UML FOR BUSINESS INTELLIGENCE PROJECT

Abstract : this document deals with the role of UML into business intelligence projects (like data warehousing). After a quick overview of what UML offers, it focuses on the specific needs for BI projects, and how-and-why UML can be used for it. OO paradigm is discussed. Our conclusion are summarized by two tables at the end of the document.

SOMMAIRE

1. Introduction

Cet article est une analyse de l'apport d'UML pour la conception de systèmes décisionnels : UML et son approche objet répondent-ils aux problématiques de la modélisation d'un système décisionnel ? Après une mise au point sur l'UML, nous nous intéresserons plus particulièrement aux problématiques de l'expression de besoin, de la modélisation de données et des architectures de systèmes décisionnels. Nous évoquerons également les principaux outils logiciels utilisables dans le cadre d'une conception UML.

Nos conclusions sont résumées dans un tableau en fin de document. Un glossaire permet au lecteur peu familier avec les concepts UML de s'y retrouver.

2. Le point sur l'UML

L'UML, Unified Modeling Language, est un langage formel qui permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation. Né de la fusion des méthodes objet dominantes (OMT, Booch et OOSE), puis normalisé par l'OMG en 1997, UML est rapidement devenu un standard.

Un système est décrit en UML à travers sa **structure statique** et son **comportement dynamique**. UML propose donc un ensemble de notations graphiques pour capturer les informations relevant des aspects statiques et dynamique du système. C'est donc un langage de modélisation visuel et non une méthode d'analyse et de spécification. UML possède Néanmoins une dimension méthodologique et son utilisation s'inscrit dans une démarche :

- guidée par l'expression de besoin des utilisateurs du système
- centrée sur l'architecture logicielle
- itérative et incrémentale (UML supportant très bien l'abstraction)

2.1. Les diagrammes

2.1.1. Structure statique

Cette vue du modèle comporte trois types de diagrammes :

Diagrammes d'objets : Ce type de diagramme UML montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets. Ils s'utilisent pour montrer un contexte particulier du système après une exécution.

Diagrammes de classes : La classe est le concept fondamental permettant de modéliser des caractéristiques et des comportements communs à plusieurs objets. Les diagrammes de classes expriment d'une manière générale la structure statique d'un système en termes de classes et de relations entre classes. On peut construire des diagrammes de classes en se focalisant sur les classes qui participent à un même cas d'utilisation, un même paquetage, ou un même scénario.

Diagrammes de cas d'utilisation : Ils correspondent au modèle conceptuel. Les use cases permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système. Ils centrent l'expression des exigences du système sur ses utilisateurs et partent donc du principe que les objectifs du système sont tous motivés. Ces diagrammes permettent une meilleure compréhension du système, servent d'interface entre tous les acteurs du projet, et sont des éléments de traçabilité vis à vis de l'expression de besoin.

1 cas d'utilisation est une manière spécifique d'utiliser un système. L'ensemble des cas d'utilisation décrit donc le comportement du système du point de vue d'un utilisateur.

2.1.2. Comportement dynamique

Cette vue du modèle comporte quatre types de diagrammes :

Diagrammes de séquence : Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel. Ils représentent les échanges de message entre objets au cours du temps. Ils s'utilisent de 2 manières : pour la documentation des cas d'utilisations (scénarios) et pour la représentation précise des interactions entre objets (usage plus informatique).

Diagrammes de collaborations : Par rapport aux diagrammes de séquences, ici le contexte des objets est représenté de manière explicite. Ces diagrammes montrent des interactions entre objets dans un contexte (état des objets) particulier.

Diagrammes d'états-transitions : Ils constituent une modélisation du comportement de l'objet et relient des événements à des états en spécifiant la séquence d'états provoquée par une séquence d'évènements. Ils servent donc à représenter des automates à états finis. En UML ces diagrammes sont associés à une classe !

Diagrammes d'activités : Une activité représente l'exécution d'un mécanisme, un déroulement d'étapes séquentielles. Ces diagrammes permettent de représenter graphiquement le déroulement d'une méthode ou d'un cas d'utilisation. Ils sont une variante des diagrammes d'états-transitions organisée par rapport aux actions.

2.1.3. Constructions d'implémentation

Cette vue du modèle comporte deux types de diagrammes :

Diagrammes de composant : Ils permettent de décrire l'architecture physique et statique d'une application en termes de modules de différentes nature (fichiers sources, librairie, exécutables…). Ils montrent la mise en œuvre physique du système avec son environnement de développement.

Diagrammes de déploiement : Ils montrent quant à eux la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels. Ils mettent ainsi en évidence l'arrangement physique des ressources d'exécution (les machines et leurs interconnexions)

2.2. Intérêts d'UML

UML est un langage formel et normalisé (notation ouverte et riche qui permet un gain de précision et de stabilité), mais c'est aussi un support de communication performant qui cadre l'analyse et facilite la compréhension de représentations abstraites. C'est donc un excellent moyen de communication dans une équipe qui donne à travers sa représentation d'un système, une bonne compréhension de son fonctionnement.

Enfin UML, de par sa notation ouverte, offre une grande polyvalence dans ses domaines d'applications. A l'inverse, cette souplesse qui en fait un langage « universel » entraîne une sémantique un peu «floue » des entités de modélisation. (On doit les interprèter selon le domaine et la nature du système à modéliser)

C'est donc ce qui motive cette étude, donner un sens à la modélisation en UML dans un contexte de conception de data warehouse et de modélisation de données (« data modelling »).

3. Expression du besoin client

L'expression de besoin dans la conception du data warehouse au niveau de la structuration des données donc au niveau des données décisionnelles revient à identifier les données porteuses de sens pour le client. L'expression de besoin correspond donc ici à un ensemble de données spécifiques requises par des acteurs de l'entreprise pour l'exercice de leur métier (Marketing, qualité, finance, …).

3.1. Données décisionnelles

L'exercice de ces métiers passe par l'analyse de ces données décisionnelles qui se déclinent en trois types distincts :

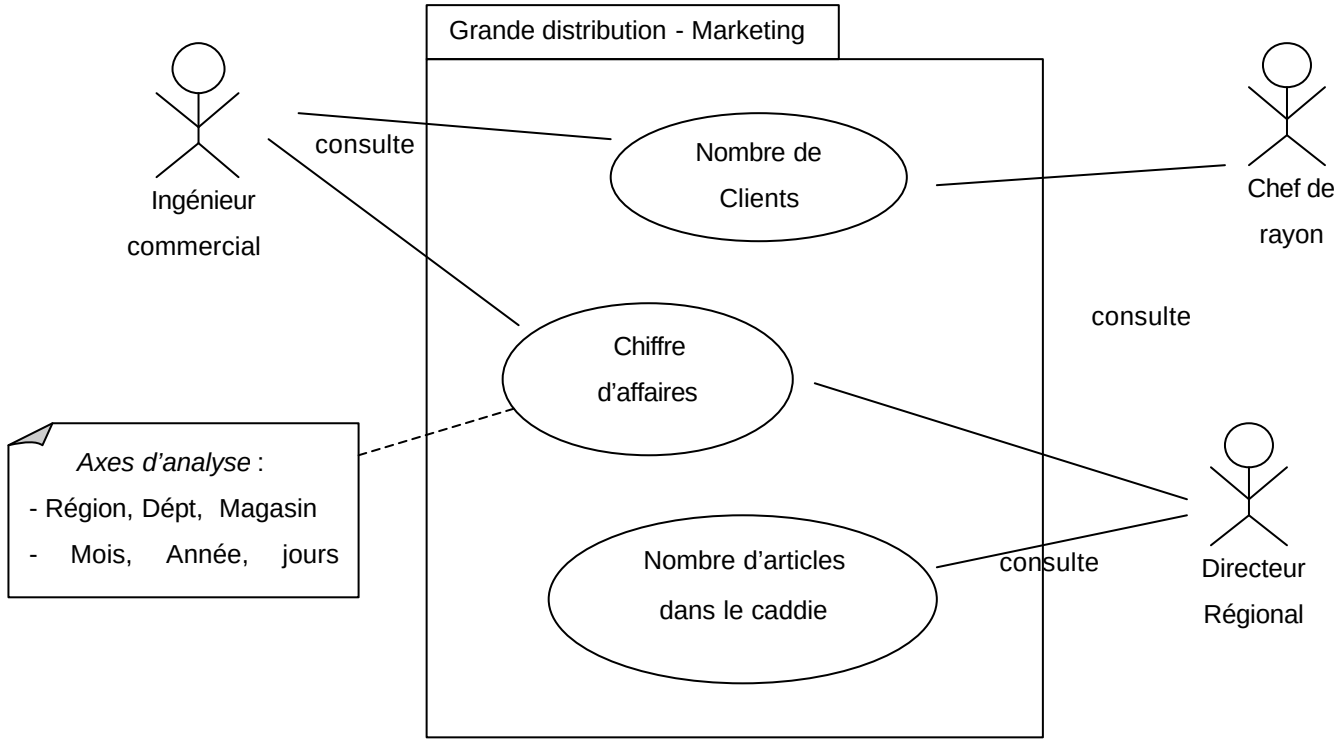
- Indicateurs
- Axes d'analyse
- Filtres

