République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Intelligence Artificielle

Sujet :

Réutilisation des Procédés Logiciels a base d'Agents

Présenté par: AMROUCHE Abdelmadjid
AZROU ISGHI Fatima Zohra

Promoteur : Mme AOUSSAT .F

Soutenue le: 07/07/07, devant le jury composé de :

Nom.Mme. Bensttiti, C.C, USDB	**Président**
Nom.Mlle. Boustia M.A, USDB	**Examinateur**
Nom.M.Hammouda A , USDB	**Examinateur**

- 06/2006-2007-

Remerciements

Nous remercions le bon Dieu pour nous avoir guider vers le bon chemin du savoir, pour nous avoir donner du courage et de la volonté afin de pouvoir réaliser ce mémoire.

Nous tenons à remercier notre promotrice Mme Aoussat .F pour son soutient inconditionnel, son apport décisif et pour tous ses conseils.

A tout les enseignants de la faculté des science exactes de Blida et surtout les enseignants du département d'informatique.

Enfin, nous remercions, de tout cœur, tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Dédicace

A ma raison de vivre, d'espérer,
A ma source de courage, à celle qui j'ai de plus cher :
Ma *maman* ,
Pour leur amour , leur confiance et leur sacrifice sans limites,
A mes trois adorables frères :
Zaki ,Abdelhalim et Mohamed, à mes sœurs Nassima et Rachida.

Abdelmadjid

A ma raison de vivre, d'espérer,
A ma source de courage, à ceux qui j'ai de plus cher :
Mon *papa* , ma *maman* ,
Pour leurs amour , leurs confiance et leurs sacrifice sans limite,
A mes deux adorables frères :
Omar et Alaedine, à ma sœur Hasna.

Fatima Zohra

Résumé

Ce travail se situe dans le cadre du génie logiciel qui vise une meilleure maîtrise du développement de logiciels. Cet objectif nous amène à nous intéresser en particulier à la modélisation et à la mise en oeuvre assistée des procédés logiciels centré activité, en exploitant les concepts de réutilisation des composants procédés logiciels.

Le développement d'un logiciel est une activité généralement complexe qui implique la coopération de différents acteurs. Pour prendre en compte cette réalité, nous proposons une approche pour la modélisation et la mise en oeuvre de procédés, à l'aide des systèmes multi agents.

L'amélioration de la description et de la réutilisation des procédés logiciels est enjeu essentiel du Génie logiciel dans la décennie avenir.

Mots-clés

Procédé de développement, Réutilisation des procédés logiciels, Composants procédés, Base de composants, Procédés logiciels, Stockage des fragments procédé, Recherche des fragments procédé, Agents réactifs.

Abstract

This work is classified in the fields of software engineering which aims at a better control of the software development. This objective leads us to interest us in particular in modelling and the implementation assisted of the software processes centred activity, by exploiting the concepts of re-use of the software process components.

The software development is generally a complex activity which implies the co-operation of various actors. To take into account this reality, we propose an approach for modelling and implementing of processes has using systems multi agents.

The improvement of the description and the re-use of the software processes is essential stake of the software in the decade future engineering.

Key Words

Development processes, Re-use of the software processes, Component processes, Bases components, software processes, Stockade of the fragments processes, Re-church of the fragments processes, Agents reactive.

TABLES DES MATIERS

PAGE

CHAPITRE I

CHAPITRE II

PARTIE 01

PARTIE 02

CHAPITRE III

PARTIE 01

CHAPITRE IV

CHAPITRE V

Chapitre II

PARTIE I

PARTIE II

PARTIE III

Chapitre III

PARTIE I

PARTIE II

Chapitre IV

Chapitre II

PARTIE III

Chapitre III

PARTIE I

Chapitre I

Introduction

INTRODUCTION GENERALE

La maîtrise des processus de développement du logiciel est une tâche complexe qui implique la coopération de différents acteurs et qui doit prendre en considération différents produits, outils et contraintes.

Chaque programmeur a ses astuces pour accomplir la tâche qu'on lui a affectée. Une des astuces utilisées est la réutilisation

Puisque le but des "Procédés Logiciels" est de formaliser et d'automatiser la répétitivité de certaines suites de tâches du développement logiciel, on peut les utiliser dans la réutilisation.

Problématique :

Lors du développement d'un produit logiciel, la réutilisation de partie (fragment) de logiciel déjà existantes telles que : du code source, des interfaces, des idées, des raisonnements se fait depuis toujours et généralement intuitif. Ces parties sont jugées intéressantes parce que d'un côté, elles reviennent toujours dans certains types de logiciels, tel que les interfaces de saisie; et d'un autre coté, ce sont des parties du logiciel déjà prêtes à l'emploi et qui ont fait leur preuve; par souci de gain de temps et d'argent, dans certains cas, il est préférable de faire du « copier-coller » avec des modifications que de le refaire en entier. Ceci peut s'appliquer à tous type de logiciel y compris les procédés logiciels. Le responsable de la réalisation du procédé logiciel peut, lui aussi, réutiliser des parties de procédés logiciels existantes, qui ont fait leur preuve pour modéliser un nouveau procédé logiciel qui répond aux demandes formulées d'un nouveau procédé logiciel.

Non seulement la réutilisation "copier-coller" des composants logiciels n'est pas si évidente que ça, mais aussi, «... la réutilisation des procédés logiciel diffèrent de celle des composants code source, les procédés logiciels sont typiquement plus complexes... » **[mi 92].** Ainsi la réutilisation des composants de procédés logiciels se heurte a plusieurs problèmes les plus importants sont :

1- La plupart de ces procédés logiciels sont rigides, et difficilement modifiables **[Jam 05]**.

2- La plupart de ces procédés se basent sur un grand nombre de concepts spécialisés. Ces concepts sont souvent complexes, et leur nombre important surcharge le modèle de procédé et ajoute beaucoup de difficulté à sa compréhension, par conséquent, à sa réutilisation **[Jam 05].**

3- Les concepts et les entités utilisés peuvent changer d'un procédé à un autres cela dépend bien de beaucoup de paramètres, nous citons : le type du procédés logiciel, Les objectifs à atteindre, les priorités de développement, l'environnement de développement et de la méthode de développement.

4- Différents langages de modélisation et moteurs d'exécution sont utilisés pour les procédés logiciels ce qui complique l'agencement des fragments procédés logiciel de différents langages.

Objectif du travail :

Plusieurs approches ont été proposées pour la réutilisation des composants procédés logiciels les plus importantes sont : l'environnement RHODE, PYNODE, OPC, APPEL.
L'objectif de notre travail est d'appliquer la réutilisation au procédé de logiciel, en exploitant le niveau 2,3,4 de l'architecture a quatre niveaux de l'OMG **[Jam 05]** qui sont des métas procédés.
Les procédés logiciels réutilisés sont des procédés représentés en UML plus précisément le diagramme des classes. Le procédé logiciel doit être centré activité et manipule le concept produit. Ces deux concepts sont des éléments de base présents dans la plupart des procédés logiciels. Nos objectifs principaux sont :

1- La caractérisation du composant procédé qui est une étape primordiale et qui déterminera la qualité de la réutilisation, il sera ainsi facile d'identifier, de stocker de manière bien clair le composant recherché.

2- Stocker les composants procédés dans une base de données.

3- Déterminer l'algorithme de recherche et d'assemblage des composants pour qu'à la fin nous ayons un procédé logiciel rassemblé à partir des composants déjà existants.

4- Exploiter le concept agent pour l'identification des composants, le stockage, la recherche et le montage des composants procédés logiciels.

Méthodologie suivie :

Pour attendre nos objectifs, nous commençons par une présentation de l'état de l'art des procédés logiciels, des approches et environnements de réutilisation et des systèmes multi agents

Ensuite nous présentons les outils utilisés à la modélisation de ce genre de problème et nous proposant une approche capable à réutiliser des procédés logiciels.

Finalement nous présentant la description détaillée de l'approche proposé a l'aide de langage U ML guidé par le processus RUP et sa mise e œuvre en utilisant le langage C ++.

Présentation de mémoire :

Notre mémoire est composé de quatre chapitres :

Chapitre 1: Dans ce chapitre, nous présentons une introduction générale, un problématique et les objectifs.

Chapitre 2 : Ce chapitre est décomposé en trois parties :

- Dans la première partie nous présentons les notions de base concernant les procédés logiciels.
- Dans la deuxième partie nous présentons deux approches de réutilisation en insistant sur l'environnement RHODES.
- Dans la troisième partie nous présentons la technologie des agents.

Chapitre 3 : Ce chapitre est décomposé en deux parties :

- Dans la première partie nous présentons une généralité sur le langage de modélisation UML et le processus de développement RUP et les modèles de réutilisation.
- Dans la deuxième partie nous présentons les déférentes étapes de conception de notre système.

Chapitre 4 : Dans ce chapitre, nous proposons une implémentation de l'approche proposée on utilisant le langage C++Builder.

Chapitre II

Partie I

Notions De Base

I. DEFINITION DES PROCEDES:

Les procédés sont définis comme étant une suite d'étapes réalisées dans un but donné :
"A sequence of steps performed for a given purpose" **[lee 90].**

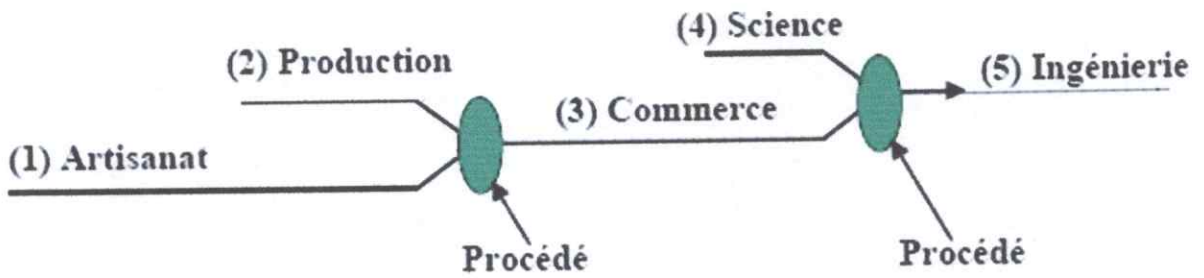


Figure II.01 : L'apparition des procédés dans le monde **[Sha 90]**

« Les procédés sont apparus dans le monde de l'informatique dans les années 80, lorsque les informaticiens se sont aperçus de la répétitivité de certaines suites de tâches, et ont cherché à les formaliser et/ou les automatiser. Les informaticiens se sont intéressés en premier à la formalisation des tâches du développement logiciel, ce qu'ils ont appelé le "Procédé Logiciel". L'utilisation des procédés s'est ensuite étendu à d'autres domaines. Elle a permis de faire évoluer ces domaines, en introduisant cette nouvelle technique » **[Jam 05].**

La Figure -2- représente une liste des domaines utilisant les procédés, et les dates du début de l'utilisation de procédé pour chacun d'entre eux.

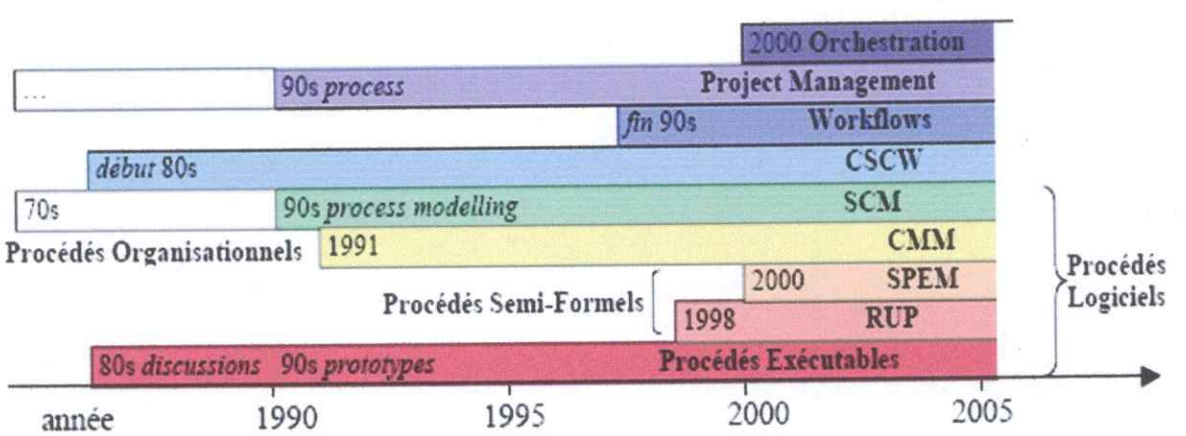


Figure II.02 : L'histoire des procédés **[Jam 05]**

« Par procédé on entend l'ensemble des activités faisant intervenir des équipes de personnes (souvent nombreuses), des outils et des techniques pour assurer le développement et la maintenance de produits ou de services. » **[Gal 00]**

«L'utilisation de procédés est le meilleur moyen pour réaliser un travail de manière bien organisée, que ce soit pour faire du développement logiciel, pour automatiser les tâches à exécuter dans un système logiciel, dans le domaine de la gestion, etc. en effet, le procédé offre de bons moyens pour modéliser et de décrire ce travail. » **[Jam 05]**

II. DEFINITION DES PROCEDES LOGICIELS:

On peut définir les procédés logiciels comme étant une suite d'étapes réalisées dans un but donné qui servent à gérer et assister le développement logiciel.

Les procédés logiciels sont exprimés sous forme de modèle de procédé, décrit dans un Langage de Modélisation de Procédé (PML : Process Modelling Language).

«On appelle procédé logiciel l'ensemble des activités, des équipes, des outils et des techniques dont le but est d'assurer le développement et la maintenance de système logiciel. » **[Ami 99]**

«Les procédés logiciels sont relativement instables. (Développement d'activités peut être changé ou raffiné quand le procédé est en cours d'exécution.) **[Gal 00]**

III. EXEMPLE DE PROCEDE LOGICIEL:

Les aspects technologiques étudiés se focalisent sur l'élaboration d'environnements destinés à supporter les modèles développés. Un exemple est le modèle en cascades **[Jcj 93]** de cycles de vie du logiciel, la figure 3 illustre un procédé suivi pour le développement d'un logiciel. Les flèches arrondies indiquent le flux d'exécution des différentes activités dans le développement du logiciel. Par exemple quand l'activité *Analyser le problème* soit fini, l'activité *Concevoir la solution logicielle* sera initiée. Si dans le développement de cette activité on trouve des erreurs, l'activité *analyser le problème* sera appelé pour réaliser les corrections pertinentes au modèle. Les flèches droites indiquent que tous les produits fournis par les activités seront validés, dans l'activité *Valider les modèles et l'implémentation*.

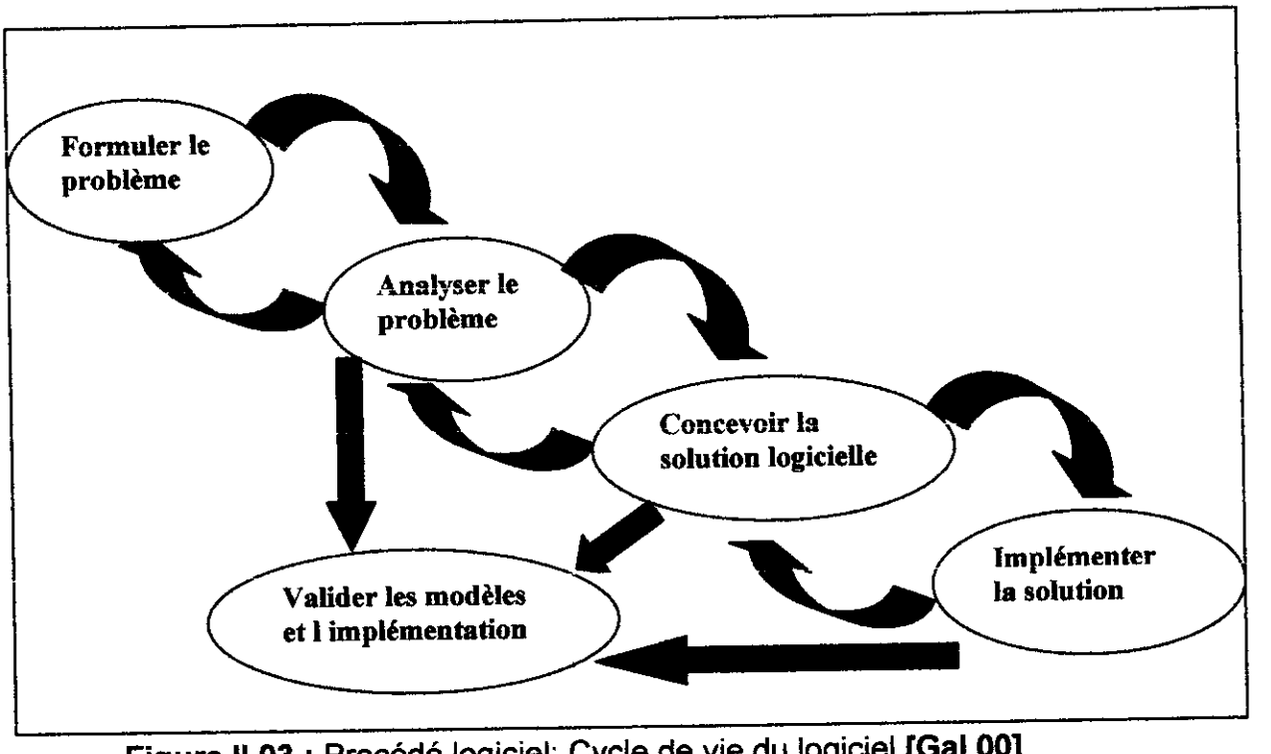


Figure II.03 : Procédé logiciel: Cycle de vie du logiciel **[Gal 00]**

IV. CLASSIFICATION DES PROCEDES LOGICIELS :

Les "Procédés Logiciels" (Software Process) servent à gérer et assister le développement logiciel. Il existe plusieurs sous-classes de Procédés Logiciels :

- **Les procédés exécutables :**

Le Procédé Logiciel Exécutable définit la manière dont le développement logiciel est organisé, géré, mesuré, assisté, et amélioré (indépendamment du type de support technologique choisi pour le développement) **[Der 99].**
Il existe trois catégories de procédés logiciels exécutables : les Procédés centrés Activités, centrés Produits, et centrés Rôles, cela suivant le concept central sur lequel l'accent a été mis.
A la fin des années 90, les entreprises ont commencé à formaliser leur fonctionnement, en mettant en place des workflows. Les workflows sont des procédés exécutables, centré Activités, souvent restreints à des enchaînements simples d'activités.

- **Les procédés semi formels :**

Les procédés semi-formels sont des procédés logiciels non exécutables, qui ont pour but de décrire la façon dont le logiciel va être développé. Ils recommandent fortement l'utilisation de procédé pour formaliser et modéliser le développement logiciel.

Les procédés semi-formels ont beaucoup apporté au domaine de la modélisation du développement logiciel. Ils ont eu beaucoup de succès en entreprise, surtout RUP (voir chapitre -3-), qui présente les meilleures pratiques de développement. Ceci indique le grand intérêt qu'ont ces meilleures pratiques chez les industriels.

L'utilisation de procédés est le meilleur moyen pour réaliser un travail de manière bien organisée, que ce soit pour faire du développement logiciel, pour automatiser les tâches à exécuter dans un système logiciel, dans le domaine de la gestion, etc. en effet, le procédé offre de bons moyens pour modéliser et de décrire le travail.

- **Les procédés organisationnels :**

CMM (Capability Maturity Model) et CMMI (Capability Maturity Model Integration) considèrent que le développement logiciel ne peut être mature et efficace que s'il est établi sur une infrastructure de procédé fondée sur des pratiques efficaces d'ingénierie logiciel et de gestion. Ils définissent alors les niveaux de maturité, et décrivent les éléments clés d'un procédé logiciel efficace. CMM et CMMI ont eu un énorme succès auprès des industriels, ils ont réussi à faire entrer le procédé dans l'industrie, et à ancrer dans les esprits l'importance de se baser sur un procédé lors du développement logiciel.

- **Les procédés de gestion de configuration :**

La gestion de configuration est le contrôle de l'évolution des systèmes complexes. Un système complexe est un système ayant :

- Un grand nombre de composants, et de versions de composants (des milliers).
- Un grand nombre de personnes impliquées (des dizaines ou des centaines).
- Des contraintes strictes de temps (cycles de quelques mois).
- Une longue durée de vie (jusqu'à 10 ou 20 ans).

L'objectif de la gestion de configuration est de contrôler l'évolution du système, et d'aider à satisfaire les contraintes de délais et de qualité.

La gestion de configuration est apparue au début des années 80. A cette époque, elle s'intéressait à la programmation globale (*programming in the large*). Pendant les années 90, la gestion de configuration s'est concentrée sur la programmation coopérative (*programming in the many*), et à la fin des années 90, elle s'est concentrée sur la programmation distribuée (*programming in the wide*).

- **Les procédés logiciels récents : SPEM**

Nous présentons, pour les procédés logiciels récents, SPEM (Software Process Engineering Meta model) un des procédés logiciels récents **[Spm 05].**

SPEM est une proposition de l'OMG (Object Managemant Group) qui définit un formalisme dédié à la description du processus de développement logiciel. Les outils basés sur SPEM sont des outils de rédaction et de personnalisation de processus.

SPEM est à la fois un méta modèle MOF (Meta-Object Facility) et profil UML, des correspondances ont été établies entre ces deux formalismes.

SPEM contient plusieurs éléments de structure du procédé, permettant de construire une description de procédé. Ces éléments sont assez similaires aux éléments du méta modèle de Procédé Logiciel **[Jam05].**

SPEM est un formalisme dédié à la description du processus de développement logiciel. Il ne s'adresse pas à la planification ni à l'exécution d'un projet.

V. MODELE ET META MODELE:

Afin d'organiser et de structurer les modèles, l'OMG a défini une architecture appelée : "Architecture à quatre niveaux". Ces quatre niveaux sont (Figure 4) :

Le niveau M0 : C'est le niveau des données réelles. Il est composé des informations que l'on souhaite modéliser. Ce niveau est souvent considéré comme étant le monde réel.

Le niveau M1 : Lorsque l'on veut décrire les informations appartenant à M0, le MDA considère que l'on fait un modèle appartenant au niveau M1. Un modèle de procédé appartient donc au niveau M1. De même, tout modèle de niveau M1 est exprimé dans un langage dont la définition est fournie explicitement au niveau M2.

Le niveau M2 : Ce niveau est composé de langages de définition des modèles, appelés aussi métas modèles. Le méta modèle de procédé appartient au niveau M2, et définit la structure interne des modèles de procédé.

Le niveau M3 : Ce niveau contient le MOF (Meta-Object Facility), le langage unique de définition des métas modèles, aussi appelé le méta méta modèle. Le MOF définit la structure de tous les métas modèles qui se trouvent au niveau M2.

Différentes utilisations de ces modèles ; Des modèles UML et d'autres modèles Le méta modèle UML et autres métas modèles Le MOF M0 M1 M3 M2**méta-méta model** "le monde réel"(procédé).

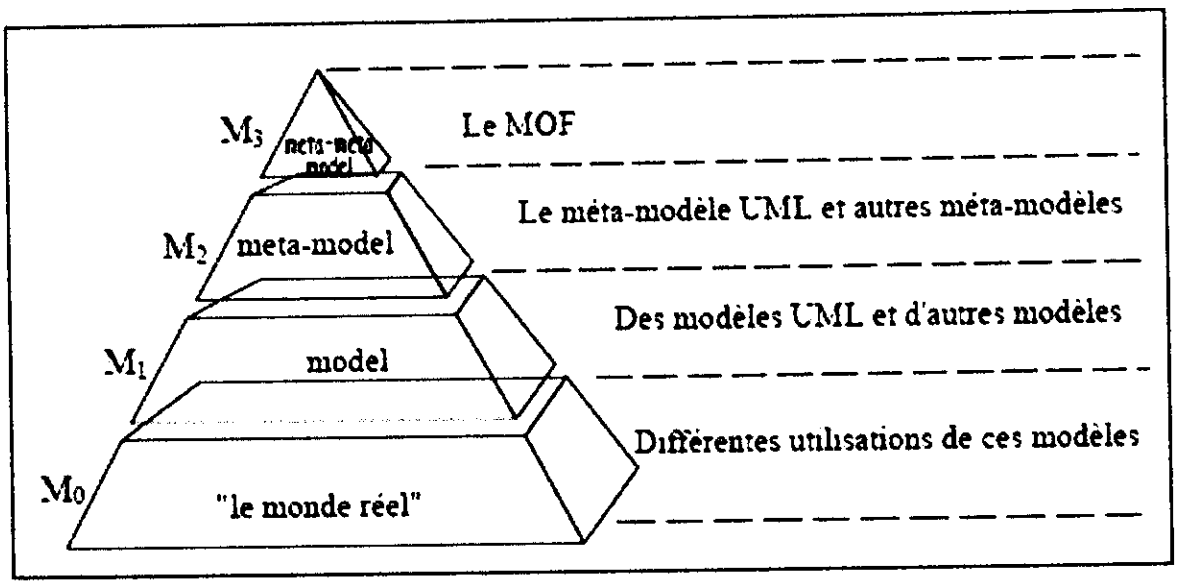


Figure II.4 : Architecture à quatre niveaux de l'OMG **[Jam 05].**

VI. MODELE DE PROCEDE:

Un modèle de procédé est une représentation des activités du monde réel, il est développé, analysé, raffiné, transformé, et/ou exécuté conformément au méta modèle du procédé **[Der 99].**

De même il est écrit dans le méta modèle auquel il se conforme (en fonction de l'aspect étudié par le modèle). **[Mer 05]**

Un modèle exécutable de procédé de développement est utilisé au cours d'un projet pour contrôler et assister les développeurs. Il est donc important que ce modèle de procédé ne contienne pas d'erreurs

VII. META MODELE DE PROCEDE:

VII.1- Définition :

Un méta modèle de procédé définit la structure interne des modèles de procédé.

L'intérêt essentiel d'un méta modèle et de faciliter la séparation des préoccupations. **[Mer 05]**

VII.2- Eléments du méta modèle de procédé logiciel :

Les éléments du méta modèle de procédés logiciels se décomposent en éléments basiques et éléments secondaires. Les éléments basiques sont les suivants :

-Activité

Une activité est une tâche pendant laquelle des opérations sur le logiciel à développer sont accomplies. Elle est souvent associée à une ou des personnes responsables de cette activité, et à des outils de production.

Une activité peut être décomposée en d'autres activités, formant ainsi plusieurs niveaux d'abstraction. L'activité peut être concurrente et coopérative, déterministe ou non déterministe.

Le modèle de procédé logiciel comprend des activités de développement du logiciel et de maintenance, des activités de gestion de projet et d'assurance qualité, et des activités de méta procédé. Les activités peuvent avoir différents niveaux de granularité, et sont généralement associés à des rôles pouvant être entrepris par certaines catégories d'utilisateurs et/ou d'outils. Les produits sont les données d'entrée et de sortie des activités. **[Der 99]**

-Produit :

Un produit est souvent un artefact (persistant et versionné), pouvant être simple ou composite, formant les données d'entrée et de sortie des activités. Les produits peuvent être des parties du logiciel à développer, et des documents associés (documents de conception, documentation de l'utilisateur, données de test).

-Rôle

Un rôle décrit les droits et les responsabilités d'un humain. Un humain joue un rôle dans une activité (ou plusieurs rôles dans des activités différentes).

-Humain

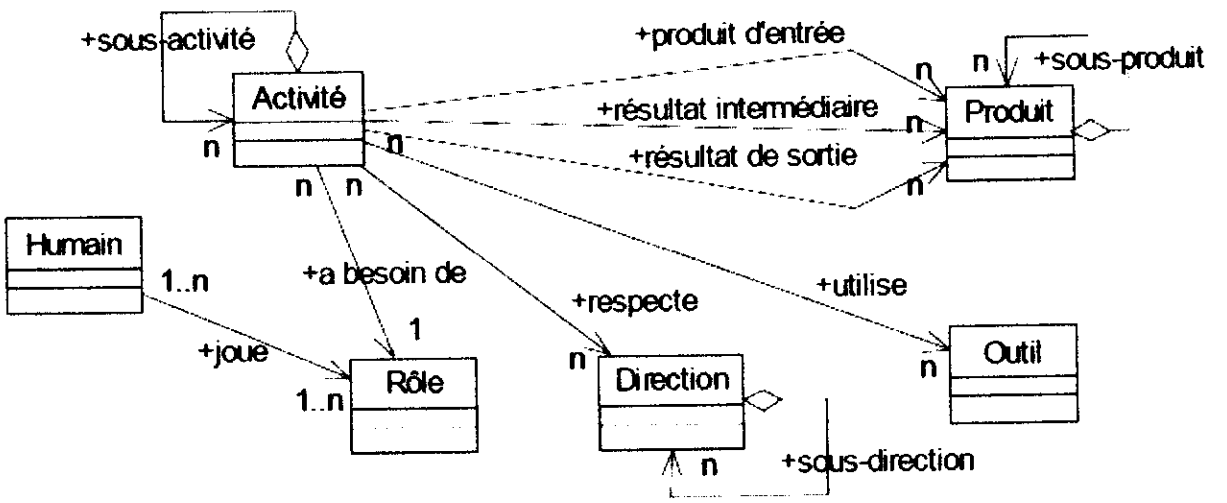
Les humains sont des agents (ou des développeurs) du modèle de procédé logiciel, que l'on peut organiser dans des groupes. Un rôle est attribué aux humains ayant les compétences et les responsabilités nécessaires pour jouer ce rôle. Un humain peu avoir plusieurs rôles, il peut également être membre de plusieurs groupes (ces groupes peuvent aussi être imbriqués).

-Outil

Les outils sont des systèmes qui assistent la production de logiciel. Il existe deux sortes d'outils : les outils interactifs (éditeurs textuels, outils graphiques comme CASE...), et les outils simplement exécutables sans interaction (compilateurs, analyseur grammatical...).

-Support d'évolution (Directions)

Le support d'évolution aide à gérer l'évolution du procédé logiciel (sa modification), à travers les Directions (politiques, règles, et procédures). L'évolution du procédé logiciel est un besoin essentiel, à cause de sa nature orientée humain. Une façon de supporter l'évolution est d'utiliser un méta modèle de procédé offrant une assistance du point de vue conceptuel, pour les changements statiques ou dynamiques du modèle de procédé. Une assistance technique serait également importante pour supporter l'évolution.



Figurell.05: Les éléments basiques de méta modèle de procédé logiciel **[Der 99].**

D'après **[Der 99]** les éléments secondaires sont : Projet/organisation, Espace de travail, Vue utilisateur, Modèle de coopération, Modèle de versionnement/transaction, et Modèle de qualité/performance.

VIII- META PROCEDE:

Selon SPEM qui est un méta procédé basé composant procédé :
Le méta procédé permet de passer progressivement d'un procédé informel à un procédé décrit sous forme de composants. **[Spm 05].**

« Le méta procédé prend en compte à la fois la gestion globale de procédé et la description de composants de procédés de granularité fine. Ce méta procédé supporte l'évolution et la réutilisation d'entités élaborées, et il est suffisamment formel pour permettre une assistance efficace aux concepteurs de procédés. » **[Ber 03]**

IX- DEFINITION DE COMPOSANT :

« Un composant est un fragment de logiciel assez petit pour qu'on puisse le créer et le maintenir, et assez grand pour qu'on puisse l'installer et en assurer le support. De plus, il est doté d'interfaces standard pour pouvoir intéropérer ». **[Ohe 99]**

Un composant du logiciel est une unité de composition avec des interfaces,il peut être déployé indépendamment et peut être soumis à composition par la programmation des modules qui sont conçus pour intéropérer avec d'autres modules. Il peut être écrit par programmeurs différents qui utilisent des environnements du développement différents. Les composants peuvent être courus dans des machines autonomes **[Spm 05].**

X- DEFINITION D'UN COMPOSANT PROCEDE LOGICIEL:

« Un composant procédé est constitué d'un groupe significatif d'activités. Le composant procédé contient tous les rôles et produits qui sont liées à ces procédés et activités ». **[L.D 97]**

On peut dire qu'un composant procédé est une partie d'instance d'un procédé, dérivée d'un modèle stocké dans une base de données. Le composant encapsule sa représentation et fournit une interface simple pour établir la communication avec l'extérieur.

Il correspond à une activité ou à un sous procédé qui contribue à la réalisation d'un but dans le procédé. Il vise à générer ou à modifier un ensemble de produits qui correspond à des objets tels que des documents, codes ou services.

Un composant de procédé est défini informellement comme : une unité cohérente et réutilisable du procédé **[Ber 03].**

XI. L'ENVERONEMENT SUPORTANT LES PROCEDES BASES COMPOSANT:

« La tendance actuelle dans les systèmes supportant des procédés est d'offrir un environnement *coopératif* dans lequel le support se fait à travers un ensemble de *composants*. Chaque composant prend en charge la gestion d'un aspect particulier du procédé.

Les récents travaux de recherche proposent une approche par composant pour le support de procédés. Les avantages attendus sont:

_ L'obtention des environnements coopératifs et ouverts.

_ L'Interopérabilité entre composants hétérogènes.
_ Les Dépendances entre procédés sont délimitées et par conséquent mieux reconnues et gérées.
_ La réutilisation des composants procédés.»**[Gal 00]**

XII. LES PATRONS

XII.1 -Définition des patrons :

Un patron de conception est un savoir-faire formalisé : « Chaque patron décrit à la fois un problème qui se produit très fréquemment dans votre environnement et l'architecture de la solution à ce problème de telle façon que vous puissiez utiliser cette solution des millions de fois sans jamais l'adapter deux fois de la même manière ». **[Ctr 02]**

Dans le domaine de l'ingénierie des systèmes d'information les patrons sont des composants réutilisables :

- alliant problème et solution,
- offrant des solutions conceptuelles (spécifications réutilisables),
- correspondant à des composants de type « boite blanche », (le système).
- dédiés à la réutilisation de microarchitectures (4 à 5 classes pour les patrons orientés objets, 4 à 5 tâches pour les patrons processus, etc.).

XII.2- Classification des patrons:

Parmi les différents critères permettant de comparer les modèles de composants, trois d'entre eux sont particulièrement intéressants pour classifier les patrons :

- **le type de connaissance :** Il peut s'agir de capitaliser des spécifications ou des implantations de produit - un résultat à atteindre - ou de capitaliser des spécifications ou des implantations de processus - une démarche à suivre pour atteindre le résultat.

- **la couverture :** Il peut s'agir de patrons généraux, domaine ou entreprise.

- **la portée :** La portée est évaluée en fonction de l'étape d'ingénierie (analyse, conception, implantation) à laquelle le composant s'adresse.

- Les patrons procédés :

Un patron procédé est un procédé (ou partie de procédé) qui peut être réutilisé dans des contexte différents pour résoudre des problèmes récurrents. **[Ccd 02].**

Le patron peut être réutilise dans tout les processus qui incluent des activités de son type.

XIII- DEFINITION D'INTEROPERABILITE:

Deux ou plusieurs ressources sont interopérables si elles peuvent interagir pour exécuter ensemble une tâche. L'interopérabilité implique la distribution d'informations, la communication et la compréhension mutuelle entre les différents domaines d'exécution.

L'hétérogénéité des composants rend complexe l'interopérabilité. Cette hétérogénéité peut être due à la diversité des solutions proposées pour couvrir les différents aspects du procédé. En effet, pour chaque aspect dans le procédé, on peut envisager des solutions conceptuelles et technologiques différentes. Ceci concerne les concepts utilisés, les formalismes proposés et les plates formes d'exécution, etc.

On peut distinguer trois niveaux d'hétérogénéité :

- Sémantique : Chaque composant peut offrir des concepts différents pour gérer l'aspect du procédé auquel il est dédié.

- Syntaxique : Les composants peuvent offrir un formalisme différent pour décrire l'aspect de procédé qui leur correspond. Le choix d'un formalisme par rapport à un autre peut avoir un impact considérable sur l'amélioration de la compréhension des modèles et de la capture du domaine du problème.

-Des plates formes d'exécution : En considérant que l'exécution du procédé est répartie sur un ensemble de composants autonomes, chaque composant peut faire appel à des plates formes matérielles différentes. Le problème consiste ici à trouver les mécanismes appropriés qui permettent aux composants d'interagir, d'échanger des données et les contrôler.

