentées en tant que telles.

On peut préférer une autre solution pour cette communication « secondaire ». Plutôt que d'alourdir la signature du service par un paramètre « code retour », on admet un dispositif complémentaire de communication dans le système, à base de signaux. La communication normale, essentielle, reste basée sur l'appel des services mais, en plus, les services renseignent leurs solliciteurs en émettant des événements. Ces événements jouent le rôle des valeurs du code retour. Cette solution est élégante d'un point de vue modélisation : les événements sont une catégories de modélisation connue ; le concepteur les structure en arbre d'héritage et les documente. La négociation logique/technique envisage cette possibilité car le point de vue de l'architecte technique est intéressant pour reprendre ces éléments de modélisation et en faire un dispositif intégré à la solution technique[34]. La liste de la négociation inscrit ce thème sous l'appellation « signalisation ».

Le type du résultat La notation UML laisse la possibilité de définir une opération comme une fonction mathématique, en précisant le type des données qu'elle renvoie. Cette possibilité s'applique bien pour des services qui retournent une information – consultée ou calculée (dans ce cas, le service est nommé comme la donnée).

---

[32] Quelques conventions peuvent faciliter la manipulation des services : "obtenir" pour une réponse instantanée ; "demander" mode "avec rappel" ; l'absence de verbe indique un retour instantané (ex. "disponible (date)") ; un adjectif exprime un état et sera, le plus souvent, de type booléen, etc.

[33] Les trois valeurs possibles sont celles fixées par UML : 'in' (entrée), 'out' (sortie), 'inout' (entrée/sortie). Dans ce dernier cas, la valeur reçue peut être modifiée par le service. Ceci entraîne, au niveau logiciel, un passage de l'information par référence – et non par valeur. Si cette contrainte soulève des difficultés de réalisation, il est loisible de transformer un paramètre lors du passage du logique au logiciel. Le paramètre défini dans le modèle logique comme 'inout' se dérive alors en deux paramètres, l'un 'in', l'autre 'out'. L'important est que le modèle logique fixe toute l'information nécessaire pour les étapes ultérieures.

[34] Cela se rapproche, dans les langages objet, du traitement des « exceptions ».

Termes et représentations de l'aspect logique pour SOA (suite)

Les interfaces des ateliers

Définition

L'interface d'un atelier (appelée aussi façade) est un ensemble de services que l'atelier propose et qui peuvent être demandés par tout point du système ayant un accès explicite à l'interface.

Origine

L'interface de l'atelier est une notion purement logique, sans fondement dans les modèles amont.

Elle résulte d'une décision de l'architecte logique. Si l'atelier ne présente qu'une interface, celle-ci se constitue automatiquement comme l'ensemble des services publics sur les machines publiques de l'atelier (dans la définition de la visibilité donnée plus haut).

Si il existe plusieurs interfaces, elles se répartissent les services ainsi déduits. Dans ce cas, on évite les recouvrements entre les interfaces (ce n'est pas une exigence stricte).

Typologie

Aucune.

En pratique, on observe une spécialisation des interfaces pour répondre à deux catégories d'usage : les interfaces offrant les services « courants », les interfaces dédiées à l'administration ou au suivi du système (configuration, statistiques…). Cette spécialisation ne s'érige pas en règle.

Utilisation

La définition ci-dessus souligne les points suivants :

- L'accès doit être explicite, c'est-à-dire déclaré sous la forme d'une dépendance (relation d'utilisation).
- L'accès peut être déclaré sur une interface et non sur la totalité de l'atelier[35].

Caractéristiques

L'interface est une liste restreinte des services contenus dans l'atelier.

Documentation

Formalisation L'interface est, en UML, une classe portant un stéréotype pré-défini « interface ».

Définition Si l'atelier offre plusieurs interfaces, le concepteur s'efforcera d'expliquer le critère qu'il a retenu. L'interface correspond à une mission générale assignée à l'atelier.

Description On applique les mêmes consignes que pour la machine logique. La description de l'interface se limite à la liste des services qu'elle rassemble et à la structure de données. Les services à exposer et à répartir entre les interfaces se déduisent par combinaison des visibilités sur les machines de l'atelier et sur leurs services (voir p. 63).

[35] Par application de la règle d'encapsulation des machines, l'architecture logique ne tolère pas de relations d'utilisation qui aboutiraient directement à l'atelier et non à une interface. Des dérogations peuvent être accordées à titre exceptionnel. Les autres types de relations, la généralisation et l'importation, s'appliquent aux ateliers mêmes.

Le nom du service d'interface est le nom complet du service qu'il expose. Ainsi le service evaluer sur la machine Couverture est exposé sous la forme d'un service d'interface nommé Couverture.evaluer (c'est le nom de l'opération sur la classe stéréotypée « interface »).

Pour chaque service d'interface, la documentation contient un diagramme dynamique (par exemple, diagramme de collaboration ou diagramme d'activité) montrant les relais à l'intérieur de l'atelier, nécessaires pour rendre le service demandé.

Représentation Sur les diagrammes de classes, l'interface apparaît sous deux formes :

- Externe : une antenne fichée sur le symbole du paquetage ; c'est sur cette antenne qu'aboutissent les dépendances des ateliers utilisateurs.
- Interne : une classe avec son stéréotype « interface ».

La connexion entre l'interface et l'atelier s'obtient par la relation type « *implement* ».

La représentation interne montre les ressources directement utilisées par l'interface, en général : les deux machines, élémentaire et ensembliste, centrées sur la notion principale de l'atelier.

Figure PxM-40_26. Représentation externe de l'interface d'un atelier

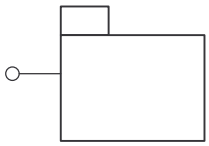
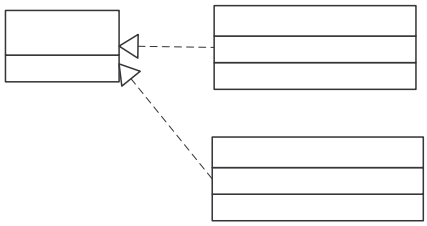


Figure PxM-40_27. Représentation interne de l'interface d'un atelier



Termes et représentations de l'aspect logique pour SOA (suite)

Les automates à états

Définition

Un automate à états contraint le fonctionnement d'une machine logique en clarifiant les étapes qu'elle doit respecter. Il contribue à établir les contrats de services.

Origine

Les automates des modèles amont sont repris tels quels dans les machines qui traduisent les classes auxquelles ils sont attachés.

Le concepteur logique ajoute de nouveaux automates au niveau logique :

- soit pour piloter de nouveaux dispositifs (fonctions transverses) ;
- soit pour traduire la logique procédurale qu'il trouve dans le modèle pragmatique (ordonnancement des transactions, par exemple).

Typologie

Aucune.

Utilisation

Le recours aux machines à états oriente la modélisation logique dans un style particulier ; il renforce la métaphore des *machines* logiques et incite à penser leur fonctionnement comme un rapport stimulus-réponse. Cette approche permet de concilier la souplesse et la robustesse du système, bien mieux qu'une approche fonctionnaliste classique.

Les automates seront convertis lors du passage au logiciel, en fonction des préconisations ou dispositifs élaborés par l'architecte technique.

Caractéristiques

Un automate à états (ou machine à états) comporte des états, des transitions, des événements et des activités, dont des liens avec les opérations activées.

Documentation

Formalisation L'automate est un élément de modélisation en soi, défini dans le méta-modèle UML (*state machine*). Sa sémantique est très riche.

Définition Sauf cas exceptionnel, une machine logique ne dispose que d'un automate. Dans le cas contraire, il serait nécessaire de bien expliquer la fonction de chaque automate (et de vérifier qu'il n'y a pas chevauchement).

Description Les états doivent être définis rigoureusement, c'est-à-dire dans les termes des autres éléments du modèle (surtout, état interne de la machine : information disponible, objets connectés…).

Représentation Le diagramme d'états (plusieurs diagrammes peuvent illustrer un même automate).

Termes et représentations de l'aspect logique pour SOA (suite)

Les structures de données

Définition

La fonction d'une machine logique est de contrôler une portion cohérente d'information du système. Cette portion est circonscrite dans les catégories des modèles amont.

Toute machine logique dispose, donc, d'une structure de données qu'elle anime.

Origine

- Pour les MLM, la structure de données dérive – en gros – de la liste des attributs (nous devrions dire : des propriétés informatives). Cela comprend les attributs dérivés (au sens d'UML).
- Pour les MLO, la structure de données traduit le contexte d'utilisation propre au cas d'utilisation. S'y empilent les informations portées par les multiples objets que le cas d'utilisation manipule, à quoi s'ajoutent des informations contextuelles (identifiant de l'acteur, par exemple).

Typologie

Aucune.

Utilisation

La structure de données de la machine intervient dans son fonctionnement et dans sa communication avec son environnement. Sa présence permet de réduire la signature des services aux informations reçues de l'extérieur et à celles produites par le service.

Les structures de données fournissent la description formelle des flux dans le système.

Caractéristiques

Par combinaison des propriétés de visibilité et de persistance sur les attributs, la machine logique dispose de trois structures distinctes, un même attribut pouvant être inscrit sur une, deux ou trois structures :

- La structure apparente est constituée des données que la machine expose à ses utilisateurs (attributs publics).
- La structure sous-jacente est l'ensemble des données dont dispose la machine pendant son exécution et qui, donc, sont immédiatement disponibles pour les services de la machine (attributs quelle que soit la visibilité).
- La structure persistante ne retient que les attributs que la machine conserve, d'une exécution à l'autre (attributs marqués par l'annotation de persistance, quelle que soit la visibilité[36]).

Documentation

Formalisation

Il y a deux façons de traiter les structures de données dans le modèle logique :

1. La structure de données n'est rien d'autre que la liste des attributs sur la classe qui représente la machine logique.

[36] Au niveau logique, on admet qu'un attribut dérivé puisse être considéré comme persistant, pour des raisons de performance. Un exemple d'attribut appartenant à la structure de données persistante mais pas aux autres est la date d'enregistrement (si elle n'a pas de valeur fonctionnelle).

2. La structure de données est arrachée de la machine et isolée sous la forme d'un type complexe.

Le choix entre ces deux options détermine grandement le style d'architecture. La première option reste inspirée de l'approche objet. Elle se défend si on voit le contenu des ateliers comme une portion d'un modèle objet[37]. Au contraire, si on souhaite bien marquer que l'architecture de service n'est pas la reprise du modèle objet, la deuxième option est meilleure. Elle matérialise le fait que les machines sont « *stateless* » (sans mémoire). La machine n'est pas l'objet. L'objet est réduit, dans l'architecture logique, au flux, décrit par la structure de données. La machine reçoit le flux et lui applique les transformations demandées. Cette façon de voir a la faveur des spécialistes de SOA.

Dans un cas comme dans l'autre, les données sont décrites sous la forme d'attributs. On utilise :

- la visibilité (publique, protégée ou privée),
- la dérivation (coche « *dynamic dependency* » sur la boîte de dialogue de l'attribut, dans Objecteering),
- l'annotation « *persistence*[38] » (*tagged value*).

Définition Seuls les attributs créés de toute pièce au niveau logique reçoivent une définition. Pour les autres, la chaîne de traçabilité permet de retrouver la définition des attributs correspondants, dans les modèles amont.

Description La description des données est traitée p. 70, à la fois pour les structures et les paramètres.

Représentation Dans l'option des types complexes, il n'est pas possible, en respectant le standard UML, d'établir des associations entre les types. Ceci renforce le style. Le fait qu'un type fait référence à un autre (par imbrication), se représente par une dépendance.

Décision Dans l'option des types complexes, se pose la question de leur localisation dans l'architecture. C'est une vraie décision d'architecture logique, à discuter lors de la négociation logique/technique car on doit vérifier la faisabilité.

[37] Dans la solution de l'agglomération (voir p. 10).

[38] Orthographe anglaise.

Termes et représentations de l'aspect logique pour SOA (suite)

Les relations dans l'architecture logique

Maîtriser le couplage

Tout système est nécessairement couplé : sans couplage ce n'est plus un système, il ne fonctionne plus ! Toute la question est de réduire le couplage, non pas à néant, mais à un optimum qui concilie à la fois :

- le fonctionnement harmonieux du système ;
- sa compréhension naturelle ;
- son comportement à long terme (ses évolutions).

L'élaboration de l'architecture logique est le moment clef pour cet exercice.

Les types de relations architecturales appliqués aux ateliers logiques

Le standard UML fixe trois types de décisions d'architecture : la relation d'utilisation (ou dépendance), l'importation et la généralisation. Notre approche d'architecture logique les utilise en les appliquant principalement aux ateliers logiques. Cette restriction vise à simplifier l'architecture et découle assez naturellement des options prises jusqu'à présent :

- La stratification structure déjà fortement le système, en définissant des zones nettement différenciées (les strates).
- Ayant postulé le principe de l'encapsulation des machines, l'unité de la machine logique se trouve exclue des grands choix de structuration (on ne voit que les ateliers).
- La définition de l'atelier logique comme unité de déploiement réduit le poids de la fabrique logique dans la structuration.

L'utilisation Le type le plus courant de relation entre les composants logiques est la relation d'utilisation. C'est une dépendance fonctionnelle qui se réalise à l'exécution par appel de services. Elle est soigneusement documentée pour permettre l'évaluation du couplage et pour guider les décisions d'architecture et de déploiement.

Elle est la seule relation qui s'applique à tous les niveaux de l'architecture logique. Elle se représente par une flèche en pointillés.

L'importation Quand les relations entre deux ateliers sont plus étroites, que l'atelier demandeur n'aurait aucun sens sans le fonctionnement de l'atelier fournisseur ou que l'atelier fournisseur a un rôle d'esclave par rapport au demandeur, alors l'architecte peut renforcer la liaison entre les deux ateliers en choisissant de la typer « *import* » (mot réservé en UML et appliqué à la dépendance).

L'importation débouchera, au niveau logiciel, sur une duplication maîtrisée. L'atelier logique reste unique dans l'architecture logique ; le composant logiciel qui le traduit sera dupliqué. Cette solution augmente l'autonomie des composants et améliore les performances.

Un bon exemple est la relation entre les ateliers principaux et les ateliers utilitaires : codification, gestion des messages… Ces fonctionnalités secondaires, d'usage fréquent et de portée locale, peuvent être incorporées dans les autres ateliers.

La généralisation La généralisation est une relation particulière dont l'intérêt peut jaillir quand on considère les partages possibles entre les systèmes partenaires. C'est une relation structurelle – comme l'importation – par laquelle un atelier se définit comme un enrichissement, une extension d'un autre.

Mode d'emploi

Toutes les relations qui existent dans les modèles amont doivent être ramenées, au niveau logique, à l'une ou l'autre de ces trois relations.

Au niveau des machines

Les machines ne sont liées que par des relations d'utilisation, résumant les appels de services. Les machines demandent des services par les deux canaux suivants :

- soit directement aux autres ML, mais uniquement à l'intérieur de l'atelier ;
- soit de l'extérieur (autre atelier) : uniquement par l'intermédiaire des services d'interface.

Au niveau des ateliers

Entre les ateliers, les relations d'utilisation résument les relations qui s'instaurent entre leurs machines constituantes. L'architecte peut décider de transformer ces relations, quand elles sont fortes, en relations structurelles : importation ou généralisation. Ces décisions doivent être argumentées.

Au niveau des fabriques

La relation d'utilisation (fonctionnelle) entre fabriques logiques ne fait que résumer les dépendances aux niveaux inférieurs. De façon fort saine, l'outil UML impose de poser d'abord les relations entre les agrégats pour pouvoir coupler les unités des niveaux inférieurs.

Précautions dans le cas des relations structurelles

Les deux derniers types portent sur les relations structurelles, relations qui précisent la composition des ateliers logiques.

L'importation améliore les performances mais introduit les risques liés à la redondance dans le système. C'est une décision de l'architecte logique en concertation avec l'architecte technique. Ce dernier conseille sur les questions de performance et de déploiement.

Le recours à ces types de relations soulève une question : qu'en est-il des données ? le support est-il unique ou doit-il être dupliqué et localisé à l'instar du code ?

Les deux cas de figure sont envisageables.

Importation avec données

Pour des données utilitaires, par exemple les codifications, il est préférable de doter l'atelier de son propre support de données : l'importation se traduit alors par la duplication complète de l'atelier, données et services. Dans le cas des codifications cette solution présente l'avantage de placer la signification des codes au même endroit que les données métiers. Outre qu'elle renforce l'autonomie de l'atelier, elle favorise le contrôle d'intégrité, lequel peut être confié au SGBD[39].

Importation du code seul

Pour des ateliers à valeur plus fonctionnelle, les services peuvent être importés pour améliorer les performances mais les données restent concentrées en un seul endroit. Une telle solution pourrait convenir dans le cas de l'atelier AL_Flux.

Cette importation partielle – services sans les données – n'est permise qu'à l'intérieur d'une même fabrique logique (voir le point sur « L'architecture des données »).

L'autonomie absolue des ateliers logiques

La règle de l'autonomie des ateliers exprime un idéal pour l'architecture logique, idéal inspiré plus par l'approche par composants que par l'approche orientée services.

Cette règle voudrait que les ateliers fonctionnent de façon autonome, sans aucun besoin vis-à-vis d'autres ateliers.

[39] Le recours aux classes *templates* permet d'isoler le nom de la table dans la définition de la machine et la réalisation des services. Ce point sera détaillé dans la conception de la solution pour les codifications.

Avantages Nous ne la pratiquons pas strictement, mais seulement comme une orientation tendancielle, motivée par le souci de réduire le couplage entre les ateliers et de faciliter le déploiement des services.

Inconvénients Rendre l'atelier complètement autonome, c'est lui interdire tout accès à l'environnement, donc toute demande d'information. En conséquence, l'information dont il a besoin pour assurer ses missions ne peut provenir que d'une des sources suivantes :

- les supports de données que l'atelier encapsule ;
- les paramètres d'entrée inscrits sur son interface.

Ce dernier point conduit à reporter le couplage à l'extérieur, d'où une moindre réutilisation et un éclatement de la logique métier.

Procédé Malgré ces inconvénients, le gain conséquent pour le système justifie l'effort d'optimisation. En tentant cette réduction, le concepteur logique respecte les consignes suivantes :

- L'optimisation est patente quand elle aboutit à retirer une relation entre deux ateliers[40]. Pour plus de clarté, on distinguera l'optimisation fonctionnelle (réduction du nombre d'appels de services) et l'optimisation structurelle (réduction des relations entre ateliers).
- L'intérêt de l'optimisation est plus grand quand on réduit le couplage entre des fabriques logiques, plutôt qu'à l'intérieur d'une fabrique.
- Les relations d'utilisation peuvent être supprimées uniquement dans le cas d'une demande d'information, jamais dans le cas d'une transformation déclenchée sur un autre atelier.
- Quand le concepteur supprime une relation d'utilisation (un appel de service), il la remplace par un passage de paramètre. Le paramètre se loge dans la signature du service demandeur, ainsi que sur le service d'atelier qui le relaye.

[40] Soit un atelier A, à l'intérieur duquel les machines invoquent plusieurs services d'un atelier B. Si l'optimisation ne permet de retirer qu'une partie des appels, alors les ateliers restent couplés. On n'a pas réduit le couplage structurel et on n'a rien changé à la configuration : les contraintes de déploiement sont les mêmes. Le seul gain est pour les performances, à l'exécution. Il faut pouvoir retirer tous les appels de A vers B pour casser la relation résumée entre les deux ateliers. C'est à cette condition seulement que A conquiert son autonomie par rapport à B.

Termes et représentations de l'aspect logique pour SOA (suite)

L'imbrication des machines

Le principe

Nos règles d'architecture font de l'atelier l'unité principale pour déduire l'espace de nommage. À l'intérieur d'un atelier, la visibilité entre les machines peut encore être réduite.

