La réutilisation : concepts et techniques

Abdelaziz GACEMI
gacemi@ensm-douai.fr
http ://csl.ensm-douai.fr/gacemi

Djamel Seriai
seriai@ensm-douai.fr
http ://csl.ensm-douai.fr/seriai

Dépt. G.I.P
Ecole des Mines de Douai
941 rue Charles Bourseul - B.P. 8 8
59508 Douai Cedex - France

Technical Report 2003-4-2
Ecole des Mines de Douai
Avril 2003

Table des matières

Table des figures

1 Introduction

La complexité croissante des systèmes et leur incessante évolution rend le développement avec les approches usuelles, dont lesquelles la construction d'un nouveau système part de rien('from scrach'), plus difficile, plus coûteux et moins fiable. D'après Boehm [ ], le seul facteur qui puisse en pratique faire augmenter la productivité des informaticiens est de réduire la quantité de code nécessaire à réaliser une fonctionnalité donnée, en d'autre terme «programmer moins pour programmer mieux». Une approche permettant d'atteindre cet objectif est celle de construire un système en *réutilisant* des composants existants ayant été produits à l'occasion de développements antérieurs.

Le concept de réutilisation a été évoqué par M. McIlroy en 1968 dans une conférence de l'OTAN consacrée à la crise de logiciel où il a proposé une solution basée sur la réutilisation de bibliothèques de composants logiciels[1]. Depuis, cette idée a évolué et la réutilisation est devenu une discipline et un thème de recherche à part entière. Comme dans le passé en ingénierie des systèmes, les activités de programmation ont été les premières concernées par le paradigme de réutilisation. Les tendances actuelles exploitent la réutilisation dans les tâches d'expression des besoins, d'analyse et de conception [20].

Ce chapitre présente les techniques actuelles en matière de réutilisation dans toutes les étapes de cycle de vie d'une application(analyse, conception, implémentation) ainsi que les aspects limitant l'intégration de ces techniques dans l'ingénierie des systèmes.

2 Ingénierie par et pour la réutilisation

Dans son article[1], M. McIloy a parlé de «*software components subindustry*», c'est à dire un axe de développement dans l'approche par réutilisation dont le but n'est pas de produire des applications prêtes à être utiliser mais il s'intéresse seulement à l'accélération du processus de développement des applications en fournissant des composants prêts à être réutiliser. Ce type d'activités, qui correspond à l'aspect « infrastructure » de la réutilisation, est appelé l'ingénierie des composants pour la réutilisation « Design for reuse ». En complément à cette discipline d'ingénierie, le monde d'ingénierie d'applications aura besoin des mécanismes permettant d'exploiter les composants réutilisables, ce qui définit une nouvelle forme d'ingénierie, qui correspond à l'aspect « opérationnel » de la réutilisation, appelée l'ingénierie d'applications par réutilisation « Design by reuse ».

[1]L'article présenté dans la conférence de l'OTAN [13]

2.1 L'ingénierie des composants pour la réutilisation

2.1.1 Les étapes à suivre

La problématique de génération des composants réutilisables demande de résoudre trois sous problématiques qui s'enchaînent au cours de la mise en œuvre de cette approche, à savoir :

2.1.1.1 Identification L'identification de composants réutilisables a pour objectif de trouver les connaissances invariantes entre plusieurs applications. On distingue essentiellement deux approches qui visent l'identification [4] :

- **Approches par la rétro-ingénierie :** qui est basées sur l'analyse et le recyclage des produits existants.
- **Approche par l'analyse du domaine :** qui se définit comme l'activité d'identification des objets et les opérations d'une classe de systèmes similaires dans un domaine particulier. Les travaux de Jim Neighbors[15] sont représentatifs de cette approche.

2.1.1.2 Spécification : La recherche dans le catalogue des composants réutilisables se fait selon des critères abstraits. La spécification consiste à définir les connaissances qui servent comme des critères de recherches. Parmi les formes de connaissances qu'on peut trouver, une description fonctionnelle du composant, la description du problème et la situation d'application du composant (comme les patterns [2]),etc.

2.1.1.3 Organisation : L'organisation des composants doit être pensée, aussi, dans un souci de faciliter la recherche et la sélection d'un composant devant satisfaire les spécifications des besoins d'un problème de développement donné. Contrairement à la spécification qui s'intéresse à un composant pris individuellement, l'organisation s'intéresse à l'ensemble des composants et les relations sémantiques entre ces composants d'une manière qui permet de traduire le processus de recherche.

