ANALYSE ORIENTE OBJET DE PROCESSUS SIDERURGIQUES DE TYPE COKIER

Aymerick Donnay, François Fouss, Manuel Kolp, David Massart, Alain Pirotte
IAG –Institut d'Administration et de Gestion, ISYS- Unité de systèmes d'information, Université Catholique de Louvain, 1 Place des Doyens, Belgique
Email: {fouss, kolp, massart, pirotte}@isys.ucl.ac.be

Résumé: Ce document constitue la première phase de l'étude orienté objet de la cokerie de Carsid au cours de laquelle nous avons modélisé le fonctionnement de la cokerie. Pour ce faire, nous avons principalement utilisé le langage de modélisation UML. Les différentes phases du développement en UML peuvent être représentées au moyen d'une série de diagrammes permettant de comprendre de manière visuelle les concepts définis. Tous les modèles s'enchaînent en passant de l'analyse à la conception, gagnant en complexité, dans un langage commun et unique, s'affinant au fur et à mesure pour arriver à l'élaboration finale du modèle. Les diagrammes permettront de comprendre sous différents angles la globalité du cas étudié en présentant une vue statique et dynamique de celui-ci. Chaque diagramme exprimera une partie de la structure totale, tout en étant un aspect particulier du modèle.

1 INTRODUCTION

L'amélioration continue des processus utilisés par une entreprise est une nécessité dans un milieu compétitif. Dans cette optique, l'information est une donnée sensible qu'une organisation se doit de maîtriser. Sa collecte, son traitement et sa diffusion doivent permettre de la valoriser.

Carsid (Carolo-Sidérurgie) est une joint venture récente issue du mouvement de concentration. Elle a été crée par les sociétés Duferco et Usinor Belgium S.A (composition du capital de 60/40).Duferco est une entreprise italienne qui est notamment active en Belgique (la Louvière et Clabecq). Usinor fait partie du géant mondial de l'acier Arcelor, en partenariat avec le luxembourgeois Arbed et l'espagnol Acelaria. Duferco possède une part majoritaire dans la société Carsid.
Carsid souhaite revoir la conception, la réalisation et l'exploitation des bases de données de production et de leurs applications informatiques dans plusieurs de ses usines : Cokerie, Agglomération, Haut Fourneau 4 et Énergie
Les procédés industriels de l'entreprise sont longs, complexes, et riches en informations et données potentiellement exploitables. Des milliers de données sur la production venant d'un grand nombre de capteurs implantés dans l'installation sidérurgique sont collectées et parviennent chaque minute aux systèmes informatiques. Les données brutes sont consolidées toutes les heures en données horaires, et ces dernières, une fois par jour, en données journalières. Les données sont périodiquement purgées des données de plus fine granularité.

L'exploitation de ces données sert essentiellement à produire :
- des synoptiques de production (contrôle graphique en temps réel de l'installation par les opérateurs)
- des rapports (suivi, historique, comptabilité de la production)
- des données pour les outils d'études (extraction EXCEL, statistiques)

Les principes de base de la représentation et de l'exploitation des données dans les différents services et usines sont suffisamment semblables pour qu'on puisse envisager le développement type d'une base de données générique qui serait portable (paramétrable) dans les quatre usines. De plus, chaque usine utilise deux bases de données : une base de données « temps réel » et une base de données destinée à la gestion informationnelle. Il est envisageable de les regrouper en une seule base par usine, vu que les contraintes temps réel sont relativement souples (processus de chargement de données relativement lents, une fois par minute).

Le présent document présente une analyse orientée objet impliquant une partie conception de bases de données, selon le standard UML, de la cokerie de Carsid. Il s'inspire du standard de l'IEEE 830/1993 traitant de la rédaction de cahiers des charges pour d'analyse informatique.
L'objectif de la présente étude fait partie d'un effort de Carsid de modernisation des systèmes et de standardisation des méthodologies utilisées. Plusieurs phases sont nécessaires afin de répondre aux objectifs :
Dans un premier temps, l'étude porte sur la proposition d'une méthodologie et d'outils informatiques. Elle consiste aussi à analyser la cokerie au moyen des outils proposés. Cette analyse permet de décrire le processus général de production de coke selon une modélisation UML - *Unified Modeling Language*.

A la fin de cette première phase, des solutions seront présentées à titre d'exemple. Elles permettront de se rendre compte des possibilités de développement de la seconde phase. Celle-ci visera à rassembler de manière effective par rétro ingénierie les différentes bases de données existantes pour produire une base de données intégrée. L'extension aux autres usines (agglo, énergie et haut fourneau) pourra ensuite avoir lieu pour les restructurer et les intégrer.
Les travaux réalisés constituent la première phase de l'étude : la modélisation orienté objet de la cokerie. Ils incluent les parties suivantes :
- Une description théorique de la méthodologie de modélisation en UML et i* de même qu'un glossaire des concepts de modélisation.
- La modélisation orienté objet et de bases de données de la cokerie (excepté le traitement des produits dérivés) au moyen de UML et de i*.
- La synthèse des bases de données existantes chez Carsid, qui devrait permettre une première comparaison entre le modèle développé et la réalité des prises et traitements de données.
- La rétro-ingénierie (reverse engineering) en un modèle objet UML des bases de données existantes de Carsid centrées sur la cokerie à savoir les bases de données **Analyse**, **Energie**, **Cokerie**, **Bascule** et **Traction**.

2 ELEMENTS DE MODELISATION

La modélisation est une des tâches les plus importantes dans le processus de développement d'un système. La phase consacrée à l'analyse peut être considérée comme plus stratégique que celles dévolues à la conception et l'implémentation proprement dites. Il faut en effet fondamentalement représenter, comprendre et identifier les exigences du système afin de concevoir puis d'implémenter ensuite une application stable et performante.
Avec l'augmentation de la complexité des systèmes à élaborer, le choix d'une méthode de développement appropriée se révèle primordial pour le succès des travaux. Pour cela, il existe plusieurs orientations disponibles, c'est-à-dire des approches différentes pour comprendre, représenter, analyser et concevoir un système. Le problème sera décomposé en plusieurs modèles différents liés entre eux.

Orienté objet. La modélisation orienté objet, plus proche de la réalité que les approches précédentes, s'est développée considérablement ces dernières années aux dépends notamment de l'approche structurelle et fonctionnelle. La décomposition du problème à traiter est perçue de manière différente: dans cette nouvelle approche, il sera nécessaire non pas de modéliser par fonction mais par objets. Un objet est « une abstraction d'une chose du domaine du problème ayant des propriétés et des comportements ».
Les objets modélisés durant l'analyse vont représenter le monde réel et pourront être identifiés selon trois caractéristiques : une identité, un état au cours du temps et un comportement représentant les capacités de l'objet. Chaque objet créé sera en conséquence unique.
L'orientation objet apporte toute une série d'atouts à une modélisation adéquate. Les solutions obtenues sont en effet indépendantes des modifications physiques de l'architecture et donc plus facilement adaptables aux évolutions du système. Il sera possible d'étendre les possibilités de traitement des logiciels en rajoutant aux schémas existants de nouveaux objets mettant à jour la fonctionnalité globale des modèles.
L'uniformisation de la méthode de travail peut être d'un grand intérêt pour le développement d'un système. De nombreuses tentatives de normalisation ont eu lieu ces quinze dernières années. Finalement un langage de modélisation unifié, UML, est apparu. Il permet de bénéficier des avantages de l'orientation objet tout en posant un cadre méthodologique standard à cette approche.

UML, langage de modélisation. UML est un langage d'analyse et de conception orienté objet se basant sur la création itérative de modèles successifs de plus en plus affinés afin de mettre en place une solution au problème étudié.
La méthodologie adoptée a pour objectif de se rendre indépendant des phases techniques comme par exemple le langage de programmation. Ceci

permettra de s'affranchir des contraintes liées à la présence de différentes technologies tout en permettant de poursuivre le développement.
En uniformisant les méthodes connues auparavant, UML peut aussi être considéré comme un méta modèle, c'est à dire qu'il crée des normes pour les modèles objets, en décrivant les concepts et en précisant la sémantique des objets.
Cette formalisation et les avantages qu'elle a procurés ont conduit à l'adoption d'UML par l'*Object Management Group,* qui a pour objectif de rassembler les différentes évolutions dans le monde de l'orienté objet. Ceci permet d'assurer que le développement de l'industrie objet se fera dans la même direction, au bénéfice de tous, du fait de son interchangeabilité et interopérabilité.

La méthodologie proposée sépare en plusieurs parties distinctes le travail des données de départ.

- Tout d'abord, la phase d'analyse permet d'appréhender la nature du problème et d'aborder l'aspect « compréhension » de celui-ci. De manière simplifiée, l'attention sera dirigée vers le « quoi » du projet plutôt que le « comment ». Nous verrons aussi, notamment avec les diagrammes i*, comment se préoccuper du « pourquoi » durant l'analyse. L'analyse passe d'abord par une phase *d'analyse des besoins*, permettant de comprendre le problème en étudiant l'état de l'organisation. Ensuite, elle consistera en une *modélisation conceptuelle* qui s'attachera à décrire le système à modéliser. Une recherche et une définition des objets et des concepts du cas étudié au moyen de l'arsenal des modèles disponibles mettra en exergue la structure du problème.

- La phase suivante est le « Design » (conception), c'est à dire l'élaboration d'une solution logique au problème. L'objectif sera de mettre au point des éléments permettant le développement d'une solution transposable via la phase d'implémentation en programme informatique. En d'autres mots, l'accent sera mis sur l'étude du « comment » du projet. Cette phase se décompose en deux sous-phases : la conception architecturale qui se focalise sur l'identification de l'architecture globale du système et la conception détaillée qui fixe tous les détails juste avant l'implémentation. En terme de conception de bases de données, on parlera plus de conception logique (le schéma de la base de données suivant une technologie, par exemple le modèle relationnel mais indépendant du SGBD cible) et de conception physique (le schéma logique mais adapté aux spécification su SGBD par exemple SQL server 7).

- La dernière phase est l'implémentation. Un langage de programmation ou un système de gestion de base de données se basant sur les travaux effectués dans les deux étapes précédentes est utilisé afin de construire l'application proprement dite.
Une ultime phase de test permet de contrôler les résultats obtenus et de mettre en exergue les éventuelles faiblesses de la solution obtenue et de les corriger lors des développements successifs.
Il est important de noter que les différentes phases du développement sont fortement liées. En effet, il est possible d'observer une continuité d'une phase à l'autre et une influence mutuelle car une modification d'un aspect du système analysé influencera la constitution des trois phases.

Les différentes phases du développement en UML peuvent être représentées au moyen d'une série de diagrammes permettant de comprendre de manière visuelle les concepts définis. Tous les modèles s'enchaînent en passant de l'analyse à la conception, gagnant en complexité, dans un langage commun et unique, s'affinant au fur et à mesure pour arriver à l'élaboration finale du modèle.
Les diagrammes permettront de comprendre sous différents angles la globalité du cas étudié en présentant une vue statique et dynamique de celui-ci. Chaque diagramme exprimera une partie de la structure totale, tout en étant un aspect particulier du modèle.
L'enchaînement de modèles permettra non seulement de comprendre l'évolution du modèle mais facilitera aussi les réorganisations du processus, qu'elles soient des changements conceptuels ou des évolutions.

