

# Modélisation et développement de l'application GRC

Pour modéliser l'application de GRC, nous avons décidé d'employer UML, langage devenu aujourd'hui le standard industriel de modélisation objet.

L'expression des besoins utilisateurs concernant l'application de GRC se trouve dans les différents documents issus de collectivités locales et disponibles au début de l'étude (CdC de Besançon, CdC de Lattes). D'autres documents, tels que le manuel utilisateur de CourierCPS, la solution de gestion de courrier développée par Nuxeo, sont utilisés pour mieux appréhender les différents aspects à mettre en oeuvre.

## Processus de modélisation

### Choix d'un processus

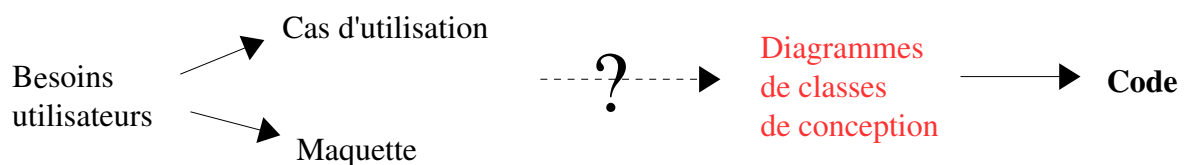
Comment passer des besoins utilisateur au code ? UML seul ne proposant aucune méthode particulière pour y parvenir, nous avons décidé d'adopter le processus de développement logiciel préconisé par Pascal Roques dans son livre « UML, modéliser un site de e-commerce », ouvrage de la série Les cahiers du programmeur, aux éditions Eyrolles. Ce processus, tel qu'il est proposé par l'auteur, se situe entre l'UP (Unified Process), un processus de développement très complet, et XP (eXtreme Programming), une approche minimaliste centrée sur le code. Il est notamment décrit comme un processus conduit par les cas d'utilisation, comme UP, et il est relativement léger, comme XP, mais sans négliger les phases d'analyse et de conception.

### Description

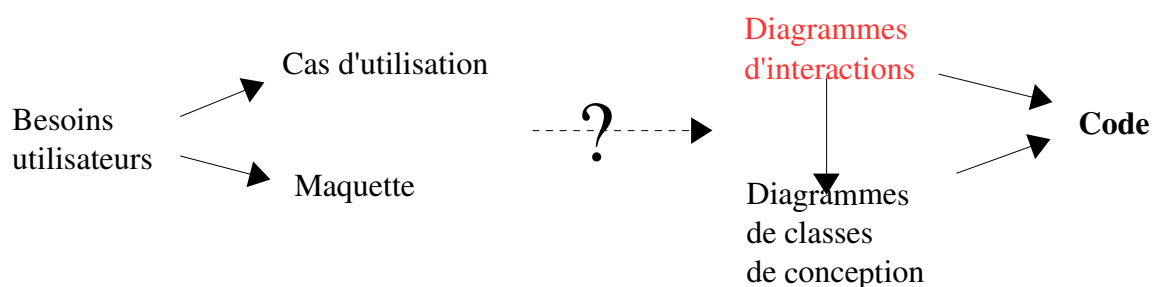
Il faut commencer par modéliser les besoins au moyen des cas d'utilisation, puis on les représente concrètement par une maquette.



En objet, la structure du code d'une application est définie par des classes logicielles, regroupées en packages. Il faut donc utiliser des diagrammes de classes de conception, montrant les données (attributs) que ces classes contiennent, les services (méthodes) qu'elles rendent et les relations qui existent entre elles.