2.1.2 Exemple

Afin de bien comprendre les trois étapes décrites précédemment, Je présente un exemple [3] simple qui consiste à trouver une conception d'un catalogue de composants destinés à être réutilisés pour la résolution des équations mathématiques. Les trois points suivants présentent la solution proposée :

[2] Le concept de pattern sera détaillé ultérieurement

[3] Cet exemple reste loin de montrer les difficultés rencontrées dans la pratique dans des systèmes réels et complexes

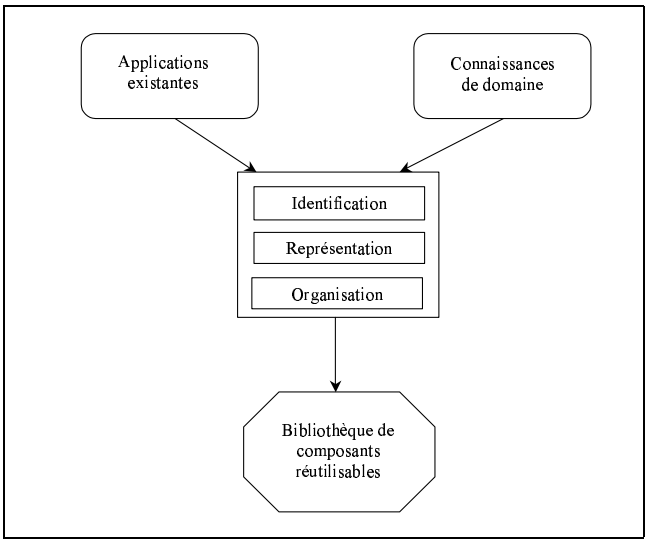

FIG. 1 – Ingénierie des composants réutilisables (Design for reuse)

a) Une identification simple et claire est celle que chaque composant servira à la résolution d'une seule classe d'équations de degré n.

b) La spécification la plus simple est d'associer à chaque composant le degré de l'équation qu'il peut résoudre, la sélection se fait selon ce critère.

c) On peut dire que les équations de degré $n-1$ est un cas spécial des équations de degré n. Donc cette relation sémantique, qui est une relation d'inclusion, est un exemple d'organisation des composants.

2.1.3 Les critères de classification

La réutilisation de logiciel inclut toute une variété de ressources utilisées et produites pendant tout le cycle de vie du logiciel. Plusieurs critères plus ou moins pertinents, ont été utilisés pour caractériser ces ressources réutilisables et proposer des classifications, parmi eux :

2.1.3.1 La nature de la connaissance : De ce point de vue, on s'intéresse au type de la connaissance exprimée par le composant réutilisable qui peut être un produit ou un processus[4].

[4]La notion de ces deux types de connaissances sera bien claire quand on montre comment les réutiliser

- **Produit :** Dans ce cas, il s'agit d'un fragment autonome passif (entité logicielle ou conceptuelle) de produits qu'il est possible de l'adapter. Les composants logiciels et les bibliothèques de fonctions mathématiques font partie de ce type.

- **Processus :** Ceci correspond à une suite d'actions qu'il faut réutiliser pour avoir un produit final. Ces d'actions sont souvent encodées dans un processeur(unité de traitement). On peut citer comme exemple l'ensemble d'algorithmes qui proposent des solutions génériques à des classes de problèmes présenté par D. Knuth dans son excellent livre « The art of computer programming »[12].

2.1.3.2 Le cycle de développement : C'est le critère le plus utilisé, qui consiste à organiser les composants réutilisables en fonction de l'étape du cycle de développement dans laquelle les composants sont produits ou utilisés.

- **Pendant l'analyse :** De nombreux travaux ont abordé la réutilisation dans cette phase de développement mais dans un domaine spécifique. On peut citer les travaux de Prieto-Diaz[16] qui a donné les fondements de la réutilisation des connaissances du domaine, c'est à dire la capture et la caractérisation d'artefacts [5] récurrents : un compte bancaire en finance avec tout un ensemble d'états et d'opérations type constituant son interface[2]. Une autre forme de réutilisation en phase d'analyse est représentée par les patterns d'analyse (analysis patterns) de Fowler [8] qui permettent de mettre au point un modèle d'analyse dans le domaine de commerce ainsi que les patterns de Coad[5] qui sont génériques indépendants du domaine d'application. Si on fait l'analogie avec les diagrammes d'UML, ces travaux assistent l'analyseur de faire essentiellement le diagramme de cas d'utilisations.