À cette fin, nous ne nous autorisons pas les sous-paquetages : UML permet autant de niveaux de paquetages que l'on souhaite. L'architecture de services selon Praxeme – comme dans toutes les méthodes d'architecture logique depuis les années 80 – s'impose au contraire un nombre réduit de niveaux d'agrégation. Le paquetage n'est utilisé qu'à deux niveaux : la fabrique et l'atelier.

Reste une possibilité pour réduire la visibilité à l'intérieur de l'atelier : l'imbrication des classes. Une machine logique considérée comme esclave d'une autre sera représenté comme une classe imbriquée.

Définition

Une machine imbriquée est une machine qui n'offre ses services qu'à la machine propriétaire.

Origine

Les modèles amont peuvent déjà avoir recours à cette notion. Il faut vérifier, alors, que le modélisateur l'a bien employée dans le sens d'une restriction de la visibilité, et non simplement comme un artifice pour mettre de l'ordre dans son modèle.

Le plus souvent, c'est au concepteur logique que revient la décision de réduire la complexité perçue de l'atelier en cachant ainsi des machines secondaires derrière des machines principales.

Utilisation

Cette solution est appréciable pour réduire la complexité du système. Évidemment, les services des machines imbriquées ne pourront pas être directement exposés par l'atelier.

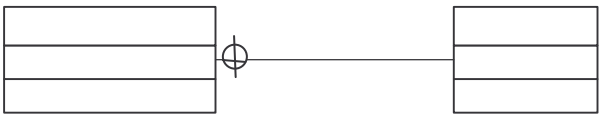
L'imbrication est réservée aux cas où les services d'une machine n'ont d'intérêt que pour une seule autre machine.

Documentation

Formalisation La classe représentant la machine esclave est définie « sous » la classe de la machine propriétaire, dans la hiérarchie des classes.

Représentation UML prévoit un symbole spécial pour l'imbrication (une croix dans un cercle, côté propriétaire), mais tous les AGL ne le supporte pas.

Figure PxM-40_28. Représentation de la relation d'imbrication



Termes et représentations de l'aspect logique pour SOA (suite)

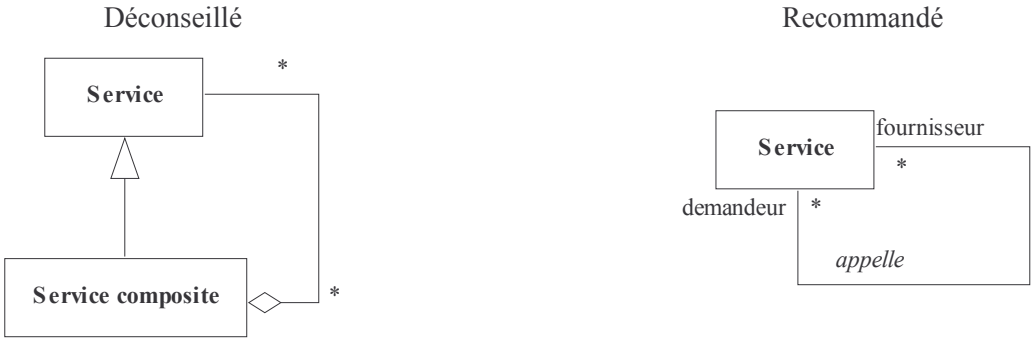
La typologie des services

Avertissement

Nous observons une dérive fréquente : les spécialistes élaborent des typologies compliquées, cherchant à identifier le maximum de types comme si leur nombre indiquait l'expertise. Ces typologies sont, souvent, de nature purement formelle car elles sont issues de généralisations théoriques et non de pratiques réelles. Très vite, ces constructions se mettent à croiser plusieurs critères, par exemple celui du niveau (la strate) et le fait qu'un service en appelle d'autres (service « composite »). Alors, la typologie devient instable et alimente des débats scholastiques à n'en plus finir.

Nous éviterons les complications inutiles et nous contenterons des termes indispensables. L'effort de méta-modélisation nous y aide. Comme le préconise le procédé de modélisation sémantique, nous ne retenons un concept que s'il est doté d'une sémantique propre et de comportements qui le distinguent des autres concepts ou objets. La figure ci-dessous donne un exemple de la simplification. Plutôt que de « réifier » la notion de composition en ajoutant une méta-classe (diagramme de gauche), nous préférons la traiter par la relation (diagramme de droite).

Figure PxM-40_29. Exemple de simplification appliquée à la typologie des services



La question à se poser avant d'introduire un nouveau terme ou un nouveau concept est : « À quoi ça sert ? ». Une typologie inutilement sophistiquée risque d'embrouiller les praticiens et de soulever plus de questions qu'elle n'en résout. Sur le thème de SOA, de prétendus méta-modèles – qui ne sont pas même des modèles, tout juste des diagrammes de classes – inscrivent de nombreux termes : ce n'est pas un service à rendre aux concepteurs. La typologie ne doit retenir une nouvelle catégorie de services que quand les comportements et contraintes changent selon la nature du service ou quand on est amené à définir des règles de dérivation différentes vers le logiciel.

Le vocabulaire

C'est la pratique qui, à la longue, crée un langage.

On parlera, par exemple :

- des services internes et des services externes, selon leur position dans l'architecture, respectivement : strate « Métier » ou strate « Organisation » ;
- des services d'interface, pour les services proposés par les ateliers, via leurs interfaces ;
- des services de sélection, d'action, de transformation, etc. pour caractériser leur fonction ;
- des services transactionnels quand ils couvrent une transaction.

Dans ces exemples, seul le dernier mérite une mention spéciale dans un profil UML. On lui associera, par exemple, une annotation (*tagged value*) qui fera référence au dispositif de gestion des transactions. Pour autant, il n'est pas nécessaire d'introduire une méta-classe : l'annotation correspond à un attribut sur la méta-classe Service.

Termes et représentations de l'aspect logique pour SOA (suite)

Le méta-modèle de l'aspect logique

La nécessité d'un méta-modèle

Établir un méta-modèle, c'est mettre de l'ordre dans la terminologie et les catégories de perception que nous utilisons. C'est donc un exercice d'hygiène intellectuel. Il a plusieurs retombées :

1. tout d'abord, pédagogiques : il rassemble et ordonne le vocabulaire utilisé, montrant les liens entre les notions ;
2. ensuite, pratiques : il permettra aux concepteurs de s'exprimer rigoureusement et de poser les bonnes questions ;
3. enfin, techniques : le méta-modèle est le modèle de l'outillage ; en tant que tel, il sera traduit sous la forme d'un profil UML qui permettra d'injecter la méthode dans l'outil UML.

La forme du méta-modèle de Praxeme

Le méta-modèle de Praxeme couvre tous les aspects de la Topologie du Système Entreprise. Ce fait n'est pas anodin car il autorise un travail fondamental : montrer les relations entre les notions d'aspects différents. Ces relations sont le support des règles de dérivation. Elles jouent un rôle considérable dans le procédé de conception SOA, selon Praxeme.

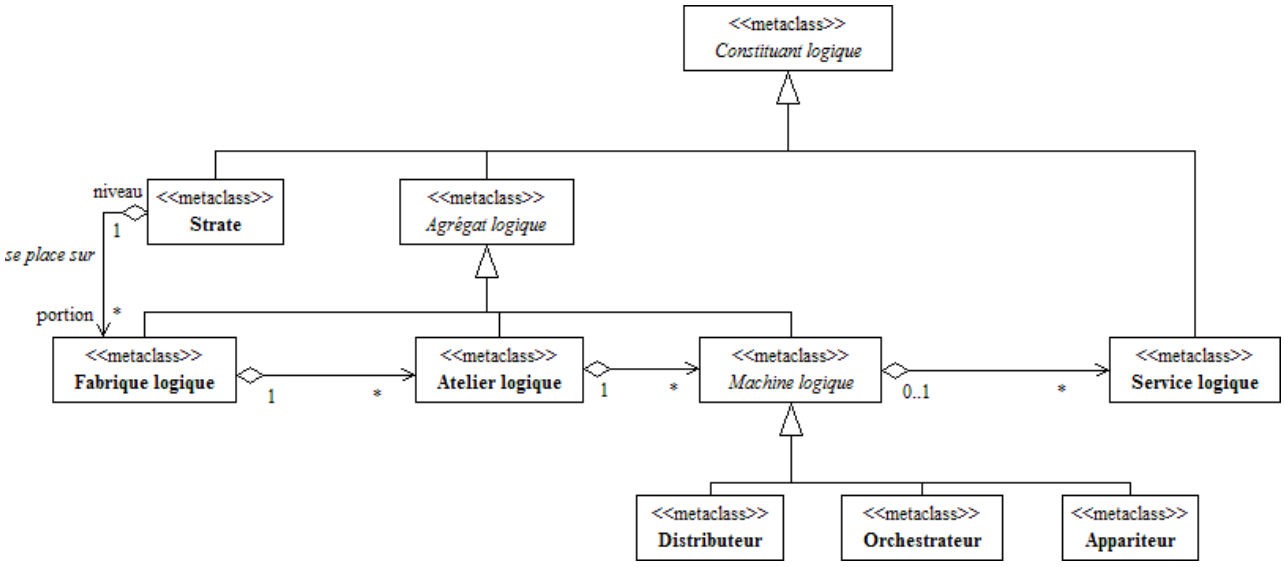
Le méta-modèle de Praxeme est un vrai modèle, c'est-à-dire qu'il est beaucoup plus qu'un simple diagramme de classes et qu'il contient les définitions, les commentaires et les justifications des décisions de modélisation[41].

Le méta-modèle est généré en hypertexte, consultable sur le site Praxeme.

Un extrait

Le diagramme de classes, ci-dessous, récapitule les principales notions de la modélisation logique, dans le style SOA. Il se limite à la statique mais le méta-modèle montre aussi les relations de dérivation avec les aspects sémantique et pragmatique.

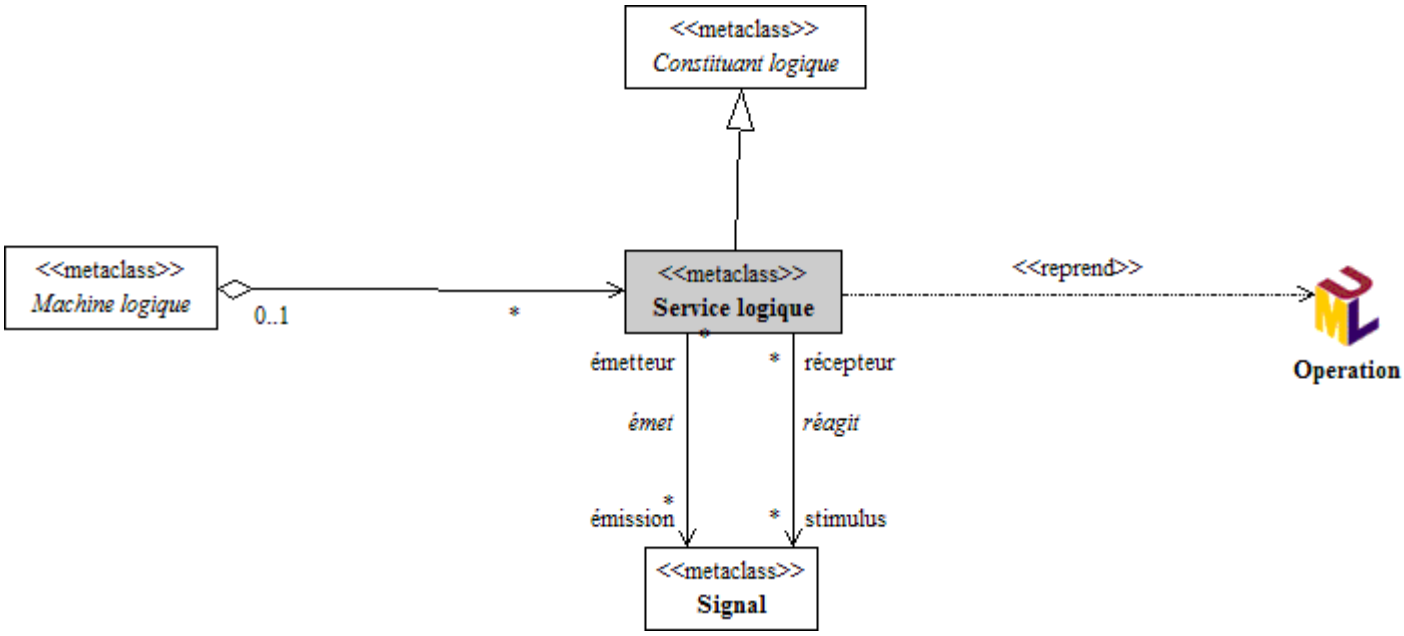
Figure PxM-40_30. Un extrait du méta-modèle Praxeme pour l'aspect logique dans le style SOA



[41] Au moment où nous publions le présent guide (version 1.0), un nouveau chantier débute qui enrichira le méta-modèle.

Chaque notion est définie et fait l'objet d'un diagramme de classes qui montre toutes ses relations. La figure suivante en donne un exemple.

Figure PxM-40_31. Le diagramme de classes montrant les relations de la méta-classe Service, selon le méta-modèle Praxeme



On voit que le méta-modèle Praxeme s'appuie sur le méta-modèle UML. C'est le sens du stéréotype « reprend » sur la dépendance qui pointe vers la méta-classe d'UML. On comprend que le service est exprimé par une opération, au sens d'UML. Cette façon de « méta-modéliser » présente deux avantages :

1. D'une part, elle évite de compliquer le méta-modèle Praxeme qui est comme une surcouche du méta-modèle UML.
2. D'autre part, elle indique précisément comment bâtir le profil UML Praxeme.

En l'occurrence, le profil Praxeme pour SOA contient un stéréotype « Service » que le concepteur peut associer à une opération, en utilisant un outil UML.

Démarche de conception des services

Vue générale

L'origine des éléments logiques

Le principe de la méthodologie Praxeme est la dérivation des modèles aval à partir des modèles amont. Les règles de dérivation donnent le détail de ces transformations. Elles permettent d'obtenir des services à fort contenu métier, puisque traduisant directement la connaissance de l'activité. De plus, les exigences de la modélisation sémantique font que le modèle source possède des qualités formelles, telles que l'économie de l'expression ou la localisation des règles de gestion.

L'origine de la strate « Métier »

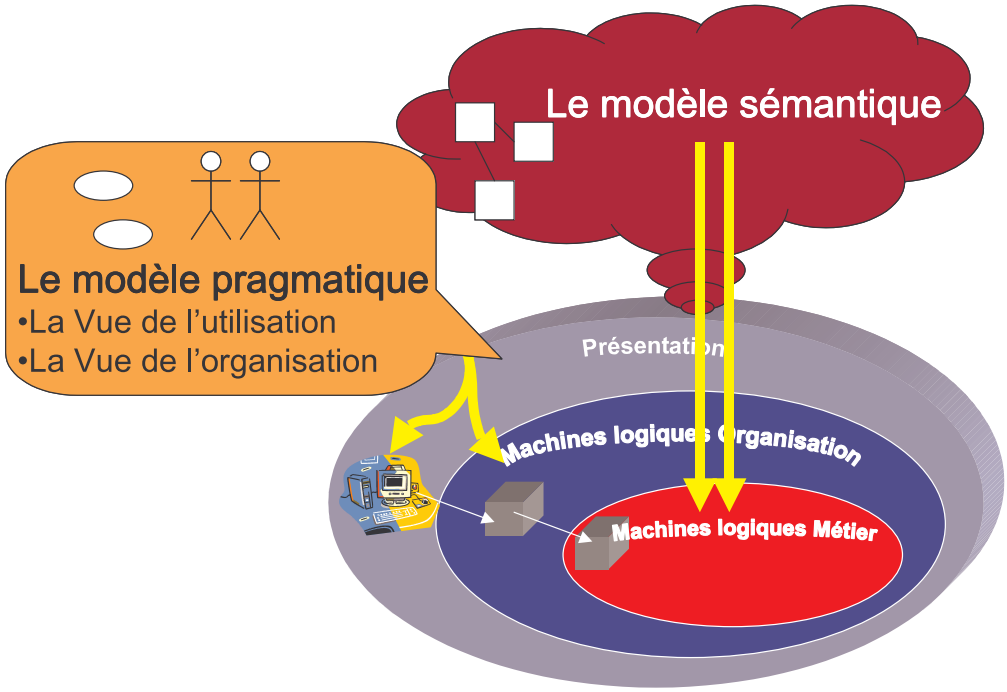
La strate « Métier » prend sa source dans le modèle sémantique.

Elle reprend les domaines d'objets et introduit des décisions de structuration propres à l'aspect logique. C'est dire que cette strate demande un effort de structuration en complément de la dérivation « mécanique ».

L'origine de la strate « Organisation »

Les éléments logiques de la strate « Organisation » dérivent directement de la description de l'activité sous la forme de cas d'utilisation. Il est à noter qu'une autre dérivation est envisageable, à partir de la Vue de l'organisation. Cette dernière se caractérise par les processus organisationnels et exprime le point de vue de l'organisateur, alors que les cas d'utilisation sont l'unité de base de la Vue d'utilisation, de portée locale.

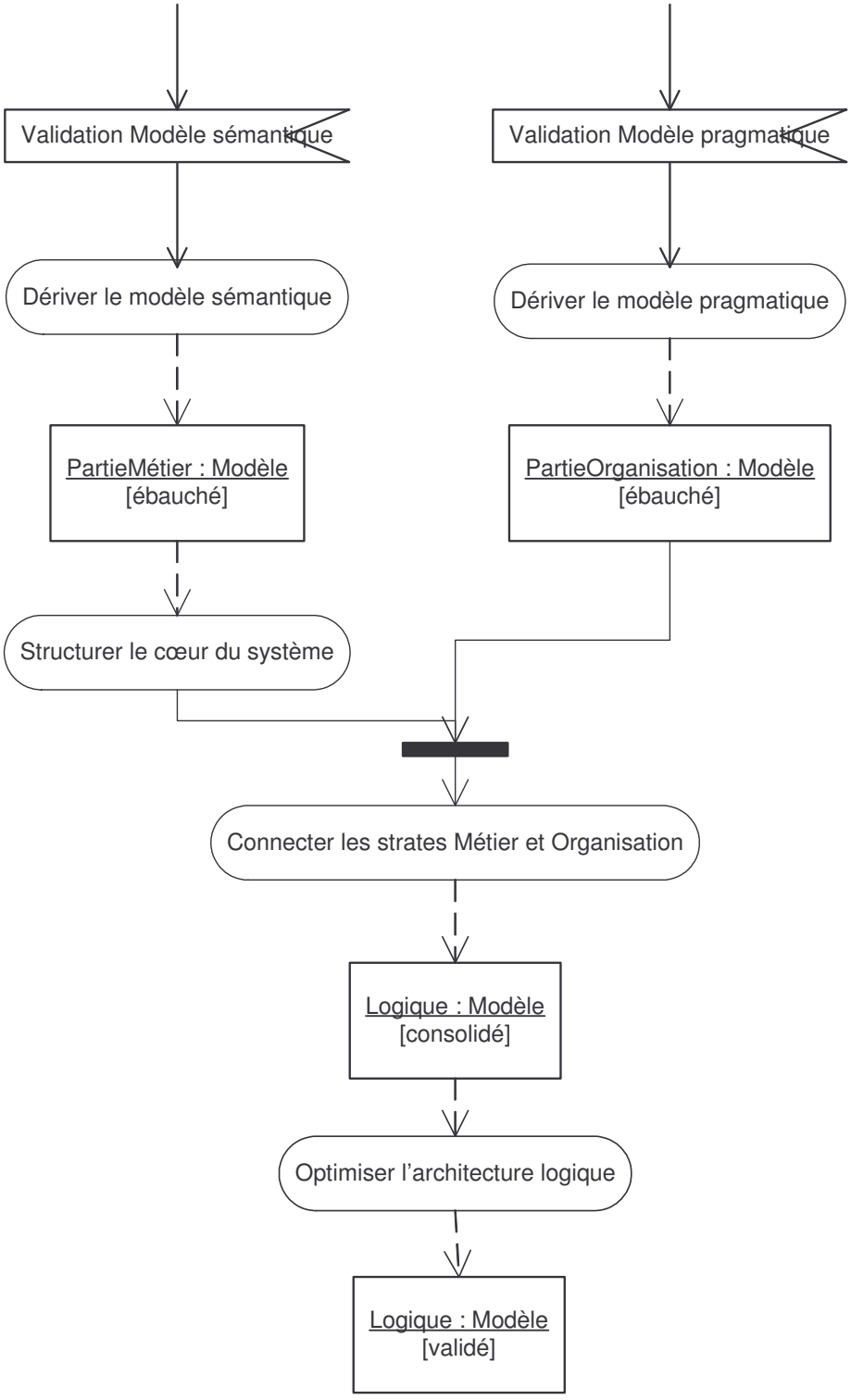
Figure PxM-40_32. La dérivation des modèles amont en architecture logique



Le processus

La figure ci-dessous propose une organisation du travail pour l'élaboration de l'architecture de services. Cette organisation exploite la possibilité de parallélisme, fondée sur la Topologie du Système Entreprise.