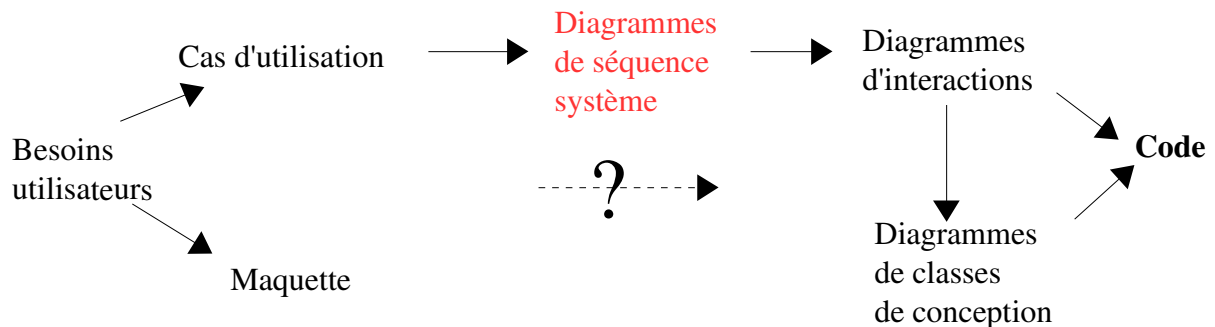


Il faut répartir correctement tout le fonctionnement du système entre les classes de conception. Pour chaque service ou fonction, il faut donc déterminer quelle est la classe qui va le/la contenir. Les diagrammes d'interaction aident à cette répartition.

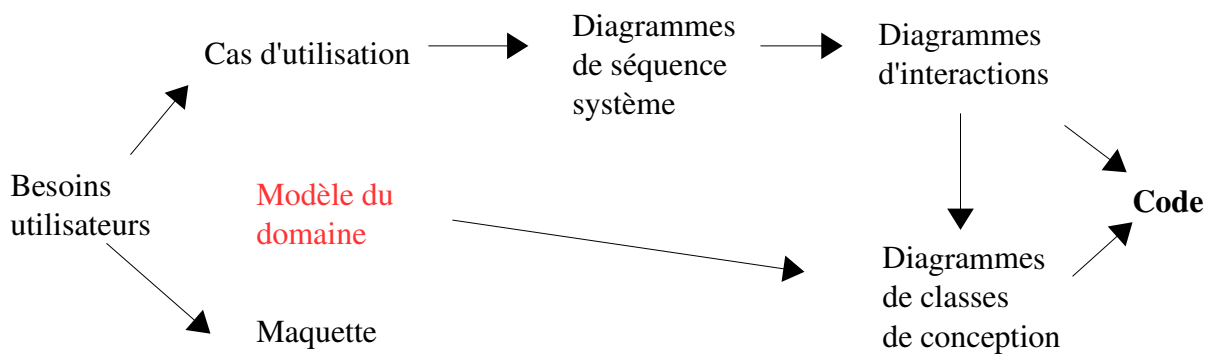


Reste à savoir comment passer des cas d'utilisation aux diagrammes d'interaction.

Chaque cas d'utilisation est décrit textuellement, mais peut aussi être représenté par un diagramme de séquence système, c'est-à-dire la séquence des interactions entre les acteurs et le système vu comme une boîte noire. Par la suite, cette boîte noire sera remplacée par un ensemble choisi d'objets de conception.