- **Pendant la conception :** Les composants réutilisables pendant cette étape décrivent d'une manière abstraite la solution. Cette dernière peut correspondre à une véritable architecture logicielle comme les frameworks ou ou encore des solutions réutilisables comme les schémas de conception (design pattern). Ces deux concepts seront détaillés ultérieurement.

- **Pendant l'implémentation :** La réutilisation pendant cette phase correspond à la forme la plus ancienne et la plus répandue. Les composants de ce niveau prennent plusieurs formes (objet, fonction, structure de donnée, texte..) mais la plus courante celle basée sur la réutilisation de fragments de code (comme les bibliothèques de fonctions mathématiques). De point de vue processus, on peut citer les générateurs d'applications qui est une forme de réutilisation dans la phase d'implémentation.

[5] Une information qui est utilisée ou produite par une démarche de développement de logiciel. Un artefact peut être un modèle, une description ou composant logiciel.

2.1.3.3 La portée du composant : Ce critère caractérise le degré de la réutilisabilité d'un composant qui peut concerner un domaine bien spécifique comme elle peut être indépendante du domaine.

- **Les composants orientés application :** Ce sont des composants spécifiques à une application donnée. Ils sont peu réutilisables. Ce type de composants est généralement engendré par des réutilisations ad-hoc, c'est à dire non planifiées.

- **Les composants orientés domaine :** appelés aussi les composants métiers. Ce sont des composants réutilisables à travers les applications d'un même domaine. Une étude récente confirme que le problème de réutilisation doit impérativement se poser sous cet angle « métier »[2]. Notons que le concept de composant métier ce n'est qu'une extension du concept « objet métier». D'ailleurs, OMG[6] a définit dans son architecture CORBA[7] les interfaces du domaine (domain interfaces) qui définissent des objets de métiers spécifiques à des secteurs d'activités comme la finance (e.g. monnaie électronique), la santé (e.g. dossier médical) et les télécoms (e.g. transport multimédia)[10].

- **Les composants "domaine indépendant" :** Ce sont des composants généraux qui ne dépendent pas d'un domaine particulier. Les patrons de conceptions de Gamma[9] sont des exemples d'un tel type.

Dans d'autres classifications, la portée peut prendre un autre sens supplémentaire, c'est celui de la portée horizontale qui caractérise le degré de la réutilisabilité à travers les étapes de cycle de développement. Dans ce cas, on s'intéresse au fait que la composant est conçu pour être utilisé dans une étape bien particulière de développement ou s'il peut conserver le sens de la réutilisabilité pendant plus d'une étape. Très peu de travaux se sont intéressés à ce dernier cas.

2.1.3.4 Le niveau d'abstraction : Il définit le niveau de visibilité des détails internes d'un composant réutilisable qu'on a besoin lors de la réutilisation. Ce qui a un impact direct sur la manière d'utilisation. Par exemple, s'il s'agit des composants d'implémentations qui sont définit par leurs interfaces, le niveau d'abstraction définit le degré de visibilité de l'implémentation de ces composants. On distingue deux niveaux principaux représentées par ces deux concepts [21] :

- **Boites blanches :** L'utilisateur d'une entité de ce type a la visibilité de tous les détails . On prend comme exemple les objets ou le source code est fournit et la réutilisation se fait en étendant le code source de la classe de

[6]Object Management Group
[7]Common Object Request Broker Architecture

l'objets. Parmi les mécanismes de la réutilisation, on trouve l'héritage.

- **Boites noires :** Dans ce cas, les droits de l'utilisateur sont très rétreints. Aucun détail ne lui est fournit autre que l'interface et les spécifications. Un composant logiciel est un exemple typique de ce type de boites, et dans ce cas la réutilisation ne se fait que par le paramétrage et l'assemblage des composants.

La couleur ne représente pas une dichotomie stricte. En fait, il existe une multitude de couleurs intermédiaires qui représente des niveaux d'abstraction intermédiaires entre les boites noires et les boites blanches.

2.1.3.5 La granularité La notion de composants réutilisables regroupe des entités atomiques comme les structures de données et les fonctions mathématiques mais aussi des véritables structures complexes comme les frameworks et entre les deux niveaux, on trouve les composants logiciels et les patrons de conceptions. D'ou, de point de vue de granularité, ils ne sont pas pareils.

2.2 L'ingénierie des applications par la réutilisation

2.2.1 Les stratégies de réutilisation

Après avoir vu les différents techniques et concepts concernant l'ingénierie des composants réutilisables, la question qui se pose : comment réutilise-t-on les composants ?, en d'autres termes, quelles sont les stratégies et les méthodes de réutilisation ?. La réponse est que cela dépend du type de la connaissance à réutiliser. On distingue deux approches de réutilisation. Dans le cas des composants de type "produit" c'est l'approche par composition qu'il faut appliquer. S'il s'agit des composants de type "processus", c'est l'approche générative qu'on doit faire appel.