La donnée porteuse de sens qui oriente les choix dans un métier donné est l'indicateur. Or la définition d'un indicateur ou la règle de gestion qui permet de l'obtenir à partir des données opérationnelles peut varier d'un métier à l'autre au sein de l'entreprise. Elle se rattache à un métier et donc à un acteur ou utilisateur du data warehouse. On remarque donc à ce niveau de l'expression de besoin que des ambiguïtés sémantiques au niveau des indicateurs existent et que la prise en compte des acteurs est essentielle pour les éliminer. L'acteur est le seul à connaître la règle de gestion qu'il utilise pour calculer tel ou tel indicateur. C'est la modélisation du métier et du savoir faire de chacun qui entre ici en jeu. Chacun à sa propre vision de l'organisation, du sens des termes et indicateurs de son métier.

Les cas d'utilisations qui permettent en UML de formaliser l'expression de besoin s'inscrivent bien dans cette démarche puisque l'identification des acteurs est une étape cruciale de leur mise en

œuvre. Le système est représenté à l'origine par tous ses cas d'utilisation qui décrivent une interaction avec ses acteurs.

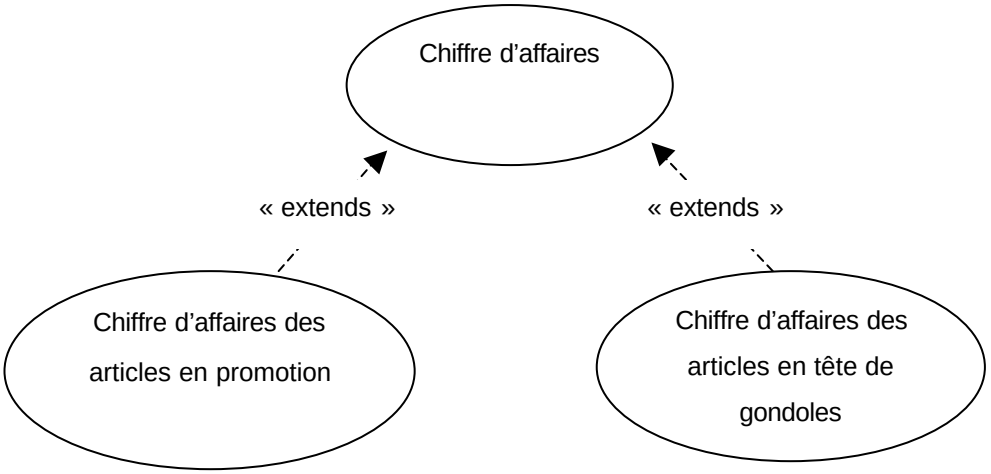
On choisit d'interpréter chaque indicateur comme un cas d'utilisation. Chaque cas d'utilisation est relié à l'acteur qui y fait appel. On regroupe les indicateurs selon des packages (dans l'exemple : Marketing) qui font intervenir les même acteurs et qui touchent un ensemble cohérent de besoins.



La description textuelle des cas d'utilisation est cruciale et correspond à la base de l'analyse des besoins et de la discussion avec les acteurs. Elle concerne ici essentiellement l'association des axes d'analyses et des filtres à chaque indicateur mais aussi de la règle de gestion qui permet de le calculer.

Sur l'exemple précédent, la vue des acteurs met en évidence les questions concernant les indicateurs « Nombre de Clients » ou «Chiffre d'affaires ». Le chef de rayon et l'ingénieur commercial compte-il les clients de la même façon ? On pourrait très bien imaginer que l'ingénieur commercial prend en compte les personnes morales en plus des personnes physiques que considère le chef de rayon bricolage. Mais tout deux parlent de clients.

Toujours dans le souci d'organiser et clarifier les besoins on peut chercher à structurer les indicateurs et à donner prévoir le grain des faits déjà à ce niveau.



Le caractère « être en promotion » n'est pas ici un axe d'analyse du chiffre d'affaire dans la mesure où le chiffre d'affaires des articles en promotion constitue une donnée directement exploitée par un acteur en tant que donnée décisionnelle. Ainsi on pourra choisir par la suite stocker à la fois un chiffre d'affaires et un chiffre d'affaires des articles en promotion dans la table des faits.

3.1.1. Limites

Un cas d'utilisation est une manière d'utiliser ou d'interagir avec le système. Il est constitué de différents scénarios décrivant les différents cas de figure. C'est la modélisation fondée sur les scénarios qui fait la force d'UML et qui offre une bonne compréhension de « ce qui se passe » dans le système. Or il n'y a pas ici de système à proprement parler, seule l'identification de l'indicateur et des ambiguïtés entre règles de gestion nous intéressent. Les tâches fondamentales que souhaite faire l'acteur se résument ici à consulter des données.

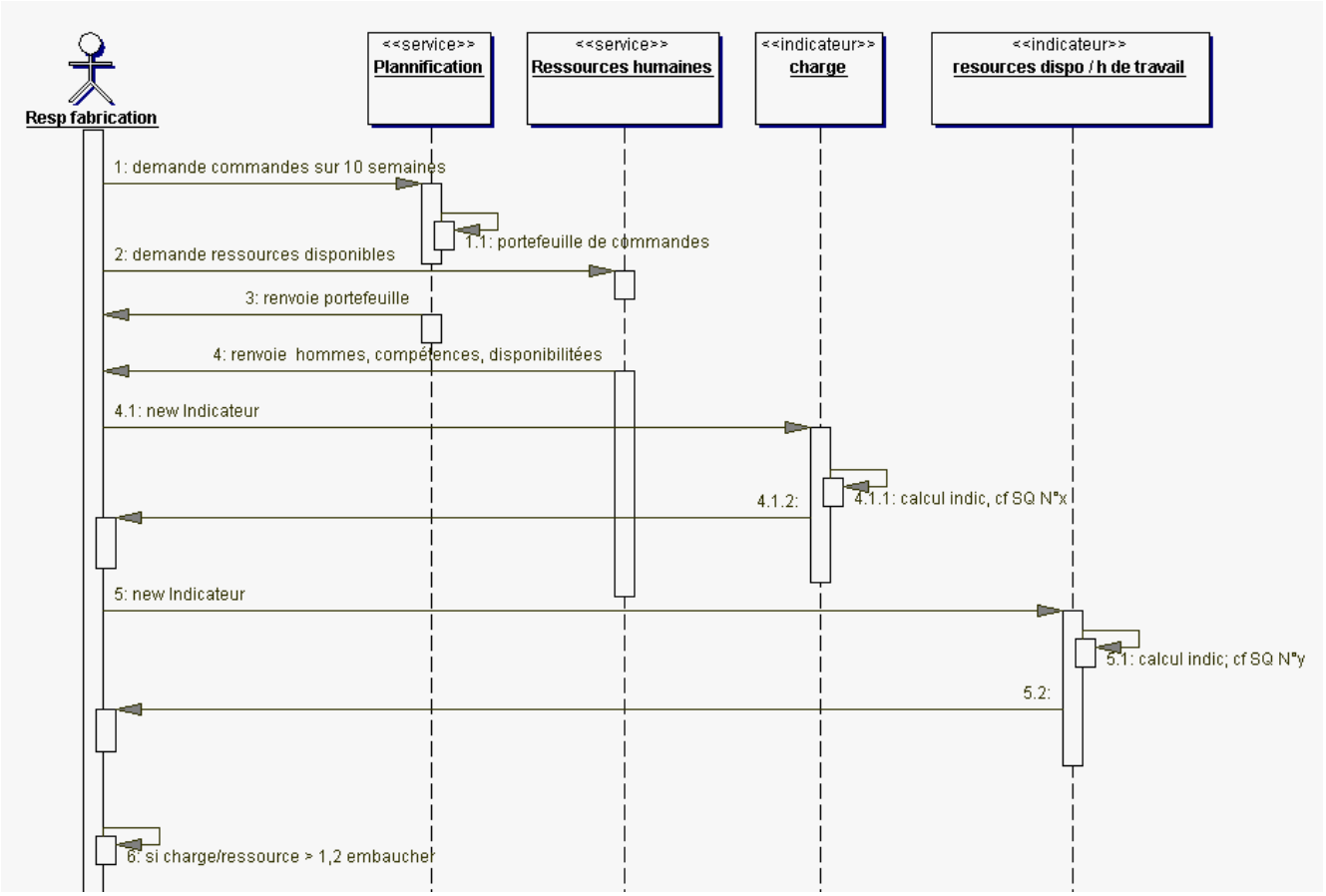
3.2. Expression des règles de gestion

Etant donné que le use case constitue le use case, on pourrait imaginer que le calcul de la règle de gestion en tant que scénario avec l'expression des cas particuliers dans des scénarios alternatifs ? Ce qui permettrait ainsi de tenir compte de l'existant.