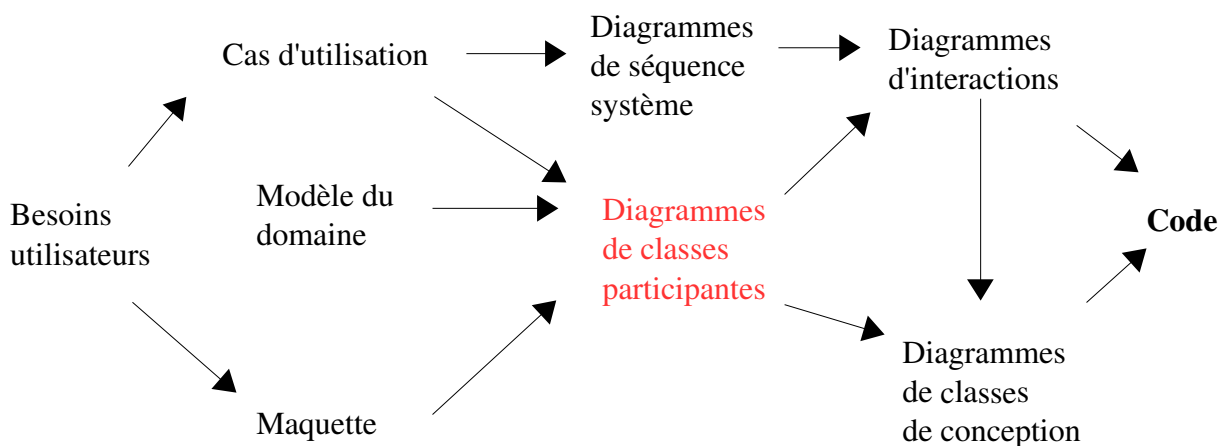


Comment trouver les classes de conception qui interviennent dans les diagrammes d'interaction ? Il faut construire un modèle du domaine, glossaire détaillé et formalisé en UML des concepts fondamentaux de l'espace du problème. Ce modèle fournit une partie des classes de conception, celles qui correspondent directement aux concepts métiers.



Toutefois, ce modèle ne permet pas d'identifier les principales classes d'IHM, ni celles qui décrivent la cinématique de l'application. Pour ce faire, il faut passer par les diagrammes des classes participantes. Ceux-ci décrivent, pour chaque cas d'utilisation, les 3 principales classes d'analyse et leurs relations. Celles-ci sont :

- les classes dialogue. Ce sont les écrans proposés à l'utilisateur (formulaires, résultats de recherche...). Elles sont issues de la maquette.
- les classes contrôle. Elles contiennent la cinématique de l'application. Elles font la transition entre les classes dialogue et les classes métier.
- les classes entité. Elles proviennent directement du modèle du domaine.



Les diagrammes de classes participantes sont très importants puisqu'ils font la jonction entre les cas d'utilisation, le modèle du domaine et la maquette d'une part, et les diagrammes de conception logicielle d'autre part (diagrammes d'interaction et de classes).

Pour finir, une exploitation détaillée de la maquette va permettre de réaliser les diagrammes d'activité de navigation. Ils mettent en jeu les classes participantes de type entité et contrôle et il représentent de manière formelle tous les chemins possibles entre les différents écrans utilisateur.

## **Etapas de modélisation**

### **Besoins utilisateur**

Ces besoins sont fournis à travers les différents cahiers des charges disponibles au début de l'étude : CdC de la ville de Besançon et de la ville de Lattes, document de synthèse (gestion courrier + gestion demandes d'interventions) rédigé par l'Adullact.

### **Spécification des exigences d'après les cas d'utilisation**

Il s'agit d'identifier les acteurs du système à concevoir, ainsi que les cas d'utilisation associés, à partir de l'expression des besoins.

#### ***Identification des acteurs***

Ce sont les entités externes au système (acteurs humains ou dispositifs matériels) qui interagissent directement. Un acteur peut consulter/modifier directement l'état du système.

#### ***Identification des cas d'utilisation***

Pour chaque acteur identifié, il s'agit de rechercher les différents cas dans lesquels cet acteur utilise le système et l'on trace le diagramme des cas d'utilisation correspondant.

Les cas d'utilisation peuvent être regroupés en ensembles fonctionnels cohérents appelés packages.

#### ***Relation entre les cas d'utilisation***

Il faut ensuite rechercher si les cas d'utilisation découverts sont liés entre eux par un des trois types de relations proposés par UML : l'inclusion, l'extension et la généralisation/spécialisation.

#### ***Classement par priorité***

Pour finir, il convient de classer les cas d'utilisation de façon hiérarchique à l'aide de deux facteurs, la priorité fonctionnelle et le risque technique. Cette méthode permet de planifier l'ordre (l'itération) dans lequel seront traités les différents cas d'utilisation.

Si la priorité est haute et le risque important, il faut traiter le cas en premier.

Si la priorité est basse et le risque faible, on peut traiter le cas à la fin.