Rational Rose. Rational Rose Entreprise Edition (figure 1) est l'outil phare de la société Rational. Cette entreprise est leader sur de nombreux marchés de produits basés sur UML. Les concepteurs de ce langage (Booch, Jacobson et Rumbaugh) travaillent d'ailleurs pour cette entreprise. L'outil rational offre un support large et complet des modèles récents de UML. La version proposée est Entreprise Edition 2002.
L'intérêt de cet outil se situe aussi sur le partage du modèle entre plusieurs personnes d'une équipe de travail. Les avancées peuvent se réaliser de manière séparée sans influencer les parties du projet des autres membres de l'équipe. Les changements sont rendus disponibles après travaux afin que les personnes concernées puissent en prendre connaissance.
L'outil rational rose permet la génération de gabarits de code dans plusieurs langages, comme par exemple C++, Java ou Visual basic. Les générations SQL pour les SGBD objet relationnel sont aussi

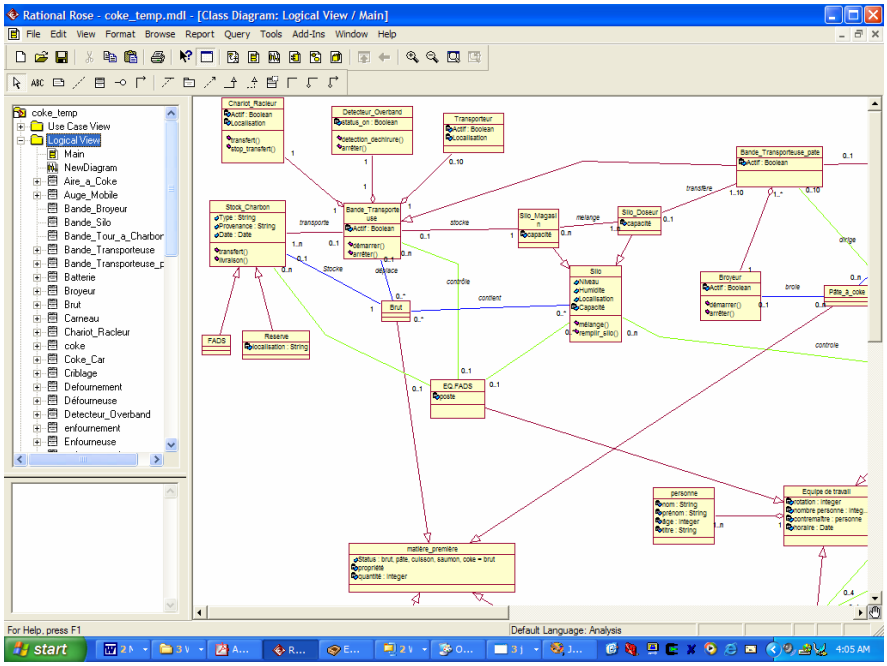

Figure 1: Rational Rose Entreprise Edition

supportées, notamment pour Oracle, Sybase, SQL Server et IBM DB2. L'outil supporte aussi l'ingénierie inverse. Ceci permet la récupération des schémas relationnels en diagramme de classe UML sous rational. Les processus itératifs sont aussi développables au moyen de Rational Rose Entreprise Edition, car il supporte les changements de structure dynamique et il permet de modifier l'implémentation en générant de nouveaux codes pour la conception.

Together. Cet outil est similaire à *Rational Rose* et est développé par *TogetherSoft*. Il est cependant plus complexe et moins facile d'accès. La documentation n'est pas aussi abondante que pour le produit *Rational*, ce qui peut poser de nombreux problème et faire perdre pas mal de temps. *Together* est plus orienté *software engineering* et *design*. Le produit propose des modèles supplémentaires à UML tels que les *business* process, *robustness* ou *EJB*, *design patterns* (J2EE, GoF, Gamma), ... Il supporte l'importation de modèles développés en *Rational Rose* sans toutefois être entièrement compatible. Cet outil est complètement développé en Java et est donc utilisable sur n'importe quelle plateforme sur laquelle une machine virtuelle java peut être installée.

Power designer. Ce troisième outil est considéré comme un des plus intéressants présents sur le marché de la conception de bases de données. Il a été développé par la société *Sybase*. Ses fonctionnalités sont nombreuses. L'interface est agréable et les commandes sont plus faciles d'accès que pour *Rational Rose*. Il est possible de faciliter les liens entre les différents diagrammes UML en incluant des références sur les liens existants. Des fichiers *Rational Rose* peuvent être utilisés directement dans *Power designer*, bien que leur conversion ne soit pas totale. Il est par exemple possible de perdre certaines associations lors du passage à *Power designer*.
De nombreux langages sont disponibles, ce qui rend le software attractif. Il est important de noter que tous ne sont pas certifiés. C'est par exemple le cas pour les codes générés de Visual Basic, qui ne sont en conséquence pas de qualité parfaite. Cependant, l'outil dispose des mêmes qualités que les deux précédents.
Les langages certifiés sont C++, Cobra, Java, Power Builder et XML. Les différents modèles ER, DFD et outils de *reporting* sont inclus dans ce produit.

Concept de représentation i*. *i** est un cadre conceptuel qui propose des notions telles que *acteur, agent, rôle, position, goal, softgoal, tâche, ressource, croyance* ainsi que différents types de *dépendance* sociale entre les acteurs. Il s'agit d'un graphe où chaque nœud représente un *acteur* et chaque lien entre deux acteurs indique qu'un acteur

dépend de l'autre pour la réalisation d'un certain objectif. Une dépendance décrit un accord (appelé *dependum*) entre deux acteurs : le *depender* et le *dependee*. Le depender est l'acteur dépendant et le dependee, l'acteur dont le premier dépend. Le type de la dépendance décrit la nature de l'accord. Les dépendances de *goal* représentent une délégation de responsabilité pour la réalisation d'un goal; des dépendances de *softgoal* sont similaires aux dépendances de goal mais leur réalisation ne peut pas être définie précisément (par exemple, l'appréciation est subjective ou la réalisation n'est obtenue que dans une certaine mesure); des dépendances de *tâche* sont utilisées quand le *dependee* est requis pour accomplir une certaine activité; et des dépendances de *ressource* demandent au *dependee* de fournir une ressource au *depender*.

3 DESCRIPTION DU PROCESSUS DE COKÉFACTION

Le travail de préparation du coke à partir d'un charbon adéquat nécessite un long et lourd processus de transformations successives du charbon (figure 2). Il s'agit de produire un coke de bonne qualité tout en traitant tous les co-produits générés aux cours des différentes étapes. La cokerie est divisée en trois secteurs.

Secteur préparation. La réception et le traitement primaire du charbon cokéfiable constituent le début de la chaîne dans les installations de cokéfaction.
Les réceptions de charbon transitent par les ports d'Anvers et de Rotterdam en provenance des États-Unis, de Pologne, d'Afrique du sud et d'Australie. Les caractéristiques propres des charbons diffèrent selon leurs origines. Le transfert depuis les ports a lieu selon trois moyens de transport : par route, par eau et par voie ferroviaire, cette dernière manière étant la plus régulièrement utilisée à la cokerie de Carsid.
Les charbons qui arrivent sont stockés soit dans une fosse soit dans une réserve qui sert de stock de sécurité afin de protéger l'entreprise des aléas éventuels des fournisseurs ou des livreurs. La production réalisable au moyen de ce stock est d'environ 15 jours au maximum. Le stockage prolongé pourrait en effet nuire à la qualité du charbon utilisé, ce qui se répercuterait inévitablement sur la qualité du coke produit.
Une série d'instruments de transports (chariots racleurs, bandes transporteuses,…) sont utilisés pour le transport de charbon depuis la réserve vers des silos de stockage. Ce transport est surveillé par une machine qui détecte et capte les métaux qui seraient éventuellement présents afin d'éviter les déchirures de la bande. Les silos sont divisés en deux catégories selon qu'ils soient silos de réserve ou doseurs. Les silos doseurs ont une capacité de 250 tonnes et sont au nombre de 10. Ils permettent de mélanger les charbons de différentes origines afin d'obtenir les caractéristiques chimiques désirées. Ce mélange se fait en dessous des silos : les quantités de charbon désirées sont déversées depuis chaque silo via 10 chariots racleurs sur une bande de transport unique qui conduit le mélange vers le broyage.
Les silos de réserve ont une capacité dix fois plus grande que les silos doseurs et permettent de remplir ces derniers en cas de besoin.
Le premier traitement agissant sur la nature physique du charbon débute lors du broyage du lot transporté à la sortie des silos doseurs. Après un passage par une série de bandes transporteuses, un broyeur va agir sur les différents charbons présents dans le mélange afin d'obtenir une granulométrie homogène. Celle-ci diffère en effet en fonction de l'origine des charbons. Un détecteur de métaux identique à celui du transport de charbon brut contrôle les opérations afin de protéger la machinerie. A l'issue du broyage, le produit obtenu a pour dénomination « pâte à coke ». Elle est transférée au moyen d'une série de transporteurs de la tour de broyage vers la tour à charbon. Cette tour est située au dessus des fours et possède une capacité de 3.600 tonnes, soit les besoins équivalent à une journée de production.

Secteur fours. Avant de débuter l'opération de distillation de la pâte à coke, il faut transférer la pâte dans les fours. L'équipe chargée des fours transfère à cet effet une quantité de pâte dans une machine appelée enfourneuse. Cette machine, située au dessus des batteries, remplit ensuite les fours par des ouvertures situées au dessus de ceux-ci, les bouches d'enfournement. Ces ouvertures se présentent sous la forme de bouchons et au nombre de quatre par four. Durant le remplissage, une autre machine appelée défourneuse s'occupe elle d'égaliser la charge de pâte tout le long du four afin d'assurer une meilleure répartition de la charge et homogénéiser la cuisson. Les remplissages, ou enfournements, suivent un planning précis qui tend à maximiser le nombre d'enfournements journaliers.
Le remplissage fini, les ouvertures supérieures et latérale sont fermées et le processus de cuisson de la pâte débute. L'opération dure de 16 à 19 heures, selon la batterie dans laquelle se situe le four. Cependant, il est difficile d'évaluer les durées exactes de cuisson à cause de nombreuses

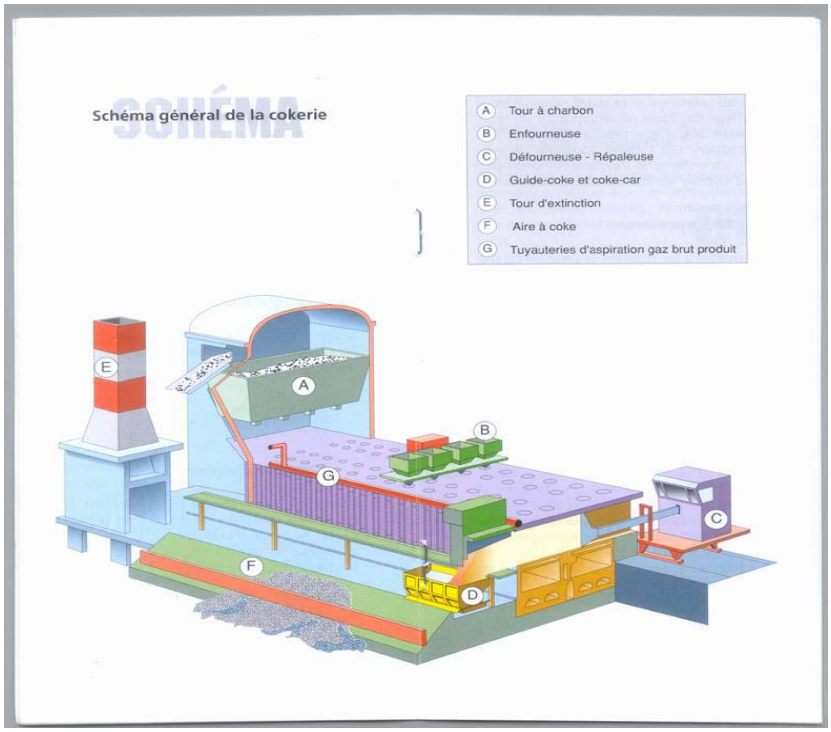

Figure 2: Schéma général de la cokerie: enfournement, défournement et refroidissement

incertitudes liées aux fours. Les enfournements sont effectués selon un pas de cinq, ce qui signifie que la procédure est d'enfourner le four (n+5) après avoir enfourné le four (n). Cette méthode permet d'éviter de trop grandes variations de température dans les fours.