2.2.1.1 Approche par composition : Dans cette approche, la construction d'applications se fait par composition des composants réutilisables existants. Cette approche exige de disposer d'un langage, ou un outil, permettant de décrire la composition qui se prête bien à la vérification et à la validation :

- La vérification qui consiste à vérifier si la composition est faisable. Généralement, la vérification porte sur les interfaces des composants.

- La validation qui consiste à voir si la composition satisfait les spécifications souhaitées.

La tendance actuelle en matière de développement d'applications opte pour cette approche. On peut citer comme exemple les travaux sur l'assemblage des composants logiciels.

2.2.1.2 Approche générative : Dans ce cas, la réutilisation concerne un composant actif qui permet via un processus de génération de produire des applications. Ce composant a comme entrées des règles et des spécifications permettant de spécifier le produit. On distingue trois catégories de ce type de composants :

- **Les générateurs de code :** Comme indiquent leurs noms, ce sont des composants qui permettent de générer du code source en recevant comme entrée les spécifications du produit. LEX[8] et YACC[9] sont deux exemples typiques d'une telle approche. LEX est un générateur d'analyseurs lexicaux, il reçoit en entrée les spécifications des entités lexicales d'un langage(les tokens), il génère du code en C ou en pascal (selon la version de LEX). Ce code correspond à un analyseur lexical. YACC est un générateur d'analyseur syntaxique. Il reçoit en entrée la grammaire du langage cible, qui représente sa spécification, il génère le code de l'analyseur syntaxique.

- **Les systèmes de transformations :** Ce sont des systèmes qui permettent de transformer ce qu'ils reçoivent en entrée selon les règles de transformation. Citons comme exemple les processeur[10] XSL qui permet de transformer des documents XML en d'autres documents XML ou HTML en se basant sur des règles de transformation écrites à l'aide du langage XSLT[11][2].

- **Les générateurs d'applications :** Ce sont des systèmes qui permettent de générer des applications à partir des spécifications de haut niveau. C'est le cas du système « Draco » mis en œuvre par J. Neighbors[14][17].

2.2.2 Les problématiques sous-jacentes

Après la production des composants réutilisables, leur utilisation n'est pas systématique mais passe par la résolution de trois sous-problématiques qui sont plus ou moins difficiles selon le produit à réutiliser.

2.2.2.1 Recherche et sélection : Étant donné un catalogue de composants réutilisables. La recherche consiste à trouver un composant qui satisfasse des critères de recherche et réponde à un problème particulier de développement. La difficulté de cette étape réside dans le fait que dans la plupart des situations, il n' y a pas un seul composant candidat qui répond au besoin mais plusieurs composants candidats, à cause de la non conformité exacte entre les spécifications des composants et les critères de recherche. Dans ce cas, il est nécessaire

[8] LEXical analyzer generator
[9] Yet Another Compiler-Compiler
[10] J'ai gardé la terminologie de l'organisme W3C
[11] Extensible Stylesheet Language Transformations

de mettre en œuvre un mécanisme de sélection qui choisisse le composant qui est le plus proche des besoins exprimés.

2.2.2.2 Adaptation : Les composants sélectionnés ont besoin, dans certaines situations, d'être soumis à des changements afin qu'ils soient adéquats pour un besoin particulier. On recense plusieurs techniques d'adaptation, telles que la paramétrisation, la spécialisation et l'instantiation.

2.2.2.3 Intégration : La réutilisation devient effective si le composant est intégré au produit en cours du développement et est devenu opérationnel. A mon avis, Cette étape interfère avec la notion d'assemblage s'il s'agit de construire un nouveau système à partir des composants existants alors qu'aucune hypothèse sur l'existence d'un ancien système n'est faite.