XIIII. NOTION DE REUTILISATION

La complexité croissante des systèmes d'information et leur évolution de plus en plus rapide ont motivé un intérêt accru pour les modèles et méthodes de réutilisation. **[Ctr 02]**

La notion de la réutilisation de logiciels a été durant ces trente dernières années comme un moyen de réduire considérablement le coût de développement de logiciels. Le besoin de réutilisation de logiciels est devenu urgent lorsque la

taille et la complexité des logiciels a commencé à s'intensifier et lorsque le cycle de développement du produit a été compressé.

En effet, les pressions compétitives ont obligé les vendeurs à réduire les cycles de développement de produits de 18-24 mois, une décennie auparavant, à 3-9 mois aujourd'hui **[Mer 05].**

La réutilisation de logiciels a été citée comme le moyen le plus efficace pour améliorer la productivité dans les projets de développement de logiciels **[Pau 99],**

La réutilisation de logiciels dans un projet de développement est généralement supposée augmenter la productivité, améliorer la fiabilité du produit, et baisser les coûts globaux. En effet, plusieurs projets de développement de logiciels ont rapporté l'augmentation de productivité jusqu'à 50% avec de grands degrés de réutilisation de logiciels. **[spm 05]**

La réutilisation du logiciel est encore une nouvelle discipline; beaucoup de gens comprennent mal la notion de réutilisation du logiciel comme la réutilisation de code source. Récemment la réutilisation de code source et du dessin est devenue populaire avec bibliothèques de classes (orienté objet), les structures de l'application, et le modèle de dessin. Les composants du logiciel fournissent un véhicule pour la réutilisation systématique et en projet.

La réutilisation d'un logiciel systématique et la réutilisation d'influence des composant presque le processus de programmeur entier (indépendant de ce qu'un composant est). Les modèle processus de logiciel on été développés pour fournir le conseil dans la création du système logiciel de qualité par les équipes à coûts prévisibles. Les modèles originaux ont été basés sur la conception que les systèmes sont construits à partir du zéro d'après les exigences stables. Avec une réutilisation croissante de logiciels, les nouveaux modèles pour programmeur émergent, ces modèles-ci sont basés sur la réutilisation systématique de composants.

« Le logiciel en voie de développement avec réutilisation exige l'organisation pour la réutilisation, développement pour la réutilisation et avec la réutilisation et fournir la documentation pour la réutilisation, la documentation adéquate est une nécessité pour la réutilisation systématique de composants. Si nous continuions à négliger la documentation ne sera pas capable d'augmenter la productivité a travers la réutilisation de composant » **[Ref 01]**

« La réutilisation du composant de logiciel suggère aussi la réutilisation de documentation ».**[Ref 02]**

XV- EXEMPLES DE REUTILISATION :

1-Réutilisabilité logicielle Java :

Les programmeurs Java se concentrent sur la conception de nouvelles classes et la réutilisation de classes existantes. Un grand nombre de bibliothèques de classe existent et sont développées mondialement. Les logiciels, se construisent à partir des composants existants, qui sont bien définis, testés adéquatement, suffisamment documentés, portables et largement disponibles.

Cette réutilisation logicielle accélère le développement d'applications puissantes et de haute qualité. Le développement rapide d'applications, RAD, suscite aujourd'hui un grand intérêt. **[Cpj 05]**

2- Réutilisation des composants procédés : Pynode

Pour cette approche de réutilisation, qu'est basée sur la réutilisation des composants logiciels tenant compte de l'aspect statique des composants. (Pour plus de détail, voir la partie 2).

3- Réutilisation des patrons procédés : Rhodes

Rhodes est un environnement basé sur l'approche SPEM, pour réutiliser les procédés logiciels ils ont étudié les deux aspects de procédé logiciel : l'aspect statique et l'aspect dynamique (Pour plus de détail, voir la partie 2).

CONCLUSION

Dans cette section de ce premier chapitre nous avons présenté en détail les différentes notions de base qu'ont relation avec ce problème.

Dans la deuxième partie de ce chapitre nous entamerons en détaille les deux notions, Pynode comme approche, et Rhodes comme environnement.

Chapitre II

Partie II

Approches Et Environnement

INTRODUCTION

La maîtrise des processus de développement du logiciel est une tâche complexe qui doit prendre en considération différents produits, utilisateurs, outils et contraintes. La communauté du génie logiciel a fait beaucoup d'efforts pour décrire et formaliser les procédés logiciels, en proposant des techniques similaires à celles qui sont utilisées pour les logiciels. Ces propositions englobent des notations, des langages de description des procédés (LDP), des outils et des environnements pour automatiser l'exécution des procédés.

Pour améliorer la productivité et la qualité des développements logiciels, plusieurs méthodes de développement et ateliers correspondants ont été proposés dans la dernière décennie.

Cependant, dans ces environnements, les procédés sont généralement informellement décrits informellement ce qui les rend peu évolutifs et difficiles à réutiliser. Or, les processus logiciels sont intrinsèquement évolutifs: leurs descriptions doivent pouvoir varier pour tenir compte de l'évolution des techniques et des méthodes de développement.

C'est ainsi que les Ateliers de Génie Logiciels centrés procédés (AGL-P) offrent une alternative intéressante aux ateliers classiquement dédiés à une méthode ou à une catégorie de méthode. Les AGL-P permettant d'une part de décrire formellement un procédé de développement de logiciel avec un LDP,et d'autre par, l'exécuter / interpréter le procédé ainsi décrit. Ce pondant, le travaille de description /formalisation de procédé est lui-même un processus qui est souvent lourd et répétitif. **[Bel 96].**

Par ailleurs, il n'y a pas de standard actuellement reconnu pour la description des procédés.

I.L'ENVIRONNEMENT PYNODE

I.1 Approche pynode :

C'est une approche à base de composant pour la modélisation des procédés, elle a pour but d'organiser des procédés de développements dans un objectif de réutilisation.

I.2. Définition de Pynode :

La notion de composant procédé réutilisable a été proposée pour les procédés de l'environnement **Pynode,** chaque composant de procédé ce compose de deux parties :
-une partie interface.
-une partie implantation.

Des ports sont utilisés dans la partie interface du composant pour l'assemblage des composants.

L'implantation des composants est réalisée dans un langage de description de procédés. L'environnement Pynode fournit la possibilité de composition dynamique (à l'exécution) de composant de procédés par l'intermédiaire de vue d'exécution et de vue d'observation.**[Bre 02]**

K.Gary et al. Ont proposé la notion de composant ouverts de procédé **OPC** (Open Process Components). Pour modéliser des procédés de développement de logiciel. Une architecture à trois niveaux est utilisée : le framework, la présentation, et les composants.

-Le framework se compose des entités génériques du procédé. **[Omg 02]**

-La présentation est un raffinement du framework elle dépend de la sémantique des langages de description de procédés. Elle comprend des réseaux de pétri, des présentations à base d'événement, etc.

-Le niveau des composant (ou domaine d'application) dépend d'un domaine spécifique d'application .Un composant de procédé est défini comme *une entité encapsulée de procédés.*

-La composition d'un composant est réalisée par des lois définies dans un méta modèle et par l'héritage de composant de la couche de présentation.

-Cette approche s'intéresse beaucoup à l'évolution dynamique (la composition dynamique de composant de procédé). Ils s'intéressent peu à l'évolution statique de composants de procédé et à l'assistance aux concepteurs de procédés.

- D'autre part, contrairement à la plupart des approches, nous considérons que décrire des procédés de développement avec une granularité fine est souhaitable pour fournir une assistance précise aux développeurs et pour contrôler sa progression dans le développement en particulier. **[Ref 03]**

II. L'ENVIRONNEMENT RHODES

L'environnement RHODES sera étudié en détail car notre stratégie de réutilisation est plus au moins similaire.

II.1. Définition:

L'environnement RHODES est un atelier de génie logiciel centré procédé (AGL-P) qui a été développé au laboratoire de L'ENSEEIHT-IRIT . Pour rendre la réutilisation de processus efficace,ils ont appliqué à RHODES l'approche basée sur les composant et nous avons introduit la notion de composant de procédé. **[Cou 01]**

Parce que"les processus logiciels sont aussi des logiciels"**[Ost 97]** la technique pour développer des logiciels peut être appliquée à la modélisation des procédés de développement.

RHODES s'appuie sur une description formelle du procédé qu'il exécute pour contrôler le développement et assister les développeurs.

II.2. Architecteur RHODES:

Dans RHODES, un composant de procédé est défini informellement comme "une unité cohérente et réutilisable du procédé" en considérant deux niveaux de composant de procédé: des composant élémentaires et des composants complexes.

-Les composants élémentaires de procédé sont les entités de base (activité, produit, rôle, et stratégie...) pour décrire statiquement les procédé de développement. **[Cre 97]**

-Les composants complexes sont utilise pour gérer et réutiliser plus effacement les composants élémentaires de procédé. Chaque composant complexe possède une spécification et peut posséder plusieurs implantations. **[Cou 01]**

L' Architecteur RHODES est composée de cinq constituants principaux :

-le méta modelé de composant
-la base de composent de procédé.
-le noyau d'exécution.
-Des outils pour les concepteurs de procédé (éditeur, compilateur...).
-des instances de RHODES.

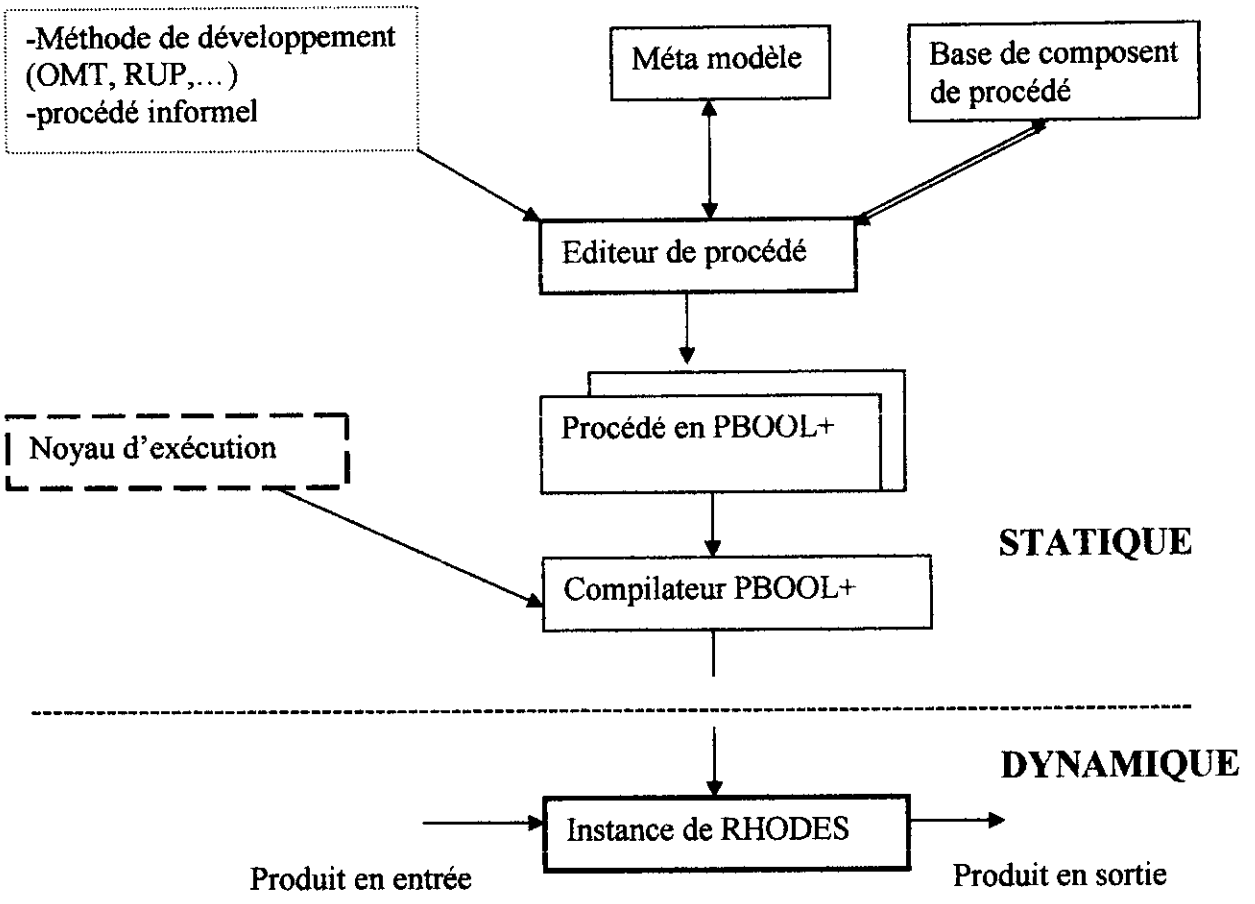


Figure II.06- : Architecteur de l'environnement RHODES **[Ber 03]**

Les composent de procédé sont les instances du méta modèle. Ils sont décrits formellement dans PBOOL+, une extension du langage cible PBOOL .nous obtenons une instance de RHODES pour mettre en œuvre un développement conformément au procédé décrit. **[Cho 00]**

II.3. Principe de méta procédé RHODES :

Pour définir et réutiliser les composant de procédé, ce méta procédé permet de passer progressivement d'un procédé informel à un procédé décrit sous forme de composant en PBOOL+.Issu de cycle de vie de logiciel et des méthode de développement, le méta procédé prend en compte à la fois la gestion globale de procédé et la description de composants de procédé de granularité finie. Ce

méta procédé supporte l'évolution et la réutilisation d'entités élaborées. Et il est suffisamment formel pour permettre une assistance efficace aux composant de procédés.

II.4. Vue structurelle du méta procédé :

Le méta procédé RHODES est un procédé comprenant trois cycle itératifs :
-un cycle général de modélisation.
-un cycle de formalisation d'un composant de procédé.
-un cycle de recherche et de réutilisation/adaptation d'un composant de procédé.
Le cycle général est utilisé pour définir un procédé. Les deux autres cycles sont les cycles de raffinage.

Remarque :

Chaque cycle du méta procédé est intercalé itérativement dans une étape ou une activité du cycle précédant. Le cycle général peut être instancié lui-même dans sa décomposition.

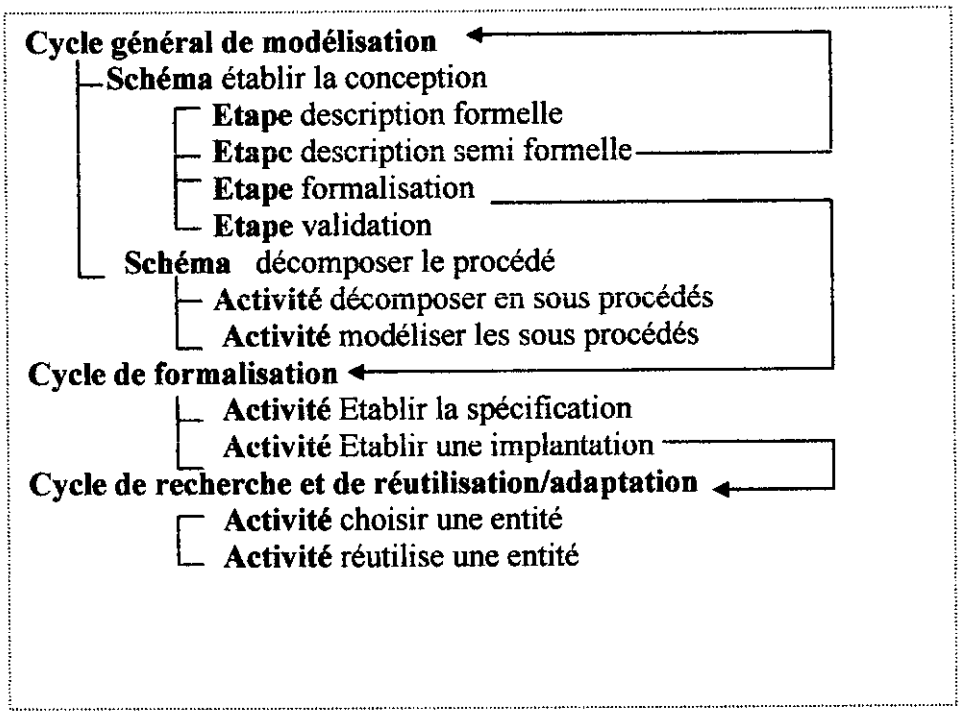


Figure II.07- Les cycles du méta procédé RHODES **[Ber 03]**

II.5. Etape du cycle générale de modélisation :

Dans ce cycle général, selon sa complexité, on peut décomposer le procédé en sous procédés ou établir sa conception en effectuer les quatre étapes.

-dans le cas d'une décomposition en sous procédés, le cycle général est utilisé récursivement pour concevoir chaque sous procédé. (Figure II.08).

-les quatre étapes du cycle général de modélisation se composent d'activités de définition de procédé (figure II.08).Ces activité sont formalisées dans le langage PBOOL+.

II.6. La description informelle :

Elle a pour but de décrire intuitivement le procédé en cours de conception .Cette étape est réalisée par les experts du domaine grâce à leurs connaissances méthodologiques, au savoir faire acquis lors des développement précédents et aux procédés informel existants. Les experts méthodologiques définissent une description informelle du procédé. Ils mettant en évidence dans cette description les produits manipulés tout long du développement, des activités du développement, et des rôle des agent participant au développement.

L'étape est décomposée en deux activités identifier les composants participant au procédé et décrire informellement chacun des composants identifiés. La premier est une activité itérative qui permet, au moyen de quatre sous activités, d'identifier et sauvegarder les composants. La deuxième a pour but de définir globalement les composant identifiés en restant à un niveau informel.

II.7. La description semi formelle :

Elle établit une conception provisoire du procédé de développement. L'étape peut être considérée comme une traduction informelle en une description semi formelle du procédé. A partir de la description informelle précédente du procédé, les concepteurs de procédés décrivent les entités du procédé en utilisant un langage de modélisation semblable à UML pour produire des diagrammes. La racine de l'étape, « *définir les diagrammes* », qui est une activité de répétition interactive de trois sous activités.

II.8. La formalisation :

Ella a pour but de formaliser le procédé à laide d'un langage de description de procédés (LDP).Les concepteurs de procédés utilisent la description semi formelle précédente pour formaliser le procédé en cours de conception .Le résultat de cette étape est un ensemble de composants réutilisables. Ces composants sont liés les uns aux autres par des liaisons dont la sémantique est définie par LDP. L'activité « formalise les composant », est racine de cette étape. Elle est une activité de répétition contrainte. Chacun des composant des deux étapes précédentes est manipulé par la sous activité « formalise un composant » qui est l'instanciation de l'activité racine de cycle « formalisation d'un composant de procédé ».

II.9. La validation d'un procédé :

Elle a pour but de vérifier si la formalisation en LDP du procédé est valide. Le procédé de validation est compose de deux étapes :

-étape de validation statique (avant l'exécution de procédé), le réviseur et le concepteur utilisent l'activité « valide statiquement un composant » itérativement pour valider chaque composant qui a été formalisé. Cette activité est décomposé en trois sous activité « relire un composant », « vérifie la cohérence statique et annoter », et « évaluer ». La troisième activité contient deux schémas de réalisation correspondant à l'acceptation ou le refus du composant considéré.

-étape de validation dynamique (en cours de exécution de procédé).

II.10. Cycle de formalisation d'un composant de procédé :

Le but du cycle est de définir formellement, chaque composant du procédé, en cours de conception. L'activité racine est décomposée en deux sous activités « *établir la spécification du composant* » et « *établir une implantation du composant* »qui élaborent un composant grâce à la description semi formelle de procédés (Figure II.08).Les différents schémas de réalisation permettent aux concepteurs d'élaborer une entité que ce soit à partir de zéro ou bien par réutilisation/adaptation des élément de la base de composants. Par exemple il y trois manière de définir l'implantation du composant : création d'une nouvelle implantation, recherche d'une implantation existante pour la réutiliser/adapter, ou la décomposition de l'implantation en sous composants. Dans le cas de la décomposition, chaque sous composant est formalise en appliquant récursivement le cycle. Dans le cas de la recherche, le « cycle de recherche et de réutilisation/adaptation » est utilisé.

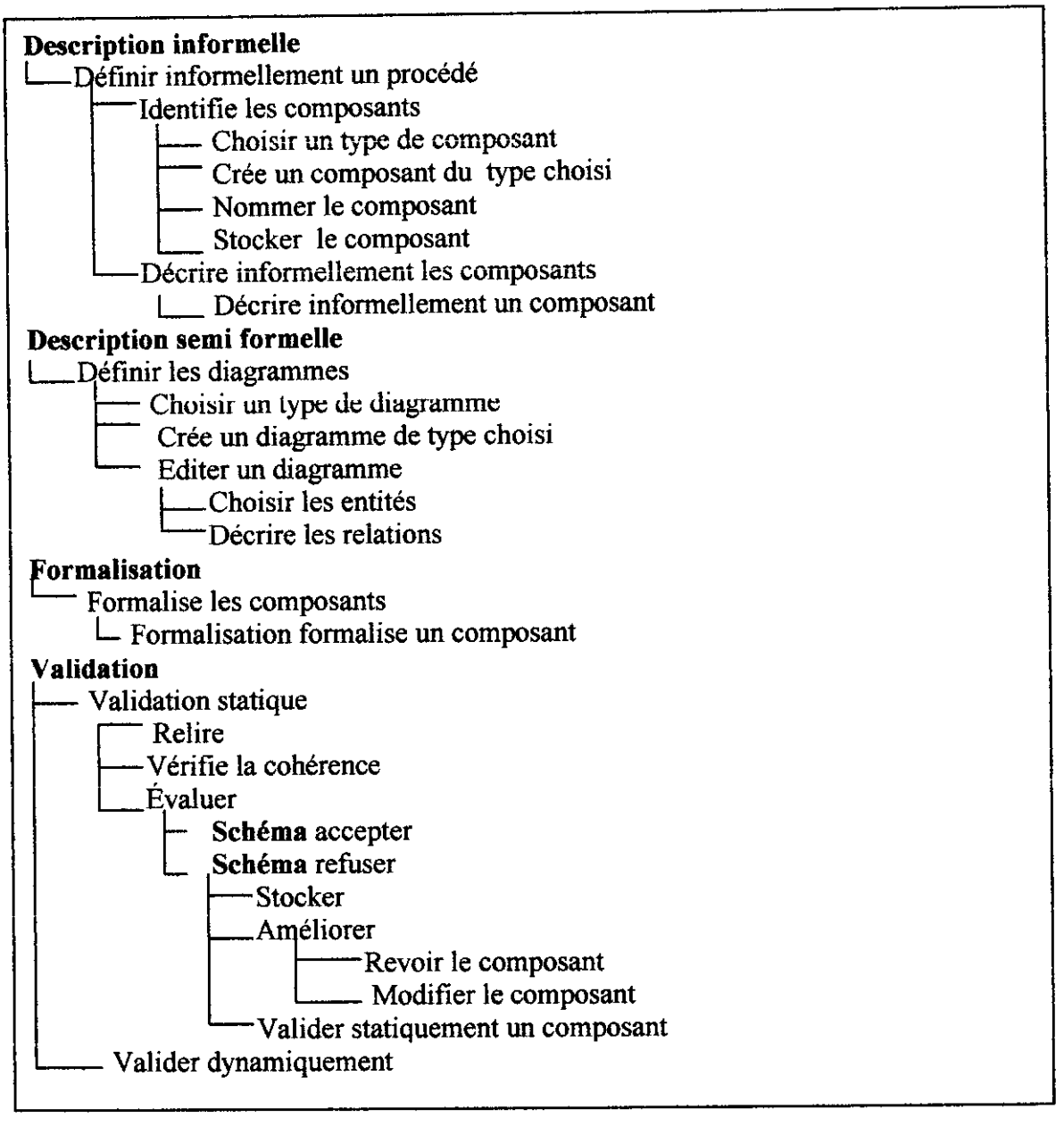


Figure II.08 : Les activités dans les étapes du cycle général **[Ber 03]**

II.11. Cycle de recherche et de réutilisation/adaptation d'un composant de procédé :

Ce cycle est employé pour retrouver, dans la base, les composant dont les spécifications sont conformes à une spécification donnée et permet de choisir un composant convenable pour le réutiliser.

Ce cycle possède un paramètre **S**, qui peut être remplacé par une activité effective décrivant un moteur de recherche d'élément d'un type spécifique. Par

exemple :ce paramètre peut être une instance avec une activité telle que « rechercher les spécifications » ou « rechercher les composant » l'activité racine est décomposée en trois sous activités pour la recherche (l'activité formelle correspondant au paramètre **S),**pour choisir,et réutiliser une unité de procédé. L'activité « réutiliser une entité » propose trois schémas de réalisation qui permettent de décrive une nouvelle entité à partir d'une entité existante.

II.12. Le langage PBOOL+ :

Le langage **PBOOL+** est une extension de langage de description de procédé **PBOOL** auquel il ajoute la notion de composants complexes. Il propose une représentation graphique des composants de procédés pour favoriser le travail des concepteurs de procédés. Nous utilisons un sous ensemble de UML pour décrire graphiquement les composants élémentaires (activités, produits, rôles, et stratèges) et des graphes de dépendances (d'utilisation) pour décrire les composants complexes.

II.13. SPEM (Software Process Engineering META MODEL):

SPEM est un méta modèle en cours d'adoption par L'OMG comme norme pour la description des procédés. Le SPEM s'appuie sur le formalisme UML et s'intègre dans l'approche MDA (Model Driven Architecteur) préconisée par L'OMG. C'est donc une instance du méta méta modèle MOF (Meta Object Facility) et à la fois un profil UML. La (figure II.09) illustre la position du SPEM dans l' architecteur de quatre couches. **[Ref 04]**

Le SPEM est découpe en sept paquetages (figure II.10).
-le paquetage « Data Types » introduit les types de base.
- le paquetage « Core » intègre de nombreux concepts communs avec le paquetage Core d'UML.
- le paquetage « Model management » définit la notion de paquetage.
-les quatre dernier paquetages définissent les unité véritablement spécifiques pour modéliser des procédés.

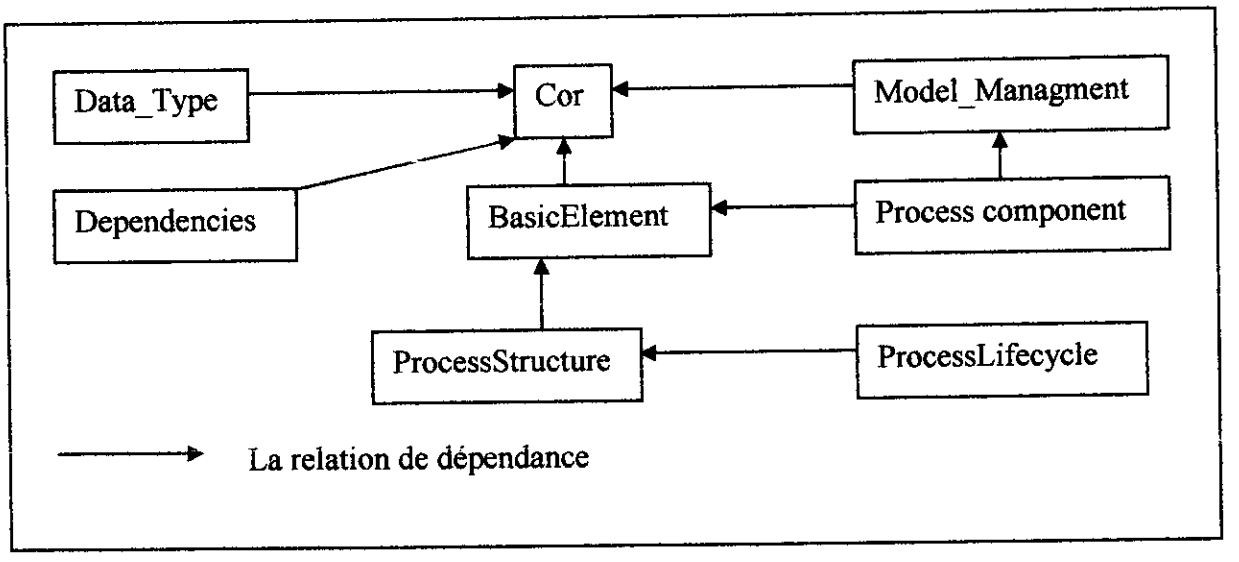


Figure II.09: Décomposition du SPEM en paquetages **[Ber 03]**

Dans le méta modèle SPEM, un procédé (Process) est considéré comme une suite d'activité (Activity), réalisée par le rôle (ProcessRole),qui vont utilisera et consommera des produits (workProduct) et des informations (informationEliment).

Une activité peut être décomposée en plusieurs activités élémentaires (Step). Une activité (Activity), une étape (Phase), une itération (Iteration) et un cycle de vie (LifeCycle) sont des sous-classes de la définition du travail (WorkDefinition). Elles sont des structures hiérarchiques qui permettent de décomposer une définition de travail, l'ordre suggéré dans SPEM est lifecycle, phase, iteration, activity, step. Mais cet ordre n'est pas imposé, la seule véritable contrainte est qu'un cycle de vie ne doit contenir que des étapes. Une définition de travail à des pré conditions (precondition) et des objectif (Gool), qui peuvent être vus comme des post-condition.

Les activités peuvent être regroupées selon des disciplines thématiques (Discipline). Il est également possible de définir des procédés de façon modulaire avec les composants de procédé (Process Component).

Le SPEM peut être employé comme un méta modèle autonome ou comme un profil d'UML. En tant que profil UML. Le SPEM définit les capacités de modélisation consacrée au domaine de développement, et bénéficie du pouvoir d'expression d'UML. Les notion proposées par le SPEM incluent des diagrammes basés sur UML (diagramme de classes, diagramme de composant, diagramme de cas d'utilisation, diagramme de séquence, diagramme d'état/transition et diagramme d'activité) et des symboles suggérés.

II.14. Synthèse générale de l'environnement RHODES :

Plusieurs auteurs ont travaillé sur la caractérisation de la notion de composant logiciel. J. Sametinger J. donne une synthèse intéressante. Nous proposons ci - dessous une adaptation des caractéristiques proposées par cet auteur dans le contexte des composants de procédés.

(C1) La séparation entre spécification et implantation : c'est une Caractéristique qui permet de développer séparément la partie
Implantation et la partie spécification. Elle permet de développer plusieurs implantations pour la même spécification. Elle joue un rôle important pour l'évolution de composants.

(C2) L'auto description («self-described»): chaque composant doit contenir des informations qui le décrivent, C'est à dire qu'il contient des informations pour décrire à quoi il sert, comment il peut être réutilisé, quelle est son origine, etc. Ces informations sont nécessaires pour la réutilisation.

(C3) L'autonomie (« self - contained ») : cette caractéristique permet de réutiliser un composant indépendamment des autres dans le cas idéal. Celle ci exige la propriété de fermeture transitive de la dépendance d'utilisation pour un composant. Cette propriété est cependant trop restrictive. Nous l'avons remplacée par une Propriété plus faible: chaque composant doit déclarer explicitement tous les composants dont il dépend.

(C 4) La possibilité d'évolution : c'est l'évolution de chaque composant durant son cycle de vie. L'évolution est étudiée à deux niveaux: l'évolution statique et l'évolution dynamique. L'*évolution statique* concerne l'état des composants de procédés avant l'exécution du procédé. C'est la possibilité de gérer des versions différentes de procédés, d'améliorer le procédé, de modifier le procédé avant son exécution. L'*évolution dynamique* concerne l'état des composants de procédés à l'exécution du procédé.

(C 5) La cohérence: elle permet de décrire rigoureusement le processus de développement.
Nous nous intéressons à la cohérence statique et à la cohérence dynamique. La *cohérence Statique* est contrôlée avant l'exécution du procédé. Elle comprend la cohérence syntaxique des descriptions (contrôlée par des règles de production) et la cohérence structurelle (propriétés des relations entre composants de procédés avant l'exécution).La *cohérence dynamique* est contrôlée à l'exécution du procédé Voici. Un exemple de cohérence statique: *tout produit utilisé par une sous - activité doit apparaître redans les résultats d'une autre sous activité ou comme entrée de l'activité parente*. Dynamiquement, il doit également être vérifié que le produit a été effectivement renseigné avant d'être utilisé (dépendance entre activités). Ce ci correspond à une vérification de cohérence dynamique. Nous avons établi un système de Prédicats, qui sont considérés comme les prédicats axiomatiques de RHODES. La cohérence dynamique est également assurée par

des prédicats définis sur les activités
(Pré-, post-conditions et invariants). Ces prédicats caractérisent le bon déroulement du développement et sont donc contrôlés à l'exécution par le noyau d'exécution.

(C6) *La description graphique et textuelle*: elle permet aux concepteurs de procédés de décrire facilement des composants de procédés. Des outils peuvent être développes pour faciliter la saisie et la modification de ces descriptions de composants. **[Cou 02]**

CONCLUSION

Dans cette partie nous avons présenté deux grandes approches basées sur la réutilisation, nous sommes intéressé par RHODES car c'est l'approche la plus récente et la plus efficace jusqu'a ce jour.

Chapitre II

Partie III

S.M.A

Dans cette partie, nous présenterons présenté la notion d'agent, les types et les caractéristiques des agents et systèmes multi agents.

I. CONCEPT D'AGENT

I.1. Définition :

Il n'existe pas une définition standard d'agent mais d'après notre point de vue un agent est une entité logicielle ou physique à laquelle est attribuée une certaine mission à accomplir, d'une manière autonome et en coopération avec d'autres agents.

On peut dire aussi qu'un agent est un système informatique qui appartient à un environnement complexe et dynamique, dédier à des buts spécifiques.

I.2. Caractéristiques d'un agent :

« On pouvons identifier des caractéristiques générales pour un agent :

Situé : l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement.

Autonome : l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne.

Proactif : l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au bon moment.

Réactivité : l'agent doit avoir une capacité à effectuer une nouvelle tache lorsque un événement survient.

Social : l'agent doit être capable d'interagir avec les autres agents (logiciels ou humains) afin d'accomplir des tâches ou d'aider ces agents à accomplir les leurs.

Bien qu'il existe d'autres caractéristiques spécifiques aux domaines d'utilisation d'agent comme :

Coopération : la capacité qu'ont les agents d'effectuer un travail collaboratif pour accomplir une tâche donnée.

Adaptation : un agent capable d'utiliser ses connaissances afin de modifier son comportement (augmentation de l'espace disque, détection d'un seuil,... etc.)

Communication : la possibilité d'échanger des messages entre agents. »
[Phi 00]

I.3. Environnement d'agents:

Du point de vue d'un agent, l'environnement correspond à tout ce qui lui est extérieur :

Les autres agents font aussi, à priori, partie de l'environnement.

Cependant, l'environnement est souvent considéré comme une structure centralisée dynamique mais non autonome, et qui contient l'ensemble des entités avec lesquelles interagissent les agents.

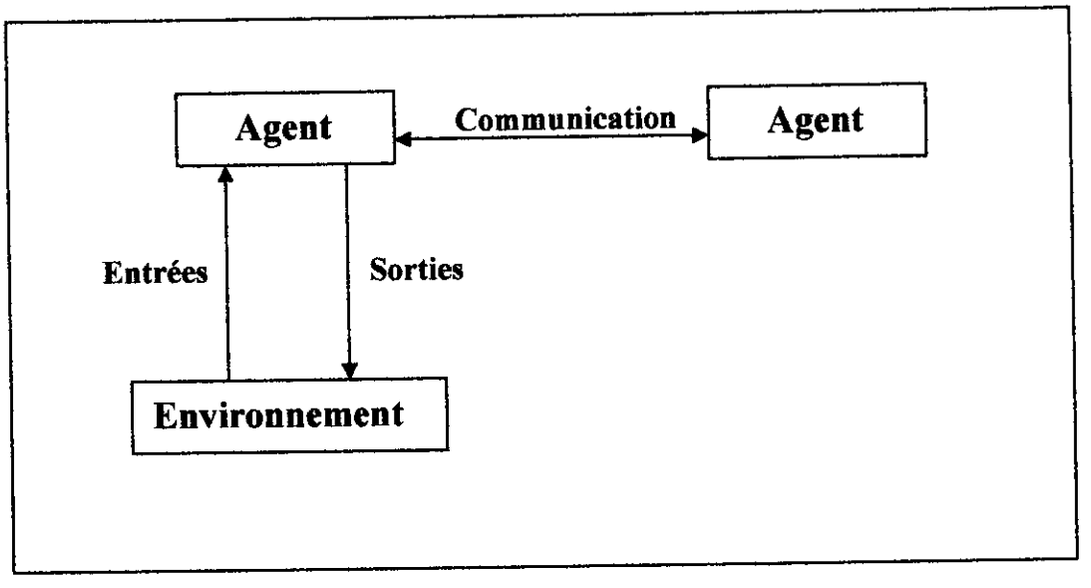


Figure II.10 : Interaction entre un agent est son environnement. **[Ref 10]**

I.4. Types d'agents

On distingue deux types d'agents :

a) Les agents réactifs :

Ces agents ne possèdent pas une représentation complète de son environnement et ne sont pas capable de tenir compte de ses actions passées. Ils ont un comportement du type stimulus – réponse.

b) Les agents cognitifs :

Ces agents sont plus évolués. Ils ont une représentation globale de leur environnement et des autres agents avec lesquels ils communiquent. Ils savent tenir compte de leur passé et s'organisent autour d'un mode social d'organisation.