Les fours sont rassemblés par série dans des ensembles appelés batteries. Celles-ci sont au nombre de quatre et composées de 20 à 50 fours, ce qui donne un ensemble total de 122 fours pour la cokerie de Carsid. Les propriétés des fours au niveau des temps de cuisson théoriques et des quantités enfournables sont différentes selon la batterie à laquelle ils appartiennent. Les quatre batteries sont alignées en ligne avec la tour à charbon située au milieu. Cette disposition permet une circulation plus facile de la machinerie autour des fours.

La composition en batterie trouve son origine dans une volonté de réduire les pertes calorifiques dans chaque four. En effet, l'espace entre les fours, appelé piedroit, est composé d'une série de cheminées verticales par lesquelles passent les gaz permettant la cuisson dans les fours. Ces cheminées, appelées des carneaux, contribuent donc à la cuisson de deux fours contigus. L'échange de calories se fait par conductivité sans contact direct avec la pâte à coke. Les gaz utilisés sont soit issus de la cokerie soit du haut fourneau, ce qui permet de rendre la cokerie indépendante au niveau énergétique. Les fumées de combustion sont récupérées en passant dans les empilages des briques réfractaires lors d'un premier cycle. Lors d'un second cycle, les gaz de combustion les parcourent afin de récupérer leur chaleur. Le passage d'un cycle à l'autre est dénommé « inversion ». Ce passage a lieu toutes les demi heures.

Le four atteint une température de plus de 1200°C durant la cuisson, ce qui permet la transformation de la pâte en saumon de coke, coke à très haute température possédant déjà les caractéristiques finales requises. Les gaz issus de la distillation sont récupérés en vue de leur valorisation.

Secteur refroidissement. A la fin de la cuisson, l'opération de défournement, c'est à dire de vidage du four, débute. Pour l'effectuer, le four est ouvert par ses portes latérales. D'un côté, la défourneuse expulse la charge du four en la poussant. De l'autre, le guide coke, un couloir muni d'une auge mobile, réceptionne la charge et la transfère jusqu'à un

wagon, le coke car. L'opération de refroidissement débute. Le coke car est conduit vers la tour d'extinction. Une quantité d'eau de 25m³ y est déversée. Elle doit refroidir et éteindre le saumon de coke. Une grande partie de cette eau s'évapore directement au contact du saumon. Les eaux résiduelles sont traitées dans un bassin de décantation afin de les réutiliser ultérieurement. Le coke est éventuellement arrosé manuellement dans le cas où il ne serait que partiellement éteint.
L'étape suivante est la mise en attente du coke, qui transite par un quai d'étalement afin de perdre son humidité. Après une période d'attente d'environ trente minutes, le coke car transférera le coke obtenu vers la tour à gros coke. Là, l'opération de criblage débute. Il s'agira de séparer les fragments que l'on considère comme « coke sidérurgique », c'est-à-dire propres à être utilisés dans l'élaboration de l'acier, des autres, valorisables à d'autres usages. La base de cette sélection se fait sur la taille des morceaux, qui doit être suffisante pour garder une bonne perméabilité de la charge. Les fragments considérés comme sidérurgiques sont expédiés vers le haut fourneau tandis que le « petit coke » est dirigé vers l'agglomération.
Certaines tâches de réparation doivent être effectuées sur les fours. Les équipes de maçons ont la responsabilité de réparer les fissures et autres problèmes pouvant survenir au niveau des fours. Ceux-ci s'usent et peuvent connaître des problèmes au niveau de leurs briques de revêtement. La réparation d'un four se fait à la température de 700°C. Il est impossible de descendre en dessous de ces températures car les briques changeraient d'état et deviendraient cassantes comme du verre. Les opérations de réparation sont donc des travaux lourds et difficiles, compte tenu des conditions de température présentes.

Il existe finalement une équipe de mesure et de réglage qui a pour but de veiller au bon fonctionnement de l'appareil de production de la cokerie. Elle contrôle la température des fours en mesurant la chaleur dans les carneaux des fours et peut donc réguler cette dernière. Elle est aussi responsable de l'inversion du cycle de chauffe de la batterie, qui a lieu toutes les demi heures environ.

4 APPLICATION D'UML A LA COKERIE DE CARSID

4.1 Analyse des besoins

Diagramme des cas d'utilisation (Figure 3). Les cas d'utilisation modélisés identifient les fonctionnalités nécessaires au processus qui conduit à la production du coke. Ces cas d'utilisation ont été répertoriés en plusieurs groupes dérivés de l'organisation de la production du coke à la cokerie et liés aux acteurs qui y sont associés : secteur préparation (réception du charbon, réception FADS, réception réserve, transfert charbon silo,

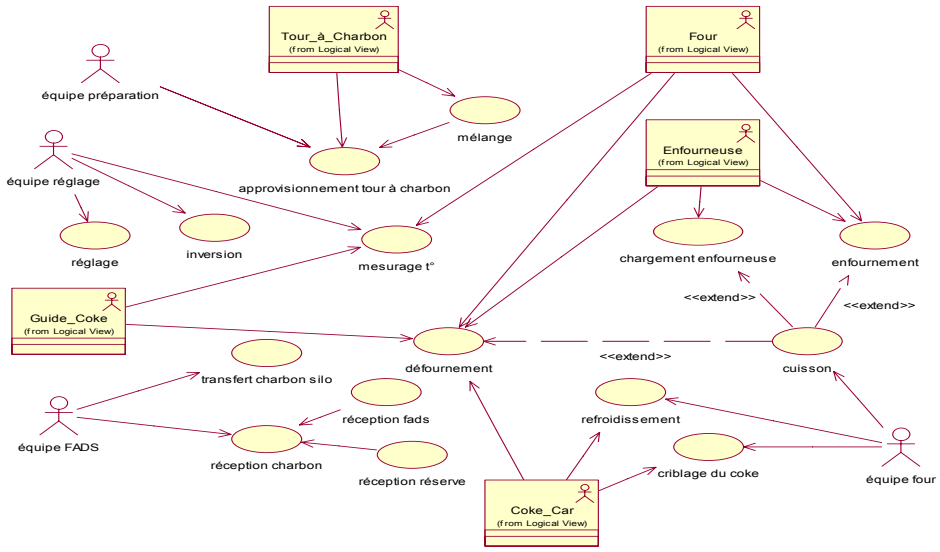

Figure 3: diagramme de cas d'utilisation

approvisionnement tour à charbon et mélange), secteur cuisson (chargement enfourneuse, enfournement, cuisson, inversion, mesurage t° et réglage) et secteur récupération (défournement, refroidissement et criblage du coke).

En plus de descriptions textuelles des fonctionnalités et des intervenants humains ou machines, un diagramme présente une vue organisationnelle des cas d'utilisation et des acteurs importants impliqués. Le diagramme montre le rôle des acteurs humains et machines (représentées par des classes stéréotypées acteur) impliqués dans des fonctionnalités qui seront décrites en détails par des diagrammes d'états et des diagrammes d'interaction. Ces diagrammes exprimeront la dynamique du système et des dialogues des acteurs. Nous verrons plus loin une modélisation de même niveau que les cas d'utilisation mais possédant une sémantique plus riche grâce aux diagrammes i*.

On notera en particulier:
- Le cas d'utilisation « cuisson », dont trois autres cas d'utilisation (chargement enfourneuse, enfournement et défournement) étendent la fonctionnalité primaire par différentes fonctionnalités nécessaires au processus de production. Cette extension est modélisée par le stéréotype « extend » dans le langage UML. Le détail et la séquentialité des opérations de chargement de l'enfourneuse, d'enfournement et du défournement du four à la fin de cuisson, seront modélisés par les diagrammes d'interactions.
- Le cas d'utilisation « réception du charbon » qui est une fonctionnalité générique des deux types de réceptions, la réception FADS et la réception réserve.
- Les acteurs représentés par des classes sont souvent actifs dans plusieurs cas d'utilisation au côté d'autres acteurs. Par exemple, pour le cas d'utilisation « défournement », quatre classes sont actives en plus de l'acteur « équipe four ».
- Les liens indiquent une relation de dépendance. Cela signifie qu'un changement dans un élément du modèle implique un changement dans l'autre élément.

Nous pouvons prendre comme exemple la description du cas d'utilisation '**Cuisson**' :
Objectif : Cuisson de la pâte à coke en vue de sa transformation en saumon de coke
Description : La pâte étant dans les fours, il faut procéder à la distillation du charbon à l'abri de l'air afin d'obtenir le saumon de coke. La température des carneaux du four est portée à 1300 degrés pour élever le charbon à une température voisine de 1000 degrés. Le temps de cuisson dépend des caractéristiques du four et de la batterie ainsi que du planning de production influençant les températures de cuisson dans les fours ou d'autres facteurs tels que panne, maintenance, imprévus, …
Intervenants : Equipe four, four
Cas d'utilisations associé: enfournement, défour-nement, chargement enfourneuse

Représentation *i.** Comme évoqué précédemment, les diagrammes de cas d'utilisation n'incluent que trois types de concepts (acteurs, cas d'utilisation et associations). Les diagrammes *i** permettent une meilleure identification des besoins et exigences du système en termes de buts fonctionnels et non fonctionnels. Le modèle de dépendances stratégiques (figure 4) met l'accent sur une perspective organisationnelle des acteurs dépendant les uns des autres tandis que le modèle de raisonnement stratégique (figure 5) se focalise sur la logique de raisonnement de chaque acteur pour l'accomplissement de ses buts.
Certaines spécifications formelles doivent également être représentées et commentées. L'exemple qui suit concerne le but '**Cuisson**' :

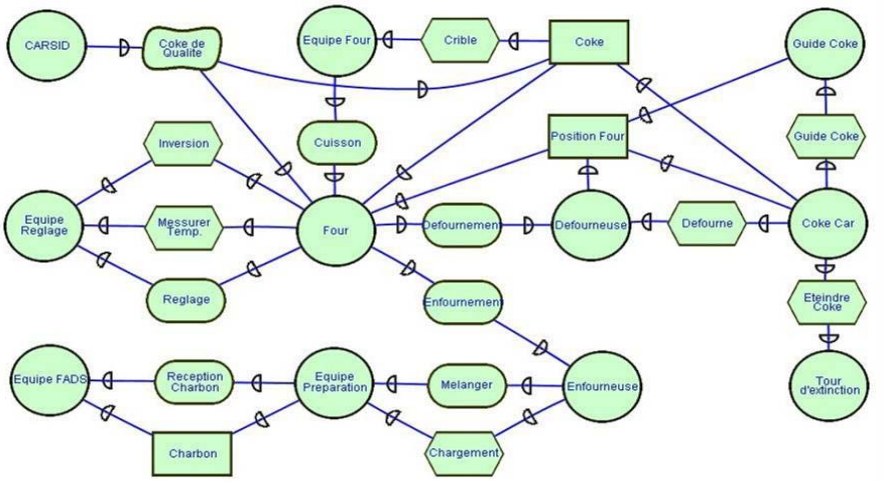

Figure 4: modèle de dépendances stratégiques

Dependency Cuisson
Type Goal
Mode Achieve
Depender EquipeFour ef
Dependee Four f
Attribute input : Matiere dans le four
Fulfillment
DansLeFour(input, f) ∧ Rempli(f) ∧ PateACoke(input) ∧ (f.porte.etat = 'fermée') →
$\Diamond_{16<t<19}$ SaumonCoke(input)

Le but 'Cuisson' est accompli si le four est rempli de pâte à coke, si la porte du four est dans l'état 'fermée' et si, après un temps t (temps de cuisson) compris entre 16 et 19 heures, la pâte à coke est cuite (saumon coke)

Diagramme d'activité de la cokerie. Il est possible de compléter les diagrammes de cas d'utilisation et i* par une représentation de la séquence des activités qui s'enchaînent tout au long du processus. Le diagramme d'activité (figure 6) permet en effet de