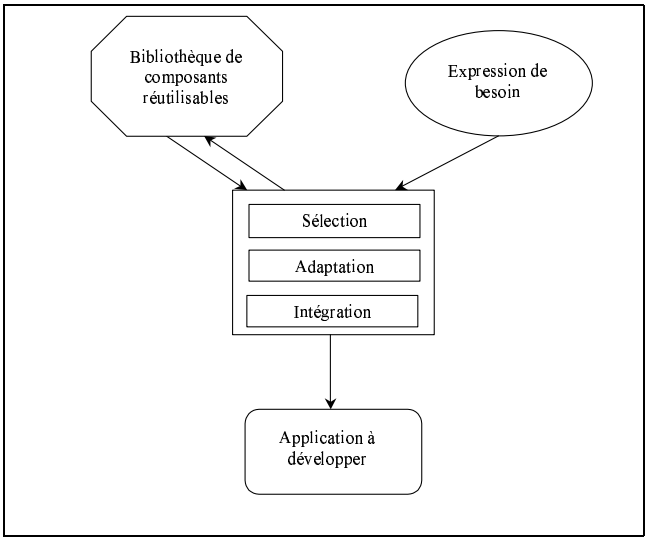

FIG. 2 – Ingénierie d'applications par réutilisation (Design by reuse)

3 Les schémas de collaborations

Les activités de développements pour la réutilisation et celles de développement pour la réutilisation sont des processus indépendants. Cependant, une meilleure organisation ainsi qu'une meilleure collaboration entre les deux processus peut améliorer la productivité de l'approche de la réutilisation. Dans la littérature, on distingue deux organisations de collaboration (voir la figure .2)

3.1 Organisation producteur/consommateur

Dans ce type d'organisation, le processus de développement des composants réutilisables représente le producteur et le processus de développement basé sur la réutilisation représente le consommateur. Ce dernier ne participe pas à la spécification des composants produit. Il se contente que de consommer. Le couplage entre les deux processus est faible .

3.2 Organisation entrelacée

Dans ce cas, le processus de développement basé sur la réutilisation consomme les composants réutilisables mais contribue aussi au développement des composants en fournissant les caractéristiques souhaitées d'un composant produit au processus de production.

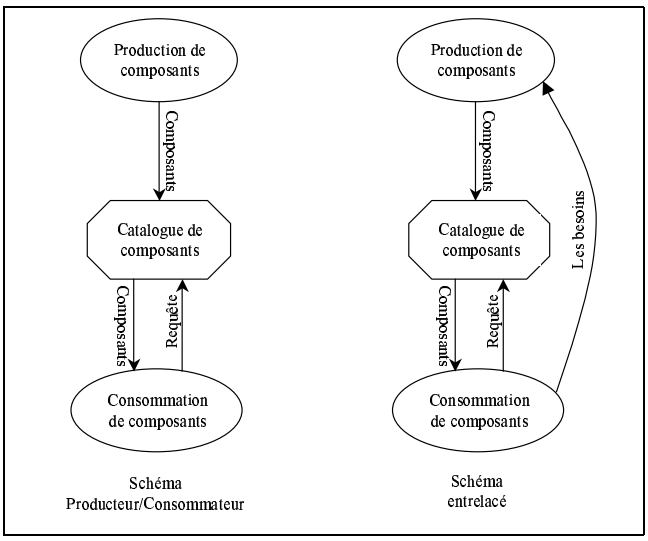

FIG. – Les schémas de collaboration

4 Les formes de réutilisation dans l'ingénierie des systèmes

Plusieurs techniques ont été proposé, toutes dans l'objectif d'offrir aux analyseurs, concepteurs et développeurs une meilleure infrastructure de réutilisation. Je présente ici quelques exemples les plus répandus.

4.1 Les ontologies

Une ontologie est une base de connaissances contenant une représentation structurée et opérationnelle d'un ensemble de concepts et de leurs relations sémantique. Dans le domaine des systèmes multi-agents, une ontologie est utilisé pour avoir une compréhension consensuelle des connaissances d'un domaine donnée afin de résoudre les problèmes de communication entre les agents coopérants du système[22]. Une ontologie est une forme de réutilisation de connaissances.

4.2 Les patrons de conception

Un patron de conception(*design patterns*)est une micro architecture décrivant un problème de conception récurrent et l'architecture de sa solution qui devient réutilisable pour toute situation relevant de ce problème.

Historiquement, le terme de patron de conception a été introduit par l'architecte Christopher Alexender qui assimile le patron à un savoir-faire formalisé « Chaque patron décrit à la fois un problème qui se produit très fréquemment dans votre environnement et l'architecture à ce problème de telle façon que vous puissiez cette solution des millions de fois sans jamais l'adapter deux fois de la même manière »[1][2].

