GENOME : un Modèle pour la Simulation d'Émergence de Structures d'Objets

Dalila Tamzalit

UNIVERSITE DE NANTES

UFR SCIENCES ET TECHNIQUES

ÉCOLE DOCTORALE

**SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATERIAUX**

2000

Thèse de DOCTORAT

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

Dalila TAMZALIT

Le 14 décembre 2000

A la Faculté des Sciences et Techniques, Université de Nantes

GENOME : un Modèle pour la Simulation d'Émergence de Structures d'Objets

Jury

Président	:	Jean Jacques CHABRIER, Professeur, Université de Bourgogne
Rapporteurs	:	Annie CAVARERO, Professeur Université de Nice Dominique RIEU, HDR LSR IMAG – INPGrenoble
Examinateurs	:	Marie Françoise COLINAS, Responsable département CNET Noureddine MOUADDIB, Professeur, École Polytechnique de Nantes Jean-François SANTUCCI, Professeur, Université de Corte

Directeur de Thèse : Mourad Chabane OUSSALAH
IRIN,
2 rue de la Houssinière
BP 92208
44322 NANTES cedex 3

N° ED 0366-xxx

TABLE DES MATIERES

LISTE DES FIGURES[1]

[1] Les figures sont identifiées par le **numéro de chapitre** et leur **ordre d'apparition séquentiel** dans ce même chapitre, ainsi que le **numéro de page** dans le manuscrit, sur le modèle : *Figure numChapitre-numFiguredansChapitre* et le *numéro de page*.

LISTE DES TABLEAUX[2]

[2] Les tableaux sont identifiés par le **numéro de chapitre** et leur **ordre d'apparition séquentiel** dans ce même chapitre, ainsi que le **numéro de page** dans le manuscrit, sur le modèle : *Tableau numChapitre-numTableaudansChapitre* et le *numéro de page*.

INTRODUCTION

1.1 - Cadre de l'étude

Les caractéristiques de l'approche objet telles que l'héritage, la composition, l'abstraction, le polymorphisme… permettent de modéliser des systèmes et de développer des applications de plus en plus robustes mais en même temps de plus en plus complexes, ce qui a immanquablement posé le problème crucial de l'évolution, de l'adaptation et de la maintenance de ces systèmes et de ces applications. Grâce à ces caractéristiques, l'approche objet est souvent vue comme l'approche par excellence rendant possible l'extension et l'adaptation d'un système en réutilisant le plus possible l'existant. Il est vrai que les méthodes d'analyse et de conception orientées objet contribuent au développement d'applications ouvertes mais restent malheureusement confrontées aux problèmes d'évolution de spécifications existantes et d'auto-adaptation. En effet, la plupart des stratégies d'évolution intègrent uniquement le concept d'évolution mais n'intègrent pas le processus d'analyse et de conception de l'évolution.

Il est connu que les besoins, aussi bien en termes de spécifications que d'évolution, peuvent être non identifiés ou mal définis. En effet, ces problèmes liés à l'évolution peuvent concerner un modèle et apparaître durant tout son cycle de vie (conception, utilisation, maintenance).

Les modèles à objets courants [Banerjee& 87], [Bratsberg 93], [Casais 91], [Kim& 88], [Nguyen& 89], UML [Rational 97], C++ [Stroustrup 96], Java [Sun 98] … souffrent de manque de concepts pour prendre en compte les besoins et lacunes suscités mais en même temps disposent de puissants concepts (héritage, encapsulation, surdéfinition…) pour adresser de tels problèmes. Notre objectif est de répondre à ces insuffisances en proposant un modèle objet évolutif visant à :

– offrir à un concepteur un outil permettant de simuler l'évolution d'objets sur des applications existantes ;

– expliciter les différents processus d'évolution d'un objet aussi bien durant la conception que durant l'exploitation.

1.2 - Objectif : un Modèle pour la Simulation de l'Évolution d'Objets

Nous considérons que l'évolution d'objets peut aussi bien être abordée à partir des spécifications conceptuelles, donc des classes et de leur hiérarchie, qu'à partir des instances quand cela est utile. Nous proposons une approche complémentaire de l'évolution d'objets classique, qui est généralement axée sur l'évolution de classes. Pour permettre de faire face à des situations de changements imprévues dans le cadre d'applications d'ingénieries complexes et volumineuses, nous proposons à un concepteur un modèle d'évolution basé sur la simulation. Notre objectif est de rechercher et de dégager diverses voies d'évolution des structures conceptuelles à partir de l'évolution dynamique de la structure d'instances. Afin de ne pas provoquer de rupture dans le cycle de vie du système, nous permettons au concepteur d'exprimer directement de nouveaux besoins sur des instances, et de simuler diverses voies possibles d'évolution. Nous nous assurons que cette simulation reste cohérente avec les spécifications existantes. Le principe général sur lequel nous nous reposons est d'arriver à faire émerger des structures conceptuelles adaptées et plus complètes à partir des besoins ponctuels et des informations présentes dans la base. Nous permettons par la suite de choisir la structure la plus intéressante puis de détecter les endroits d'insertion et de changements dans le modèle de classes, et de cerner les impacts aussi bien au niveau du modèle de classes que sur les instances existantes. Le processus d'évolution que nous proposons s'appuie sur des processus de recherche de données plus ou moins adaptées au problème, d'extraction et de combinaisons de spécifications. Ceci fait l'objet en détails de ce manuscrit dans les chapitres 3, 4 et 5.

Nous abordons dans ce manuscrit notre approche de la problématique d'évolution d'objets, ainsi que les solutions que nous préconisons. Le manuscrit est organisé en chapitres. Voici son plan de lecture :

1.3 - Plan du manuscrit

Outre cette introduction, le manuscrit est organisé en six chapitres principaux, suivis d'une conclusion :

1.3.1. Chapitre 1 : Contexte de travail, motivations et hypothèses de travail

Dans ce chapitre, nous situons et décrivons le contexte initiateur de notre travail. Il s'agit d'un contexte de modélisation de réseaux de télécommunications. La description du contexte nous permet d'introduire, au vu d'un exemple illustratif, la problématique d'évolution d'objets à laquelle nous nous intéressons.

En effet, la problématique initiatrice de nos travaux est celle de l'évolution des réseaux de télécommunications face à des situations et des besoins d'évolution nouveaux. Ces situations et ces besoins peuvent ne pas être nouveaux mais peuvent être partiellement définis ou mal spécifiés.

Pour un concepteur, la gestion de tels besoins d'évolution est d'autant plus accrue que ces applications sont particulièrement volumineuses et complexes. Ce type de problèmes se rencontre dans tout type d'applications importantes, tel qu'en CAO par exemple. De plus, Le concepteur, qui a la charge de telles applications, a du mal à faire face à des évolutions imprévues ou mal cernées. La seule certitude dont dispose le concepteur est l'endroit ponctuel et local où il peut apporter des modifications, à savoir une ou plusieurs instances existantes.

Aussi, nous mettons en avant dans ce chapitre nos motivations : la problématique de l'évolution d'applications complexes et volumineuses face à des situations de changements imprévues nous conduit à aborder l'évolution à partir de l'évolution dynamique de la structure d'instances. Celles-ci permettent alors l'apparition de nouvelles structures conceptuelles : c'est l'*Émergence*.

Nous complétons ce chapitre en posant clairement nos différentes hypothèses de travail ainsi que quelques définitions que nous jugeons nécessaires, notamment celle de produit pour désigner toute structure conceptuelle (classe, attribut, lien, instance). Ces hypothèses de travail permettent de faciliter la compréhension au lecteur du manuscrit, et par conséquent la démarche de notre travail.

1.3.2. Chapitre 2 : État de l'Art

Nous consacrons ce second chapitre à l'état de l'art. Nous en consacrons la majeure partie à l'évolution d'objets. Nous consacrons la seconde partie à l'Évolution Artificielle car elle représente notre source d'inspiration pour quelques aspects, notamment l'idée de permettre l'évolution dynamique d'instances et d'en tirer profit.

Nous nous intéressons à plusieurs aspects dans la principale partie de l'état de l'art :

– Notre premier intérêt se porte sur la description de l'évolution d'objets par rapport à ses propres caractéristiques, que nous dénommons *facettes*. Nous définissons ainsi l'*objet de l'évolution*, le *type de l'évolution* et le *processus de l'évolution*. L'objectif est de décrire complètement l'évolution d'objets au travers de ses caractéristiques et de pouvoir classifier les stratégies d'évolution existantes par rapport à ces facettes. Nous définissons ainsi une représentation triptyque de l'évolution d'objets.

– Nous nous intéressons par la suite à une problématique particulière de l'évolution d'objets. Nous nous focalisons en effet sur la problématique de l'évolution du produit (aussi bien l'évolution de classes que l'évolution d'instances). Pour cela, nous nous intéressons aux principales stratégies d'évolution par modifications de classes, car elles présentent la même approche que la nôtre : celle de modifier directement les structures conceptuelles existantes. Nous rappelons pour chacune d'entre elles son principe, ses avantages et ses inconvénients.

– Pour pouvoir mieux comprendre leur fonctionnement et comparer les stratégies d'évolution par modifications de classes, nous élaborons une taxonomie d'opérations d'évolution. Nous proposons une taxonomie d'*opérations de base* et d'*opérations avancées*. Ces taxonomies non seulement dégagent les opérations d'évolution et leur séquencement pour les stratégies d'évolution étudiées, mais elles mettent également en évidence qu'elles ne suffisent pas à mener notre approche de l'évolution, celle de la simulation de l'évolution par l'évolution dynamique d'instances, c'est-à-dire l'*Émergence* de nouvelles structures conceptuelles.

– Nous présentons également des travaux connexes. Pour la plupart, il s'agit de stratégies d'évolution connues qui assurent l'évolution non pas par modification mais par dérivation. Toutes les stratégies étudiées sont situées par rapport aux facettes de l'évolution. D'autres travaux sont également cités. Il s'agit de divers travaux s'étant inspirés plus ou moins directement de travaux de l'Évolution Artificielle.

L'Évolution Artificielle est la partie qui clos ce chapitre de l'état de l'art. Nous présentons les principaux travaux qui nous ont inspirés dans certaines idées et certains concepts. Il s'agit des travaux de la Vie Artificielle et des Algorithmes Génétiques.

Les trois chapitres suivants se complètent et décrivent le modèle que nous proposons. Nous dénommons ce modèle GENOME pour ***G****énétique pour l'****E****volutio****N*** ***O****bjet et* ***M****odèl****E*** ou encore ***G****enetic* ***E****volutio****N*** ***O****bject Mod****El***. Le terme *génétique* est utilisé pour noter l'inspiration des Algorithmes Génétiques et de la Vie Artificielle pour traiter l'évolution d'objets face à des situations de changements imprévues.

1.3.3. Chapitre 3 : GENOME : les concepts

Nous présentons dans ce chapitre les concepts du modèle GENOME basé sur un modèle objet classique (classe/instance), enrichi de nouvelles notions et de nouveaux concepts nécessaires pour mener à bien nos objectifs.

Les concepts de GENOME sont de deux types :

– Les concepts de base qui véhiculent en fait les concepts objet de base (classe, instance, attributs, méthodes). Nous présentons ces concepts sous l'évocation 'génétique'. Ainsi, les concepts d'attribut et de méthode sont des caractéristiques élémentaires et nous les considérons comme des *gènes*. Ces gènes sont définis dans les classes qui jouent ainsi le rôle de *génotypes*. Elles permettent de créer des instances, qui sont ainsi des *phénotypes*, et forment ainsi une ou plusieurs *populations*.

– Les concepts avancés étendent les concepts de base, les enrichissent et les complètent. Nous montrons aussi que nous considérons différents types de gènes, chacun ayant un rôle et une prédominance précis. Ces concepts interviennent grandement dans les processus d'évolution, principalement le concept de *schème*.

Nous illustrons les concepts de modèle sur un exemple simple et didactique. Cet exemple revient dans les chapitres suivants également pour illustrer les différents concepts et notions que nous définissons.

Nous complétons ce chapitre en donnant la synoptique du méta-modèle pour prendre en compte l'évolution.

1.3.4. Chapitre 4 : GENOME : les processus d'évolution

Nous présentons dans ce chapitre les processus d'évolution du modèle GENOME. Nous définissons les deux principaux processus d'évolution du modèle GENOME : le *développement* et l'*émergence*. Afin de décrire un processus d'évolution, nous le définissons au travers de trois *phases* distinctes, par lesquelles passe tout processus d'évolution de GENOME.

Pour mener les étapes opératoires d'un processus d'évolution, nous définissons un certain nombre d'opérateurs que nous qualifions d'*opérateurs objets génétiques*, dont le plus important est le *croisement* qui permet l'émergence de nouvelles structures conceptuelles mieux adaptées aux nouveaux besoins.

Chaque phase d'un processus d'évolution est spécifiée de manière détaillée par un automate décrit grâce à des diagrammes d'états-transitions. Chaque cas d'évolution est illustré par au moins un exemple. Une discussion achève cette partie.

La dernière partie de ce chapitre présente le concept du *lien d'évolution*. Il s'agit d'un lien qui est spécifié entre deux classes après qu'une évolution aura été menée à terme. Il permet ainsi de *capitaliser* toute nouvelle évolution. Il permet ainsi au modèle de pouvoir réagir et répondre si une telle situation d'évolution survient à nouveau.

1.3.5. Chapitre 5 : GENOME : les métriques pour contrôler l'émergence

Nous présentons dans ce chapitre le contrôle de l'émergence dans GENOME. Nous proposons d'utiliser des métriques. Nous rappelons l'introduction des métriques dans l'approche objet et présentons quelques métriques objet standards. Nous présentons par la suite une technique, la technique GQM, que nous utilisons pour détecter avec précision à quels endroits d'un processus d'évolution une métrique peut être utile.

Nous proposons deux catégories de métriques. La première permet de mesurer certaines caractéristiques de la constitution interne d'une structure conceptuelle émergente et pour permettre de détecter d'éventuelles anomalies. La seconde catégorie de métriques permet de mesurer certains critères d'une structure conceptuelle émergente par rapport à la hiérarchie de classes existantes. Nous montrons également que l'utilisation des métriques peut parfois obéir à des exigences du concepteur. Nous dénommons ces exigences comme étant des *invariants*.

1.3.6. Chapitre 6 : Implémentation

Ce chapitre présente les parties du modèle ayant fait jusqu'à présent l'objet d'une implémentation. Nous présentons avant tout le projet Présage qui permet de modéliser les réseaux de télécommunications. En effet, le modèle proposé doit s'intégrer comme une nouvelle fonctionnalité pour gérer l'évolution des réseaux de télécommunications par simulation que le concepteur pourra mener. Ce chapitre met en évidence les étapes du développement et les raisons des choix faits. Il met également en évidence les limites et les extensions nécessaires à apporter.

1.3.7. Conclusion : Contributions et Perspectives

Enfin, ce chapitre conclue notre présent manuscrit. Il rappelle les contributions de nos travaux mais met également en évidence les limites et les quelques perspectives d'ouvertures.

Chapitre 1 : CONTEXTE DE TRAVAIL, MOTIVATIONS ET HYPOTHESES DE TRAVAIL

1.1 - Introduction

L'intérêt suscité par l'approche objet depuis une trentaine d'années dans différents domaines (Bases de Données [Kim& 90], Conception Assistée par Ordinateur [Orel& 95], Méthodes de Conception de Systèmes d'Information [Coad 90], Intelligence Artificielle [Haton& 91]) a permis le développement de systèmes et d'applications de plus en plus robustes mais en même temps de plus en plus complexes. Cette expansion a immanquablement posé le problème crucial de l'évolution, de l'adaptation et de la maintenance de ces systèmes et de ces applications. De par les caractéristiques d'abstraction et de modularité, l'approche objet est considérée comme étant celle qui se prête le mieux à la gestion de la complexité, à l'extension et à l'adaptation des systèmes en réutilisant le plus possible les spécifications existantes. Pourtant, les résultats obtenus en pratique montrent que les systèmes modélisés et les applications développées ont une faible flexibilité et des difficultés d'adaptation.

En effet, l'expérience et le recul acquis aujourd'hui dans le développement de systèmes et d'applications objets permettent de cerner certaines de leurs lacunes. Parmi elles, celles qui reviennent à l'écart non comblé entre la réalité et le modèle objet qui en est abstrait :

– *La réalité est souvent empreinte de nuances ; un modèle objet est impératif et strict :* les données de la réalité ne se prêtent pas toujours à une modélisation selon une classification aux structures pré-établies. En raison de la variété des mécanismes proposés par les langages orientés objets, les meilleurs choix à faire pour modéliser la réalité en termes de classes ne sont pas évidents à faire. De plus, la modélisation objet s'appuie lourdement sur le mécanisme d'héritage qui porte plusieurs significations (héritage de code, spécialisation, partage de propriétés et de code, application de contraintes…). Il n'est pas évident de dégager et d'exprimer la sémantique souhaitée par le concepteur. Souvent dans le contexte réel, les représentants d'une même entité

réelle ne partagent pas forcément les mêmes caractéristiques. Ils peuvent varier, dans une certaine mesure, dans leurs propriétés et leurs comportements. Pour distinguer de telles différences au niveau de la modélisation, il n'y a pas d'autre issue que de développer autant de sous-classes que nécessaire. Cette action conduit inévitablement à la construction d'importantes hiérarchies de classes. Cette rigidité devient alors une contrainte et pénalise drastiquement les applications : les classes obtenues ne reflètent pas la réalité du système ainsi abstrait. Cette contrainte se fait particulièrement ressentir dans les nouvelles applications d'ingénierie (CAO, Multimédia, librairies digitales, …) où les objets manipulés présentent la particularité de ne pas être facilement modélisés avec les concepts classiques du paradigme objet.

– *Les besoins peuvent être non identifiés ou mal définis ; les spécifications conceptuelles sont établies de façon fixe :* sauf si elles sont modifiées explicitement pour s'adapter. Les problèmes liés à l'évolution peuvent aussi bien concerner un modèle en phase de conception qu'en phase d'utilisation et de maintenance. Dans le premier cas, le modèle peut souvent être instable et partiellement défini, nécessitant une spécification incrémentale. Dans le second cas, il peut être revu et reformulé durant son exploitation et sa maintenance. Sa stabilité et son entière spécification passent alors par divers processus de validation. Ces processus peuvent être stratégiques (changements de besoins, de stratégie de conception, …) ou purement techniques (tests et vérifications).

Nous verrons dans la section qui suit un cas d'applications concernées par ces problèmes :

1.2 - Contexte du travail : Modélisation des réseaux de télécommunications

Nous présentons dans cette section le contexte à partir duquel le présent travail a été initié. Le contexte est celui d'un système de planification de réseaux de télécommunications. Les besoins sont des besoins d'évolution de ces réseaux. Nous présentons par la suite la problématique d'évolution et nous soulignerons que la problématique ne concerne pas uniquement ce type de système, mais tous les systèmes à objets manipulant de grandes bases d'objets.

1.2.1. Le projet Présage

Le système Présage est un outil d'aide à la planification de réseaux de télécommunications [Caminada 92, Talens 94]. Il s'appuie sur une modélisation objets des réseaux de télécommunications. Un des principaux besoins est la maintenance et l'évolution de ces réseaux. En effet, une fois qu'ils sont modélisés, ils doivent pouvoir évoluer pour prendre en compte les nouvelles technologies, ainsi que les nouveaux

besoins et problèmes qui leur sont associés. Nous pouvons considérer que les besoins d'évolution se départagent en deux catégories :

– Les besoins connus à l'avance et pouvant être intégrés dans le cadre des spécifications conceptuelles. L'évolution est donc prise en compte et peut être menée à bien lorsque le besoin se présente. Les travaux de [Talens 94] et [Urtado 98] ont été développés dans ce cadre et se basent sur des stratégies d'évolution telles que le versionnement ;

– Les nouveaux besoins imprévus, les besoins mal ou faiblement définis. Généralement, le système ne peut pas réagir à ces situations, et il peut connaître une conjoncture de rupture dans son cycle de vie. Il est souhaitable d'éviter ce type de problèmes. Nous nous intéressons à ce type de problèmes de l'évolution.

Dans ce chapitre, nous présentons les caractéristiques des réseaux de télécommunications ainsi que leur modélisation dans le cadre de Présage. Par la suite, nous expliquons les besoins que peut rencontrer un concepteur chargé de la maintenance et de l'évolution dans le cadre du fonctionnement des modèles de réseaux de télécommunications. L'énumération de ces besoins nous amènera alors à une problématique plus large, aux motivations et aux objectifs de ce travail ainsi qu'à préciser également nos hypothèses de travail.