Figure PxM-40_33. Les travaux pour élaborer le modèle logique



Démarche de conception des services (suite)

Quelques repères

Avant de détailler les étapes du processus général proposé ci-dessus, nous rappelons quelques recommandations relatives à la conduite des projets, selon Praxeme.

Le processus de construction

Le guide PxM-40, sur l'aspect logique, ne fait qu'évoquer au passage la dimension du processus mais ne contient pas la méthode pour conduire les activités relatives à SOA. Il s'intéresse, avant tout, à la nature de l'aspect logique (dimension du Produit) et aux disciplines de la conception logique et de l'architecture logique (dimension des Procédés). Le document PxM-03 traite, plus spécifiquement, de la dimension du Processus[42].

Dans l'ensemble des activités de construction ou de transformation du Système Entreprise, SOA s'insinue dans les activités de conception du système informatique et de développement.

La nouvelle dynamique

SOA provoque un grand changement dans les pratiques des directions informatiques. En effet, depuis longtemps, les pratiques de développement sont largement dominées par un des modes de production : le mode projet. L'essentiel des ressources pour le développement des systèmes est mobilisé selon ce mode d'organisation, c'est-à-dire sous la forme d'équipes temporaires, rassemblées pour un objectif précis et daté : livrer une application. Sans remplacer ce mode de production qui a montré ses vertus, SOA oblige à le contrebalancer en renforçant les activités « transverses », seules aptes à porter les objectifs long terme et la vision du système à construire.

La visée architecturale que suppose SOA, dans le plein sens de cette appellation, conduit donc à mettre en place une « nouvelle dynamique » et à articuler différemment les projets et les activités transverses. Faute d'une telle révision dans l'organisation des activités de la DSI, les vœux de réutilisation et d'urbanisation du SI resteront lettre morte.

Le travail en amont

Les procédés de conception des services, décrits dans ce guide, exploitent les modèles sémantiques et les modèles pragmatiques[43]. Le secret de la conception logique réside principalement dans la dérivation de ces modèles. Le processus de construction doit donc ménager une juste place à la modélisation « amont » ou modélisation « métier ». Cette réhabilitation de la modélisation en amont de l'informatique offre l'opportunité d'une nouvelle donne et d'un nouveau rapport avec les acteurs des métiers de l'entreprise.

Dans l'enthousiasme des départs, le lancement des projets SOA présente le risque de négliger les modèles amont pour arriver plus vite aux premiers services. Le cas échéant, le projet produit effectivement des services dont on peut assurer la promotion, mais on n'améliore pas forcément la qualité du système. De deux choses l'une :

- soit l'effort de modélisation est reporté sur la conception logique et confié à des ressources qui n'en ont pas forcément la compétence ;
- soit les services ainsi produits ont un contenu « métier » limité et ne sont rien d'autre que des classiques accesseurs.

---

[42] Sur les trois dimensions de la méthodologie, voir « Praxeme : le Livre blanc », référence « SLB-02 ».

[43] Pour la définition de ces modèles, voir le Guide général, référence « PxM-02 ».

On n'insistera jamais assez sur la nécessité de la modélisation « amont ».

La négociation logique/technique

La montée en charge de la conception logique détaillée ne peut commencer qu'après le moment fatidique de la négociation logique/technique[44]. La démarche des projets et le processus de construction du système doivent impérativement tenir compte de ce pré-requis. Sinon le détail des décisions, du moins les grandes lignes de la négociation doivent avoir été arrêtées en préalable à la conception logique, à grande échelle. Ne pas respecter cette condition entraîne des gaspillages.

Dans le cadre de la nouvelle dynamique, la négociation logique/technique devrait se mener, pour l'essentiel, au niveau du système plutôt que localement pour chaque projet. Elle requiert la participation des architectes logiques et techniques, avec un haut niveau de compétence et une bonne compréhension des enjeux. La négociation suppose un positionnement équilibré des deux métiers d'architecture de SI : architecture logique et architecture technique.

[44] Voir définition p. 17.

Démarche de conception des services (suite)

Étape 1 : « Dériver le modèle sémantique »

Ligne directrice

La dérivation du modèle sémantique applique les règles données dans la synthèse (p. 74).

Le travail de structuration logique range, sur la strate « Métier », les machines logiques « métier » dans des ateliers logiques.

Actions

Les actions du concepteur logique se déroulent comme suit :

- Les domaines d'objets deviennent des fabriques logiques. Ceci peut s'obtenir par copie des paquetages du modèle sémantique vers le modèle logique. Les paquetages obtenus sont stéréotypés « Fabrique logique ».
- Les classes sont dérivées en une ou deux machines logiques (élémentaire et ensembliste).
- Les machines logiques sont rangées dans des ateliers logiques qui reflètent les grappes de forte cohérence, trouvées sur le modèle sémantique. Cette action peut être menée avant ou en parallèle à la précédente (voir détail p. 81).
- Les dépendances entre les machines sont dessinées, à partir des besoins pour le fonctionnement des services. Le concepteur utilise le diagramme de classes sur lequel il porte exclusivement des relations d'utilisation (ou dépendances). Dans le style radical présenté p. 10, les associations n'ont plus cours dans le modèle logique. On obtient ainsi un modèle logique applicable quelle que soit la technologie.

Résultat

À ce stade, on obtient une première version du modèle logique pour la strate « Métier ». Les fabriques, ateliers, machines et services sont recensés et déjà reliés. Ils ne sont pas encore complètement documentés, en tout cas pas les services. En effet, on attend l'étape suivante avant de fixer la signature des services et leur algorithme. En revanche, le concepteur logique peut exprimer les missions (définitions, contrats) des machines, ateliers et fabriques.

Démarche de conception des services (suite)

Étape 2 : « Structurer le cœur du système »

Ligne directrice

La dérivation mécanique, appliquée lors de la première étape, transporte, dans le modèle logique, le couplage lié au mode de la propagation[45]. On obtient ainsi un style d'architecture marqué largement par la propagation : les constituants logiques communiquent autant que les objets coopèrent. La deuxième étape a pour souci de réduire le couplage entre les ateliers de la strate « Métier ».

Actions

- Décomposer les fabriques logiques en ateliers logiques.
- À partir des dépendances entre services, faire le bilan du couplage entre ateliers logiques.
- Choisir la nature de la relation entre les ateliers : utilisation ou importation (voir ci-dessous).
- Dans le cas des relations d'utilisation, pour chaque paire d'ateliers couplés, étudier les possibilités de supprimer le couplage. Cette étude procède en commençant par les ateliers les plus distants (voir, ci-dessous, la définition de la distance).
- Quand le couplage entre deux ateliers est supprimé, remplacer l'appel du service par un passage de paramètres.

Résultat

Le travail de cette étape est un travail d'architecture pure, c'est-à-dire qu'il porte exclusivement sur la structuration du système. Il intervient sur le modèle logique produit lors de l'étape précédente, modifie les relations et retouche la signature des services. La valeur ajoutée est la réduction du couplage, qu'il est intéressant de mesurer. Cette mesure fait partie des résultats de l'étape, au même titre que les justifications des décisions d'architecture.

Analyse du couplage

Le type de relation, la distance entre ateliers et la force du couplage caractérisent l'architecture logique à un moment donné et permettent d'éclairer les décisions de structuration. Ces notions sont définies ci-après.

Type de la relation

Par défaut, c'est-à-dire après dérivation, la relation entre les ateliers logiques est de type « **utilisation** ». Elle se représente par une flèche simple, en pointillés. Dans cette relation, l'atelier demandeur envoie une ou des demandes de services vers l'atelier fournisseur et en reçoit des réponses.

Quand les relations entre les ateliers sont plus étroites, au point que l'on aurait du mal à concevoir le fonctionnement de l'atelier demandeur indépendamment de l'atelier demandé, l'architecte peut décider de lier les ateliers par une relation **d'importation**. Elle se représente comme une relation d'utilisation, mais stéréotypée avec le mot réservé « *import* ».

L'importation est réservée aux cas de couplage fonctionnel fort. L'atelier fournisseur reste distinct pour des raisons de réutilisation (il sera appelé par d'autres ateliers demandeurs). Un exemple est fourni par la dépendance entre l'atelier AL_Sinistre et l'atelier AL_Flux. Dans le cas de l'importation, il faut préciser sur la relation si elle concerne uniquement les services ou à la fois les données et les services (éventuellement, pour des services utilitaires tels que Message ou Codification).

[45] Voir discussion p. 12.

Distance La distance entre deux ateliers couplés A et B est une relation ordinale définie comme suit (dans l'ordre croissant) :

1. A et B sont directement couplés, à l'intérieur d'une même fabrique.
2. A et B appartiennent à la même fabrique et A est relié à B par plusieurs autres chemins (par l'intermédiaire d'autres ateliers qu'il appelle).
3. A et B appartiennent à deux fabriques différentes.

Force Le couplage entre deux ateliers est d'autant plus fort que les appels sont nombreux, de l'un à l'autre. La force du couplage présente deux aspects :

- structurel (statique) : on s'intéresse aux types d'appels (nombre de services différents appelés et nombre de services appelants) ;
- fonctionnel (dynamique) : on prend en compte les fréquences d'appel des mêmes services.

Démarche de conception des services (suite)

Étape 3 : « Dériver le modèle pragmatique »

Ligne directrice

La dérivation ne peut appliquer les règles formulées p. 76 que si la Vue de l'utilisation a été structurée rigoureusement[46] et qu'elle a banni la redondance. Uniquement dans ce cas, la structure de la strate « Organisation » peut décalquer celle du diagramme des cas d'utilisation, en garantissant une bonne qualité structurelle.

Autrement dit, la qualité de l'architecture logique pour cette strate repose sur l'effort de structuration de l'aspect pragmatique. Ceci est une conséquence naturelle du positionnement et de la vocation de la strate « Organisation »[47].

Actions

- Les domaines fonctionnels deviennent les ateliers logiques de la strate « Organisation ».
- Les cas d'utilisation se dérivent en machines logiques « organisation » (MLO). Celles-ci reproduisent les relations existant entre les cas d'utilisation.
- Les activités élémentaires deviennent des services logiques qui se distribuent sur les MLO de la même façon que les activités sont assignées aux cas d'utilisation.
- On ajoute les services transactionnels (ou services d'action).
- La logique interne du cas d'utilisation (logique procédurale ou contraintes d'enchaînement ou d'états) peut, si elle est complexe, se représenter sous la forme d'un automate à états, propre à la MLO. Chaque valeur d'état désigne un jalon dans la procédure ou un palier dans l'alimentation des données. Les scénarios du cas d'utilisation peuvent avoir des logiques contextuelles différentes. En ce cas, l'automate doit permettre de faire tourner ces différentes logiques. C'est pourquoi le diagramme d'états est un meilleur outil que le diagramme d'activité[48].
- Dans les cas plus simples où la logique procédurale reste unie et plutôt linéaire, l'automate à états ne s'impose pas. La procédure peut être décrite soit sous forme graphique (diagramme d'activité), soit sous forme textuelle.

Résultat

Cette étape construit la strate « Organisation ». Puisque la dérivation ne fait que reproduire la structure du modèle pragmatique, il n'y a pas à y revenir.

La documentation de cette strate apporte peu d'informations nouvelles. Elle consiste :
- d'une part, à associer les éléments logiques aux éléments du modèle pragmatique (du domaine fonctionnel à l'activité élémentaire) ;
- d'autre part, à préciser les structures de données et les contextes informationnels, éventuellement pilotés par des automates.

---

[46] Cf. Guide de l'aspect pragmatique, référence « PxM-20 ».

[47] Dans cette version du Guide de l'aspect logique, on ne traite pas la dérivation de la Vue de l'organisation, l'autre partie du modèle pragmatique, basée sur les processus organisationnels.

[48] Normalement, si la modélisation pragmatique a été poussée à son terme, elle a produit les automates à états, inscrits sur les cas d'utilisation. La dérivation du modèle pragmatique absorbe alors ces automates. Le travail du concepteur logique s'en trouve grandement réduit.

Démarche de conception des services (suite)

Étape 4 : « Connecter les strates Organisation et Métier »

Ligne directrice

La démarche Praxeme offre la possibilité de produire les modèles sémantique et pragmatique parallèlement. Si cette option est prise, le point de rencontre se situe, au plus tard, sur l'aspect logique. Le travail consiste à analyser les activités élémentaires, dérivées en services externes, et à détecter les services internes qui leur sont nécessaires.

Le mouvement, dans cette étape, va donc des MLO vers les MLM.

Actions

- Analyser le contenu des services de MLO en détectant les services internes qu'ils doivent demander à la strate « Métier ».
- Éventuellement, formuler des demandes de nouveaux services internes, dans le cas où les services proposés ne satisferaient pas (ces demandes entrent dans un processus d'arbitrage et ne sont tranchées que sous le contrôle de l'urbaniste).
- Inscrire les connexions dans le modèle logique (sous la forme de relations d'utilisation vers les ateliers).

Résultat

Après cette étape, nous disposons d'une architecture logique complète, à l'exclusion de la strate « Présentation ».

Démarche de conception des services (suite)

Étape 5 : « Optimiser l'architecture logique »

Ligne directrice

La conception se poursuit en analysant les informations disponibles au cours des échanges entre constituants logiques. Le but de l'étape est d'optimiser l'architecture logique en tenant compte des contextes d'utilisation. Cela permet de réduire encore le couplage par rapport à l'étape 2.

Actions

- Faire le bilan des flots d'informations et de l'information disponible sur la MLO au moment de l'appel d'un service interne. Cette information se présente sous la forme d'un contexte. La structure du contexte correspond à la structure de données de la MLO. Elle s'enrichit progressivement, au fur et à mesure des interactions entre l'utilisateur et le système (dit en termes logiques : entre les strates « Présentation » et « Organisation »).
- Puisque les services internes répondent plus largement que ce qui leur est demandé[49], ce bilan peut révéler que la MLO dispose déjà d'une information en avance sur la logique procédurale. Dans ce cas, la MLO peut passer, aux ateliers « métier » qu'elles sollicitent, des informations qui éviteront des appels en propagation.
- Les couplages entre les ateliers, dans la strate « Métier », sont alors remplacés par des paramètres supplémentaires dans les services internes et dans les services d'interface. Cet exercice vaut surtout quand il permet de réduire le couplage entre les ateliers de la strate « Métier ».

Résultat

Cette approche est nommée « contextualisation » parce qu'elle examine les contextes d'utilisation (traduction des cas d'utilisation) et les contextes d'information sous le contrôle des MLO. Par cet effort, elle peut, dans certains cas, réduire le couplage dans le cœur du système. La contrepartie est la suivante :

- alourdir les interfaces des ateliers « Métier » ;
- déformer encore un peu la logique métier, plus encore que dans l'étape 2 puisque, ici, on introduit un point de vue externe pour structurer l'interne.

[49] Ce phénomène s'explique par la volonté de réduire le nombre de services et de faire des services réutilisables. Plutôt que de concevoir des services *ad hoc*, à partir des besoins locaux et extérieurs, l'architecture propose des services de plus large portée. Ils émettent davantage d'information que requise par le demandeur, à charge pour celui-ci de prélever ce qui l'intéresse.

Thématique de la conception logique

La visibilité et les interfaces

Rappel

Les valeurs de la visibilité utilisées dans notre modèle logique sont[50] :

- Publique = utilisable par tout point du système ayant un accès explicité par le graphe d'architecture.
- Protégé = accessible par le propriétaire et ses héritiers.
- Privé = sous le contrôle exclusif du propriétaire (visibilité limitée à l'espace de nommage : les machines privées ne sont vues que par les autres machines du même atelier ; les attributs privés ne sont vus que des services de leur machine).

Une machine privée n'est vue que de l'intérieur de son atelier d'appartenance (toutes les autres machines peuvent solliciter ses services publics).

Un service privé n'est vu que de la machine elle-même : seuls les autres services de cette machine peuvent le solliciter.

La déduction des interfaces

En combinant la visibilité des services et des machines à l'intérieur d'un atelier, le concepteur logique déduit les services d'interface que l'atelier doit exposer. Il lui reste à décider de leur répartition éventuelle sur plusieurs interfaces, s'il perçoit des regroupements justifiés par l'usage.

Le tableau ci-dessous éclaire les règles de déduction, indiquant selon quelles combinaisons les services appartiennent à l'interface de l'atelier.

La classe stéréotypée « *interface* » est attachée à l'atelier par une relation « *implement* ». Elle ne voit que les machines publiques de cet atelier. Elle peut être placée à l'intérieur ou à l'extérieur de l'atelier[51].

Sélection des services à exposer dans l'interface de l'atelier

Visibilité des machines logiques	Visibilité des services logiques		
	Publique	Protégée	Privée
Publique	**OUI**	pour les héritiers	NON
Protégée	NON	pour les héritiers	NON
Privée	NON	NON	NON

[50] Voir les définitions appliquées à la machine et au service, pp. 35 et 39.

[51] Discussion : placer les interfaces à l'intérieur des ateliers présente l'avantage d'un rangement plus ordonné (on ne voit que les ateliers à un niveau donné du navigateur) ; en revanche, les interfaces à l'extérieur des ateliers peuvent être réalisées par plusieurs ateliers, ce qui offre plus de souplesse pour l'urbanisation. Un exemple est donné p.100 à propos de AL_Organisation. C'est donc cette solution qui remporte notre faveur.

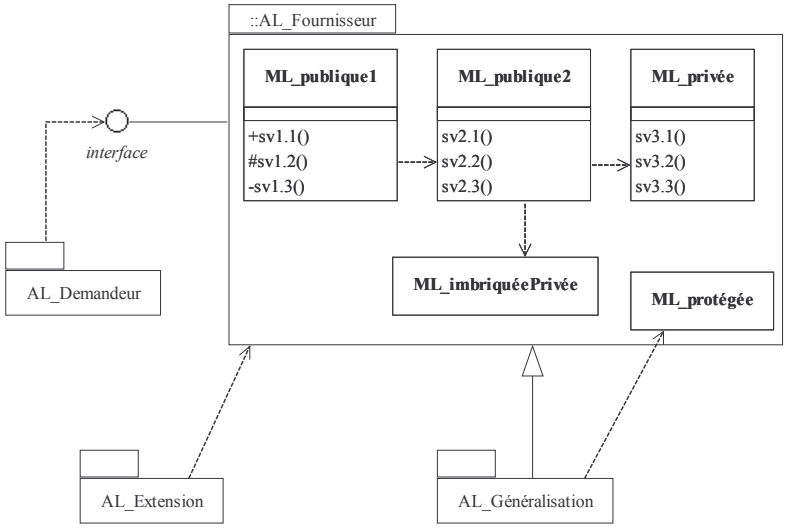
La visibilité à partir des différents points

Le tableau ci-dessous précise les conditions de visibilité à partir de différents points de départ. Ce sont les règles de visibilité fixées par UML.