Un catalogue de patrons de conception a été proposé par Gamma et *al.*[9]. regroupant 2 schémas de conception. Un « patron de Gamma » est formalisé à l'aide des éléments suivants :

- Le nom du pattern
- Le problème à résoudre
- Le contexte d'utilisation(problèmes particuliers, indication d'utilisation)
- La solution qui est représentée graphiquement à l'aide de la notation OMT[19]
- Des exemples concrets

La recherche dans le catalogue des patterns se fait selon le problème qu'on cherche à résoudre. Vu que les patrons sont des entités de conception abstraites, une étape d'adaptation au domaine de l'application est nécessaire. l'adaptation peut correspondre à une simple imitation de la solution ou une réelle instantiation. Pour l'intégration, il s'agit de relier les différents patrons constituant l'application[2].

4.3 Les frameworks

Les frameworks ont été étudiés dans le contexte de Smalltalk[7] mais ils ont été adoptés comme étant une approche générale dans le domaine de l'orienté objets.

Un framework est un ensemble de classes coopérantes, parmi elles il pourrait y avoir des classes abstraites, qui forme une conception réutilisable pour une catégorie spécifique de logiciel[21]. Avec les frameworks, le grain de réutilisation n'est pas les classes mais la réutilisation d'un système constitués de classes en décrivant comment leurs instances interagissent. Une idée clé dans les frameworks est l'utilisation des classes abstraites[11].

Les frameworks peuvent être considérés comme une forme de génération d'applications du fait qu'ils définissent des maquettes d'une application qui peuvent être personnalisés par un développeur[11].

Parmi les caractéristiques importantes des frameworks est *l'inversion de contrôle*. Traditionnellement, un développeur réutilise les composants à partir d'une bibliothèque. C'est à la charge du développeur de faire les appels des composants et de définir l'aspect structurel et comportemental de son application. Dans le cas d'un framework, l'utilisateur implémente les classes abstraites alors que la structure et le comportement de l'application seront totalement définit par le framework.[11]

En comparaison avec les patrons de conception, les frameworks sont moins abstraits. Leur portée, qui dépend généralement d'un domaine bien spécifique, est moins large que celle d'un patrons de conception et leur granularité est plus grande. D'ailleurs, un framework peut être vu comme une architecture composée d'un ensemble de classes et des patrons de conception. Ces derniers doivent être adaptés en leur diminuant leur abstraction jusqu'au le niveau souhaité[18].

4.4 Les bibliothèques

Les bibliothèques constituent la technique de la réutilisation la plus courante actuellement. Elles permettent particulièrement la réutilisation du code. L'utilisateur leur fait appel sans que une étape d'adaptation soit nécessaire.Souvent, elles sont faites pour implémentés des fonctionnalité élémentaires de granularité fine telles que les bibliothèques de fonctions mathématiques.

4.5 Les composants logiciels

Un composant logiciel[12] est une unité de composition qui assure une fonctionnalité spécifique, possède des interfaces de besoins et des interfaces de services et des contextes particuliers d'exécutions. Il peut être déployé indépendamment et composé avec d'autre composants[2]. La notion de composant peut être couplée avec celle du framework pour avoir ce qu'on appelle « component framework», et dans ce cas, les classes seront remplacés par des composants.

[12]Tout un chapitre sera consacré aux composants logiciels

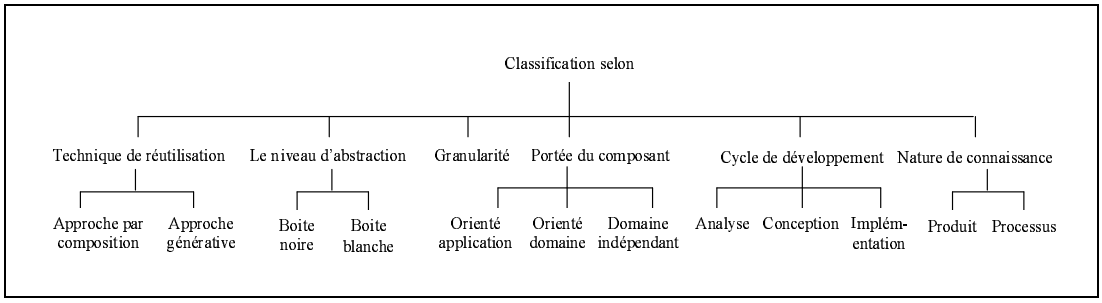

FIG. 4 – Taxonomie de la réutilisation

5 La réutilisation en C++

Pour éclaircir quelques concepts qu'on vient de les voir tout au long des sections précédentes, Je propose d'étudier brièvement le cas de la réutilisation dans le langage C++ qui est basée sur la réutilisation de code.