1.2.2. L'évolution dans les réseaux de télécommunications

Le domaine des Télécommunications [Peyrade 87] se caractérise par la complexité et l'évolution rapide des besoins en communication et des technologies utilisées. La planification des réseaux de télécommunications consiste à calculer l'évolution d'un réseau initial vers un état final. Cette évolution doit satisfaire un certain nombre de contraintes dues d'une part, à l'état initial du réseau et d'autre part, aux nouvelles exigences de transmission : contraintes techniques dues aux équipements utilisés, règles d'ingénierie, contraintes économiques reflétant la politique générale suivie quant à l'évolution globale du réseau… Les calculs sont effectués sous la forme de séries d'exécution de programmes de planification qui font évoluer la structure du réseau et le flot de données que le réseau supporte [Talens 94].

Les difficultés de modélisation et de gestion de l'évolution des réseaux de télécommunications viennent de :

– leur *complexité* : un réseau est composé de nœuds et d'arêtes et un nœud et/ou une arête peut se décomposer en plusieurs nœuds et arêtes ;

– la *grande quantité* et la *grande diversité* d'informations qu'ils contiennent et qu'il faut gérer ;

– la *diversité* des types de réseaux à traiter ;

– l'*évolution rapide de la technologie* de télécommunications : les réseaux évoluent, il faut donc pouvoir prendre en compte cette évolution. Les nouveaux réseaux sont utilisés pour des études futures.

1.2.3. Modélisation des réseaux de télécommunications et illustration par un exemple

Dans le cadre des projets initiateurs de Présage [Caminada 92] et au vu des besoins et objectifs respectifs, les réseaux de télécommunications sont modélisés dans Présage sur les bases suivantes :

– Une classe représente un modèle d'applications de réseaux. Elle décrit principalement les différents types d'éléments utilisés pour modéliser un type donné de réseaux (urbain, longue distance…). Elle comporte notamment les différents types de nœuds et d'arêtes. Une classe décrit par exemple les différents types d'éléments utilisés dans un réseau urbain ;

– Une instance d'une classe est un graphe représentant un réseau de télécommunications. Ce graphe est créé en utilisant les différents types d'éléments décrits dans la classe. Une instance représente par exemple la structure d'un réseau urbain donné aussi bien dans sa configuration spatiale que dans sa composante matérielle.

Considérons les classes décrivant des caractéristiques de réseaux urbains :

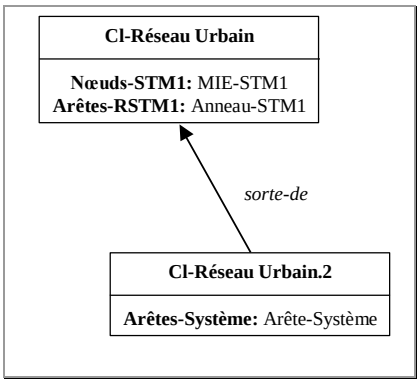


Figure 1-1: Classes Présage décrivant des caractéristiques de réseaux de télécommunications urbains.

La première classe, **Cl-Réseau Urbain**, permet de définir des graphes de réseaux constitués de nœuds de type **MIE-STM1** reliés entre eux avec des arêtes de type **Anneau-STM1**. La seconde classe, **Cl-Réseau Urbain.2**, qui spécialise la première, définit des arêtes de type **Arête-Système**, en plus de ceux définis dans sa super-classe.

La Figure 1-2 illustre une instance de la classe **Cl-Réseau Urbain.2** et donne la configuration suivante d'un réseau de télécommunications urbain :

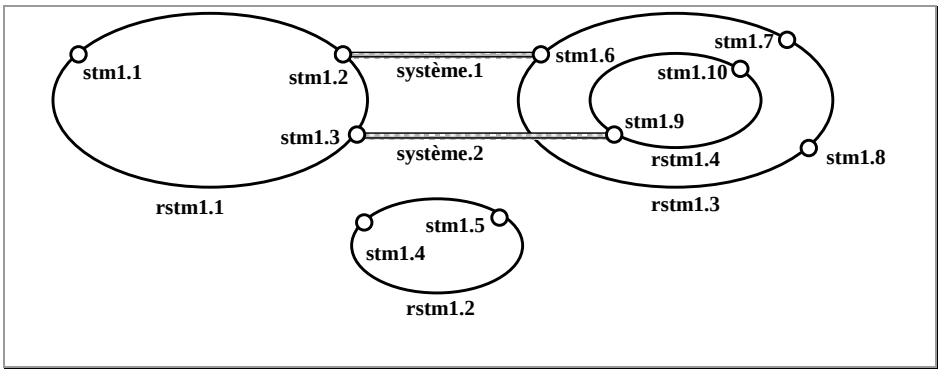


Figure 1-2 : une instance de la classe Cl-Réseau Urbain.2. de Présage et représentant un réseau urbain.

L'instance représente un réseau de télécommunications qui a été créé en utilisant les éléments contenus dans la classe **Cl-Réseau Urbain.2**. Le réseau urbain, représenté par cette instance, est donc constitué de composants dont le type est décrit dans la classe **Cl-Réseau Urbain.2**. Cette instance représente les voies de communications d'un réseau urbain. Avec cette représentation, le concepteur réalise rapidement qu'il obtient une saturation au niveau de la demande pour les deux anneaux de type '**STM1**' (à droite de la Figure 1-2). Il faut donc remédier à ce problème.

La problématique d'évolution de ces réseaux dans leur modélisation est abordée dans la section suivante :

1.3 - Problématique

Nous abordons dans cette section un aspect particulier des besoins d'évolution auxquels un concepteur d'applications de réseaux de télécommunications est confronté, à savoir les besoins d'évolution inconnus, mal ou pauvrement spécifiés. Nous illustrons ce type de besoins par un exemple avant d'exprimer cette même problématique en termes de modélisation objet.

1.3.1. Problématique du concepteur

La problématique initiatrice du travail se résume comme suit. Le concepteur a la charge de la gestion, la manipulation et la planification des réseaux modélisés dans la

base. Plusieurs données lui parviennent ayant trait plus ou moins directement aux différents types de réseaux. Elles peuvent porter sur des contraintes techniques, des règles d'ingénierie, des contraintes économiques, des études de planification…. L'analyse de ces données lui révèle, par exemple, que certains réseaux urbains risquent d'arriver à saturation. C'est une situation courante. Dans le cadre de la planification, il est crucial d'anticiper ce genre de problèmes. Les réseaux peuvent nécessiter d'être modifiés dans leur structure et/ou dans les types de composants (nœuds et/ou arêtes) pour être plus adaptés à la conjoncture.

La plupart des besoins d'évolution sont soit nécessaire pour faire face à un problème qui se présente et qu'il faut de suite prendre en charge, soit pour anticiper justement les problèmes de saturation, les problèmes d'économie ou de sécurisation de réseaux existants. Cette anticipation nécessite un travail de planification, dans le cadre duquel le concepteur souhaite pouvoir simuler diverses évolutions possibles de réseaux, pouvoir comparer les différents résultats et pouvoir choisir la plus adéquate au vu des contraintes et des acquis.

En résumé, dans le cadre de l'évolution des réseaux de télécommunications, le concepteur désire pourvoir manipuler la structure des réseaux, c'est-à-dire :

- ajouter, enlever ou modifier des nœuds et/ou arêtes et des types de nœuds et/ou arêtes, tout en respectant leurs contraintes spécifiques ;
- chercher diverses combinaisons et les évaluer ;
- pouvoir choisir parmi les modèles de réseaux dégagés ;
- intégrer les plus adaptés dans la base existante.

Nous avons utilisé la terminologie propre à la modélisation des réseaux de télécommunications pour décrire la problématique et le contexte réel de travail d'un concepteur d'applications pour situer ses besoins et leur importance. Il s'agit pour nous d'exprimer et de projeter la problématique du concepteur en termes de problématique d'évolution de modèles à objets.

1.3.2. Problématique sous l'angle de vue de la modélisation et de l'évolution d'objets

Pour répondre aux besoins sus-cités, il est nécessaire de porter les modifications nécessaires au niveau de la classe, principalement pour modifier les types de composants. En soit, l'opération est certes lourde et coûteuse, mais elle s'avère nécessaire. Le concepteur explique qu'il prendrait malgré tout le parti de l'appliquer s'il pouvait déterminer sans ambiguïté l'emplacement des changements à apporter au niveau des spécifications existantes. Or, les modifications apportées peuvent s'avérer inadéquates, et qu'il n'est par conséquent pas approprié d'intégrer dans la base. Les modifications auront été non seulement coûteuses en termes de temps, de réorganisation et mises à jour de la base, et de blocage d'accès, mais elles auront été également inutiles. Le besoin d'évolution n'aura finalement pas été résolu et il y aurait eu en sus beaucoup de pertes.

Dans le cadre de la planification de ce type d'applications, il est nécessaire de pouvoir simuler, voire parfois tâtonner, en expérimentant des changements aussi bien sur la dimension spatiale que sur la dimension composante d'un réseau, puis observer et retenir les meilleures structures. Une fois cette sélection faite, les structures retenues peuvent être intégrées de manière durable dans la base en qualité d'un modèle de réseaux nouvellement admis.

1.3.3. Conclusion

Ce type de besoins d'évolution n'est pas le propre des applications de télécommunications, mais peut s'appliquer à tous les domaines de modélisation manipulant des applications orientées-objet complexes et volumineuses et connaissant des besoins importants d'évolution. A titre d'exemple, nous citerons le domaine de la CAO (Conception Assistée par Ordinateur). En effet, la marge de nouvelles conceptions en CAO est de l'ordre de 20% alors que le reste des efforts s'inscrivent dans le cas de reconception et donc d'adaptation par évolution de projets de conception existants, soit l'évolution et l'adaptation de modèles existants [Harani 97]. Les instances en CAO peuvent modéliser des objets en cours d'élaboration : elles sont alors soit incomplètes, temporairement incohérentes, multi-représentées et dépendantes fonctionnellement ou existentiellement d'autres instances [Rieu 99]. D'autres travaux se sont également intéressés à la problématique de l'évolution d'applications volumineuses et complexes dans la composante particulière que sont les relations. Dans [Rashid& 99], les auteurs s'intéressent à l'évolution dynamique de tous les types possibles de relations au sein d'un modèle à objet.

Nous développons dans la section qui suit notre approche d'une telle problématique d'évolution :

1.4 - Approche de la problématique

La problématique du concepteur dans un domaine d'ingénierie, comme nous l'avons vu, n'est pas typique uniquement de la planification des réseaux de télécommunications, mais elle s'étend et se justifie pour toutes les applications orientées-objet complexes et volumineuses.

1.4.1. Nos motivations : faire face à des situations imprévues de changements

Un système orienté objet dit évolutif doit garantir la cohérence, la consistance et la pertinence des informations modélisées. Le processus d'évolution doit être harmonieux et doit éviter au maximum l'introduction d'incohérences et une refonte inutile dudit système. En effet, les problèmes liés à l'évolution se posent tout au long du cycle de vie des modèles et des applications. Ils peuvent aussi bien concerner un modèle en phase de conception qu'en phase d'utilisation et de maintenance. Pour gérer l'évolution d'applications d'ingénierie, un concepteur dispose d'un grand nombre de stratégies. Elles diffèrent dans la manière de gérer l'évolution suivant les buts et les besoins mais elles ont pour point commun de s'appliquer sur les schémas de classes et les classes. Un concepteur peut utiliser, selon ses besoins, une ou plusieurs de ces stratégies, à condition d'avoir cerné tous les besoins durant la phase d'analyse et de les avoir intégrés dans la phase de conception.

Tant qu'une application rencontre des besoins prévus, son cycle de vie ne connaît pas ou peu de perturbations importantes. Par contre, lorsque de nouveaux besoins apparaissent, qu'ils soient mal spécifiés ou nouvellement identifiés, le concepteur doit les intégrer dans les spécifications de classes. Dans le cadre d'applications d'ingénierie présentant d'importants graphes de classes avec des bases volumineuses d'objets, le problème de l'évolution, de l'adaptation et de la maintenance se pose de manière cruciale. Les besoins d'évolution sont souvent imprévus, mal définis ou mal anticipés et leur intégration s'avère être une tâche particulièrement délicate. Étant donné le volume important de données, le concepteur n'arrive pas toujours à avoir une vision à la fois globale et détaillée de l'application. Il peut alors rencontrer de grandes difficultés pour détecter à quel endroit du graphe de classes et sur quelle partie des spécifications (bout de schéma, classe(s), lien(s)…) il doit apporter les modifications. Il aura de plus grandes difficultés à anticiper et à évaluer les impacts, peut être préjudiciables, de ces modifications. Son unique certitude réside dans les changements nécessaires à apporter à son application pour répondre à un problème donné. Ces changements sont en général mieux perçus au niveau de l'application, donc des instances. Malheureusement, la dualité entre les concepts de classe et d'instance ne le permet pas. Les spécifications conceptuelles prédominent et dirigent, souvent pleinement, l'évolution. Ceci explique que la plupart des stratégies d'évolution ont été développées pour gérer l'évolution de classes et de schémas de classes ainsi que leurs impacts sur les instances mais uniquement comme conséquence.

A notre connaissance, peu de stratégies (à titre d'exemple, les travaux décrits dans [Li& 94] au chapitre 2) permettent de gérer l'évolution d'instances avec des impacts sur les classes. Aussi, lorsqu'un concepteur doit faire évoluer une base importante d'objets, il dispose uniquement d'outils de la première catégorie. Pourtant, l'expérience montre [Li& 98], [Price 97] que le plus souvent, les nouveaux besoins ou les besoins mal définis apparaissent durant la manipulation d'applications, donc des instances.

Notre approche de la problématique se base sur la manière dont nous proposons de considérer les instances :

1.4.2. Reconsidérons les instances

L'évolution d'une instance passe généralement par celle de ses classes génératrices. La gestion de l'évolution des classes a pris le pas sur celle des instances qui leur restent conformes, au point où cette règle est ressentie, dans certaines situations, comme étant une contrainte. A la base, le concept de *classe* a été introduit pour spécifier une structure et un comportement communs à un ensemble d'objets (appelés instances de la classe) qui modélisent des entités réelles. Or, ces dernières ont une grande capacité d'évolution, ce qui n'est pas le cas de leurs représentants objets. En effet, certaines applications nécessitent des données nouvelles qui n'ont pas été spécifiées au niveau des classes et qui gagneraient à être prises en compte directement au niveau des instances.

Comment ? En considérant les instances, en tant que représentantes d'entités réelles, comme des *individus* à part entière. Une instance a donc connaissance de ses propres spécificités, et surtout permet l'acquisition de nouvelles connaissances plus adaptées à son nouvel environnement. Elle peut ainsi s'adapter et faire remonter au niveau conceptuel non seulement des données mais également une voie possible d'évolution pour les instances issues de la même classe.

Cette manière de considérer les instances induit de facto l'analogie avec des individus tels qu'ils sont considérés dans le cadre des travaux de l'Évolution Artificielle ([Holland 75], [Heudin 94], [Goldberg 94] …), du moins dans certaines de leurs propriétés : celles d'évoluer dans leur environnement en s'y adaptant. Une adaptation d'un individu ne se cantonnera pas à lui seul mais sera répercutée dans son espèce en modifiant son code génétique. Cet individu fera ainsi profiter de son évolution sa population et sa progéniture.

1.4.3. Notre contribution : un modèle pour la simulation d'évolution

Compte tenu que l'évolution de classes a été largement traitée dans la littérature [Banerjee& 87], [Bratsberg 93], [Casais 91], [Nguyen& 89], [Kim& 88], [Oussalah& 99] …, notre approche de l'évolution se veut complémentaire car notre intérêt se porte essentiellement sur l'évolution d'instances. Elle nous paraît en effet cruciale dans le développement et l'exploitation d'applications réelles. Le modèle d'évolution que nous proposons doit prendre en compte des besoins non ou mal anticipés en permettant à un objet d'évoluer par lui-même à partir de l'information qu'il détient mais aussi qu'il peut retrouver et déduire, minimisant ainsi l'intervention externe du concepteur (celle-ci devient essentiellement stratégique), en proposant la création de nouvelles structures conceptuelles.

Pour ce faire, nous proposons d'étendre l'évolution d'instances de la simple modification de leurs valeurs à la modification de leur structure (ajout /retrait d'attributs) selon les besoins qu'exprimerait directement le concepteur sur l'instance. Nous étudions également les processus d'évolution d'un modèle à objets de manière à lui permettre de s'adapter par lui-même lorsqu'un changement s'opère.

1.5 - Hypothèses de travail

Nous regroupons dans cette section les principales hypothèses de travail :

– Nous nous intéressons à l'évolution d'un modèle à objets à partir de l'évolution dynamique de la structure d'instances existantes. Notre objectif est de dégager plusieurs structures conceptuelles au vu de ces évolutions d'instances. Pour cela, nous nous intéressons uniquement à l'évolution de structures. Nous utilisons donc le terme de '**structure conceptuelle**' pour désigner toute entité constituée d'un ensemble d'attributs. Nous pouvons ainsi avoir des structures conceptuelles existantes, à savoir les attributs définis dans une classe ; des structures conceptuelles émergentes de l'évolution d'instances ; des structures conceptuelles véhiculées par des instances qui évoluent, et qui peuvent ne plus être conformes aux structures conceptuelles de classes existantes.

Toutefois, lorsque l'expression '**structure conceptuelle**' est trop lourde dans la formulation d'un paragraphe, nous parlerons alors tout simplement de '**structure**'.

– Nous ne traitons pas de l'évolution du comportement des instances. Nous ne nous intéressons donc pas à l'évolution des méthodes, du moins pas de manière directe. Une partie de la gestion des méthodes est une conséquence de l'évolution de structures conceptuelles. Faire évoluer une instance dans sa structure revient à ajouter, enlever ou modifier un attribut. Le seul contrôle que nous pouvons faire sur les méthodes est de vérifier que l'attribut est manipulé par des méthodes. Nous ne pourrons contrôler que les paramètres de la signature. Il en est de même pour les attributs retirés de la structure d'une instance. Le contrôle que nous proposons s'applique uniquement sur la signature des méthodes. Nous ne contrôlons pas le corps des méthodes.

– Il est important de souligner que nous considérons les choix de conception initiaux comme étant corrects. L'évolution se base sur les spécifications existantes, et elle ne cherche pas à vérifier la cohérence et la complétude de la conception établie. Par contre, si des contradictions apparaissent entre les propositions d'évolution et les spécifications existantes, celles-ci sont signalées et peuvent par conséquent être prises en compte pour réviser les choix de conception initiaux.

– Nous utilisons indifféremment les termes '**objet**' et '**instance**' tout au long du présent manuscrit. De plus, lorsque nous parlons de '**propriété**', il s'agit indifféremment d'attribut ou de méthode.

1.6 - Conclusion

Dans le cadre de besoins d'évolution d'applications orientées objet complexes et volumineuses, nous nous intéressons à la possibilité de prendre en compte et d'intégrer au mieux les besoins imprévus ou mal définis lorsqu'ils apparaissent tout au long du cycle de vie. L'ensemble des besoins change en raison de la nature instable des nécessités des applications pour répondre aux attentes.

Cycle de vie : Besoins -> Conception -> Implémentation

Nous considérons que les besoins de changements sont de deux catégories :

1. Les besoins d'évolution qui sont perçus lors de la phase d'analyse et sont pris en compte lors de la phase de conception ;
2. Les besoins d'évolution imprévus, faiblement ou mal spécifiés qui s'imposent durant le cycle de vie des applications.

Il est difficile d'évaluer la part d'importance de chacun des deux types de besoins, mais le second cas de figure est loin de représenter des situations exceptionnelles ; bien au contraire. Les deux catégories de besoins peuvent concerner tous les niveaux des applications :

- *Conceptuel :* schéma de classes (mettant en jeu les différents liens d'héritage, d'associations, de composition…) et classes (les attributs dans leur définition, leur type et les méthodes dans leur nom, leur signature et leur corps).
- *Implémentation :* les instances (création, attribution de valeurs) et les programmes.