3.3. Modélisation du métier

On souhaite s'intéresser à **Pourquoi l'utilisateur veut accéder à tel ou tel indicateur** et modéliser sous forme de scénarios l'analyse possible via des indicateurs et des axes successifs. Les scénarios serviraient ici la modélisation du métier de l'acteur. Imaginons un responsable fabrication qui gère ses effectif en effectuant d'une part un calcul de charge de travail vis à vis des commandes fournies par le service planification et un calcul sur les ressources humaines disponibles. La comparaison de ces deux indicateurs lui permet de prendre la décision de d'embauche d'intérims par exemple. On comprend ainsi pourquoi les indicateurs entrent en jeu, une expression de d'un tel besoin pourrait se

faire sous forme de diagrammes de séquence. Il s'agit bien ici d'une règle métier qui combinent d'autres règles.



3.4. Conclusion

L'UML apporte :

- La vue des acteurs
- La structuration des besoins sans chercher l'exhaustivité

Cependant, on notera l'absence du point fort du modèle conceptuel d'UML, les **scénarios** : cela découle que le besoin n'est pas ici réellement la modélisation d'un système ni de processus métiers mais l'identification de données, les indicateurs et leurs axes.

L'UML seul ne présente pas d'intérêt fort au niveau de la modélisation des besoins décisionnels. Il ne serait intéressant que dans la mesure où les cas d'utilisations seraient intégrés automatiquement avec la modélisation, ce qui n'est pas réalisable sans un modèle au-dessus d'UML.

4. Architecture du data warehouse

4.1. Déploiement

Les éléments à modéliser en data warehouse du point de vue de l'architecture sont principalement :

- ?? ODS
- ?? Data Mart
- ?? Applications opérationnelles
- ?? Serveur et base de données (OLAP ou SGBDR)

4.1.1. Analyse

La modélisation doit donc prendre en compte les aspects suivants pour donner une vision pertinente du problème de la conception d'une architecture en data warehouse :

- La vision des outils de collecte, intégration et valorisation
- Les technologies pour éviter des problèmes d'incohérence dans l'utilisation de technologies hétérogènes avec l'existant.
- Les flux et les modèles (opérationnels, multidimensionnels, problèmes de synchronisation)

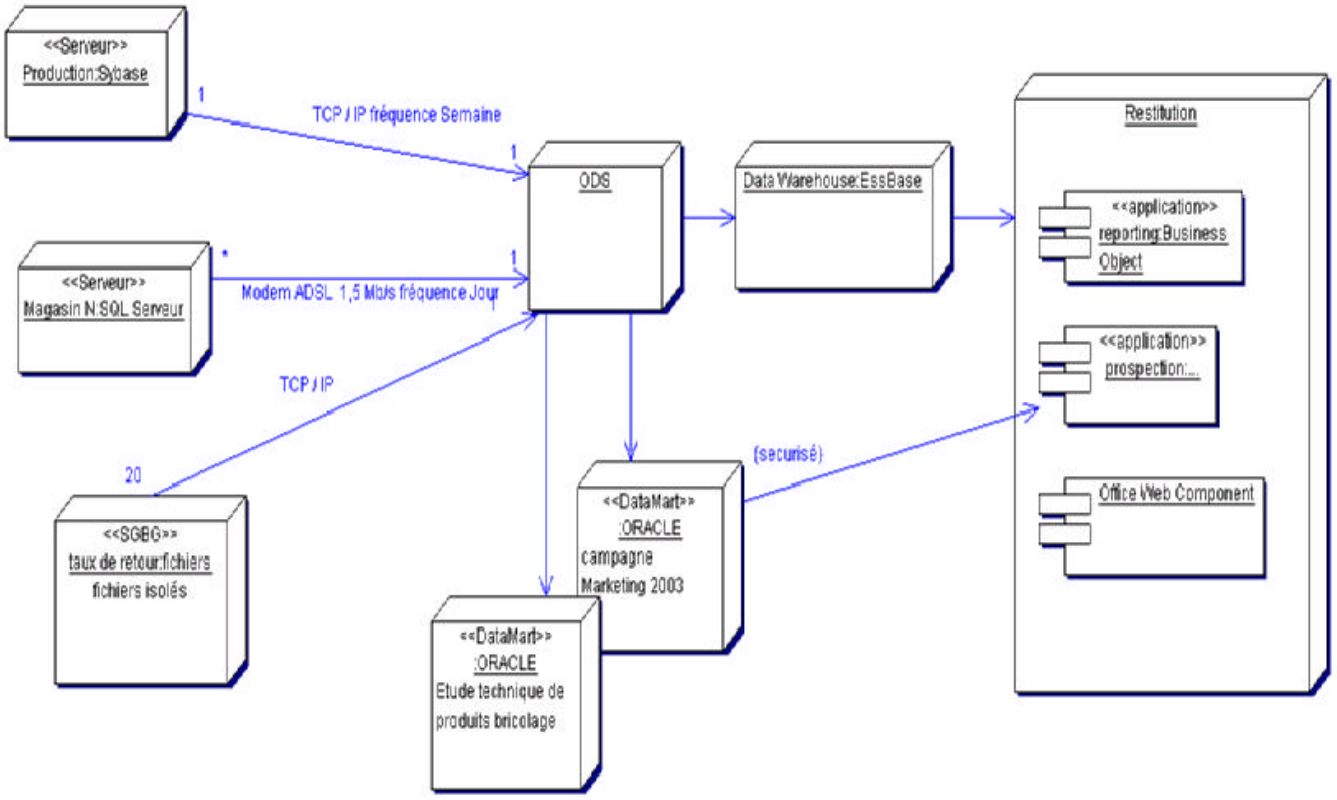
En UML, la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels est décrite à travers les diagrammes de déploiement.

Les ressources matérielles sont représentées sous forme de nœuds et la nature des lignes de communication et leurs caractéristiques peuvent être précisées. **Point positif** : Les diagrammes de déploiement peuvent montrer des instances de noeuds (un matériel précis), ou des classes de noeuds.

Autrement dit l'abstraction joue ici un rôle important puisque chaque nœud peut ainsi être détaillé en composants pour une modélisation plus fine, et ainsi de suite. Le détail des flux par exemple peut être précisé à une itération suivante de la conception du data warehouse. Mais si l'on considère que l'alimentation de l'ODS est une opération à lancer très tôt, on peut s'intéresser immédiatement à rendre compte de l'existant et des contraintes technologiques qui pèsent sur cet existant.

La collecte en lot peut très bien être modélisé par les mêmes diagrammes à un détail plus fin en considérant chaque lot comme un composant par exemple.

4.1.2. Exemple

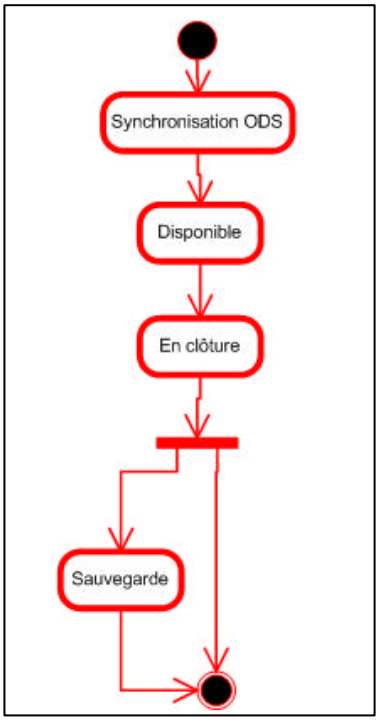

<<Serveur>>
Production:Sybase
1
TCP/IP fréquence Semaine
1
ODS
Data Warehouse:EssBase
Restitution
<<application>>
reporting:Business Object
<<Serveur>>
Magasin N:SQL Serveur
*
1
Modem ADSL 1,5 Mbits fréquence Jour
<<application>>
prospection:...
TCP/IP
20
(securisé)
<<DataMart>>
:ORACLE
campagne Marketing 2003
Office Web Component
<<SGBG>>
taux de retour:fichiers
fichiers isolés
<<DataMart>>
:ORACLE
Etude technique de produits bricolage

4.1.3. Conclusion

Par rapport au besoin du data warehouse en terme de modélisation d'architecture, l'approche d'UML sous la forme de diagrammes de composants et de déploiement n'a rien d'exceptionnelle mais répond au besoin de l'expression des technologies, des flux et des contraintes sur les éléments de l'architecture.