Praxeme est plus limitatif puisque la règle de l'encapsulation des machines interdit l'accès aux machines à partir de l'extérieur de l'atelier. Un atelier demandeur ne peut solliciter que les services présentés dans une interface.

	Point d'arrivée à l'intérieur de l'atelier logique		
Point de départ	**ML publique**	**ML protégée**	**ML privée**
Atelier externe	NON mais via l'interface de l'atelier	NON	NON
Interface de l'atelier	OUI (uniquement services publics)	NON	NON
Machine de l'atelier	OUI (uniquement services publics)	OUI (uniquement services publics)	OUI (uniquement services publics)
Atelier avec généralisation	OUI (uniquement services publics)	OUI (uniquement services publics et services protégés à partir des machines qui héritent[52])	NON
Atelier avec importation	OUI	NON	NON

Figure PxM-40_34. Exemple de graphe illustrant les relations permises



L'atelier : intérieur et extérieur

La figure ci-dessous montre toutes les relations autorisées par UML entre les composantes d'un atelier : ses machines (avec différentes valeurs de visibilité) et son interface.

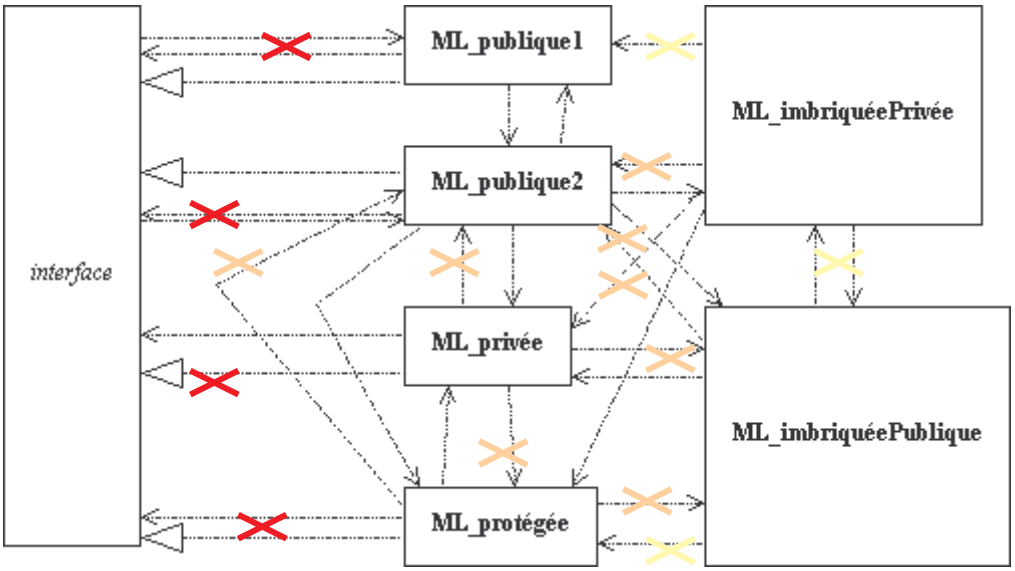
Praxeme impose des contraintes supplémentaires, prohibant certaines relations que la notation accepte. Les interdictions sont montrées par les croix.

[52] L'héritage est peu utilisé dans notre architecture de services. Les cas d'héritage se limitent à la relation aux machines génériques (ML_Ensemble et ML_Element) ainsi que l'enrichissement permis des machines utilitaires.

Légende

- Strictement interdit
- Déconseillé (relations mutuelles à l'intérieur de l'atelier)
- À éviter si possible (visibilité descendante)

Figure PxM-40_35. Les relations entre les classes du modèle logique



Commentaire

1. L'interface n'est qu'une spécification de service. Elle ne voit rien du contenu de l'atelier.
2. Une ou plusieurs machines se déclarent comme réalisant tout ou partie des services déclarés dans l'interface[53].
3. Le plus souvent, l'interface sera outillée par deux machines, l'une élémentaire, l'autre individuelle. C'est la conséquence du critère retenu pour délimiter les ateliers : l'objet métier.
4. Dans les cas où l'atelier couvre plusieurs concepts, l'interface rassemblera des services de plus de deux machines. L'architecte peut préférer donner une plus grande unité à l'atelier en créant une classe stéréotypée « façade ». C'est, alors, cette classe qui « implémente » l'interface.
5. Les deux machines représentées à droite de la figure sont imbriquées dans la ML_publique2. Les conseils pour réduire les dépendances sont plus stricts que UML. Ils visent à réduire la complexité de l'architecture. Les machines imbriquées permettent de cacher des détails. Ils correspondent à des notions ou informations satellites, utiles pour le fonctionnement d'une machine normale mais sans intérêt en dehors de celle-ci.

[53] Sur ce diagramme de classes, la relation « *implement* » (réalise) est exprimée par le trait en pointillé terminé par un triangle blanc du côté de l'interface.

Thématique de la conception logique (suite)

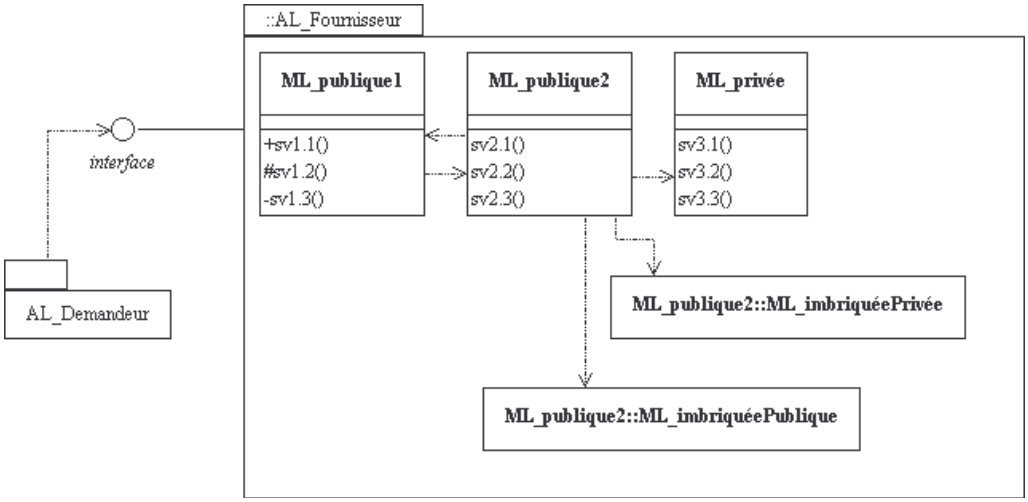
Application à l'architecture logique

Le schéma type

La figure suivante applique les règles de la modélisation logique ainsi définies. C'est un graphe typique tel que le modèle logique en présentera de nombreux. Les ateliers externes sont principalement connectés par les interfaces.

À l'intérieur de l'atelier, les machines se voient les unes les autres, à l'exception des machines imbriquées qui ne sont connues que de leur propriétaire.

Figure PxM-40_36. Graphe local d'architecture, typique



Thématique de la conception logique (suite)

Une typologie des services logiques

Introduction

Praxeme ne pose pas une typologie de services *a priori* ; la typologie se dégage plutôt *a posteriori* : elle découle des règles d'architecture qui ont été données et motivées par la qualité du système. Les expressions données ci-dessous ont été recueillies sur le terrain, à partir de projets appliquant Praxeme. Ce vocabulaire ne revêt pas une grande importance théorique mais il apporte une commodité pour la communication sur le terrain.

Le type d'un service donné n'est donc pas une décision première, lors de la création du service, mais se déduit des facteurs topologiques et fonctionnels :

- position du service dans l'architecture ;
- contenu du service.

Sauf exceptions, les types obtenus n'ont pas d'incidence sur la dérivation du service logique en composant logiciel. En retour, l'architecture technique n'impose pas de contraintes particulières quant à la spécification logique du service.

Position sur les strates

La première caractéristique d'un service est son appartenance à une des strates de l'architecture logique.

Service interne Un service est dit « interne » ou « métier » s'il appartient à une machine localisée sur la strate « Métier ».

On dit aussi « service de MLM » ou « SLM ».

Service externe Un service est dit « externe » ou « organisation » s'il appartient à une machine localisée sur la strate « Organisation ».

On dit aussi « service de MLO » ou « SLO ».

Position dans l'atelier

Les services sont toujours inscrits sur une classe. La classe est stéréotypée soit « Machine logique », soit « interface ».

Au niveau logique, selon Praxeme, seuls les ateliers présentent des interfaces (il s'agit d'une application restreinte du standard UML).

Service d'interface Un service d'interface est un service porté par l'interface d'un atelier logique.

Normalement, seuls les services d'interface sont connus à l'extérieur de l'atelier (principe d'encapsulation des machines).

On dit aussi, de façon moins précise, « service d'atelier ».

Service local Un service local est un service porté par une machine d'un atelier considéré. Il n'est pas perçu à l'extérieur de l'atelier, sauf dans les cas de relations particulières entre les ateliers.

Fonction du service

Une autre façon de catégoriser les services consiste à considérer leur contenu fonctionnel : on distingue ainsi les services d'information et les services de transformation. Cette distinction se surimpose aux catégories données précédemment.

Service d'information

Un service est dit « d'information » s'il ne fait que livrer de l'information, sans modifier l'état interne du système.

Autres désignations : service de sélection, service d'interrogation, service de consultation…

Le service d'information produit l'information demandée :

- soit en la lisant dans la structure de données de la machine ou le support de données sous-jacent ;
- soit en la calculant (quand le service traduit un attribut dérivé, trouvé sur le modèle amont).

Service de transformation

Le service est dit « de transformation » quand il modifie l'état interne du système.

Autres désignations : service d'action, service de modification…

La transformation peut être directe ou déléguée. Dans le premier cas, elle porte sur la machine où se trouve le service ; dans le second, la transformation s'applique à une autre machine. La transformation peut s'accompagner d'une transition d'état sur l'automate des machines impliquées.

Service de transaction

Le service de transaction est un cas particulier de service de transformation.

C'est un service qui pilote des transformations au sein d'une transaction et qui garantit l'unité et la cohérence de la transaction. Il fait donc appel à un mécanisme de gestion des transactions et déclenche des services de transformation, éventuellement en cascade.

Thématique de la conception logique (suite)

Le passage des identifiants

Identification interne au système

La modélisation sémantique adopte les techniques de représentation de l'approche orientée objets. De plus, visant l'essentiel, elle ne se préoccupe pas de données artificiellement construites. Pour ces deux raisons, le modèle sémantique ne présente pas systématiquement les identifiants sur les classes ; il applique l'identité d'objet, propre aux technologies objets. Il a pu, tout de même, marquer certains attributs comme propriétés naturellement identifiantes. Certaines relations dans le modèle permettent de construire l'identifiant des notions satellites.

Dans le modèle logique, l'identification est une préoccupation systématique. Le style choisi impose les règles suivantes :

- Toute machine porteuse d'information doit disposer d'un identifiant logique permettant de désigner de façon univoque les objets à manipuler.
- L'identification des objets est logique, c'est-à-dire indépendante des solutions organiques et comprises de l'extérieur des ateliers.
- L'activation d'un service élémentaire[54] suppose d'avoir désigné l'objet sur lequel le service s'applique.

Identification multi-système

L'architecture que nous élaborons se veut multi-système, applicable à un SIIO[55] extensible et reconfigurable. Le dispositif d'identification logique doit tenir compte de cette exigence.

Identifiant concaténé

Un objet comme « client » peut recevoir un identifiant logique unique et simple (sous réserve de pouvoir l'identifier dans plusieurs systèmes…). Au contraire, l'objet « couverture » est identifié par la concaténation des identifiants de dommage et garantie (la garantie couvre le dommage). L'identifiant de la garantie peut lui-même agglomérer l'identifiant du contrat, du type de contrat, etc. Cette façon de faire respecte au plus près la sémantique. Elle présente, donc, les avantages de la stabilité et de l'universalité.

Dans le cas de ces identifiants concaténés, on peut hésiter sur la façon de passer les valeurs dans la signature des services. Les deux options sont :
1. un seul paramètre (chaîne de caractères) recevant le résultat de la concaténation ;
2. autant de paramètres qu'il y a d'objets impliqués dans l'identification.

La deuxième solution, quoique apparemment plus complexe, est plus proche de la sémantique car elle conserve les éléments d'information exploitables. De cette façon, les paramètres portent du sens, sans autre intervention, et le service qui les reçoit peut se retourner vers les machines qui contrôlent les objets liés[56].

---

[54] Les services des machines ensemblistes, le plus souvent, portent sur l'ensemble des objets. Il n'est donc pas nécessaire de désigner un objet particulier.

[55] SIIO : système d'information inter-organisationnel. On parle aussi de « fédération de systèmes ». Cette dimension, imposée par l'évolution de l'économie, change l'approche des systèmes, pas seulement sur leur architecture générale mais aussi sur des thèmes précis dont l'identification des objets fournit un bon exemple.

[56] Dans l'autre solution, il faudrait que le service décortique lui-même un identifiant où tout est fondu (et dont la structure risquerait de changer).

Thématique de la conception logique (suite)

Les données : déclaration et communication

Les paramètres et données

Le nom des paramètres respecte la même orientation que le nom du service : il doit être **court et intelligible**, proche de la programmation tout en préservant au maximum la signification.

Une convention de nommage doit être établie au niveau du système considéré.

Quelques exemples :

- non pas "identifiant du sinistre", ni "Id Sinistre" mais "idSinistre" ;
- non pas "Montant principal à présenter" mais "mtPrincipal" ;
- préfixe "id" réservé (tout en minuscules) pour désigner l'identifiant (à la place d'une référence à un objet dans le modèle sémantique) ;
- nom de paramètres en minuscules (comme les attributs), sauf pour la première lettre des noms d'objets et pour séparer les mots (notation Camel) ;
- idSinistre, montantPrincipal, codeRetour…

Cette règle permet de distinguer : "date" (nom du paramètre) et "Date" (nom du type).

Déclaration des données

Dans la structure de la machine comme dans la signature des services, le concepteur applique les mêmes conventions de nommage et de typage. Le détail de ces conventions fait l'objet d'un document isolé, sous la responsabilité des architectes logiques. Les conventions de nommage sont vérifiées avant homologation des services.

Pour définir des structures complexes tels que des couples de valeurs (par exemple : code, libellé), UML nous fournit les types complexes (*data types*).

Mots réservés

Les mots réservés et les préfixes utilisés dans les dénominations ('cd' pour code, 'id', etc.) sont décrits dans le même document sur les conventions.

Communication des données

On peut envisager deux options :

1. Dans un souci de réduire les volumes échangés (volume des flux), les services ne communiquent que les codes (ex. liste de mouvements ; nature de mouvement), à charge pour le demandeur de solliciter lui-même le service de décodage (d'où nombre d'appels plus élevés ; exemple : analyserNature appelé pour tous les mouvements de la liste).
2. Au contraire, on peut chercher plutôt à réduire la fréquence des appels, quitte à augmenter le volume échangé, voire : risquer la redondance (dans la liste des mouvements, on passe tous les ingrédients de natureMvt ; redondance puisqu'il y a des mouvements de même nature, donc décomposition identique du code).

La deuxième option est assez conforme à l'utilisation du langage xml. Elle est aussi plus commode, même si elle oblige à une gestion rigoureuse de l'actualité des données. En effet, au moment de valider une transaction, on ne pourra pas faire confiance au seul contexte d'information : il faudra vérifier que les objets impliqués n'ont pas changé parallèlement.

Thématique de la conception logique (suite)

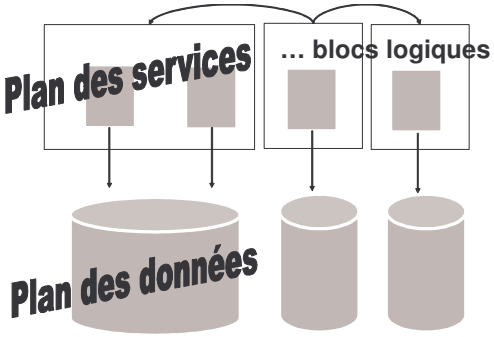
L'architecture des données

Le modèle logique des données

L'aspect logique se compose de deux plans distincts :

- le plan des services dont le présent guide fournit l'approche ;
- le plan des données.

Figure PxM-40_37. Les deux plans d'une architecture de services



Le modèle logique des données dérive des mêmes sources que le modèle des services, mais avec des règles de dérivations différentes. En effet, les technologies retenues pour les services ne reposent pas sur le même paradigme que celles des SGBD. De plus, le modèle des données doit obéir à des contraintes distinctes, telles que la politique logiciel, la prise en compte de l'existant ou la sécurité des données.

Ces questions font l'objet d'un autre document du référentiel méthodologique[57]. Nous en donnons, ci-dessous, un aperçu.

Les décisions

La principale décision à prendre concerne le périmètre de la ou des bases de données, nécessaires au système. Plus exactement, il s'agit d'établir la correspondance entre la base de données et un des types de constituants logiques : fabrique logique ou atelier logique. Cette question se pose, bien sûr, quand la reconstruction du système va de pair avec une évolution des bases de données. Il n'est pas inutile de se la poser, également, dans le cas d'un existant. L'exercice ouvre toujours des pistes d'amélioration.

La représentation

Le modèle logique incorpore le modèle logique des données. Il est, en effet, indispensable de montrer les ressources que mobilisent les services, en premier lieu les supports de données. Un lien de dépendance est dessiné entre la machine logique et la table dont elle est responsable.

Pour l'outillage, le mieux est de rester dans l'outil UML et d'utiliser les profils – généralement fournis par les éditeurs – pour traiter des SGBD.

Les codifications

Le thème des codifications a été évoqué comme illustration de certains types de relations dans l'architecture. Il appartient à l'architecture des données et le dispositif des codifications doit faire l'objet d'une conception complète.

Il est à noter que les valeurs licites de codifications dépendent du contexte de gestion. Celui-ci doit donc être pris en compte dans le dispositif. La question du multilinguisme se pose aussi.

Les messages

Les messages associés aux codes retour (dont les messages d'erreur) et les niveaux de réaction sont également un thème de conception logique. Du point de vue des services, le dispositif est susceptible d'une solution très proche de celle des codifications.

[57] Cf. « *La dérivation du modèle sémantique* », référence PxM-41.

Thématique de la conception logique (suite)

L'activité de l'architecte logique

Décisions

Parmi les décisions majeures que doit prendre l'architecte logique, on peut citer :

- Le style d'architecture.
- Le calcul du couplage des constituants et l'estimation de la volumétrie des échanges.
- La localisation des types complexes (sont-ils inscrits au plus près des machines utilisatrices ou bien sont-ils rassemblés dans un espace unique ?). Ce point est évoqué dans la liste de la négociation logique/technique.
- La correspondance entre les bases de données et les constituants logiques.
- Les conventions de nommage.
- Le dispositif de signalisation.
- Le dispositif de codification.

Vérification

Avant d'arriver à une architecture correctement construite, l'architecte logique a une activité importante de surveillance. Il reste vigilant sur l'évolution du couplage et de la redondance, dans le système, au fur et à mesure des projets.

Il vérifie que les principes de l'architecture de services sont respectés. Au-delà, il détecte des détails qui peuvent ruiner la qualité du système, comme dans les exemples suivants :

- Un atelier « ALO_Utilitaires », dans la strate « Organisation » a été introduit alors que l'on a déjà une fabrique « fUtilitaires »…
- Une machine « mAdressePersonne » apparaît et entre en concurrence avec une machine préexistante « mAdresse ».
- Les machines de la strate « Organisation » délèguent à un atelier général, tous les travaux d'éditique. En soi, c'est une bonne chose, mais l'architecture est conçue de telle manière que, au lieu de passer les flux à cet atelier, c'est à lui de chercher l'information dans le reste du système. L'atelier est donc largement couplé. La solution est mauvaise.