Dans les deux autres situations, il convient de discuter de chaque cas pour choisir l'itération dans laquelle il sera traité.

En parallèle, ces cas d'utilisations sont représentés concrètement à l'aide d'une maquette d'interface homme-machine.

## **Analyse du domaine : découverte des objets métier**

### ***Identification des concepts du domaine***

Il s'agit de décomposer le domaine étudié en classes conceptuelles représentant les entités significatives de ce domaine, c'est-à-dire des objets du monde réel.

### ***Ajout des associations et des attributs***

En UML, le modèle du domaine est constitué de diagrammes de classes dans lesquelles aucune opération n'est définie. Ces diagrammes contiennent donc :

- des classes conceptuelles
- des associations entre ces classes
- les attributs des classes

### ***Généralisation***

Lorsque cela est possible, on améliore le modèle en identifiant et en extrayant les similitudes entre classes. Ceci fait

apparaître des super-classes dont héritent nos classes de conception.

### ***Structuration en packages***

Comme précédemment, il s'agit de regrouper les concepts découverts (ici, des classes conceptuelles) en ensembles cohérents, sous forme de packages. Ceci doit être fait en respectant deux principes fondamentaux : cohérence et indépendance. La cohérence s'entend au niveau sémantique. Il faut regrouper les classes de façon homogène. L'indépendance, quand à elle, consiste à minimiser les relations entre packages.

## **Spécification détaillée des exigences**

Il convient maintenant de faire une description détaillée des exigences, pour affiner l'expression des besoins, avant d'attaquer l'analyse et la conception objet.

### ***Description détaillée de chaque cas d'utilisation***

Chaque cas d'utilisation doit être décrit en détails de façon textuelle. UML ne proposant aucun modèle de description, P.Roques propose le suivant :

- description de tous les scénarios possibles , c'est-à-dire le scénario nominal ainsi que toutes ses extensions (conduisant aussi à un succès ou bien à un échec).
- description des préconditions (ce qui doit être vrai en amont du cas d'utilisation) et des postconditions (ce qui doit être vrai à la fin).
- exigences supplémentaires (non fonctionnelles, de performance...), se rapportant seulement au cas d'utilisation traité, et non à l'ensemble du système.

### ***Mise à jour des diagrammes de cas d'utilisation***

La description textuelle détaillée conduit parfois à modifier les diagrammes de cas d'utilisation.

### ***Diagrammes de séquence système***

Les acteurs interagissent avec le système par l'intermédiaire de messages. Les diagrammes de séquence système permettent de représenter graphiquement ces échanges. On définit ainsi pour chaque cas d'utilisation quels sont les traitements internes, appelés opérations système, déclenchés par chaque acteur.

### ***Liste des opérations système***

L'ensemble des opérations système ainsi découverte constitue l'interface publique du système, celui étant vu comme une boîte noire. En UML, il est alors représenté par une classe avec le décorateur <<system>> et contenant l'ensemble de ces opérations. Cette liste non exhaustive est complétée au fur et à mesure de l'étude.

## **Réalisation des cas d'utilisation : classes d'analyse**

Cette étape consiste à identifier les classes d'analyse qui vont participer à la réalisation de chaque cas d'utilisation. Ces classes seront représentées dans des diagrammes de classes participantes.

Les classes entité vont seulement posséder des attributs. Ceux-ci représentent généralement les infos persistantes de l'application.

Les classes contrôle vont seulement posséder des opérations. Celles-ci correspondent à la logique de l'application, aux règles métier.

Les classes dialogue vont posséder des attributs et des opérations. Les attributs vont représenter des champs de saisie. Les opérations représenteront les actions de l'utilisateur sur l'IHM.

Des associations seront rajoutées entre classes d'analyse, mais avec des règles strictes :

- les dialogues ne peuvent être reliés qu'à des contrôles ou d'autres dialogues. En général, les associations sont unidirectionnelles, de dialogue vers contrôle.
- les entités ne peuvent être reliées qu'aux contrôles ou à d'autres entités. Toujours unidirectionnelles et dans le sens contrôle vers entité.
- les contrôles ont accès à tout type de classe, y compris d'autres contrôles.