4.2. Scénarios de reprise sur incident

A notre sens, l'UML présente un fort intérêt pour concevoir et représenter toutes les opérations de reprises sur incident (ou exceptionnelles).



Ces opérations nécessitent une description de l'état du système, des transitions entre les états, ainsi qu'une représentation chronologique des opérations à effecteur. UML fournir les diagrammes adaptés à ces besoins : les diagrammes d'états et les diagrammes de séquence. Même les transitions complexes (divergences en ou, en et, en ou exclusif par exemple) sont représentées simplement.

Il est possible de représenter les différentes étapes dans un formalisme assez intuitif. Le schéma ci-contre montre le cas d'une solution technique ou la sauvegarde à chaud n'est pas possible : l'état «sauvegarde en cours » n'est atteignable qu'après être passé par la phase de clôture. On peut l'agrémenter de notes indiquant les heures de la journée qui correspondent aux différentes phases.

Bien qu'il soit possible d'exprimer formellement les conditions de transitions (à l'aide de l'OCL) nous le déconseillons (voir 5.2.3).

Le diagramme de séquence à l'avantage de représenter chronologiquement les interactions entre les parties du système et l'utilisateur, ce qui rend clair un scénario de reprise sur incident. Ici, on a voulu modéliser le comportement d'un système data warehouse lorsqu'une donnée source n'est pas disponible au moment de l'alimentation.

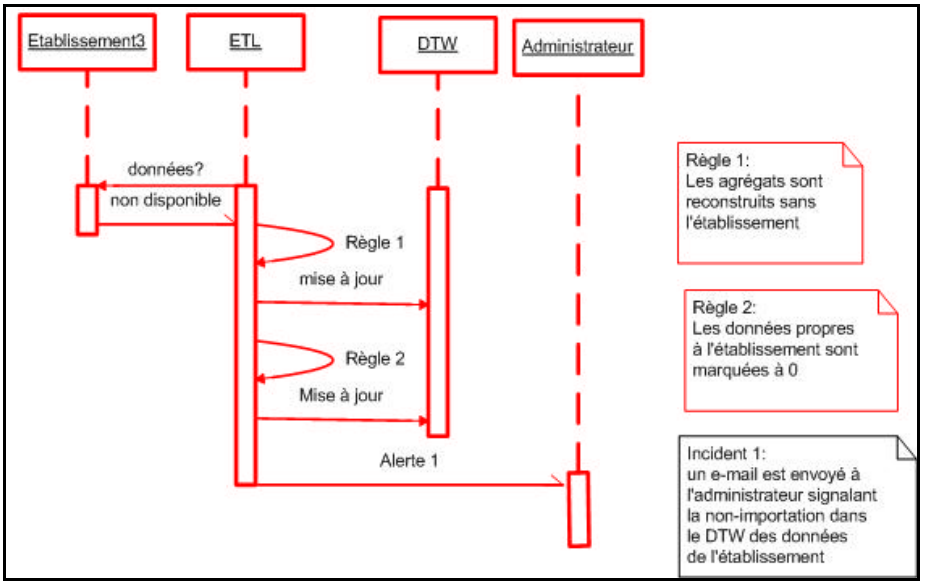


Diagramme de séquence

5. Modélisation DB

Nous allons nous intéresser dans un premier temps à la réalisation de modèles conceptuels et physiques en BDD relationnelle ou multidimensionnelle dans le cadre de la conception d'une base de données sur un SGBDR.

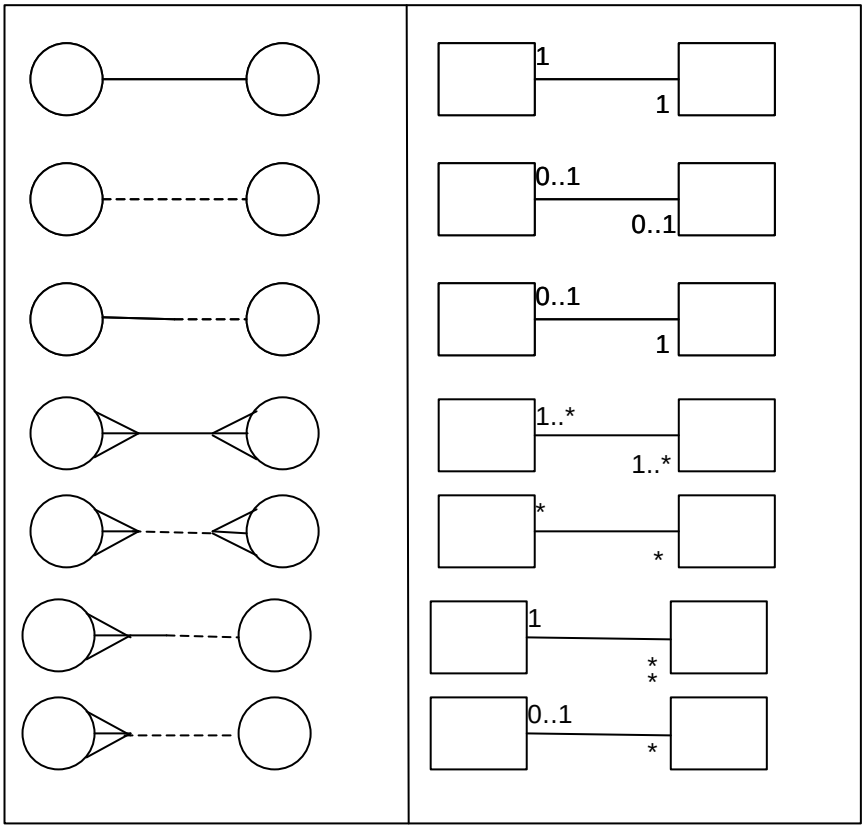
Ce choix est fondé sur le fait que les bases de données en production sont des bases de données relationnelles. Celles-ci sont des modèles « plats » (toutes les données sont accessibles entre entités, il n'y a pas de hiérarchie forte entre entités) qui réduisent la portée de l'UML, comme nous allons le voir. Certaines solutions (JDeveloppper) permettent de modéliser en UML des classes Java sur une base de données Oracle ; mais ce cas où UML prend ton son intérêt reste une exception.

5.1. Modèle conceptuel relationnel

Tout d'abord, les diagrammes ERD qui constituent l'approche classique en matière de conception de base de données relationnelle, sont une instance particulière de l'UML : ce qui est faisable en ERD est faisable en UML. Nous proposons d'ailleurs des tables de traduction ERD/UML qui illustre ce fait. La notation est un peu plus lourde en UML en raison de l'approche générique de ce langage de modélisation. Des extensions spécifiques à la modélisation des données sont prévues avec la norme UML 2.0 pour pallier ce défaut.