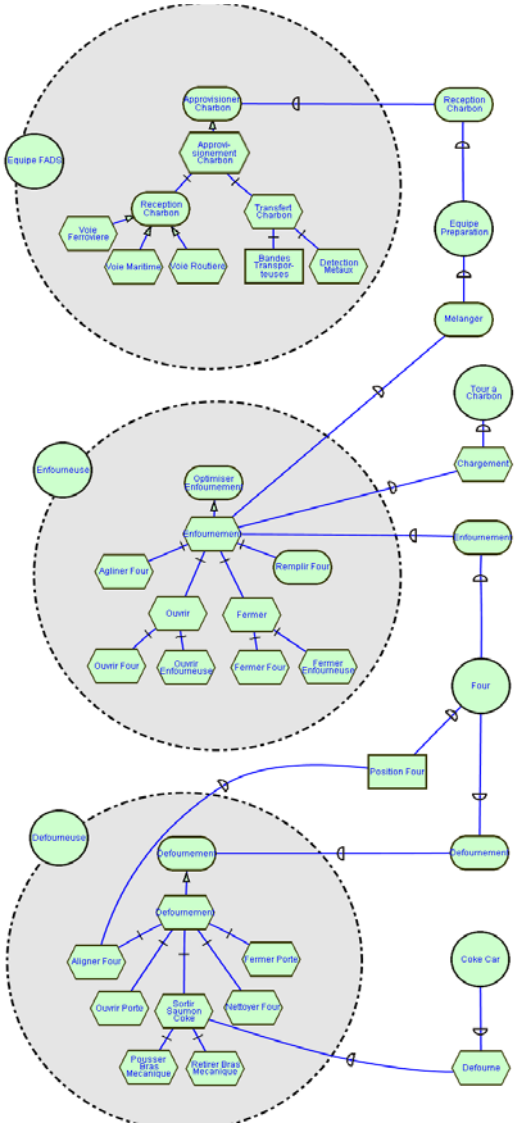

Figure 5 : modèle de raisonnement stratégique

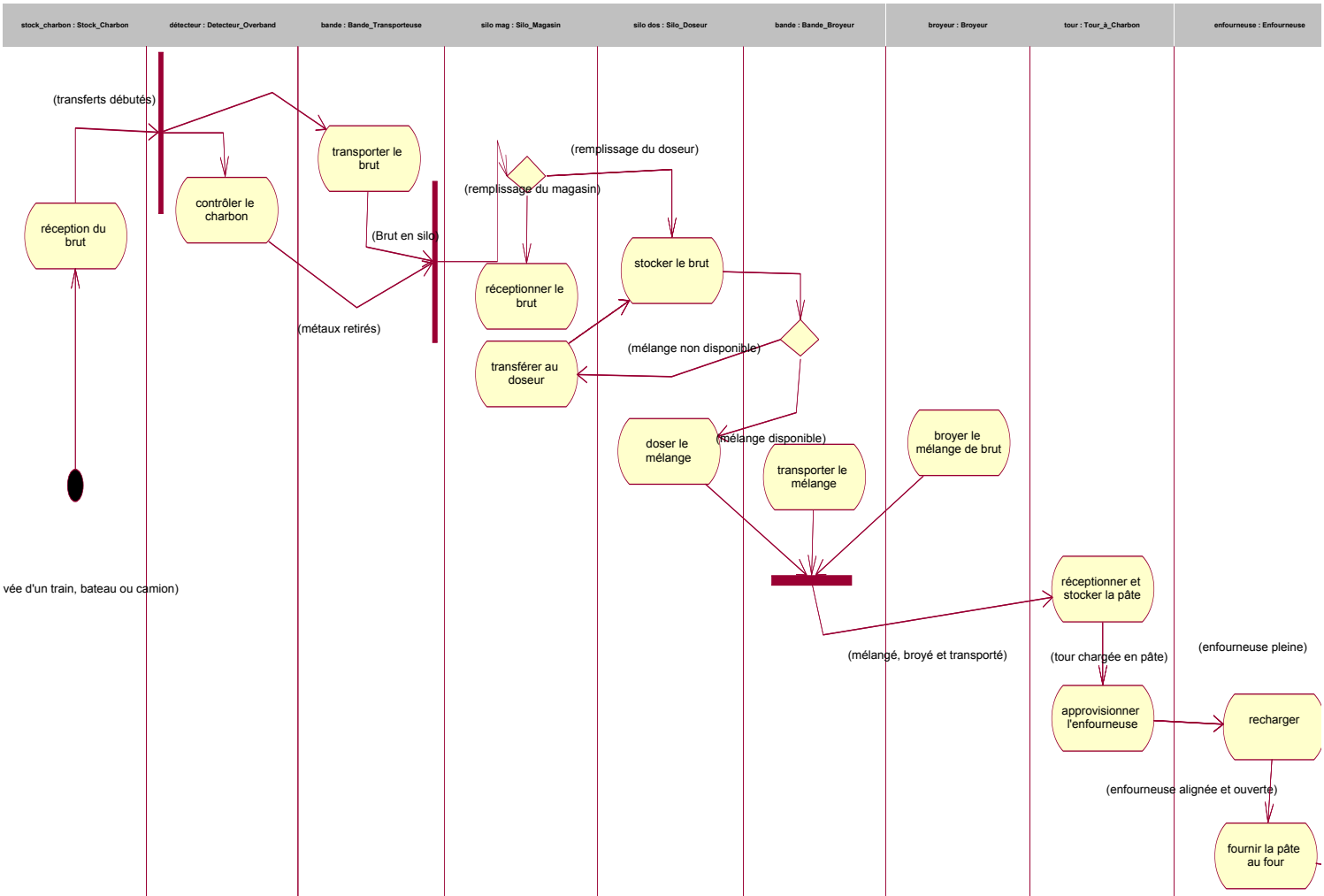

Figure 6: diagramme d'activité

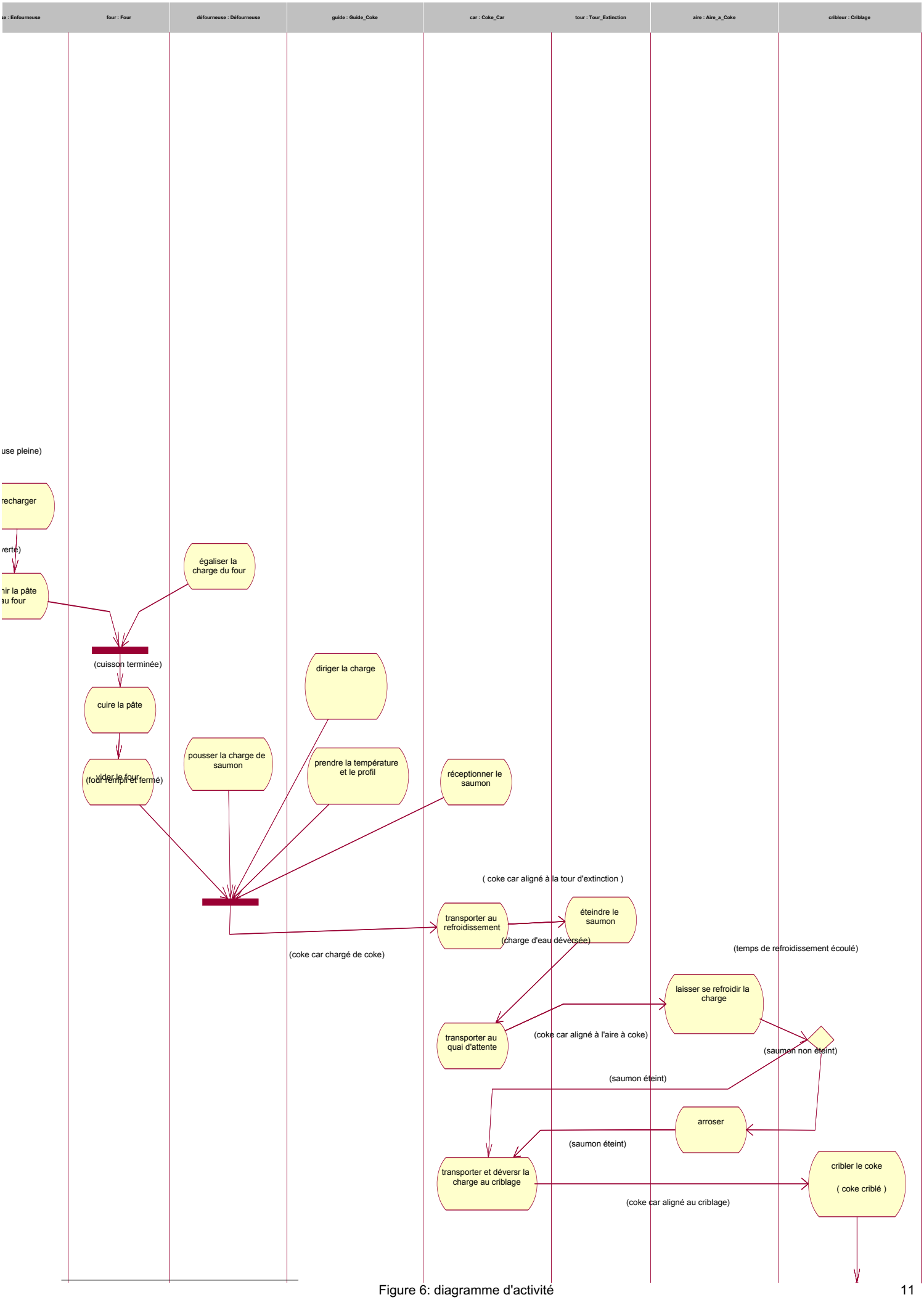

Figure 6: diagramme d'activité

représenter les enchaînements et les synchronisations nécessaires au bon fonctionnement du travail de production de coke. Il fournit aussi les conditions nécessaires à l'accomplissement de certaines actions.
Il est important de noter que les activités insistent sur des dynamiques, ceci de manière « process-driven », au contraire des diagrammes d'états, qui sont eux « event-driven » comme nous le verrons plus loin.
Les « swimlanes » dérivent directement des acteurs identifiés précédemment et sont représentés sous forme de divisions verticales qui permettent de séparer les activités des acteurs considérés comme des objets.

4.2 Modélisation conceptuelle

Diagrammes d'interactions. Le comportement dynamique des objets et des acteurs est représenté au moyen des diagrammes d'interaction : diagrammes de séquences et de collaborations. Pour les différents cas d'utilisation modélisés, un diagramme de séquences et un diagramme de collaborations représentent le détail du dialogue des objets pour l'accomplissement du cas d'utilisation. Dans un diagramme de séquences, il se dégage une structure temporelle des messages qui sont échangés entre les différents objets impliqués dans la réalisation d'un cas d'utilisation. La dimension verticale montre les enchaînements temporels des messages. Les réponses des différents objets aux messages reçus sont aussi clairement représentées et compréhensibles. La figure 7 représente les interactions du cas d'utilisation « Réception du charbon ». Il s'agit plus particulièrement de la livraison vers la réserve. Les messages vont des classes stocks vers le transporteur chariot racleur, qui transfère le brut demandé, dont la spécification est un attribut du message.

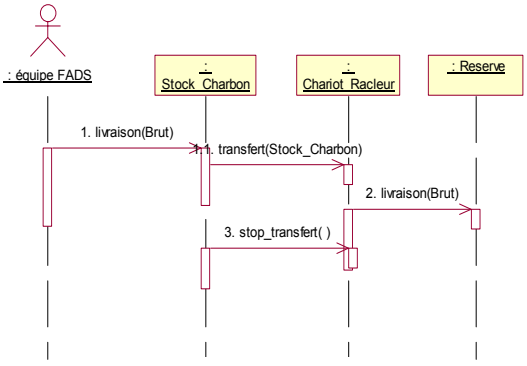

Figure 7: diagrammes de séquences de la réception du charbon

La figure 8 représente quant à elle le diagramme de collaboration issu du cas d'utilisation « Approvisionnement tour à charbon ».