Pour gérer les besoins d'évolution, une multitude de stratégies ont été définies et développées ([Banerjee& 87], [Bancilhon& 88], [Bertino 92], [Borgida 88], [Bratsberg 93], [Casais 91], [Clamen 94], [Kim& 90], [Nguyen& 89], [Zdonik 90] …). Certaines ont été particulièrement utilisées et le sont toujours. Elles ont également inspiré bon nombre d'autres stratégies. Les stratégies d'évolution s'accordent sur certains points, dont bien sûr celui de gérer l'évolution, et diffèrent sur bon nombre d'autres points, et qui concernent :

- Le domaine de recherche et d'application (Intelligence Artificielle, Génie Logiciel, BDOO…) ;
- La politique d'évolution adoptée (garder l'historique, dériver les propriétés,…) ;
- Les niveaux sur lesquels s'appliquent les changements (classes – liens – instances – programmes) ;

- Les concepts manipulés ainsi que les concepts et processus proposés pour supporter l'évolution.

Nous remarquons que les aspects traités ainsi que la façon dont ils sont traités rendent toutes les stratégies difficiles à appréhender d'une façon identique, ce qui rend leur comparaison difficile.

Nous abordons dans le chapitre suivant l'état de l'art sur l'évolution d'objets.

Chapitre 2 : ÉTAT DE L'ART

2.1 - Introduction

Nous abordons l'évolution orientée objet dans le présent chapitre en deux principales parties :

1- l'évolution d'objets décrite dans ses caractéristiques et les stratégies d'évolution vues selon ces mêmes caractéristiques ;

2- le positionnement de notre approche de la problématique de l'évolution aussi bien par rapport à l'évolution d'objets qu'à l'évolution artificielle.

La première partie a pour but de positionner, de définir et de décrire l'évolution d'objets dans ses caractéristiques que nous dénommons *facettes*. Ensuite, nous montrons comment situer toute stratégie d'évolution par rapport à ces facettes. Selon la facette abordée, les stratégies se positionnant sur celle-ci, peuvent être comparées entre elles. Nous considérons que cette manière d'aborder l'évolution d'objets a pour principal avantage de mettre en évidence tous les aspects de l'évolution, et de positionner une stratégie par rapport à un référentiel commun et indépendant, et non par rapport à une autre stratégie.

Ce chapitre se termine en abordant toujours l'évolution mais cette fois dans le cadre des travaux de l'Évolution Artificielle, car nous nous sommes inspirés de certains principes et de quelques concepts de ces travaux pour proposer l'évolution d'objets à partir de l'évolution dynamique de structures d'instances.

2.2 - Les trois facettes de l'Évolution d'Objets

2.2.1. Introduction

La problématique de l'évolution dans les systèmes à objets concerne plusieurs domaines d'activités : la Conception Assistée par Ordinateur, le Génie Logiciel, l'Intelligence Artificielle, les Systèmes d'Information, les Interfaces Homme-Machine, les Bases de Données ainsi que les domaines de l'Ingénierie. De cette diversité de domaines, divers travaux ont été issus du monde de la recherche et de l'industrie. Ils définissent des systèmes et des langages de programmation à objets proposant des stratégies et des mécanismes pour gérer l'évolution. L'expérience a montré, aussi bien au niveau de la conception que celui du développement, les limites des approches évolutives actuelles. Les stratégies d'évolution existantes sont diverses et variées et ne répondent que partiellement aux besoins.

De façon générale, pour préparer un système ou une application à l'évolution, il est nécessaire de pouvoir :

a) Formuler les changements afin d'atteindre l'objectif visé (le modèle d'arrivée, à savoir le modèle ayant évolué).

b) Gérer l'impact engendré par ces changements.

c) Établir le lien entre le modèle de départ et le modèle d'arrivée.

Plutôt que de classifier quelques stratégies d'évolution en prenant comme repère des critères qui leur sont communs mais qui ne sont pas exhaustifs, nous proposons une classification reposant sur les critères propres à l'évolution. Cette classification présente un double avantage : celui de cerner les aspects traités de ceux non ou peu traités de l'évolution, et celui de pouvoir positionner les principales stratégies d'évolution existantes par rapport au même référentiel.

2.2.2. Les trois facettes de l'évolution d'objets

Avant de pouvoir choisir quelle approche adopter, voire quelle stratégie adopter, pour gérer l'évolution d'un modèle à objets, il est important avant tout de connaître le contexte de travail. Le choix d'une stratégie à adopter doit venir en aval. Les aspects qui nous paraissent importants dans l'évolution sont introduits sous forme d'interrogations :

– Sur quoi porte l'évolution, autrement dit quel est l'objet de l'évolution : l'aspect statique ou l'aspect dynamique ?

– Comment est gérée l'évolution ? Les besoins doivent-ils être impérativement pris en compte lors de la phase d'analyse et spécifiés lors de la phase de conception pour assurer l'évolution ? Ou alors, peuvent-ils survenir ou se préciser tout au long du cycle de vie et l'évolution est malgré tout assurée ?

– A quel(s) niveau(x) de l'application sont effectués les changements et vers quels autres niveaux sont propagés les impacts ?

De ces trois questions fondamentales, il nous apparaît que les trois principaux aspects devant être explicités lorsque l'évolution d'objets est abordée sont :

1- L'*Objet de l'évolution* ;

2- Le *Type de l'évolution* ;

3- Le *Processus d'évolution*.

Nous dénommons ces aspects les *facettes* de l'évolution. Nous les définissons comme suit :

2.2.2.1. *Objet de l'Évolution*

L'évolution peut porter sur un *produit* ou sur un *processus* :

– Nous entendons par *produit* tout élément structurel qui s'intègre directement dans le modèle en cours de conception. Ces éléments sont considérés comme passifs et peuvent être composés, généralement par le concepteur, entre eux grâce à différents liens. Dans l'approche objet, ces éléments peuvent être simples, tels que les attributs, ou complexes, telles que les classes et les instances. Ils peuvent être encore plus complexes tel qu'un schéma de classes dans lequel les classes sont connectées entre elles par des liens d'héritage, d'association ou de composition. Ces éléments sont également nécessaires pour définir des structures plus complexes tels que des domaines d'applications. Pour notre part, nous nous cantonnons aux hiérarchies de classes, les classes, les liens (d'héritage, d'associations, de composition), les instances et les attributs et méthodes.

– Nous entendons par *processus* tout fragment de démarche ou de raisonnement pouvant être exécuté à chaque utilisation. Dans l'approche objet, ces fragments sont généralement décrits sous forme modulaire. Ils peuvent être simples (la construction syntaxique d'un langage de développement), plus élaborés (des sous-programmes, des fonctions, des packages). Ils peuvent être organisés de manière plus complexes, telles que dans des bibliographies de fonctionnalités, et ils peuvent même définir des infrastructures logicielles (frameworks), fournissant ainsi des canevas pour le développement de familles d'applications [Cauvet& 99].

2.2.2.2. *Type de l'Évolution*

Nous considérons qu'il existe deux cas de figures :

– L'*Évolution Préventive* : l'évolution peut être assurée pour les besoins d'évolution pris en compte lors de la phase d'analyse et spécifiés lors de la conception. Il s'agit de besoins qui sont identifiés et cernés dés le début. Dans ce cas, l'évolution est dite *préventive* ou *anticipée*. Au vu des trois conditions citées en §2.2.1 pour mener à bien

l'évolution, il y a des situations ou la condition c) n'est pas vérifiée (soit parce qu'elle est impossible à établir ou parce qu'elle n'est pas nécessaire). Aussi :

- L'évolution préventive est dite *avec rupture* si cette condition c) n'est pas vérifiée, c'est-à-dire que le lien ne peut pas être établi entre le modèle *avant* son évolution et *après* son évolution.
- L'évolution préventive est dite *continue* si au contraire, la même condition c) est vérifiée et assurée, c'est-à-dire que le lien peut être établi entre le modèle *avant* son évolution et *après* son évolution.

– L'*Évolution Curative* : l'évolution peut être assurée pour les besoins nouveaux qui apparaissent de façon imprévue durant le cycle de vie, ou lorsque des besoins ont été mal ou peu spécifiés et que la conjoncture permet enfin de les enrichir ou de les corriger et surtout d'être en mesure de les prendre en compte en les intégrant. Dans ce cas, l'évolution est dite *curative* ou *non-anticipée*.

2.2.2.3. *Processus d'Évolution*

L'évolution de systèmes et d'applications orientés objet est le plus fréquemment menée à partir de celle des spécifications conceptuelles, donc des classes. Aussi, nous distinguons deux principaux cas de figures :

– Le *Développement* : la plupart des stratégies proposent des processus d'évolution menés à partir de l'évolution des classes. Les impacts peuvent être assurés vers d'autres classes existantes ainsi que l'ensemble des instances concernées. Nous dénommons ce type de processus d'évolution, les processus de *développement*. Il s'agit : du développement des classes en intention : leurs spécifications ainsi que celles des classes concernées ; du développement des classes en extension : les instances existantes et concernées.

– L'*Émergence* : peu de processus d'évolution sont assurés à partir de l'évolution d'instances. Nous dénommons ce type de processus, des processus d'*émergence*. Il s'agit d'émergence de nouvelles informations conceptuelles à partir de l'évolution d'instances[3]. Schématiquement, les processus d'évolution sont représentés comme suit, et mettent en évidence les deux niveaux considérés : celui des classes et celui des instances :

[3] Confère Vie Artificielle (section 2.8.2.1) pour situer l'inspiration de cette idée d'émergence.

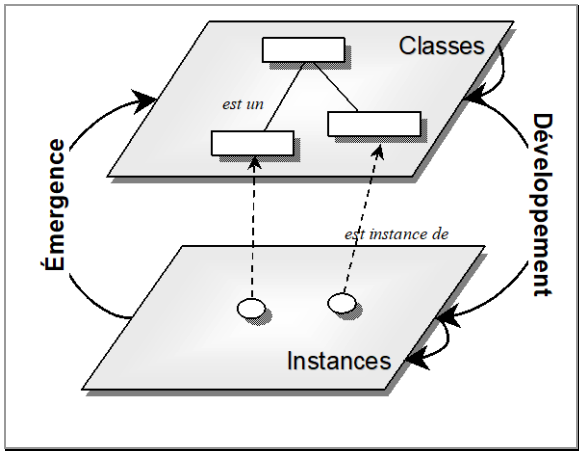


Figure 2-1: le Développement et l'Émergence : les deux processus d'évolution.

Nous mettons explicitement en évidence les impacts intra et inter-niveaux. Le *développement* représente les impacts d'évolution des classes et des instances. Nous rappelons que l'évolution de classes vers d'autres classes et vers des instances, ainsi que l'évolution d'instances dans leurs valeurs ou dans leur rattachement à une autre classe se situent également dans le processus de développement. L'*émergence* représente les impacts d'évolution des instances vers les classes.

Nous abordons dans la section suivante, la représentation de l'évolution d'objets au travers de ses trois facettes : la *représentation triptyque*.

2.2.3. Représentation triptyque

En représentant chacune des trois facettes par un axe. Nous obtenons la caractérisation tridimensionnelle de l'évolution d'objets suivante (Figure 2-2) :

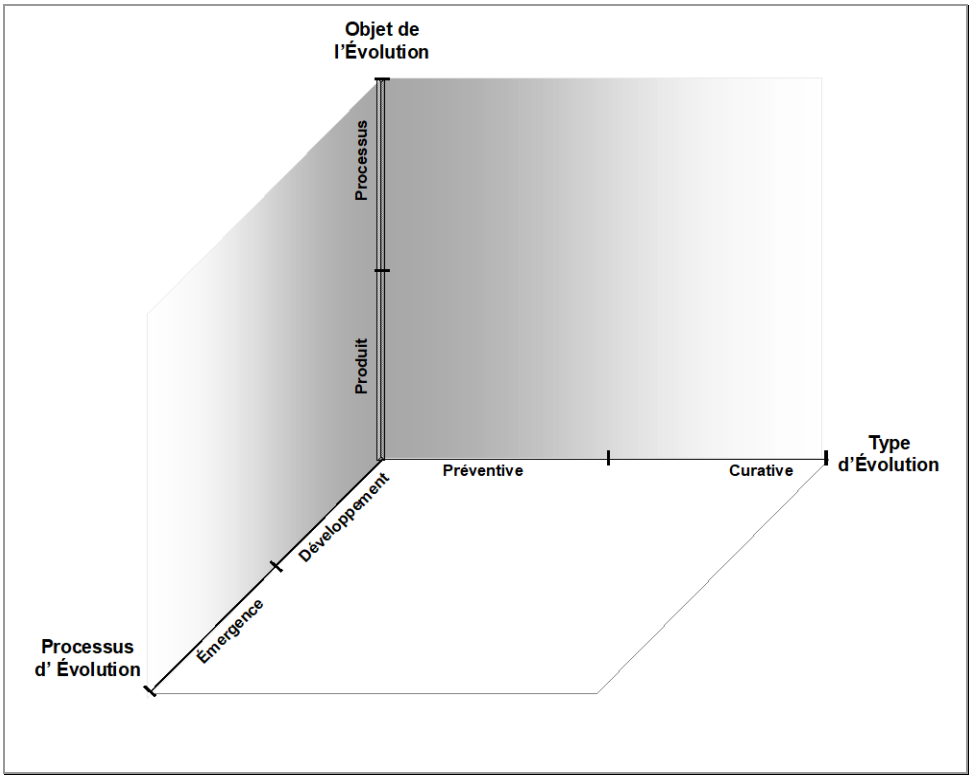


Figure 2-2: Les trois facettes de l'évolution d'objets.

Cette représentation triptyque a pour axe principal *Objet de l'Évolution*, c'est-à-dire sur *quoi* porte l'évolution, quelle est l'entité concernée par l'évolution. Ensuite, se situent le *Processus d'Évolution* ainsi que le *Type d'Évolution* qui sont menés pour assurer l'évolution.

En analysant cette figure, nous mettons également en évidence que :

– La problématique de l'évolution du *Produit* et du *Processus* a été plus largement traitée de façon *préventive* et principalement dans le cadre du développement. En situant la plupart des stratégies d'évolution, elles ont effectivement une politique d'évolution majoritairement *préventive* et principalement dans le cadre du *développement* (confère paragraphe 2.3 - , et plus particulièrement le paragraphe 2.4.7).

– La problématique de l'évolution du *Produit* et du *Processus* a été peu traitée de façon *curative*, et encore moins dans le cadre de l'*émergence*.

En conclusion, les triplets (*produit ; préventive ; développement*) et (*processus ; préventive ; développement*) sont concernés par un bon nombre de stratégies d'évolution. Les aspects *émergence* et *curatif* restent les parents pauvres de l'évolution d'objets.

2.2.4. Comment positionner une stratégie d'évolution ?

Pour classer les stratégies existantes, il suffit de répondre pour chacune d'entre elles aux trois questions suivantes :

1. *Quel est l'objet de l'évolution (produit ou processus) ?*
2. *Quel est le type d'évolution qu'elle propose (curative ou préventive) ?*
3. *Quel type de processus permet-elle (développement ou émergence) ?*

C'est en répondant à ces questions, que nous avons remarqué que les stratégies étudiées œuvrent exclusivement sur le *produit*. La plupart d'entre elles font de l'évolution *préventive* et s'inscrivent dans des processus de *développement* (paragraphe 2.4.7).

La problématique de l'évolution de produit est abordée dans la section suivante :

2.3 - Problématique de l'Évolution du produit

Dans cette section, nous nous focalisons principalement sur une partie de la représentation triptyque de l'évolution. Nous nous intéressons en effet principalement au produit (confère la définition en section 2.2.2.1), car notre approche s'intéresse également sur l'évolution du produit. Nous abordons la problématique au travers de la problématique de l'évolution des classes, des impacts de l'évolution des classes vers les instances et de l'évolution des instances. Dans la présente section, nous nous intéressons uniquement aux stratégies d'évolution par modification de produit, plus précisément de classes, car notre approche préconise également des modifications directes sur le produit invoqué. Les autres stratégies d'évolution ainsi que celles qui gèrent les impacts sur les classes et instances et l'évolution d'instances sont présentées plus sommairement dans la section 2.6.1.

Les différents concepts du produit connaissent une certaine problématique d'évolution et représentent le point d'intérêt de bon nombres de stratégies d'évolution :

2.3.1. Problématique dans l'Évolution du Produit

Les opérations d'ajout, de suppression ou de modification de données ou de fonctionnalités d'un système ou d'une application orienté objet conduisent systématiquement à leur évolution. Ainsi, les changements dans la hiérarchie et la définition de classes doivent être propagés aux instances de la classe modifiée et des sous-classes concernées [Nguyen& 89]. Il faut alors gérer les divers impacts tels que : la propagation correcte des changements vers les sous-classes, les problèmes de conflits de noms, la réorganisation des liens d'héritage, etc. [Keen 89], [Kim& 88], [Skarra& 87], [Zdonik 90].

Plusieurs stratégies d'évolution ont été décrites dans la littérature de l'Évolution d'objets, mais chacune ne répond que partiellement aux besoins sus-cités. Nous avons

étudié les stratégies d'évolution les plus importantes portant sur l'évolution de produit. Nous les présentons en trois catégories, selon le niveau auquel elles s'intéressent :

- L'évolution de classes : l'évolution peut être par modification, par versionnement ou par dérivation de classes et/ou de schémas de classes ;
- L'impact de l'évolution de classes sur les instances ;
- L'évolution d'instances.

2.3.1.1. Problématique de l'évolution de classes

Une classe spécifie la structure et le comportement de ses instances. Elle peut évoluer tant dans la spécification de la structure que celle du comportement. L'expérience montre que les besoins des utilisateurs sont rarement stables et que les concepteurs trouvent des difficultés à classer des composants logiciels dans des taxonomies prédéfinies. En effet, la représentation de la réalité à l'aide de classes et de liens (héritage, composition, ...) n'est pas toujours triviale : l'information du monde réel présente souvent des nuances et des incomplétudes.

D'autres problèmes sont liés à la difficulté de construire des classes stables et réutilisables, et l'extension d'une hiérarchie par des sous-classes peut entraver la conception et le développement de nouveaux composants. Le premier point s'explique par le fait que des classes stables et réutilisables sont généralement obtenues suite à un processus itératif de tests et de corrections. Le second point s'explique par le fait qu'une hiérarchie constitue un noyau standard de fonctionnalités qui peut être étendu avec des sous-classes additionnelles. Ceci peut gêner la conception et le développement de nouveaux composants et le partage des classes par plusieurs utilisateurs.

2.3.1.2. Problématique de l'évolution d'instances

Dans une approche objet classique, une instance doit toujours être conforme à la spécification de sa (ses) classe(s). Elle peut évoluer selon deux cas de figures : suite à l'évolution de sa (ses) classe(s) ou à sa propre évolution :

- dans le premier cas, toute modification dans la spécification d'une classe peut entraîner l'évolution non seulement d'autres classes, mais aussi de ses instances. Cependant, les besoins sont parfois plus fins et plus nuancés : l'utilisateur peut vouloir modifier une classe sans que toutes les instances soient remises en cause ;
- dans le second cas, tout changement dans les informations portées par des instances peut provoquer leur évolution directe. Faire évoluer une instance revient à modifier ses valeurs tout en restant cohérente avec la définition de sa classe génératrice. L'instance doit obéir à la politique d'évolution dictée par sa classe.

2.3.2. Problématique de notre approche

Nous rappelons (chapitre 1) que nous nous intéressons à l'évolution d'objets à partir de l'évolution dynamique de la structure d'instances. C'est le moyen que nous proposons à un concepteur d'applications d'objets complexes et volumineuses pour faire face à des situations de changements imprévus ou mal définis. Pour résumer et situer notre approche par rapport à l'évolution d'objets, nous répondons aux trois questions présentées en section 2.2.4 :

1. *Quel est l'objet de l'évolution auquel nous nous intéressons ?*

   - le **produit**

2. *Quel est le principal type d'évolution que nous proposons ?*

   - l'évolution **curative**

3. *Quel est le principal type de processus que nous proposons ?*

   - l'**émergence**

Nos travaux se veulent complémentaires des stratégies préventives. Nous travaillons donc fondamentalement sur le *produit* en considérant essentiellement l'*émergence* et en s'inscrivant dans le cadre de l'*évolution curative*.

Nous considérons donc le produit comme étant aussi bien la classe, que la hiérarchie de classes (basée sur le lien d'héritage) ainsi que les instances aussi bien dans leur structure que dans leurs valeurs.

2.3.3. Stratégies d'Évolution étudiées

Nous présentons dans la section suivante uniquement les principales stratégies d'évolution de produit par modification de classes que nous avons étudiées (les autres sont présentées en section 2.6.1). Chaque stratégie est présentée par son principe général, ses avantages et inconvénients ainsi que le fonctionnement de la politique d'évolution.

Cette section met en évidence leur similarité entre elles et avec notre approche. En effet, toutes ces stratégies modifient le produit, plus précisément les classes et leur hiérarchie.