Terme ERD	Terme UML
entité	classe
Instance d'entité	Objet
relation	Association
Supertype/subtype	Généralisation/Spécialisation
Dépendance de relation	Composition (en général)
Attributs	Attributs

On peut traduire également les symboles ERD en UML :

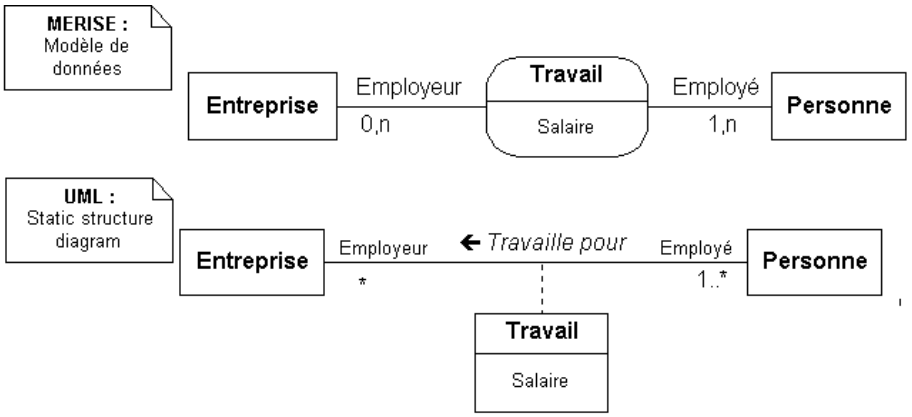


Traduction ERD / UML

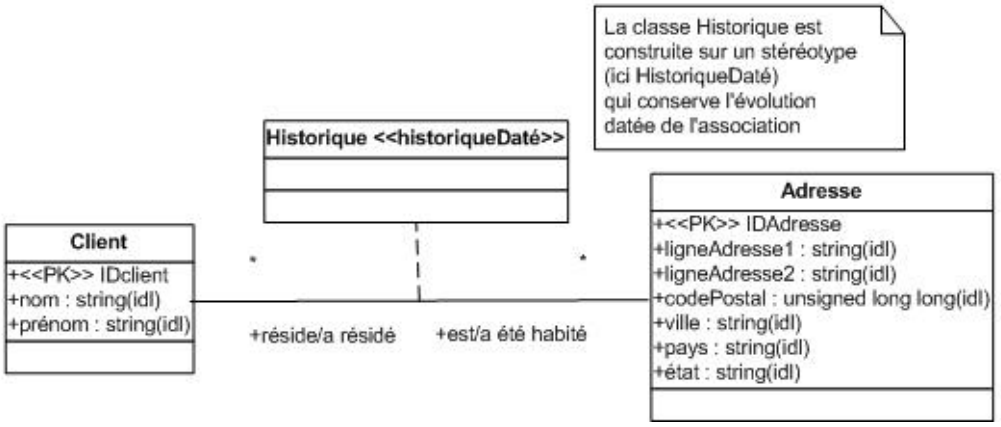
Cependant lorsque que l'on veut aller plus loin (représentation de contraintes comme les clés externes), l'absence de sémantique associée à l'UML complique les choses. On trouve alors des variantes de notations UML pour la modélisation de données (voir la proposition de Rational).

5.1.1. Exemples

Comparaison d'une relation entre une Personne et une Entreprise ; dans un cas on utilise les standards Merise, dans l'autre l'UML.



On notera la différence sur la modélisation de l'association. Un exemple de modélisation UML utilisant une relation historisée suit :

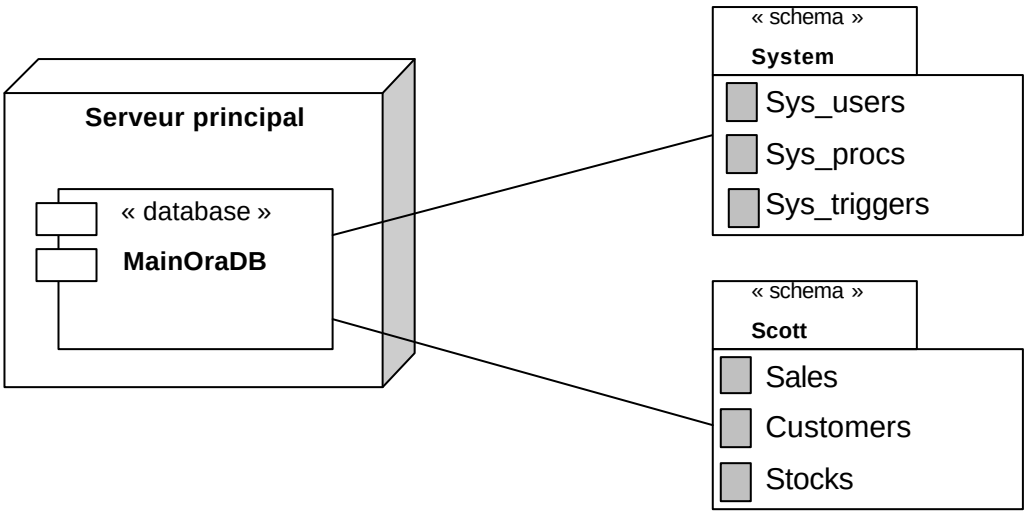


Par rapport au même modèle avec Merise, on gagne en lisibilité et en concision : toutes les entités nécessaire à un historique daté n'apparaissent pas, on voit juste une classe de stéréotype *historiqueDaté*, lequel est défini une fois pour toute ailleurs dans le document.

5.2. Modèle physique relationnel

Pour la représentation physique des tables, Oracle et Rational Rose proposent des sémantiques très proches. Le principe consiste à utiliser des opérations stéréotypées pour modéliser le comportement de la base de données : on représente ainsi les contraintes de clé primaire, de clé étrangère, les index, les triggers, la contrainte d'unicité et les contraintes de type CHECK. Les classes correspondant aux tables utilisent le stéréotype TABLE.

Les diagrammes de composants UML permettent la représentation des différents schémas de la base de données, ainsi que des processus de la base. Par exemple :



5.2.1. Apports de l'OOD

Examinons l'apport en décisionnel des notions propres à l'OOD : l'agrégation, l'héritage et les modes de visibilité des données (voir Glossaire, « encapsulation »).

Les limitations sur l'accès aux données sont dans la majorité des cas pertinentes vis-à-vis de l'**utilisateur** et non pas **des entités du modèle**, et les outils relationnels permettent de gérer cela via les vues ou les droits sur les tables du modèle physique. Mais notons que la gestion de différent niveaux de confidentialité des données (et la problématique des données calculées par agrégation) se traite mieux par une approche objet. Des travaux dans ce sens sont proposés en Bibliographie, SECURED DATA WAREHOUSING.

Exemple de problématique : si le chiffre d'affaires par magasin est une donnée Confidentielle, le nombre de clients par magasin une donnée non sécurisée, quel est le niveau de confidentialité de l'agrégat valeur moyenne du caddie ? Quelles entités peuvent calculer cet agrégat ?

Les entités manipulées ne sont pas les parties d'un même système mais **des indicateurs calculés par des règles métier** difficilement «factorisables ». Par ailleurs il y a peu d'intérêt à définir des méthodes sur des indicateurs ; les seules pertinentes pourraient être celles liées à l'historique de celui-ci, de type *double evolution(anneeDebut, anneeFin)* qui calculerait l'évolution d'un indicateur. Mais sur une base de données relationnelle (implémentation en PL/SQL par exemple), il est plus pertinent de laisser cela aux logiciels en *front-end* (autonomie des utilisateurs plus grande, meilleure flexibilité). Notons enfin la traduction des relations d'héritage/agrégation n'est pas triviale, **ni automatisée** avec les logiciels du marché.

Par conséquent, les notions d'agrégation/héritage/visibilité des données **ne font pas la différence** dans une logique basée sur le métier des utilisateurs.

5.2.2. Variabilité des besoins

La variabilité des besoins n'est pas sensiblement mieux traitée par la modélisation objet. Une alternative de l'UML, l'**ORM** (voir Glossaire) apporte des éléments intéressants dans ce sens mais elle n'est **pas intégrée** dans l'offre logicielle actuelle. Les modifications à apporter en cas de prise en compte d'un nouvel indicateur sont quasiment les mêmes entre un modèle ERD et un modèle UML.

Exemple : supposons que l'on ait une entité Personne, dont on veut stocker la nationalité. La modélisation E/R ou UML modélise la nationalité comme un attribut de l'entité Personne. Si plus tard nous décidons d'enregistrer toute la population, il est plus pertinent d'utiliser une entité/classe Pays séparée de l'entité/classe Personne. Dans les deux cas, il faut créer quelque chose de nouveau et définir une relation entre l'existant et le nouveau. Cela rend les requêtes existantes caduques et conduit aux mêmes modification. L'ORM supprime la notion d'attribut et permet de traiter ce changement de besoin comme une simple modification de la nature d'une relation.

L'apport d'UML est donc à chercher dans sa faculté de capturer les contraintes propres au décisionnel (règle métier complexes) de manière plus efficace que les modèles ERD.

5.2.3. Expression des règles métier

Il est possible en UML de définir ses propres contraintes (sous forme de stéréotypes), ce qui permet d'élargir la panoplie ERD (unicité, non nul, clé étrangère, clé primaire…). L'UML permet de s'affranchir des contraintes de normalisation des ERD ; mais dans la mesure où il est utilisé sur une base de données relationnelles, ces contraintes réapparaîtront lorsque l'on voudra générer le modèle physique !
Notons donc que l'UML permet de modéliser sur un même schéma les entités conceptuelles et les règles métier, contrairement à l'ERD qui conduit à des formulations séparées.