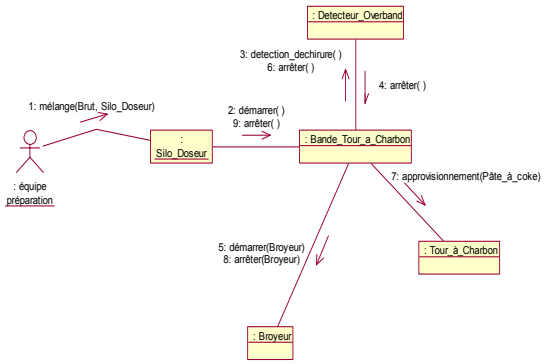

Figure 8: diagramme de collaboration pour l'approvisionnement de la tour à charbon

Diagrammes d'états transitions. Ce type de diagramme décrit les différentes transitions d'états qui s'opèrent au cours du temps de vie d'un objet. L'état est une période de temps au cours de laquelle l'objet satisfait une série de conditions, sur son état lui-même ou sur ce qu'il est en train d'accomplir. Les différents états de l'objet sont liés entre-eux par des transitions qui ont lieu lors de certains événements. Ces derniers peuvent être entre autre des messages généraux, des conditions temporelles,… L'exemple qui suit (figure 9) est le diagramme d'état de matière première. La matière première, le charbon, est décomposable en quatre états principaux, qui sont illustratifs de l'avancement du processus de cokéfaction. Des états d'entrée et de sortie sont présents dans le diagramme ; ils indiquent la naissance et la mort de l'objet.

Diagramme de classes. Le diagramme de classes (figure 10) est une représentation statique des classes d'objets et de leurs relations les unes avec les autres. Il a pour but de modéliser les différentes classes, leurs compositions et leurs associations.
Classe : Une classe est la description d'un groupe d'objets possédant des propriétés communes ainsi que des comportements similaires. Un objet est instance d'une classe particulière.
Attributs : Les attributs illustrent les propriétés des classes. Ils sont caractérisés par un type (entier, booléen,…). Les attributs ont une visibilité (privé, public ou protégé) qui dépend de leur nature.
Opérations : Les opérations constituent les comportements que les objets peuvent avoir, les

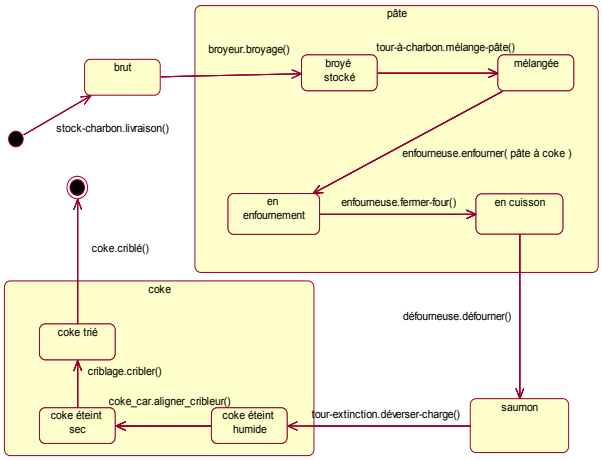

Figure 9: diagramme d'état de la matière première

actions dont ils connaissent la manière d'être conduites.

Associations : Les classes sont liées entre elles au moyen d'associations, connections représentées de manière conceptuelle par un lien ou par une classe. Elles sont caractérisées par des multiplicités ou cardinalités, qui indiquent le nombre d'objets pouvant être liés par une association. Elles peuvent exprimer des notions telles que la généralisation ou l'aggrégation.

4.3 DESCRIPTION DETAILLEE DU DEFOURNEMENT

Le diagramme de classes produit au cours de la phase d'analyse clarifie la structure et les fonctions de l'environnement du système d'information. Cette section a pour objectif de décrire plus en profondeur le défournement des fours à coke afin d'illustrer la logique utilisée pour créer le modèle UML de la cokerie et les spécifications pouvant être intégrées lors de l'utilisation de *Rational Rose*. Le modèle de classes du défournement (Figure 11) reprend des éléments fixes de la cokerie : batteries, fours, piedroits et carneaux; des éléments mobiles : enfourneuses, défourneuses, coke car, guide coke et auges mobiles; et des éléments organisationnels : planning des enfournements et défournements. Un diagramme d'état détaillé ainsi que les spécifications y afférant est également inclus dans cette section.

Cette description permettra enfin de comparer la modélisation de la cokerie avec les résultats générés par *Rational Rose*. Ceux-ci se situent à deux niveaux comme nous le verrons plus loin: tout d'abord la création d'une base de données SQL serveur. Ensuite, la génération de code Visual Basic, qui implémente les différentes classes dans un projet VB.

Eléments fixes

Les *Batteries* sont composées de *piedroits*. Ces derniers possèdent des *Carneaux*.

Chaque *Batterie* est caractérisée par un *nom* de type String (soit coppée1, coppée2, koppers, ou didier)

Chaque *Pied Droit* est caractérisé par un *numéro* de type Integer.

Chaque *Carneau* est caractérisé par un *numéro* de type Integer.

Chaque *Four* est caractérisé par :

- un *numéro* de type Integer ;
- un attribut *respecte_planning* de type Boolean dont la valeur est « true » tant que le planning des enfournements et des défournements est respecté pour ce four ;
- une fonction booléenne *porte_entrée_ouverte()* qui retourne la valeur « true » lorsque la porte du four côté défourneuse est ouverte et « false » lorsque cette porte est fermée ;
- une fonction booléenne *porte_sortie_ouverte()* qui retourne la valeur « true » lorsque la porte du four côté guide coke est ouverte et « false » lorsque cette porte est fermée ;
- une fonction booléenne *est_vide()* qui retourne la valeur « true » tant que le four est vide ;
- une fonction booléenne *est_enfourné()* qui retourne la valeur « true » une fois l'enfournement terminé ;
- une fonction booléenne *est_égalisé()* qui retourne la valeur « true » une fois le contenu du four égalisé par la défourneuse ;
- une fonction booléenne *cuisson_terminée()* qui retourne la valeur « true » une fois la cuisson du coke terminée ;
- une fonction booléenne *trappe_ouverte()* qui retourne la valeur « true » lorsque la trappe permettant à la défourneuse d'égaliser le contenu du four est ouverte et « false » lorsque cette trappe est fermée ;
- une procédure *activer()* qui permet la remise en service du four après maintenance (700°C -> 1300°C) ;
- une procédure *désactiver()* qui permet de diminuer la température du four afin d'en permettre la maintenance ou de mettre fin à son activité ;
- une fonction *température_estimée()* qui retourne une valeur de type Double (estimation de la température du four à partir des températures