Agents cognitifs	**Agents réactifs**
Représentation explicite de l'environnement	Pas de représentation explicite
Architectures complexes	Architectures simple
Organisation explicite	Organisation implicite ou induite
Communication explicite	Communication via l'environnement
Petit ou moyen nombre d'agents	Grand ou très grand nombre d'agent
Peut tenir compte de son passé	Pas de mémoire de son historique

Tableau1 II.01 : Comparaison entre agents **(cognitifs - Réactifs)**

I.5. Les agents intelligents

Ce sont des composants logiciels et / ou matériels combinant les trois caractéristiques (autonomie, coopération, adaptation) à leur plus haut niveau.
Ils sont en général dotés de la capacité d'apprentissage.

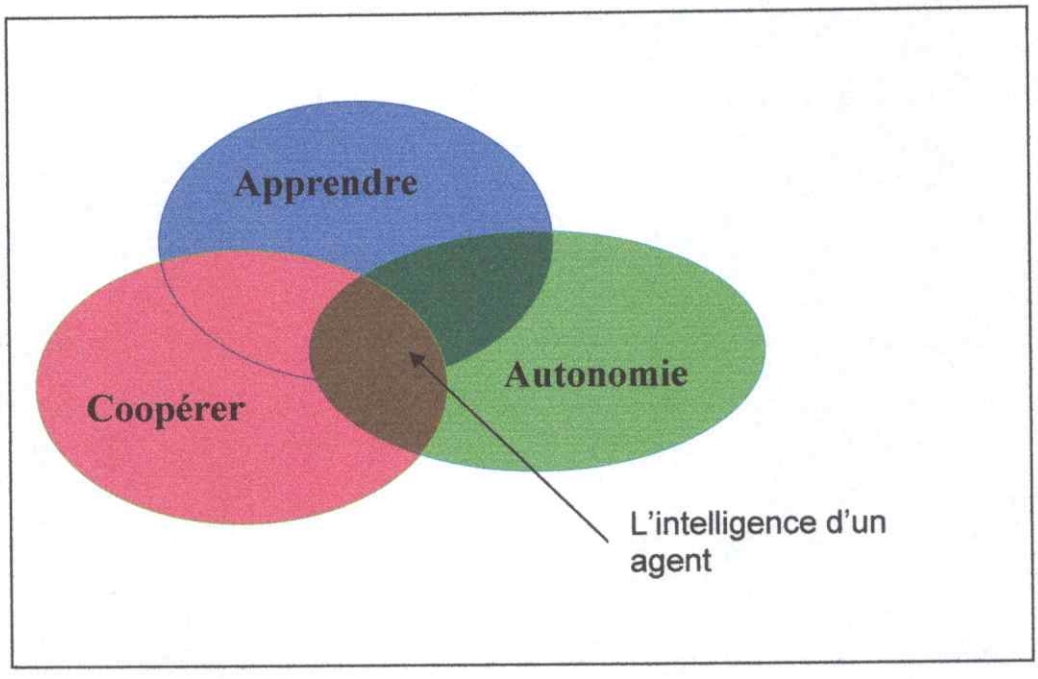
« Les agents intelligents appartiennent à la classe d'agents cognitifs. Ils sont capables à des degrés différents :

D'être autonome : Fonctionnement automatique.

De communiquer : Echange des informations avec d'autres programmes ou des hommes.

D'apprendre : capables de réagir avec un environnement, de s'adapter et de prendre une décision ou d'enrichir eux mêmes leur propre comportement, sur la base d'observations qu'ils effectuent. » **[Ref 09]**

Le schéma suivant représente les caractéristiques d'un agent intelligent.



Figurell.11 : Les caractéristiques d'un agent intelligent. **[Ref 09]**

I.6. Domaine d'application

Les domaines d'application sont loin d'être limités. Il est cependant possible de distinguer les espaces d'applications suivants.

- **Résolution de problèmes et intelligence artificielle distribuée**

Exemples : Gestion de système de puissance, contrôle de trafic aérien, soin médical, gestion de réseaux de télécommunication, contrôle de navette spatiale, gestion de réseaux de transport.

- **Agent d'interface**

Ce sont des logiciels qui emploient les techniques d'intelligence artificielle pour fournir assistance à un utilisateur pour une application particulière. L'agent observe les actions de l'utilisateur, lui suggère des astuces, lui apprend des raccourcis.

- **Agent d'information**

Un agent d'information est un agent qui a accès à une ou plusieurs sources d'information et qui est capable de récolter et manipuler cette information pour répondre à des demandes d'utilisateurs ou d'autres agents.

Exemple : si l'utilisateur a entendu dire qu'un chercheur effectue un travail sur tel domaine dans telle université, il demande à l'agent d'effectuer une recherche, et celui-ci ramène un rapport technique ainsi que les coordonnées du chercheur concerné.

I.7. Composants de l'agent intelligents : [Ref 05]

Un agent intelligent contient un ou plusieurs des éléments suivants :

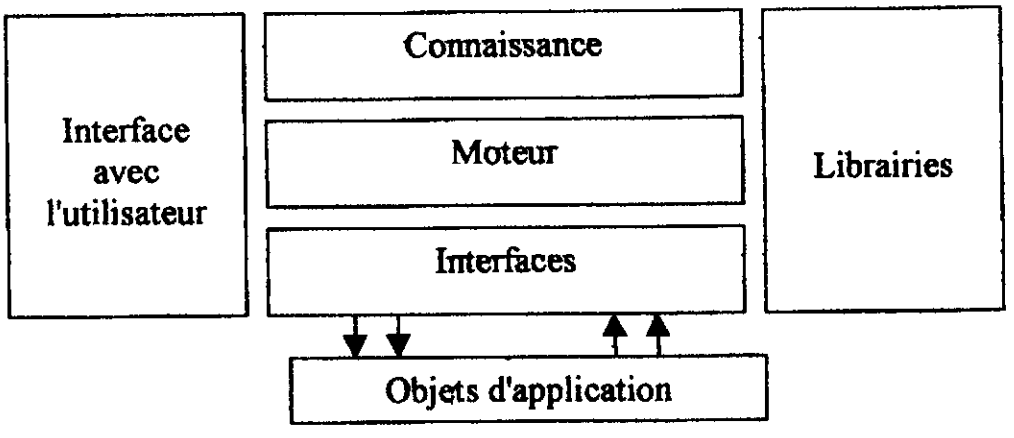
Le moteur : (le cerveau de l'agent) lui permettant de tenir des raisonnements plus ou moins complexes,

Une base de connaissance prédéfinie : ce que l'agent sait, croit et pense. La connaissance peut être basée sur des applications existantes.

L'interface avec les autres applications : C'est l'interface qui détermine le champ d'application d'un agent.

L'interface avec l'utilisateur : permet la communication entre l'agent et l'utilisateur. L'utilisateur donne ses instructions à l'agent et ce dernier renvoie les résultats ou donne des informations.

Un mécanisme d'apprentissage



Figurell.12 : Schémas de la représentation les composants de l'agent intelligent **[Ref 06]**

II. SYSTEME MULTI AGENTS

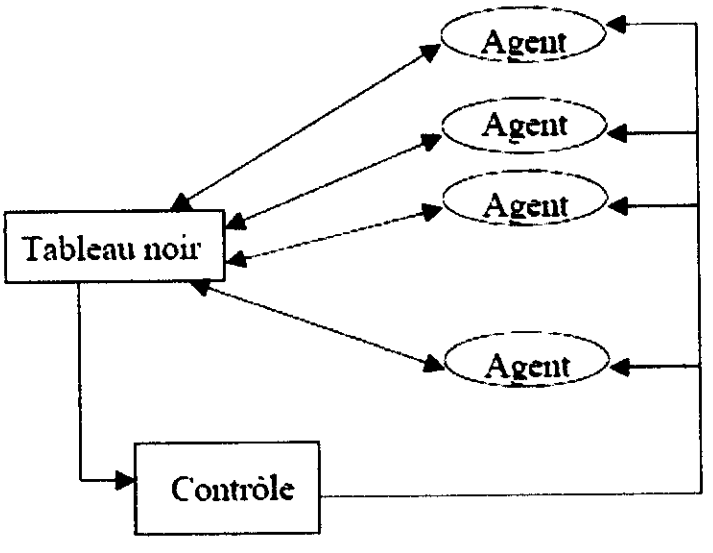
II.1. Définition (Système multi agents)

« Un SMA est un ensemble d'agents situés dans un certain environnement et interagissant selon une certaine organisation » **[Ref 07].**

Faire coopérer un ensemble d'entités (agents) dotées d'un comportement intelligent, coordonner leurs buts et leurs plans d'actions pour résoudre un problème.

Un système multi agents est constitué d'un ensemble d'agents vivants au même moment, partageants des ressources communes et communicant entre eux.

II.2. Architecture des SMA :



Figurell.13 : Architecture de SMA **[Ref 08].**

II.3. Caractéristiques d'un SMA : [Gul 00]

Le système multi agents possède deux caractéristiques importantes :

-Ouvert ou fermé : tous les agents de système sont développés ou non par le même programmeur.

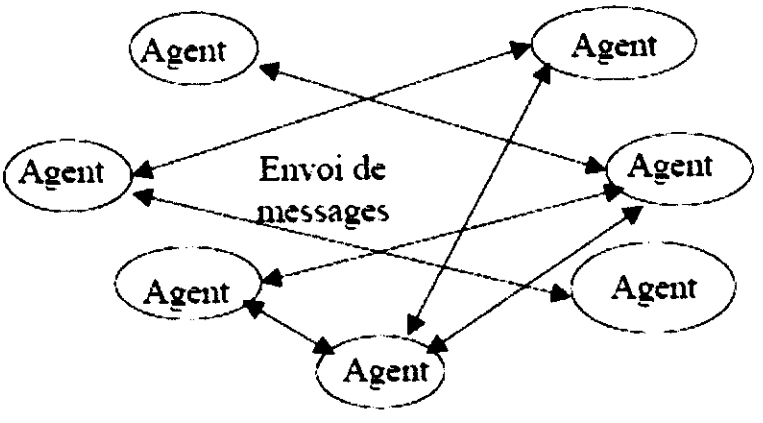
-Homogène ou hétérogène : les agents poursuivent tous ou non le même but.

Comme il existe d'autres caractéristiques :

- Distribution totale des connaissances et du contrôle.

-Traitement local.

-Communication par envoi de message.



Figurell.14 : Communication entre les agents **[Ref 07].**

II.4. Apport des SMA

L'approche Multi agents est justifiée par :

- **L'adaptation à la réalité :** réalise des tâches réelles par l'utilisation de ses connaissances afin de modifier son comportement.

- **La coopération :** les agents de système effectuent un travail collaboratif pour accomplir une tâche donnée.

- **La résolution de problèmes complexes :** le système peut résoudre les problèmes de différentes complexité.

- **La modularité :** chaque agent à son propre module dans le système.

- **L'efficacité :** le système multi agents est plus efficace à résoudre les problèmes q'un agent.

- **La fiabilité :** le système multi agents est fiable.

- **La réutilisation :** la structure de système multi agents assure la réutilisation.

II.5. Exemples d'application d'agents

Applications temps réel :

Les agents ont été bien évidemment appliqués au domaine des systèmes temps réel; ce dernier maintient des systèmes à contrainte souple (soft real time system). Nous voyons de plus en plus des systèmes temps réel dit Hard utilisant des agents.

Pour la recherche d'Information :

Une grande partie des applications de Système Multi agents est dans le domaine de recherche d'information. Parmi ces nombreuses applications dans ce domaine, nous pouvons trouver " NetSA" : une architecture de système multi agents pour la recherche d'information dans des sources hétérogènes et réparties. Ce système comporte plusieurs types d'agents comme :

- Un agent utilisateur : en charge de la cueillette et du filtrage des informations provenant et allant vers l'usager.

-Un agent courtier : servant de répertoire pour les agents qui évoluent au sein de NETSA.

- Des agents ressources reliés chacun à une ressource d'informations et pouvant rapatrier et mettre à jour les données.

- Un agent d'exécution en charge de la décomposition des tâches et du suivi du déroulement d'exécution des différentes sous tâches.

- Un agent ontologie en charge du maintien de la cohérence des concepts utilisés par les agents.

CONCLUSION

Dans cette dernière partie de ce chapitre nous avons présenté les défferents principes d'agent et leurs caractéristiques, dans le but d'assurer la compréhension de la suite de notre travail.

Chapitre III

Partie I

Généralité Sur La Modélisation

Notre objectif est de concevoir une base de composants logiciels pour leur réutilisation afin de modéliser un procédé logiciel a base de composant.

Pour la conception nous utilisons comme langage de modélisation l'UML basé sur le processus de développement RUP. Ainsi nous présentons un rappel sur ces deux concepts.

I. DEFINITION UML

UML (Unified Modeling Language) est né de la consolidation de trois méthodes objet :
OMT (Rumbaugh), Booch, OOSE (Jacobson). Cette consolidation a été marquée par trois étapes :

- Le regroupement des trois équipes au sein de la société Rationnel ;
- Le recentrage du projet de standardisation sur le langage de modélisation, les aspects purement méthodologiques étant laissés de coté.
- La décision de l'Object management group en 1997.

Le succès a été immédiat et UML est aujourd'hui universellement accepté et supporté par l'ensemble des outils de développement **[Rem 99].**

UML est un langage de modélisation et n'est pas une méthode. Il n'est pas propre a l'approche objet (une partie importante des concepts et des outils est empruntée aux méthodes classiques).

UML fournit un cadre conceptuel commun mais avec une liberté à l'acteur pour développer leurs méthodes et leurs outils.

Schématiquement, UML peut être défini par trois plans :

- Les concepts: dont la sémantique est complètement et formellement définie.

-Les digrammes : utilisés pour spécifier les besoins et les systèmes.

-Les mécanismes d'extensions : pour intégrer aux processus des stéréotypes spécifiques aux différents méthodes et contextes applicatifs.

- La section suivante à pour but de représenter le langage de modélisation de données, UML pour :
-Décrire les concepts fondamentaux du langage.
-Montrer les différentes relations entre les classes.
-Présenter l'ensemble des diagrammes.

I.1. Les concepts

UML supportent quatre types de concepts **[All 01].**

.Les concepts structurels : représentés par les classes, les interfaces, les collaborations...

.Les concepts comportementaux : représentés par les interaction et les états d'objets

.Les concepts annotationnels : représentés par les notes. Où une note est un commentaire attaché à un ou plusieurs éléments de modélisation **[Mul 01].**

.Les concepts de regroupement : représentés par les sous systèmes et les paquetages.

I.2. Les digrammes

Le méta modèle suivant présente les neuf digrammes de langage UML.
On peut définir les procédés logiciels comme étant une suite d'étapes réalisées dans un but donné et qui servent à gérer et assister le développement logiciel.

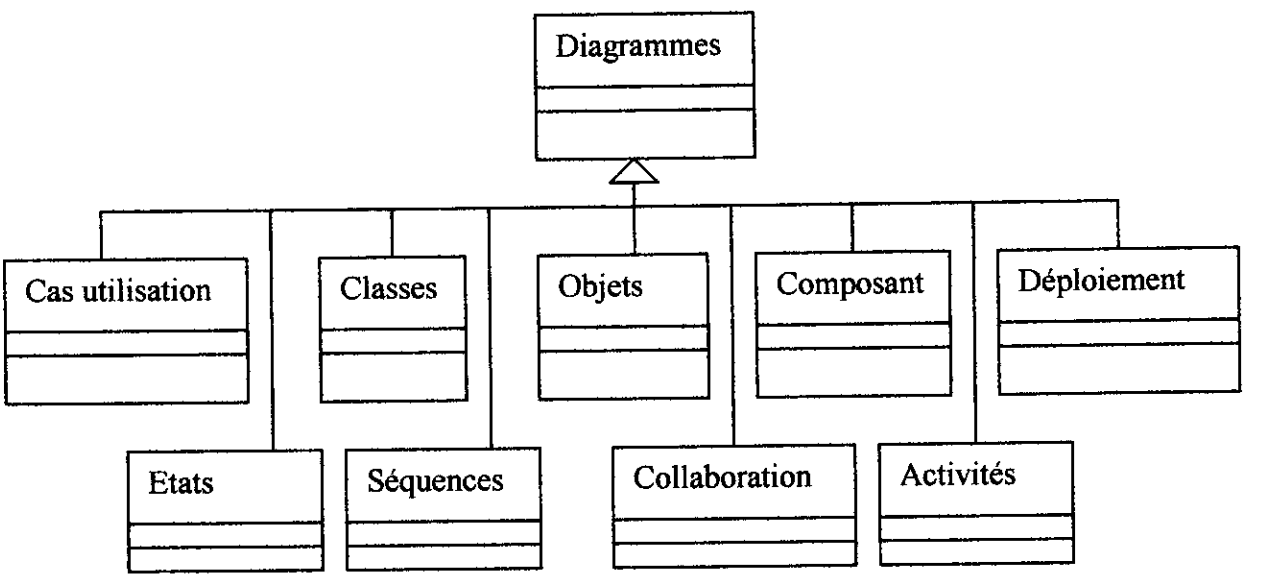


Figure III - 01 : Les diagrammes d'UML **[All 01]**

On peut distinguer quatre groupes essentiels de diagrammes de l'UML :

- **Diagramme de cas d'usage :** pour modélises les fonctionnalités des applications.

• **Un acteur :** représente un rôle joué par une personne ou une chose qui interagit avec un système **[Mul 01].**L'acteur interagit avec les cas d'utilisation par envois de messages.

• **Un diagramme de cas d'utilisation**: permet de décrire les interactions entre les acteurs de l'organisation et le système dans chacun des cas d'utilisation envisagés. Il décrit le comportement d'un système au point de vue de l'utilisateur et fixe les limites du système et les relations entre les systèmes et l'environnement **[Mul 01]**

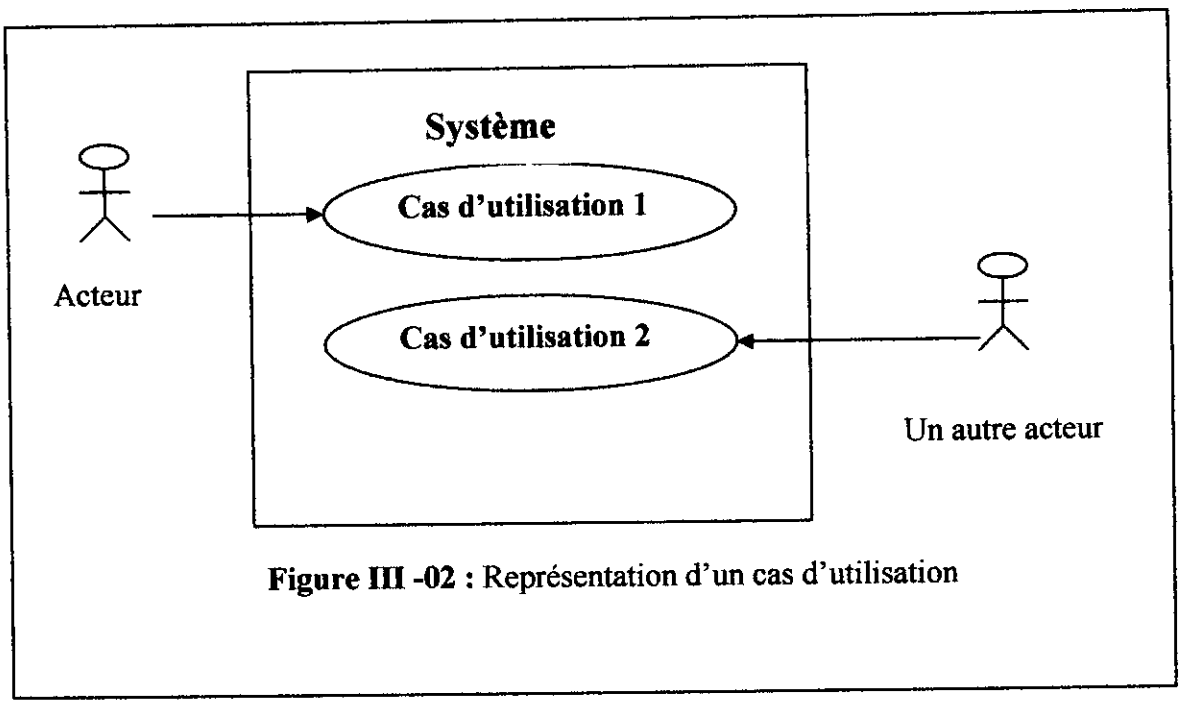


Figure III -02 : Représentation d'un cas d'utilisation

Les relations qui peuvent exister dans un diagramme de cas d'utilisation sont :

• **La relation d'utilisation :** lorsque un ou plusieurs tâche sont utilisées régulièrement, on peut les factoriser dans un même cas d'utilisation et faire de telle sorte que d'autre cas d'utilisation l'utilisant en le pointant par une flèche **[Gab 98].**

-comme nous pouvons décomposer un cas d'utilisation complexe en plusieurs sous cas d'utilisation.

• **La relation d'inclusion :** le cas d'utilisation source comprend également le comportement de son cas d'utilisation destination. Cette relation à un caractère obligatoire (a la déférence de la généralisation) et permet ainsi de décomposer des comportements partageable entre plusieurs cas d'utilisation différent **[Gab 98].**

• **La relation d'extension :** le cas d'utilisation source ajoute son comportement au cas d'utilisation destination. L'extension peut être soumise à condition. Cette

relation permet de modéliser des variantes de comportement d'un cas d'utilisation **[Gab 98].**

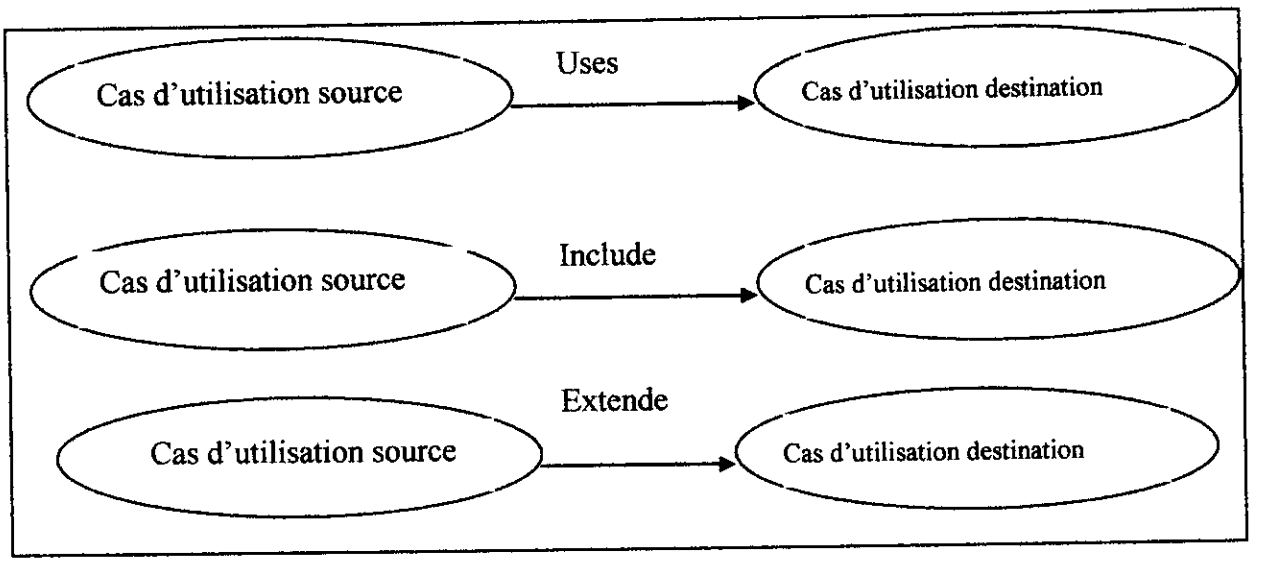


Figure III -03 : Les trois relations entre les cas d'utilisation.

- Diagramme de Classes :

Pour modéliser les objets gérés par le système.
Les digrammes de classes montre la structure du système et les éléments des classes tels que : les classes, les relations d'héritages entre classes, les associations, dont les agrégations, les attributs, les opérations et la spécification d'opération et contraintes au niveau des entités
[Ber 02].

-Stéréotype : permet de définir une utilisation particulière d'élément de modélisation existant ou de modifier la signification d'un élément.

-Acteur : représente les rôles des interlocuteurs du système.

-Objet contrôle : représentés les classes effectuent des traitements internes au système.

-Objet interface : représentés les limites du système.

-Objet entité : représente les objets du domaine **[Her 00].**

- Diagramme d'interactions :

Pour modéliser la manière dont les objets collaborent à la réalisation des cas d'usage. Et pour cela nous avons (les diagrammes de séquence et les diagrammes de composant).

-Les diagrammes de séquence :

Les diagrammes de séquences permettent de représenter les interactions entre objets en précisent la chronologie des échanges de messages. Ils peuvent être utilisés pour représenter les scénarios d'un cas d'utilisation donnée **[Gab 98].**

-Interaction : modélisant un comportement dynamique entre objets, en insistant sur la chronologie des envois de messages **[Ber 02].**

-les messages : les messages échangés sont représentés au moyen de flèches horizontales partant de l'émetteur vers le récepteur. L'ordre de l'envoi est donné par la position sur l'axe vertical
[Her 00].

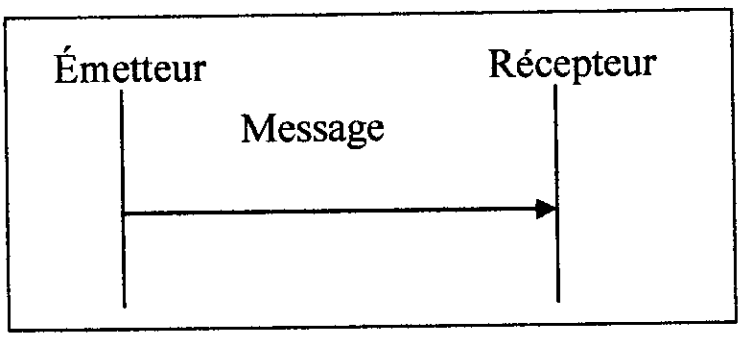


Figure III -04 : Agencement de message.

-Période d'activation : Correspond au temps lequel un objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet.

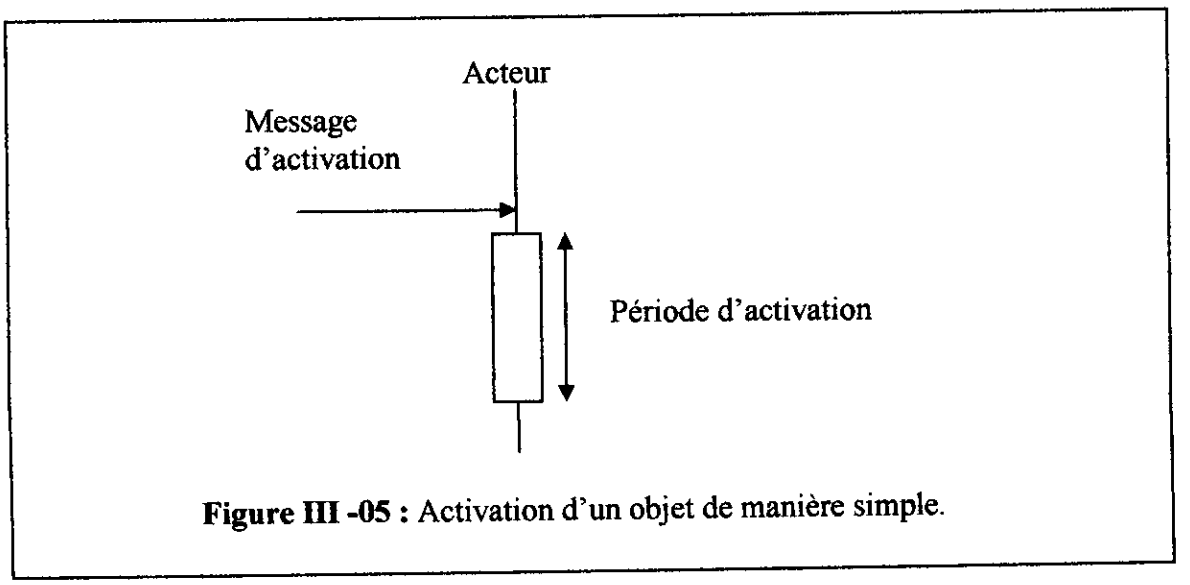


Figure III -05 : Activation d'un objet de manière simple.

-Diagramme de composant et déploiement :

Pour modéliser les éléments logiques et physiques du système.

-Diagramme de composant :

-composant : élément physique qui représente une partie implémentée d'un système. Il peut être du code, un script, un fichier de commandes .les composants présentent un ensemble d'interfaces **[Ber 02].**

-Les diagrammes de composant permettant de décrire l'architecture physique et statique d'une application en terme de modules : fichier sources, librairie, exécutable etc.
Ils décrivent les éléments physiques et leur relation dans l'environnement de réalisation.

-Les composants du système sont : le sous système, le module, le programme, les sous programmes, le processus et la tâche.

-Les sous système :

Un sous système regroupe des éléments de modélisation **[Gab 98]** : cas d' utilisation de classes, objets, modules ou composant. L'importation permet aux éléments d'un sous système d'accéder aux éléments d'un autre sous système

-Module : un sous système peut être décomposé en modules, chaque module correspond à un ensemble d'éléments physiques (fichier, sous ensemble de logiciel.).

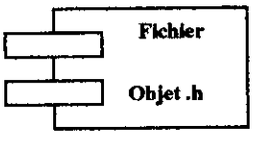


Figure III - 06 : Représentation d'un composant.

-Les diagrammes de déploiement :

Les diagrammes de déploiement montrent la relation physique des matériels qui composent le système et la représentation sur ces matériels.

Un diagramme de déploiement permet également de représenter les relations entre différents nœuds.

Puisque UML est un langage de modélisation est n'est pas une méthode cette situation implique qu'il y à manque de la démarche de développement (processus de développement) est pour cela nous proposons le processus de développement RUP.

II. LE PROCESSUS DE DEVELOPPEMENT RUP

RUP est le mnémonique de Rationel Unified Process qui est un processus de développement et de conduite de project. Pour comprendre RUP il faut savoir se qu'il signifie :

Rationel : Se qu'est fondé sur la raison, conforme au bon sens, déterminer par des calcules ou par des raisonnements.

Unified : Unifié.

Process : Ensemble de fait ou de phénomènes successifs, déroulement d'une opération.

En résumé : RUP c'est comment organiser de manière efficace, unifié et moins coûteuse un processus de conduit de projet.

RUP fournit une méthode de conduite de projet informatique entièrement basée sur l'UML, qui prend en charge tous les aspects liés au travail collaboratif. Se progiciel permet donc naturellement la réalisation de diagrammes UML **[Réf 11].**

RUP est définie par ces créateurs comme un framework de processus génériques destiné à gérer l'ensemble des aspects d'un projet de développement logiciel. En clair signifie que RUP peut être adapté à tous les type de projets, en tenant compte : de la taille du projet, de niveau de compétence initial d'acteurs.

RUP est piloté par les cas d'utilisation, centré sur l'architecteur, itératif et incrémental **[Réf 12].**

-Pilote par les cas d'utilisations :

Les cas d'utilisation dans RUP, ne servant pas qu'à améliorer la capture des exigences fonctionnelles (exigence qui sont liées aux fonctionnelle de l'application, il existe aussi des exigence non fonctionnelles liées aux performances, à la disponibilité, ...etc.) Des utilisateurs.
Il servent d'intrant aux différents activité de processus : analyse, conception, implémentation, rédaction des testes. Le projet dans son ensemble a pour objectif de réaliser les cas d'utilisation.

-Centré sur l'architecteur :

L'architecteur sert :
-A comprendre le système lorsque ce dernier est complexe.
- A piloté le projet (en découpant les taches en fonction des composants significatifs identifies et des fonctions qu'ils entretiennent).
-A favoriser la réutilisation.

-les 4+1 vues :

RUP utilise le model des 4+1 vues pour guider l'élaboration de l'architecture, ce modèle comprend quatre vues principales et une vue coordinatrice :

.La vue logique : qui correspond à la structure de la l'application en terme de classe. De sous système.

.La vue des processus : qui correspond à une description des tâches à exécuter par l'application (processus, threads), et de leurs relations.

.La vue de développement : qui est une traduction concrète du modèle logique (structure en package, en bibliothèque)

.La vue physique : qui constitue une description de la réparation de l'application dans les différents nœuds de la plate forme de production.

.La vue des cas d'utilisation **:** qui sert à cadrer les vues précédentes.

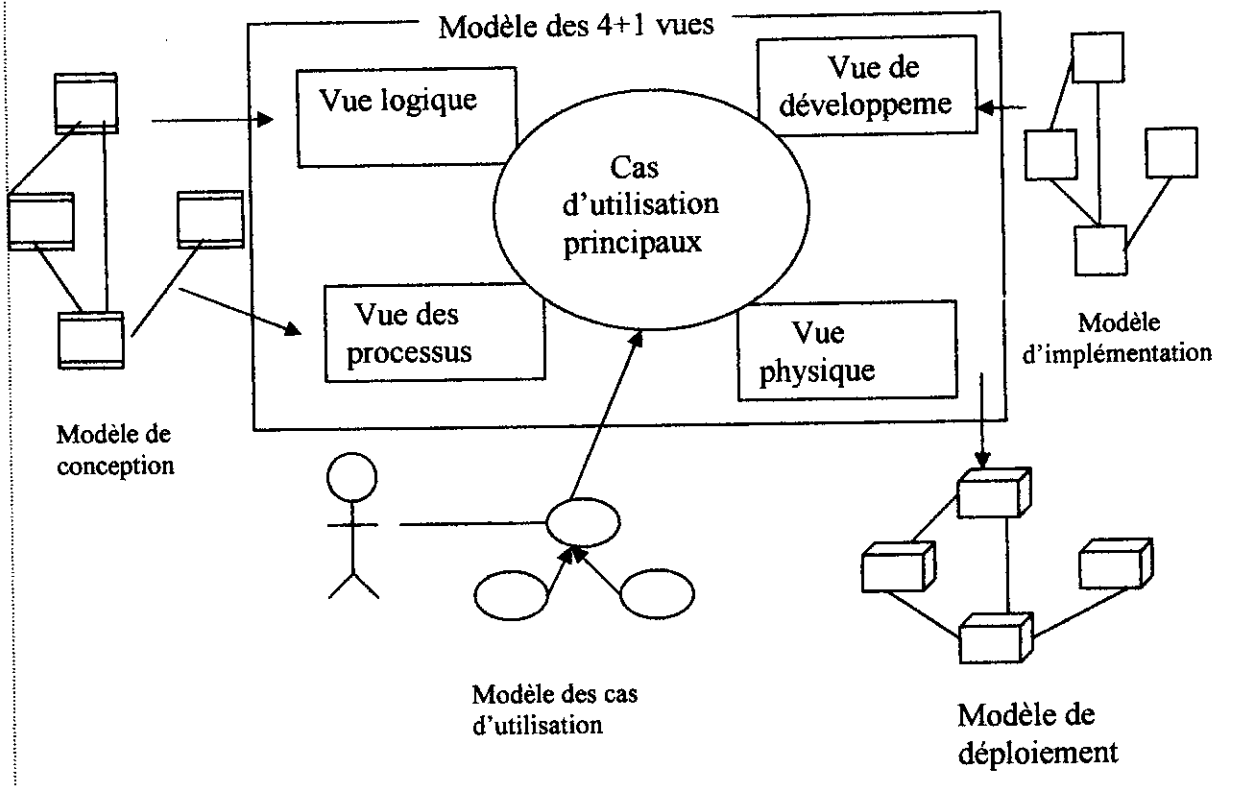


Figure III-07 : Lien entre les 4+1 vues de P.Krutchen et les modèle RUP.