Le langage formel d'expression de contraintes d'UML, l'OCL, n'est pas à recommander à cause de sa difficulté de manipulation qui peut causer des ambiguïtés entre les acteurs du projet. Il nécessite une certaine pratique. Dans la mesure où il n'y a pas d'outil de validation des contraintes exprimées sous forme algébrique, nous préférons utiliser des stéréotypes avec des descriptions en langage naturel. C'est également l'opinion de P. Stevens et R. Pooley (références mondiales sur l'UML) dans *Using UML.*

- ?? ORM plus pertinent que l'UML ? Les Business Process n'apparaissent pas clairement, programmation par aspects ? Problème : il n'est pas intégré dans les outils actuels.

5.2.4. Conclusion

Dans l'état actuel, utiliser l'UML comme langage de modélisation de données apporte plus d'inconvénients (passage au modèle physique délicat) que d'intérêts.

Les extensions de l'UML sont à surveiller car leur intégration dans des outils tels que PowerAMC ferait de l'UML une alternative crédible.

5.3. Modélisation multidimensionnelle

Sur un schéma UML il est possible de d'indiquer la nature additive, semi-additive ou non additive des indicateurs. Cela a un intérêt tout particulier en phase de design. La possibilité de définir ses types de relations offre un intérêt dans la représentation de relations historisées entre entités (plus grande légibilité qu'un schéma E/R).

5.3.1. Etoiles en UML

Il n'est actuellement pas possible de générer d'étoiles à partir de diagrammes UML avec les outils du marché. Des travaux pour définir un modèle UML capable d'assurer la traduction existent ; on peut sans s'avancer prédire que ces outils se développeront rapidement (une implémentation existe pour SQL Server, voir Bibliographie / CWM group).

5.3.2. Modélisation des hiérarchies

C'est un domaine de recherche actif. Nous n'entrerons pas dans le débat de l'utilité d'un modèle conceptuel de base de données multidimensionnelle.

L'UML permet de modéliser les niveau de hiérarchies par des classes et de définir des relations spécifiques au multidimensionnel, tels que le *drill-down* sur une classe Cube, qui dérivent des concepts objets (agrégation, spécialisation). Le modèle UML en permettant des types de relations plus fins que les modèles relationnel donne plus d'informations visuelles à l'utilisateur. Cela peut être pertinent pour des utilisateurs experts, car une hiérarchie explicite et des informations sur la modélisation sont des atouts pour écrire et interpréter des requêtes ad-hoc. Dans des problématiques de performances, où l'éclatement des tables de dimensions est nécessaire, la modélisation UML permet de construire un modèle plus complexe (dimensions **spécialisant** d'autres dimensions par exemple).

Exemple : un modèle conceptuel en flocon peut faire apparaître une dimension (sous forme d'une classe Personne) qui se ramifie en Employé et Client (sous-classes). Il est possible de générer des requêtes impliquant simultanément les deux entités Employé et Client (l'usage d'un attribut dans Personne est plus limitatif et complique la navigation dans les données). Cela permet entre outre la parallélisation des requêtes (un processeur traitant la partie de requête impliquant Employé, l'autre Client).

5.3.3. Conclusion

La modélisation multidimensionnelle reste très peu développée en UML, bien qu'il y ait de fortes potentialités dans ce domaine. Nous retenons que l'apport de l'UML dans la conception de base de données multidimensionnelle réside dans la description formelle des hiérarchies dans les structures

en flocons. La nécessité de définir soi-même sa sémantique UML en multidimensionnel limite néanmoins la portabilité des solutions.

6. Synthèse

L'UML AUJOURD'HUI

Avantages	Inconvénients
Capture des règles métiers sur les schémas de modélisation de données	Les notions objets ne sont pas utiles sur une base de données relationnelle.
Modélisation des états d'un data warehouse et de la reprise sur incidents	L'UML n'est que faiblement intégré dans les outils de conception de base de données
Modélisation des hiérarchies en flocons explicite	**Il n'y a pas de sémantique « universelle »**
Capture des besoins utilisateurs	Peu d'intérêt du point fort du modèle conceptuel UML : la modélisation par scénario

DIAGRAMMES UML PAR BESOIN DECISIONNEL

1 : apport significatif 2 : apport limité 3 : aucun apport 4 : perte de temps

Diagramme UML	Utilisation possible en décisionnel	Intérêt
Diagramme de classe	Modèle conceptuel de base de données relationnelle	3
	Modèle physique de base de données	4
	Modèle conceptuel de base de données multidimensionnelle	2
Diagramme de séquence	Modélisation des flux de données pour operational data store / data warehouse / Data marts.	2
	Modélisation des règles de gestion, illustration de calcul d'indicateurs (use cases)	2
Diagramme d'objet	Pas d'utilisation	4
Diagramme de collaboration	Modélisation détaillée des flux	3
Diagramme de déploiement	Architecture du système décisionnel d'un point de vue physique.	1
Diagramme de composants	Représentation des éléments de l'architecture physique (les différentes applications qui viennent utiliser le système, les postes utilisateurs)	1
Diagramme d'états-transitions	Représentation de la disponibilité du système	1
Diagramme de cas d'utilisations	Enoncé du besoin utilisateur	2
Diagramme d'activité	Règles de reprise sur incident	1

7. Perspectives

L'UML propose une large panoplie de diagrammes dont nous avons étudié la pertinence. Mais les limitations dues à sa généralité constituent un frein à son utilisation en data warehouse. C'est pourquoi de nombreux travaux visent à l'étendre pour construire une norme spécifique aux besoins en conception de data warehouse.

La définition d'une sémantique pour le data warehouse comprenant des mécanismes de traduction automatique, et prenant en compte les métadonnées existe : nous recommandons au lecteur désireux d'utiliser UML de s'appuyer sur les modèles proposés par le **CWM group**. La réalisation d'outils de développement implémentant ce modèle constitue à notre sens l'avenir en conception de data warehouse !

Enfin la norme UML 2.0 devrait proposer des extensions pour permettre une modélisation de données relationnelles plus efficace.

8. Principaux outils pour faire de l'UML

Outils	***Description***	***Points Forts***	***Points Faibles***
Rational Rose *Prix :* *~2500 $*	Rational fournit une méthode de conduite de projet informatique entièrement basée sur l'UML, qui prend en charge tous les aspects liés au travail collaboratif. Son progiciel Rose permet donc naturellement la réalisation de diagrammes UML. C'est avec cet outil que travaillent les inventeurs de l'UML	**Nombreux langages** **Contrôle de version**	**Interface**
Together *Prix :* *~6000 $*	AGL fondé sur l'UML. Il inclut tous les diagrammes standards UML des sémantiques prêtes à l'emploi pour les différentes phases œ qui facilite la prise à main	**Excellente interface** **Contrôle de version** **Rétro Ingénierie** **Liens avec Rose** **Supporte Java/VB/C++**	**Lenteur** **Génère du code « à la volée »** **Prix**
Visio *Prix :* *~500 $*	inclus dans Visual Studio .net. C'est un outils de dessin de diagrammes qui prend notamment en compte l'UML	**Génère du code C#**	**pas de sémantique livrée** **Pas de contrôle de syntaxe**
ArgoUML Prix : 0 $	Il s'agit d'un outil OpenSource écrit en java.	**Gratuit** **Simple d'utilisation** **Génération de code**	**Lenteur** **Faiblesse sur certains diagrammes**
UML Studio Prix : ~500 $	Cet outil est complètement dédié à UML.	**Génération de doc rtf ou html** **Création de stéréotype**	**Pas de vérification de modèle** **Navigation graphique limitée**

On pourrait également citer **Innovator, ObjectiF, Poseidon** parmi ceux qui sont généralement retenu dans la longue liste des outils intégrant l'UML aujourd'hui. Tous ces outils proposent des solutions qui mettent plus ou moins en avant différents aspects, globalement il s'agit de :

- La création, l'édition et la manipulation de diagrammes
- La génération de code
- La rétro ingénierie

- La génération de documentation
- Le contrôle de versions
- La vérification de modèle.

Le choix d'un outil repose donc surtout sur le besoin vis à vis de ces différents critères.

9. Bibliographie

Les documents ci-dessous sont disponibles sur Internet, le lecteur intéressé pourra les trouver avec l'aide d'un moteur de recherche :

- **COMMON WAREHOUSE MODEL**, http://www.cwmforum.org : norme reposant entre autres sur l'UML pour la modélisation des data warehouses.
- **UML EN FRANCAIS**, http://uml.free.fr : presentation générale de l'UML.
- **INFORMATION SYSTEM ARCHITECTURE FOR SECURE DATA WAREHOUSING**, Alberto Abello and Marta Oliva and Jose Samos and Felix Saltor
- **UML CLASS DIAGRAMS AND ORACLE8 DATABASE DESIGN,** *David A. Anstey, ARIS Corporation*
- **DATABASE MODELING IN UML,** Geoffrey Sparks, Spring 2001,Methods & Tools (guide pratique)
- **MAKING THE TRANSITION FROM ERD'S TO UML AND ORACLE'S DATABASE DESIGNER**, Dr. Paul Dorsey, Dulcian Inc.