Commençons par le concept de classe qui est mécanisme pour la réutilisation. En fait, les objets instanciés d'une même classe réutilisent sa structure et le comportement décrits par la classe. Dans ce cas, l'adaptation de l'entité réutilisée se fait par instanciation

Une autre forme de réutilisation très courante, fournie presque par tous les langages orientées objets, il s'agit de l'héritage. Le code de la classe héritée,la classe de base, est étendu afin de produire une nouvelle classe. Ceci dit que le code de la classe de base est bien disponible ce qui implique qu'on est dans le cas d'une réutilisation « des composants de type boite blanche ». Notons que, dans ce cas, l'adaptation du composant réutilisé se fait par la spécialisation

Le langage C++ propose aussi une forme de réutilisation flexible basée sur le paramétrage des classes, Il s'agit des classes génériques, appelées aussi les patrons de classes. Le fragment de code suivant[13] écrit en C++ décrit l'entête de la classe implémentant le fameux schéma de collaboration « Producteur/Consommateur ». la réutilisation se fait juste en paramétrant la classe patron (par exemple CProd_Cons< int > fileEntier).

```
// La description de la classe CProd_Cons
template < class T > class CProd_Cons
{
  private :
          int Buffer_Size ;
```

[13]Cet exemple est tiré de mes programmes réalisés durant mon stage de fin d'étude

```
        int Indice _Lecture ;
        int Indice _Ecriture ;
        T * Buffer ;
        int Indice _Lecture ;
        int Indice _Ecriture ;
        CSemaphore Sem_Lecture ;
        CSemaphore Sem_Ecriture ;
        CSemaphore Mutex_Lec ;
        CSemaphore Mutex_Ecr ;

public :
        CProd_Cons(int) ;
        ~CProd_Cons( ) ;
        T Lire( ) ;
        void Ecrire( T ) ;
}
```

Cette classe présente deux dimensions de réutilisation

- De point de vue conceptuelle, ce code représente le pattern servant à résoudre les problèmes liés au partage de ressources entre des processus concurrents dans un environnement parallèles. C'est une réutilisation au niveau conceptuel.

- Au niveau d'implémentation, ce code est réutilisable via le paramétrage de la classe CProd_Cons .

Le langage C++ propose ainsi un concept très proche des patrons de classes basée cette fois-ci sur le paramétrage des fonctions, il s'agit des patrons de fonctions(template function). voici un exemple de déclaration de ce type de fonctions :

```
 // Déclaration de fonctions patrons portant le même nom.
template f1( T, U, float ) ;
template f1( T, U ) ;
template f1( T, W) ;
```

Ces lignes de codes représentent des prototypes de fonctions génériques. Lors de la compilation, C++ crée une sorte d'un catalogue incluant toutes les fonctions génériques. La réutilisation se fait simplement en précisant le nom du composant qu'on veut le réutiliser, et les paramètres d'adaptations, par exemple $f1(int, int, int)$, ce qui déclenche la recherche dans le catalogue des fonctions génériques. Le compilateur suit une méthodologie de recherche en se basant, au début, sur les composants comportant le même nom de fonction. Si plusieurs choix candidats se présentent, ce qui est le cas ici, le compilateur fait la sé-

lection selon le nombre de paramètres. Ce qui permet , dans notre exemple, de sélectionner et de laisser un seul choix, il s'agit de la fonction générique $f1(T, U, float)$. Une fois que le composant est choisi, le compilateur entame la procédure d'adaptation. Tout d'abord, il remplace les deux types T, U par int. En suivant certaines règles de conversions implicites de types ($chart$ -> $short$ -> int > $long$ -> $float$ -> $double$) le compilateur décide de remplacer int par $float$.

En C++, La tâche de sélection pourrait échouer dans le cas ou des situations d'ambiguïtés se présentent. Dans ce cas, c'est le programmeur qui sera invité pour résoudre le conflit (Exemple : $f1(int, char)$). De même, la phase d'adaptation pourrait ne pas aboutir (exemple : $f1(int, int, double)$).

La construction des programmes par composition ne constitue pas le souci du langage C++, et les langages orientés objets en général. La composition se fait d'une manière ad-hoc, à travers les appels de méthodes et la manipulation des évènement pour lier les différents modules. Cependant, l'aspect structurel et comportemental de l'application reste flou.

En plus des mécanismes permettant la réutilisation, Le langage C++ fournit une librairie standard *Standart Template Library(STL)* comprenant des classes patrons pouvant être réutilisé à travers le paramétrage et représentant les structures de données les plus répandues telles que les vecteurs, les listes, les piles, les tableaux associatifs, les tables de hachage... [6]

6 Conclusion

Dans ce chapitre, j'ai présenté les différents concepts liés à la réutilisation, les différentes classifications ainsi que les critères qui sont à la base de ces classifications J'ai terminé par présenter quelques techniques de réutilisation mises en pratique, voire commercialisées.