La traçabilité

La nécessité

L'architecture de services s'élabore sur le long terme. Le système se construit progressivement selon un plan préconçu formulé par le graphe d'architecture logique. Pendant la refonte du système, les spécifications ou le métier même peuvent évoluer. Il est impératif de mesurer l'impact des évolutions, sur tous les aspects du Système Entreprise. Cette exigence est celle de la traçabilité. Elle suppose de mettre en place un dispositif documentaire rigoureux. Le modèle logique contient les connexions entre ses éléments et ceux des modèles « amont » dont ils dérivent.

L'outillage UML est, ici encore, d'un grand secours. Le concepteur peut utiliser le stéréotype « trace » appliqué aux dépendances. On peut aussi créer un stéréotype plus explicite comme « dérive », pour conserver le lien entre un constituant logique et son origine dans un des modèles en amont.

Thématique de la conception logique (suite)

Les traitements différés et *batch*

Ce thème fait l'objet d'un document dédié : « Les traitements *batch* dans l'architecture de services », référence « PxM-45 ».

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[58], 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.

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

Synthèse des règles

Les règles de dérivation à partir du modèle sémantique

La dérivation structurelle

Le tableau ci-dessous résume les règles à suivre pour transformer les éléments du modèle sémantique en éléments logiques. Ces règles peuvent être amendées en fonction du style choisi pour l'architecture logique.

Terme de départ	Terme d'arrivée (logique)	Remarque sur la transformation
Domaine d'objets	Fabrique logique	Les domaines d'objets traduisent des hypothèses de structuration fortes et de nature logique, qui ont été imposées au modèle sémantique. Leur dérivation en fabriques logiques est donc naturelle.
Classe sémantique	Machine Logique Métier élémentaire	La MLM rassemble tous les services élémentaires déduits des propriétés de la classe. Voir ci-dessous : attributs, opérations et association. À cela s'ajoutent des services de navigation.
	Machine Logique Métier ensembliste	La plupart des classes sémantiques entraînent une machine ensembliste. Cette MLM fournit les services appliqués à l'ensemble des instances manipulées par la machine individuelle. En quelque sorte, la MLM ensembliste dérive la classe perçue « en extension ».[59]
	Services ensemblistes	Les machines ensemblistes proposent les services de création, de recherche, d'administration (comptage, statistique…). À ceux-là s'ajoutent les éventuels services statistiques ou liés à des comportements de masse.
Attribut d'une classe sémantique	Structure de données de la MLM	Les attributs de la classe, quelle que soit leur visibilité, se distribuent sur les structures apparente, sous-jacente et permanente. La structure de données est représentée sous la forme d'un type complexe. Ces types décrivent les flux du système et constituent le « langage pivot ».
Opération d'une classe sémantique	Service logique (dit « interne »)	1 pour 1, en général (le modèle sémantique ne retient que les opérations à contenu sémantique[60]).
Association	Services logiques de navigation	La localisation des services de navigation ainsi que leur signature dépendent de plusieurs facteurs : l'orientation de l'association, le type de l'association, les cardinalités.

[59] Une autre façon de dériver la classe sémantique consiste à ne faire qu'une machine avec deux interfaces pour séparer les services élémentaires et les services ensemblistes.

[60] On ne trouve pas, dans le modèle sémantique, des opérations comme « lire », « enregistrer », « lister », etc. Elles sont implicites et n'ajoutent rien à la connaissance du métier.

Synthèse des règles (suite)

Les règles de dérivation à partir du modèle sémantique (suite)

La dérivation contractuelle

Les règles du tableau suivant portent sur les éléments mis au jour par la modélisation contractuelle, toujours sur l'aspect sémantique : les automates, les règles et contraintes.

Ces éléments sont susceptibles d'être repris, au niveau logiciel, par des dispositifs particuliers. On peut penser à des *patterns* de transformation des automates, à des moteurs de règles, à des gestionnaires d'événements, etc. Aussi, la dérivation ne les touche-t-elle pas et se contente-t-elle de les transporter du modèle sémantique vers le modèle logique.

À cette occasion, le modélisateur vérifie tout de même la cohérence de la conception et peut s'imposer de formuler les contraintes de façon plus rigoureuse. L'intérêt de cet effort dépend de la transformation suivante, du logique vers le logiciel. Il serait inutile, par exemple, de formuler les règles en pseudo-langage si, ensuite, elles doivent être exprimées en JRules[61].

Terme de départ	Terme d'arrivée (logique)	Remarque sur la transformation
Automate à états	Automate à états	L'automate inscrit sur la classe sémantique est conservé au niveau logique. Il est reformulé en fonction de règles issues de la négociation logique/technique. Les résultats de cette négociation pourraient, dans certains cas, obliger à renoncer aux automates.
Signal	Signal	Si on choisit un style pur SOA, les événements de l'automate issu du modèle sémantique sont réduits à des appels de services. Si, au contraire, on conserve les événements, c'est que l'on opte pour un style intermédiaire entre SOA et EDA. Dans tous les cas, les signaux sont utiles au minimum pour la gestion des incidents.
Contrainte	Expression formelle	Les contraintes et règles de gestion ont été localisées sur le modèle sémantique. Lors de l'exécution, leur vérification est sous la responsabilité des services. Elles ne sont donc pas dérivées mais simplement transportées du modèle sémantique vers le modèle logique qui conserve leur point de localisation et les formalise pour les vérifier puis les réaliser.

La dérivation fonctionnelle

Les opérations sur les classes sémantiques sont décrites non seulement en termes de contrats (pré et post-conditions) mais également par leur fonctionnement. Cette description peut se faire en langage naturel ou en pseudo-code[62]. Quant aux services logiques, leur contenu sera toujours décrit en pseudo-code. Quand le pseudo-code existe déjà sur l'opération, il peut être repris dans le service qui en dérive. Il sera retouché pour tenir compte de l'environnement logique, au moins pour nommer les services appelés en respectant les règles d'expression logique. Un autre changement s'impose : remplacer, par des appels de services, la notation pointée utilisée pour naviguer dans le modèle sémantique. La notation pointée est utilisée dans le modèle logique mais l'architecture logique lui impose des limites que le modèle sémantique ne connaissait pas.

[61] À moins que ce pseudo-langage permette de préparer l'incorporation des contraintes dans le moteur de règles. Ces considérations sont abordées au cours de la « négociation logique / technique », moment clef dans la chaîne de production et préalable à l'engagement de l'effort de conception logique (voir p. 17).

[62] Il faut dire qu'une fois exprimées les pré et post-conditions, il ne reste plus grand chose à préciser dans les opérations, la plupart du temps.

Synthèse des règles (suite)

Les règles de dérivation à partir du modèle pragmatique

Introduction

Le modèle pragmatique comporte deux volets : la Vue de l'utilisation et la Vue de l'organisation. La Vue de l'utilisation fournit une vision locale de l'entreprise, du point de vue d'une unité organisationnelle ou d'un métier ; elle se formule en termes de cas d'utilisation. La Vue de l'organisation adopte un point de vue global et décrit les processus organisationnels qui traversent l'entreprise.

Ramenées à l'architecture logique, les deux vues se traduisent dans les mêmes termes.

Terme de départ	Terme d'arrivée (logique)	Remarque sur la transformation
Organisme	Fabrique logique	L'organisme est l'entité (société, organisation, groupe) impliquée dans le SIIO étudié (système d'information inter-organisationnel). Cette dérivation offre le maximum de liberté pour adapter les services de base à l'organisation et aux habitudes de travail du partenaire.
Domaine fonctionnel	Atelier logique	Le domaine fonctionnel (e.g. Gestion des sinistres) ne sert pas à structurer l'intérieur du système mais uniquement sa périphérie. Contrairement à l'atelier de la strate « Métier », celui de la strate « Organisation » est ouvert, de façon à offrir une plus grande facilité d'accès aux services externes.
Classe pragmatique (acteurs, objets organisationnels)	Machines logiques (élémentaire et ensembliste)	Machines et services obtenus selon les mêmes règles de dérivation qu'à partir d'un modèle sémantique. Ces machines s'apparentent aux machines de la strate « Métier » et sont rangées dans des ateliers de même type que les ateliers « Métier ».
Cas d'utilisation	Machine logique « Organisation »	Les règles de dérivation supposent que la Vue de l'utilisation a été structurée de façon optimale et qu'elle ne souffre d'aucune redondance. La MLO rassemble les services d'accès et de transformation déduits des activités propres au cas d'utilisation.
	Service logique transactionnel	À chaque cas d'utilisation, correspond un service logique appelé quand l'utilisateur demande la validation de son travail ; il déclenche et contrôle la transaction.
Automate à états du cas d'utilisation	Automate de la MLO	L'automate du cas d'utilisation, s'il existe, exprime les contraintes minimales qui conditionnent l'enchaînement des activités au sein d'un dialogue ou d'un traitement. L'automate de la MLO le traduit dans les termes de l'architecture logique.
Relation entre cas d'utilisation		Les MLO entretiennent entre elles des relations d'utilisation qui reflètent les relations établies entre les cas d'utilisation.
« *include* »	Relation d'utilisation entre MLO	La MLO correspondant au cas d'utilisation de base (principal) sollicite les services d'une autre MLO, correspondant au cas d'utilisation subordonné.
	Héritage ou utilisation	La négociation logique/technique décide de conserver ou pas l'héritage dans l'architecture logique. Dans la négative, les relations d'héritage entre cas d'utilisation sont converties en relation d'utilisation entre MLO. Le fonctionnement de l'IHM s'en trouve un peu compliqué.

Terme de départ	Terme d'arrivée (logique)	Remarque sur la transformation
« *extend* »	Points d'extension gérés par l'atelier	Solution *ad hoc* touchant la MLO du cas de base, l'atelier et la MLO du cas subordonné. Si un tel dispositif n'est pas prévu dans l'architecture, les relations « *extend* » sont traitées comme des « *include* », en prenant soin d'inverser l'orientation.
Activité élémentaire	Service logique (dit « externe »)	L'activité élémentaire est une activité d'un cas d'utilisation, ne nécessitant aucun échange avec l'utilisateur.

Les processus

Les cas d'utilisation permettent aisément d'identifier les services « externes », c'est-à-dire les services de la strate « Organisation ». En revanche, ils ne peuvent pas exprimer les logiques d'ordonnancement ou les contraintes d'une portée qui dépasse la situation individuelle de travail. Quand il s'agit de reprendre les processus organisationnels, le point de départ est la Vue de l'Organisation. Les processus s'expriment avec deux outils de représentation :

- le diagramme d'activité ou toute autre représentation de processus ;
- l'automate à états attachés aux objets « métier » ou aux objets de nature organisationnelle.

Dans le deuxième cas, il n'est pas besoin de règles de dérivation supplémentaire. Dans le premier cas, quand l'expression du processus n'a pas pu être entièrement absorbée par le modèle des objets réels ou administratifs, la question se pose de savoir comment l'architecture logique doit prendre en compte le processus.

Dans tous les cas, la logique du processus est traduite en orchestration de services. Les services orchestrés pour couvrir le processus peuvent être des services externes (fournis par les MLO), des services internes (via les interfaces d'ateliers de la strate « Métier » ou une combinaison des deux.

Les solutions sont les suivantes :

- Un dispositif transverse se charge de ce type d'orchestration, sans rien ajouter. Il exécute la description du processus et appelle les services.
- Une machine logique dédiée au processus s'ajoute à l'architecture. Elle surveille ou anime les flots d'objets impliqués dans le processus et les actions sur ces objets.

Synthèse des règles (suite)

Les règles de structuration

Les principes directeurs

La considération du couplage est d'autant plus critique qu'elle porte sur des agrégats de grande portée. On cherchera avant tout à réduire le couplage entre les fabriques logiques, puis entre les ateliers logiques et on tolèrera plus facilement le couplage entre les machines logiques, à l'intérieur d'un même atelier.

Les relations

Dans l'architecture logique, les natures de relations entre les agrégats logiques sont :

- la relation d'utilisation ;
- la relation d'importation ;
- la relation de généralisation.

On bannit complètement les associations dont la modélisation sémantique fait un large usage. Les associations sont remplacées par des relations d'utilisation.

L'utilisation

La relation d'utilisation est la principale relation dans l'architecture logique. Elle correspond à l'invocation d'un service. Elle est représentée par une flèche en pointillés et obéit à plusieurs règles.

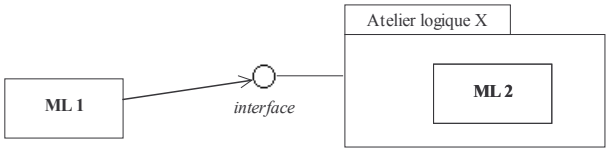
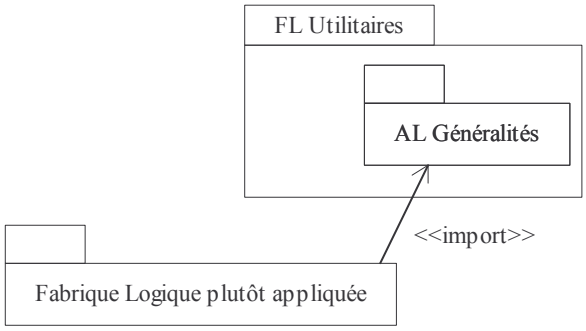


Figure PxM-40_38. Exemple de relation d'utilisation

L'importation

L'importation peut s'appliquer entre deux agrégats logiques qui ont des échelles différentes. Une fabrique peut importer un atelier (cas le plus fréquent). Un atelier peut, exceptionnellement, importer une machine. Ce point est détaillé dans la suite.

Figure PxM-40_39. Exemple de relation d'importation



La généralisation

La généralisation s'applique entre agrégats logiques de même niveau. L'agrégat reçoit tous les services publics et protégés de l'agrégat dont il hérite et se présente comme un agrégat plus riche.

Figure PxM-40_40. Exemples de relations de généralisation

Synthèse des règles (suite)

Les règles de structuration (suite)

Les contraintes topologiques

Le tableau ci-dessous récapitule les règles qui guident les décisions d'architecture logique.

Règle	Énoncé	Application
La prohibition des relations mutuelles et des cycles de dépendance	Deux agrégats ne peuvent pas être liés, directement ou indirectement, par des relations réciproques. (Si A utilise B, B ne peut pas utiliser A) (Si A utilise B qui utilise C, C ne peut pas utiliser A)	Règle absolue. Pas de dérogation possible pour les FL et AL. Exceptionnelle pour les ML d'un même atelier..
La polarisation des communications	Les appels sont orientés de la strate « Organisation » vers la strate « Métier ». De même, entre la strate « Présentation » et la strate « Organisation ».	Règle absolue. Elle découle du principe de stratification et joue un rôle essentiel dans la simplification du système.
Le cloisonnement procédural	Les ateliers logiques de la strate « Organisation » ne sont pas liés entre eux. (Ils n'ont de relations que vers la strate « Métier »)	Règle absolue, à l'exception des ateliers utilitaires (AL_Organisation)et des fonctions de support (ex. archivage).
L'encapsulation des ateliers logiques	Les services ne sont accessibles que par l'intermédiaire de l'interface des ateliers. (couplage via les interfaces des ateliers)	.
L'autonomie des ateliers	Les ateliers sont conçus pour fonctionner de la façon la plus autonome possible, en réduisant les appels vers les autres ateliers.	Orientation donnée aux décisions d'architecture. Ce n'est pas une règle absolue.

Les règles de conception

Validité

Les deux règles exposées ci-dessous s'appliquent à tous les niveaux de la structure : machines, ateliers et fabriques.

La cohérence de l'architecture

Le concepteur logique vérifie les relations d'utilisation (elles résument les relations qui existent dans le détail du modèle : associations entre classes, envois de messages, extension).

La quantification de l'architecture

Le concepteur rassemble l'information quantitative qui permettra de dimensionner l'architecture et de guider les choix de localisation des composants. Ces informations : volume et fréquence des échanges sont conservées dans le modèle logique sous la forme d'annotations (*tagged values*).

Strate « Métier »

La dérivation du modèle sémantique

Introduction

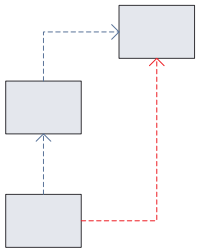
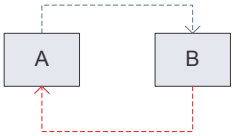
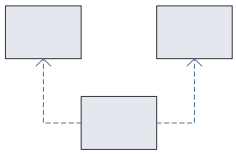
Ce chapitre apporte des compléments par rapport aux règles exposées dans la synthèse.

La dérivation des domaines d'objets

Elle doit s'accompagner d'une vérification des connexions entre les fabriques car les contraintes imposées par les relations d'utilisation entre paquetages n'ont peut-être pas été rigoureusement respectées dans le modèle sémantique.

Cette vérification n'est complète qu'une fois analysés tous les besoins de services à partir des machines.

Quand le concepteur découvre une impossibilité d'appeler un service parce que sa fabrique n'est pas reliée à la fabrique de départ, il soulève la question auprès de l'architecte logique. Ce dernier doit arbitrer. Avant de se résoudre à ajouter une relation au niveau des fabriques logiques, il dispose d'une série de décisions de moindre portée[63] :

	Problème posé	Solution proposée
Besoins en cascade		On n'autorise pas la relation supplémentaire de A vers C, si elle doit alourdir le graphe d'architecture logique (surtout si cela impose une nouvelle relation au niveau des fabriques). On enrichit l'interface de B pour que A en obtienne l'information recherchée.
Prohibition des relations mutuelles	 Quand les relations mutuelles paraissent incontournables, c'est l'indice d'une décomposition inadaptée.	Une solution radicale est la fusion des deux parties, possible uniquement à des niveaux fins. La solution passe souvent par le dégagement d'un nouveau concept C, qui conjoint la sémantique de A et B. 

Procédure préalable au partage du travail de conception logique détaillée

1. Transformer les domaines d'objets en fabriques logiques.
2. Délimiter les ateliers logiques (établir le premier graphe d'architecture logique).
3. Attribuer les machines aux ateliers (après dérivation).
4. Vérifier que la communication entre machines fonctionne dans le cadre des contraintes d'architecture logique.

[63] Les cas de figure présentés ici valent particulièrement aux niveaux des fabriques. Ces solutions génériques s'appliquent également aux niveaux inférieurs, ateliers et machines, quoique avec moins d'urgence.

Strate « Métier » (suite)

La délimitation des ateliers logiques

Un guide

Les Ateliers logiques de la strate « Métier » rassemblent des machines logiques proches et qui manipulent (ou encapsulent) les mêmes ressources (notamment, les tables). Ils résultent de décisions purement logiques et ne sont guidés que par un critère faible : la proximité sémantique. Les paragraphes ci-dessous aident l'architecte à circonscrire les ateliers.

Règles de transformation

- Les machines logiques dérivées d'une même classe sémantique se rangent dans le même Atelier. (La machine ensembliste se place dans le même atelier que la ML individuelle.)

- Les classes subordonnées[64] et les sous-classes sont, normalement, rassemblées dans le même Atelier que la classe principale ou la classe mère[65].

- En cas d'agrégation et de composition, si le composant B appartient au même domaine que A, les machines issues de B sont impérativement disposées dans le même atelier que les machines de A[66].

Règles de conception

- On range dans le même atelier les machines qui accèdent à la même table[67] (parfois, à une grappe de tables fortement liées).

- À l'inverse, une même table ne doit absolument pas être manipulée directement par plusieurs ateliers. Seule dérogation possible : quand le plan des services s'appuie sur une base de données existante[68].

- On réduit la visibilité sur le contenu de l'atelier : par une interface, elle-même associée aux machines, élémentaires et ensemblistes, dérivant la classe principale. Les machines principales sont forcément publiques. On admet qu'il puisse y avoir plusieurs notions centrales couvertes par un même atelier.