Ces stratégies ont :

- une politique d'*évolution préventive*, c'est-à-dire en prenant en compte les besoins connus ;
- et ce, dans le cadre du *développement*, mis à part la catégorisation qui fait de l'*émergence* et provoque du *développement*, c'est-à-dire en provoquant des impacts sur les autres classes et les instances concernées.

Dony, Huchard et Libourel [Dony& 97] rappellent à juste titre dans [Oussalah& 99] que la conception et l'évolution de hiérarchies sont des tâches complexes en soulignant entre autres que :

- La taille de certaines hiérarchies les fait échapper à un contrôle critique de leur(s) concepteur(s) ;
- L'évolution des hiérarchies (ajout ou retrait de classes, de propriétés,…) nécessite des modifications qui doivent rester sous le contrôle des concepteurs.

D'où l'intérêt des stratégies qui proposent une automatisation, même partielle, des procédés de construction et de maîtrise de l'évolution de hiérarchies de classes. Les travaux intègrent une dimension algorithmique :

- Ils s'appuient sur des algorithmes de traitement de hiérarchies de classes, ou ils utilisent des primitives de modifications locales.
- Les algorithmes peuvent être globaux : ils construisent la hiérarchie à partir des classes mises à plat.
- Ils peuvent être incrémentaux : ils partent alors d'une hiérarchie pré-existante et d'une classe, et ne modifient que ce qu'il faut de la hiérarchie pour y intégrer convenablement la classe. Sans oublier l'importance de l'intervention du concepteur au cours du processus et dans quelle mesure il peut intervenir.

Aussi, dans le cadre du fonctionnement de l'évolution des stratégies, nous détaillons pour chacune d'entre elles sur quelle partie du produit elle agit et quelles sont les actions menées pour assurer le processus d'évolution et le type d'évolution, toujours guidés par les objectifs et la politique de la stratégie. Nous ne reprenons pas le détail du déroulement de chacune des stratégies, mais uniquement les actions qu'elles effectuent dans le cadre de leur application. Dans un premier temps, nous énumérons les actions menées, et dans un second temps, nous exprimons ces actions en opérations connues de modification d'une hiérarchie de classes, ainsi que la sémantique qui leur est associée.

Ensuite, au vu de cette présentation, nous proposons une taxonomie du produit ainsi que des opérations d'évolution par modification que nous départageons en opérations de base et opérations avancées :

2.4 - Principales Stratégies d'Évolution par modification de produit

Plusieurs stratégies d'évolution ont été décrites dans la littérature de l'Évolution d'objets, mais chacune ne répond que partiellement aux besoins sus-cités. Nous étudions de manière plus détaillée les stratégies d'évolution qui ont pour politique d'agir directement sur la hiérarchie et la structure des classes, ce qui est également le cas de notre approche. Voici, en substance, leur principe, leur avantages et leurs limites ainsi que leur fonctionnement :

Les trois premières stratégies sont considérées comme étant d'*approche algorithmique* : elles ont pour but d'insérer une classe dans une hiérarchie de classes existante.

2.4.1. Réorganisation de classes

2.4.1.1. Principe, avantages et inconvénients

Principe : la *réorganisation de classes* [Casais 91] s'appuie sur un algorithme pour insérer une nouvelle classe en restructurant la hiérarchie de manière à éliminer les redondances d'information et à factoriser les propriétés communes.

Avantages : l'intérêt de cette stratégie, ainsi que des deux suivantes, est la factorisation de propriétés communes, évitant ainsi l'existence de propriétés redondantes et de propriétés indéfinies. Elles assurent également la minimalité du nombre de classes.

Inconvénients : cependant, la classification peut déranger l'ensemble du schéma de classes, et durant l'exploitation, les conséquences peuvent s'avérer coûteuses. La réorganisation peut insérer des classes uniquement au niveau des feuilles existantes dans la hiérarchie. La catégorisation peut initier des classes mais ne peut pas assurer leur cohérence sémantique.

2.4.1.2. Fonctionnement de l'évolution

Casais propose [Casais 91] un algorithme qui analyse les redéfinitions qui sont apportées lorsqu'une classe est ajoutée dans la hiérarchie, et restructure la hiérarchie pour découvrir des classes éventuellement manquantes. Il estime que les concepts de la réalité doivent être proprement encapsulés en tant que classes, de façon à être facilement spécialisées ou combinées. Une structure de la hiérarchie inadéquate, des classes manquantes dans la hiérarchie, une déficience dans la modélisation peuvent sérieusement affecter la réutilisation. La collection de classes doit alors évoluer de façon à éliminer de tels défauts et améliorer sa robustesse et sa réutilisabilité.

Il distingue deux types de réorganisation : la *décomposition* et la *restructuration* :

– La *décomposition* consiste à éclater diverses classes entremêlées en un lien d'héritage simple. Le but de cette opération est de réduire la surcharge sémantique du lien d'héritage et de détecter de possibles modélisations alternatives.

– La *restructuration* consiste à extraire les propriétés partagées par plusieurs classes et les isoler ensemble au sein d'un nouveau et commun ancêtre. Les buts sont de factoriser les propriétés communes.

Les opérations principalement utilisées dans l'algorithme proposé sont :

Opération	Produit concerné	Sémantique
Création	Classe	Création d'une classe
Transfert	attribut d'une classe vers une autre	Copie dans la classe cible avec/sans suppression dans la classe source
Scission	classes en deux (attributs acceptés et attributs restants)	Manipulation de groupes d'attributs
Vérification	de références non résolues	
Re-direction	d'un lien d'héritage	Changement des source et cible du lien (peut se faire par création/suppression de liens)
Suppression	de classes redondantes dans le graphe	Suppression d'attributs/méthodes ainsi que des liens impliqués.

Tableau 2-1: actions menées dans le déroulement de l'algorithme incrémental de la réorganisation.

Ces opérations peuvent se décomposer en des ensembles d'opérations plus simples. Elles se résument comme suit :

Opération	Produit concerné	Sémantique
Attachement	d'une classe	ajout d'un lien d'héritage
Héritage	de méthode/attribut	ajout (copie) de méthode/attribut
Création	d'une classe	création lien d'héritage, attributs/méthodes
Mise en facteur	d'attributs	création d'une classe
Attachement	de classes	ajout/suppression d'arcs
Déconnexion	de classes	ajout/suppression d'arcs
Fusionne	des classes	ajout et suppression d'arcs (+propagation)
Suppression	de classes	suppression d'arcs (+propagation)

Tableau 2-2: taxonomie des opérations et leur sémantique pour la réorganisation de classes.

Toutes les opérations n'ont pas la même granularité et impliquent des impacts et des propagations plus ou moins limités.

2.4.2. Classification de classes

2.4.2.1. Principe, avantages et inconvénients

Principe : les mécanismes de *classification de classes* [Dekker 94], [Capponi 94] consistent à organiser et à maintenir une hiérarchie de classes, en insérant de façon appropriée de nouvelles classes.

Avantages : l'intérêt de la factorisation de propriétés communes, évitant ainsi l'existence de propriétés redondantes et de propriétés indéfinies. Elles assurent également la minimalité du nombre de classes.

Inconvénients : cependant, la classification peut déranger l'ensemble du schéma de classes, et durant l'exploitation, les conséquences peuvent s'avérer coûteuses.

2.4.2.2. Fonctionnement de l'évolution

La classification de classes consiste à déterminer la position d'une classe dans la hiérarchie de classes compte-tenu de sa description en termes d'attributs [Capponi 94]. Étant donné que l'algorithme repose essentiellement sur une succession de phases de comparaison et de positionnement des attributs de la classe à classifier et des attributs existants, nous nous intéressons plus aux phases d'insertion et des impacts conséquents qu'au principe détaillé de l'algorithme. Ainsi, l'insertion d'une classe dans

une hiérarchie de classes correspond à l'insertion de chaque attribut de cette classe dans le graphe des attributs. La phase finale de l'algorithme revient à faire un choix final de l'emplacement de la classe. Son insertion provoque la modification de la hiérarchie de classes. Les opérations reviennent à toutes les opérations de manipulation d'une hiérarchie de classes telles que définies de façon détaillée par [Banerjee& 1987].

2.4.3. Catégorisation

2.4.3.1. Principe, avantages et inconvénients

Principe : La *catégorisation* [Napoli 92] quant à elle regroupe des instances ayant des caractéristiques et un comportement similaires au sein d'une classe, qui doit être ensuite insérée dans la hiérarchie.

Avantages : outre l'intérêt de la factorisation de propriétés communes et la minimalité du nombre de classes, la catégorisation peut initier de nouvelles classes mathématiquement correctement définies.

Inconvénients : la catégorisation peut initier des classes mais ne peut pas assurer leur cohérence sémantique.

2.4.3.2. Fonctionnement de l'évolution

La catégorisation revient à la construction de spécifications d'une classe à partir d'un ensemble d'instances potentielles. Une fois la spécification trouvée, son insertion nécessite le recours aux même types d'actions que celles utilisées par la classification.

2.4.4. Correction de classes ou Chirurgie de classes :

2.4.4.1. Principe, avantages et inconvénients

Principe : elle est essentiellement utilisée dans les bases de données orientées objet [Banerjee& 87], [Penney& 87]. Le principe est de répercuter les opérations de mises à jour demandées sur la classe ou la collection de classes impliquée, en la modifiant voire en réorganisant totalement la hiérarchie. Elle rend la mise à jour des classes effective après tout changement. La consistance du modèle est vérifiée et maintenue en cadrant l'évolution des classes par des contraintes d'intégrité qui doivent être satisfaites après chaque modification. Le but étant de maintenir la consistance d'une hiérarchie de classes, tous les ajustements nécessaires doivent être prévus

en respectant des invariants et en définissant une taxonomie des opérations spécifiques et des primitives de modifications.

Avantages : le principal avantage est la maintenance de la cohérence et la définition de stratégies de répercussions des changements du schéma de classes vers les instances.

Inconvénients : le principal inconvénient de la chirurgie de classes est qu'elle ne donne aucune directive telle que pourquoi et quand des modifications spécifiques doivent être effectuées. Les modifications ont des impacts locaux et ne peuvent pas traiter les changements sur plusieurs classes. La stratégie ne tient pas compte également de l'historique des différentes évolutions qui auront été menées. Elle ne peut donc être utilisée que si l'historique de l'évolution n'est pas souhaité. De plus, la base reste invalide durant les changements, qui peuvent rapidement devenir complexes et onéreux. Les applications qui s'exécutent en utilisant la base de données peuvent également devenir invalides si elles s'appuient sur des caractéristiques qui auront été perdues après la mise à jour [Borgida 88].

2.4.4.2. Fonctionnement de l'évolution

Sous le modèle Orion [Banerjee& 87], un concepteur de schémas spécifie une évolution en termes de taxonomie et le système vérifie l'évolution en déterminant si elle est consistante avec les invariants et ensuite ajuste le schéma et la base de données au vu des règles appropriées. Les auteurs proposent une structure formelle pour l'évolution de schéma, en termes d'un ensemble de propriétés invariantes d'un schéma orienté-objet et de règles pour les préserver.

La taxonomie complète des opérations est établie selon que les changements sont apportés au contenu d'un nœud (1), à un arc (2) ou à un nœud (3). La taxonomie est suffisamment détaillée pour exprimer aussi bien les opérations que le produit concerné ainsi que la sémantique :

Opération	Produit concerné	Sémantique
Changement	Contenu d'un nœud	– Ajout d'une nouvelle variable d'instance à une classe – Retirer une variable d'instance d'une classe – Changement du nom d'une variable d'instance d'une classe – Changement du domaine d'une variable d'instance d'une classe – Changement de l'héritage (parent) d'une variable d'instance (hérite d'une autre variable d'instance avec le même nom)

		– Changement de la valeur par défaut d'une variable d'instance – Manipulation d'une valeur partagée d'une variable d'instance (Ajout d'une valeur partagée, Modification d'une valeur partagée, Retrait d'une valeur partagée) – Retrait de la propriété lien de composition pour une variable d'instance – Changement à une méthode
Changement	Arc	– Rendre une classe S comme une super-classe d'une classe C – Retirer une classe S de la liste des super-classes de la classe C – Changer l'ordre des super-classes d'une classe C
Changement	Nœud	– Ajouter une nouvelle classe – Retirer une classe existante – Changer le nom d'une classe

Tableau 2-3: taxonomie détaillée des opérations de modifications

Il est à noter, comme cela est rappelé à juste titre dans [Penney 87], que les problèmes qu'ils traitent vont plus loin que la modification de classes, pour également concerner le versionnement de classes, d'objets et de méthodes. En effet, leur méthodologie implémente la modification en utilisant des versions. De plus, ils travaillent sur des graphes de classes où l'héritage multiple est permis. Dans le cadre de stratégies modifiant la structure de la hiérarchie de classes et dans le cadre de l'héritage simple, cette taxonomie se réduit. Elle est présentée de manière plus appropriée dans [Zicari 1991], qui de plus fait un parallèle entre ces opérations et celles de manipulations de graphes.

GemStone [Penney 87] permet l'ajout d'une classe comme feuille dans l'arbre d'héritage. Les opérations proposées ne sont pas exhaustives. Ils présentent uniquement les opérations qui peuvent apporter des modifications utiles, bien comprises et sont raisonnablement implémentables :

Opération	Produit concerné	Sémantique
Renommer	le nom d'une variable d'instance	Le renommage est propagé dans toutes les sous-classes.
Ajouter	une variable d'instance nommée(référençable)	Si le nom de variable d'instance existe déjà, l'opération est rejetée. Sinon, les modifications sont propagées.
Retirer	une variable d'instance nommée	Toutes les instances de la classe sont réécrites pour retirer la variable d'instance. Ne peut être retirée si elle est héritée d'une super-classe.

Modifier	une contrainte sur une variable d'instance	Une contrainte peut être spécialisée ou généralisée. Si elle n'est ni héritée si spécialisée, la variable d'instance associée est détruite et re-ajoutée avec la nouvelle contrainte.
Retirer	une classe	Une classe ne peut être enlevée si elle a des instances. Il est possible d'enlever toutes les instances d'une classe. Le passage d'une classe vers une autre peut utiliser les opérations sur l'ajout et le retrait de variables d'instance.
Ajouter	une classe	Ne peut être ajoutée qu'en tant que feuille.

Tableau 2-4: taxonomie des opérations de modifications utilisées dans GemStone

Comme conséquence de la sémantique des opérations pré-citées, d'autres opérations sont donc également utilisées, principalement l'ajout, le retrait, et la modification des instances.

2.4.5. Versionnement de classes

2.4.5.1. Principe, avantages et inconvénients

Principe : même si le point fort est le versionnement, il n'empêche que sur une version donnée, les changements se font le plus souvent par modification directe de la hiérarchie de classes. Cette stratégie est largement utilisée dans le domaine des bases de données orientées objet [Kim& 88], [Zdonik 90], [Katz& 87], ainsi qu'en CAO [Talens 93] et en Génie Logiciel [Ahmed-Nacer 94]. Une évolution apportée à une classe provoque la dérivation d'une nouvelle version de cette classe. Le versionnement permet de garder une trace des variations intrinsèques d'une application, d'une hiérarchie de classes ou d'une classe tout au long de ses différents changements. Les mises à jour sont assurées uniquement sur des copies, en créant de nouvelles versions de classes de manière à ce que les applications existantes continuent à se dérouler sur les versions originelles. Les principales actions consistent à structurer les différentes catégories de versions ; de déterminer quelle version est utilisée quand une classe est instanciée ; de décider quelles sont les opérations qui justifient la création de nouvelles versions ; d'utiliser des fonctions de conversions pour garantir une cohérence entre les classes copiées et les classes originelles.

Avantages : le versionnement offre l'avantage de ne pas perdre d'informations tout au long de l'évolution de la spécification d'une classe et de garder ses

différents états. La stratégie peut être considérée comme étant une stratégie complémentaire à la correction de classes. Elle s'avère nécessaire dans le cadre d'environnements multi-utilisateurs. Elle permet la restauration et les mises à jour de BDOO, ce qui constitue un avantage indéniable pour toute application nécessitant la reprise d'anciennes spécifications, telles qu'en CAO. Elle est également un bon support pour le processus de conception. Elle permet d'améliorer les performances de systèmes distribués et constitue un bon support pour garder trace des différentes évolutions effectuées sur le système.

Inconvénients : cependant, c'est une stratégie qui impose une navigation entre deux structures fortement liées : la hiérarchie d'héritage et le graphe de dérivations (des différentes versions). La gestion des différentes versions implique un grand nombre d'informations à gérer. Les mécanismes de propagation de versions peuvent mener à une prolifération de versions inutiles, rendant complexe la gestion et la navigation des graphes d'héritage et de dérivation.

2.4.5.2. Fonctionnement de l'évolution

Nous ne traitons pas le mécanisme de versionnement mais les modifications au sein d'une même version. Nous pouvons donc considérer qu'il s'agit de l'évolution d'une hiérarchie de classes. Cela peut se faire par une réorganisation de classes ou une correction de classes, selon les besoins.

2.4.6. Migration de caractéristiques

2.4.6.1. Principe, avantages et inconvénients

Principe : la migration de caractéristiques est utilisée dans les Systèmes d'Information. Une migration est une opération qui transfert des caractéristiques entre classes, valeurs d'attributs d'instances ou d'instances [Jacobson& 91]. Pour cela, il est proposé de définir un graphe de transfert de migration. Un composant d'une migration est une partie de cette migration, constituée de classes de départ et de classes d'arrivées ainsi que de liens de transfert qui transmettent des caractéristiques entre ces deux ensembles de classes [Jiang& 96].

Avantages : la stratégie est particulièrement intéressante pour la conception de l'évolution de systèmes orientés-objet avec un grand nombre de classes (>200) et peut supporter les migrations dans le cadre de transformations de modèles (par exemple, d'un modèle Entité-Association vers un modèle Orienté Objet).

Inconvénients : cependant, il est difficile de rechercher et par conséquent de trouver les meilleures partitions de migrations.

2.4.6.2. Fonctionnement de l'évolution

Les objectifs sont l'analyse, la conception, et l'exécution des migrations des caractéristiques de classes, de valeurs d'attributs d'instances et d'instances. Le principe étant de faire migrer des caractéristiques de classes (attributs, contraintes, opérations…) vers d'autres classes, ils utilisent des opérations pour gérer l'évolution de hiérarchies de classes [Jiang& 96]. En plus des opérations classiques de modification, ils définissent des opérations de réutilisation, de migration et de réorganisation :

Opération	Produit concerné	Sémantique
Création/ Copie/ Suppression / Modification /	Classes/ Valeurs d'attributs d'instances/ Caractéristiques de classes : attributs, contraintes, opérations/ Instances	Opérations de base de modification de hiérarchies de classes et d'instances.
Réutilisation	Classes/ Valeurs d'attributs d'instances/ Caractéristiques de classes : attributs, contraintes, opérations/ Instances	Opérations qui s'appuient sur les opérations de base.
Transfert	Caractéristiques	Le transfert s'appuie sur la migration. Un lien de transfert est défini entre une classe source et une classe cible. Il est à noter que les liens d'héritage ne sont pas transférés.
Migration	caractéristiques de classes	La migration utilise des liens de transferts, organisés en graphe connexe. Pour un lien de transfert, il y a une source et une cible. Le transfert peut se faire avec ou sans suppression dans la source, avec ou sans modification dans la source. La source et la cible sont de même type : classes – valeurs d'attributs d'instances – instances.
	valeurs d'attributs d'instances	
	Instances	
Réorganisation	hiérarchie de classes	La hiérarchie nécessite d'être réorganisée après une migration.

Tableau 2-5: les opérations pour la migration de caractéristiques de classes

Au vu des différentes opérations menées et des différents éléments manipulés, nous remarquons que des opérations ont la même sémantique et invoquent d'autres opérations plus techniques selon l'élément concerné. Pour cela, nous dégageons les taxonomies suivantes de produit et d'opérations. Nous nous inspirons directement de la taxonomie des concepts proposée dans [Zicari 91] et la taxonomie des opérations de mises à jour de schémas de BDOO proposée par [Banerjee& 87].

2.4.7. Premier bilan : positionnement des stratégies étudiées dans le triptyque

En répondant pour chacune des stratégies d'évolution par modification étudiées, au triplet de questions de la section 2.2.4, nous arrivons à les situer sur les deux axes *Type d'évolution* et *Processus d'évolution*.