-Itératif :

RUP est comme la plupart des nouveaux processus de développement itératif. L'objectif de la recherche est toujours de favoriser une diminution progressive des risques des le début du projet.

-Incrémental :

RUP set aussi incrémental car chaque itération est planifiée, chaque itération est un projet à l'intérieur du projet qui met à jour une partie de l'ensemble.

III. BASE DE COMPOSANT DE PROCEDES LOGICIEL REUTILISABLE :

La base de composants de procédés est utilisée pour le stockage et la réutilisation des composants de procédés, pour ce la nous gérons les composants de procédés en tenant compte de deux aspects :

III.1. Aspect textuel :

Ce type de gestion nous permet de faire évoluer statiquement les composants de procédés, c'est à dire nous décrivons et nous modifions les composants de procédés avant de les stoker dans la base.

Ce travail fait partie de la formalisation des composants de procédés. Pour réaliser cette formalisation nous devons rajouté la classe phase qui contient les différentes phases exécutées par ce procédé, de plus on lui affecte un numéro de méta procédé pour que nous pouvons l'identifier, ainsi ce numéro il devient une instance dans la classe méta méta procédé. Ces deux dernières étapes sont réalisées pour chaque méta procédé utilise a la réutilisation.

III.2. Aspects structurels :

Cette gestion permit l'évolution structurelle des composants de procédés .elle prend en compte les relations entre composant telle que l'héritage modulaire, relation, cardinalité ...

III.3. Les procédés logiciels utilisés :

Il existe différents types de procédés logiciels. Dans notre travail nous utilisons un type de procédé logiciel bien spécifie. Pour les besoins de la réutilisation (réutilisation des métas modèles de même type), les procédés logiciels traités sont des procédés logiciels centrés activité. Cela implique que dans chaque modèle de procédé la classe activité est présente.

III.4. Les étapes de réutilisation des procédés logiciels :

L'objectif de notre travail est de construire un nouveau procédé logiciel par la réutilisation des procédés existants. Et dans cette phase nous allons détailler notre approche de réutilisation. Cette phase est décomposée en deux grandes parties.

1 -Stockage des procédés existants : cette étape est faite comme suit :

-Formalisation des procédés logiciel : dans cette étape nous passons d'un ensemble de procédés logiciels non formel à un ensemble formel de procédés.

-Remplissage de la base des composants procédé logiciel : après l'étape de formalisation des procédés nous devons remplir la base méta procédés logiciel.

-Le stockage des procédés formel dans le méta méta procédé : cette étape est réalisée par l'agent stocké, dans cette étape l'agent remplira la base de méta méta procédé, cette base sera accessible par tout l'ensemble des agents pour assuré la réutilisation.

2-Modélisation de procédé logiciel : cette étape contient :

-La recherche des procédé à réutiliser : cette étape est réalisée par l'agent recherché, pour rechercher les métas procédés à base de contraintes spécifiées par l'utilisateur.

-L'identification des procédés pour la réutilisation : cette étape et réalisé par l'agent identifie, il se base sur les procédés logiciel déjà recherchés à l'étape précédente pour sélectionner les procédés à utiliser dans la réutilisation.

-La construction de nouveau procédé logiciel : cette étape est réalisée par l'agent collé pour rassembler les sous procédés identifiés pour construire le nouveau procédé logiciel.

Le schéma ci-dessous résume les étapes de notre approche de réutilisation des procédés logiciels :

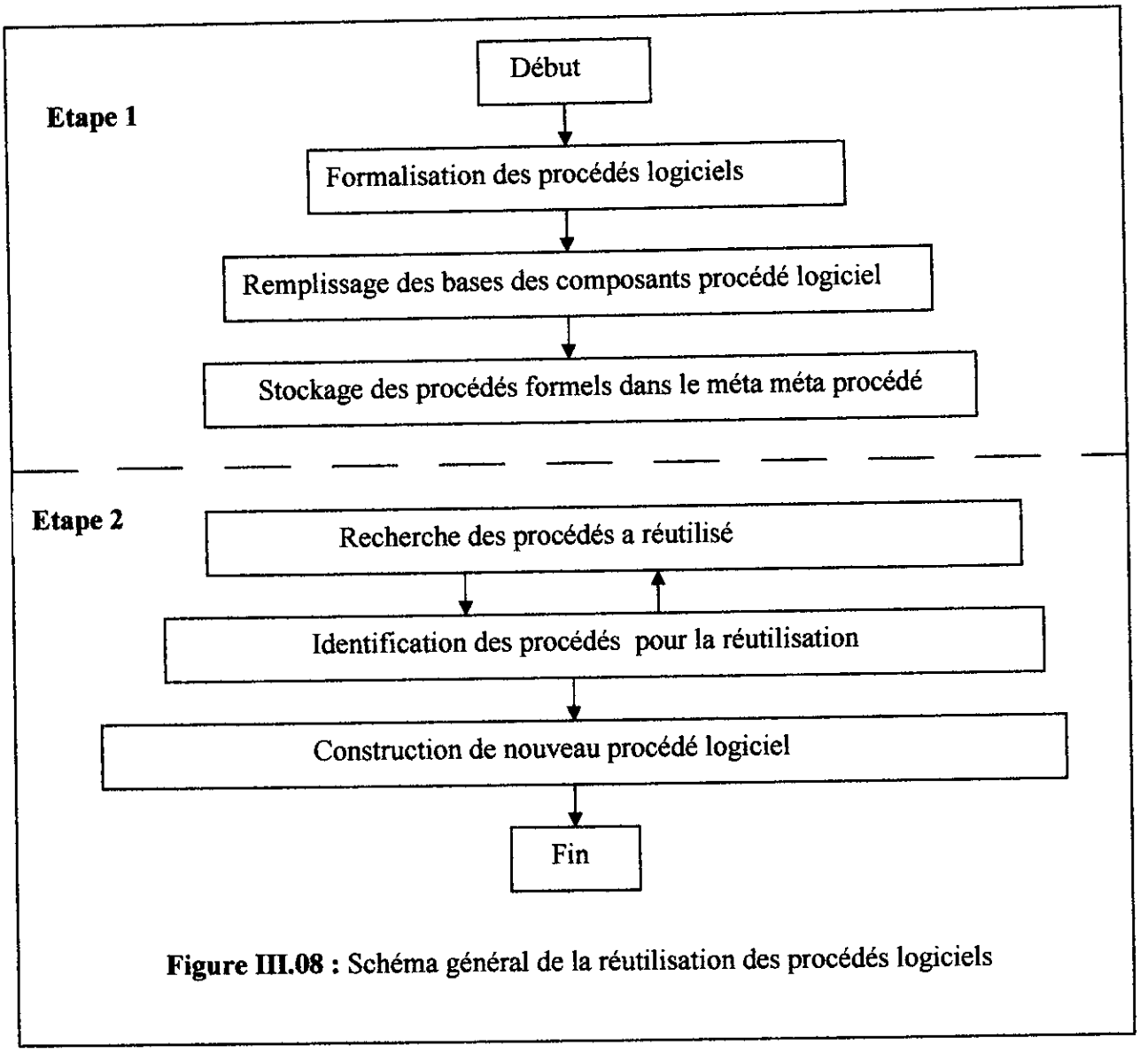


Figure III.08 : Schéma général de la réutilisation des procédés logiciels

III.5. Utilisateurs de système :

Puisque notre application est destinée à modéliser des systèmes logiciels (créer un diagramme de classe et le procédé logiciel qui sera applique) ; l'exécution de notre application nécessite parfois l'intervention de l'utilisateur (en cas de la construction manuelle des fragments procédés logiciel) en plus la gestion de notre système multi agents nécessite de spécifier les entrées / sorties de chaque agent qui sont différent d'un méta procédé à un autre (ces données seront inspirées à partir de l'étape d'analyse des besoins du système à modéliser).

En se basant sur ces trois contraintes précédentes pour dire que l'utilisateur de notre système doit être un expert du domaine du génie logiciel (c'est a dire que l'utilisateur doit avoir l'habitude de modélisé des systèmes dans le domaine du génie logiciel, ou il à des connaissances dans ce domaine).

III.6. Les agents de système :

Notre système est composé de quarte agents sont tous de même type (réactifs) :

- l'agent stocké : pour stocker les données du système.

- l'agent recherché : pour rechercher les composants du nouveau procédé.

- l'agent identifié : pour identifier les sorties du système.

-l'agent montage : pour concaténer les composants du nouveau procédé logiciel.

III.7. Définition de composant procédé logiciel élémentaire (fragment procédé logiciel) :

Pour que nous assurions une bonne gestion de notre application nous proposons une architecture de fragment procédé logiciel à utiliser qui contient essentiellement les champs suivant :

- champ 1 : code fragment (chaque fragment doit avoir son propre code).
- champ 2 : nom phase (il faut affecter le fragment à une phase spécifiée).
- champ 3 : code activité (la première instance de fragment).
- champ 4 : instance de la deuxième classe qui construit le fragment.
- champ 5 : nom relation (la relation ou se situe le fragment élémentaire).
- champ 6 : nom classe (la classe ou se trouve le fragment s'il n'existe pas dans une relation).

La figure ci-dessous représente graphiquement notre procédé élémentaire :

code fragment	nom phase	code activité	instance classe2	nom relation	nom classe

Tableau III.01 : Représentation d'un procédé logiciel élémentaire

III.8. Définition de catalogue d'un méta modèle :

Pour éviter la recherche dans l'ensemble des bases de données (minimiser le temps d'exécution) nous proposons un catalogue pour chaque méta modèle. La création de catalogue est réalisée à l'inscription de méta modèle.
L'architecture du catalogue est la suivante :

Notre catalogue est un tableau de cinq colonnes, chaque colonne contient :
-colonne 1 : le nom du méta modèle.
-colonne 2 : les classes de méta procédé réutilisable.
-colonne 3 : les relations de méta procédé réutilisable.
-colonne 4 : les phases de ce méta procédé.
-colonne 5 : l'ensemble des fragments applicables sur ce méta modèle.

Graphiquement le catalogue est représenté par la figure suivant :

Nom de méta modèle	classes	Relations	phases	Fragments
nom méta modèle	Classe 1 Classe 2 . . . Classe N	Relation 1 Relation 2 . . . Relation N	Nom phase 1 Nom phase 2 . . . Nom phase N	Fragment 1 Fragment 2 . . . Fragment N

Tableau III.02 : Représentation d'un catalogue

-cette organisation facilite la tâche réalisée par l'agent recherché et identifié pour que la recherche soit plus vite et plus facile.

III.9. Entrées / sorties du système :

A la construction de chaque système en génie logiciel la première étape est l'analyse des besoins. Cette étape est très importante dans notre travail car après le remplissage de cahier de charge on doit avoir une idée sur le système à réaliser et pour cela nous nous connaîtrons au minimum :
1- les noms des phases nécessaires.
2- les noms des classes nécessaires.
3- les noms des relations nécessaires.

Et on ce base sur ces connaissances on peut choisir les métas procédés à utiliser pour la réutilisation parmi l'ensemble des métas procédés existants.

III.10. Méta modèle de test centré activité :

Pour les besoins de la réalisation nous définissons des métas procédés centrés activités modélisées en UML.

Le méta procédé de test numéro 1

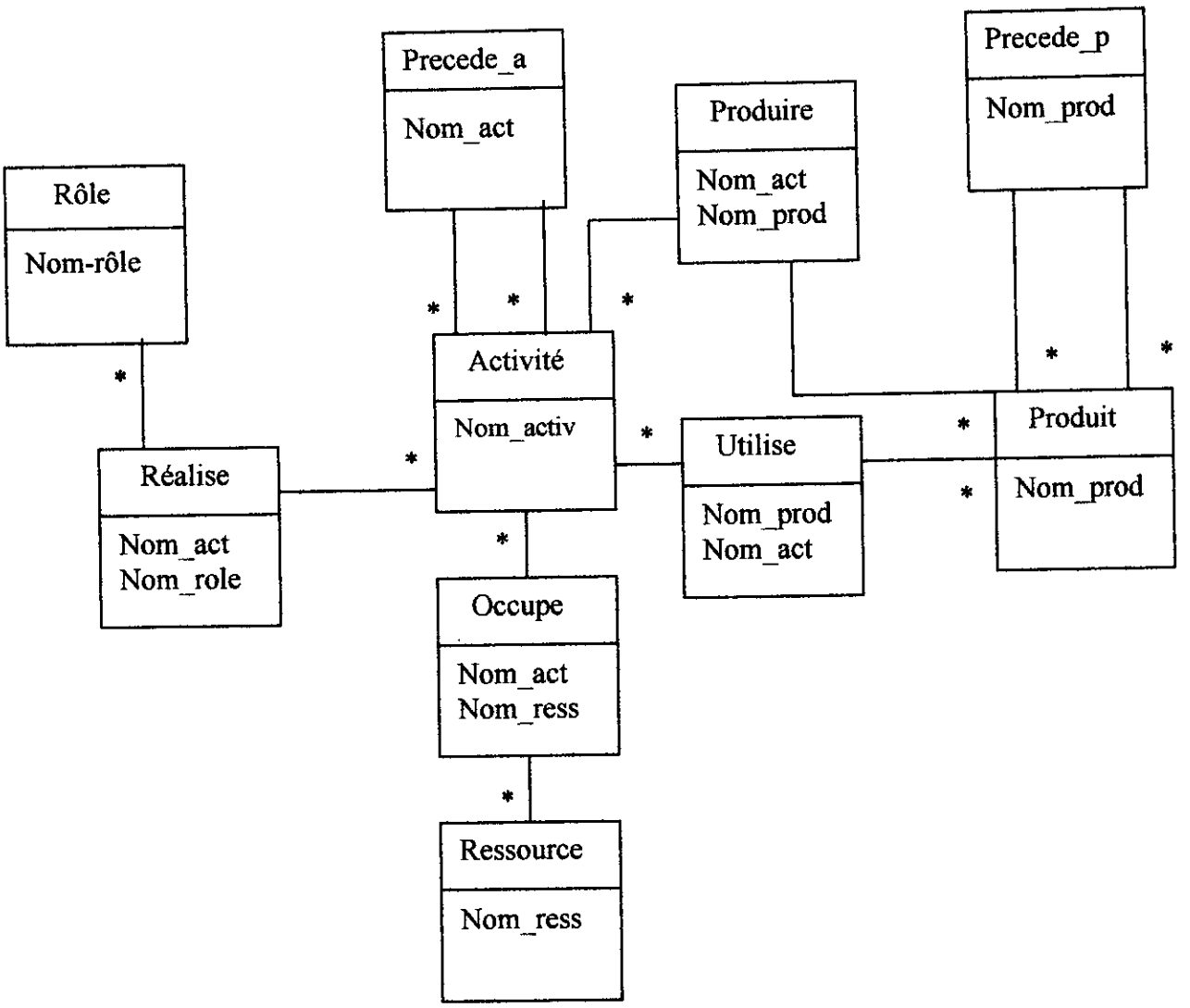


Figure III.09 : Le méta procédé de test numéro 1

Le méta procédé de test numéro 2 :

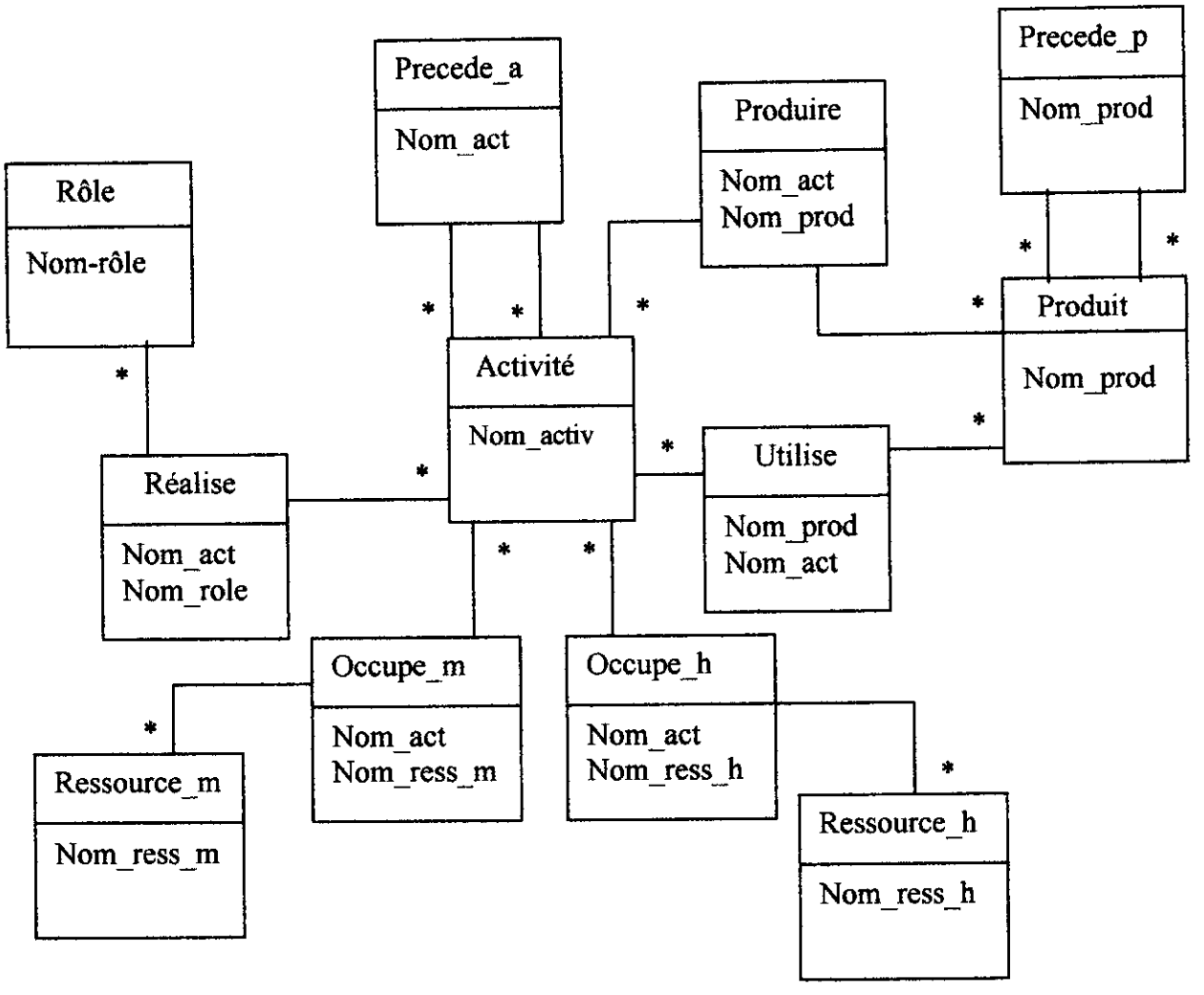


Figure III.10 : Le méta procédé de test numéro 2

Le méta procédé de test numéro 3:

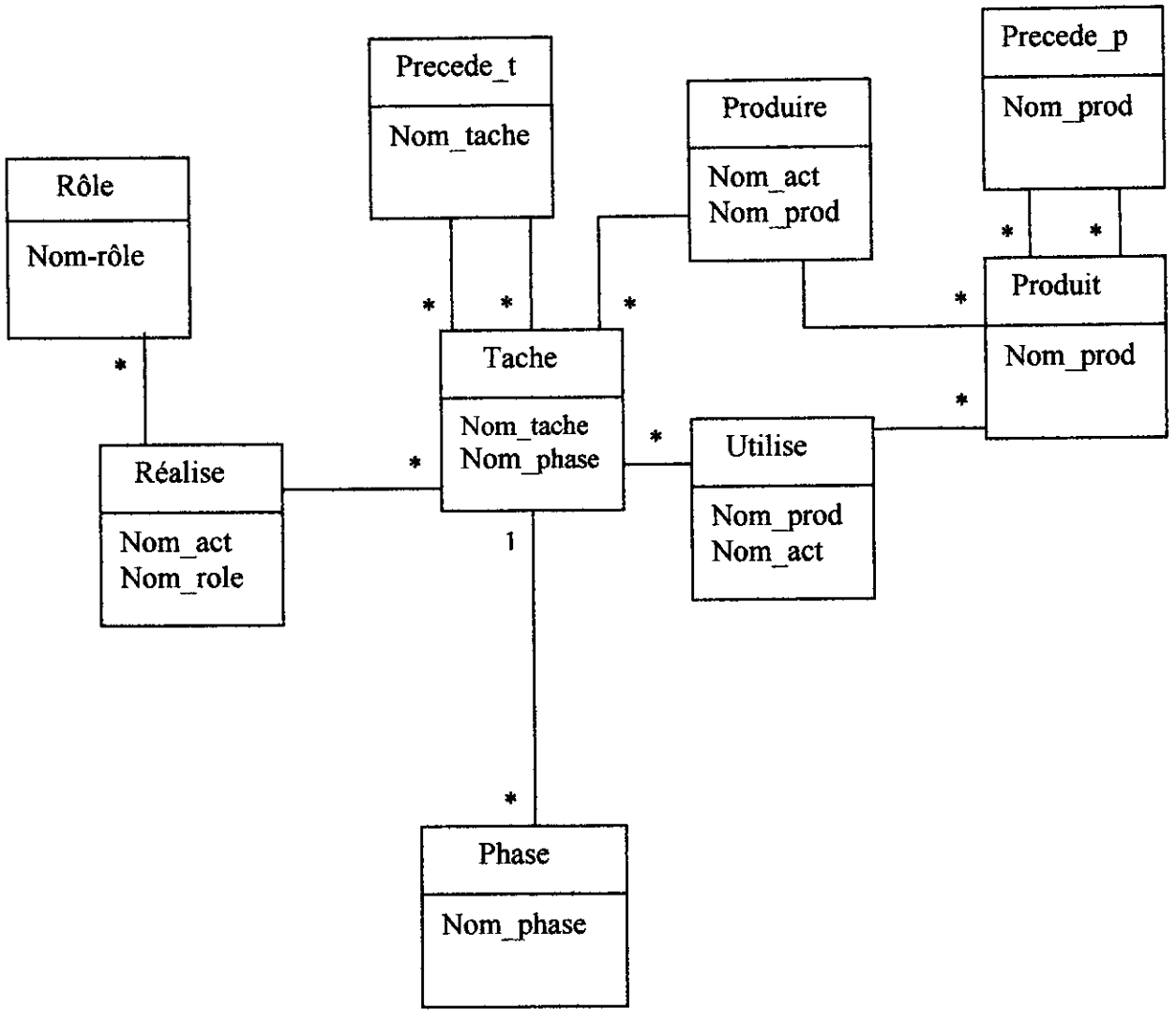


Figure III.11 : Le méta procédé de test numéro 3

Le méta procédé de test numéro 4 :

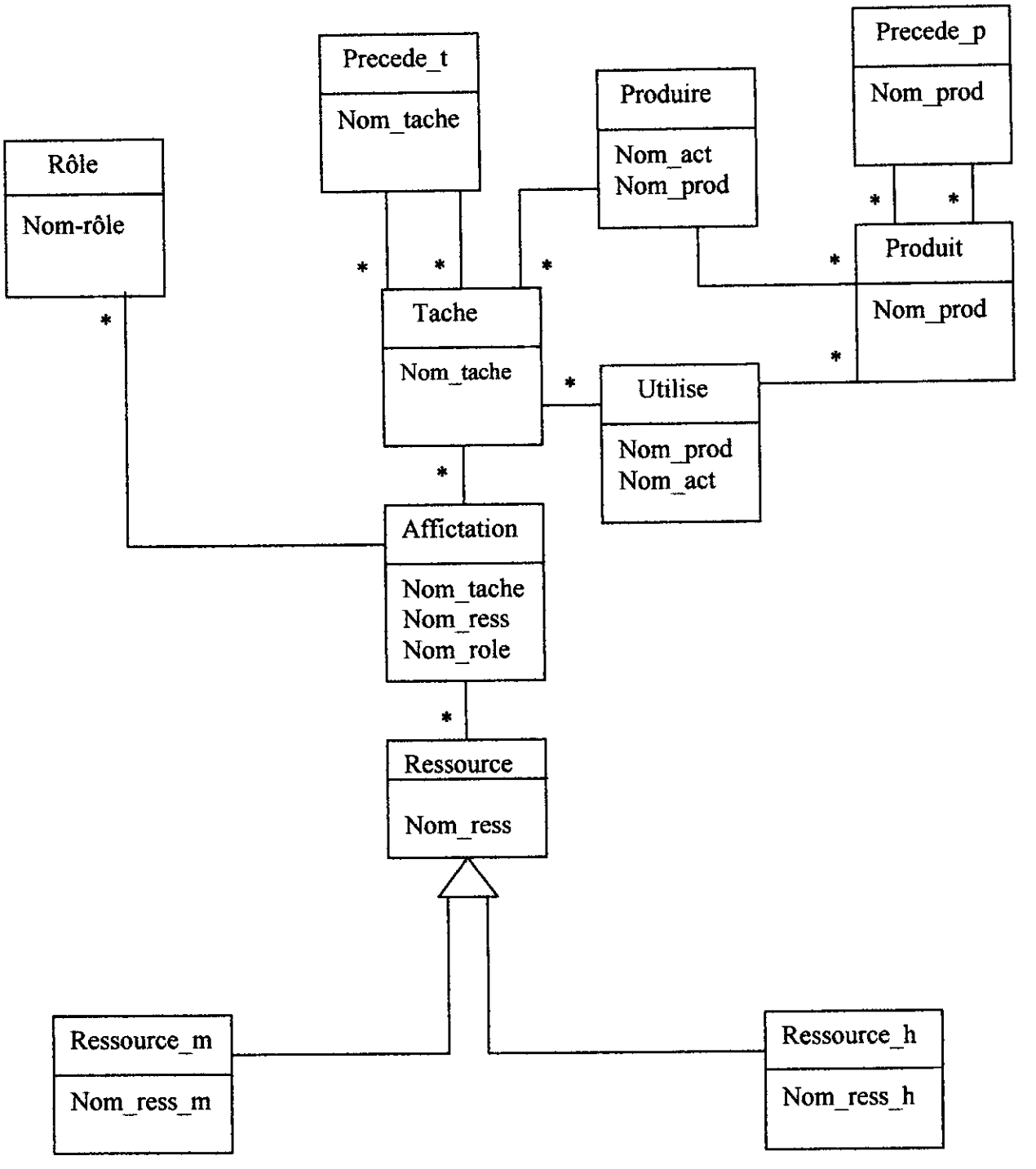
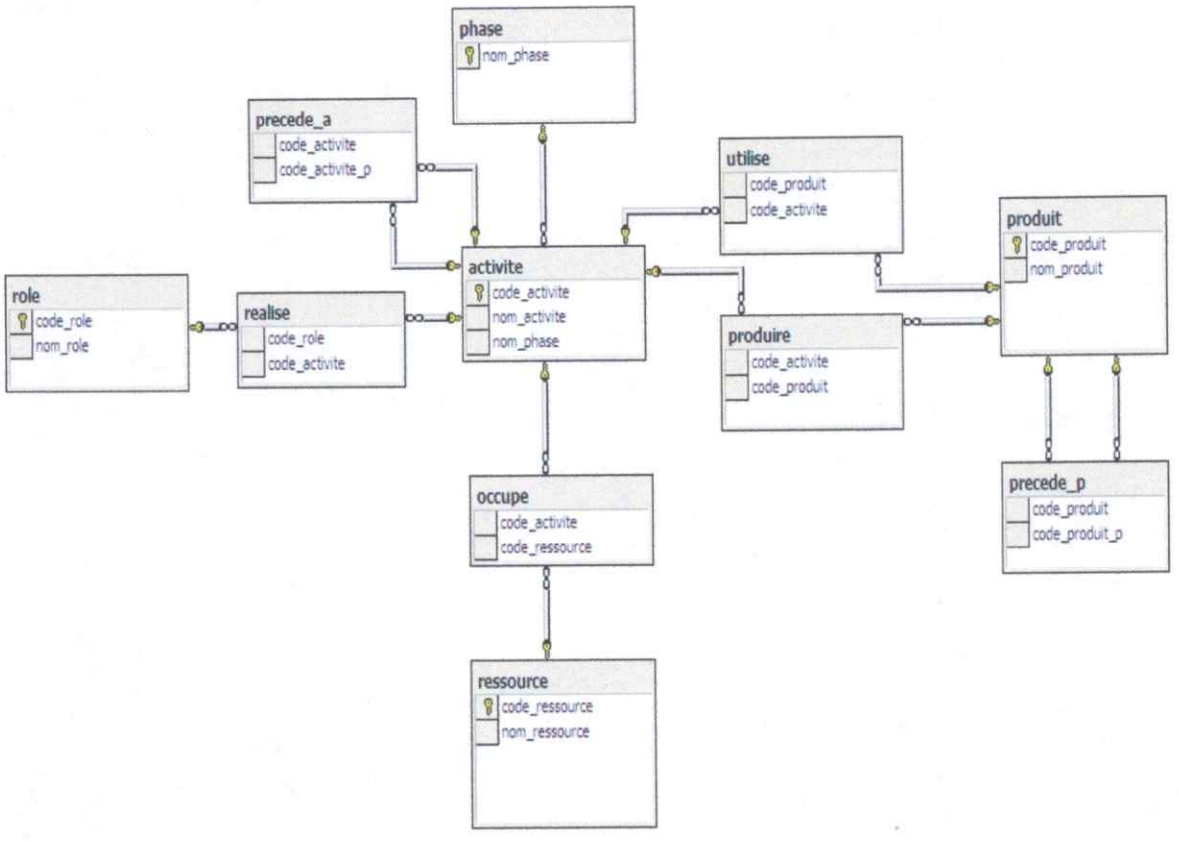


Figure III.12 : Le méta procédé de test numéro 4

Exemple de formalisation :



FigureIII.13: Formalisation de modèle de test 1 dans le SQL Server

CONCLUSION:

Dans cette section nous avons présenté à la première étape les déférents diagrammes de langage de modélisation UML et le processus de développement RUP.

Et dans la deuxième étape nous avons présenté toute définition nécessaire pour réaliser la conception de notre système.

Chapitre III

Partie II

Conception Du Système

Introduction

Dans cette partie nous présenterons les différents diagrammes qui modélisent notre système avec la description détaillée de chaqu'un.

D'après le processus de développement RUP les diagrammes qui modélisent notre système sont :

-*Diagrammes de cas d'utilisation :* ils donnent une vue générale sur le système.
-*Diagrammes de séquence et scénario :* illustrent les séquences entre les différents modules de système.
-*Diagramme de classe :* c'est le cœur d'UML.
-*Diagramme d'état :* pour schématiser les différents états du système.
-*Diagramme d'activité :* pour décrire les interactions entre les modules du système.
-*Diagramme de composant :* pour décrire les différents composants du système.

I - FONCTIONNEMENT DU SYSTEME :

1-Stocker les métas procédés existants, leurs classes et les relations entre classes (réalisé par l'agent stockage)

2-Pour chaque méta procédé faire :
2.1-Rechercher si les classes et les relations existent, alors afficher les numéros des méta procédés trouvés (réalisé par agent de recherche).
2.2-Si non : ce fragment n'existe pas dans nos bases de données, et il faut le construire manuellement.

3-Stocker les phases et leurs fragments pour chaque méta procédé trouvé (réalisé par l'agent stockage).

4.1-L'agent identifiant sélectionne le fragment qui est du type demander s'il existe.
4.2-Sinon: il faut construire ce fragment manuellement.

5.1-Si la recherche des fragments à réutiliser est terminée aller á 6.
5.2-Sinon aller á 2.

6-Coller les fragments trouvés dans un ordre séquentiel croissant (réalisée par agent montage).

7-Vérification et validation.

II. DIAGRAMMES DES CAS D'UTILISATION ET DE SEQUENCES :

Diagramme de cas d'utilisation Globale du système :

Il contient les taches principales de réutilisation réalise dans notre système.