- Les machines logiques satellites sont privées ou protégées.

- L'exploitation des cardinalités peut aider à découper les ateliers.

---

[64] Les classes subordonnées sont celles isolées par relations d'agrégation, de composition, d'imbrication ou même d'association « terminale » (elles ne sont, elles-mêmes, associées à rien d'autre).

[65] Illustration de cette règle : la typologie des produits d'assurance est souvent fragmentée dans des domaines séparés (selon la division classique « Vie » / « IARD ») ; en suivant notre règle, tout ce qui concerne la description des produits d'assurance, quel que soit leur type, est rassemblé dans un seul atelier. Cette unité du catalogue est un changement important dans l'architecture. Elle se heurte à bien des résistances enracinées dans les habitudes. Il va de soi, pourtant, qu'un grand nombre de services logiques pour la manipulation des produits sont communs à tous les types de produits. Le refus de les traiter comme tels entraîne la « dérive des compartiments » : le sous-système pour les produits d'une catégorie s'écartera toujours davantage des sous-systèmes pour les autres catégories, avec un taux de redondance qui ira en augmentant.

[66] Évidemment, cela suppose que l'on a vérifié la sémantique de la relation et que le modélisateur sémantique n'a pas abusé de la commodité offerte par l'assemblage dans UML.

[67] Il s'agit ici des tables du modèle logique des données. Le MLD appartient au même aspect que le modèle des services.

[68] Dans ce cas, le « plan des services » donne l'image idéale (c'est-à-dire voulue) du système. Il cache le « plan des données » qui pourra progressivement évoluer de l'état actuel, éventuellement désordonné, vers un état plus conforme à la nouvelle architecture du système.

Strate « Métier » (suite)

La dérivation des classes

Les machines ensemblistes

Le modèle sémantique exclut les objets ensemblistes, du moins ne les représente-t-il pas de façon explicite. Par exemple, il ne contient pas de classe Catalogue ; le concept de Catalogue est restitué par la classe Produit, comprise en extension (comme l'ensemble de toutes ses instances).

L'architecture logique, au contraire, rend manifeste ces notions ensemblistes. La méthode prescrit même de créer une machine ensembliste pour chaque classe sémantique.

On a vu (p. 34) que, dans certains cas, les comportements ensemblistes pouvaient être absorbés par une machine élémentaire de plus forte valeur.

Les machines ensemblistes fonctionnent comme des singletons. Elles pourraient être connues du système par l'intermédiaire de variables globales.

La dérivation des attributs

Les attributs des classes sémantiques viennent s'inscrire dans la structure de données de la MLM.

À ce stade, on ne fait pas d'hypothèse sur la traduction au niveau logiciel : les services accesseurs pourront être au niveau des attributs ou au niveau de blocs de données (flux xml). La description retenue au niveau logique permettra de dériver en fonction des décisions techniques, dans l'une ou l'autre des options.

Dans le modèle logique, tous les attributs sont protégés en écriture.

La réduction de la propagation

Quand peut-on transformer l'appel de service du mode de propagation en passage d'information par paramètres ?

La réponse est : toujours dans les cas où le service appelé se limite à une fourniture d'information (consultation ou sélection), sans impact sur les machines sollicitées. Dans les cas de mise à jour, on évite cette transformation et on préfère laisser la propagation.

La transformation de l'appel produit deux effets :

1. d'une part, l'ajout d'un ou plusieurs paramètres correspondant à l'information attendue ;
2. d'autre part, l'enrichissement de l'interface de l'atelier avec les mêmes paramètres.

La transformation des appels en passage de paramètres n'est pas possible dans les cas suivants :

- Le service transforme les objets distants (il y a modification et mise à jour dans les machines sollicitées).
- Les ateliers ont été liés par une relation d'importation.

Strate « Métier » (suite)

La dérivation des relations

Les relations

Les relations présentes sur le modèle sémantique sont :

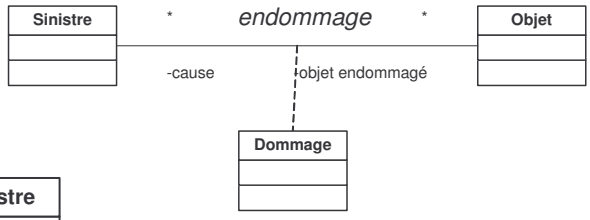
- l'association ;
- l'agrégation et la composition (cas particuliers de l'association) ;
- la classification (héritage) ;
- l'utilisation.

La transformation de l'association

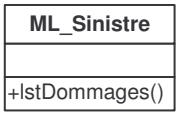
- L'association sémantique devient relation d'utilisation – ou dépendance – entre machines logiques[69], toujours si on adopte un style radical. En éliminant l'association et en lui substituant la dépendance qui se matérialisera sous la forme d'appels de services, on obtient un modèle logique transposable dans tous les environnements techniques.
- Si l'association est orientée, la relation d'utilisation est orientée dans le même sens.
- Si l'association n'est pas orientée, cela peut signifier que, pour le modélisateur sémantique, l'association est navigable dans les deux sens. Le concepteur logique peut, à la rigueur, s'autoriser l'utilisation mutuelle mais uniquement entre deux machines logiques d'un même atelier.
- Dans le cas contraire, on considère les relations d'utilisation entre les paquetages (elles ont dû conduire à orienter les associations).
- L'association donne lieu à un service de navigation, placé sur la machine utilisatrice (service de liste ou accesseur simple selon la cardinalité). L'orientation de l'association entre classes ou, à défaut, entre les machines, détermine la place du service de navigation.
- Si l'association est réifiée, la transformation produit deux services, l'un pour la classe de départ, l'autre pour la classe associative[70]. La machine de départ (ex. : ML_Sinistre) qui est avant tout une machine élémentaire (pour la classe Sinistre), fonctionne également comme la machine ensembliste pour la classe associative (dans l'exemple, Dommage).

Figure PxM-40_41. La localisation des services dans le cas d'une classe associative

Extrait du modèle sémantique



Extrait du modèle logique



[69] Le modèle logique ne présente pas d'association. L'association est une relation stable entre deux classes. La relation est reprise, au niveau logique, sous la forme d'une relation d'utilisation traduisant la possibilité d'un appel de service. La stabilité de l'association est assumée, au niveau logique, par le modèle logique des données.

[70] Exemple : sur ML_Sinistre, "lstGaranties" retourne la liste des identifiants des garanties qui couvrent le sinistre ; "couverture (id_garantie, objet)" fournit le contenu de la couverture du sinistre pour la garantie identifiée par "id_garantie".

Strate « Métier » (suite)

La dérivation des relations (suite)

La dérivation des associations

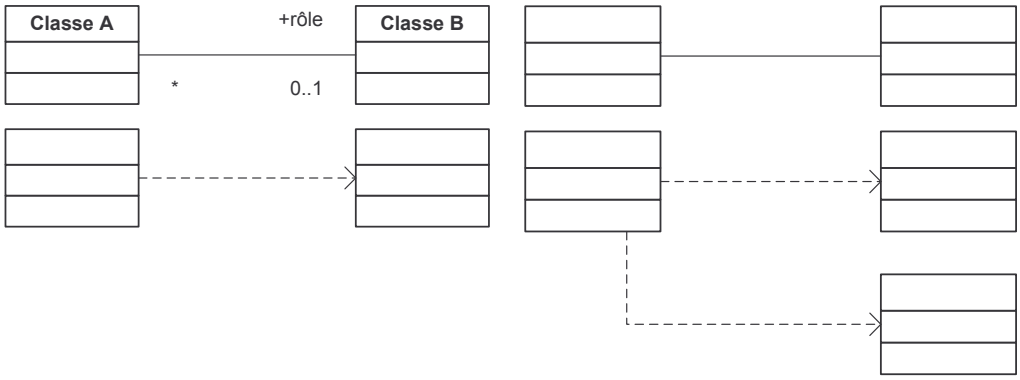
La localisation des services de navigation ainsi que leur signature dépendent de plusieurs facteurs :

- l'orientation de l'association : elle peut avoir été indiquée sur le modèle sémantique ; elle peut parfois se déduire du graphe d'architecture logique (même sens que les relations d'utilisation entre les paquetages) ;
- le type de l'association (associations spéciales : réifiées, qualifiées, ternaires) ;
- les cardinalités : à examiner en dernier ressort (après les critères précédents)[71].

Les rôles Les noms des rôles sur les associations constituent de bons candidats pour forger le nom des services.

Les modèles de la première ligne se dérivent comme indiqué en-dessous.

Figure PxM-40_42. Les rôles d'association dans la dérivation



La transformation de l'agrégation et de la composition

- L'agrégation simple n'est qu'un cas particulier d'association. On peut la considérer comme orientée du composé vers le composant, ce qui conduit à placer le service de liste sur le composé.
- Dans la composition (agrégation stricte), l'orientation est impérative. La suppression du composé entraîne celle du composant (il ne devrait pas y avoir de pré-condition sur la suppression du composant, quand elle est demandée par le composé).
- Dans le cas de la composition, la machine propriétaire peut absorber les services ensemblistes sur les composés.

La transformation de la classification

Pour rendre l'architecture logique davantage portable, on peut choisir d'éliminer l'héritage dans l'expression logique[72]. Alors, les relations d'héritage entre classes des modèels « mont » doivent être transformées dans le modèle logique.

[71] Les cardinalités déterminent surtout le contenu de la signature : soit simple nommage (cardinalité = 1), soit liste ou paramètre pour la recherche (cardinalité = *).

ML_A «uses» MI

Quatre solutions théoriques se présentent (on les retrouvera pour la transformation vers le modèle logique des données). Si B hérite de A :

1. Une ML_A avec les propriétés de la classe A et ML_B avec, uniquement les propriétés de la classe B et un appel à A pour le reste. ML_B reproduit l'interface de ML_A mais ses services sont de simples relais sur ceux de ML_A.
2. ML_A idem mais ML_B incorporant les services de A (importation).
3. ML_B sans ML_A (valable si A est abstraite).
4. ML_A additionnant les services des sous-classes de A et pas de ML_B.

Ces solutions présentent chacune des avantages et des inconvénients qu'il faut soupeser au cas par cas. Les choix que le concepteur arrête à propos des machines sont indépendants de ceux du modèle logique des données. La tendance consiste à créer moins de machines logiques que de tables, à partir d'une classification. À l'extrême, une seule machine correspondant à la classe mère manipule autant de tables (ou sous-tables) qu'il y a de classes filles. De cette façon, l'architecture de services se simplifie. Le choix sur le modèle logique des données obéissent à d'autres préoccupations.

La transformation de l'utilisation

Une relation d'utilisation, présente sur le modèle sémantique, se conserve entre les machines dérivées des classes. La relation d'utilisation correspond à l'appel d'une opération de la classe ou un accès à ses attributs.

Le concepteur vérifie que le sens de l'appel est compatible avec les choix architecturaux.

La transformation de l'association réifiée

Une machine ensembliste dérive d'une classe sémantique quand celle-ci représente un concept fort, central. À l'inverse, quand une classe S est subordonnée à une classe principale P, les services ensemblistes pour manipuler S sont agrégés aux services de la machine élémentaire pour S.

Le cas typique est celui de la classe associative.

Dans ce cas, la question se pose de savoir vers laquelle des classes associées les services doivent migrer : pour cela, il faut avoir décidé de l'orientation des relations d'utilisation. Les services sur la classe associative s'inscrivent sur la classe qui est au départ de l'association. Le plus souvent, l'association n'est pas orientée, mais le raisonnement vaut sur l'architecture logique : là, les relations sont forcément orientées.

Le cas des associations qualifiées

Par exemple : Sinistre-Intervenant--->Personne (« intervenant » est le qualificateur de l'association).

- Si l'association est réifiée, la machine qui correspond à la classe associative incorpore le qualificateur.
- Si la cardinalité maximale est 1 de chaque côté de l'association, le qualificateur est incorporé à la machine correspondant à la branche opposée au qualificateur.
- La machine utilisatrice fournit un service de recherche des objets cibles contenant le qualificateur parmi les critères.

[72] Si on ne choisit pas cette option au niveau logique, c'est au niveau logiciel qu'il faudra transformer l'héritage des modèles « amont », du moins dans les environnements techniques qui ne le proposent pas.

Strate « Métier » (suite)

La prise en compte des contraintes et des règles de gestion

Origine

Le modèle sémantique contient :

- des contraintes (parfois représentées sur le diagramme de classes) ;
- des pré et post-conditions sur les opérations ;
- des invariants de classe.

Transformation

Le modèle logique ne dérive pas, à proprement parler, ces contraintes. Elles ne donnent pas lieu à des services spécifiques mais sont incorporées dans les services. Normalement, une règle donnée n'est localisée que dans un seul service. Si ce n'est pas le cas, on peut se poser des questions sur la qualité du modèle.

Il est essentiel que les contraintes et règles de gestion aient été *localisées* sur le modèle sémantique[73].

Documentation

L'expression des règles de gestion dans le modèle logique dépend du type de transformation envisagé lors du passage au logiciel.

Deux solutions se présentent :

- la programmation de la règle, « en dur » ;
- l'enregistrement dans un dispositif ad *hoc* (le moteur de règle).

Dans l'un et l'autre cas, le mode d'expression des règles dépend des possibilités de transformation automatique. Ce qui est certain, c'est que le modèle logique doit donner une expression non ambiguë et strictement contrôlée des règles. Le choix de l'option intervient lors de la négociation logique/techniquue.

L'automate à états et les signaux

Conformément aux résultats de la négociation logique/technique, les machines à états sont conservées dans le modèle logique. Leur transformation s'effectuera lors du passage au logiciel.

[73] Cf. « Guide de l'aspect sémantique », référence « PxM-10 ».

Strate « Organisation »

La structuration de la strate « Organisation »

Les fabriques logiques « Organisation »

La fabrique logique de la strate « Organisation » correspond à l'organisme (intégré au réseau d'entreprises et acteur d'un cas d'utilisation du système décrit).

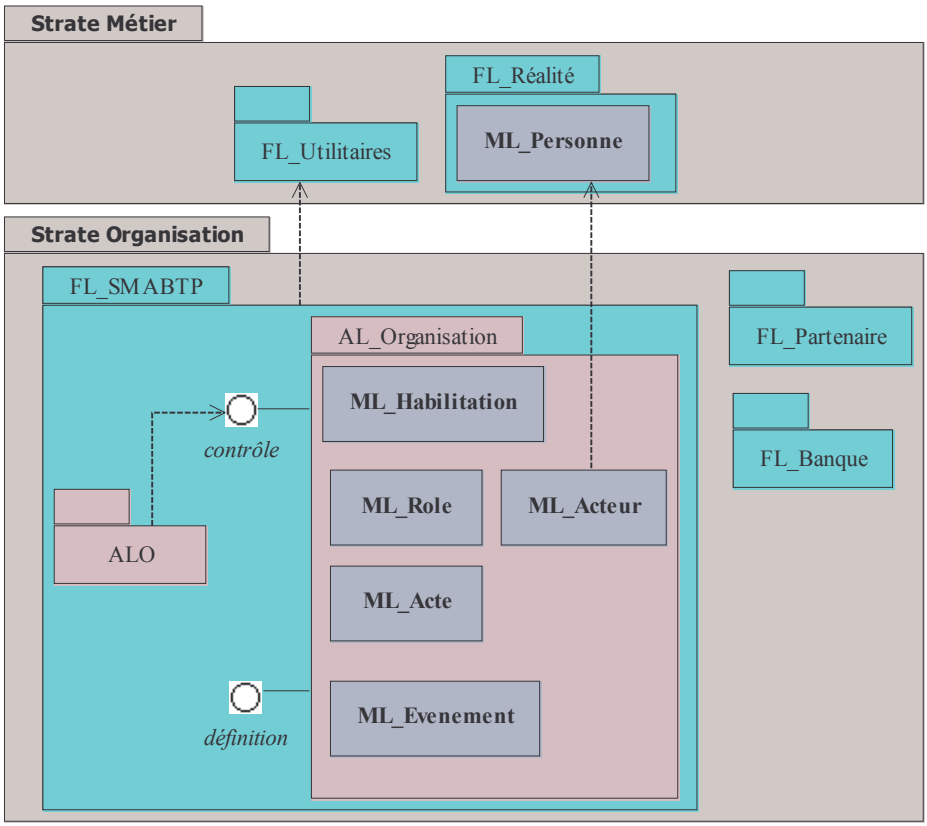
Il y a autant de FLO (fabriques logiques « Organisation ») que d'organismes intégrés au SIIO (système d'information inter-organisationnel). Ainsi, l'architecture s'accorde avec la dimension réseau d'entreprise. Le graphe d'architecture peut représenter des fédérations de systèmes. Chaque organisme dispose d'une liberté d'adaptation qui s'exerce dans la strate « Organisation ». La mutualisation et l'interopérabilité sont assurées par l'existence de la strate « Métier », elle-même construite à partir d'un modèle sémantique universel.

- Réutilisation : La FLO peut incorporer des ateliers repris d'autres organismes.
- Adaptation : La FLO contient des ateliers qui lui sont propres et qui inscrivent les modes opératoires spécifiques à l'organisme.

La structure type des ateliers

Tous les ateliers de la strate « Organisation » sont construits sur la même structure qui, d'emblée, leur permet d'accéder aux services génériques d'habilitation et de structure.

Figure PxM-40_43. L'atelier logique de la strate Organisation, dans l'ensemble de l'architecture



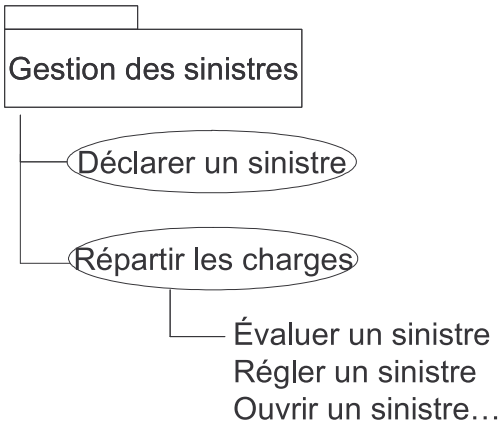
Strate « Organisation » (suite)

La dérivation du modèle pragmatique

Les conditions de la dérivation

Les règles de dérivation formulées dans la synthèse ne s'appliquent que sur une vue des cas d'utilisation qui a été rigoureusement structurée. Si ce travail n'a pas été mené, l'application mécanique des règles reconduirait la redondance de la vue d'utilisation dans l'architecture logique.

Figure PxM-40_44. L'effort de structuration des cas d'utilisation doit bannir la redondance



Les activités élémentaires

Les cas d'utilisation sont décomposés en activités élémentaires qui correspondent à des échanges simples – unitaires – avec les utilisateurs. Chaque activité est attribuée à un et un seul cas d'utilisation. UML impose que cette attribution s'effectue par l'intermédiaire d'un « graphe d'activité ».

Le modèle pragmatique (plus exactement : la Vue de l'utilisation) doit être structuré comme montré dans le schéma de droite.