Le positionnement des stratégies étudiées donne :

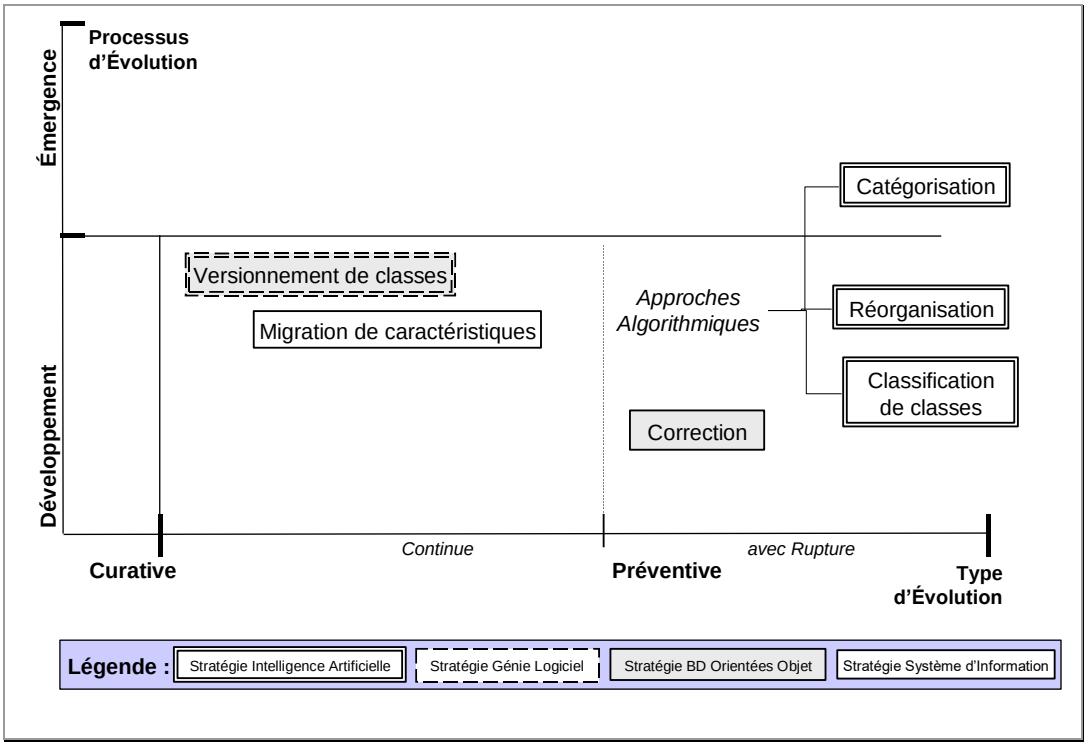


Figure 2-3: première classification triptyque : stratégies d'évolution par modification.

Nous faisons un zoom sur ces stratégies qui agissent sur les mêmes éléments de produit que notre approche, à savoir les stratégies qui agissent directement sur la structure de la hiérarchie de classes, via le lien d'héritage, les spécifications de classes ainsi que le lien d'instanciation.

Nous présentons dans les sections suivantes les principales stratégies d'évolution qui permettent de gérer les impacts d'évolution de classes vers les instances ainsi que celles qui permettent de gérer l'évolution d'instances :

2.4.8. Impacts de l'évolution de classes vers les instances

Les modifications sur des classes impliquent le plus souvent leur propagation au niveau des instances. Ces propagations peuvent être mises en place de trois manières différentes :

2.4.8.1. Conversion d'instances

Principe : les instances sont rendues conformes à la nouvelle définition de leur classe en les réorganisant physiquement. Il existe deux principales stratégies : la *conversion immédiate* [Penney& 87] et la *conversion différée* [Banerjee& 87], [Lerner& 90]. La conversion immédiate transforme les instances automatiquement après l'évolution de leur classe. La conversion différée consiste, quant à elle, à adapter les instances individuellement uniquement lorsqu'elles sont accédées la première fois suite à la modification de leur classe. Ces deux stratégies peuvent être combinées entre elles, tel que le fait le système O_2 [Bancilhon& 88].

Avantages : la conversion immédiate permet de garder la base cohérente. La conversion différée réduit le nombre de conversions et est donc économique en termes d'accès.

Inconvénients : la conversion immédiate rend la base inaccessible durant la conversion. Elle est également coûteuse en termes de performances. La conversion différée conserve les données incohérentes simultanément durant le laps de temps que dure la modification de la classe et celle de ses instances.

2.4.8.2. Émulation d'instances

Principe : l'émulation d'instances est également appelée filtrage, car elle utilise un filtre pour adapter logiquement (et non physiquement) une instance à une nouvelle définition de sa classe. Lors de l'accès à l'instance, celle-ci est interprétée grâce à ce filtre entre la représentation physique de l'instance et la définition de sa classe. Cette technique est souvent utilisée avec la technique de versionnement de classes [Skarra& 87].

Avantages : l'instance est logiquement adaptée et non physiquement, ce qui garantit la cohérence de l'état réel de l'instance et évite de causer des changements peut être inopportuns et coûteux.

Inconvénients : l'émulation peut entraîner une dégradation des performances si le nombre d'incompatibilités entre les instances et les classes augmente, et la définition d'une procédure d'exception est lourde à assurer.

2.4.8.3. Versionnement d'instances

Principe : le principe est d'associer une version d'une instance à chaque version de sa classe [Clamen 94], [Talens& 93]. Ainsi, après la modification d'une classe, dès qu'une instance est accédée, le système en crée automatiquement une version conforme à cette nouvelle version de classe. Des contraintes de dépendance entre les versions d'une classe permettent de garder la cohérence entre elles.

Avantages : cette stratégie assure la cohérence des instances et la compatibilité des applications.

Inconvénients : l'inconvénient est qu'à chaque nouvelle version d'une classe, une version de chacune de ses instances est créée, avec toutes les conséquences quant aux coûts de stockage, de navigation dans les hiérarchies de classes et celles de versions.

2.4.9. Évolution d'instances

L'évolution des instances relève, comme vu précédemment, soit d'une conséquence de l'évolution des classes, soit d'une évolution directe des instances. Dans cette dernière situation, un changement dans la valeur des attributs d'une instance et/ou dans son lien d'appartenance à une classe constituent les deux cas de figures d'évolution d'une instance. Trois principales stratégies existent pour la gérer :

2.4.9.1. Contraintes d'intégrité

Principe : elles permettent de baliser l'évolution de l'instance en définissant des contraintes structurelles, comportementales et applicatives. S'il y a violation, l'évolution est rejetée par le système [Meyer 88], [Bounaas 95].

Avantages : les contraintes d'intégrité permettent ainsi de contrôler l'évolution des instances en vérifiant la cohérence des valeurs modifiées et/ou de réagir face à une violation d'une contrainte.

Inconvénients : elles n'ont qu'un impact local.

2.4.9.2. Versions d'instances

Principe : elles sont utilisées pour garder une trace des divers états que peut prendre une instance. Les versions d'instances permettent également de générer des instances constamment conformes à leur définition.

Avantages : la stratégie est utile dans les environnements de conception.

Inconvénients : mais, elles sont coûteuses et il est difficile de gérer les liens de dépendance entre les versions.

2.4.9.3. Classification d'instances

Principe : elle est utile lorsqu'une instance n'est plus conforme à la spécification de sa classe d'appartenance. La classification d'instances œuvre à rattacher les instances à des classes de la hiérarchie potentiellement candidates, revenant ainsi à un mécanisme de raisonnement au sens de l'Intelligence Artificielle. Dans cet ensemble de classes candidates peut se trouver une classe plus adéquate au nouvel état de l'instance. Dans ce cas, l'instance est propagée dans le graphe de classes, en complétant les valeurs si nécessaire. Elle est utilisée dans des systèmes à base de connaissances [Rechenman 89].

Avantages : la classification d'instances permet à une instance d'évoluer au sein d'une hiérarchie de classes (en termes de rattachement à une classe) et infère les valeurs de façon à la compléter par rapport à sa nouvelle définition (pour être conforme à sa nouvelle classe).

Inconvénients : la classification d'instances s'avère être une stratégie coûteuse lorsqu'elle est utilisée dans les langages et les bases de données orientées objet.

Nous nous intéressons maintenant à la manière dont les stratégies d'évolution de classes par modification (présentées en section 2.4 -) mettent en œuvre l'évolution, autrement dit les opérations qu'elles utilisent. Nous proposons de dégager des taxonomies de produit et d'opérations d'évolution, de manière à mettre en évidence les concepts et les opérations communs aux stratégies d'évolution par modification du produit :

2.5 - Taxonomies de produits et d'opérations

2.5.1. Taxonomie du produit

Nous considérons la structure hiérarchique de classes comme un graphe constitué de nœuds et d'arcs [Zicari 91] :

2.5.1.1. Nœud

Un nœud représente une classe ou une instance (voire une collection de classes pour des stratégies oeuvrant sur des portions ou des schémas de classes). La manipulation d'un nœud dépend du type de nœud. En effet, l'ajout d'une classe et l'ajout d'une instance se ressemblent sémantiquement puisqu'il s'agit d'ajout, mais les opérations techniques invoquées diffèrent.

2.5.1.2. Contenu d'un nœud

Il s'agit d'un attribut (variable d'instance) et/ou d'une méthode, voire de contraintes. Selon la stratégie, les actions portent sur aussi bien des attributs que des méthodes ou uniquement d'attributs ou uniquement (voire principalement) de méthodes. Nous nous intéressons principalement aux attributs.

2.5.1.3. Arc

Un arc est tout type de lien liant deux nœuds. Il s'agira essentiellement du lien d'héritage, selon l'utilisation qui en est faite (spécialisation, implémentation, restriction). Il peut également s'agir des autres types de liens de références (association, composition) et du lien qui lie une instance à sa classe.

2.5.2. Taxonomie des opérations

Nous remarquons, au vu de la politique d'évolution, que nous pouvons dégager deux catégories d'opérations : des *opérations de base* et des *opérations avancées*. Quelques-unes de ces opérations peuvent être invoquées sur les différents types de produit alors que d'autres ne peuvent s'appliquer qu'à un seul type de produit.

2.5.2.1. Opérations de base

Nous entendons par *opération de base* toute opération qui s'applique sur un produit donné : un nœud, le contenu d'un nœud ou un arc. Une opération de base a une *sémantique* donnée : *modification*, *ajout* ou *suppression*. Selon le produit concerné, l'implémentation d'une opération est différente et sa portée est également différente. L'implémentation d'une opération dépendra donc du produit. La portée de l'opération dépend également du produit :

Produit \ *Opération*		Modification	Ajout	Suppression
Nœud	Instance	⊕	⊕⊕	⊕⊕
	Classe	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
Contenu d'un nœud	Attribut	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
	Valeur d'attribut	⊕	⊕	⊕
	Méthode	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
Arc	Lien d'héritage	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
	Lien de référence (association, composition)	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

⊕ : portée ponctuelle (au niveau du produit invoqué)

⊕⊕ : portée ponctuelle ∨ portée locale (vers les instances)

⊕⊕⊕ : portée ponctuelle ∨ portée locale ∨ portée globale (vers d'autres classes et les instances)

∨ : opérateur 'ou_logique'

Tableau 2-6: opérations de base

Ainsi, l'ajout d'une instance est différent de l'ajout d'une classe. La portée de l'ajout d'une instance est à la fois ponctuelle (ajout de l'instance) et locale (modification de l'extension d'une classe). De même, l'ajout d'une classe en tant que nœud interne d'une hiérarchie ou en tant que feuille d'une hiérarchie implique des portées différentes. Dans le premier cas, la portée concerne l'ensemble des sous-classes et les instances existantes. Dans le second cas, elle concerne l'ajout de la classe (et par conséquent de ses attributs et méthodes) et, par la suite, la création et la manipulation de ses instances.

2.5.2.2. Opérations avancées

Nous entendons par *opération avancée* toute opération qui s'applique sur deux produits. Le premier produit est la source de l'opération et le second est sa cible. Nous dégageons, au vu de l'étude des opérations mises en jeu par les stratégies étudiées de manière plus approfondie, et distinguons les opérations avancées suivantes : *Transfert* (ou *Migration*), *Scission* et *Fusion*. Une opération avancée, tout comme une opération de base, a une *sémantique* donnée. Selon le produit concerné, le transfert d'une instance vers une classe est différent du transfert d'une classe vers une autre hiérarchie de classes. L'implémentation des deux transferts est différente et les impacts sont également différents. De plus, la première nécessite, après la vérification de conformité à la nouvelle classe, la migration de l'instance. Alors que la seconde nécessite de faire appel aussi bien à des opérations de base qu'à des opérations avancées (selon la

politique de la stratégie) pour mener à bien le transfert et ses impacts. L'implémentation d'une opération avancée dépend donc du type du produit source et de celui du produit cible. La portée de l'opération en dépend également. Nous représentons les opérations avancées comme suit :

Opération / *Produit*		Transfert		Scission		Fusion	
		Source	Cible	Source	Cible	Source	Cible
Nœud	Instance	Instance Classe	Instance Classe	-	-	-	-
	Classe	Classe	Classe	Classe	Classes	Classes	Classe
Contenu d'un nœud	Attribut	Classe	Classe	-	-	Attributs	Classe
	Valeur d'attribut	Instance	Instance	-	-	-	-
	Méthode (signature et corps)	Classe	Classe	-	-	Méthodes	Classe
Arc	Lien d'héritage	Schéma	Schéma	-	-	-	-
	Lien de référence (association, composition)	Classe Schéma	Classe Schéma	Classe	Classe	Classe	Classe

Tableau 2-7: opérations avancées

Chaque opération avancée nécessite, pour son implémentation, d'avoir recours aux opérations de base. Nous mettons en évidence principalement la source et la cible de chaque opération avancée, selon le type de produit. Nous n'exprimons pas la portée de chaque opération premièrement pour des raisons de lisibilité et secondement parce qu'elle peut se déduire des opérations de base qui sont nécessaires pour mener à bien chaque opération avancée.

Il est à noter que les opérations de base sont nécessaires dans la mise en œuvre de toutes les stratégies, mais les opérations avancées ne sont pas forcément toutes utilisées : cela dépend de la politique de la stratégie d'évolution choisie.

2.5.3. Bilan

Nous avons initié la présentation des opérations et la distinction entre les opérations de base et les opérations avancées pour plusieurs raisons :

– Cette façon de considérer les opérations, aussi bien de base qu'avancées, permet d'uniformiser la présentation des stratégies, et par conséquent de les comparer.

– De plus, elles permettent d'exprimer, en termes d'opérations recensées et connues, la politique d'évolution de chaque stratégie.

– Cela permet de mettre en évidence comment est assuré le processus d'évolution de développement ou d'émergence et quelles sont les opérations menées et surtout la source et la cible des opérations ainsi que leur séquence. Ainsi, il est possible d'exprimer de façon opératoire comment est assuré le type d'évolution : du curatif ou du préventif (avec rupture ou continu). En effet, les mêmes opérations peuvent être utilisées dans les deux cas, aussi bien pour du développement que de l'émergence, mais la différence est marquée par :

- Le niveau de départ de l'évolution (classes ou instances) ;
- La séquence dans laquelle les opérations sont effectuées ;
- Le niveau final de l'achèvement de l'évolution ainsi que ses impacts.

De plus, pour exprimer la politique d'évolution de toute stratégie faisant de la modification d'une hiérarchie de classes, nous pouvons avoir recours à ces deux ensembles d'opérations. En conclusion, nous pouvons exprimer de manière plus claire la représentation triptyque de l'évolution d'objets par modification de hiérarchies de classes. L'axe principal reste le produit, et le type d'évolution ainsi que le processus d'évolution se complètent mutuellement par rapport au produit considéré.

Nous récapitulons en faisant remarquer que ces opérations d'évolution sur le produit sont utilisées différemment mais surtout dans des séquences différentes lors de l'exécution (de façon automatisée ou non) pour assurer les objectifs des stratégies :

Type Évolution / **Processus d'Évolution**	**Préventif**		**Curatif**
	avec rupture	**continu**	
Développement	- Réorganisation de classes - Classification de classes - Correction de classes	- Versionnement de classes - Migration de caractéristiques	
Émergence	Catégorisation		Notre modèle

Tableau 2-8: stratégies d'évolution par modification d'une hiérarchie de classes

En ce qui concerne notre approche d'émergence curative, pouvons-nous utiliser uniquement des opérations de ces deux taxonomies ? Pour pouvoir répondre à cette

question, nous rappelons les objectifs que nous nous sommes fixés jusque là (chapitre 1). Ces objectifs sont classés par ordre décroissant, du plus conceptuel au plus 'technique' :

– Offrir un modèle pour la simulation d'évolution,

– À partir de l'évolution dynamique de la structure d'instances.

– Pour cela, il faut tirer parti d'une part des structures véhiculées par les instances (qui ne sont probablement plus entièrement conformes aux classes existantes) et des informations déjà connues, donc des structures existantes. L'objectif est de trouver d'éventuelles nouvelles structures, en combinant celles précédemment citées,

– Comment les combiner ? Comment varier les solutions tout en assurant un minimum de cohérence et de sémantique pour ces solutions ? Car il faut en effet arriver à faire émerger plusieurs structures potentiellement candidates,

– Il faut ensuite faire un choix et répercuter les conséquences de ce choix dans la hiérarchie de classes existante.

– Il apparaît que nous pouvons avoir recours à quelques-unes des opérations, voire à l'ensemble des opérations de modifications lors de la modification de la hiérarchie de classes. Nous pouvons également avoir recours aux opérations de base d'ajout et de suppression d'attributs sur des instances existantes. Par contre, aucune ne nous permet de mener la simulation et les résultats escomptés de cette simulation d'évolution à partir de l'évolution d'instances.

Avant d'aborder dans la section 2.7 - comment nous proposons d'aborder l'émergence curative, nous présentons dans la section suivante d'autres stratégies d'évolution :

2.6 - Autres travaux portant sur l'Évolution d'Objets

Nous présentons dans cette section quelques stratégies d'évolution connues pour illustrer différentes politiques et techniques d'évolution proposées autres que par modification, avant de faire un bilan aussi bien en termes de limites rencontrées que de positionnement de toutes les stratégies présentées dans le triptyque. Nous parlerons également de quelques stratégies d'évolution de processus simplement à titre illustratif de cette problématique complémentaire à celle du produit.

2.6.1. Autres stratégies d'évolution de classes

Nous avons étudié les stratégies d'évolution les plus importantes portant sur l'évolution de produit par modification en section 2.4 - . Nous présentons dans cette section d'autres stratégies d'évolution de classes. Voici, en substance, leur principe, leur avantages et leurs limites :

2.6.1.1. Ajustement de classes

Principe : elle est principalement utilisée en Intelligence Artificielle et en Génie Logiciel. Elle consiste à adapter légèrement les définitions de classes lorsque ces dernières ne se prêtent pas à une spécialisation aisée. Les définitions existantes des classes ne sont pas directement modifiées et les adaptations sont appliquées aux propriétés héritées au moment de dériver de nouvelles sous-classes. Le but principal est de garder une hiérarchie de classes stable en évitant de la modifier à chaque évolution. Les mécanismes habituellement utilisés sont ceux des langages orientés objet : le renommage d'attributs et de méthodes, la redéfinition de propriétés (types des variables, corps des méthodes et les interfaces de classes) ainsi que la surcharge [Meyer 88].

Avantages : l'ajustement est utile pour assurer de légers ajustements, par exemple pour des changements sur la définition d'une classe, tels que le renommage ou la surcharge, dans le but de permettre plus de flexibilité dans la conception, et ce au vu de la rigidité d'une hiérarchie. De plus, elle ne nécessite pas de factorisation de propriétés communes et les techniques utilisées peuvent être facilement implémentées.

Inconvénients : cependant, il existe certains risques si l'on ne reste pas vigilant quant à son utilisation. Elle peut mener à des structures de spécialisation incompréhensives et l'utilisateur n'a aucun moyen d'éviter le développement de collections de classes désorganisées. Une utilisation intempestive peut rapidement alourdir et compliquer le graphe de classes.

2.6.1.2. Approches de représentations multiples : Points de vue, Rôles, Vues

Principe : ces stratégies s'intéressent à l'évolution du comportement. Le but des *Points de vue*(7) [Bratsberg 93] est de rendre l'évolution d'une classe transparente en représentant son évolution grâce à des points de vue qui reflètent la sémantique. Un point de vue confère sur un objet de nouvelles propriétés qui ne sont pas dérivées de propriétés existantes [Benatallah& 97]. Le concept de *Rôle* a été introduit pour répondre aux besoins de représentation d'un comportement évolutif et variable d'un

objet et lui permet d'avoir des rôles multiples durant son cycle de vie [Pernici 91]. Quant aux *Vues*, elles permettent de représenter différentes perspectives de la même entité. Elles ont été utilisées pour gérer l'évolution de schéma [Bertino 92]. Les modifications de schémas ne sont pas réellement effectuées sur le schéma mais sont simulées par la création de classes virtuelles.