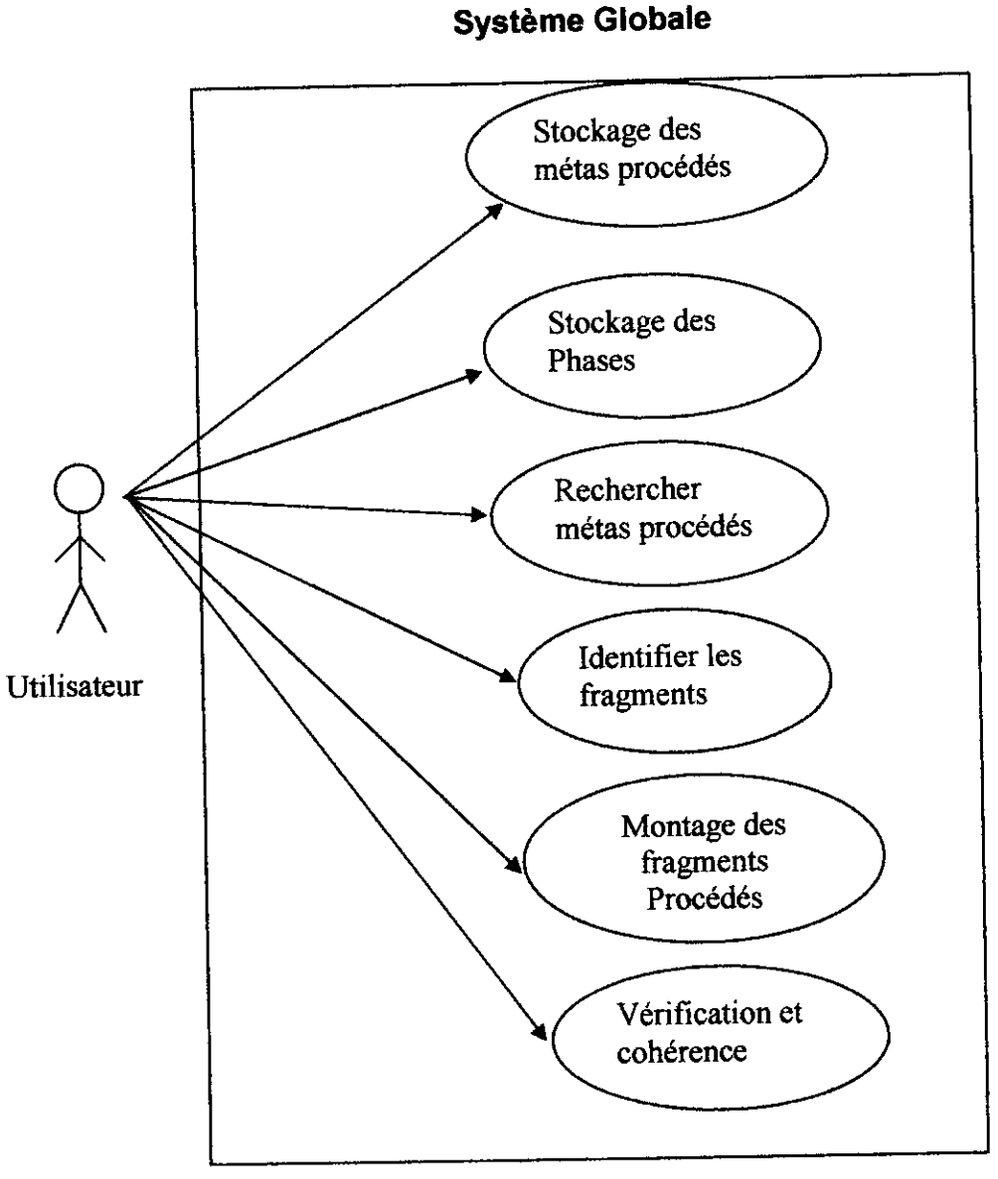
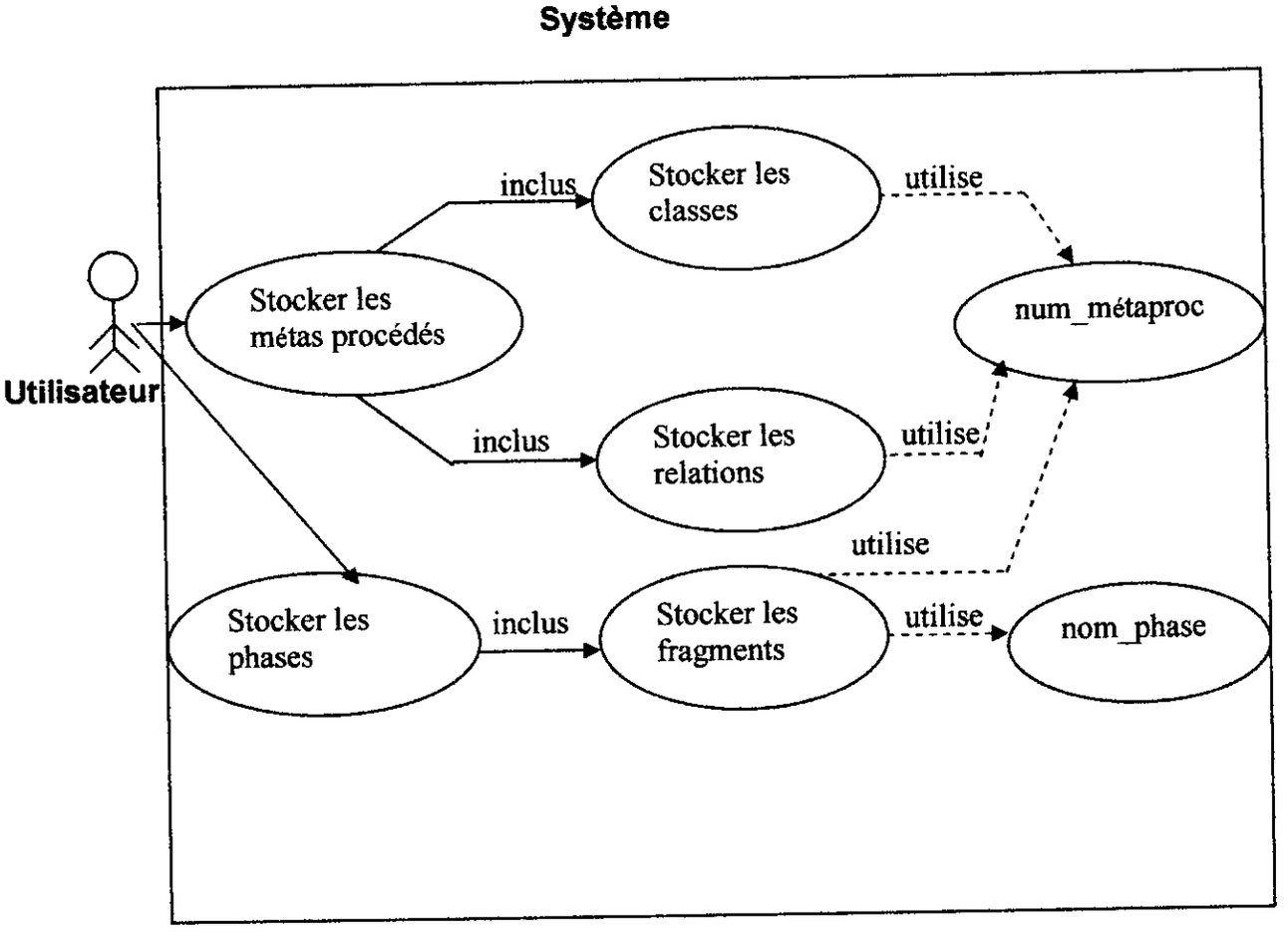


Figure III.14. Diagramme de cas d'utilisation globale du système

Chaque tâche de diagramme de cas d'utilisation globale est bien détaillé dans se qui suit.

II.1. Diagramme de cas d'utilisation pour le stockage des métas procédés et phases :

Ce diagramme est destiné à stocker les métas procédés et leurs phases.
Pour stocker les métas procédés il faut stocker les classes et les relations en parallèle en utilisant le numéro de métas procédé. Pour stocker les phases cette étape inclus le stockage des fragments procédés logiciel en utilisant le numéro de méta procédé et le nom de la phase.
La figure si dessous schématise les différentes étapes.



FigureIII.15 : Diagramme de cas d'utilisation pour le stockage

1. Diagramme de séquence pour le stockage des métas procédés :

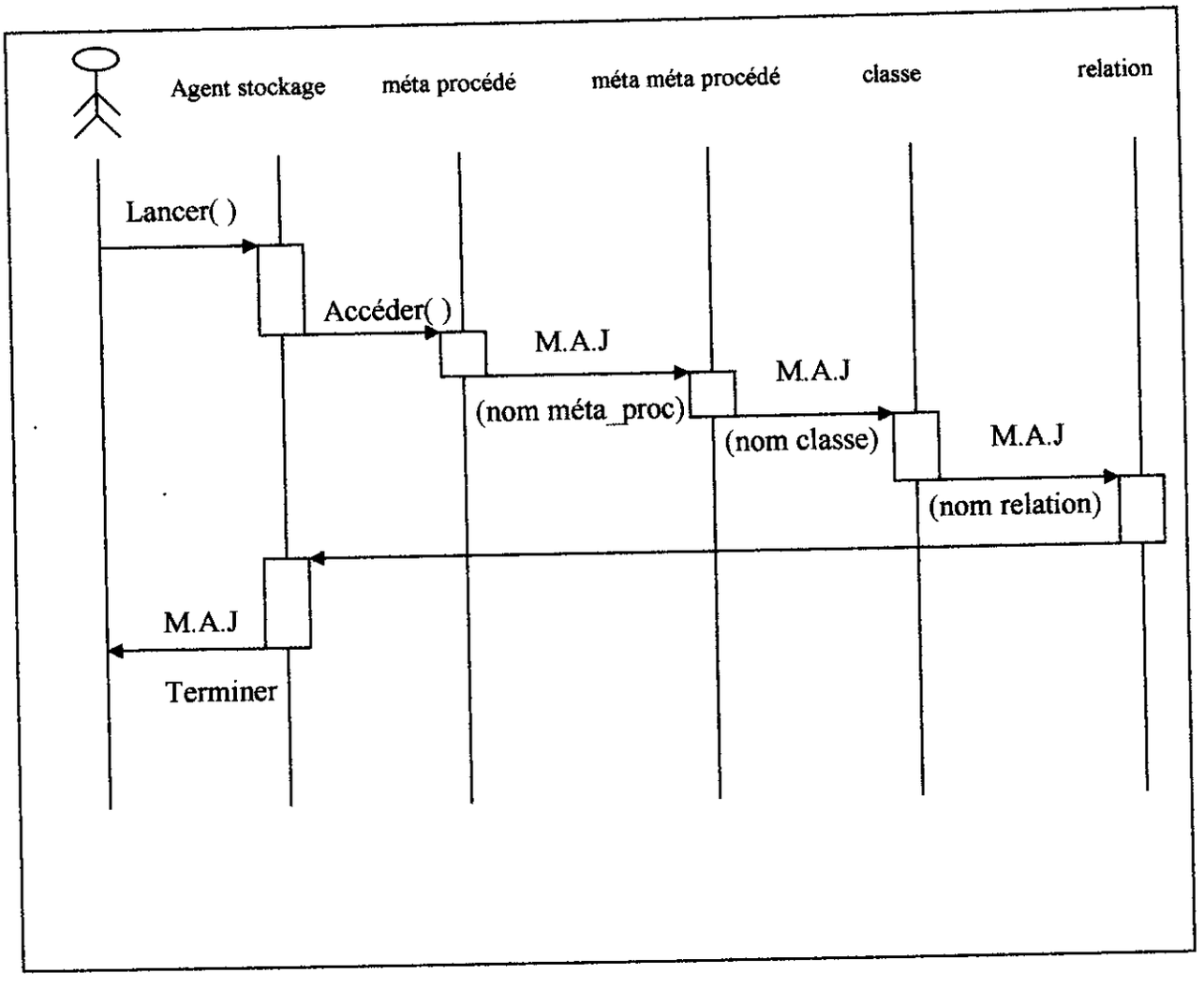


Figure III.16: Diagramme de séquence pour le stockage des métas procédés.

Scénario de diagramme de séquence pour le stockage des métas procédé :

1- L'utilisateur lance l'agent stockage.
2- L'agent stockage accède à la base de données.
3- L'agent stockage met à jour la classe Méta procédé.
4- L'agent stockage met à jour la classe Classe.
5- L'agent stockage met à jour la table Relation.
6- L'agent stockage envoie un message à l'utilisateur pour l'informer que le stockage est terminé.

2. Diagramme de séquence pour le stockage des phases.

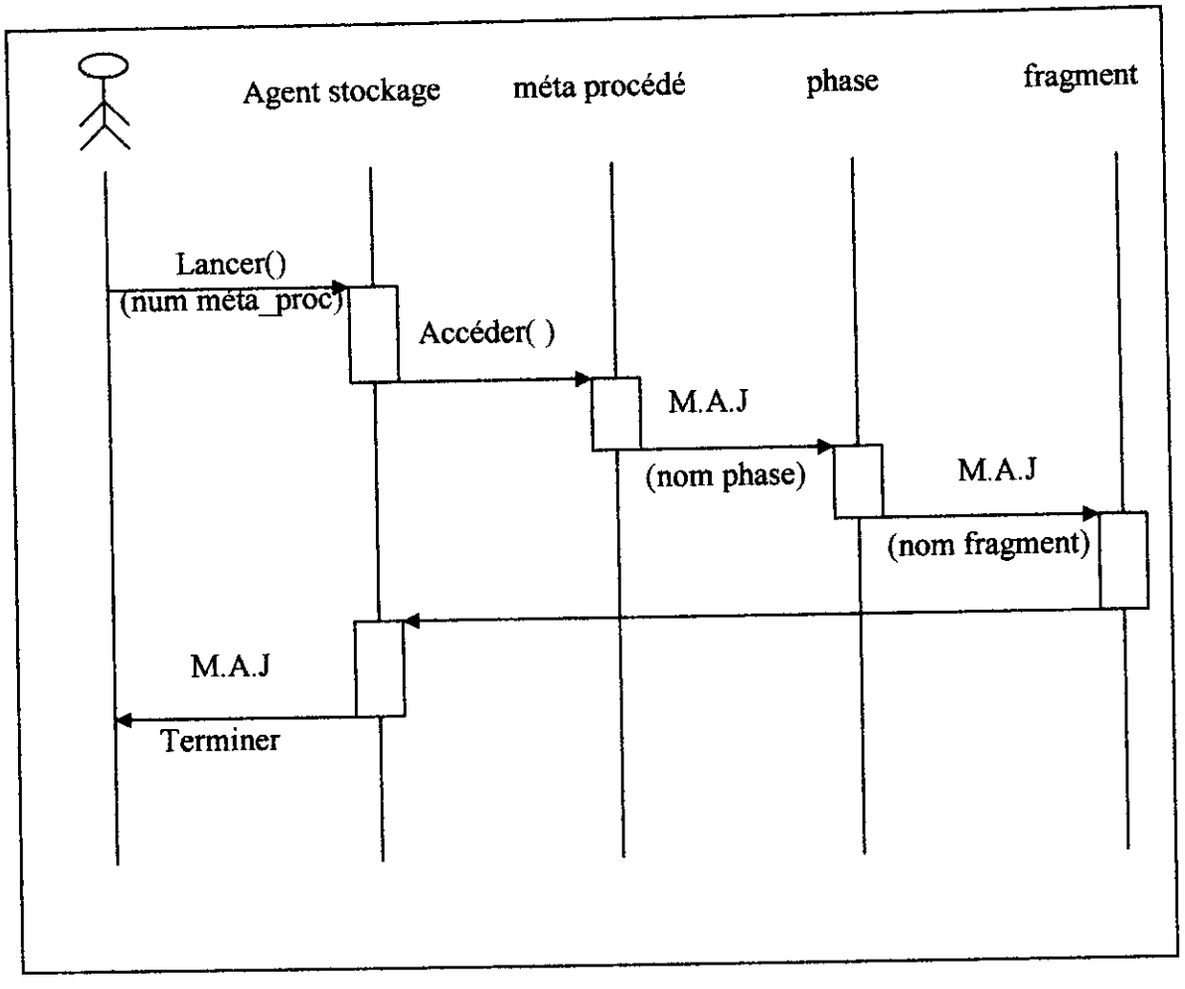


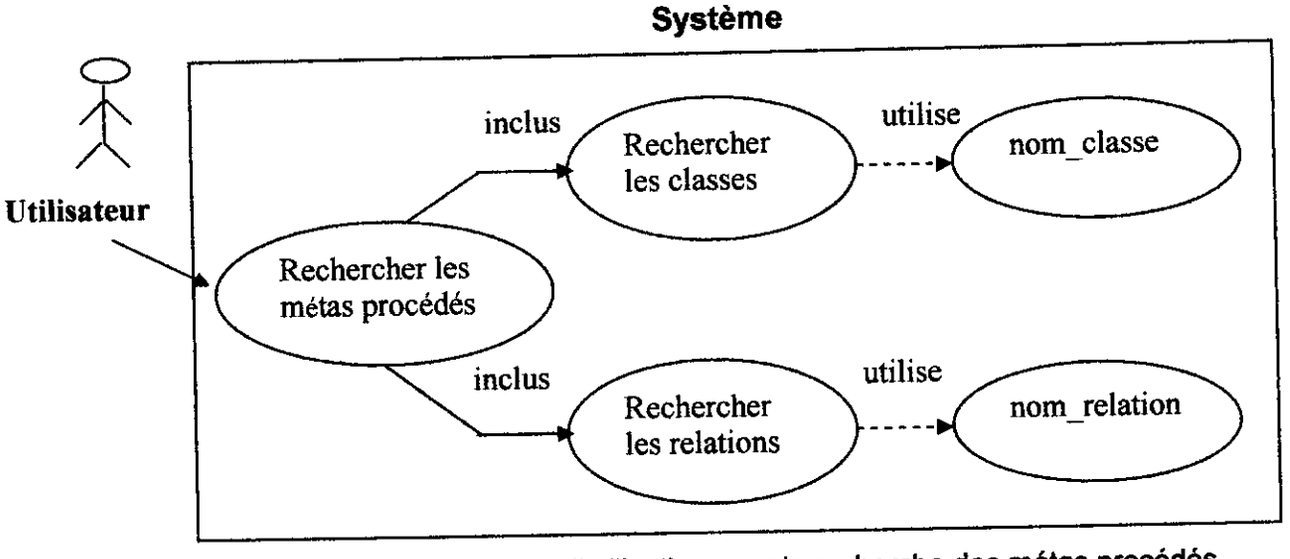
Figure III.17 : Diagramme de séquence pour le stockage des phases.

Scénario de diagramme de séquence pour le stockage des phases. :

1- L'utilisateur lance l'agent stockage.
2- L'utilisateur lance l'agent stockage avec le paramètre (num_méta proc).
3- L'agent stockage accède à la base de donnée.
4- L'agent stockage met à jour la table phase.
5- L'agent stockage met à jour la table fragment.
6-L'agent stockage envoie un message à l'utilisateur pour l'informer que le stockage est terminé.

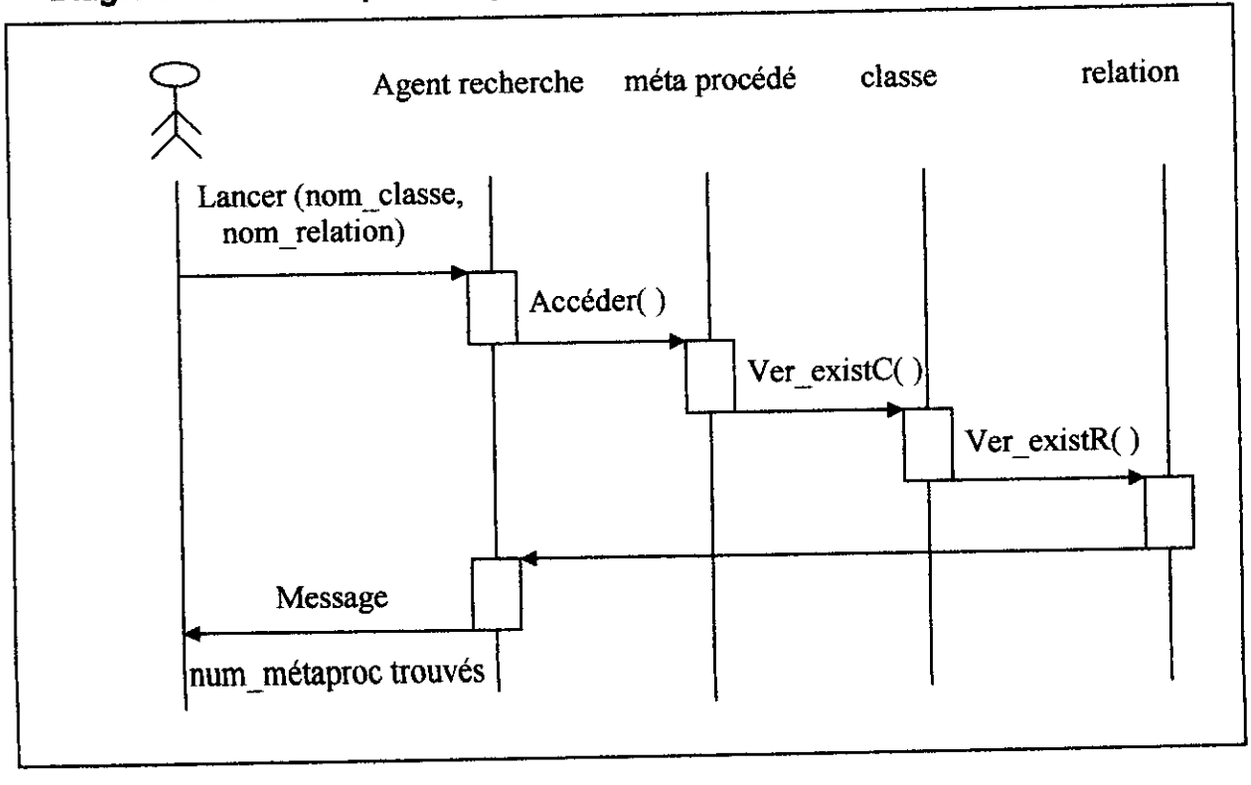
II.2- Diagramme de cas d'utilisation pour la recherche des métas procédés:

Pour rechercher un méta procédé à réutiliser l'agent recherche doit chercher les classes et les relations entre ces classes en utilisant les noms des classes et les relations.
Si ses deux dernières existent alors ce méta procédé peut être réutilisé.



FigureIII.18 : Diagramme de cas d'utilisation pour la recherche des métas procédés.

- Diagrammes des séquences pour la recherche des métas procédés:

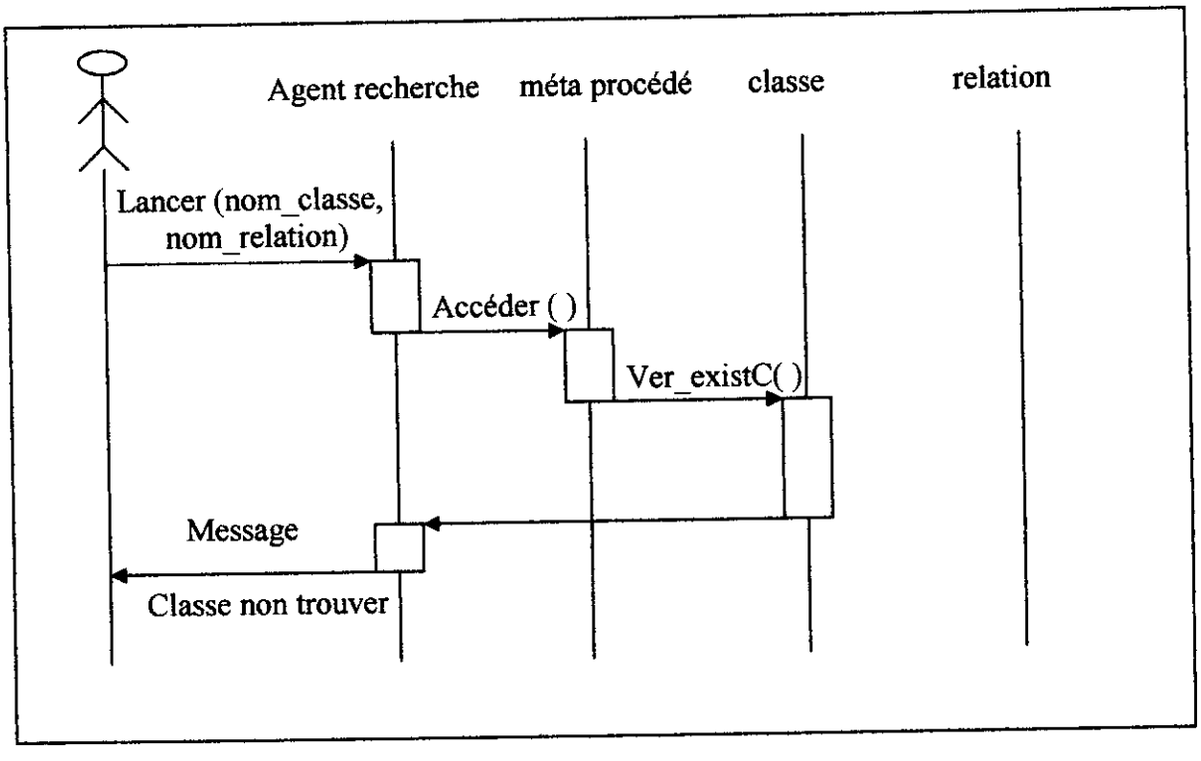


FigureIII.19 : Diagramme de séquence pour la recherche des métas procédés.

Scénario de diagrammes des séquences pour la recherche des métas procédés :

1- L'utilisateur lance l'agent recherche avec les paramètres (nom_classe, nom_relation).
2- L'agent recherche accède à la classe méta procédé.
3- L'agent recherche accède à la table classe et vérifie l'existence de la classe demander.
4- L'agent recherche accède à la table relation et vérifie l'existence de la relation demandée.
5- L'agent recherche envoie un message à l'utilisateur et lui affiche les numéros des métas procédés trouvés.

- **Diagrammes des séquences pour la recherche on cas d'exception 1 :**



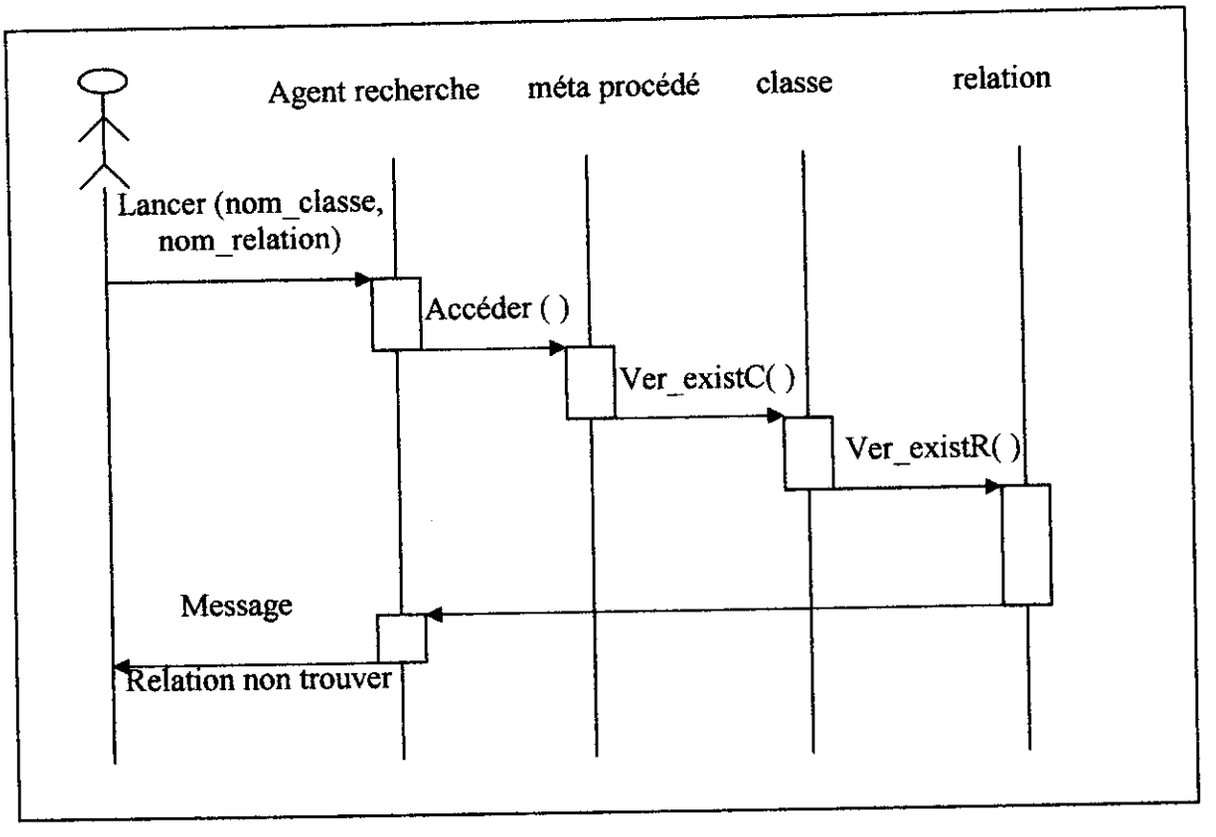
FigureIII.20 : Diagrammes des séquences pour la recherche on cas d'exception 1.

Scénario de diagramme de séquence pour la recherche on cas d'exception1:

1- L'utilisateur lance l'agent recherche avec les paramètres (nom_classe, nom_relation).
2- L'agent recherche accède à la table méta procédé.
3- L'agent recherche accède à la table classe et vérifie l'existence de la classe demande.

4- La classe à vérifier non trouvée, l'agent recherche envoie un message à l'utilisateur pour lui informer.
5- L'utilisateur construit ce fragment manuellement.

- Diagramme de séquence pour la recherche on cas d'exception 2 :



FigureIII.21: Diagrammes des séquences pour la recherche on cas d'exception 2.

Scénario de diagramme de séquence pour la recherche on cas d'exception2:

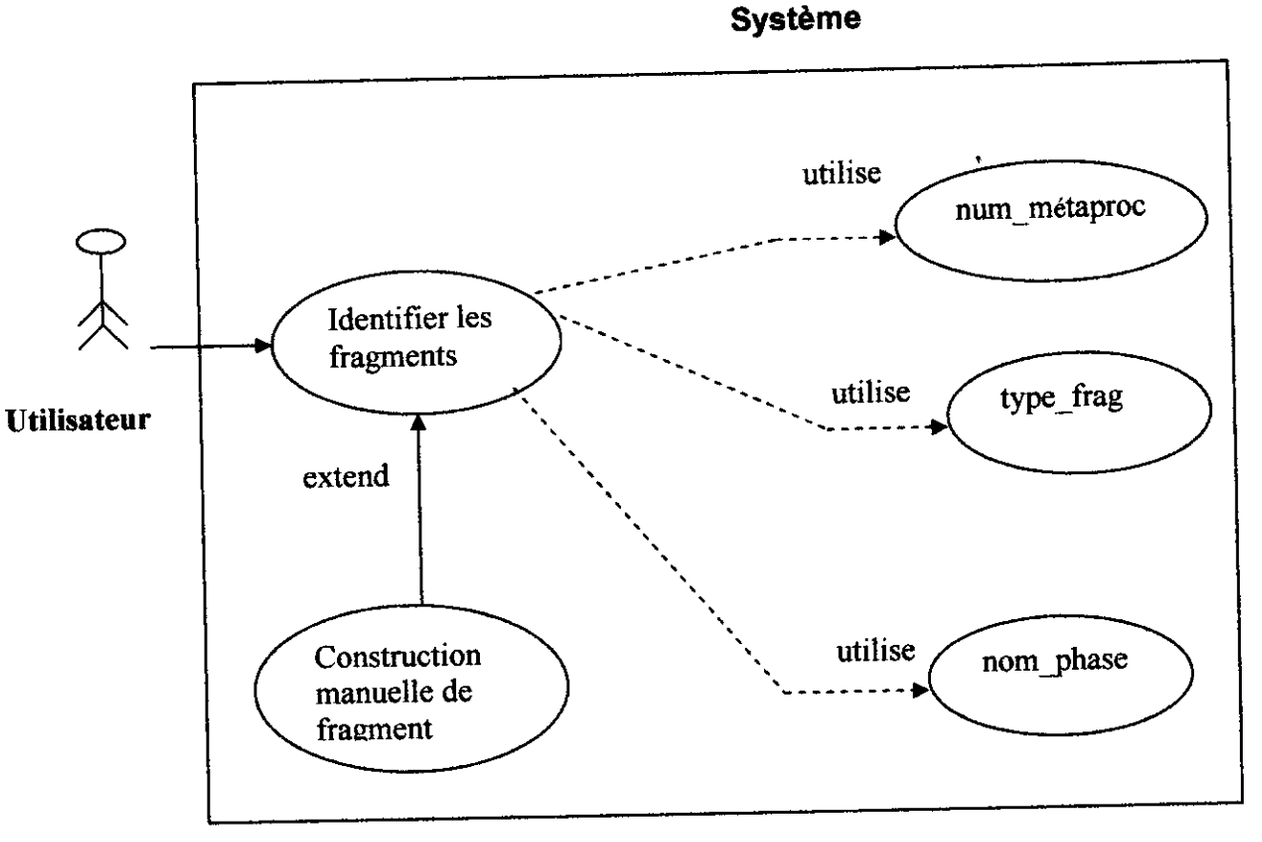
1- L'utilisateur lance l'agent recherche avec les paramètres (nom_classe, nom_relation).
2- L'agent recherche accède à la table méta procédé.
3- L'agent recherche accède à la table classe et vérifie l'existence de la classe demandée.
4- L'agent recherche accède à la table relation et vérifie l'existence de la relation demande.
5- La relation à vérifier non trouvée, l'agent recherche envoie un message à l'utilisateur pour l'informer.
6- L'utilisateur construit ce fragment manuellement.

II.3- Diagramme de cas d'utilisation pour l'identification des fragments:

Ce diagramme illustre les étapes exécuté par l'agent identifie pour déterminer l'ensemble des fragments procédé logiciel accepté pour la réutilisation.

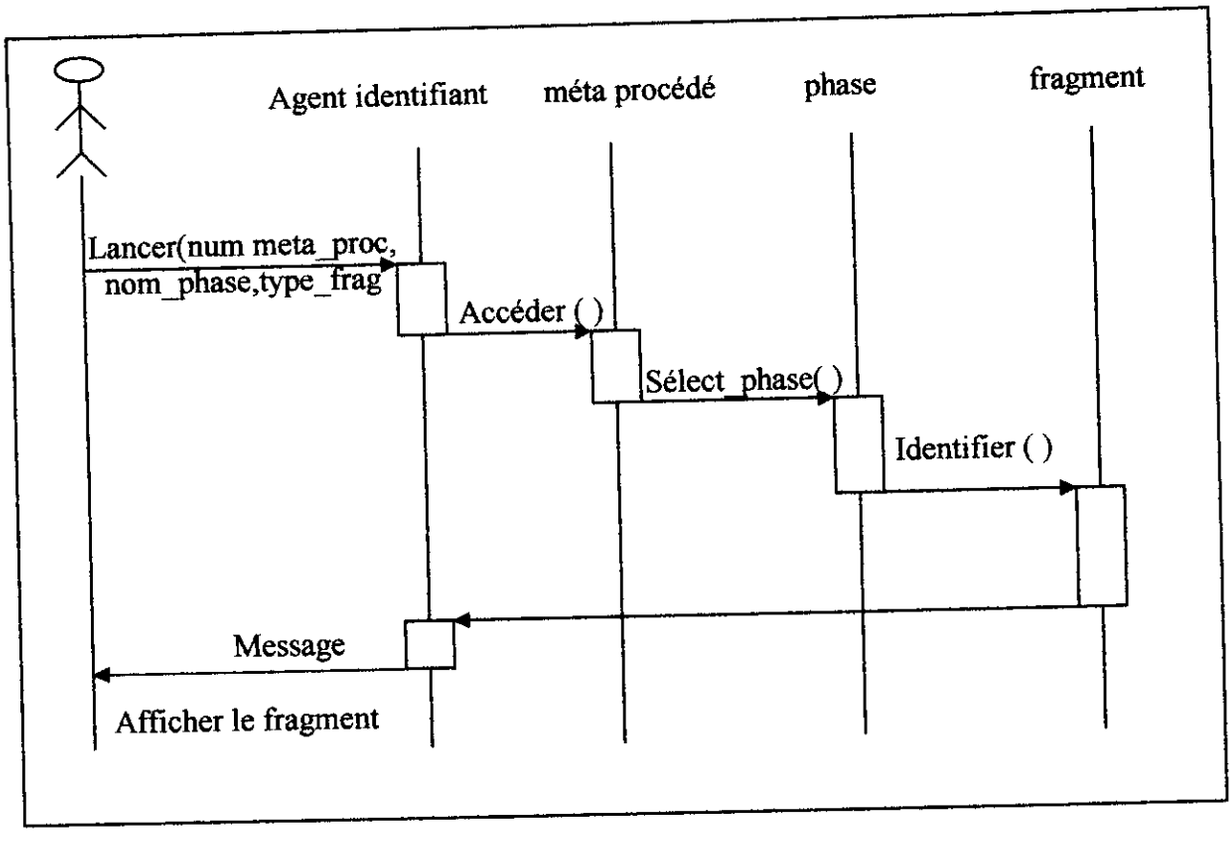
Il utilise comme paramètre d'identification :
-Nom de méta procédé.
-Type de fragment.
-Nom de la phase.

Si le résultat de l'identification est négatif, l'agent identifie donne la main à l'utilisateur pour insérer ce fragment manuellement.



FigureIII.22 : Diagramme de cas d'utilisation pour l'identification des fragments

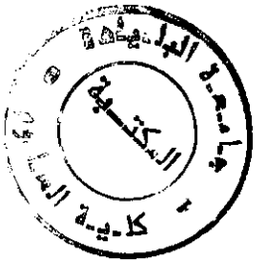
- Diagrammes des séquences pour l'identification des fragments:



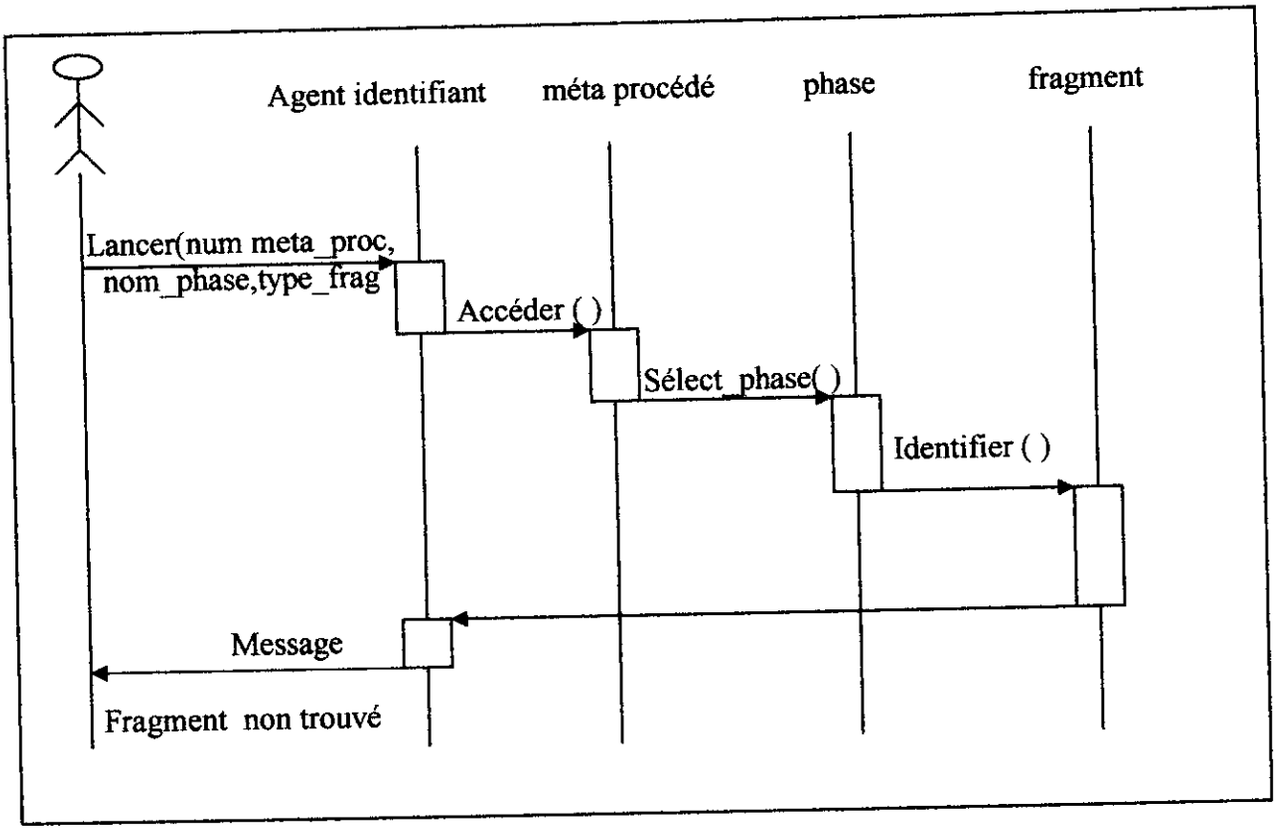
FigureIII.23 : Diagramme de séquence pour l'identification des fragments.