Figure PxM-40_45. Mauvaise structure du dossier des cas d'utilisation

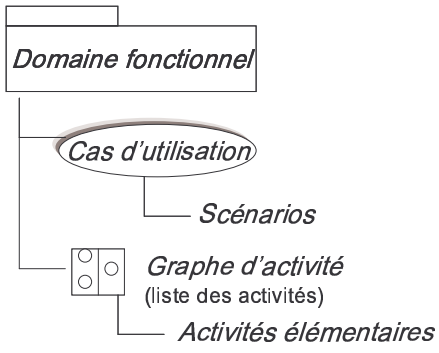
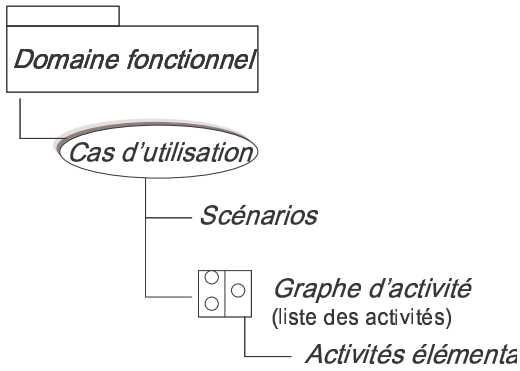


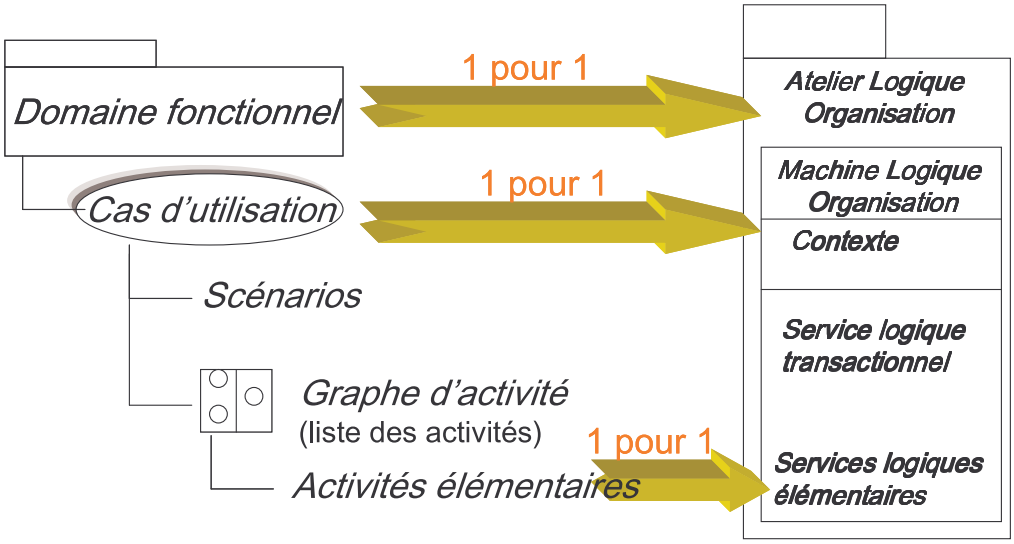
Figure PxM-40_46. Structure prescrite pour appliquer les règles de dérivation



Récapitulatif

La figure ci-dessous récapitule les règles de dérivation à partir du modèle pragmatique (Vue de l'utilisation).

Figure PxM-40_47. Les règles de dérivation de la Vue d'utilisation vers la strate « Organisation »



Le domaine fonctionnel

L'atelier logique de la strate « Organisation » correspond au domaine fonctionnel.

Le domaine fonctionnel se définit comme partie d'un système d'information répondant aux besoins d'une des fonctions de l'organisme[74].

Les objets de l'organisation

Le modèle pragmatique peut décrire des objets de nature organisationnelle.

Deux cas se présentent :

1. les objets génériques (acteurs, structures, droits, événements…) décrivant l'organisation ;
2. les objets particuliers nécessaires à un processus ou à un domaine fonctionnel (ex. : dossiers, paramètres, contextes d'utilisation…).

L'atelier « Organisation » traite le premier cas et fournit ses services transverses à tous les ateliers de la strate « Organisation » (voir p. 87, paragraphe « La structure type des ateliers »).

Dans le deuxième cas, on applique les règles de dérivation identiques à celles sur le modèle sémantique. Des solutions stéréotypées (plutôt *analysis patterns*) peuvent se présenter pour ces sujets.

Règles d'architecture

Les MLO n'appellent que les services d'interface des ateliers (jamais directement les services internes des MLM).

Les relations entre les deux strates ne peuvent être que des relations d'utilisation. Une relation d'importation contredirait la règle de stratification. Une dérogation peut, tout de même, être accordée pour les services utilitaires – ce qui est un cas très particulier.

[74] C'est le domaine pris au sens des méthodes traditionnelles (approche verticale conduisant à une architecture en silos, avec peu de partage). Exemples : la gestion des sinistres, le pilotage, le marketing, la vente.

Strate « Organisation » (suite)

La dérivation des cas d'utilisation

Principe

Le concepteur logique exploite les descriptions de cas d'utilisation ou les spécifications fonctionnelles pour détecter les besoins d'interrogation ou d'action sur le système. Ces besoins doivent être couverts par les services.

Le déroulement

Au cours de l'exécution d'un cas d'utilisation

L'utilisateur apporte de l'information au système à travers une suite d'échanges. Chaque échange se réalise sous la forme d'un service externe, appelé à partir de l'interface homme-machine. Le système accumule cette information.

Ces services participant à l'échange sont élémentaires. Ils correspondent à l'activité élémentaire trouvée dans la description des cas d'utilisation ou situations de travail.

Les services trouvés à partir des activités élémentaires peuvent être invoqués à partir de plusieurs cas d'utilisation ou à travers des interfaces différentes. Leur granularité est inférieure à celle du cas d'utilisation.

La logique procédurale

L'ordonnancement des actions obéit, éventuellement, à un ensemble de contraintes qui impose une logique procédurale. Pour la reconstituer, il convient d'exploiter tous les scénarios. La MLO doit la contrôler, sans pour autant la rigidifier. Or, à partir de l'expression littéraire du cas d'utilisation ou de l'acte de gestion, le risque est grand de percevoir l'activité sous un schéma exagérément linéaire. Pour échapper à ce travers et offrir une plus grande liberté – sinon à l'interface, du moins dans les services logiques – Praxeme conseille vivement de recourir à la technique des automates à états.

Chaque valeur de l'état marque une étape significative dans la procédure ou un niveau d'information atteint. Les transitions établissent les contraintes et possibilités de déclenchement. Le concepteur prend soin d'ajouter aux transitions évidentes du déroulement linéaire, les transitions permettant d'accueillir toute perturbation qui peut survenir. À un stade avancé de la modélisation, le concepteur inscrit les services à déclencher, sur les transitions et à l'intérieur des états.

À la fin du cas d'utilisation

Il reste à valider la suite cohérente des échanges. C'est le rôle du service de transaction. En surface, cela correspond au bouton « Valider » qui clôt l'acte de gestion.

Illustration

Le déroulement d'un cas d'utilisation est illustré par le diagramme de séquence de la page suivante.

Commentaire

L'utilisateur échange avec le système par l'intermédiaire d'une interface homme-machine. Au niveau logiciel, celle-ci peut se composer de plusieurs objets graphiques, non représentés ici. L'IHM relaye les interactions de l'utilisateur vers le système. Elle ne s'adresse qu'à la strate « Organisation ». Un même cas d'utilisation peut mobiliser plusieurs machines logiques de cette strate. Les MLO pilotent les machines de la strate « Métier ». Une même MLO peut manipuler plusieurs MLM et en assembler les services internes dans la perspective d'un acte de gestion ou d'un processus organisationnel.

Le recours aux services d'habilitation et aux services utilitaires n'a pas été représenté ici.

Figure PxM-40_48. Diagramme de séquence (théorique) illustrant le déroulement d'un cas d'utilisation

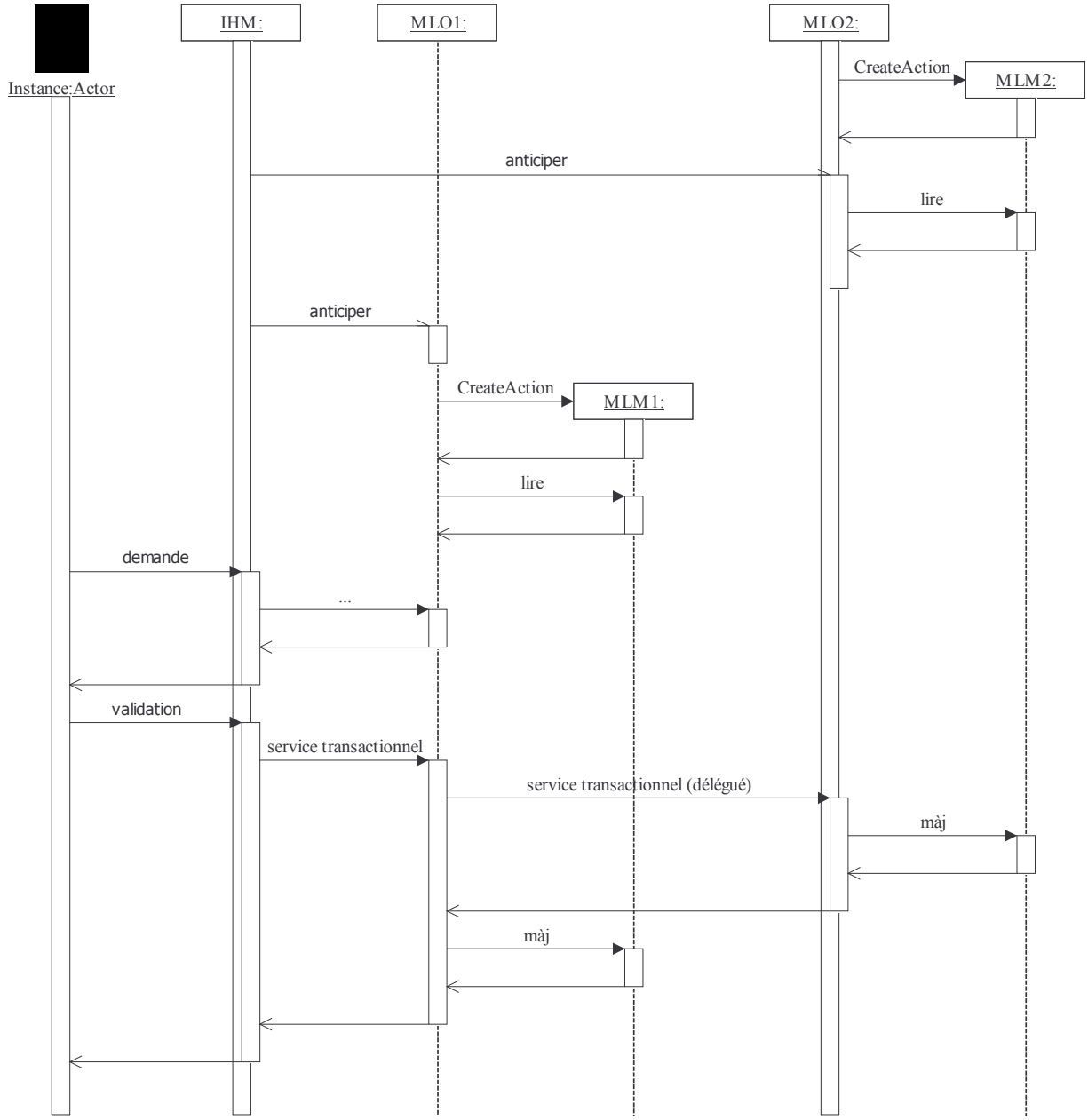


Illustration (suite)

Points remarquables

Les MLO instancient les MLM selon leurs besoins. La durée de vie des MLM est un point d'architecture technique, non considéré ici.

Elles peuvent **anticiper** les besoins d'information (d'où communication asynchrone représentée sur le diagramme).

Dans l'exemple, la transaction se propage en cascade (cas d'utilisation subordonné).

Strate « Organisation » (suite)

Les transactions

Définition

Le terme « transaction » revêt plusieurs significations qu'il faut dissocier et positionner par rapport aux aspects.

Point de vue de l'acteur

Dans l'activité de l'acteur, la transaction est un ensemble cohérent d'actions, produisant un résultat unique, du point de vue métier. Elle correspond à l'acte de gestion ou, en termes UML, au cas d'utilisation. Cette définition se range dans l'aspect pragmatique, au sens de Praxeme.

Cette notion appelle une réflexion : assez souvent, la transaction couvre un ensemble assez vaste d'actions au cours duquel plusieurs objets métier sont modifiés. Par exemple, la description d'un sinistre peut englober la création d'objets endommagés et d'intervenants. Dans une conception classique, il n'y a qu'une transaction pour couvrir l'ensemble. En conséquence, si la transaction échoue, ce sont plusieurs actes qui sont considérés comme caducs. À rebours de cette approche, on peut préférer dissocier la création ou la modification de chacun des objets impliqués, chaque fois qu'ils ne sont pas liés par des dépendances. Cette façon de faire augmente le confort de l'utilisateur et sa productivité. Par exemple, on acceptera de conserver les informations déposées pour les intervenants et les dommages, même si la description du sinistre a échoué globalement. Ainsi, en cas d'incident, l'utilisateur ne perdra qu'une partie de son travail. La condition qu'impose cette conception des transactions est que l'utilisateur puisse facilement retrouver les informations déposées par les « micro-transactions », lors d'une reprise ultérieure.

Point de vue logique

Sur l'aspect logique, la notion de transaction se concrétise sous la forme du « service transactionnel », i.e. le service qui réalise les transformations et les transporte au cœur du système, après validation par l'utilisateur.

Il faut bien comprendre que le service transactionnel ne couvre pas toute la durée de la transaction au sens fonctionnel donné ci-dessus. En effet, il n'est déclenché qu'à la fin du cas d'utilisation, quand l'utilisateur confirme ses modifications. Le service transactionnel a pour objectif de mettre à jour l'information et de modifier l'état du système, suite aux interventions de l'utilisateur.

Point de vue technique

La transaction, au sens technique, est un mécanisme mis en œuvre dans les SGBD et qui permet d'assurer la cohérence de l'information répartie dans plusieurs tables.

Ce mécanisme se complique si le système s'appuie sur plusieurs bases de données. Il dépend étroitement de l'architecture des données.

La transaction technique est encapsulée dans les services transactionnels. L'architecture technique peut comprendre un mécanisme de verrou applicatif qui complète le *two-phases commit* du SGBD. La gestion des transactions est un point dur inscrit dans la liste canonique de la négociation logique/technique[75].

Conclusion

La conception logique assure les actions suivantes :

- définir des services transactionnels, conformément aux règles de transformation à partir des cas d'utilisation ;
- étudier l'enchaînement éventuel des services transactionnels ;
- analyser le périmètre des transactions fonctionnelles (à partir des spécifications fonctionnelles) et poser les verrous en conséquence (par exemple, lors de l'instanciation des machines logiques).

[75] Cf. « PxM-42 ».

Strate « Organisation » (suite)

Le service transactionnel

Définition

Le service transactionnel est un service logique externe assurant la mise à jour du système en respectant sa cohérence.

Contenu

Le service logique transactionnel ouvre la transaction (au sens technique). Il dépose l'information dans les machines « Métier », déclenche les vérifications sur l'information si elles n'ont pas été menées avant (c'est l'automate qui le renseigne sur l'état de l'information). Il appelle le moteur de règles.

Si les contrôles sont passés, le service ferme la transaction et ramène l'information sur le résultat de la transaction (ACK / NACK).

Conception

Origine Chaque cas d'utilisation donne lieu à un service logique transactionnel, sauf s'il se limite à de la consultation.

Localisation Un choix d'architecture logique peut être de réserver les services transactionnels à la strate « Organisation ». Ce choix est motivé par le fait que la logique transactionnelle se confond, presque toujours, avec le déroulement d'un dialogue. Pourtant, un tel choix n'est pas toujours tenable, particulièrement si l'on conçoit des services de masse (*batch*) sur les machines de la strate « Métier ».

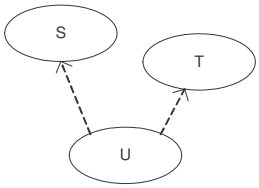
Désignation Le nom de ce service évoque le cas d'utilisation (ou reprend tel quel le nom du cas d'utilisation)[76].

Signature L'architecture technique impose parfois des limites dans la conception des transactions, en excluant par exemple les micro-transactions ou les transactions imbriquées. En pareille occurrence, il convient de réfléchir à la signature des services transactionnels. Cette signature peut comporter un paramètre supplémentaire pour demander ou pas l'exécution de la transaction technique. De cette façon, ce service pourra être sollicité ou assemblé par un autre service transactionnel, dans un contexte d'utilisation plus large[77]. Cette solution contourne l'absence de transactions imbriquées dans la technologie retenue.

[76] Il ne faut pas, pour autant, confondre ce service avec le cas d'utilisation lui-même. Le service de transaction ne doit pas rejouer tout le cas d'utilisation. Il n'en est que la conclusion, le cas s'étant déroulé comme une suite d'interactions entre l'utilisateur et le système. Chaque interaction a donné lieu à l'appel d'un service externe, lequel a pu solliciter plusieurs services internes (phénomène d'orchestration).

[77] Si on considère le diagramme des cas d'utilisation, on constate qu'un même cas d'utilisation peut fonctionner tantôt de façon indépendante, tantôt de façon subordonnée.

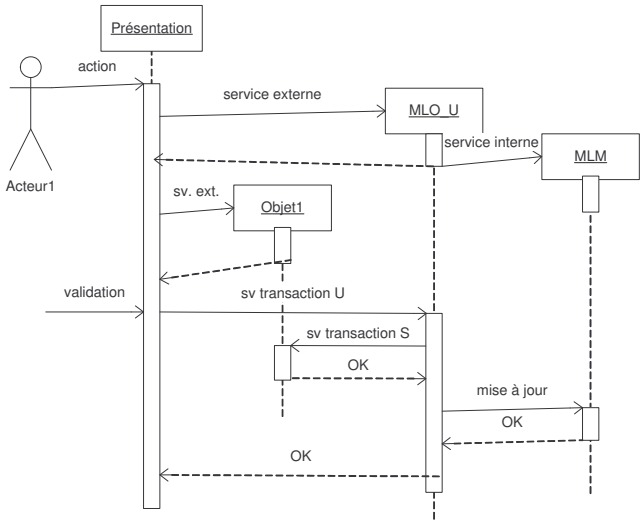
Les transactions emboîtées



Considérons le diagramme des cas d'utilisation ci-contre, avec un cas U qui « inclut » les cas d'utilisation S et T. Les questions sont : comment effectuer la validation de la transaction ? à quel moment appeler les services transactionnels des machines correspondant aux cas d'utilisation subordonnés, S et T ?

La solution la plus simple et aussi la plus confortable pour l'utilisateur consiste à appeler les services transactionnels à chaque fois à la fin du cas subordonné. Cela correspond, par exemple, à la fermeture de la fenêtre ou au changement de page, perçu comme validation. Cependant, la spécification fonctionnelle peut définir l'unité transactionnelle comme devant couvrir U, S et T[78]. Dans ce cas, le service transactionnel de la ML_U est le seul à être directement appelé à partir de la strate « Présentation ». Il ouvre la transaction technique avant d'appeler les services transactionnels de S et de T. Si les transactions imbriquées ne sont pas permises, ces appels précisent de ne pas ouvrir de transaction. Ensuite, si les services appelés se sont correctement déroulés, le service transactionnel de ML_U poursuit et valide la transaction, comme le montre la figure suivante.

Figure PxM-40_49. Diagramme de séquence montrant la relation entre deux MLO



[78] On déconseille cette conception des transactions. Le confort et l'efficacité pour l'utilisateur augmentent si les transactions ont un périmètre réduit, limité au cas d'utilisation.

Strate « Organisation » (suite)

La dérivation des processus

Le processus dans le modèle pragmatique

Dans le modèle pragmatique, le processus est enregistré sous la forme d'un graphe d'activité, lui-même décrit par un ou plusieurs diagrammes d'activité[79].

Deux cas de figure se présentent :

- Si le processus est intra-fonctionnel, le graphe d'activité est attaché au paquetage qui représente le domaine fonctionnel.
- Si le processus est inter-fonctionnel, c'est-à-dire transverse, le graphe d'activité est attaché à un paquetage des activités transverses ou au paquetage qui contient le modèle pragmatique.

Le principe de la dérivation

Comment les processus sont repris dans le modèle logique ?