Une conclusion que je peux faire est que le domaine de la réutilisation, malgré tous les travaux faits durant trois décennies, est un domaine ou il reste beaucoup à faire. Ceci dit que le problème de la réutilisation n'est pas encore résolu tant que on n'arrive pas à faire disparaître le coté technique de la programmation dans le développement des applications, et de laisser que le coté métier qui concerne les problèmes du domaine afin de donner aux experts du métier la possibilité , dans une durée courte et avec plus efficacité, de « monter », d'« assembler », de « configurer » leurs applications eux-même, selon leurs savoir-faire, sans avoir besoin des compétences techniques de développement.

La réutilisation est aujourd'hui pratiquée avec des composants logiciels. Les modèles à base de composants logiciels s'avèrent prometteurs et suscitent des études plus approfondis. Pour cette raison, les composants logiciels feront l'objet

du chapitre suivant. Notons que la compréhension complète des concepts et des techniques de réutilisation est très important pour aborder le monde des composants[21].

Références

[1] Christopher Alexander. *The Timeless Way of Building*, volume 1 of *Center for Environmental Structure Series*. Oxford University Press, New York, NY, 1979.

[2] Franck Barbier, Corine Cauvet, Mourad Oussalah, Dominique Rieu, and Carine Souveyet. Composants dans l'ingénierie des systèmes d'information : concepts clés et techniques de réutilisation. *Assises Nationales du GDR I3 : : information - interaction - intelligence*, December 2002.

[ ] Barry W. Boehm. Improving software productivity. *Computer*, 20(9) :4 – 57, 1987.

[4] Yasmina Chikhi. *Réutilisation de structures de données dans le domaine des réseaux électriques*. Thèse de doctorat, Université Pierre et Marie Curie Paris VI, Paris, juillet 1998.

[5] Peter Coad, David North, and Mark Maryfield. *Object Models : Strategies, Patterns, and Application*. Yourdon Press, 1995.

[6] Claude Delannoy. *Programmer en langage C++*. Edition Eyrolles, France, deuxième edition, 1998.

[7] L. P. Deutsch. Design reuse and frameworks in the smalltalk-80 system. In T. J. Biggerstaff and A. J. Perlis, editors, *Software Reusability*, pages 57–71. ACM Press, New York, 1989.

[8] Martin Fowler. *Analysis patterns : reusable objects models*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, 1995.

[10] J. M. Geib and C. Gransart et P. Merle. *CORBA : Des concepts à la pratique*. Masson Editeur, 1997.

[11] Ralph E. Johnson. Components, frameworks, patterns. In *ACM SIGSOFT Symposium on Software Reusability*, pages 10–17, 1997.

[12] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 10 January 197 .

[1] M. D. McIlroy. Mass produced software components. In P. Naur and B. Randell, editors, *Proceedings, NATO Conference on Software Engineering*, Garmisch, Germany, October 1968.

[14] J. Neighbors. Software construction using components, 1980.

[15] Jim Neighbors. *Software Constructon using Components*. PhD thesis, Departement of Information and Computer Science, University of California, irvine, 1981.

[16] R. Prieto-Diaz. Domain analysis : an introduction. *ACM SIGSOFT - Software Engineering Notes*, 15(2) :47–54, april 1990.

[17] R. Prieto-Diaz. Status report : Software reusability. *IEEE Software*, 10( ) :61–66, May 199 .

[18] Pascal Rapicault. *Modèles et techniques pour spécifier, développer et utiliser un framwork : une approche par méta-modélisation.* Thèse de doctorat, Université de Nice Sophi-Antipolis, Nice, Mai 2002.

[19] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall International Editions, New York, NY, 1991.

[20] Farida Semmak. *Réutilisation de composants de domaine dans la conception des systèmes d'information.* Thèse de doctorat, Université Paris I, Paris, Février 1998.

[21] Clemens Szyperski. *Component software : beyond object-oriented programming.* ACM Press/Addison-Wesley Publishing Co., 1998.

[22] Mike Uschold and Michael Grüninger. Ontologies : principles, methods, and applications. *Knowledge Engineering Review*, 11(2) :9 –155, 1996.

[2] World Wide Web Consortium. XSL transformations XSLT, 1999. W C Recommandation, www.w .org/TR/xslt.