10. Glossaire

Abstraction : L'abstraction est un des piliers de l'approche objet. Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité en vue d'une utilisation précise. L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

Abstraite (classe) : certaines classes ne peuvent pas permettre la création d'objet. Typiquement, elles servent de superclasses pour d'autres classes.

Agrégation : l'agrégation permet à une classe de contenir d'autres classes. Par exemple, la classe Segment peut agréger deux objets de classe Point pour son fonctionnement interne. Cela permet de factoriser les caractéristiques communes aux objets (meilleure maintenabilité).

Classe : c'est un patron qui peut servir à fabriquer des objets (dits instances de cette classe). Par exemple, la classe Employé peut être instanciée par les objets Jean et Jacques.

DB : acronyme de data base, base de données. Dans cet article nous nous plaçons dans le cadre des bases de données relationnelles, sauf mention explicite.

ERD : acronyme de *Entity Relationship Diagram*. C'est une méthode de conception très utilisée pour la conception de base de données (notamment à travers les diagrammes MCD de Merise).

Encapsulation : en OOD , une classe masque son fonctionnement interne pour ses utilisateurs. Ce masquage requiert une protection de certaines données (dites privées). L'encapsulation est le mécanisme qui permet de restreindre la visibilité de certains éléments.

Héritage : spécialisation de la définition d'une classe pour créer une autre classe. Exemple : la classe Employé qui hérite de la classe Personne (elle récupère tout le travail déjà fait sur personne pour gérer les adresses, par exemple). L'avantage de l'héritage sur l'agrégation réside dans le fait qu'un code fonctionnant avec des objets de classe Personne fonctionnera sans modification avec des objets de classe Employé.

Interface : une interface spécifie un ensemble de méthodes que doit implémenter une classe. C'est un contrat que doit respecter la classe.

Méthode : c'est une opération rattachée à une classe. Par exemple, *verserSalaire* peut être une méthode de la classe Employé.

Modèle : Représentation schématique d'un processus, d'une démarche raisonnée. Il s'agit donc une abstraction de la réalité (voir abstraction). C'est une vue subjective, mais pertinente de la réalité.

OOD : acronyme de *Object-oriented development.* Méthode de conception qui utilise les concepts propres à la programmation objet (voir Héritage, Encapsulation).

ORM : acronyme de *Object Relationship Model*. C'est une extension d'UML qui vise à modéliser en priorité les rôles et interactions des objets entre eux.

SGBDR : acronyme de Système de Gestion de Bases de Données Relationnelles. On peut citer comme SGBDR célèbres Oracle Database, SQL Server et DB2.

Stéréotype : mode d'associer un comportement à une classe, qui autorise également des représentations plus claires (icônes dans la case de nom de la classe).

Sous-classe : classe qui hérite d'une autre classe, dit superclasse.

Spécialisation : une classe qui hérite d'une autre classe «spécialise» la superclasse en rajoutant des fonctionnalités propres. Par exemple, la classe Employé a une méthode *verserSalaire* que n'a pas sa superclasse Personne : c'est dans ce sens qu'elle est une spécialisation de la première.

Superclasse : classe de laquelle hérite une autre classe, dit sous-classe.

Visibilité : la visibilité d'une donnée désigne les modes d'accès à cette donnée (voir Encapsulation).

ANNEXE

Diagramme d'objets :

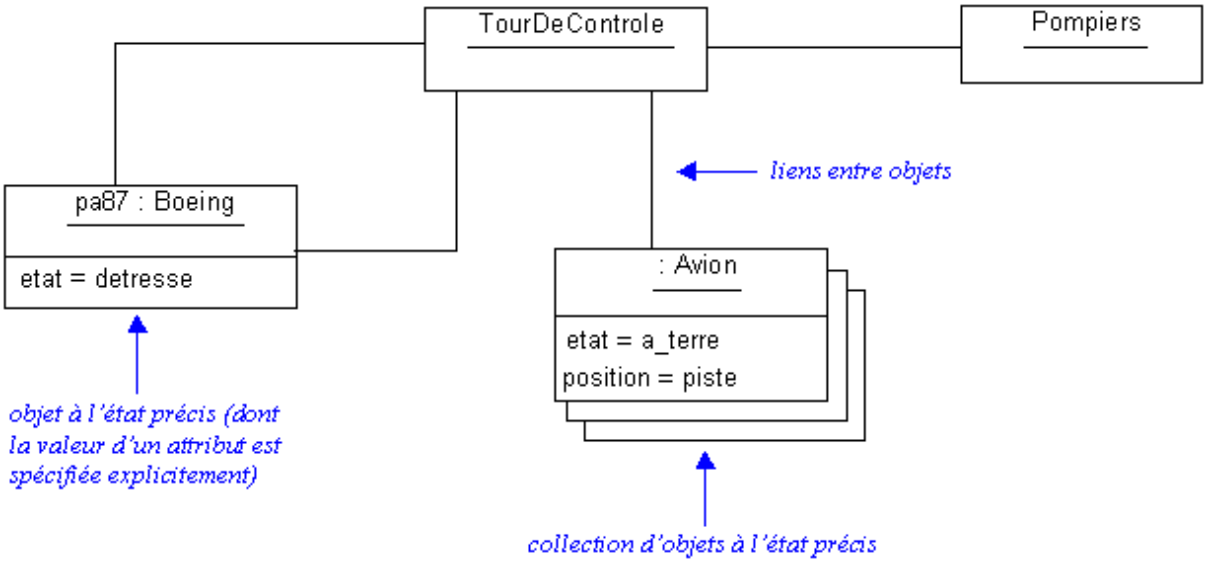


Diagramme de classe :

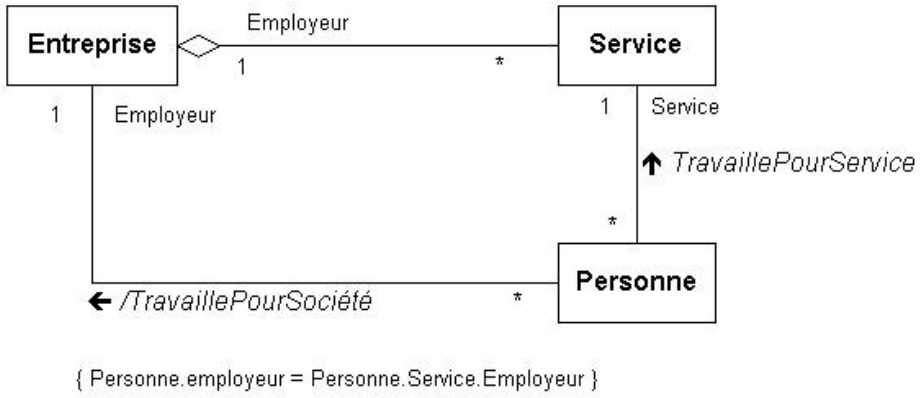
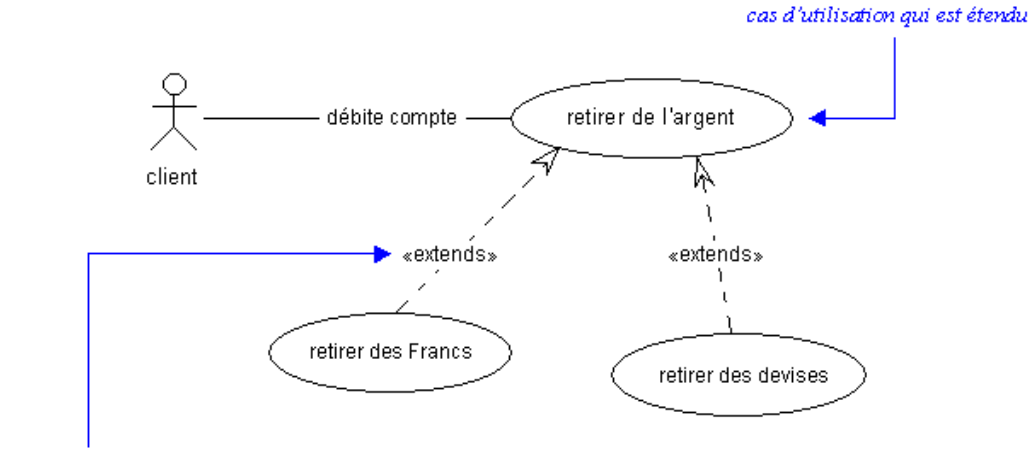