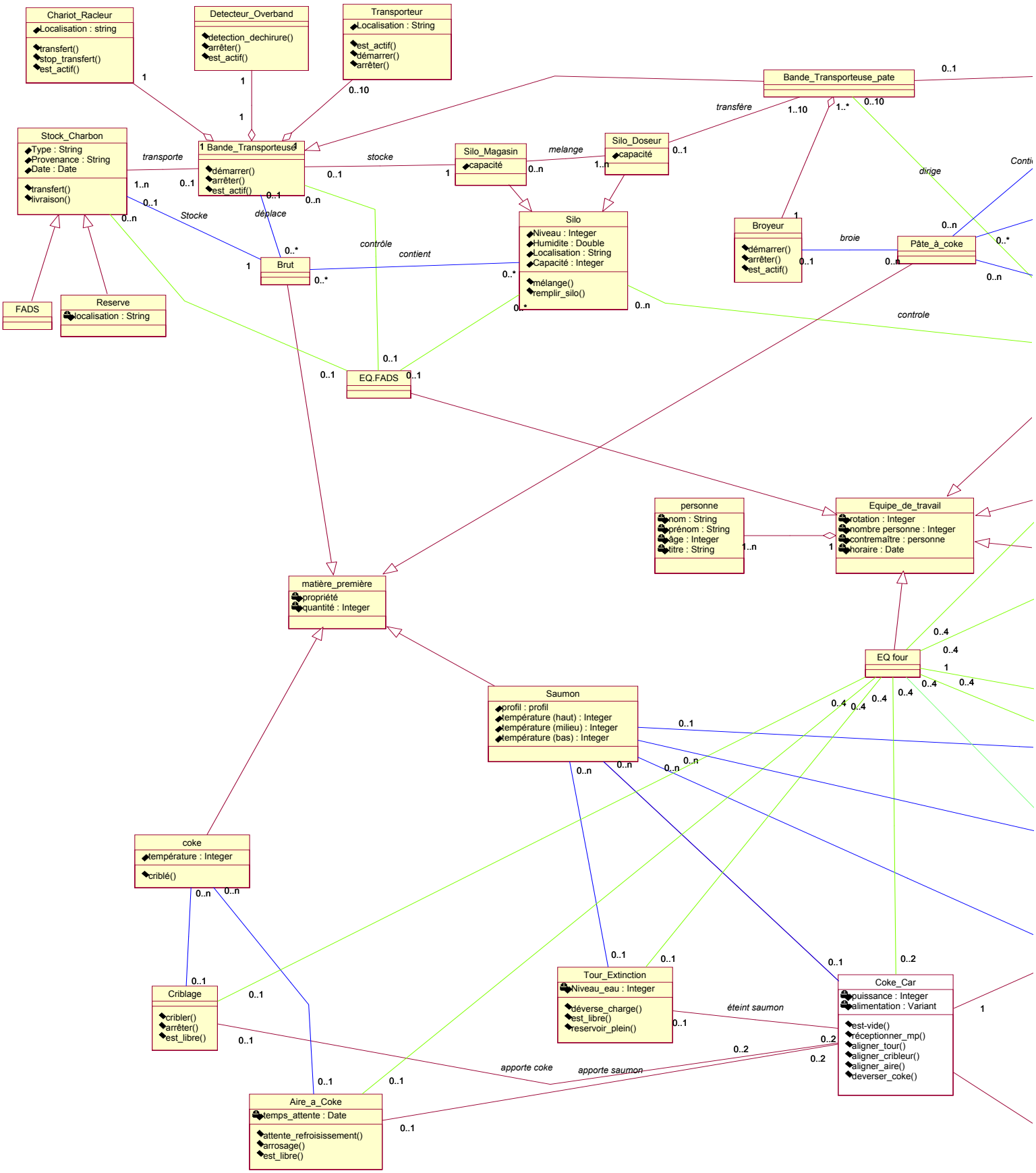

Figure 10: diagramme de classe

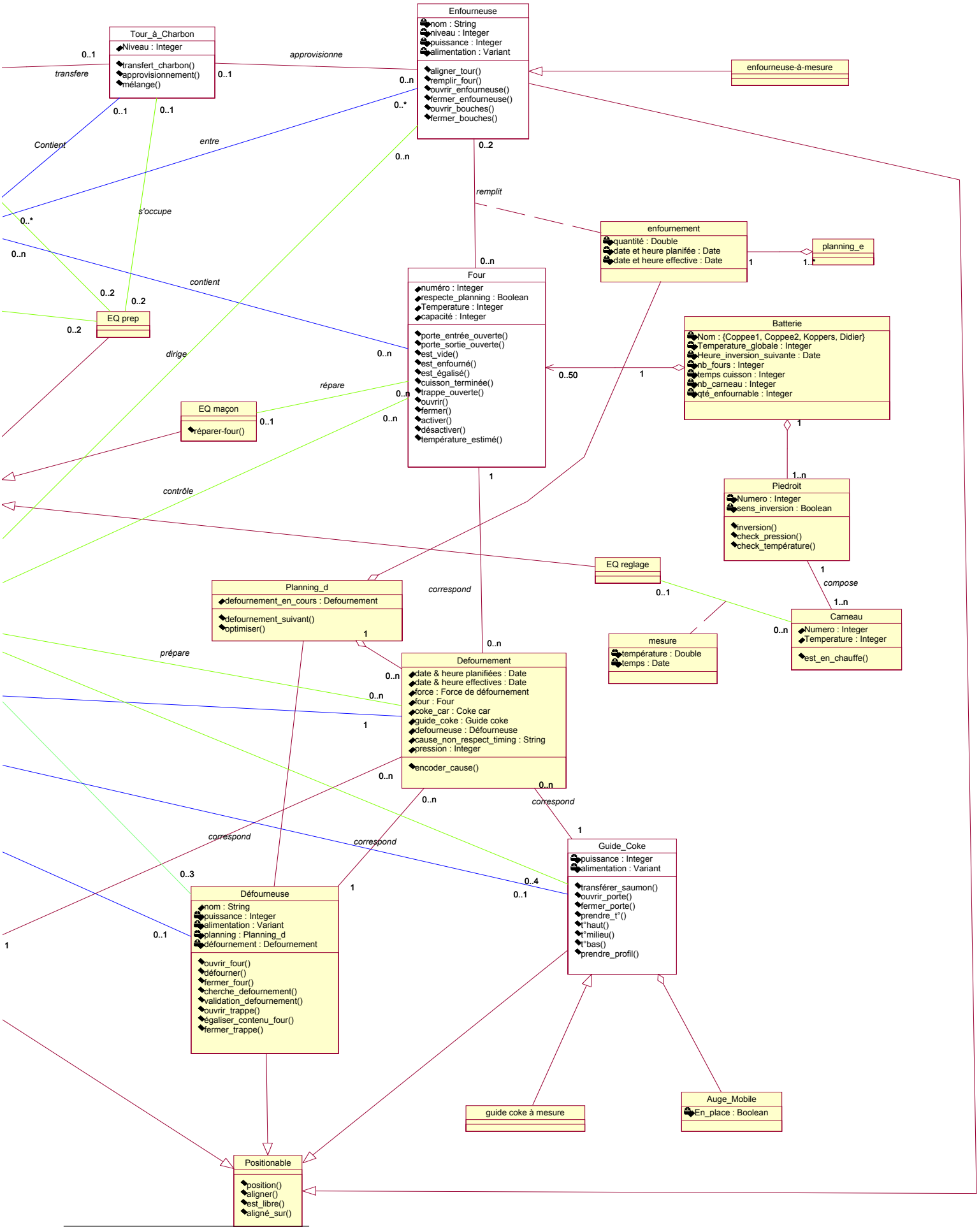

Figure 10: diagramme de classe

prises au niveau des carneaux des pieds droits qui l'entourent) ;
- deux procédures ouvrir() et fermer() qui s'appliquent au four.

Eléments mobiles
A l'exception de l'*Auge mobile* qui est solidaire du *Guide coke*, toutes les classes modélisant des éléments mobiles (*Enfourneuse*, *Coke car*, *Guide coke* et *Défourneuse*) héritent de la classe *Positionable*.

Positionnable : La classe *Positionable* possède trois méthodes :
- la fonction *position() : Four* qui retourne le four sur lequel est positionné le positionable
- la fonction booléenne *aligné_sur(f : Four)* qui retourne la valeur « true » lorsque le positionable est aligné sur le four f et « false » dans les autres cas ;
la procédure *aligner(f : Four)* qui permet d'aligner le positionable sur le four f (postcondition : aligné_sur(f)) ;
- la fonction est_libre() qui indique si le positionnable est en cours d'utilisation ou non.

En ce qui concerne le défournement, les classes *Coke car*, *Guide coke* et *Auge mobile* ne nécessitent pas d'être décrites plus en détail.

Défourneuse : La classe Défourneuse possède plusieurs attributs et neuf méthodes. Les attributs sont entre autres :
- *nom : String*, le nom de la défourneuse ;
- *planning : Planning*, le planning que doit respecter la défourneuse ;
- *defournement : Defournement*, le défournement en cours;

Les méthodes sont :
- *ouvrir_four():* ouvre la porte du four (defournement.four)
Preconditions:
aligné_sur (defournement.four)
not defournement.four.porte_entrée_ouverte()
Postconditions:
defournement.four.porte_entrée_ouverte()

- *défourner():* défourne le four defournement.four
Preconditions:
aligné_sur (defournement.four)
cuisson_terminée()
defournement.four.porte_entrée_ouverte ()
defournement.four.porte_sortie_ouverte ()
defournement.guide_coke.aligné_sur (defournement.four)

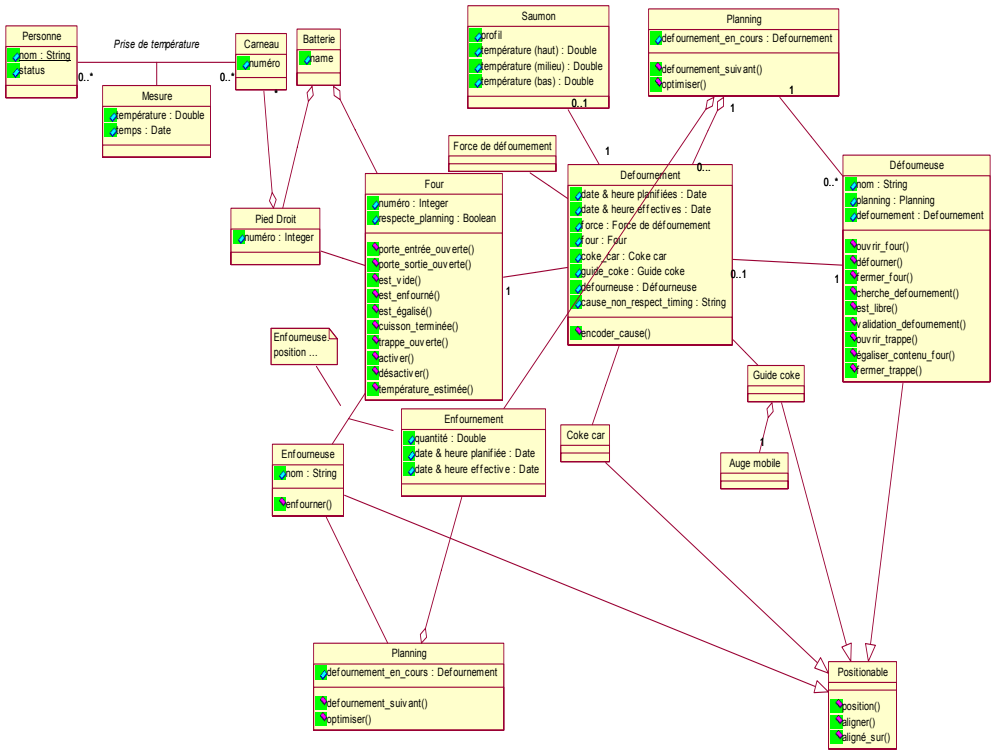

Figure 11 : Classes du défournement

defournement.coke_car.aligné_sur (defournement.four)
defournement.coke_car.est_vide
defournement.force = Void

Postconditions:

aligné_sur (defournement.four)
defournement.four.porte_entrée_ouverte ()
defournement.four.porte_sortie_ouverte ()
defournement.guide_coke.aligné_sur (defournement.four)
defournement.coke_car.aligné_sur (defournement.four)
not defournement.coke_car.est_vide ()
defournement.four.est_vide

- *fermer_four() :* fermer la porte d'entrée du four defournement.four

Preconditions:

aligné_sur (defournement.four)
defournement.four.porte_entrée_ouverte()

Postconditions:

aligné_sur (defournement.four)
not defournement.four.porte_entrée_ouverte()

- *cherche_defournement():*consulte le planning pour connaître le prochain defournement

Preconditions:

est_libre()
planning /= Void

Postconditions:

defournement = planning.defournement_en_cours

- *est_libre(): Boolean*, la défourneuse est-elle libre?
- *validation_defournement()*, Suivi et validation des enfournements et défournements effectués durant la pause précédente et repérage des fours hors séries

Preconditions:

defournement.date & heure effectives == Void

- *ouvrir_trappe()*, ouvrir la trappe du four en vue d'égaliser son contenu après l'enfournement

Preconditions:

not defournement.four.trappe_ouverte()
defournement.four.est_enfourné()

Postconditions:

defournement.four.trappe_ouverte()

- *égaliser_contenu_four()*, égaliser le contenu du four après enfournement

Preconditions:

defournement.four.est_enfourné()
not defournement.four.est_égalisé()
defournement.four.trappe_ouverte()

Postconditions:

not defournement.four.est_enfourné()
defournement.four.est_égalisé()
defournement.four.trappe_ouverte()

- *fermer_trappe()*, fermer la trappe du four

Preconditions:

defournement.four.trappe_ouverte()

Postconditions:

not defournement.four.trappe_ouverte()

Eléments organisationnels

Planning : Le *Planning* reprend la liste des *Defournements*.

La classe Planning possède un attribut et deux méthodes. L'attribut est :

- *defournement_en_cours : Defournement*, le defournement en cours.

Les méthodes sont :

- *defournement_suivant()*, met à jour *defournement_en_cours* avec le defournement qui suit sur le planning ;
- *optimiser()*, optimise le planning.

Enfournement : La classe *Enfournement* possède trois attributs :

- *quantité : Double*, la quantité enfournée ;
- *date & heure planifiée : Date*, la date et l'heure planifiées pour l'enfournement ;
- *date & heure effective : Date*, la date et l'heure auxquelles l'enfournement a effectivement eut lieu.

Défournement : La classe Defournement possède huit attributs et une méthode. Les attributs sont :

- *date & heure planifiées : Date*, la date et l'heure planifiées pour le défournement ;
- *date & heure effectives : Date*, la date et l'heure auxquelles le défournement a effectivement eut lieu ;
- *force : Force de défournement*, la force du défournement (ces données, la poussée en fonction du temps, sont représentées par le type (i.e., la classe) *Force de défournement*;
- *four : Four*, le four défourné;
- *coke_car : Coke car*;
- *guide_coke : Guide coke*;
- *defourneuse : Défourneuse*;
- *cause_non_respect_timing : String*, descriptions des causes de divergences entre l'heure de défournement planifiée et l'heure de défournement effective.

La méthode est :

- *encoder_cause(c : String)* : encodage des causes de divergences entre l'heure de défournement planifiée et l'heure de défournement effective.

Preconditions:

cause_non_respect_timing = Void
date & heure planifiées /= date & heure effectives

Postconditions:

cause_non_respect_timing = c

Saumon : La classe *Saumon* possède quatre attributs :

- *profil : Profil*, le profil du saumon mesuré par le profilocoke du guide coke lors du défournement ;
- *température (haut) : Double*, la température du haut du saumon de coke prise par le capteur du guide coke lors du défournement ;

Contrainte : au moins un des deux attributs (*profil*, *température (haut)*) doit avoir une valeur nulle.

- *température (milieu) : Double*, la température du milieu du saumon de coke prise par le capteur du guide coke lors du défournement ;
- *température (bas) : Double*, la température du bas du saumon de coke prise par le capteur du guide coke lors du défournement ;

Contrainte au défournement : les différents positionnables impliqués dans le défournement (*Défourneuse*, *Guide coke*, *Coke car*) doivent être alignés sur le four à défourné.

Diagramme d'état du four

Le four a une longue durée de vie et ne peut descendre pour des raisons de composition en dessous de 700°C. Le passage par cette température n'aura en conséquence lieu qu'à sa mise en activité et lors de son entretien. Le four passera aussi par cet état lors de sa mise hors service, c'est-à-dire la mort de l'objet. Son activité s'effectue à la température de 1300°C. Les différents états du four (figure 12) suivent le processus de cuisson : il est tout d'abord en rempli (« est enfourné »), égalisé (« est égalisé »), avec des périodes d'attente entre ces états. La cuisson a lieu. Lorsqu'il est prêt, il est mis en attente de défournement après avoir été ouvert (« défourneuse.ouvrir_four() » et « guide_coke.ouvrir_porte() »). A la fin de cette opération, il passe par l'état « défourné ». Il est donc vide et peut passer dans deux états :

- soit il passe par l'entretien et sa température baisse jusqu'à 700°C
- soit il retourne à l'état initial d' « attente défournement » avant d'être enfourné à nouveau.

En plus du diagramme d'état, les spécifications des états du four sont spécifiées ci-dessous. Elles apportent des définitions formelles des différents états en fonction des différentes méthodes qui ont été développées pour le four dans le modèle de la cokerie.