La dérivation du processus intéresse la mise en œuvre et le suivi des processus, par exemple grâce à la technologie BPEL. Après la négociation logique technique, les règles de dérivation sont établies. Elles dépendent des orientations techniques, particulièrement de la présence d'un dispositif d'exécution des processus.

En présence d'un tel dispositif, le modèle logique reprend les descriptions de processus (graphes d'activité), éventuellement sous une forme plus contraintes que dans le modèle pragmatique. L'attachement de ce graphe d'activité reprend celui en vigueur dans le modèle pragmatique :

- Pour un processus intra-fonctionnel, le graphe d'activité appartient à l'atelier logique qui reprend le domaine fonctionnel.
- Pour un processus inter-fonctionnel, il s'accroche soit sur la fabrique logique (correspondant à l'organisme), soit sur un atelier créé pour les activités transverses.

Dans d'autres cas – soit parce que l'architecture ne prévoit pas de dispositif d'exécution des processus, soit en complément –, il peut s'avérer nécessaire d'ajouter une machine logique pour matérialiser le processus. Cependant, cette machine ne doit pas être trop lourde : s'il elle le devient, c'est que le modèle pragmatique n'a pas assez étudié les situations de travail individuelles, sous la forme de cas d'utilisation.

Quelle que soit la solution de principe retenue pour la dérivation des processus – soit en reprenant les graphes d'activité, soit en ajoutant des machines – il reste, au concepteur, à établir la correspondance entre les activités des processus et les services à appeler. Ces services sont exclusivement des services externes, placés sur la strate « Organisation ».

La transformation

Le concepteur logique reformule le processus. Cette reformulation a pour but d'introduire les services logiques dans le déroulement du processus et de préparer son automatisation. Dans l'aspect logiciel, les solutions de BPEL et de *workflow* s'alimenteront à partir du modèle logique. Plus précisément, la conception logique fournit les informations qui permettront l'orchestration des services et le suivi d'exécution, dans le cadre du processus.

Le diagramme d'activité montre les services qui s'enchaînent dans le processus. Les interventions humaines sont indiquées par des actions (*action state*) qui font référence aux cas d'utilisation. Qu'elles représentent des services

[79] Cf. *Guide de l'aspect pragmatique*, réf. « PxM-20 ».

logiques ou des interventions humaines, les actions se placent dans les couloirs dévolus aux éléments de modélisation : soit les constituants logiques (normalement, machines logiques « Organisation »), soit des types d'acteurs.

La dérivation des règles d'organisation

Les règles d'organisation

Il n'en va pas de même avec l'organisation qu'avec la sémantique. L'organisation est une variable d'ajustement à la disposition des dirigeants. On ne peut pas concevoir et imposer une organisation idéale. Dans un même secteur d'activité, la façon d'organiser les ressources, le style de management, le mode de contrôle et de délégation, etc. vont fournir des facteurs discriminants qui, parfois, se révèleront avantages concurrentiels. Dans une même entreprise, la prescription des modes opératoires (les processus) doit pouvoir, sans cesse, être corrigée à partir des rétroactions. Le système d'information doit donc accueillir ces variations. C'est un sujet critique pour l'architecture logique, en vue de garantir l'agilité du système informatique.

Variabilité

Les règles d'organisation sont les plus volatiles. Leur variabilité s'analyse dans deux dimensions :

- selon les habitudes de travail et les organisations (variations d'un partenaire à l'autre) ;
- dans le temps, en cas de reconfiguration organisationnelle ou de reconception des processus ou de changement organisationnel.

Le traitement de ces règles dans un dispositif séparé présente donc un grand intérêt.

Documentation

Pour cette raison, les règles d'organisation sont traitées totalement dans le moteur de règle.

Cela n'enlève rien à la question de leur expression dans le modèle logique, soulevée précédemment à propos des règles de gestion.

Représentation

Dans les algorithmes (diagrammes d'activité associés aux services), il est inutile de représenter l'appel au moteur de règles puisqu'il est présent dans tous les cas.

L'invocation du dispositif de vérification des règles se loge dans les actions traitant les pré et post-conditions. Elle peut être prise en charge par le *framework* technique, auquel cas la formulation logique peut être impactée et simplifiée.

Strate « Organisation » (suite)

Éléments propres aux machines de la strate « Organisation »

Le contrôle des habilitations

Les habilitations par rapport aux types d'acteurs se lisent sur le diagramme des cas d'utilisation. Un acteur est habilité pour un cas d'utilisation si le type d'acteur auquel il appartient est relié à ce cas d'utilisation.

Au niveau logique, les habilitations sont absorbées par l'atelier transverse AL_Organisation de la strate « Organisation ».

Tous les services de MLO ont en charge la vérification des habilitations avant de se tourner vers le cœur du système. Ce contrôle est effectué en pré-condition des services externes. Pour optimiser les échanges entre les strates « Présentation » et « Organisation », on peut mettre à profit la persistance de la MLO ou le contexte de session. Celui-ci peut véhiculer les droits de l'utilisateur que la machine n'a, alors, plus besoin de rechercher.

La construction d'une MLO

Presque toutes les MLO correspondent à un cas d'utilisation[80].

Elles présentent autant de services externes élémentaires qu'il y a d'activités élémentaires inscrites sur le cas d'utilisation (plus exactement, sur le graphe d'activité attaché au cas d'utilisation).

Elles offrent, en outre, un service correspondant à l'action finale (la validation) du cas d'utilisation ; c'est le service transactionnel.

L'état de la MLO

Les MLO gèrent logiquement leur état interne. Cet état dépend du stade d'avancement du cas d'utilisation ainsi que de l'état des informations manipulées. Il comprend, donc, deux facettes : côté « Présentation » (interaction avec l'utilisateur) et côté « Métier » (état des données métier).

Afin qu'une MLO réagisse correctement, elle doit disposer des éléments suivants :

1. le contexte d'utilisation (organisme ou contexte de gestion, acteur, identifiant + date et heure de la session ; pays et langue, nécessaires pour la table de codification, au moins) ;
2. le contexte informationnel (ensemble des informations assemblées par l'exécution du cas d'utilisation : à la fois, nouvelles informations saisies et informations obtenues des services internes).

Le contexte d'utilisation est véhiculé sous la forme d'un type, défini pour l'atelier ML_Organisation.

[80] Les exceptions sont liées aux processus ou aux objets de l'organisation : acteur, acte, habilitation, etc. Ces objets se montrent sur les diagrammes de classes du modèle pragmatique.

Strate « Organisation » (suite)

Les automates à états sur les MLO

L'automate à états pour traduire la logique procédurale

La MLO peut être équipée d'un automate à états. Il lui permet de gérer facilement son état interne, tel qu'évoqué ci-dessus. Ces automates ne reproduisent pas l'automate à états des machines « Métier » que la MLO pilote. Ils expriment la logique procédurale qui s'impose au cas d'utilisation. Le choix de l'automate comme moyen d'expression conduit à assouplir le comportement de la machine, donc à élargir sa gamme de réponses. Ainsi exprimée, la logique sera moins linéaire et plus réceptive aux aléas de l'activité. La MLO aura donc plus de chance de convenir à plusieurs organismes. Des contraintes plus rigides liées aux habitudes de travail peuvent toujours être mises en place au niveau de la strate « Présentation » ou à travers de règles d'organisation gérées par un moteur de règles en fonction du contexte organisationnel.

L'origine

L'automate à états tel que défini dans le paragraphe ci-dessus trouve sa place naturelle dans le modèle pragmatique. En fait, la meilleure façon de faire est d'attacher cet automate au cas d'utilisation. La notation UML permet, en effet, d'associer un automate à un cas d'utilisation.

Le cas échéant, l'automate du cas d'utilisation se formule en termes fonctionnels : il mentionne des événements et des activités élémentaires. L'automate de la MLO dérive l'automate du cas d'utilisation auquel la machine correspond. Ce nouvel automate a la même forme que le premier, mais les termes qui apparaissent sur les transitions sont des termes logiques : services de la MLO et événements (si l'architecture logique les tolère).

Le plus souvent, la conception logique fera un usage restreint de la syntaxe offerte par UML pour l'automate à états[81].

Commentaire

L'exemple de la page suivante est un diagramme d'états qui décrit l'automate de la MLO, dérivée à partir du cas d'utilisation « Répartir les charges ».

Ce cas d'utilisation entretient une relation de généralisation avec le cas « Décrire un sinistre », lui-même enrichissant le cas « Déclarer un sinistre ». Autour de ces cas gravitent plusieurs cas subordonnés.

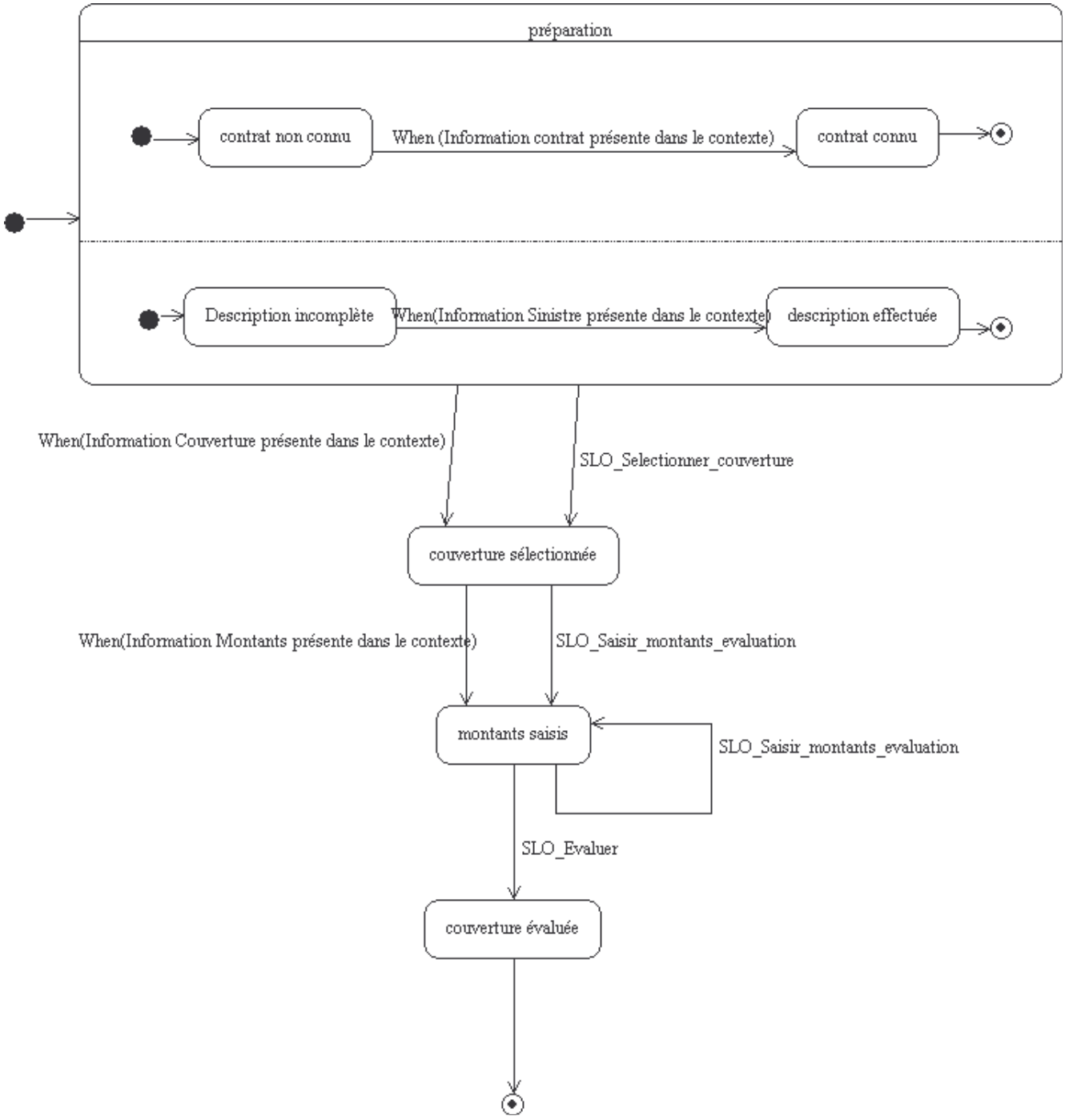
« Préparation » L'état résumé « préparation » trahit la dépendance du cas d'utilisation à l'égard des autres. Il couvre la détermination du contexte informationnel de la machine.

Les conditions De nombreuses transitions portent des expressions « When ». Ce sont des transitions qui se dérouleront automatiquement en fonction de l'état de l'information disponible. Certaines fois, ces transitions doublent des transitions sur action, c'est-à-dire réalisées par l'appel d'un service. Elles surmontent les interruptions que subit le déroulement du cas d'utilisation :

- soit que l'acte de gestion a été interrompu (mais le système a conservé le maximum d'information ;
- soit que l'architecture technique fait que la MLO n'a pas de persistance et qu'elle est instanciée à chaque échange entre l'utilisateur et le système.

[81] Cette syntaxe est, en effet, extraordinairement riche. Elle excède et les compétences des modélisateurs, et les possibilités de restitution par le logiciel.

Figure PxM-40_50. Un exemple : l'automate de la machine « MLO_Répartir_charges »



Strate « Organisation » (suite)

L'atelier « Organisation »

Définition

L'atelier AL_Organisation est un atelier logique de la strate « Organisation ». Il a une vocation particulière : il ne dérive pas d'un domaine fonctionnel (comme la gestion des sinistres) mais du modèle des objets de nature organisationnelle : type d'acteur, droit, rôle, etc. C'est donc un atelier transverse. Tous les autres ateliers externes (de la strate « Organisation ») ont recours à AL_Organisation.

Le service d'habilitation est sollicité à partir des pré-conditions des services externes.

Réalisation

La réalisation de cet atelier exploite des technologies particulières :

- moteur de règles (pour les règles d'organisation),
- *workflow*…

Ces choix d'architecture technique sont encapsulés dans les services de AL_Organisation et n'ont aucun impact sur le reste du système.

Architecture logique

Les services de cet atelier pourront faire l'objet de réalisations très différentes, d'un système à l'autre[82]. La solution d'architecture est montrée dans le schéma ci-dessous. Elle consiste à définir une interface universelle, laquelle pourra recevoir une réalisation différente selon les systèmes[83].

Figure PxM-40_51. Principe pour le déploiement des services d'habilitation

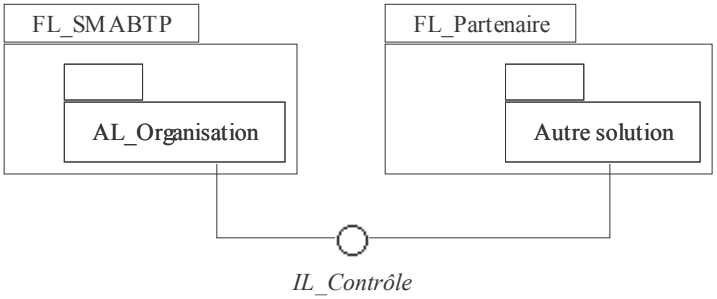
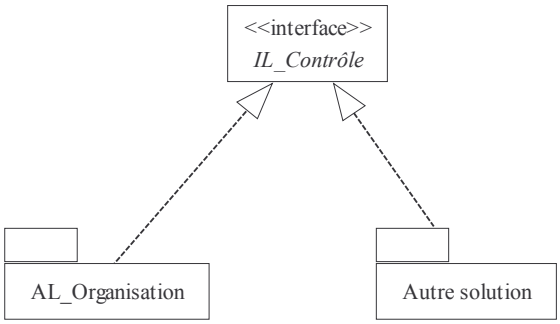


Figure PxM-40_52. Autre représentation



Cette figure représente la même chose sous la forme expansée. Les flèches traduisent la relation « d'implémentation ».

[82] Par exemple, JRules est un choix de la SMABTP mais ne sera pas nécessairement partagé par les partenaires.

[83] Cette solution ménage la plus large indépendance entre les systèmes, du moins sur ce point du contrôle des règles organisationnelles. Une autre solution consiste à définir un atelier générique AL_Organisation et, pour chaque organisme, un atelier qui en hérite, donnant les détails de la réalisation.

Strate « Organisation » (suite)

L'atelier « Services généraux »

Définition

L'atelier « Services généraux » est un autre incontournable. Il traite de la préparation et de la diffusion des informations, quel que soit le canal (éditique, courrier, courrier électronique, fax, SMS, etc.).

Fonctionnement

Cet atelier décharge les ateliers à contenu fonctionnel. Ces derniers lui soumettent des flux, tels que décrits par les structures de données. Les « Services généraux » s'occupent du reste : ils traitent les flux, dans la forme, sans en comprendre le contenu. Comprendre le contenu du flux supposerait une connexion réciproque vers l'atelier émetteur, ce que nous interdisent nos règles d'architecture. Les services généraux exploitent le flux qui leur est soumis : tri, sélection horizontale (colonnes), sélection verticale (ligne), rupture.

Les fonctionnalités d'archivage pourraient être rangées dans cet atelier des « services généraux ».

Positionnement

Les dépendances vont des ateliers ordinaires vers l'atelier « Services généraux », pas l'inverse.

Strate « Présentation »

Quelques remarques

Principe

La strate « Présentation » n'est pas abordée dans ce guide. La raison en est que nous n'avons pas expérimenté la modélisation logique de cette strate, les composants graphiques étant, souvent, directement programmés (aspect logiciel).

On peut soutenir, cependant, l'intérêt qu'il y a à décrire *logiquement* le dialogue homme-machine, c'est-à-dire sans hypothèse quant à la réalisation. Une telle description n'a pas à se limiter à l'interface homme-machine mais peut s'étendre à tous les types d'interfaces, notamment avec les connexions à des systèmes externes.

Origine

La strate « Présentation » exploite, également, la Vue de l'utilisation. À ce niveau, les relations entre les cas d'utilisation se matérialisent plus durement dans l'IHM, soit statiquement par composition de l'interface, soit dynamiquement par la cinématique. Le cas d'utilisation exprime, souvent, plus de contraintes que ce qui s'impose réellement. Son expression a tendance à rigidifier la façon de travailler. Ceci tient à deux raisons :

- D'abord, les cas d'utilisation restituent surtout les pratiques existantes. S'y mêlent habitudes de travail et traces d'anciennes décisions.
- Ensuite, la forme même de l'expression conditionne la description : le cas d'utilisation ou le scénario est « raconté », exposé linéairement[84].

La machine logique de la strate « Organisation », à travers son automate, ne retient que ce qui s'impose vraiment en tant que logique procédurale. Ceci, sous la forme d'un automate à états. Cependant, on peut souhaiter contraindre davantage le dialogue homme-machine, par exemple pour se conformer aux habitudes de travail. Ceci s'obtient par le biais de l'IHM, qui met en scène les services externes. Bien sûr, elle respecte l'automate à états ; elle peut, en plus, guider pas à pas l'utilisateur et simplifier le cheminement en interdisant certaines possibilités de navigation.

En respectant cette séparation, les MLO sont plus réutilisables, d'un organisme à l'autre.

La dérivation à partir des SFD

Une approche plus classique des spécifications fonctionnelles souffrent plus encore de la tendance décriée à propos des cas d'utilisation.

Les spécifications fonctionnelles, sous forme tectuelle, proposent une vision externe du système. Elles conditionnent, donc, directement la strate « Présentation ». Par une approche bottom-up (plus exactement, externe-interne), le concepteur détermine les besoins en services de MLO pour faire fonctionner l'interface.

L'état de la MLO peut être exploité pour ajuster le comportement de l'IHM. Les possibilités d'action s'ouvrent ou se ferment en fonction de la valeur de l'état. Ceci suppose que la MLO fournit son état, via un service qui peut être générique ou, plus simplement, par la structure de données qui lui est associée.

[84] Cf. le *Guide de l'aspect pragmatique*, référence « PxM-20 ».

Index

P

R

S

T

U

V