Scénario de diagramme de séquence pour l'identification des fragments:

1- L'utilisateur lance l'agent identifiant avec les paramètres (num_métaproc, nom_phase, type_frag).
2- L'agent identifiant sélectionne la phase demander dans la table phase.
3- L'agent identifiant identifié le type de fragment demandé.
4- L'agent identifiant envoie un message à l'utilisateur avec le champ de fragment valide.



Diagrammes des séquences pour l'identification en cas d'anomalie



FigureIII.24: Diagramme de séquence pour l'identification des fragments en cas d'anomalie.

Scénario de diagramme de séquence pour l'identification en cas d'anomalie:

1- L'utilisateur lance l'agent identifiant avec les paramètres (num_métaproc, nom_phase, type_frag).
2- L'agent identifiant sélectionne la phase demandée dans la table phase.
3- L'agent identifiant identifier le type de fragment demandée.
4- L'agent identifiant envoie un message à l'utilisateur pour l'informer que le type de fragment demandée n'existe pas.
5- L'utilisateur construit ce fragment manuellement.

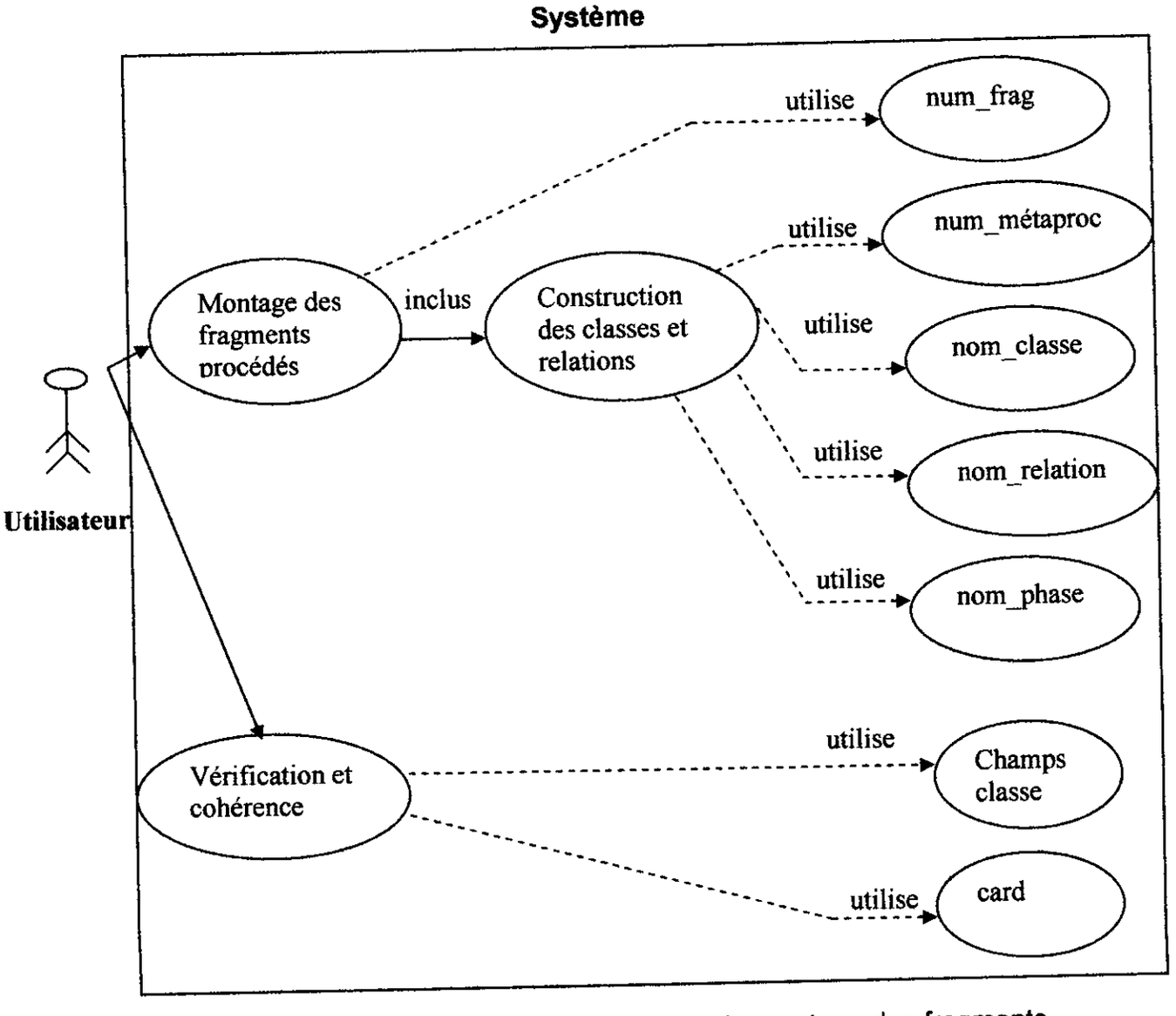
II.4. Diagramme de cas d'utilisation pour le montage des fragments:

Cette étape est réalisée par l'agent coller, elle est composée de trois partie :
- Montage des fragments procédés logiciels.
-Dessin de diagramme de classe.
-Vérification et cohérence.

Dans la première partie l'agent utilise le numéro de fragment pour classer les fragments déjà identifies dans la phase précédente et le résultat sera le nouveau procédé logiciel.

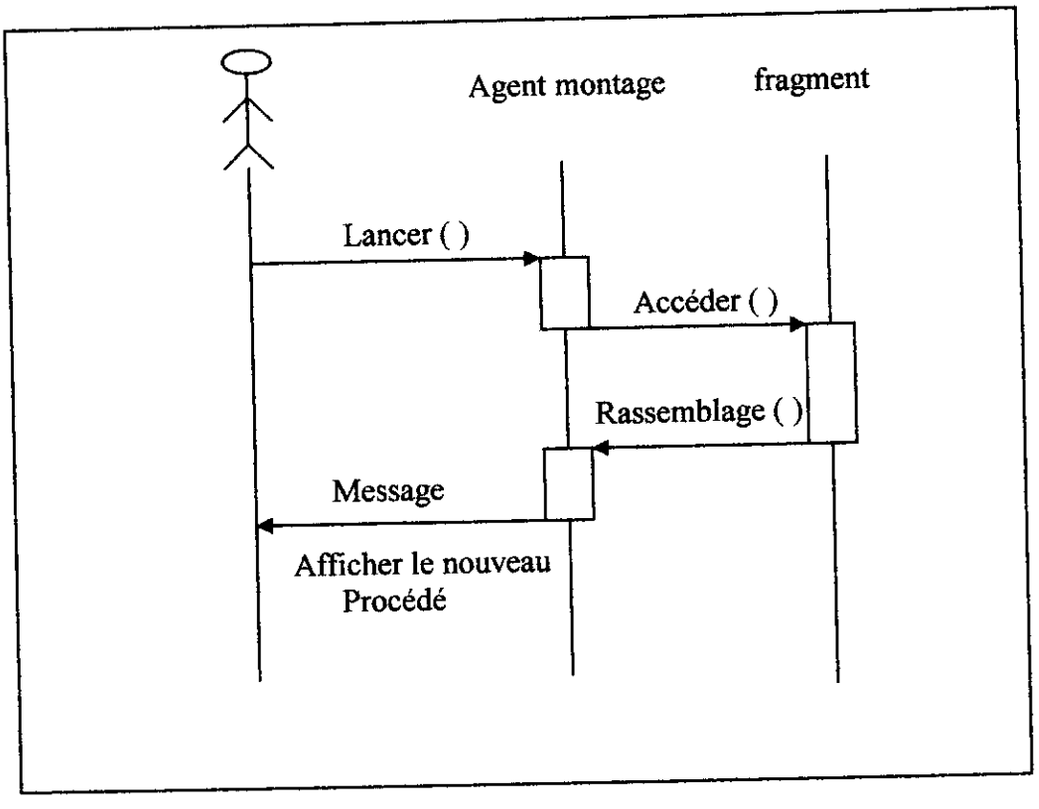
En parallèle, l'agent construit le diagramme de classe associé à ce procédé logiciel.

Comme dernière partie, l'agent réalise une vérification pour valider le résultat



FigureIII.25 : Diagramme de cas d'utilisation pour le montage des fragments

- Diagrammes des séquences pour le montage:

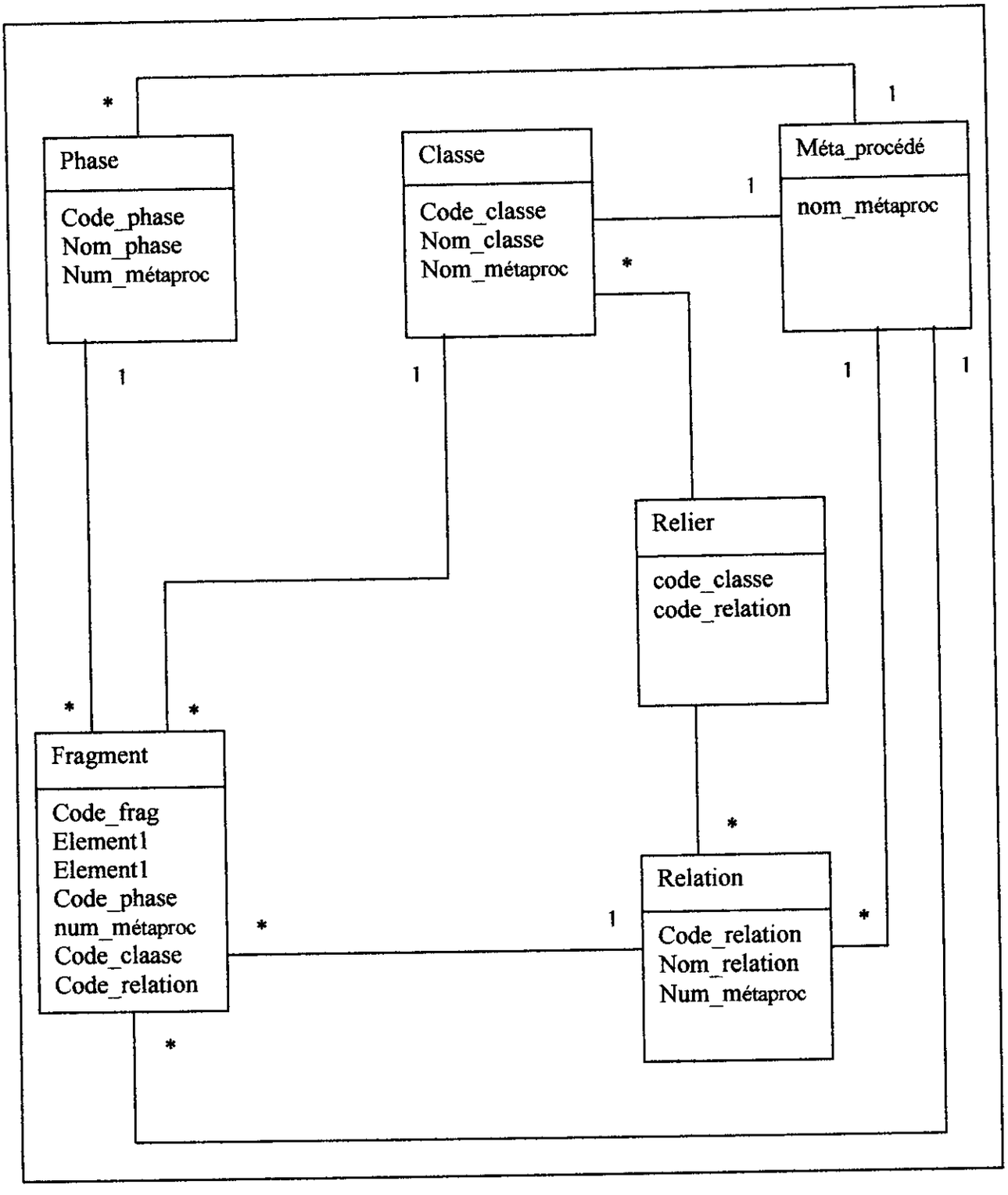


FigureIII.26 : Diagrammes des séquences pour le montage.

Scénario de diagramme de séquence pour le montage :

1- L'utilisateur lance l'agent montage.
2- Agent montage accède à la table des fragments valides qui contient les fragments de nouveaux procédés à construire.
3- Agent montage colle les fragments (l'ordre est prédéfini par le processus de recherche).
4- Agent montage affiche notre nouveau procédé à l'utilisateur.

III -Diagramme de classe :



FigureIII.27 : Diagramme de classe de la base de donnée.(méta méta modèle niveau 3 de l'OMG) .

III.1. Explication de diagramme de classe :

Notre méta méta procédé contient les tables suivantes :

- une table méta procédé pour stocker les noms des méta procédés à réutilisé.
- Une table phase pour stocker les noms des phases appartient à chaque méta procédé.
- Une table classe pour stocker les noms des classes de chaque méta modèle.
- Une table relation pour stocker les noms des relations de chaque méta modèle.
- Une table fragment pour stocker les fragments de chaque méta modèle par apport les phases (pour chaque phase de chaque méta modèle).

III.2. Règles de gestion :

- un méta procédé peut avoir plusieurs phases.
- un méta procédé peut avoir plusieurs classes.
- un méta procédé peut avoir plusieurs relations.
- un méta procédé peut avoir plusieurs fragments.
- Une phase peut avoir plusieurs fragments.
- Une classe relier a une au plusieurs relation.
- Une classe peut avoir plusieurs fragments.
- Un fragment appartient à une seule classe.
- Un fragment appartient à une seule relation.
- Une phase appartient à un seul méta procédé.
- Une classe appartient à un seul méta procédé.
- Une relation appartient à un seul méta procédé
- Une relation peut relie une ou plusieurs classes
- Une relation peut avoir un ou plusieurs fragments
- Un fragment appartient à une seule phase.
- Un fragment appartient à une seule classe.
- Un fragment appartient à une seule relation.
- Un fragment appartient à un seul méta procédé.

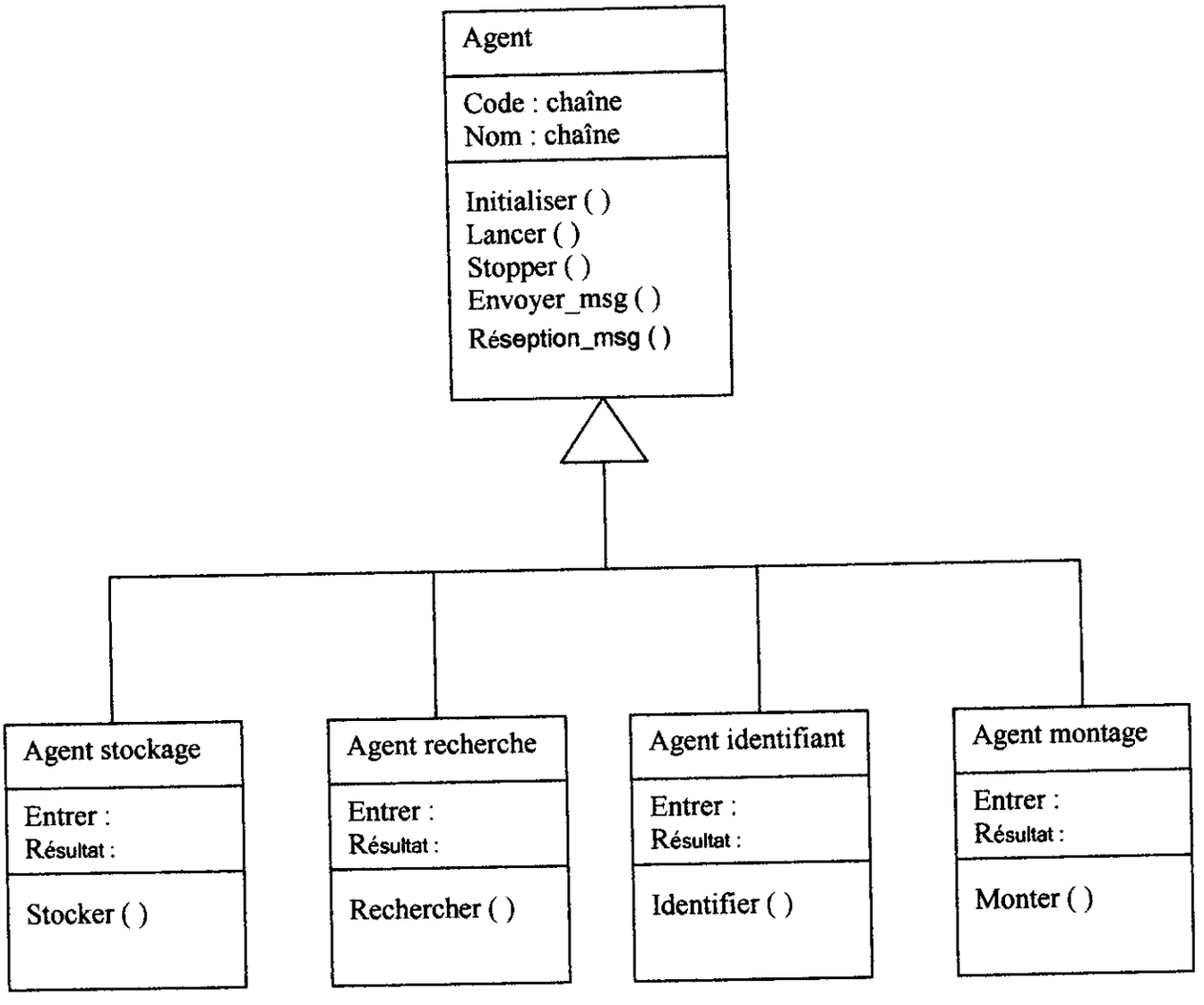
IV- Diagramme de classe des agents :

Description de diagramme des agents:

Notre système multi agent est construit par quatre types d'agents (agent stocké, recherché, identifié, collé), chaque type est représenté par une classe de code source C++. Les agents intèrchangent des messages entre aux pour assurer la communication entre les éléments du système. Nous définissons des méthodes comme Envoyer msg () et Réception msg () ;...etc.

Pour que l'utilisateur puisse gérer le système et pour que nous assurions la communication entre le système et l'utilisateur nous définissons les méthodes lancer () ; stopper () ; pour déclencher ou arrêter un agent. Ces fonctions

précédentes sont généralisées pour tous les éléments de système (agents) comme il existe d'autres méthodes spécifiques pour chaque agent, ces méthodes sont dépendantes des taches réalisées par l'agent.

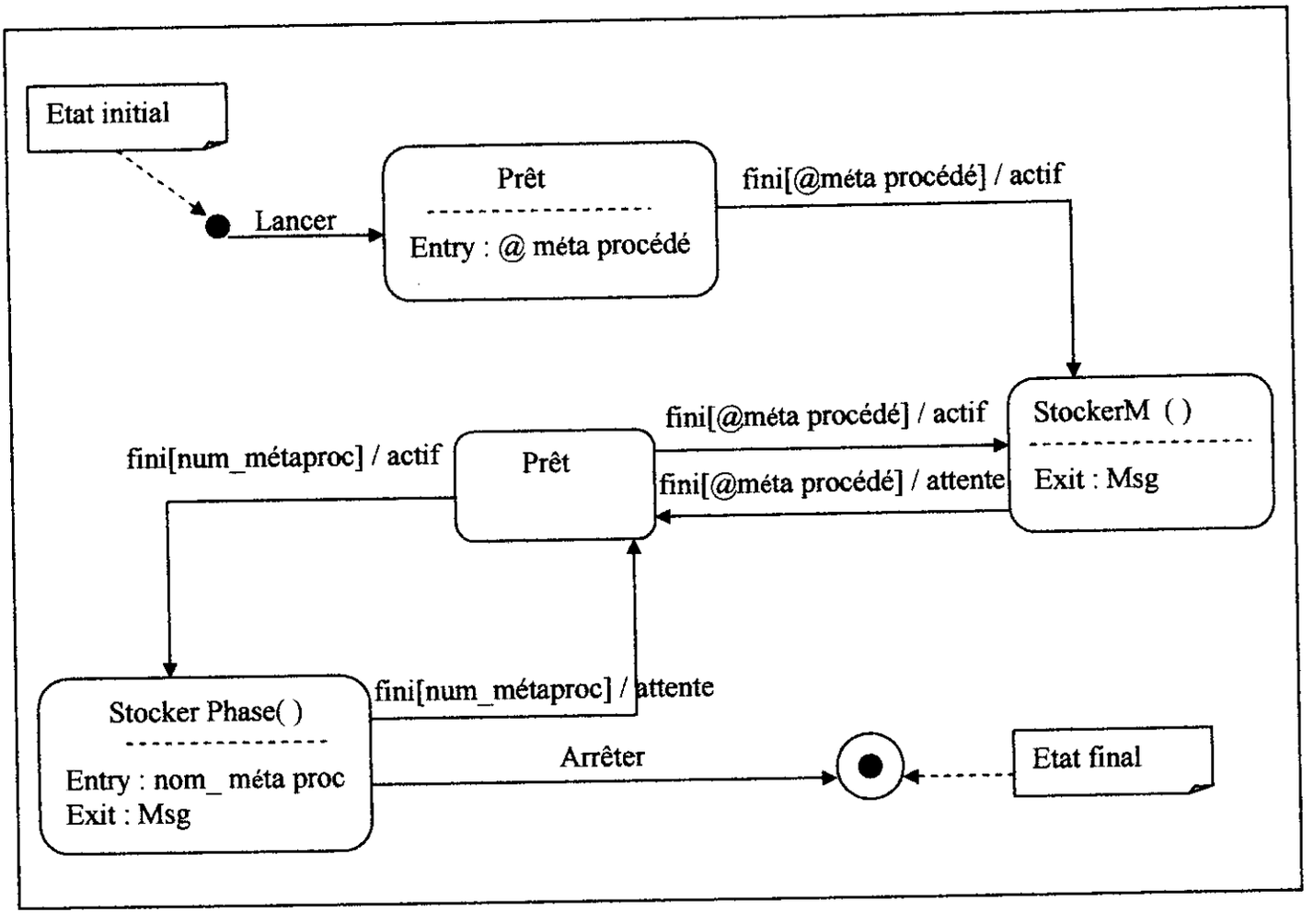


FigureIII.28: Diagramme des agents

V- DIAGRAMMES D'ETATS :

V.1. Diagramme d'état pour l'agent stockage :

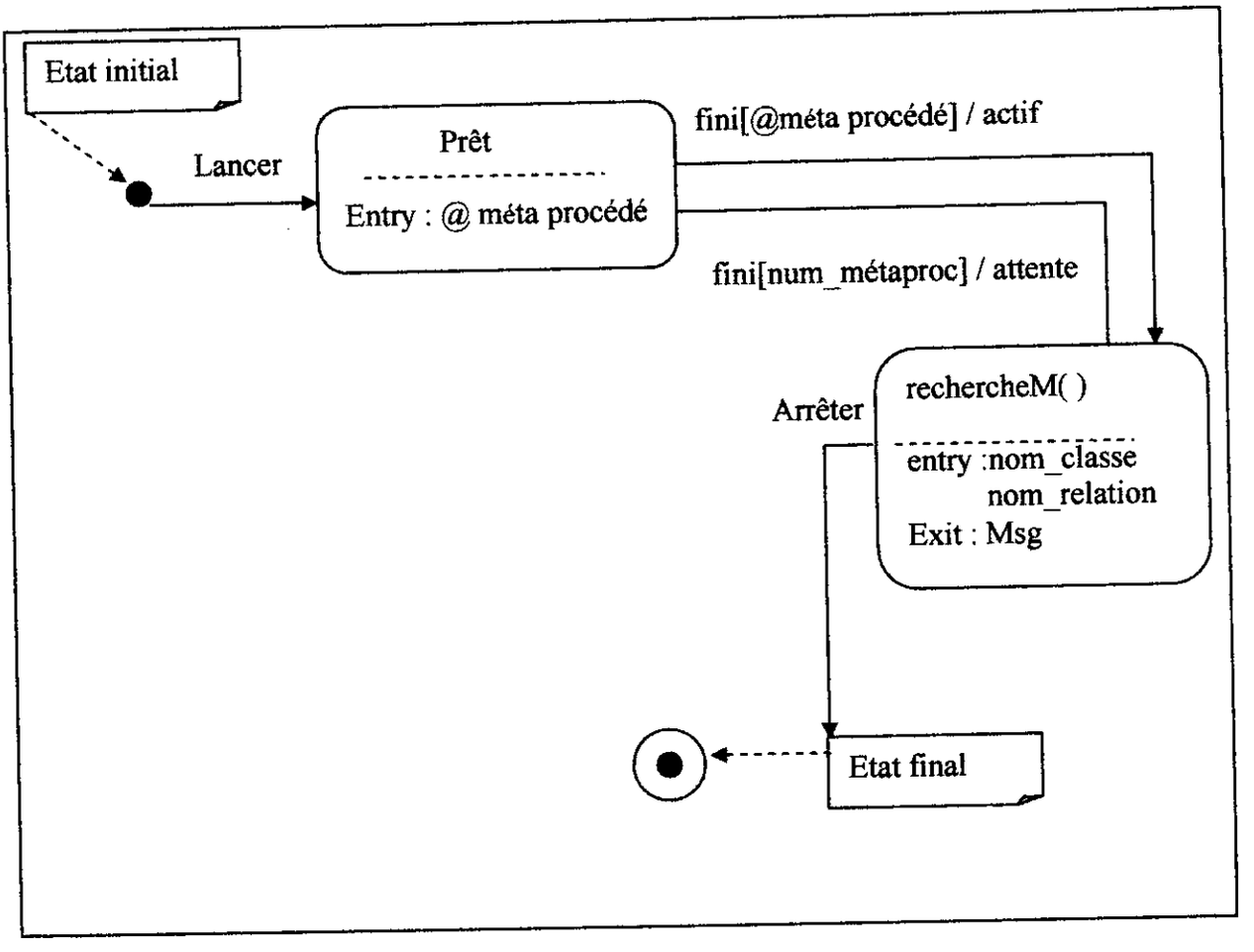
Pour le stockage des méta procédés et les phases l'agent stockage passe par trois principaux états, le premier c'est l'état ou l'agent est actif pour stocker les métas procédés
(StockerM( ))et le deuxième c'est pour stocker les phases (Stocker Phase( ))et l'état intermédiaire c'est l'état prêt pour réaliser des mises à jour si nécessaire.



FigureIII.29 : Diagramme d'état pour l'agent stockage

V.2. Diagramme d'état pour l'agent Recherche :

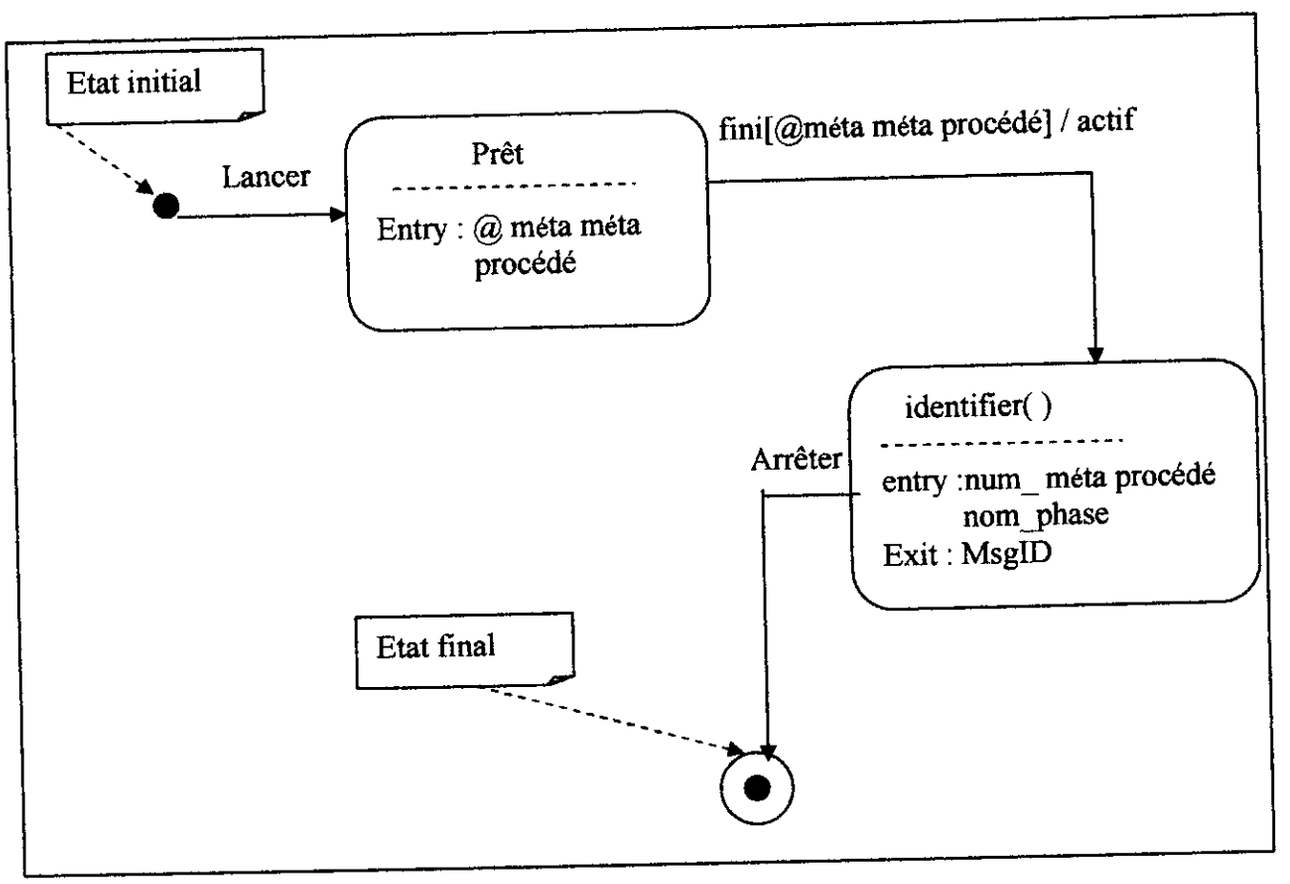
Dans ce cas l'agent recherche à deux états, soit il est prêt et il attend des ordres pour les exécuter soit il est en train de rechercher des métas procédés.



FigureIII.30: Diagramme d'état pour l'agent Recherche

V.3. Diagramme d'état pour l'agent identifie :

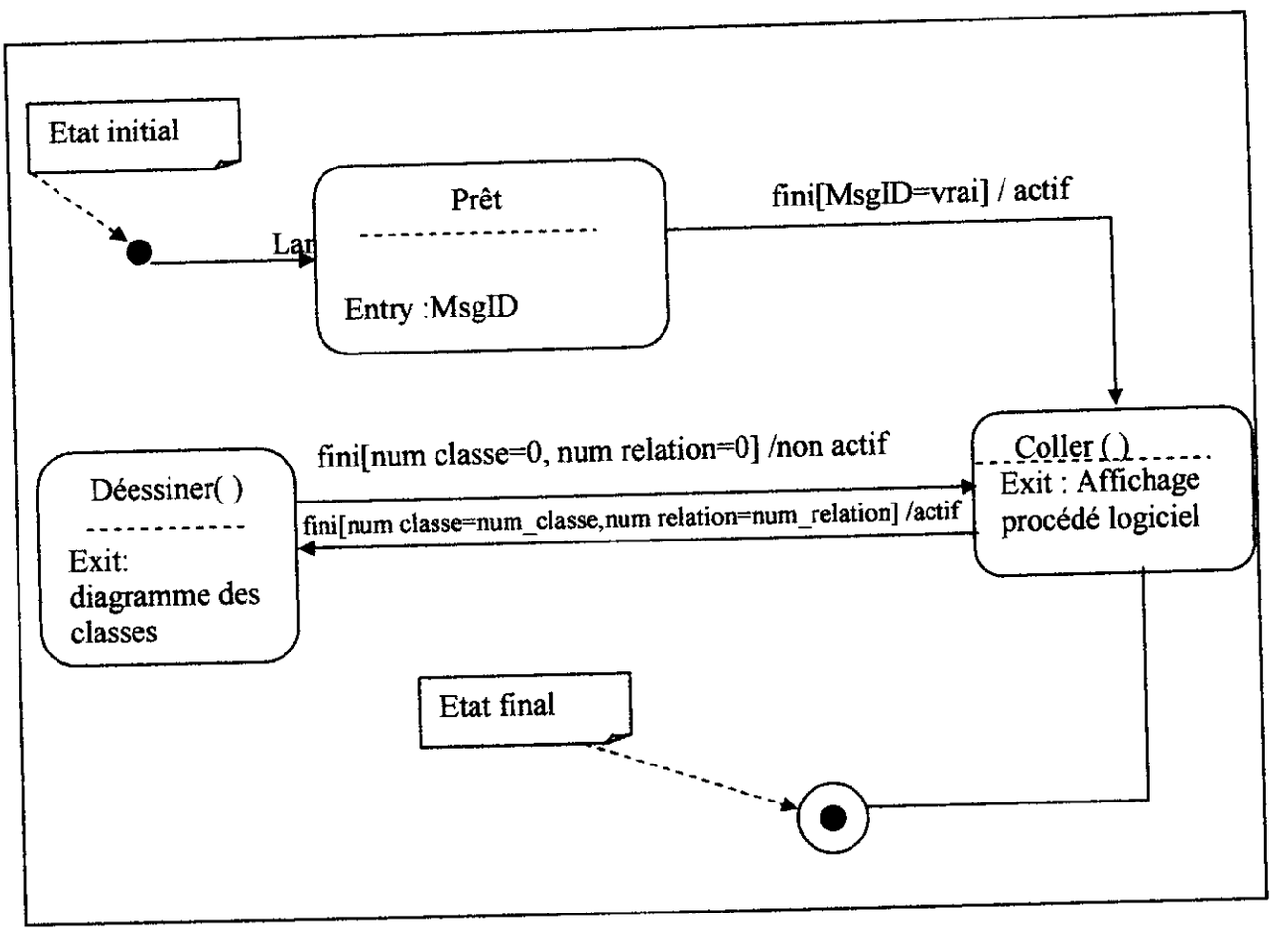
L'agent identifié à deux principaux états, soit il est prêt à exécuter les ordres de l'utilisateur, soit il est en train d'identifie les métas procédés valides a réutilisé.