Les acteurs seront rajoutés à ces diagrammes, un acteur ne pouvant être relié qu'à un dialogue.

## **Modélisation de la navigation**

Dans cette étape, les diagrammes d'activité (un des diagrammes dynamiques d'UML) sont utilisés pour modéliser avec

précision la navigation dans l'application web.

### ***Diagrammes d'activité***

Ils contiennent des éléments standards (activités, transitions avec ou sans conditions, branchements, plusieurs fins...) mais aussi des conventions propres à la navigation dans un site web.

### ***Conventions spécifiques utilisées***

Les conventions graphiques suivantes seront utilisées :

- <<page>> pour une page complète du site.
- <<action>> pour une action simple (ex : classement d'une liste de courriers).
- <<exception>> pour une erreur ou un comportement inattendu du système.

Pour plus de lisibilité, il convient de structurer la modélisation de la navigation d'abord par acteur. Ensuite, si les cas d'utilisation pour un acteur sont indépendants, il est possible de produire un diagramme de navigation par cas. Dans le cas contraire, les cas d'utilisation sont liés les uns aux autres et les contraintes induites pourront être représentées de façon formelle dans un seul et même diagramme de navigation.

## **Conception objet préliminaire**

Cette étape consiste à attribuer des responsabilités précises de comportement aux classes d'analyse précédemment identifiées et de représenter ceci sous forme de diagrammes d'interaction (séquence et/ou collaboration). Ces diagrammes sont particulièrement utiles pour l'attribution des bonnes responsabilités aux bonnes classes, point délicat de la conception objet. On remplace le système boîte noire, tel qu'il a été décrit précédemment, par un ensemble d'objets en collaboration. On change de niveau d'abstraction.

### ***Diagrammes d'interaction***

Ce sont soit des diagrammes de séquence (chaque objet est sur sa ligne de vie), soit des diagrammes de collaboration (les interactions entre objets sont représentées sous forme de graphes).

### ***Classes de conception préliminaires***

On retourne au modèle du domaine et aux classes participantes, puis à l'aide des diagrammes d'interaction que l'on vient de réaliser, nous allons :

- ajouter ou préciser les opérations dans les classes.
- ajouter des types aux attributs, ainsi qu'aux paramètres et aux retours des opérations.
- affiner les associations entre classes.

## **Conception objet détaillée**

L'objectif de cette dernière étape est d'incorporer dans les diagrammes l'architecture web et les choix technologiques, ce qui va conduire à la modification des classes de conception préliminaire et à l'ajout de nouvelles classes plus techniques.

Dans le cas de l'application GRC, le choix technologique se portera sur la plate-forme J2EE et la couche de présentation sera réalisée à l'aide du framework Struts.

# Aspects techniques à étudier

## Choix d'une solution de workflow

Une petite recherche sur le web a permis de faire une liste des moteurs de workflow open source les plus connus. Cette liste doit permettre de faire le choix de quelques-uns, de les tester et de définir quel est le produit le plus adapté à nos besoins, c'est-à-dire suffisamment éprouvé sans être d'une mise en oeuvre complexe.

- Enhydra Shark (<http://shark.objectweb.org/>) - un moteur de workflow java/XML basé sur les spécifications du WFMC (WorkFlow Management Coalition) et accompagné de Enhydra jaWE, un éditeur graphique pour produire les définitions de processus au format XPDL.
- jBpm (<http://www.jbpm.org/>) - possède son propre langage de définition des processus, jPdl.
- WfMOpen (<http://wfmpopen.sourceforge.net/>) - une autre implémentation des spécifications WFMC. Les workflows sont décrits au format XPDL, plus quelques extensions.
- OFBiz Workflow Engine (<http://www.ofbiz.org/docs/workflow.html>) - implémente les spécifications WFMC et OMG. Utilise lui aussi XPDL.
- Bonita (<http://bonita.objectweb.org/>) - un système de workflow coopératif hébergé par ObjectWeb.
- OpenWFE (<http://openwfe.sourceforge.net/>) est un suite de gestion de processus complète. Est fourni sous licence de type BSD.