Diagramme de cas d'utilisation :



relation d'extension : *indique que le cas d'utilisation source étend (précise) les objectifs (le comportement) du cas d'utilisation destination.*

Diagramme de séquence :

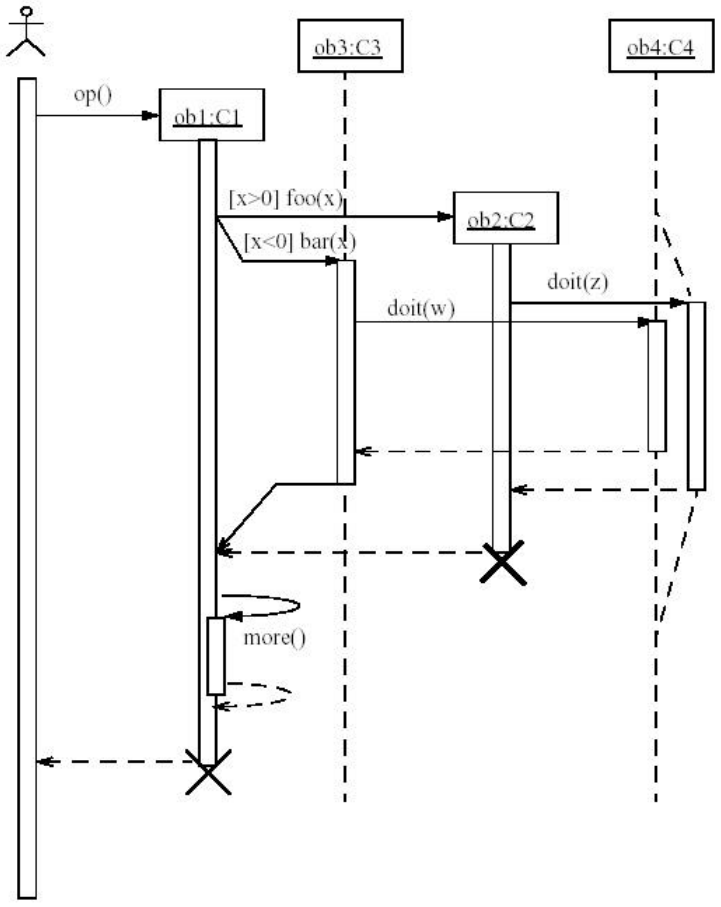


Diagramme de collaboration :

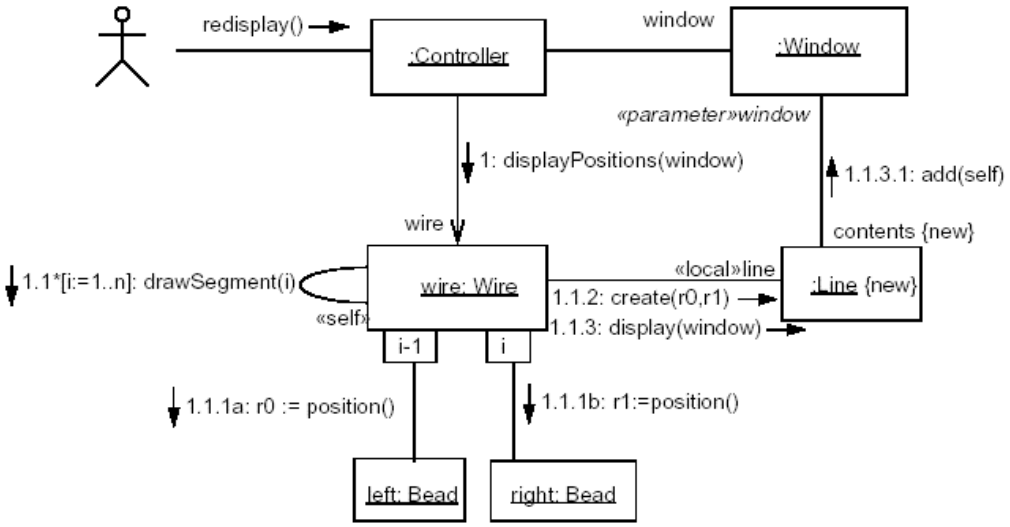


Diagramme d'activité : machine à café

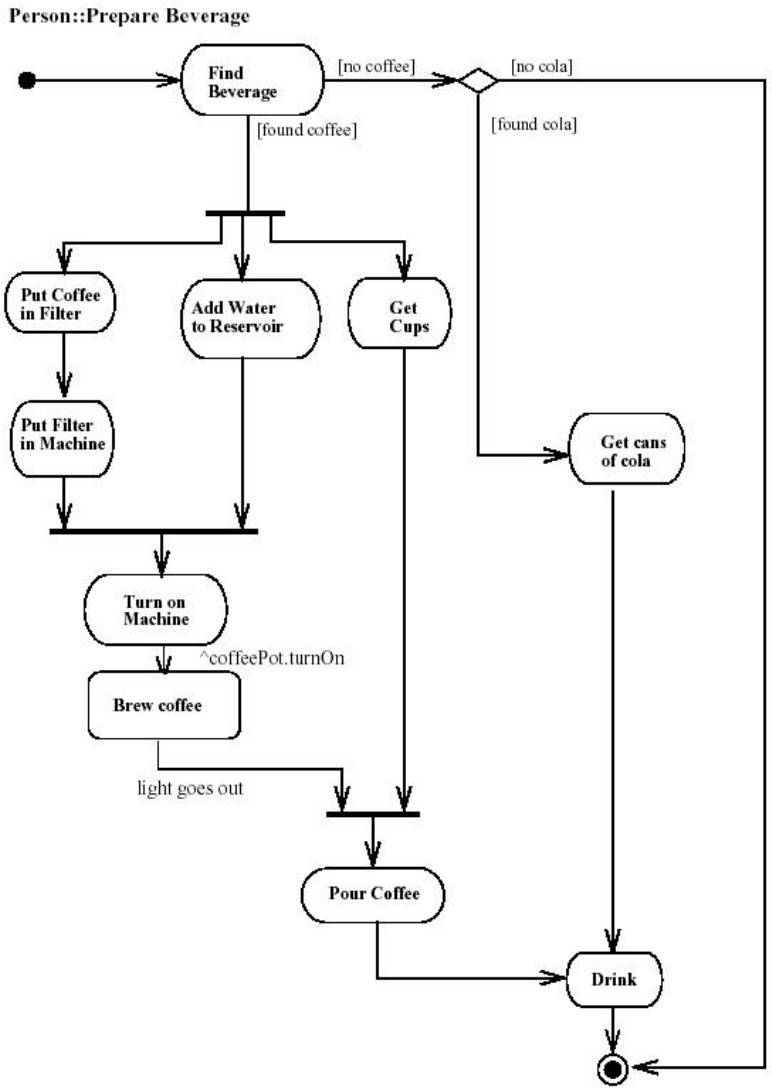


Diagramme d'états :

(grille-pain)

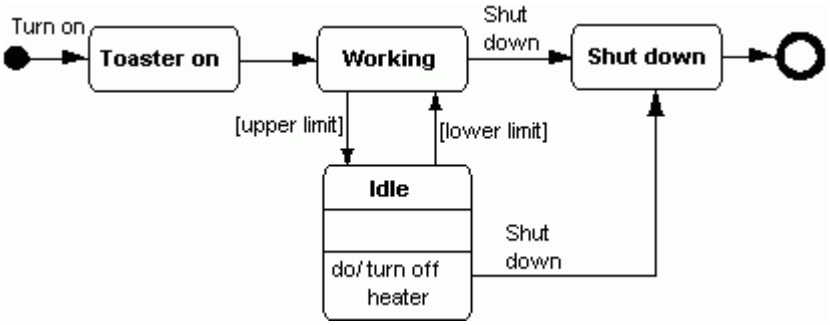


Diagramme de composants :

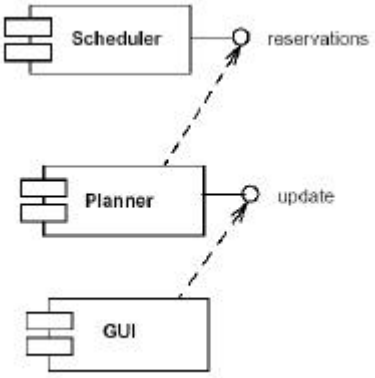


Diagramme de déploiement :