FigureIII.31 : Diagramme d'état pour l'agent identifie

V.4. Diagramme d'état pour l'agent montage :

Après le lancement de l'agent collé il sera prêt à exécuter les tâches reliées à cette partie. Après la concaténation des fragments procédés logiciel l'agent affiche le nouveau procédé logiciel, puis il dessine le diagramme des classes associé à ce procédé. Comme dernière étape il affiche ce diagramme pour qu'il passe à l'état final.



FigureIII.32: Diagramme d'état pour l'agent montage

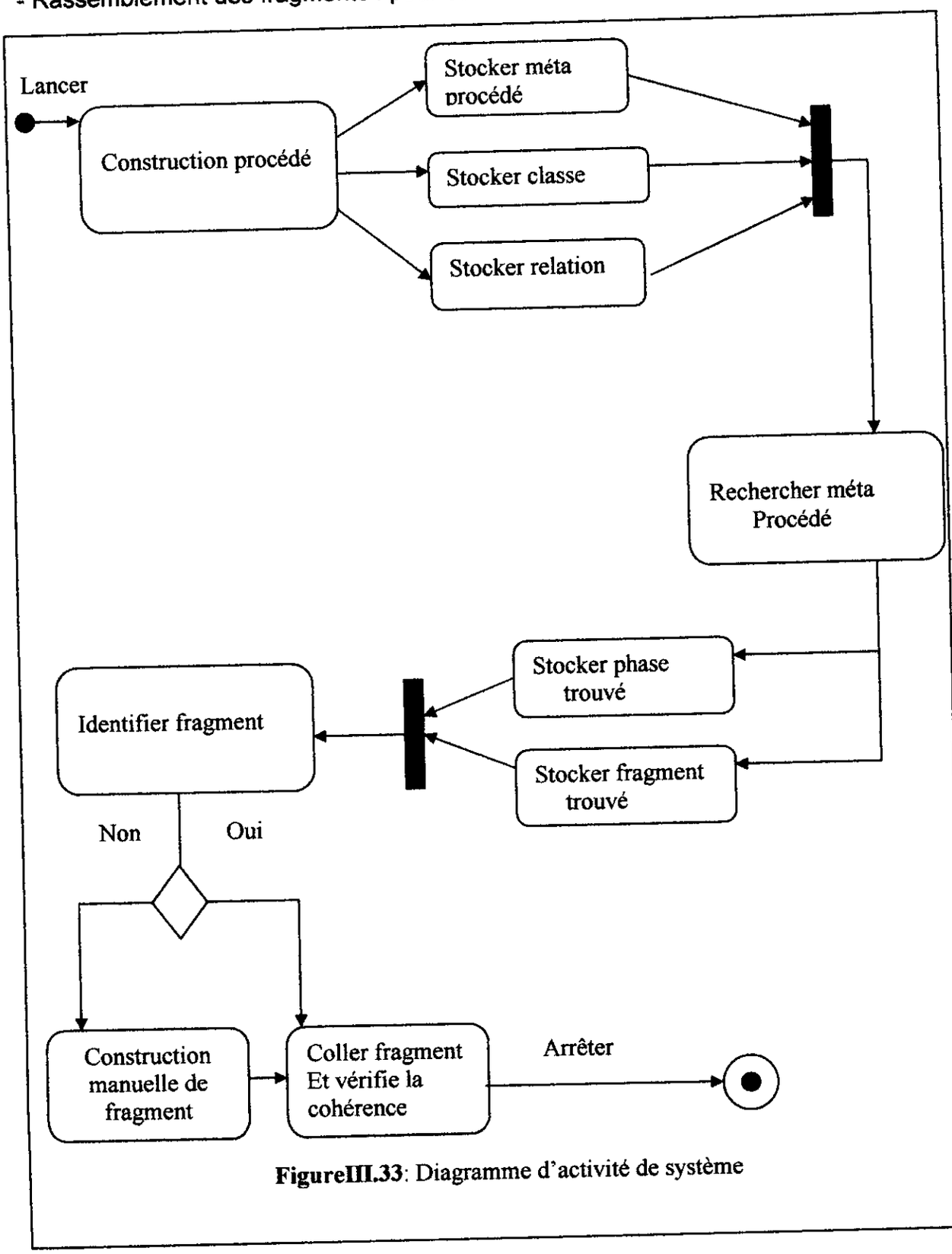
VI- Diagramme d'activité :

Ce diagramme d'activité représente les différentes activités de système. Il est composé par quatre activités principales.

- construction de procédé logiciel : pour réaliser cette tâche il faut stocker les méta procédés logiciels, les classes et les relations de chacun.

-Rechercher des méta procédés : après la réalisation de cette étape il faut stocker les différentes phases et fragments des métas procédés trouvés.

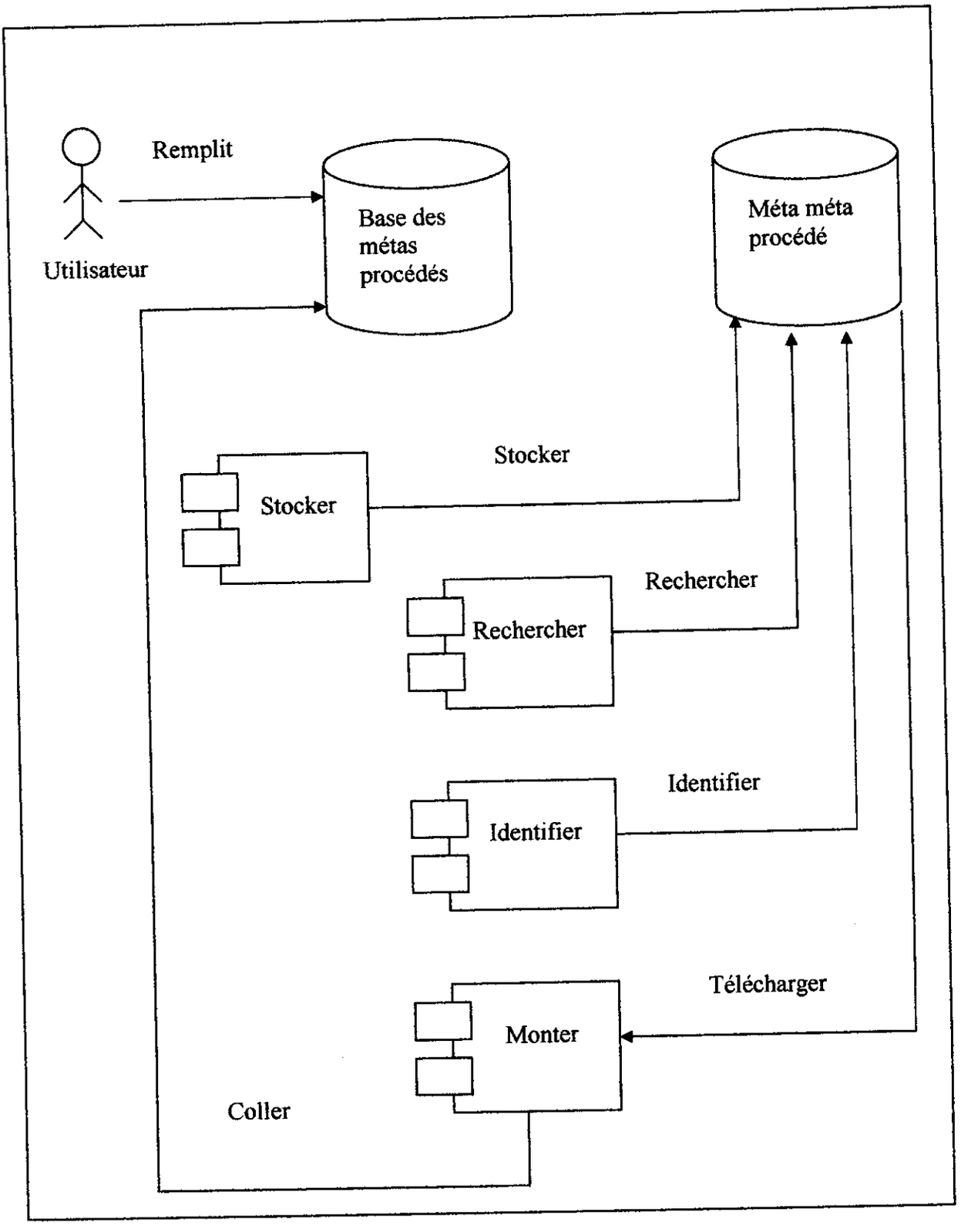
- Identification des fragments : soit le fragment existe il sera identifié, soit il n'existe pas il sera construit manuellement.
- Rassemblement des fragments : pour construire le nouveau procédé logiciel.



FigureIII.33: Diagramme d'activité de système

VII- Diagramme des composants :

Le système est composé de quatre composants. Les composants représentent les agents du système. Les données de système sont stockées dans deux bases de données. La première, est la base des méta procédés qui contient les métas procédés à réutiliser, et la base des métas méta procédés pour manipuler les données du système.



FigureIII.34 : Diagramme des composants

Conclusion :

Dans ce chapitre nous avons présenté les différent diagrammes de modélisation pour notre système suivie d'un scénario d'application.

Notre système globale est décomposé en sous système ou chaque sous système représenté par un agents pour mieux géré l'application.

Chapitre IV

Réalisation De Système

Introduction

Dans ce chapitre, nous allons mettre en ouvre le système d'agents qui réalise les défirent étapes de notre approche de réutilisation.

Ce système comporte des agents réactifs caractérisé par :

- Une architecture simple.
- Communication via l'environnement.
- Pas de mémoire de l'historique.
- Même réaction pour le même environnement.
- Décision a base de l'algorithme des 3 cycles.

LES OUTILS DE DEVLOPPEME

SQL Server :

Est un système de gestion de bases de données relationnelle (SGBDR) établit les règles de l'algèbre relationnelle. Il est caractérisé par :

-SQL Server est multithread, c'est-à-dire que chaque processus en cours sur le serveur utilise un fil d'exécution de processus SQL Server indépendant des autres threads est qui possède ses propres zones de mémoire et ses propriétés.

-SQL Server fonctionne en préallocation d'espace.

-Chaque base de données contient un journal des transactions, dont le rôle est d'assurer le stockage de toutes les mises à jour apportées à la base de données.

-Une base de données peut être définie selon plusieurs points de vue. Pour l'administrateur, c'est un ensemble de fichiers contenant des données organisées qui doivent être sauvegardées, nettoyées, réorganisées, sécurisées... .Pour l'utilisateur c'est un espace, lui permettant d'enregistrer des information et lui retrouver quant il en besoin.

-Une base de données SQL Server est constituée d'au moins deux fichier :

•Un fichier de données, avec une extension .MDF
•Un fichier de journal, avec une extension .LDF **[Mar 01]**.

C++ Builder :

Plusieurs langages destinés à la conception d'agents, mais beaucoup de ces langages sont surtout destiné à mettre en évidence certains principes et contraintes, et l'architecture de notre système nous guidons vers le langage C++ Builder.

Le C++ et un langage parmi les langages de base les plus utilisés aux environnements de développement des systèmes multi agents. Et d'après la référence **[Lem 01]** « Les langages de base les plus utilisés au environnement de développement des systèmes multi agents : LISP, C++, PROLOGE, SMALTLK, aussi que les langages d'acteurs qui supportant les mécanismes d'exécution parallèle.

Pour plusieurs raisons, les développeurs continuent à bénéficier de la puissance de C++ et des techniques de programmation orienté objet. Aujourd'hui, avec C++ Builder, on peut bénéficient a la fois de la puissance et de la facilité de développement .C++ Builder rend en effet le développement d'une application Windows incontestablement plus rapide.

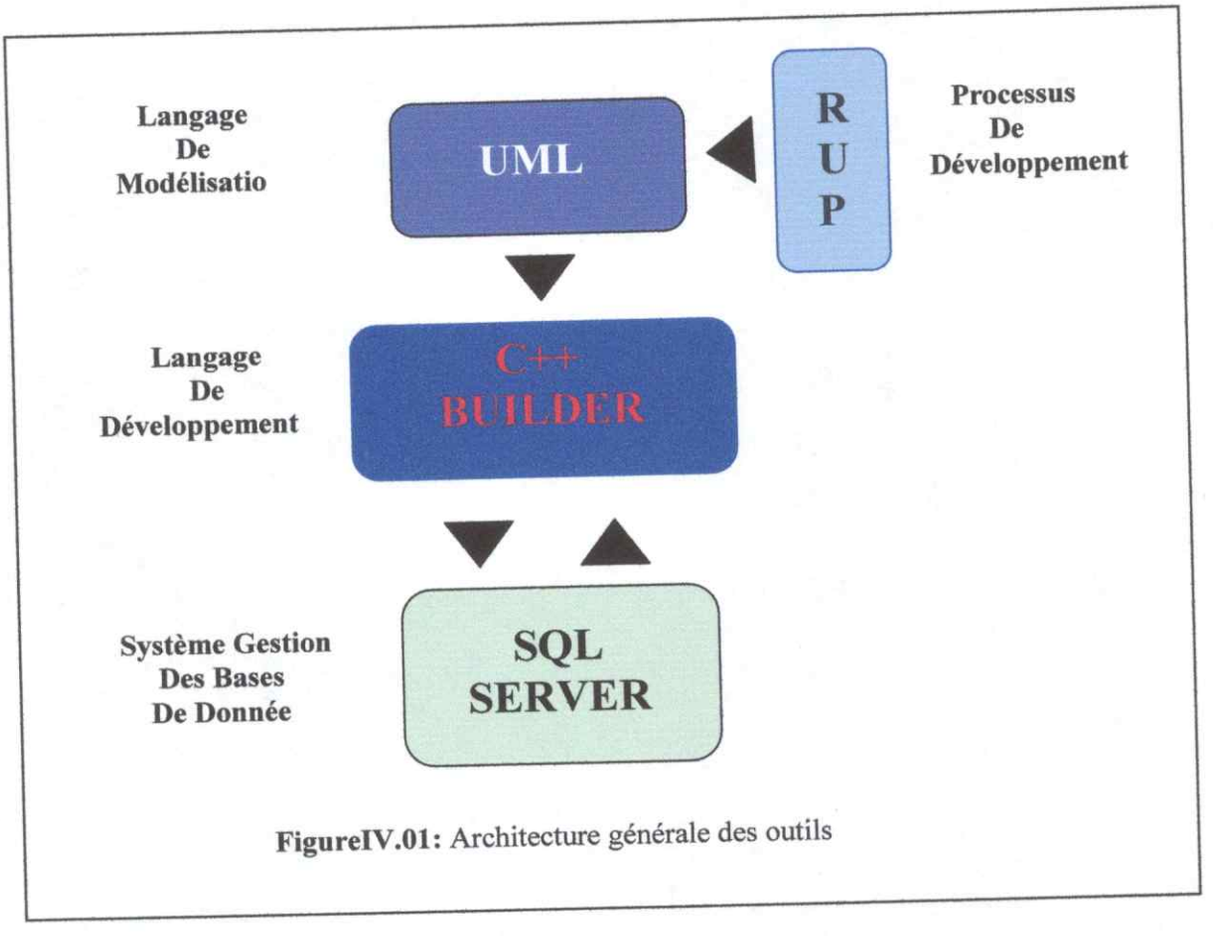
Développer une application Windows à l'aide de C++ Builder consiste en effet à :

- concevoir l'interface utilisateur de manière particulièrement visuelle et interactive.
- Spécifie de manier tout aussi interactive les caractéristiques. appelées propriétés des devers composant (fenêtre, boutons de commende, zone d'édition, etc.).
- Spécifie le code à exécuter lorsque l'utilisateur effectue telle ou telle action.

La technique s'applique également aux composant non visuels qui donnent accès aux fichiers et aux base de données. Accéder a des données y compris celles qui résident sur d'autres machines du réseau, est en effet devenu d'une simplicité inimaginable jusqu'à ce jour.

Les fenêtres jouent un rôle fondamental dans l'environnement Windows. Pour le programmeur, une application minimale consiste en un objet "Application" (qui n'a aucune matérialisation a l'écran) et, au moins, en un objet " Fenêtre". C'est dans cet objet "fenêtre" qui l'on insère toute une série de composant (boutons, boites de liste, etc.) **[Bul 01]**.

ARCHETECTURE GENERALE DES OUTILS



FigureIV.01: Architecture générale des outils

IMPLIMENTATION DU SYSTEME

Dans ce qui suit, nous allons présenté l'implémentation de l'approche de réutilisation présenté dans la partie conception.
Le système qui nous allons réalisé contient :

- Des métas modèles formels, de type centré activité qui sont déjà existe pour la réutilisation. Chaque méta modèle qu'un diagramme des classes, représenté dans le langage SQL. avec sont procédé logiciel inscrit dans ces tables.
- Un méta méta modèle pour stocker les informations concernant les métas modèles en cours d'exécution du système, ce méta méta modèle dispose

d'une base de données qui contient des tables pour stocker les informations nécessaire.
- Un environnement d'exécution des objets de système (agent de notre système) qu'est l'environnement C++.

Implémentation de l'agent stocké:

L'agent stocké est une instance de classe C++, il jeu un rôle très important dans notre système, c'est le premier agent qui sera lancé dans le système, sont rôle est compose en deux parties :

Dans la première partie il sera lancé pour stocker les métas modèles, les classes et les relations de chaque méta modèle. Et pour cela il réalise des différente connotations avec l'ensemble des bases de données existantes pour récupéré les données nécessaire et après cet étape il passe a l'état repos. Par la suit il passe la main a la l'agent recherché.

Dans la deuxième étape il sera interrogé pour stocker les phases et les fragments de chaque méta modèle retourné (retrouvé) par l'agent recherché.

La figure ci-dessous illustre la forme de la classe de l'agent stocké.

```
Class agent_stocke{

Public :
Void Initialise();
Void stop() {……}
Void run(){……}

/////////////// Methods proper a l'agent stocké.//////////////////
        ///// première étape de l'agent stocké/////
……
Void Stocke_meta_modele( ) {……}
Void Stocke_classes( ) {……}
Void Stocke_relations( ) {……}
……
        ///// Deuxième étape de l'agent stocké/////
……
Void Stocke_phases ( ) {……}
Void Stocke_fragments( ) {……}
……
};
```

FigureIV.02: La forme de la classe agent stocké é (Montage)

Implémentation de l'agent Recherché :

Comme tout les éléments de système l'agent recherché est une instance de classe C++, le rôle réalisé par cet agent est de rechercher les méta modèle nécessaires à la réutilisation à base des classes et relations spécifier par l'utilisateur. Le résultat de cet agent sera un catalogue qui contient toutes les informations sur les paramètres saisis (classes, relations).

La figure IV.03 illustre la forme de la classe recherche.

```
Class agent_recherche{

Public :
Void Initialise();
Void stop() {.....}
Void run(){.....}

//////////////// Methode proper a l'agent recherché/////////////
.....
Void recherche_classe( ) {.....}
Void recherche_relation( ) {.....}
.....
};
```

FigureIV.03: La forme de la classe agent recherché

Implémentation de l'agent identifié

L'agent identifié c'est le troisième objet qui sera interrogé dans notre système, le rôle réalisé par cet agent et d'identifié les fragments qui serrant utilisé à la construction de notre nouveau procédé logiciel. Le choix des fragments et basé sur l'algorithme des trois cycles. Nous pouvons bénéficier de cet algorithme a la réalisation de l'agent identifié, par superposition il suffit de remplacer le moteur d'inférence par l'agent identifie, la base des connaissances par le méta méta procédé et les règles par les fragments dans ce cas on peut utiliser cet algorithme a l'identification des fragments. (Voir annexe).

Comme il exécute des exceptions en cas au les fragments recherche n'existe pas.

La figure IV.04 illustre la forme de la classe identifie.

```
Class agent_identifie{

Public :
Void Initialise();
Void stop() {……}
Void run(){……}

////////////// Méthodes proper a l'agent idenfihié/////////////
……
Void identifie_classe( ) {……}
Void identifie_relation( ) {……}
Void identifie_fragment ( ) {……}
……

};
```

FigureIV.04: La forme de la classe agent identifie.

Implémentation de l'agent montage

L'agent collé c'est le dernier objet qui participe au système,

- il va réaliser l'assemblage des fragments procédés identifiés dans l'étape précédente pour construire le nouveau procédé logiciel
- puis il crée une nouvelle base de donnée avec ces nouvelles classes et relation.
- Il remplie les champs de chaque table on utilisant les informations existantes dans le procédé logiciel
- et à la fin, il réalise une vérification de toutes les informations et il met a jours la nouvelle base de donnée pour qu'ont puisse l'utilisé dans une autre réutilisation d'un autre procédé logiciel.

La figure IV.05 illustre la forme de la classe collée.

```
Class agent_colle{

Public :
Void Initialise();
Void stop() {……}
Void run(){……}

////////// Méthodes proper a l'agent collé//////////////
……
Void monter_fragment( ) {……}
Void construir_procédé_ligiciel( ) {……}
Void CreatDataBase( ) {……}
Void CreateTable( ) {……}
Void remplir_table( ) {……}
Void RestaurerBaseDonnee( ) {……}
……

} ;
```

FigureIV.05: La forme de la classe agent collé.

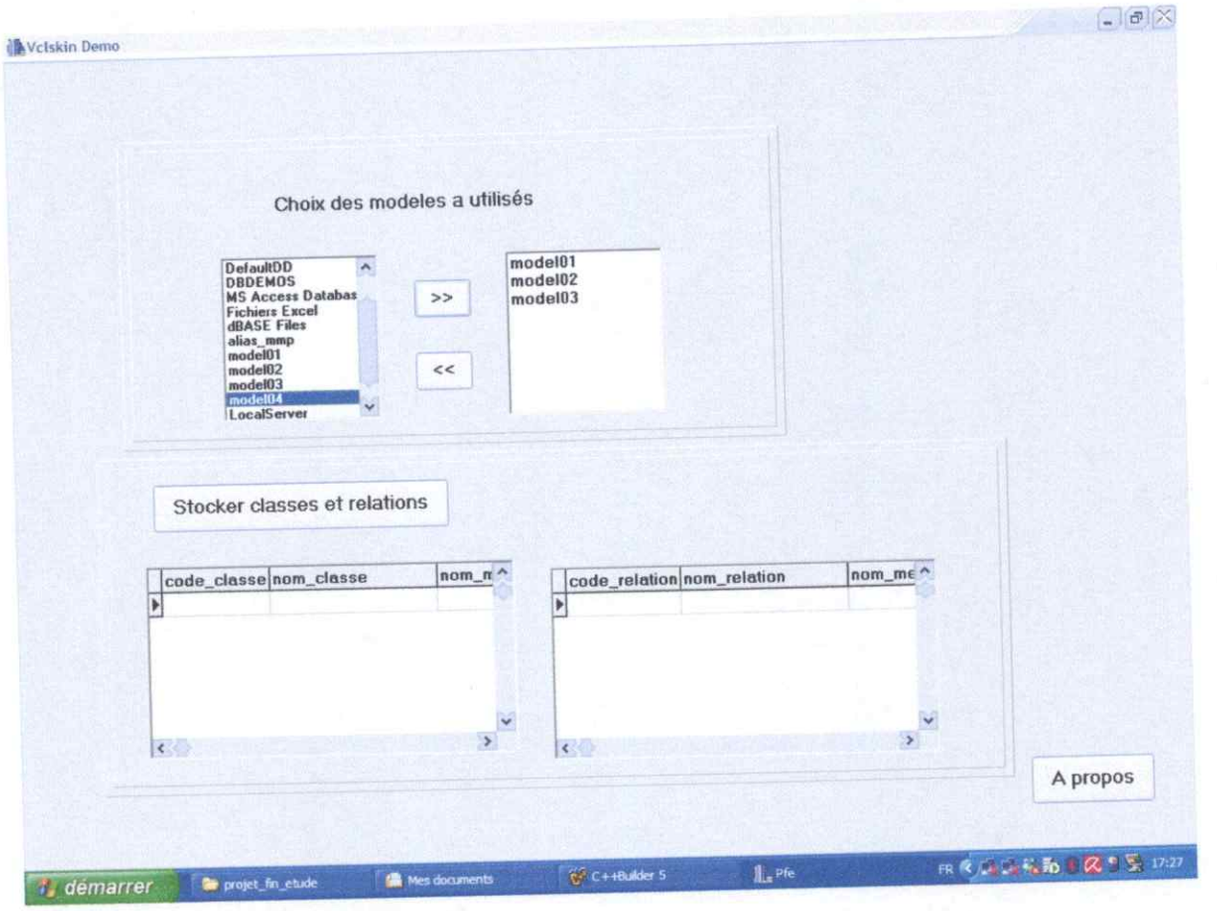
TEST ET RESULTAT

Cette figure représente l'interface initiale de l'agent stocker, la premier étape réalise par cette agent est le stockage des métas modèles, les classes et les relations. Pour parcourir cette interface vous suivez les étapes suivantes :

-stocker les modèles à utiliser.

L'agent stocke affiche tout les connotation des bases de donnée existe, il reste à l'utilisateur de sélectionné les noms des métas modèles qu'il vue les utilisés (toute dé pond de problème posé).

Après la sélection des modèles à réutiliser, le résultat sera affiche pour l'utilisateur dans le champ juste adroit, et dans le méta méta modèle la table méta procédé sera remplie par ce résultat.



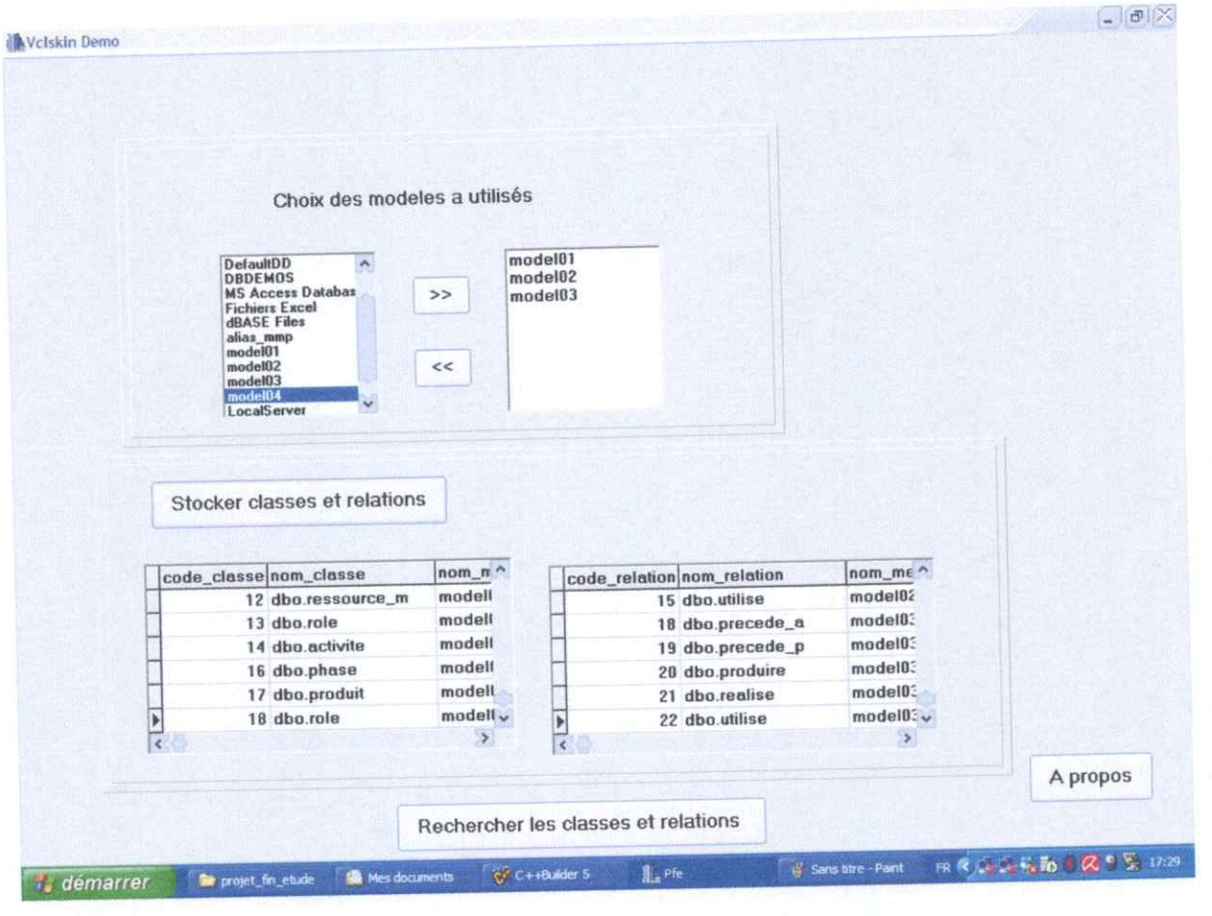
FigureIV.06: Interface résultat de stockage des métas modèles

Apres cette étape il faut stocker les classes et les relation on cliquant sur le Botton « stocker classes et relations » le résultat sers le suivant :

L'agent dans cette étape il ce base sur les noms des méta modèle stocké dans l'étape précédente pour stocké les noms des classes et relations.

Le résultat sera affiché a l'écran pour l'utilisateur dans deux tables la première c'est la table des classes stocké et la deuxième c'est la table des relations associé a ces classes.

En parallèle les deux tables dans le méta méta modèle seront rempli. L'affichage sur l'écran est le suivant :



FigureIV.07 : interface résultat de stockage des classes et relations.

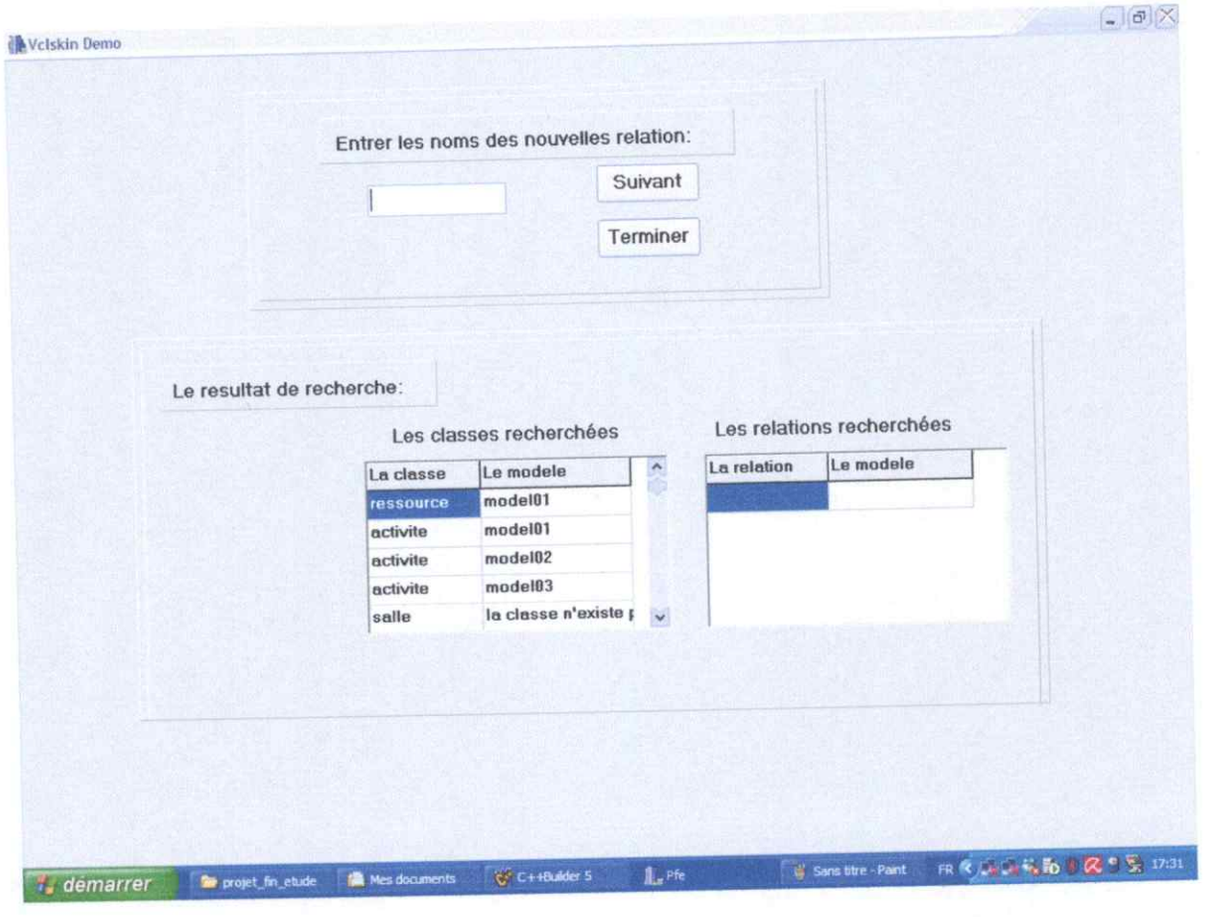
Après cette étape la première partie de l'agent stocké est terminé , l'agent stocker exécute une attente.

L'étape suivant sera réalise par l'agent recherché ou ce dernier va nous recherché les noms des métas modèles pour les quelles l'agent stocké va nous stocké leurs phases et leurs fragments associé a chaque phase. En cliquant sur le bottons « Rechercher les classes et relations ».

Dans cette interface, l'agent recherché va affiché un champ d'édition pour que l'utilisateur édité les noms des classes et relations nécessaire à la construction de notre nouveau procédé logiciel.

Pour lancer cet agent, il suffie de saisis dans le champ d'édition les noms des classes a recherché, après la saisis de chaque nom en cliquant sur le bouton « Suivant », pour terminer en cliquant sur le bouton « Terminer »

L'interface de cet agent pour la recherche des classes sera la suivante.

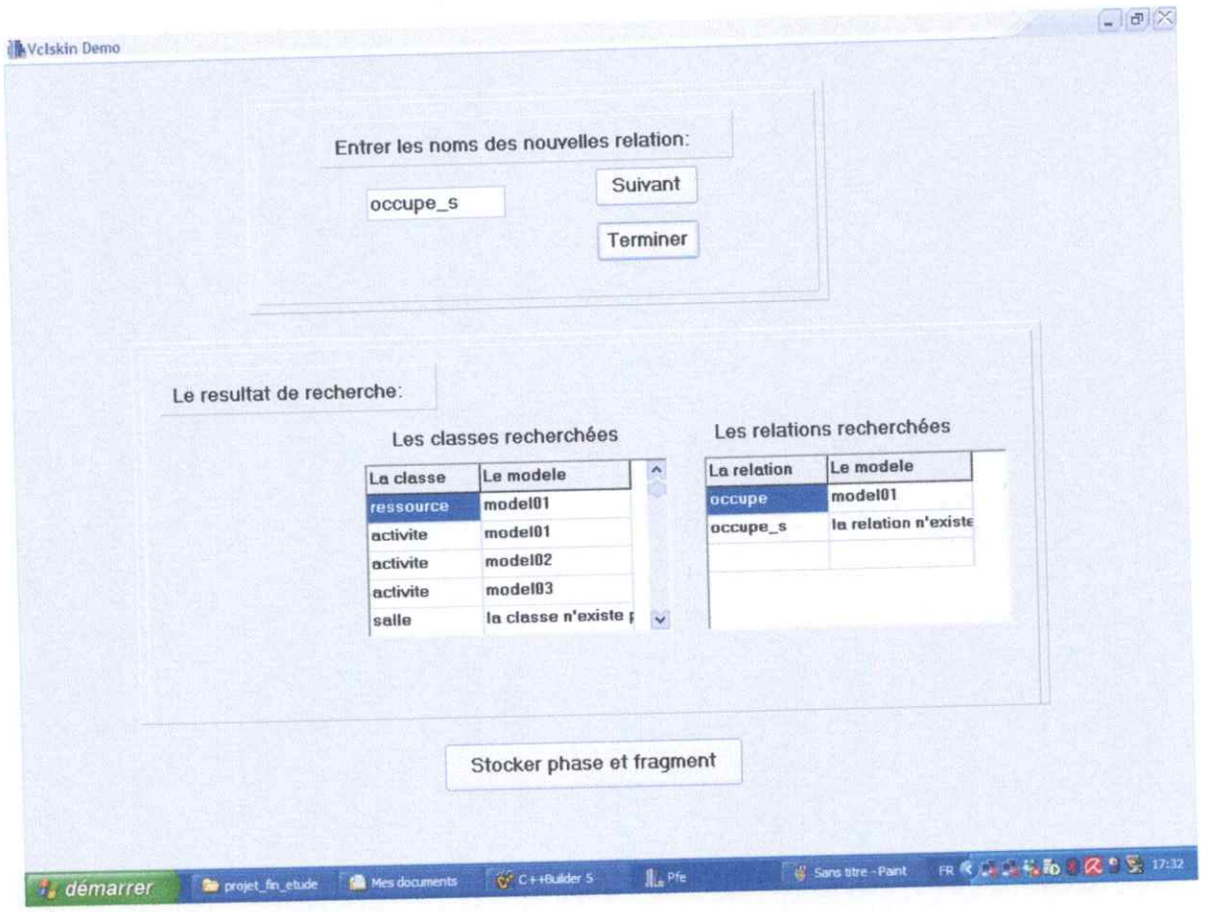


FigureIV.08 : Interface résultat de l'agent recherché pour la recherche des classes.