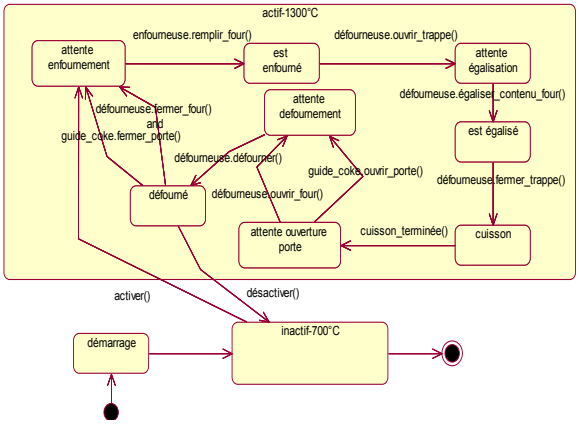

Figure 12 : diagramme d'état du four

1. inactif-700°C

Documentation: la température est abaissée aux environs de 700°C pour permettre la maintenance du four
Définition : température < 800 **and** température > 650
Evénement (entrée) : entretien ou démarrage (spécifications à formuler)
Evénement (sortie) : remise en route ou fin de vie (spécifications à formuler)

2. actif-1300°C

Documentation: la température est supérieure à 1200°C pour la production de coke
Définition : température > 1200
Evénement (entrée) : remise en route (spécification à formuler)
Evénement (sortie) : entretien (spécification à formuler)
Sous-états : Attente Enfournement, Est Enfourné, Attente Egalisation, Est Egalisé, Attente Ouverture Porte, Attente Défournement, Défourné et Cuisson.

Cuisson

Documentation: la cuisson du coke
Définition :
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not est_vide()
est_enfourné()
est_égalisé()
not cuisson_terminée()
not trappe_ouverte()
Evénement (entrée) : Défourneuse.fermer_trappe()
Evénement (sortie) : cuisson_terminée()

Attente Enfournement
Documentation: avant l'enfournement de la pâte à coke
Définition :
est_vide()
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not trappe_ouverte()
not est_enfourné()
not est_égalisé()
not cuisson_terminée()
Evénement (entrée) : Défourneuse.fermer_four() **and** Guide coke.fermer_porte()
Evénement (sortie) : Enfourneuse.remplir_four()

Est Enfourné
Documentation: le four est rempli de pâte à coke
Définition :
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not est_vide()
est_enfourné()
not est_égalisé()
not cuisson_terminée()
not trappe_ouverte()
Evénement (entrée) : Enfourneuse.remplir_four()
Evénement (sortie) : Defourneuse.ouvrir_trappe()

Attente Egalisation
Documentation: le four, rempli de pâte à coke, attend trappe ouverte que son contenu soit égalisé par la défourneuse
Définition :
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not est_vide()
est_enfourné()
not est_égalisé()
not cuisson_terminée()
trappe_ouverte()
Evénement (entrée) : Defourneuse.ouvrir_trappe()
Evénement (sortie) : Defourneuse.égaliser_contenu_four()

Est Egalisé
Documentation: le four est rempli de pâte à coke égalisée
Définition :
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not est_vide()
est_enfourné()
est_égalisé()
not cuisson_terminée()
trappe_ouverte()
Evénement (entrée) : Defourneuse.égaliser_contenu_four()
Evénement (sortie) : Defourneuse.fermer_trappe()

Cuisson
Documentation: le coke cuit
Définition :
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not est_vide()
est_enfourné()
est_égalisé()
not cuisson_terminée()
not trappe_ouverte()
Evénement (entrée) : Defourneuse.fermer_trappe()
Evénement (sortie) : cuisson_terminée()

Attente Ouverture Porte
Documentation: le four est rempli de coke cuit
Définition :
not porte_entrée_ouverte()
not porte_sortie_ouverte()
not est_vide()
est_enfourné()
est_égalisé()
cuisson_terminée()
not trappe_ouverte()
Evénement (entrée) : cuisson_terminée
Evénement (sortie) : Guide coke.ouvrir_porte() **and** Defourneuse.ouvrir_four()

Attente Défournement
Documentation: le four est rempli de coke cuit et ses portes sont ouvertes
Définition :
porte_entrée_ouverte()
porte_sortie_ouverte()
not est_vide()
est_enfourné()
est_égalisé()
cuisson_terminée()
not trappe_ouverte()
Evénement (entrée) : Guide coke.ouvrir_porte() **and** Defourneuse.ouvrir_four()
Evénement (sortie) : Defourneuse.défourner()

Défourné
Documentation: le four est vide
Définition :
porte_entrée_ouverte()
porte_sortie_ouverte()
est_vide()
not est_enfourné()
not est_égalisé()
not cuisson_terminée()
not trappe_ouverte()
Evénement (entrée) : Defourneuse.défourner()
Evénement (sortie) : Defourneuse.fermer_four() **and** Guide coke.fermer_porte()

5 CONCEPTION ASSISTEE AVEC *RATIONAL ROSE*

5.1 Forward Engineering sur *Rational Rose*

Les travaux qui ont été présentés précédemment constituent la phase d'analyse du cycle de développement UML. La structure du système et le comportement statique et dynamique de la cokerie ont été décrits selon les différents diagrammes existants. Le passage à la phase de design devrait permettre d'implémenter des solutions au modèle. Deux types de design assisté ont été testés.
Tout d'abord, un projet Visual Basic 6.0, produit de Microsoft. Le choix n'est pas neutre : il s'agit du langage de programmation utilisé par Carsid. L'exemple de génération sera donc plus pertinent que l'utilisation de n'importe quel autre langage. Il sera comparable à la structure existante et compréhensible par les personnes qui seront amenées à consulter ces travaux.
Ensuite, une base de données relationnelle en SQL Server 7.0. L'objectif est lui aussi de comparer les résultats de ces travaux avec les bases de données existantes (ce travail sera accompli dans la deuxième phase de l'étude initiée dans le présent document). Cette comparaison sera assez utile dans le but d'implémenter des changements dans les structures de bases de données de Carsid. Le modèle qui a été développé dans ce projet se rapproche dans bien des aspects des bases de données existantes, sous une forme ou une autre, dans l'entreprise Carsid.
Le choix d'utiliser le produit de développement UML *Rational Rose Entreprise Edition 2002* est justifié par sa complétude, sa solidité et sa popularité. Il permettra en plus de générer les deux types de scripts nécessaires : un projet Visual Basic et un script SQL Server. Il permettra aussi d'assurer la rétro-ingénierie c'est-à-dire la transformation de code en un modèle objet, autant pour SQL que pour Visual Basic, ce qui facilitera aussi les travaux futurs.

Code Visual Basic. Le code Visual Basic qui a été généré par *Rational Rose* a été transposé dans un projet de Visual Basic 6.0. Le code complet du modèle représente un volume assez important et ne sera pas présent ici. Seul le script du four est montré comme exemple :

```
Option Explicit

Public numéro As Integer
Public respecte_planning As Boolean
Public Temperature As Integer
Public capacité As Integer
Public NewProperty As Collection
Public NewProperty2(0 To 4) As EQ_four
Public NewProperty3 As EQ_maçon
Public NewProperty4 As Collection
Public NewProperty8 As Collection

'La porte d'entrée est-elle ouverte?
Public Function porte_entrée_ouverte() As Boolean
End Function

'La porte de sortie est-elle ouverte?
Public Function porte_sortie_ouverte() As Boolean
End Function

'Le four est-il vide?
Public Function est_vide() As Boolean
End Function

'Le four est-il enfourné?
Public Function est_enfourné() As Boolean
End Function

(…)
'message général déclanchant l'ouverture du four.
'il s'ensuit l'intervention de la défourneuse et du guide coke qui se
chargent d'ouvrir les portes latérales.
Public Sub ouvrir()
End Sub

'message général déclanchant la fermeture du four.
'il s'ensuit l'intervention de la défourneuse et du guide coke qui se
chargent de fermer les portes latérales.
Public Sub fermer()
End Sub
(…)
```

Base de données SQL Server 7.0. La transformation d'un modèle objet en modèle de données nécessite quelques transformations du modèle UML. Il est possible d'attribuer certaines caractéristiques spécifiques au modèle. Les options disponibles permettent d'implémenter les tables, leurs liens mutuels ainsi que leurs éléments. Les clés permettent de relier les tables entre elles et d'assurer l'intégrité référentielle dans les relations entre les tables. Les contraintes de clés peuvent être définies au préalable ou, à défaut, elles seront choisies par le modeleur de données de *Rational Rose*.

Les contraintes créées sont les contraintes de clé primaire, les contraintes de clé unique et les contraintes de clé étrangère.
Une table (table du four) est présentée ici à titre d'exemple. Les commentaires sur le script sont indiqués entre parenthèses :

```
GO
CREATE TABLE T_Four (
    numéro INT NOT NULL,
    respecte_planning BIT NOT NULL,
    Temperature INT NOT NULL,
    capacité INT NOT NULL,
 (Les attributs ont été crées et leur type identifié)
T_Four_ID INT IDENTITY NOT NULL,
    T_Batterie_ID INT NOT NULL,
    T_Equipe_de_travail_ID INT NOT NULL,
    COL_6 INT NOT NULL,
 (Le lien avec les autres tables a été créé)
    CONSTRAINT PK_T_Four19 PRIMARY KEY
NONCLUSTERED (T_Four_ID)
    )
```

Il est à noter qu'avant de générer un script SQL de type DDL, donc de niveau conception physique ayant pour cible un SGBD (système de gestion de bases de données) particulier, en l'occurrence SQL Server 7.0, *Rational Rose* exige la transformation du modèle conceptuel (diagramme de classe) en un modèle de niveau conception logique, en l'occurrence un modèle de base de données relationnel. Cela se réalise par la déclaration des classes dans le diagramme de classes comme « persistantes » et la transformation automatique de ce diagramme de classes en un diagramme de modèle de données. Celui-ci est représenté à la figure 13 pour ce qui concerne les classes du défournement. Après quoi, ce diagramme de base de données relationnel pourra être utilisé pour générer un script SQL. La figure 14 donne le résultat du script SQL Server 7.0 généré à partir du modèle relationnel de la figure 13 et chargé dans SQL Server 7.0.
Il faut également noter que, dans la suite du projet, nous étudierons et présenterons les possibilités dues à l'application de technologies de data mining et de data warehouse et nous développerons des exemples de cubes OLAP (On-Line Analytical Processing). C'est pour ces raisons que nous utiliserons SQL Server 2000 car cette version inclu des améliorations importantes par rapport à la version précédente en ce qui concerne ce type d'études.

5.2 Reverse Engineering sur *Rational Rose*

Cette partie introduit la deuxième phase de l'étude à savoir la conception type en utilisant UML/*Rational Rose* d'une base de données intégrée basée sur l'existant de Carsid et plus particulièrement les bases de données liées au processus cokier.
La connaissance des différentes bases de données existantes ou en projet doit permettre de faciliter les travaux de création d'un système unifié. La cokerie de Carsid a une architecture basée sur plusieurs niveaux de bases de données et d'applications. Des données sont prises au moyen de capteurs disposés le long de la chaîne de production ou saisies par des employés. Ces données sont traitées sur plusieurs niveaux :

- sur les machines (pc ou autres) des employés d'un secteur particulier;
- au niveau de la cokerie en général, en passant par des bases de données centralisées;
- au niveau du processus industriel global, le partage d'informations se fait avec les autres secteurs situés en amont et aval de la cokerie.