Avantages : leur principal avantage réside dans leur aptitude à permettre d'étendre et d'enrichir la définition de l'entité à modéliser. De plus, le mécanisme est dynamique (les données ne sont pas stockées mais elles sont calculées dés que la vue est utilisée). Elles constituent un outil performant pour la conception incrémentale, principalement parce que l'utilisateur peut modifier le schéma (dans le cas des vues), le rôle et le point de vue (dans le cas respectivement des rôles et des points de vue) sans propagation des modifications dans le schéma/rôle/point de vue partagé par les autres utilisateurs.

Inconvénients : l'expression de contraintes sur et entre les points de vues, ou les rôles ou les vues est relativement ardue. Cependant, le mécanisme de vue est généralement insuffisant pour supporter des évolutions de schémas supplémentaires et rencontre des problèmes pour positionner une classe virtuelle dans un schéma.

2.6.1.3. Approche par Scénario

Principe : c'est une stratégie qui s'inscrit également dans la gestion de l'évolution de processus. Un scénario est un comportement défini pour un ensemble d'objets associé à une classe. Il spécifie les interactions de ces objets avec leur environnement d'exécution. Il permet l'évolution de leur comportement dans le temps, au regard de situations changeantes et imprévisibles. Un scénario est donc défini et attaché à une classe et représente une définition d'un comportement au vu d'un environnement d'exécution [Guetari 95].

Avantages : elle est utile pour prendre en compte des changements imprévus étant donné qu'un scénario est décrit en termes de règles qui sont activées selon la situation.

Inconvénients : cependant, des contradictions peuvent exister entre des scénarios si certaines règles sont contradictoires entre elles. De plus, le concepteur n'a pas de visibilité globale des différents scénarios.

2.6.2. Limites des approches actuelles

Les travaux présentés précédemment montrent qu'il y a actuellement plusieurs propositions, traitant chacune un aspect de l'évolution d'objets suivant le type de

problèmes auxquels est confronté l'utilisateur. La plus large part des stratégies revient à la gestion de l'évolution des classes. Les différentes stratégies précitées offrent des techniques pour assurer l'évolution des spécifications de classes ainsi que leurs répercussions. Chacune d'entre elles présente des avantages et des inconvénients.

Par exemple :

– La réorganisation, la classification ainsi que la catégorisation touchent à la hiérarchie de classes. Elles s'articulent essentiellement autour du repositionnement de classes au sein d'un graphe après l'insertion d'une nouvelle classe.

– La correction, l'ajustement et le versionnement de classes portent, quant à elles, sur la gestion des modifications des spécifications au sein d'une classe. Chacune œuvre à optimiser un aspect du modèle. La première rend la mise à jour des classes effective après tout changement et préserve la cohérence globale. La seconde quant à elle, propose de légers ajustements dans la définition d'une classe. Le versionnement quant à lui garde une trace des évolutions des spécifications d'une classe.

– Enfin, les techniques de l'impact de l'évolution des classes sur les instances et l'évolution des instances elles-mêmes n'offrent pas de stratégies réelles d'évolution mais se contentent uniquement de rendre les instances conformes à la spécification de leurs classes.

L'évolution d'objets est basée sur la prédominance du concept de classe, et par conséquent de son évolution, alors que le concept a été initialement introduit pour représenter et abstraire les caractéristiques d'entités du monde réel. Ces dernières sont représentées par des instances qui sont bien plus bridées que les entités qu'elles représentent. Les changements se font de manière plus naturelle dans la réalité que dans les modèles qui les abstraient. L'évolution d'instances est toujours limitée par celle des classes, un aspect rigide et non naturel de leur évolution.

2.6.3. Second bilan : positionnement des principales stratégies dans le triptyque

Il est important de noter que nous n'avons présenté que les stratégies d'évolution qui nous semblent pertinentes pour notre travail. Parmi celles-ci, nous avons cité et étudié les stratégies les plus connues et les plus utilisées pour gérer l'évolution de produit. En répondant pour chacune d'entre elles au triplet de questions de la section 2.2.4, nous arrivons à situer les stratégies sur les deux axes *Type d'évolution* et *Processus d'évolution* (selon qu'elles agissent au niveau classe, des classes vers les instances ou au niveau des instances). De la même manière, nous pourrons situer toute stratégie d'évolution par rapport au repère triptyque de l'évolution d'objets en répondant au triplet de question.

Le positionnement des stratégies étudiées donne :

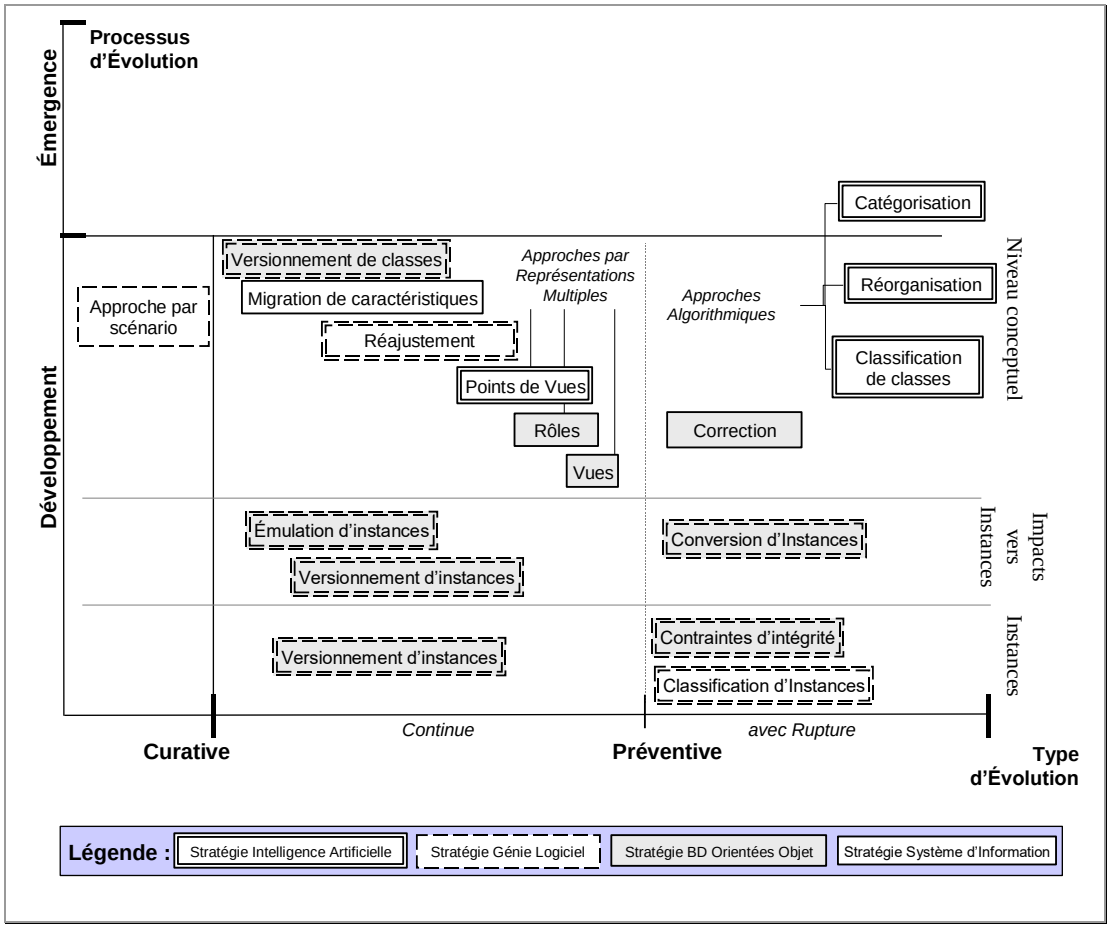


Figure 2-4: Seconde classification triptyque des stratégies étudiées.

Nous nous intéressons à la gestion de l'évolution de produit, que ce soit dans un cadre préventif ou curatif, et que ce soit dans le cadre du développement ou de l'émergence. Nous avons pu situer les stratégies d'évolution présentées par rapport aux facettes de l'évolution. Nous mettons en évidence, par le biais de cette illustration des stratégies d'évolution, les aspects de l'évolution d'objets qui sont traités plus ou moins largement dans la littérature de l'évolution d'objets.

Nous abordons dans la section suivante certains travaux de recherche qui présentent des similitudes sur des aspects différents, et parfois ponctuels, avec notre approche et notre problématique. Nous dénommons ces travaux comme étant connexes au nôtre et non pas comme étant forcément relatifs car ne présentant pas les mêmes objectifs ni les mêmes démarches.

2.6.4. Travaux connexes

Nous présentons dans cette section quelques travaux de recherche que nous considérons comme connexes, car ayant un ou plusieurs points communs avec nos travaux.

La problématique de [Segapeli& 99] est l'élaboration d'une méthode de conception permettant de concevoir un schéma objet à partir d'exemples. La méthode permet de définir des spécifications « floues » à partir d'informations connues d'exemples et d'affiner au fur et à mesure dès qu'il y a plus d'informations. Le processus de conception est ainsi incrémental. C'est une méthode d'aide à la construction d'un schéma de classes en permettant de gérer l'imprécision et l'incertitude en ayant recours à la théorie du flou. La méthode se base principalement sur la catégorisation pour l'aspect conception. Les auteurs utilisent des opérateurs et des règles pour modifier le schéma de classes floues et le transformer en un schéma objet. La méthode est d'un intérêt certain pour la conception et la spécification incrémentales faces à des situations imprécises et instables. Cependant, comme la construction se fait à partir des exemples, le schéma de classes en est trop tributaire.

Les travaux de [Li& 94] s'inscrivent dans le cadre de la problématique de l'évolution des bases de données objet. Leur approche est l'adaptation d'un modèle objet aux instances. A notre connaissance, ils représentent l'un des seuls travaux s'intéressant à l'adaptation d'un modèle objet aux instances, mais sans toucher à la structure d'instances existantes. Le principe est de pouvoir tirer profit de ce que veut l'utilisateur pour faire évoluer la base de données grâce à des techniques d'apprentissage. Pour cela, ils définissent un noyau de modèle et un niveau plus élevé. Ils considèrent qu'un objet peut évoluer d'un état à un autre, donc d'un niveau à un autre. Ils proposent un certain nombre de règles et de primitives de manipulation pour assurer ces évolutions.

Le modèle proposé à l'avantage d'avoir été mené à terme depuis la conception jusqu'au développement. Cependant, il nous apparaît que la gestion qu'une évolution nécessite est lourde. De plus, le système ne réagit qu'au cas par cas et de façon tout à fait ponctuelle. Il n'ouvre pas d'horizon pour l'évolution face à des situations de changement imprévues.

Une étude empirique [Li& 98] a montré l'importance et l'impact que peut avoir le niveau d'implémentation sur le niveau conceptuel tout au long du cycle de vie d'une application. En partant des hypothèses que l'évolution du modèle de conception influence l'évolution du modèle d'implémentation, et inversement, les auteurs ont mené cette étude. L'étude s'est particulièrement intéressée aux interactions et influences entre le niveau conceptuel et le niveau implémentation. Il apparaît, entre autres, que le niveau conceptuel a peu d'impacts sur l'évolution du niveau implémentation durant la vie de l'application. Par contre, la croissance du niveau d'implémentation tend à dominer la phase finale du projet, et connaît des besoins d'évolution particulièrement importants. La principale conclusion à laquelle leur étude mène est que **la croissance du modèle d'implémentation peut largement influencer l'évolution du modèle de conception.**

Astudillo [Astudillo 97] propose une *métaphore biologique* pour des hiérarchies d'objets, qu'il utilise pour spécifier et construire de manière incrémentale, et organiser les classes et les types séparément et de manière optimale. L'objectif est de permettre une maîtrise de ces spécifications afin d'offrir une évolution et une réutilisabilité plus accrues. Astudillo s'appuie sur des résultats des sciences biologiques et de la théorie des types. Ses travaux de recherche montrent que le monde des objets peut tirer profit des aptitudes naturelles à la fois simples et puissantes qui nous entourent, et de les adapter aux spécificités des objets.

2.7 - L'émergence curative : comment ?

Nous considérons les instances, en tant que représentantes d'entités réelles, comme des *individus* à part entière. Une instance a donc connaissance de ses propres spécificités, et surtout permet l'acquisition de nouvelles connaissances plus adaptées à son nouvel environnement. Elle pourra ainsi s'adapter et faire remonter au niveau conceptuel non seulement des structures conceptuelles mais également une voie possible d'évolution pour les instances issues de la même classe. Cette manière de considérer les instances, induit de facto l'analogie avec des individus vivants, du moins dans certaines de leurs propriétés : celles d'évoluer dans leur environnement en s'y adaptant. Une adaptation d'un individu ne se cantonnera pas à lui seul mais sera répercutée dans son espèce en modifiant son code génétique. Cet individu fera ainsi profiter de son évolution, sa population et sa progéniture.

En effet, dans la Vie Artificielle [Heudin 94], il est question de processus d'évolution d'un individu et de répercussion sur son code génétique. Les Algorithmes Génétiques [Holland 75] considèrent, pour des problèmes d'optimisation combinatoire, les diverses solutions potentielles comme des ensembles d'individus qui évoluent quasi aveuglément dans leur environnement instable et qui se reproduisent jusqu'à aboutir à un ensemble de solutions élitistes. Pour cela, des mécanismes ont été développés en s'appuyant sur ce qui a été observé dans la nature : la sélection naturelle et la reproduction d'individus ; le croisement et la mutation de leur code génétique.

D'un point de vue philosophique, cela revient à un processus d'apprentissage nécessaire et qui se rapporte à l'apprentissage de toute espèce vivante pour s'adapter et survivre. L'évolution, l'adaptation, les mutations se rencontrent à tous les niveaux, aussi bien d'un point de vue psychologique que physiologique que d'un point de vue temporel (une vie, une génération ou toute une espèce).

Nous nous inspirons pour cela des quelques principes et concepts des travaux de l'Évolution Artificielle. L'évolution d'instances et l'adaptation du modèle de classes aux nouveaux besoins permettra l'*émergence* de nouvelles structures conceptuelles.

2.8 - Les travaux de l'Évolution Artificielle

Nous abordons dans cette section l'Évolution Artificielle et ses principes et nous détaillons un peu plus deux principaux axes de recherche de l'Évolution Artificielle : la Vie Artificielle et les Algorithmes Génétiques. Ces travaux nous ont inspirés dans quelques notions et concepts :

2.8.1. L'Évolution Artificielle

Nous regroupons sous l'expression d'*Évolution Artificielle* les travaux de recherche qui se sont intéressés à la définition et à la mise en œuvre de systèmes artificiels évolutifs et adaptatifs. Certains de ces travaux s'inspirent de la biologie pour en extraire les capacités d'évolution, et les simuler à des fins différentes sur des supports, principalement l'ordinateur. Ils sont également dénommés *Algorithmes Évolutionnaires* car ils s'inspirent du concept de *sélection naturelle* élaboré par Charles Darwin. Les notions, entres autres, d'*individus* (solutions potentielles), de *population*, de *gènes* (variables), de *chromosomes*, de *parents*¸de *descendants*, de *reproduction* et de *mutations* sont manipulées. Le point clef est l'apparition, par hasard, de variations individuelles. La Vie Artificielle et les Algorithmes Génétiques s'inscrivent dans cette lignée [Langton& 90] et servent de support à cette étude.

2.8.2. Principe général

Le principe général est esquissé par la figure suivante, avec l'image de Darwin en filigrane qui rappelle la source d'inspiration :

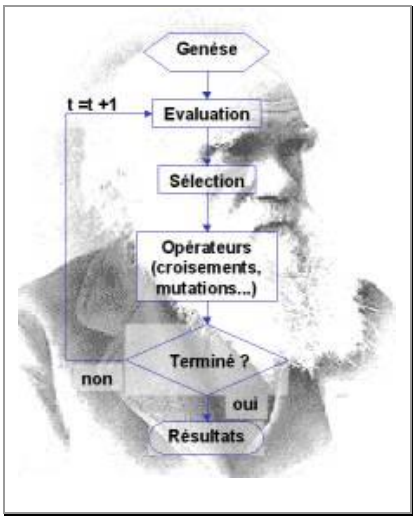


Figure 2-5: Principe Général d'un algorithme évolutionnaire.

La Figure 2-5 présente l'organigramme d'un algorithme évolutionnaire. Il s'agit de simuler l'évolution d'une population d'individus divers (généralement tirée aléatoirement au départ) à laquelle on applique différents opérateurs (recombinaisons, mutations…) et que l'on soumet à une sélection, à chaque génération. Si la sélection s'opère à partir de l'adaptation, alors la population tend à s'améliorer. Un tel algorithme ne nécessite aucune connaissance du problème : on peut représenter celui-ci par une boîte noire comportant des entrées (les variables) et des sorties (les fonctions objectif). L'algorithme ne fait que manipuler les entrées, lire les sorties, manipuler à nouveau les entrées de façon à améliorer les sorties, etc.

2.8.2.1. La Vie Artificielle

> « *…Les modèles de la Vie Artificielle … sont suffisamment puissants pour capturer une majeure partie de la complexité de systèmes vivants, déjà sous une forme plus aisément manipulable, répétitive, et sujette à des contrôles expérimentaux plus précis que ne peuvent l'être les systèmes naturels correspondants.* » C. Taylor et D. Jefferson. '*Artificial Life as a tool for biological inquiry*'. Artificial Life Journal, Volume 1, Number 1/2, Pages 1-13, 1994. The MIT Press, Cambridge, MA.

1) - Définition

La Vie Artificielle est un domaine de recherche récent. Le premier workshop a eu lieu en 1987 à Los Alamos (USA), avant de devenir le sujet de conférences internationales. Son principal objectif est la simulation et la synthèse des phénomènes biologiques. Elle repose sur l'abstraction des propriétés caractéristiques des êtres vivants, puis sur leur implémentation sur des supports arbitraires. Elle s'efforce ainsi à la compréhension de la vie en essayant de généraliser les principes sous-jacents aux phénomènes biologiques et de recréer ces principes. La Vie Artificielle propose pour cela une démarche en deux étapes. Dans un premier temps, elle tente d'abstraire les principes fondamentaux du vivant. Dans un second temps, elle 'réalise' ces principes sur un support arbitraire, l'ordinateur par exemple.

Elle s'intéresse aux processus d'évolution et elle est considérée comme la science des propriétés émergentes. En effet, du fait de leurs interactions, un ensemble de structures engendre des propriétés qui ne résultent pas de la simple superposition de leurs contributions individuelles mais donne également naissance spontanément à de nouvelles propriétés globales, absentes au niveau des structures prises indépendamment.

2) - Concepts

Aussi, pour prendre en compte les processus d'évolution, les notions de GTYPE et de PTYPE ont été empruntées respectivement au génotype et au phénotype de la biologie [Heudin 94] :

– Le génotype correspond à l'ensemble des informations génétiques encodées dans l'ADN.

– Le phénotype est l'organisme lui-même, autrement dit la structure qui émerge du développement du génotype dans le milieu environnant. Le processus du déploiement du phénotype est plus connu sous le nom de *morphogenèse*.

La Vie Artificielle généralise ces deux notions. Elle définit un génotype généralisé, appelé GTYPE et un phénotype généralisé, appelé PTYPE :

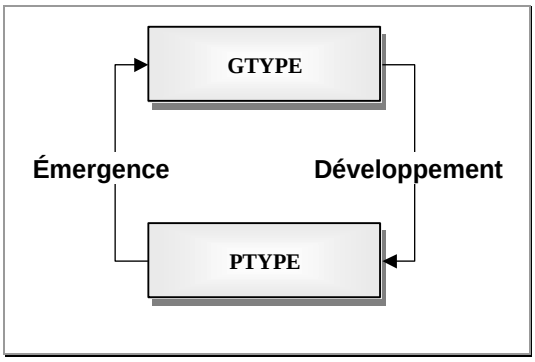
– Le GTYPE correspond à l'ensemble des informations génétiques d'un système étudié. Ses informations sont encodées sous forme de gènes au sein de chromosomes.

– Le PTYPE est constitué des individus représentatifs d'un système étudié et qui sont la réalisation du GTYPE dans un environnement donné.