Le résultat de l'étape suivante sera les noms des relations recherché par l'agent recherche, les mêmes étapes de la tache précédente.

La figure ci dessous montre l'interface contenant le résultat de l'agent recherché.

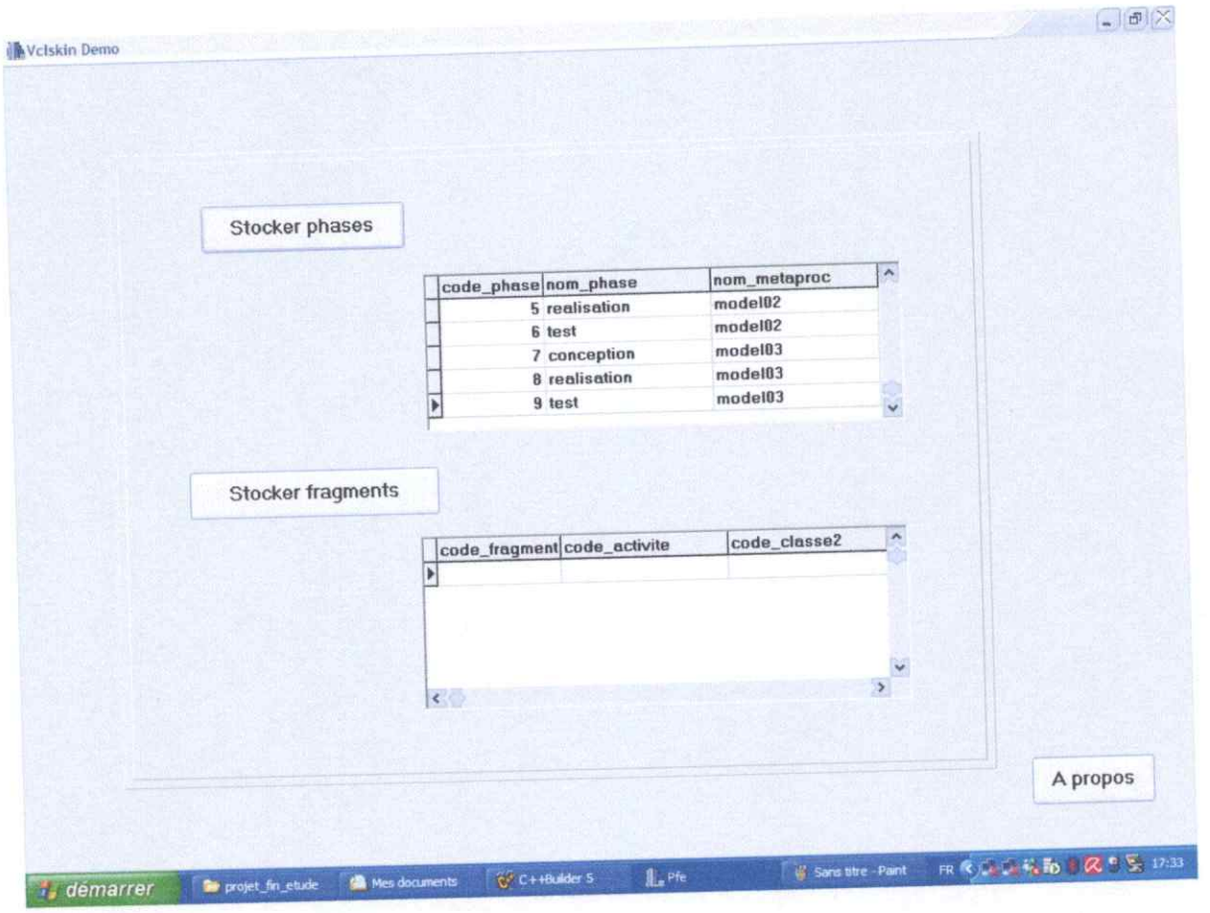
.



FigureIV.09 : Interface résultat de la recherche des relations.

Après cette étape, le rôle d'exécution revient a l'agent Stocké pour le stockage des phases et des fragments, mais le stockage sera juste pour les méta modèle résultats de l'étape de la recherche, l'interface qui illustre cette étape est la suivant :

Cette figure représente le stockage des phases des métas modèles retournées par l'agent recherché.

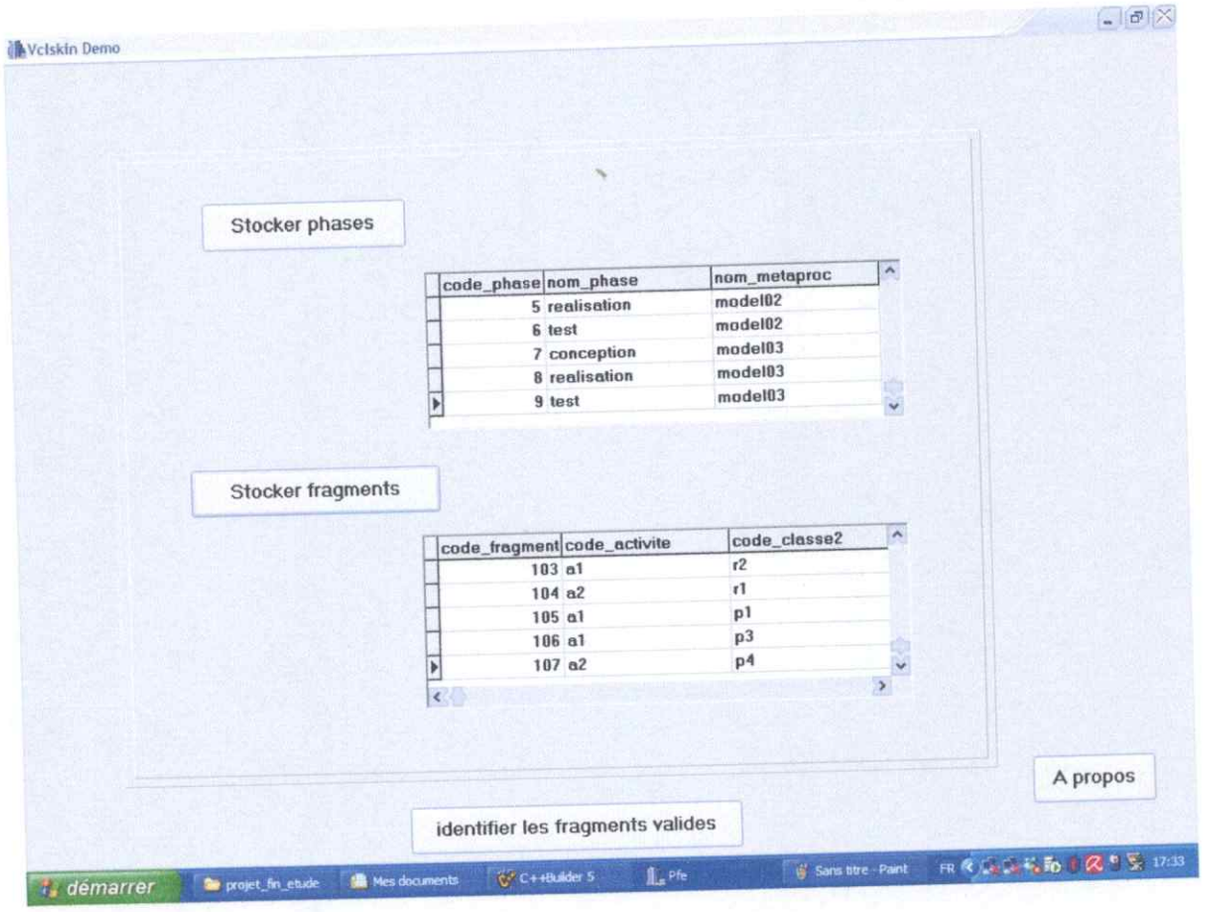


FigureIV.10: La forme de la classe agent stocké pour le stockage des phases.

Apres le stockage des phases dans le méta modèle. Il reste une dernier étape de stockage, c'est le stockage des fragments, pour cela il soufie de cliqué sur le bouton « Stocké fragments ».

Le résultat sera affiché sur l'écran pour l'utilisateur et dans le méta méta modèle la table fragment sera remplie en parallèle.

Le résultat de stockage des fragments sera représenté par la figure ci- dessous:



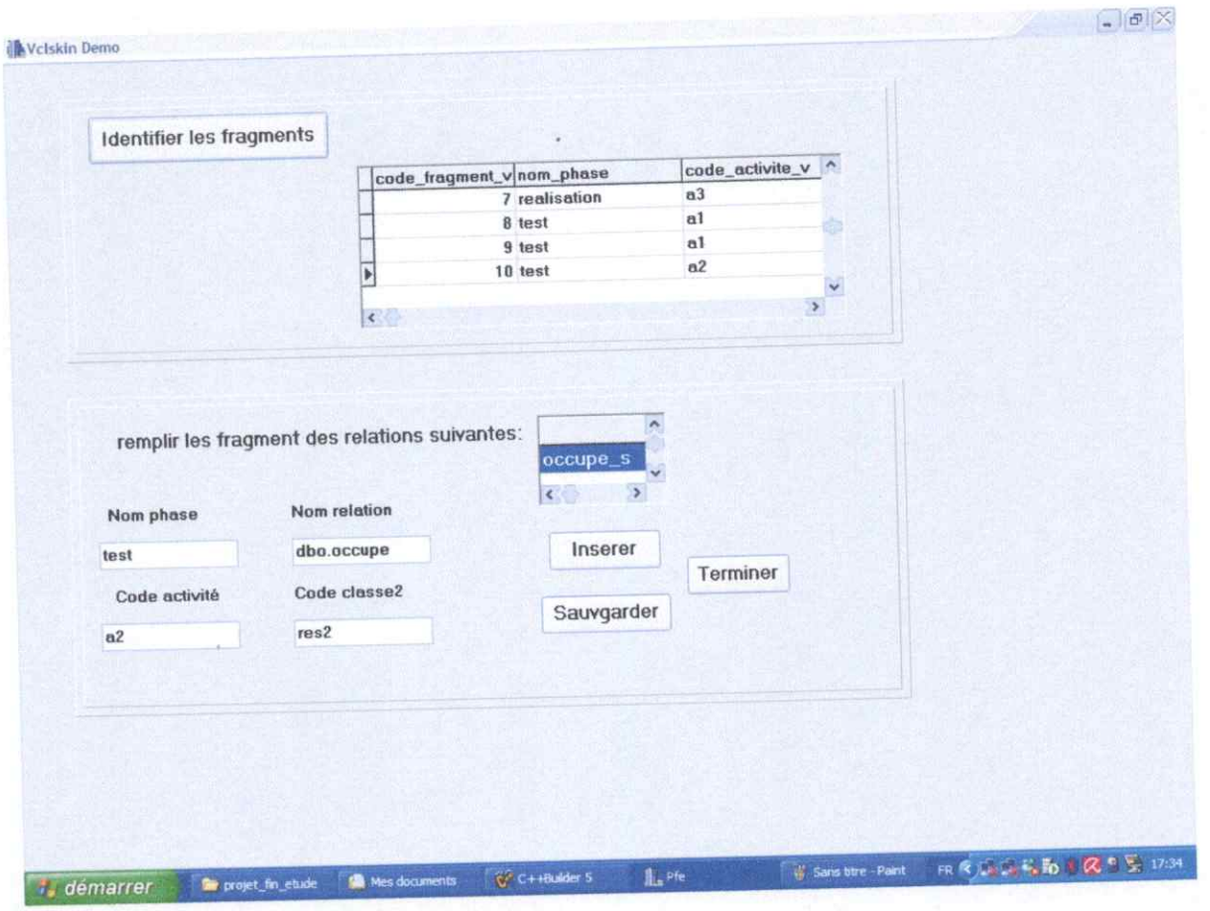
FigureIV.11: L'interface résultat de stockage des fragments procédés logiciels.

Apres cette étape il ne reste qui d'identifies les fragments de chaque phase sélectionné, pour passer la main a l'agent collé (montage) pour la réalisation de la dernier tâche.

L'interface de cet agent elle contient le bouton « identifier les fragment ». Il suffie de lancé cet agent pour qu'il retourne l'ensemble des fragments qui construira notre procédé logiciel, l'identification des fragments et basé sur les phases de procédé à construire.

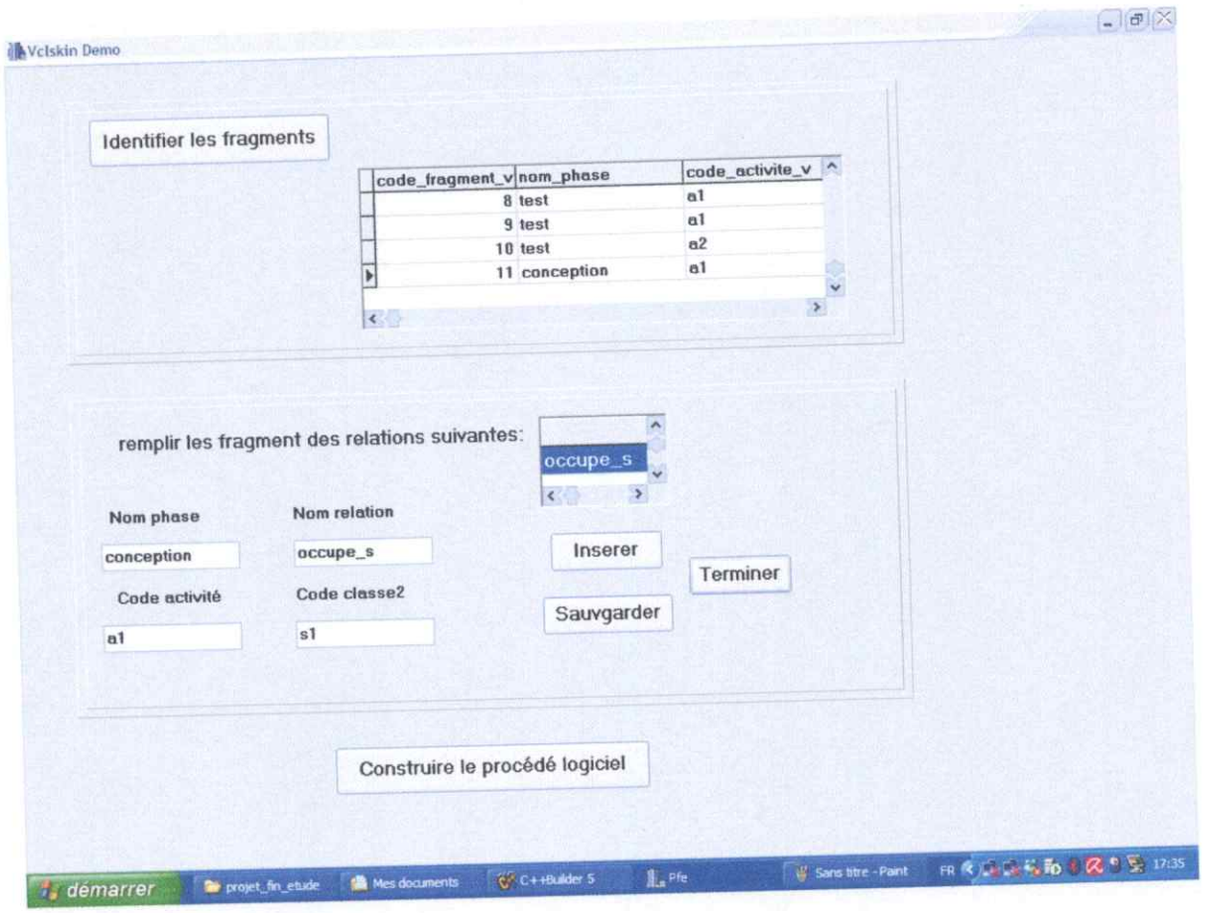
Pour cette étape l'agent identifié affichera l'ensemble des fragments existants.

L'interface de cet agent est la suivante :



FigureIV.12: interface résultat de l'identification des fragment existant.

Pour les fragments non existants, l'agent identifie permet a l'utilisateur de saisis les fragments associe aux relations non existante. L'interface ci- dessous illustre cette opération :



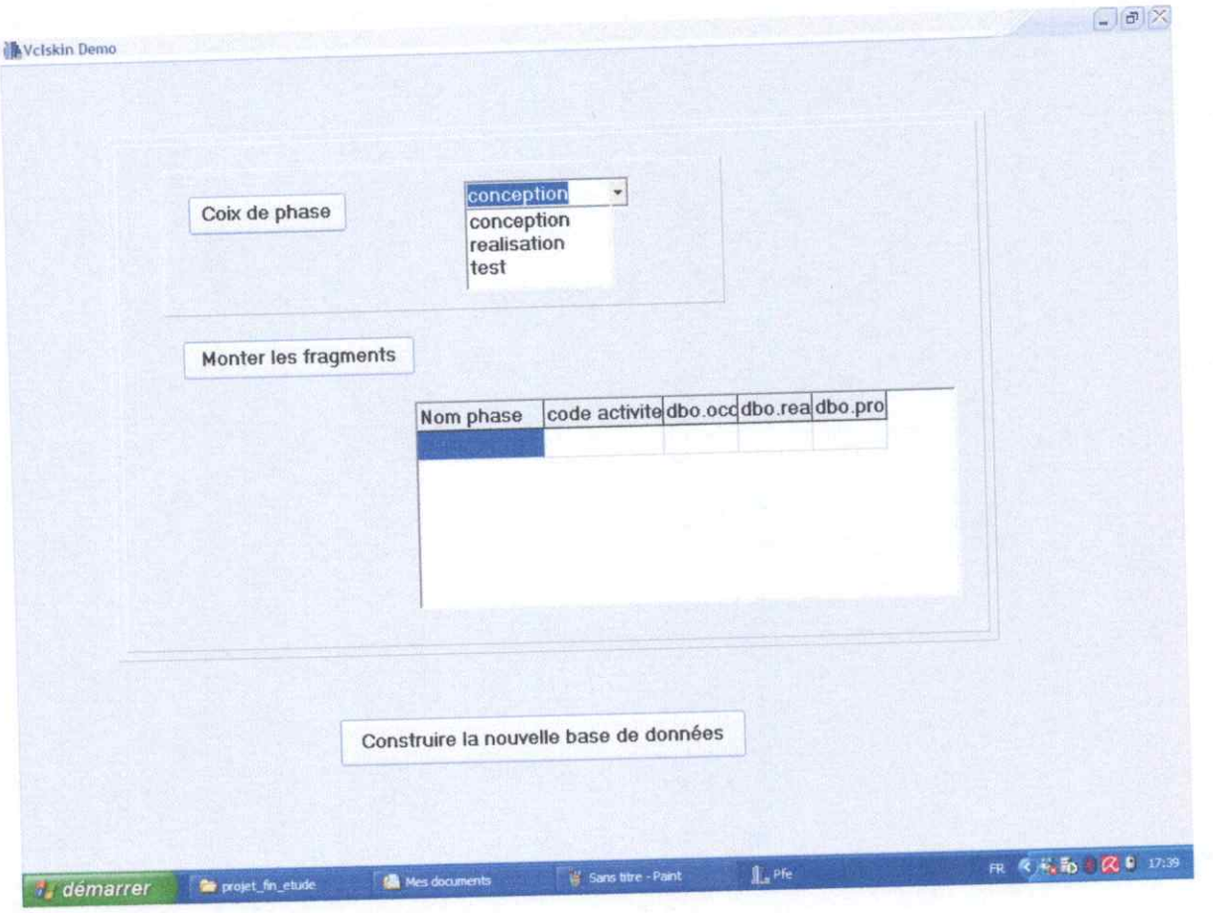
FigureIV.13: interface résultat de l'agent identifie.

Puisque les fragments sont identifié, il ne reste que les collés pour construire un nouveau procédé logiciel, ce rôle est réalisé par l'agent collé (montage), le résultat de cette étape est représenté par la figure ci-dessous.

La structure de notre nouveau procédé logiciel est la suivante :

- Un ensemble des fragments un suit l'autre dans l'ordre des phases qui construit le nouveau procédé logiciel.
- Chaque fragment sera représenté par un vecteur qui contient toute les relations qui relie les activités de chaque phase.

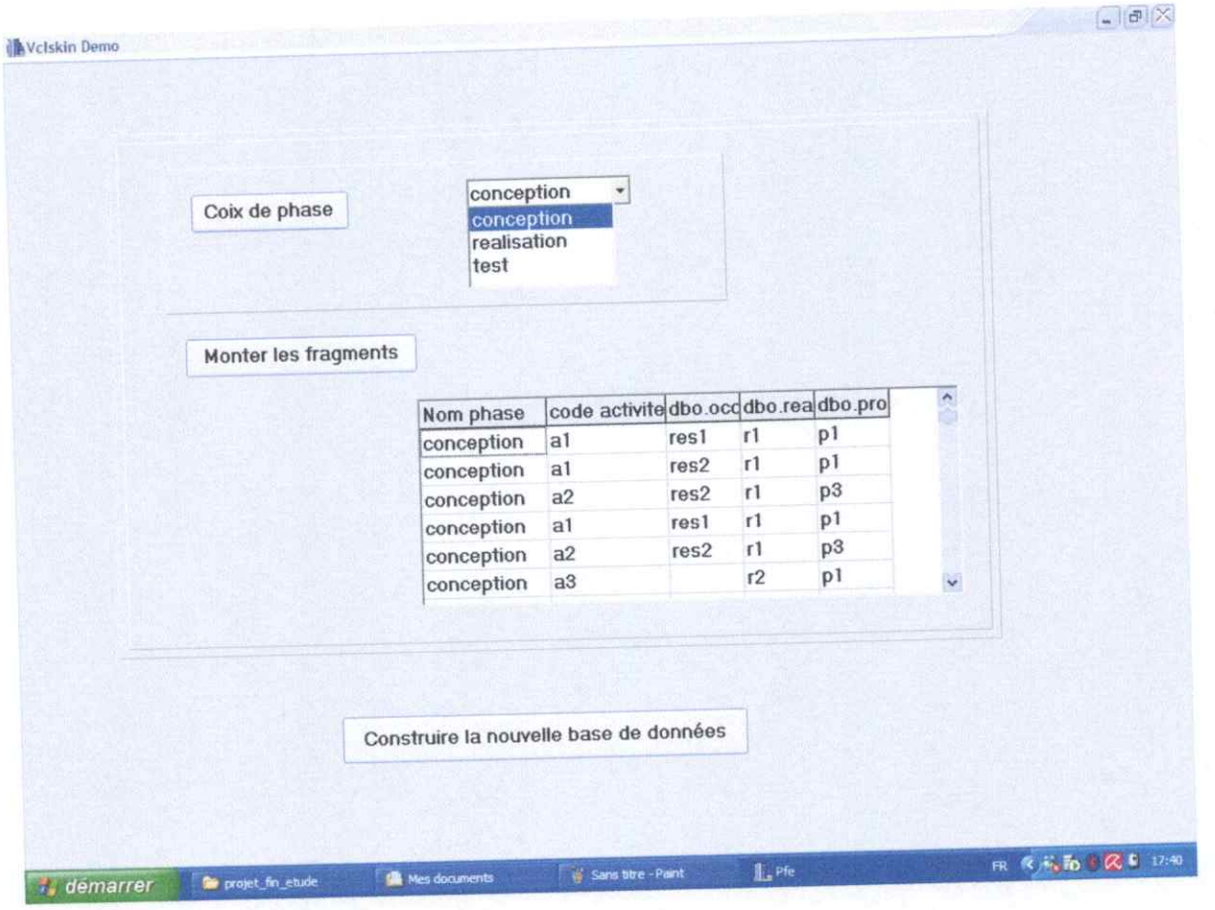
Pour lance cet agent il faut sélectionné la phase pour la quelle l'agent collé réalise le montage des fragment, en cliquant sur le bouton « choix de phase ». L'interface de cet agent sera la suivante :



FigureIV.14: interface initiale de l'agent collé

Apres la sélection de la phase, l'agent montage construira le nouveau sous procédé logiciel associé a la phase sélectionné et après que l'utilisateur sélectionnera toutes les phases nécessaires, l'agent montage affichera le nouveau procédé logiciel complet.

La figure ci- dessous illustre le résultat de cet agent.



FigureIV.15: interface résultat de l'agent collé

Ce nouveau procédé logiciel construit sera associe à une base de donnée qui contient tout les informations sur ce procédé, cette nouvelle base de donnée sera construite par le même agent collé (montage). Comme en peut utilisé ce résultat dans les prochaines réutilisations.

Chapitre V

Conclusion

Générale

CONCLUSION GENERALE

Le développement de logiciel de ces dernières années à montrer un mouvement vers des méthodes et approches qui utilise des concepts provenant du domaine du problème lui-même, la technique de la réutilisation est l'une de ces méthodes.

Aujourd'hui, la réutilisation est de plus en plus attractive vu les avantages qu'elle offre pour la modélisation des nouveaux procédés logiciel qu'est une tâche en générale complexe, néanmoins, l'utilisation de cette méthode pour la production de logiciel est encore loin des attentes des développeur, et reste limité, a cause des problèmes de l'hétérogénéité et l'interopérabilité.

Dans ce travail, nous avons d'abord présenté le domaine des procédés, plus particuliers les procédés logiciels, en suite nous avons présente quelque approche de réutilisation et nous avons présenté aussi le domaine des agents. A la fin nous avons proposé une approche de réutilisation qui répondra aux préoccupations déjà citée.

Les objectifs

Nous allons maintenant revoir l'ensemble de ce que nous avons accompli comme travail à la lumière des objectifs qui nous avons fixés.

Etudier le concept des procédés et procédé logiciel :
Nous avons présenté les différentes définitions, les approches et les environnements.

Etudier le concept d'agents :
Nous avons présenté les définitions et les caractéristiques des agents et des systèmes multi agents.

Stocker les procédés logiciels :
Nous avons présenté une technique de formalisation des métas procédés pour pouvoir les stocker dans une base de données.

Proposer une approche de réutilisation pour les métas modèles centrés activité :
Nous avons présenté l'algorithme de stockage, de la recherche, d'identification et de réassemblage des fragments procédés logiciel.

Décrire la solution proposée :
Nous avons décrit la solution en détail on utilisant le langage de modélisation UML basé sur le processus de développement RUP.

Pour illustrer l'approche de réutilisation proposé, nous avons implémenté une application basé sur les agents réactifs (agent stocké, agent recherché, agent

identifie, et agent monté). Cette application est réalisée par le langage de programmation C++ Builder.

Perspectives

Inévitablement notre application n'est pas parfaite, il reste encore des choses à ajouter.

- La réutilisation est un domaine très vaste, car même si on fait des efforts, on reste loin de pouvoir traiter tous les problèmes. Pour l'instant le système fonctionne d'une manière acceptable sur les métas modèles centrés activité mais pour d'autre type de modèle est un autre problème.

- Réalisation complète de l'approche de réutilisation car nous allons considérer dans notre application un seul type de méta modèle (centré activité) donc la réutilisation par des méta modèles centré produit et centré rôle est absente.

- Une étude ergonomique plus avancée pour l'interface utilisateur.

Enfin, ce modeste travail nous a permis d'acquérir une vraie expérience sur le terrain et de voir les difficultés qui peuvent être rencontrées aussi que les solutions qui peuvent être apportées.

Références Bibliographiques

[All 01] : Kettani N, Mignit D, Paré P., Rosenthal-Sabroux C, " D Merise à UML ", Eyrolles 2001.

[Ami 99] : Mahfoud Amiour, Thèse, LSR, université de Grenoble I, 17 Juin 1999.CR :D.2.9 « Vers une fédération de composants interopérables pour les environnement centré procédé logiciel.»

[Bel 96]: Avrilionis D, Belkhatir N, Cunin P "A Unified Framework for software process . Enactment and Improvement"4th Internationnal conference on software process, UK 1996.

[Ber 02] : F. Bernardi, « Méthode d'analyse orienté objet UML » , Dondo , 2002.

[Ber 03] : Modélisation du Méta-Procédé RHODES avec SPEM, Tran Hanh Nhi*, Bernard Coulette**, Xavier Crégut***,Dong Thi Bich Thuy*, Tran Dan Thu*,2003.

[Bre 02] : Breton E, Contribution à la représentation de processus par des technique de méta modélisation, thèse doctorale, N'ED 0366-066, Université de Nantes,2002.

[Bul 01]: livre C++ Builder, auteur Gérard Lebanc, Edition 2001

[Ccd 02]: Coulette B, Crégut X, Dong T et Tram D.T. «A Meta process to define and reuse process components » 6th world conference on integrated.Design and process technology. June 2002

[Cho 00]: Chou S, and J.J. Chen "Process program Development Based on UML and Action Case – part 2: the Method" Journal of object oriented programming (JOOP), July 2000.

[Cou 01]: Coulette B, Crégut X, Dong T. B. T. and Tran D. T. "Managing process through a base of reusable component" 3rd International Conference on Entreprise Information Systems, Setubal, July 2001.

[Cou 02]: Coulette B, Crégut X, Dong T. B. T. and Tran D. T. "RHODES a process component Centered Software Engineering Environment" 2nd International Conference on Entreprise Information Systems, Stafford, July 2002.

[Cpj 05]: Livre comment programme JAVA, ,Edition 2005

[Cre 97]: Crégut Xavier, "Un environnement d'assistance rigoureuse pour la description et l'exécution de processus de conception – Application a Approche objet" thèse doctorale, institut National Polytechnique de Toulouse, France, 1997.

[Ctr02] : « composant dans l'Ingénierie des SI, Conception et Technique de réutilisation », Franck Barbier, Corien Couvert, Mourad oussalah, Dominique rien ; décembre 2002

[Der 99] : Derniame J-C, Caba. B, « Softeware process: principles Methodology and Technology » Springer-Verlag, Lecture Notes in computer Science 1500; 1999 .

[Gab 98] : Joseph Gabay, " Merise vers OMT et UML", Masson, 1998.

[Gal 00] : Ma. Lizbeth GALLARDO DEA préparé dans l'équipe ADELE, au laboratoire LSR Soutenance à Grenoble, Le 20 Septembre 2000.

[Gul 00]: L.Gulyas; L.Kovacs; B. Pataki « An Overview of mobile Software Systems »; 2000

[Her 00]: Rémy Fannader, Hervé Lerroux, « UML principes de modélisation », Dunod ,
2000

[Iee 90] : IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology" IEEE Computer Society, September 1990.

[Jam 05] : Thèse docteur de université Joseph Fourier de Grenoble « Environnement de procédé extensible pour l'orchestration ,Application aux service web » le 13-decembre-2005.

[Jcj 93] : Jacobson, I., Christerson, M., Jonsson, P. Övergaard, Gunnar, Le génie logiciel Orienté Objet, Ed. Addison-Wesley, 2ème Edition, 1993, 536 pages

[LD 97] : Lindquist, T., Derniame J.C.Towards Distributed and Composable Process
Components. May 1997 *Available at*
http://pooh.eas.asu.edu/yfppg/pubs/abstracts.5.html#pgfId=282178

[Lem 01]: « L'intelligence Artificielle Distribuée et les Systèmes Multi Agents »
Exposé par : LEMLOUMA Tayeb ; BOUDINA Abdelmadjid ; 2001

[Mer 05] : « Un Meta Modèle Pour L'automatisation Du Déploiement D'Applications Logiciels », NOELLE Merle ,Equipe Adèle, laboratoire LSR-IMAG, université Joseph fourier 220,Rue de la chimie France,2005

[Mul 01] : Pierre-Alain Muller, " Modélisation Objet avec UML", Eyrolles 2001.

[Ohe 99] : Orfali R., Harkey D., Edwards J., Client/Serveur : Guide de survie. Troisième édition.Traduction de françois Leroy et Jean-Pierre Gout. Edit. Vuibert, Paris 1999, 782 pages

[Omg 02]: OMG, OMG UML Profile of Entreprise Distributed Object Computing , OMG Document ad/02-02-05, Fev 2002.

[Ost 97]: Osterweil Leon J. Software Process are software too, Revisited: An Invited Talk on the Most Influential Papers of ICSE9. In Procceding of the 19th international conference on software engineering 1997.

[Pau 99]: Paul Jean, Jean Bezivin, «Correspondance sttructurelle entre produit et procédé »,1999.

[Phi 00] : Philippe Sellem « Accès a des base des données repartie à travers le web ; Méthode de recherche " agent intelligent " »2000

[Rem 99] : Livre UML Principes de Modélisation, Rémy Fannader, Hervé Leroux, 1999.

[Sha 90]: Shaw M., "Prospects for an Engineering Discipline of Software", IEEE Software, November 1990, pp. 15-24

[Spm 05]: « Software Process Engineering Meta model Specification - Version 1.1, OMG, formal/05-01-06», January 2005

[Sql 01]: Livre SQL Server 2000, auteur MARC Israël, Edition octobre 2001.

Références Webographies

[Ref 01]: http://www.swe.uni-linz.ac.at/publications/abstract/TR-SE-97.04.html

[Ref 02]: http://search2.computer.org/advanced/Advanced_Search_logged.jsp

[Ref 03]: OMG, Model Driven Architecture http://www.omg.org/mda.

[Ref 04]: Objecteering documentation set http://www.softeam.org/produits.htm.

[Ref.05] :http://www.agentintelligent.com/agent_intelligent/agents_intelligents.html#Caractéristiques des agents

[Ref 06] :http://www.limsi.fr/Individu/jps/enseignement/examsma/2004/LEGUERN/#t0

[Ref 07] : http://fr.wikipedia.org/wiki/Syst%C3%83%C2%A8mes_multi-agents#Origine_et_aspects_techniques

[Ref 08]:
http://cours2.fsa.ulaval.ca/cours/sio65293/agents/introduction/KMCenter_agent.pdf

[Ref 09]: http://www.decisionnel.net/agentintelligents/index.htm

[Ref 10] : http://www.memoireonline.com/12/05/39/m_agents-intelligents.html

[Ref.11]:IBMRationalUnifiedprocessRUP:http://www.ibdconsulting.com/francais/inclBMRatinalRUP.cfm#.

[Ref.12]:IBMRationalUnifiedProcessConduitdeProjet:http://www.ibdconsulting.com/francais/incrup.cfm?menu=1#.

Annexe

Algorithme des trois cycles

Introduction

Dans le domaine d'intelligence artificielle, un système expert peut être décomposé en deux grande patries, base de connaissance et moteur d'inférence. Pour que nous puissions définie l'algorithme de trois cycles nous définissons bravement un système expert.

Définition de système expert :

Un système expert est composé d'une base de connaissance et un moteur d'inférence en minimum.

La base des connaissances :

Elle contient l'ensemble des faits relatifs au problème concéderai et l'ensemble des règles de production et des connaissances heuristique qui permet la résolution de problème.

Le moteur d'inférence :

Il réalise le fonctionnement créatif de système.
-à partir des règles et des faits le moteur d'inférence génère le nouveau fait
Afin de réaliser la résolution effective de problème .il utilise un algorithme des trois cycles. (MATCH, CONFLIC RESOLUTION, ACT)

Définition de l'algorithme :

MATCH: il consiste à rechercher l'ensemble des règles d'inférence qu'ils peuvent appliqués a un instant donné a la base des faits et a noté les faits a utilisés (choisir toutes les règles applicables)

CONFLIC RESOLUTION: Le moteur va choisir parmi toutes les règles qu'il peut utiliser s'elle qu'il va exécuter (exemple:choisir la première règle disponible)

ACT: Après avoir sélectionner la règle, le moteur l'exécute, est exécute toutes les actions associés.