Dans un premier temps, un inventaire des bases de données disponibles chez Carsid a été réalisé. Celles-ci ont été groupées selon la description des cas d'utilisation qui a été faite à la section 4.1. L'objectif de cette description est de rapprocher le modèle développé de la réalité des systèmes existants. Différents systèmes se côtoient, les bases de données sont de différents types. Les plus anciens systèmes sont des VAX, mais le remplacement de ces systèmes est en cours. La création récente d'une base de données utilisant SQL Server 7.0 a permis de centraliser un certain nombre de bases de données qui fonctionnaient sur des VAX ou d'autres systèmes. Cette centralisation devrait permettre de faciliter les travaux de création du nouveau système.
La deuxième partie concerne la rétro-Ingénierie et l'intégration des bases de données de Carsid. A partir des scripts SQL récupérés, nous allons, par rétro-ingénierie, récupérer les bases de données au niveau conception physique et produire le diagramme de classes pour chacun des schémas relationnels. Ces schémas relationnels seront alors intégrés dans un schéma logique commun portant sur une base de donnée commune de type SQL server 7.0. A partir de ce schéma, un diagramme de classes sera regénéré.
L'objectif sera alors de comparer ce diagramme de classes provenant des bases de données existantes chez Carsid et le diagramme de classe 'théorique' (figure 10) pour intégrer au mieux le premier dans le second.

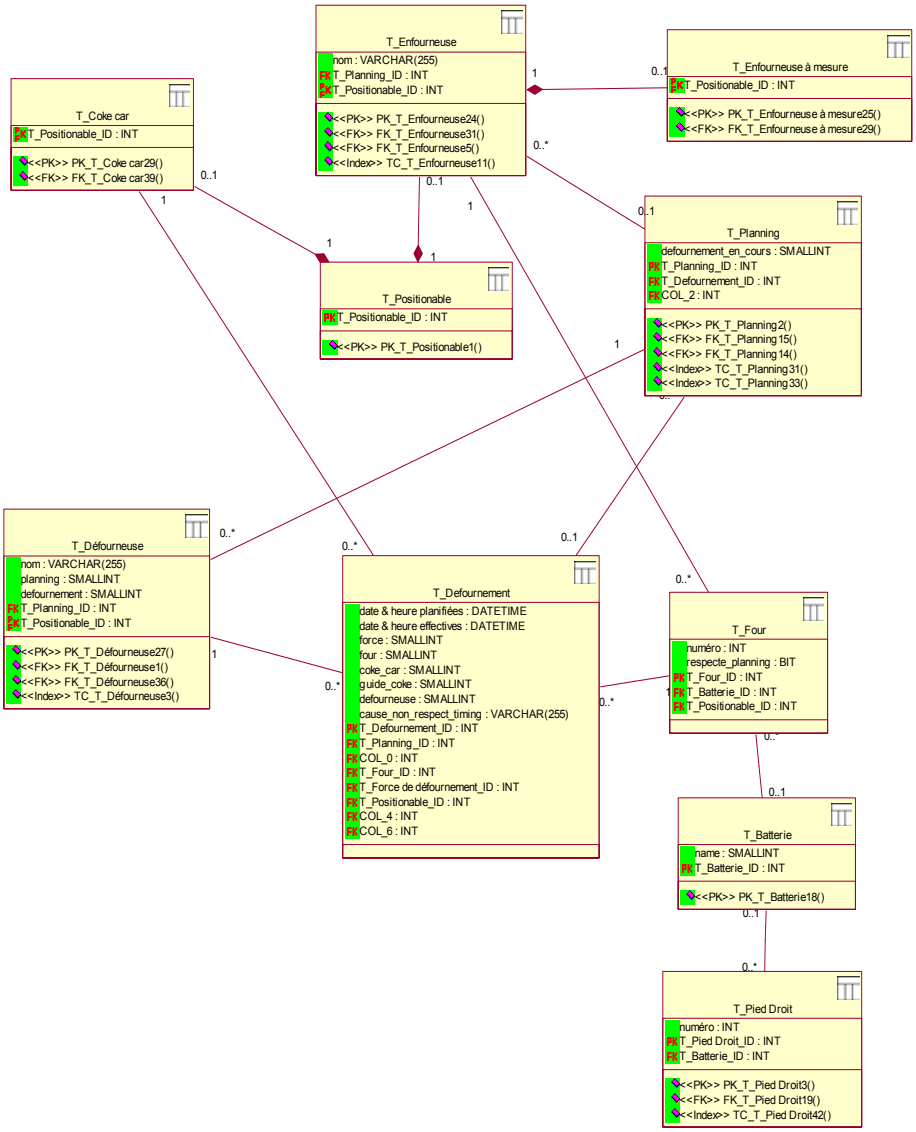

Figure 13: extrait du schéma relationnel de la cokerie

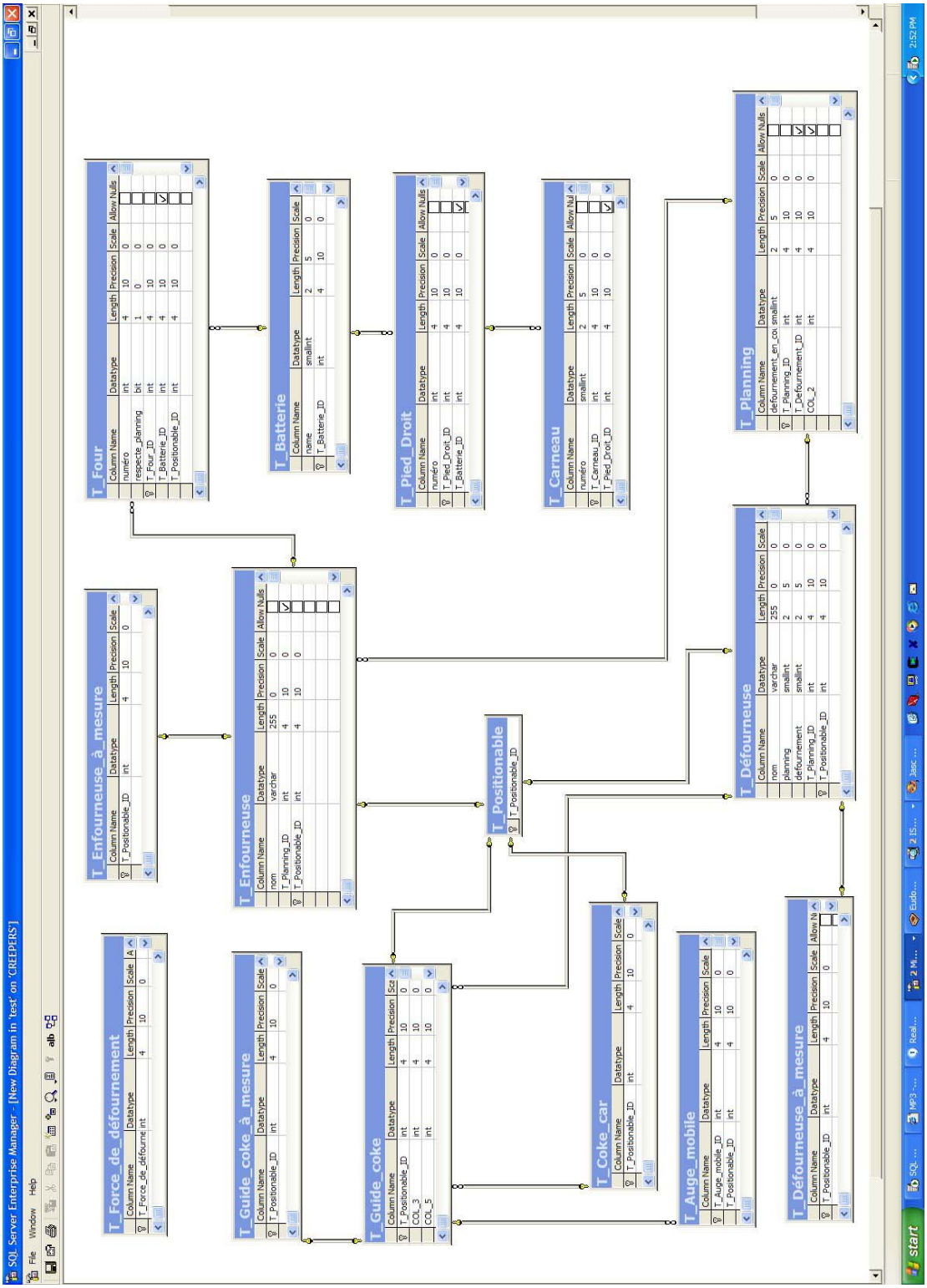

Figure 14: base de données SQL Server 7.0 pour la figure 13

6 CONCLUSION

La première phase de cette étude a permis d'ouvrir la route aux futurs travaux de développement. Nous avons développé dans ce travail une modélisation étendue de la cokerie et présenté différentes conceptions de code et de scripts sur base du modèle créé.

L'analyse du processus de production de la cokerie est satisfaisante. Elle présente en effet de nombreuses similitudes avec les systèmes déjà installés dans la cokerie de Carsid. La synthèse des bases de données disponibles nous a permis d'évaluer l'exactitude de l'analyse et de commencer la rétro-ingénierie des bases de données existantes. Il reste cependant des différences entre les informations réellement disponibles concernant le processus et les modèles développés dans cette étude. Cependant, les modèles reproduisent correctement le travail de la cokerie et ont été utilisés dans les formations à UML pour Carsid. Il a permis aux personnes qui sont appelées à travailler chez Carsid de se familiariser à UML dans le milieu familier de la production du coke.

Les éléments de design qui ont été développés ont permis de comprendre l'intérêt d'utiliser un outil informatique d'appui à la modélisation. L'utilisation du modèle dans le sens modèle vers base de données permettra d'utiliser de manière réelle les travaux de modélisation. Les capacités d'ingénierie inverse ont été testées. Elles seront utilisées pour actualiser le modèle et rassembler les travaux de cette étude avec les systèmes déjà existants.

Le défi principal est maintenant d'analyser plus en profondeur les systèmes existants chez Carsid. Il s'agira ensuite d'intégrer les éléments actuels dans un futur système standardisé, dans les limites de l'adaptabilité de l'informatique de la cokerie de Carsid. On peut espérer à plus long terme voir les différentes usines de l'entreprise Carsid fonctionner sur un système standardisé basé sur UML et des technologies similaires. Lorsque cet objectif sera accompli, l'information sera alors exploitable dans une perspective encore plus large et sa valeur augmentera comme ressource critique de l'entreprise.

REFERENCES

Anonyme, *Synthèse des applications informatiques à la cokerie de Charleroi* (document interne), CARSID, Marchienne-au-Pont, 2002.

Anonyme, *La cokerie de Marchienne* (document interne), groupe Cockerill Sambre, Marchienne-au-Pont.

P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, *Database systems: concepts, languages & architectures*, McGraw Hill College Div, 1999.

F. Balena, *Programming Microsoft visual basic 6.0*, Microsoft Press, 1999.

G.Booch, I.Jacobson, James Rumbaugh, *The unified modelling language user guide*, Addison Wesley Pub Co, 1998.

J. Castro, M. Kolp, J. Mylopoulos, *Towards requirements-driven information systems engineering: the tropos project*, In Information Systems(27) Pergamon, Elsevier, 2002.

L.Chung, B.A.Nixon, E.Yu, J.Mylopoulos, *Non functional requirements in software engeneering*, Kluwer Academic Publishers, 2000.

A. Donnay, M. Kolp, D. Massart, T. T. Do, S. Faulkner, and A. Pirotte. *Modélisation orientée objet de la cokerie de CARSID*. Rapport final Carsid, Août 2002.

R.A.Elmarsi, S.B.Navathe, Fundamentals of database systems (3rd edition), Addison Wesley Pub Co, 2000.

F. Fouss, M. Ibarz, M. Kolp, *Object-oriented reengineering of the steel production databases at Carsid,* Working Paper IAG, Université Catholique de Louvain, A paraître, 2003.

J.Rumbaugh, G.Booch, I.Jacobson, *The unified modelling language reference manual (Addison-Wesley object technology series)*, Addison Wesley Pub Co, 1998.

C.Sontou, *de UML à SQL, conception de bases de données*, Editions Eyrolles, 2002.

R.Vieira, *Professional SQL server 7.0 programming*, Wrox press Inc., 1999.

Powerdesigner, http://www.sybase.com

Rational, http://www.rational.com

Together, http://www.togethersoft.com