3) - Principes d'évolution

Heudin [Heudin 94] précise qu'il est généralement impossible de prédire un PTYPE par une simple inspection d'un GTYPE et des données initiales de l'environnement…De la même manière, il est très difficile de spécifier les modifications à apporter à un GTYPE pour obtenir un résultat particulier au niveau de son PTYPE. La seule façon générale et efficace de procéder est d'exécuter le programme dans un processus itératif d'essais et de corrections d'erreurs. Ce processus est proche du principe de sélection naturelle … En retour, cette imprévisibilité fondamentale autorise une grande richesse de variétés possibles des PTYPE.

Le GTYPE et le PTYPE sont sans cesse en interaction pour s'enrichir et s'adapter. Cela se fait au travers de processus de *développement* (souvent décrit dans la littérature comme la *morphogenèse*) et d'*émergence*[4] :



[4] Dans la littérature de la Vie Artificielle, le processus d'émergence est souvent dénommé *Reproduction*. Quant à nous, le choix se porte sur le terme d'*émergence*, au lieu du terme *reproduction*, pour éviter toute confusion avec le mécanisme de reproduction des algorithmes génétiques.

Figure 2-6: les deux processus fondamentaux entre GTYPE et PTYPE

– Le processus de *développement* décrit le développement du GTYPE pour créer de nouveaux individus et enrichir le PTYPE.

– Le processus d'*émergence* décrit l'émergence de nouvelles propriétés au niveau des individus, donc du PTYPE, et qui doivent être insérées dans le GTYPE. Il est basé sur les connaissances détenues de l'espèce, de l'environnement et du contexte ponctuel.

Trois types d'Algorithmes Évolutionnaires ont été développés isolément et à peu près simultanément, dans les années 60, par différentes communautés scientifiques : les Algorithmes Génétiques, les Stratégies d'Évolution, et la Programmation Évolutionnaire. Nous nous sommes plus précisément intéressés aux Algorithmes Génétiques. Les raisons de notre choix sont explicitées au travers de la définition que nous donnons des Algorithmes Génétiques :

2.8.2.2. Les Algorithmes Génétiques

Inspirés de la théorie de l'évolution par sélection naturelle, les Algorithmes Génétiques [Holland 75] s'adaptent particulièrement aux besoins d'amélioration ou plus exactement de recherche de meilleures solutions à un problème donné, dans un processus itératif. Ce type d'algorithmes évolutionnistes a été appliqué à des problèmes d'optimisation combinatoire. Cependant, ils ont rarement été utilisés pour étudier l'évolution en elle-même. En effet, les chercheurs se focalisent plutôt sur les résultats que sur le processus d'évolution en lui-même, pourtant bien prometteur.

1) - Définition

Les Algorithmes Génétiques appartiennent à la classe des algorithmes basés sur l'émergence. Les Algorithmes Génétiques sont, compte tenu de leurs caractéristiques, particulièrement adaptés au problème de l'optimisation des configurations pour lesquelles l'univers des solutions potentielles est trop complexe à explorer par heuristiques, itératives ou non, « déductives » ou de calculs inverses [Holland 75].

Holland a formalisé les principes fondamentaux des Algorithmes Génétiques, dont voici la synthèse :

- La capacité de représentations élémentaires, comme les chaînes de bits, à coder des structures complexes.
- Le pouvoir de transformations élémentaires à améliorer de telles structures.

Plus récemment, Goldberg [Goldberg 94] enrichit la théorie des algorithmes génétiques. Il s'appuya sur le parallèle suivant:

- Un individu est lié à un environnement par son code génétique.
- Une solution est liée à un problème par sa valeur d'adaptation.

Une « bonne » solution à un problème donné peut être vue comme un individu susceptible de survivre dans un environnement donné.

L'étude des Algorithmes Génétiques est motivée par deux objectifs. Le premier est de tenter d'expliquer le processus d'évolution et d'adaptation des systèmes naturels. Le second est d'utiliser ce processus pour la conception de systèmes artificiels plus robustes, c'est-à-dire capables de s'adapter aux perturbations de l'environnement. Ils associent, pour cela, la diversité du hasard et la survie du plus adapté. Par ailleurs, les Algorithmes Génétiques permettent de produire toute une population de solutions potentielles. Ils réalisent ainsi une recherche multidirectionnelle tout en encourageant la formation de nouvelles solutions et l'échange d'information parmi ces solutions. La population subit une évolution à chaque génération, au cours de laquelle les solutions relativement acceptables se reproduisent tandis que les autres disparaissent. Les différents « morceaux » constituant la bonne solution se propagent d'une manière exponentielle au sein de la population (c'est le parallélisme implicite). Une fonction d'évaluation, représentant l'environnement et caractéristique du domaine applicatif traité, permet de différencier les solutions.

2) - Concepts

L'algorithme génétique de base repose sur la représentation binaire. Ce choix le rend virtuellement applicable à tous les problèmes dont les solutions sont transposables en binaire. Il emploi par ailleurs les mécanismes élémentaires de la théorie des algorithmes génétiques. La terminologie employée est empruntée à la génétique :

- Les *chromosomes* sont les éléments à partir desquels sont élaborés les solutions.
- La *population* est l'ensemble des chromosomes. Les populations sont également appelées des générations.
- La *reproduction* est l'étape de combinaison des chromosomes. La *mutation* et le *croisement génétique* sont des méthodes de reproduction.

D'autres notions sont propres au domaine des algorithmes génétiques :

- La *valeur d'adaptation*, aussi appelé indice de performance, est une mesure abstraite permettant de classer les chromosomes par ordre décroissant en tant que solution à un problème.
- La *fonction d'adaptation* est la formule théorique qui permet de calculer la valeur d'adaptation d'un chromosome.

3) - Principes d'évolution

Le principe de fonctionnement peut être résumé en la recherche du maximum d'une fonction du type y=f(x). Un algorithme génétique travaille à partir d'un ensemble d'éléments choisis au hasard dans un espace de solutions potentielles. Sa connaissance du problème à résoudre se résume à une fonction d'évaluation qui mesure la qualité d'un élément en tant que solution au problème. L'algorithme génétique tire partie des meilleurs éléments à sa disposition pour fabriquer de toutes pièces un nouvel ensemble d'éléments répondant en moyenne de mieux en mieux au problème.

Fonctionnement Général

1. Création de la population initiale.
2. Évaluation de chacun des chromosomes de la population initiale.
3. Sélection et regroupement des chromosomes par paires.
4. Application du croisement et de la mutation à chacune des paires.
5. Évaluation des nouveaux chromosomes et insertion dans la population suivante.
6. Si le critère d'arrêt est atteint, l'algorithme génétique s'arrête et rend le meilleur chromosome produit; sinon, l'algorithme retourne à l'étape 3.

Pour passer de l'ancien au nouvel ensemble, les éléments sont sélectionnés aléatoirement et proportionnellement à leur qualité en tant que solution au problème. Si un élément est retenu, il contribuera à améliorer les éléments du nouvel ensemble. Dans le cas contraire, il est irrémédiablement perdu pour les ensembles futurs. Les éléments sélectionnés sont ensuite combinés en de nouveaux éléments grâce à des méthodes inspirées de phénomènes naturels, comme le croisement génétique ou la mutation.

L'algorithme génétique travaille ensuite par générations successives jusqu'à un critère d'arrêt :

- soit le temps imparti est écoulé,
- soit le meilleur élément de la dernière génération atteint un seuil de qualité fixé au départ.

Ces différentes étapes se répartissent en deux phases, sur lesquelles les Algorithmes Génétiques opèrent [Goldberg 94] :

– L'*exploration* des différentes solutions possibles à la recherche des meilleures ;

– L'*exploitation* des résultats de ces recherches. Ils opèrent sur des individus (espaces de solutions) ayant un code génétique.

Ces individus ont la caractéristique d'évoluer « aveuglément » en se reproduisant et en perpétuant les meilleurs gènes au travers des individus porteurs. Pour les mener à bien, des *mécanismes génétiques* sont utilisés :

– La *sélection* aléatoire d'individus adaptés : la sélection des codes génétiques les plus adaptés implique la mesure quantitative de cette adaptation dite *valeur d'adaptation* grâce à une fonction d'adaptation. Une des idées directrices des algorithmes génétiques est donc d'affecter une probabilité de sélection supérieure aux meilleurs individus et, inversement, d'éliminer au fur et à mesure les plus mauvais. La performance d'un individu correspond aux capacités d'adaptation qu'il développe vis à

vis du monde dans lequel il évolue. La performance de l'individu est directement liée à la valeur de la fonction d'adaptation aux coordonnées de son phénotype.

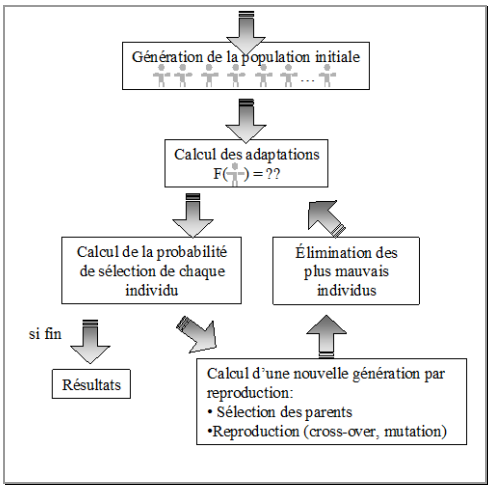


Figure 2-7: Le processus de sélection

– Le *croisement* du code génétique de ces individus pour récupérer les meilleurs gènes : l'opérateur de croisement peut être perçu comme une méthode de brassage des codes génétiques des différents individus. Elle consiste en fait à couper et à mélanger les différents patrimoines génétiques et, en alliance avec la sélection, à favoriser les meilleurs. Il existe plusieurs sortes de croisement. Le croisement binaire s'applique à une paire de chromosomes. Il est également appelé 'croisement binaire à 1 point' car les chromosomes se croisent autour d'une seule position appelée 'point de croisement'. Le point de croisement est positionné au hasard sur les chromosomes. Les bits à droite du point de croisement sont ensuite échangés pour former les deux nouveaux chromosomes.

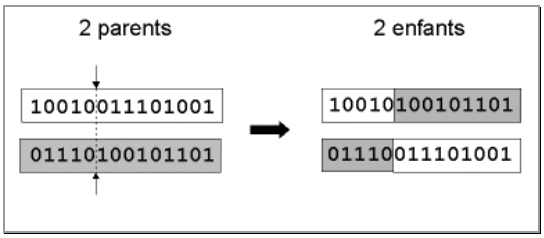


Figure 2-8: croisement à un point

- Il existe une variante, appelée croisement à n points, où les bits sont échangés entre les n points de croisement.

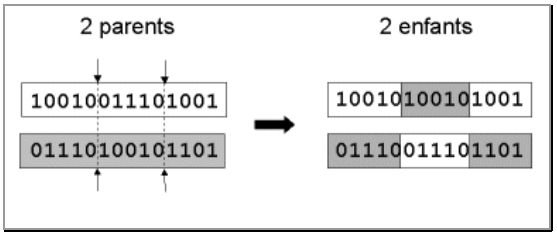


Figure 2-9: exemple d'un croisement à deux points.

- L'intérêt du croisement est de combiner les différents patrimoines génétiques de différents individus.

– La *mutation* pour faire muter favorablement un gène : contrairement au croisement, la mutation intervient sur un gène. Elle consiste à introduire une valeur aléatoire sur une position donnée, avec une probabilité assez faible d'intervenir.

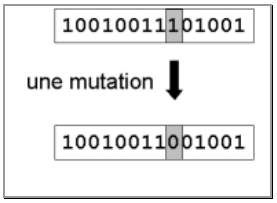


Figure 2-10: la mutation d'un gène au sein d'un chromosome

L'opérateur de mutation s'apparente à un apport de nouvelles données génétiques. Il permet en fait de faire réapparaître des gènes qui n'ont pas été sélectionnés au départ mais qui peuvent être performants par la suite.

– Les *schèmes* : un schème est une entité ayant la même structure génétique que la population qu'il représente. Il permet de spécifier un sous-ensemble de gènes d'une population. Ainsi, le schème 11### permet de représenter les individus d'une population donnée ayant les deux premiers gènes présents. Les gènes non spécifiés (les trois derniers dans l'exemple) sont considérés comme indifférents. Le schème est un moyen à la fois simple et puissant pour modéliser en même temps un ensemble d'individus.

2.8.3. Conclusion

Nous proposons dans ce chapitre une approche originale consistant à aborder l'évolution d'objets sous un angle d'évolution artificielle. L'évolution des classes a été

largement traitée. Par contre, la réflexion portant sur les instances en tant qu'individus capables d'évoluer reste à définir et à approfondir. Pour cela, nous considérons un modèle à objets comme étant un système se mouvant et évoluant dans un environnement donné. L'objet devra, de manière générale, évoluer et s'adapter en se basant sur sa propre connaissance et sur les perturbations émanant de l'environnement sans cesse instable.

Compte tenu du rôle des classes (respectivement GTYPE) et des instances (respectivement PTYPE) dans la modélisation d'un système réel ou artificiel, nous assimilons donc une classe à un génotype et une instance à un phénotype. Nous proposons d'expliciter le processus général d'évolution d'un modèle objet dans une boucle rétroactive et itérative sous l'optique de l'Évolution Artificielle en s'inspirant de [Heudin 94].

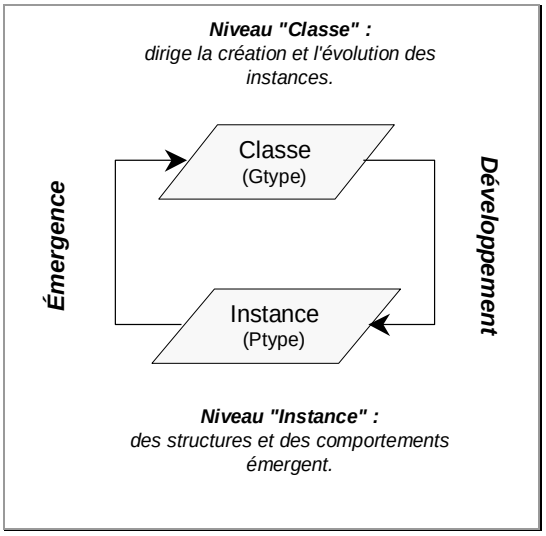


Figure 2-11. Processus d'Évolution d'Objets sous l'optique de l'Évolution Artificielle

Nous considérons l'ensemble des instances d'une classe comme son phénotype[5], et par conséquent que les gènes de l'instance sont ceux définis dans sa classe. Une classe, comme tout génotype, peut être modifiée par des mutations et des recombinaisons. Une instance est créée et modifiée à partir de sa (ses) classe(s) dans le *Processus de Développement.* Une classe peut être créée et modifiée à partir de l'expression de nouvelles caractéristiques au sein des instances dans le *Processus d'Émergence*.

A notre connaissance, peu de travaux de recherche ont combiné les approches basées sur le paradigme objet et sur la Vie Artificielle ([Astudillo 97], [Meslati& 97]). Il nous

[5] Une instance est une réalisation d'une classe de la même manière qu'un phénotype est une réalisation d'un génotype.

semble que cette nouvelle voie est prometteuse dans l'analyse et la conception de l'évolution d'objets, et peut fournir, somme toute, de nouveaux outils de modélisation et de simulation de systèmes complexes.

2.9 - Conclusion

Nous avons abordé plusieurs aspects dans le présent chapitre. Un des premiers aspects a été de proposer une classification de l'évolution d'objets par rapport à ses caractéristiques que nous avons dénommées *facettes*, qui sont au nombre de trois : l'objet de l'évolution, le type de l'évolution ainsi que le processus de l'évolution. La classification de l'évolution d'objets par rapport à ces facettes permet d'offrir un référentiel commun et indépendant des stratégies d'évolution. Chaque stratégie peut être positionnée par rapport à ce référentiel triptyque en précisant sur quel élément de chaque facette elle agit.

Par la suite, notre intérêt s'est principalement porté sur l'évolution de produit (classes – hiérarchies de classes – instances). Nous avons présenté certaines des stratégies d'évolution, chacune dans son principe (politique d'évolution et domaine d'utilisation), ses avantages et ses limites. Ces stratégies sont présentées selon qu'elles agissent au niveau classe, des classes vers les instances ou sur les instances. Par la suite, nous avons classifié ces stratégies par rapport aux facettes de l'évolution de produit. La principale conclusion est que la majorité des stratégies d'évolution font du préventif, et ce dans le cadre du développement. Notre approche, qui se veut complémentaire des stratégies d'évolution existantes, s'intéresse à faire de l'évolution de produit principalement de manière curative et principalement dans le cadre de l'émergence.

Nous nous sommes focalisés, par la suite, aux mécanismes d'évolution proposés par les stratégies d'évolution basées sur la modification de hiérarchies de classes. Nous dégageons une taxonomie des opérations de base et une taxonomie des opérations avancées. Cette taxonomie permet de définir de façon opératoire comment s'applique la politique de ces stratégies et dans quelle séquence. Elle permet également de décrire le type de l'évolution et le processus d'évolution.

Comme nous proposons un modèle pour la simulation d'évolution à partir de l'évolution dynamique dans la structure d'instances, nous remarquons que la taxonomie des opérations ne nous suffit pas pour mener à bien nos objectifs. Elle est utile en amont et en aval de la simulation mais s'avère insuffisante pour la manipulation dynamique de la structure des instances et surtout pour la simulation d'évolution. Pour cela, il est nécessaire de définir un modèle objet en gardant les avantages de la modélisation objet mais en l'étendant (en termes de concepts et de mécanismes) pour supporter une évolution de produit curative dans le cadre de l'émergence. Comment ? En s'inspirant des travaux de l'Évolution Artificielle qui considèrent et permettent aux individus d'évoluer et de s'adapter en perpétuant les meilleurs individus, donc les meilleures combinaisons génétiques. Le tout dans un but

de survie et d'adaptation à un environnement méconnu et instable. En effet, les changements se font de manière plus naturelle dans la réalité que dans les modèles objets qui les représentent, alors que l'approche objet est basée sur la prédominance du concept de classe.

Nous allons voir dans le chapitre suivant comment définir notre modèle.

Chapitre 3 : GENOME : LES CONCEPTS

3.1 - Introduction

Pour gérer l'évolution d'objets au vu de notre problématique et de nos objectifs, nous proposons un modèle. Nous dénommons ce modèle objet GENOME pour **G**énétique pour l'**E**volutio**N** **O**bjet et **M**odèl**E** ou encore **G**enetic **E**volutio**N** **O**bject Mod**E**l.

Nous abordons dans le présent chapitre la présentation et la définition des différents concepts de GENOME. Pour mieux les comprendre, nous introduisons en premier les préliminaires nécessaires pour présenter les processus d'évolution de GENOME.

Il est important de noter que nous nous basons sur un modèle objet classique s'appuyant sur les concepts de base que sont la classe, l'instance, l'héritage simple, l'attribut (simple, composé ou de référence) et la méthode. Nous nous basons sur des concepts de langages orientés objet existants et nous voulons les étendre pour prendre en compte l'évolution dynamique de la structure d'instance. Cette évolution se fait sous l'influence des perturbations émanant de l'environnement et peut provoquer des modifications dans la hiérarchie de classes existante. Nous considérons que ces perturbations sont exprimées et répercutées par la main du concepteur ayant la charge de la gestion de l'application. Pour véhiculer nos objectifs, nous enrichissons certains de ces concepts et les complétons en leur associant de nouveaux concepts.

Étant donné que nous abordons l'évolution de la structure d'instances avec des répercussions sur la hiérarchie de classes, nous avons proposé de considérer les instances comme des individus ayant la possibilité d'agir sur leurs caractéristiques, donc sur leur 'patrimoine génétique'. Pour cela, la notion la plus importante à introduire et à prendre en compte dans GENOME est celle de **mémoire génétique**, ainsi que sa perpétuation partielle ou totale selon les besoins d'adaptation.

Cette notion de mémoire génétique est en effet importante car elle permet d'une part l'éclosion, l'existence et la fin d'individus, et d'autre part elle peut être modifiée par recombinaisons et mutations grâce à la capacité des individus à s'adapter.

Cette notion de mémoire génétique se décline de diverses manières. Nous considérons qu'il existe trois entités qui la véhiculent entièrement :

– Il y a avant tout l'**espèce** considérée. Celle-ci regroupe les gènes communs à tous ses individus. Elle se perpétue par leur biais (par reproduction en biologie, par exemple). Ce groupe de gènes communs est donc l'essence même de la définition de l'espèce et justifie ainsi l'appartenance d'un individu ou non en son sein. Une **population** d'une espèce est constituée d'un ensemble d'individus existants sur une période donnée.