## Gestion des droits

L'API JAAS (Java Authentication and Authorization Service), fournie à partir de la version 1.4 de la plate-forme J2EE semble être la solution adaptée à nos besoins. Elle permet de se concentrer uniquement sur les aspects fonctionnels de l'application, puis d'appliquer à posteriori une gestion des droits. Il est toutefois nécessaire d'étudier son principe de fonctionnement et d'évaluer la facilité de sa mise en oeuvre.

## Lien avec les clients de messagerie

(à rédiger)

## Choix d'une solution de mapping

Traditionnellement, les développeurs java codent directement l'accès aux données en utilisant des requêtes SQL à travers JDBC. Aujourd'hui, les problèmes de cohabitation entre les mondes objets et relationnels, rencontrés lors du développement d'une application utilisant un langage objet et une base de données relationnelle, sont résolus grâce aux outils de mapping objet-relationnel.

Il faut donc s'intéresser à l'offre open source disponible dans ce domaine pour pouvoir choisir l'outil qui convienne le mieux au projet GRC. Plusieurs solutions sont à explorer.

L'API JDO (Java Data Objects) de Sun (= la JSR12 du Java Community Process) connaît plusieurs implémentations open source, parmi lesquelles :

- OJB (ObjectRelationalBridge), l'outil de mapping développé par le Apache DB Project (<http://db.apache.org/ojb/>)
- Speedo, solution hébergée par ObjectWeb (<http://speedo.objectweb.org/>)
- TJDO (TriActive JDO) (<http://tjdo.sourceforge.net/>)
- XORM (eXtensible Object Relational Mapping) (<http://xorm.sourceforge.net/>)
- JPOX (Java Persistent Objects) (<http://www.jpox.org/>)

Il existe aussi un plugin Eclipse permettant d'intégrer au sein de l'environnement de développement des solutions comme TJDO ou JPOX, mais dont l'objectif est de prendre en charge toutes les implémentations JDO.

Parmi les outils de mapping n'implémentant pas JDO, les plus cités sont :

- Hibernate  
(<http://www.hibernate.org/>)
- Castor, du Exolab Group  
(<http://castor.exolab.org/>)

## **Lien avec les outils bureautiques**

(à rédiger)

## Environnement de développement proposé

Le langage de développement choisi est Java, principalement dû au fait c'est le seul langage qui propose plusieurs solutions de workflow en open source.

Pour le développement et les tests de l'appli GRC, nous utiliseront Tomcat 5, le moteur de servlets /JSP du Jakarta Apache Project.

L'environnement de développement sera constitué à partir de la version 2.1.3 d'Eclipse a laquelle seront ajoutés les plugins suivants :

- EclipseUML 1.2.1 Free de la société Omondo (<http://www.omondo.com>) pour l'analyse UML et la production automatique du corps des classes java.
- Tomcat launcher V3 de la société Sysdeo (<http://www.sysdeo.com>) permettant notamment de démarrer ou arrêter Tomcat depuis Eclipse ou exporter le projet dans un fichier .war
- EasyStruts, plugin créé par Emmanuel Boudrant et disponible sur SourceForge (<http://easyststruts.sourceforge.net/>) et qui fournit un ensemble d'outils facilitant le développement d'un projet web basé sur Struts, le framework MVC2 du Jakarta Apache Project.
- DbEdit 1.0.1, plugin créé par Uwe Voigt ([http://www.geocities.com/uwe\\_ewald/dbedit.html](http://www.geocities.com/uwe_ewald/dbedit.html)) et qui propose des outils de gestion de base de données (lecture et édition de tables).
- ...editeur xml/jsp

Attention, des problèmes de mises en place et d'utilisation des plugins pré-cités ont été rencontrés avec la version 3.0 d'Eclipse. C'est pourquoi nous sommes revenus à la version stable précédente.