– Ensuite, il y a ce qui caractérise un peu plus les individus entre eux. Des groupes d'individus d'une même espèce présentent tout de même des différences. Outre l'appartenance à la même espèce, des individus peuvent avoir des similitudes plus ou moins nuancées qui les regroupent. Elles sont véhiculées par les gènes transmis par leurs **ancêtres**. Cette mémoire, ajoutée à celle commune de l'espèce, permet de définir tout l'héritage d'un individu.

– Enfin, il y a ce qui caractérise chaque **individu**, en plus de son appartenance à une espèce et à ses ascendants, les caractéristiques qui lui sont propres et uniques. Ce qui permet d'assurer la diversité et la richesse de toute une population, et voire même de toute une espèce.

Nous considérons qu'il est important de représenter et d'abstraire correctement ces différentes notions au travers de concepts et de pouvoir les manipuler. C'est ce qui fait l'objet de ce chapitre :

3.2 - Exemple

Pour illustrer au fur et à mesure les concepts, nous nous basons sur un exemple didactique modélisant les différents membres d'une université. Le modèle de classes suivant représente, à un instant donné, le modèle objet qui en a été abstrait :

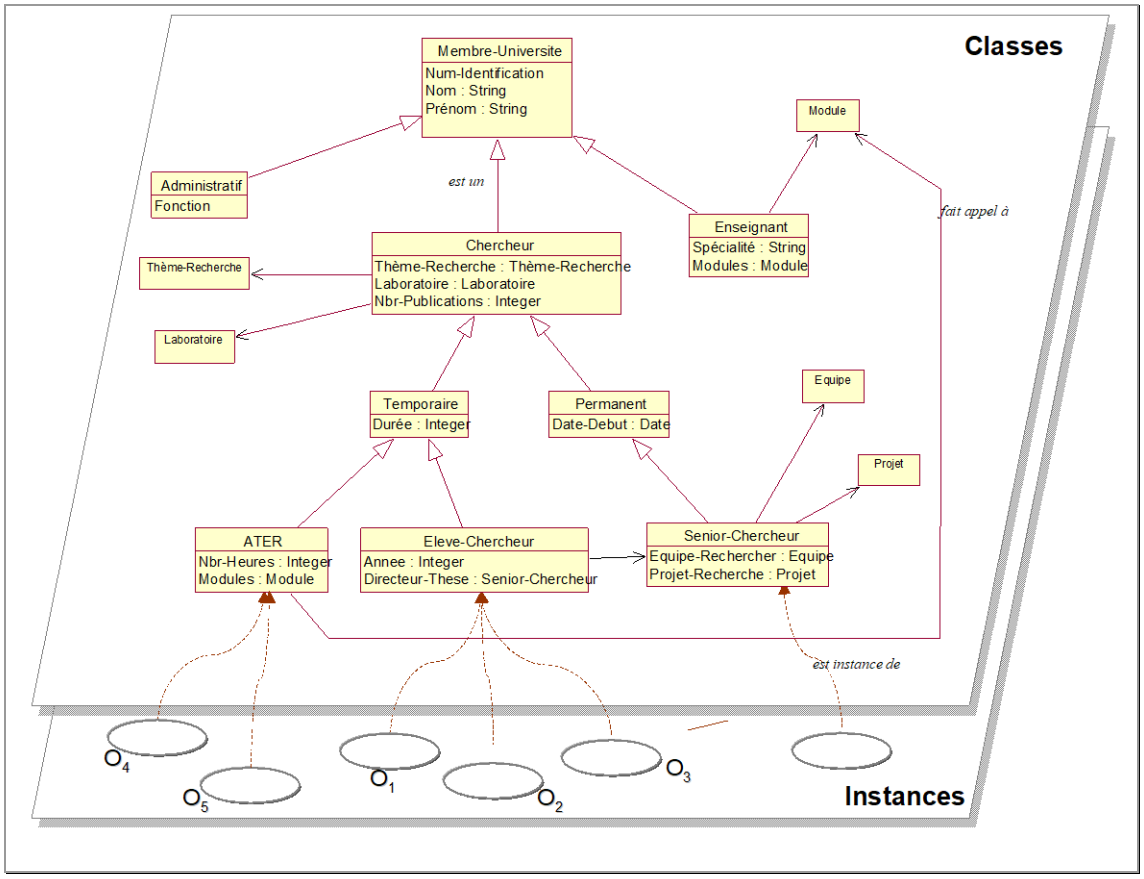


Figure 3-1: un exemple de modélisation des membres d'une université

Les différentes classes constituant les membres d'une université sont représentées. Celles-ci sont en relation avec d'autres classes (*Thème-Recherche*, *Laboratoire*, *Module*, *Equipe* ou encore *Projet*) ne représentant pas des membres d'université, mais elles appartiennent au système global. Elles peuvent donc être considérées comme faisant partie de l'environnement des membres d'une université.

La conception faite (qui est rarement unique et rarement stable au premier jet) met en évidence la classe *Membre-Université*. Celle-ci regroupe les caractéristiques communes à tous les membres. Elle constitue ainsi la racine de la hiérarchie d'héritage. Elle représente une population de l'espèce *Membre-Université*. Il est à noter que nous représentons uniquement les attributs. Les méthodes sont masquées, premièrement pour des raisons de lisibilité et deuxièmement nous rappelons que nous travaillons essentiellement sur les attributs.

Le modèle obtenu met également en évidence trois principaux membres d'université : *Administratif*, *Enseignant* et *Chercheur*. Chacun étant représenté par une classe.

Les chercheurs se scindent en Chercheurs *Temporaires* et Chercheurs *Permanents*. La principale distinction se fait sur la base de la durée de la présence du chercheur au sein de l'université :

- Un chercheur permanent est potentiellement à l'université de manière illimitée dans le temps (jusqu'à la démission, la retraite ou le décès). C'est

pour cela que seul l'attribut '*Date-Début*' est défini pour représenter la date de prise de fonction. Pour l'instant, la classe *Senior-Chercheur* est la seule modélisée.

- Un chercheur temporaire est présent pour un laps de temps précis, représenté par l'attribut *Durée*. Il n'y a pas de distinction faite sur son statut d'étudiant, de salarié ou autre. Deux classes sont représentées :
  - les *ATER*, qui en plus de leur recherche, assurent des enseignements ;
  - les *Elèves-Chercheurs* qui travaillent sous la direction d'un directeur de thèse.

Une contrainte est également posée par le concepteur. Il précise en effet que l'attribut '*Durée*' défini dans *Temporaire* et l'attribut '*Date-Début*' défini dans *Permanent* sont contradictoires, c'est-à-dire qu'un membre de l'université ne peut pas valuer les deux attributs en même temps.

Nous donnons dans la suite les notions, concepts et définitions de base de GENOME :

3.3 - Préliminaire des processus d'évolution

Les principales notions que nous introduisons dans l'évolution d'objets sont celles d'*émergence* et de *développement*, préalablement introduites dans le chapitre 2. Elles représentent en effet l'une des trois facettes de l'évolution d'objets : la facette *Processus d'évolution*. Ces notions sont empruntées à la Vie Artificielle (chapitre 2).

Nous rappelons que nous nous intéressons à l'évolution de produit. Nous considérons que *toute* évolution du produit est soit un processus de développement, soit un processus d'émergence.

3.3.1. Le développement

Le *développement* est un processus qui permet l'évolution d'un modèle en modifiant les structures conceptuelles détenues par les classes, avec tous les impacts engendrés comme conséquence sur les classes et les instances concernées. Nous reviendrons plus loin (chapitre 4) sur le détail de la définition suivante (introduite dans le chapitre 2) : il s'agit du développement des classes en intention (leurs spécifications ainsi que celles des classes concernées) et en extension (les instances existantes et concernées).

3.3.2. L'émergence

L'*émergence* est un processus qui permet l'évolution des structures conceptuelles d'un modèle comme conséquence de l'évolution de la structure de ses instances. Nous reviendrons plus loin (chapitre 4) sur le détail de la définition suivante (introduite dans le chapitre 2) : il s'agit d'émergence de nouvelles informations conceptuelles à partir de l'évolution d'instances.

Ces deux processus de développement et d'émergence se complètent et couvrent ainsi tout le cycle d'évolution d'un modèle objet. Ils peuvent également s'interpeller pour se compléter. Nous verrons (chapitre 4) que l'émergence fait appel au développement afin d'assurer la gestion des impacts de l'émergence.

Nous présentons les concepts du modèle GENOME. Nous distinguons des concepts de base et des concepts avancés.

3.4 - Les concepts de base

Les concepts de base sont plus une manière de considérer les concepts objets classiques que la définition de nouveaux concepts à proprement parler. Nos concepts de base sont les concepts objets classiques (que sont l'attribut et la méthode, la classe ainsi que l'instance) vus au travers des concepts et des notions générales, et parfois philosophiques, de l'Évolution Artificielle.

Nous estimons en effet que les concepts objets se prêtent bien à une telle interprétation. C'est ainsi que nous empruntons le rôle des concepts de *gène*, de *GTYPE*, de *PTYPE* ainsi que les notions de *population* et de *patrimoine génétique* à l'Évolution Artificielle pour les adapter et les étendre au formalisme objet (classe/instance).

3.4.1. Gène

Un gène est toute propriété structurelle (attribut ou variable d'instance) ou comportementale (méthode) pouvant être définie dans une classe et permettant ainsi de spécifier une caractéristique des instances (directes ou indirectes) de cette classe. C'est le concept élémentaire dans GENOME. Ainsi, chacun des attributs *Num-Identification*, *Nom* et *Prénom* est un gène défini dans la classe *Membre-Université* de l'exemple en Figure 3-1.

3.4.2. Classe-GTYPE

Le rôle d'une classe est de regrouper les similarités d'un groupe d'objets, tous décrits par les mêmes variables d'instances et répondant aux même messages. Une classe regroupe donc des gènes. Elle définit le code génétique de ses instances. Une classe peut ne pas avoir d'instances, du moins de manière directe (classe abstraite). Les propriétés qu'elle définit sont toutefois généralement transmises, par héritage, dans des sous-classes.

Une classe représente donc le *génotype* de ses instances. Nous qualifions une classe par une *Classe-GTYPE*, GTYPE par analogie au GTYPE de la Vie Artificielle. Ainsi, la classe Enseignant de l'exemple en Figure 3-1 détient les gènes, donc le génotype, de toutes ses instances.

Par souci de simplicité, nous utilisons parfois dans la suite du manuscrit le terme de *classe* en lieu et place du terme *classe-GTYPE*.

3.4.3. Instance-PTYPE

Le rôle d'une instance est de représenter une entité donnée du système modélisé. Elle est rattachée à une classe dont elle prend les attributs qu'elle value, et dont elle dépend en terme de comportement. Elle représente l'**individu** présenté en introduction.

Une instance est donc une réalisation d'une classe dans un environnement et un contexte donnés. Elle sera moulée et réagira selon ce qui a été défini dans sa classe d'appartenance et des ancêtres de celle-ci. Nous qualifierons une instance par une *Instance-PTYPE*, PTYPE par analogie au PTYPE de la Vie Artificielle. Ainsi, les objets O_4et O_5 de l'exemple en Figure 3-1 sont des instances-PTYPE de la classe-GTYPE Ater.

Par souci de simplicité, nous utilisons parfois dans la suite du manuscrit le terme d'*Instance* en lieu et place du terme *Instance-PTYPE*.

3.4.4. Population et Patrimoine Génétique

Une **population** est constituée d'un ensemble d'individus ayant un patrimoine génétique. Ce patrimoine génétique présente une partie commune à tous ces individus et identifie la population, donc l'**espèce**. Certains individus partagent en outre une partie de ce patrimoine qui est un héritage de leurs **ancêtres**. Et chaque **individu** peut introduire en plus quelques gènes dans ce patrimoine.

D'un point de vue objet, et en l'occurrence dans GENOME, une **population** est représentée par une classe qui est généralement la racine d'une hiérarchie de classes, et qui détient tous les gènes communs à cette hiérarchie (cf. section 3.5 - pour plus de détails sur les différents types de gènes). C'est le cas de la classe *Membre-Université* de l'exemple en Figure 3-1. L'ensemble des **individus** est l'ensemble des instances-PTYPE. Le patrimoine génétique est réparti dans les différentes classes-GTYPE. Ce que les

individus peuvent hériter de leurs ancêtres est transmis par le lien d'héritage d'un **ancêtre**, en fait une super-classe, vers un de ses éventuels descendants, en fait une de ses **sous-classes**. Ainsi, la classe Chercheur de la Figure 3-1 est l'ancêtre direct de la classe Temporaire et de la classe Permanent, qui sont à leur tour des ancêtres d'autres sous-classes. Enfin, ce qui caractérise directement un groupe d'individus, donc d'instances, est défini dans leur classe. Ainsi, la classe Ater de la Figure 3-1 introduit le **gène**, donc l'attribut, *Nbr-Heures* qui le distingue de la classe Eleve-Chercheur. De plus, un individu peut avoir des valeurs différentes.

– **Patrimoine Génétique :** l'ensemble des gènes définis par les classes d'une population constitue le *Patrimoine Génétique* de la dite population. Toute instance de la population prendra ses gènes dans la partie de ce patrimoine définie par sa classe. Nous considérons également que toute instance qui évolue au sein de la population peut potentiellement perdre un de ces gènes (qu'elle a déjà) et qu'elle peut potentiellement acquérir des gènes existants parmi ceux du patrimoine, même s'ils ne sont pas définis par sa classe. Elle peut en outre introduire de nouveaux gènes.

Pour délimiter une population, le choix est fait durant la phase de conception. Il peut être subjectif car il est laissé à l'appréciation du (ou des) concepteur(s). De la même manière que des choix sont faits pour délimiter et définir une classe, la délimitation et la définition d'une population dépendra des choix de conception faits. Par exemple, le concepteur peut décider de définir une population de *Personnes* puis une sous-population *Membre-Universite*. Un autre choix pourrait être de définir directement la population *Membre-Universite*.

Donc, au vu de toutes ces définitions, notre exemple en Figure 3-1 représente la population *Membres-Université*. La classe *Membre-Université* est la racine de la hiérarchie d'héritage associée et elle est composée de plusieurs classes et d'un ensemble d'instances.

Nous abordons maintenant la définition des concepts avancés du modèle GENOME :

3.5 - Les concepts avancés

Les concepts avancés sont de nouveaux concepts que nous proposons. Ils enrichissent et complètent les concepts de base et constituent un support pour gérer l'évolution. Nous distinguons les trois concepts avancés qui traitent du concept de classe-GTYPE et du concept avancé de schème.

Une classe définit le génotype de l'ensemble de ses instances. Chaque gène ne joue pas le même rôle et n'a pas la même prédominance ni dans la sémantique qu'il porte ni

dans l'influence qu'il peut avoir sur l'évolution des instances, ni sur la spécification d'éventuels descendants.

Au vu de nos objectifs et des notions d'espèce et de population, d'ancêtre et d'individu, nous considérons que toute classe est entièrement spécifiée au travers de trois génotypes. Chaque génotype étant constitué d'un ensemble de gènes.

3.5.1. Génotype Fondamental

3.5.1.1. Définition

D'un point de vue biologique, toute espèce du monde présente des caractéristiques fondamentales qui font sa raison d'être. Ses individus y répondent forcément tous. Autrement, ils ne pourraient pas être de cette espèce.

D'un point de vue de la modélisation objet, toute population est définie par un ensemble précis de gènes. Toutes ses classes détiennent ce même ensemble de gènes fondamentaux. Ces caractéristiques sont représentées par des gènes particuliers représentant la sémantique minimale et nécessaire sans laquelle un objet de chacune de ces classes ne peut plus appartenir à la population. Nous dénommons ce groupe de gènes le *Génotype Fondamental* ou *GF*. Ainsi, les trois propriétés *Num-Identification*, *Nom* et *Prénom* constituent le *Génotype Fondamental* de la population *Membre-Université*. C'est le **Génotype Fondamental** qui représente en fait l'**espèce**.

Nous faisons toutefois remarquer qu'une classe qui introduit le GF d'une population, peut également avoir comme toute autre classe, d'autres types de gènes (sections 3.5.2 et 3.5.3).

3.5.1.2. Principales propriétés du Génotype Fondamental

Le principal avantage du GF est d'une part de situer conceptuellement toutes les classes (et instances) d'une même population, et d'autre part de cerner l'évolution des instances. Une instance d'une des classes de *Membre-Université* peut évoluer parmi sa population, mais ne pourra pas évoluer en allant à l'encontre de sa propre population vers une autre population. Elle ne pourra pas par exemple devenir une voiture. Le GF constitue une sorte de garde-fou face à des évolutions intempestives et non contrôlées dans leur sémantique.

Une population (section 3.4.4) est donc spécifiée via son GF qui est partagé par l'ensemble de ses classes. Même si les propriétés que définissent les classes d'une population diffèrent, elles ont toutes ce groupe de gènes fondamentaux qui exprime leur appartenance à la même population.

L'existence d'une population passe donc par celui de son GF. Dès que le GF change (par le biais du concepteur) au sein d'une classe ou d'une instance, l'essence même d'une autre population (même vaguement cernée) est en train d'être exprimée.

Comme nous l'avons illustré, le GF est défini au sein d'une classe qui représente la population et qui est la racine de la hiérarchie d'héritage de toutes les classes de cette population. Le GF se transmet à ces classes via le lien d'héritage sans pouvoir être redéfini ou surchargé.

Cette même classe, comme nous l'avons déjà coté, peut définir d'autres gènes qui ne font pas partie du GF. Ces gènes se transmettent également via le lien d'héritage, mais contrairement aux gènes constitutifs du GF, ils peuvent être redéfinis ou surchargés par les sous-classes. Ceci est décrit plus en détails dans les deux sections qui suivent.

3.5.2. Génotype Hérité

Le *Génotype Hérité* ou *GH* d'une classe représente l'ensemble des gènes hérités de sa super-classe. C'est l'héritage de l'**ancêtre**. Le GH d'une classe constitue l'ensemble des caractéristiques héritées des ancêtres et qui peuvent être adaptées à la classe. Ainsi, comme toutes les propriétés héritées par une classe, celles-ci peuvent être redéfinies et surchargées selon les besoins de conception.

Le *Génotype Hérité* d'une classe est constitué du génotype hérité de sa super-classe et du génotype spécifique de cette même super-classe (la section 3.5.3. présente le génotype spécifique) :

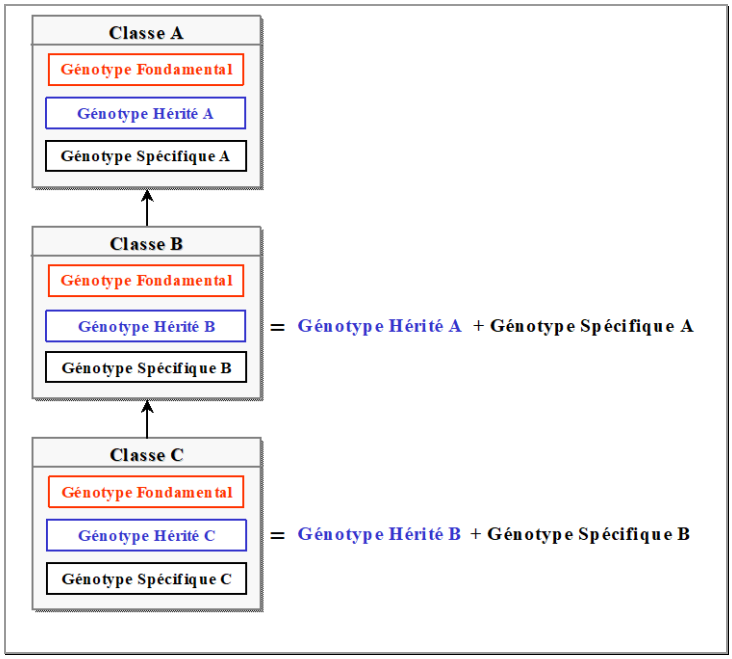


Figure 3-2: relations entre GH et GS

Les classes A, B et C appartenant à la même population. Le GF est donc identique car il définit la population.

3.5.3. Génotype Spécifique

Le *Génotype Spécifique* ou *GS* est constitué de tous les gènes localement définis au sein d'une classe, qui lui sont donc spécifiques. C'est ce qui caractérise un même groupe d'individus des autres individus de la même population.

Les GH et GS sont issus des spécificités liées à l'environnement et au contexte dans lesquels vivent les individus de la population, alors que le GF correspond aux caractéristiques spécifiques à cette même population, donc de l'espèce.

3.5.4. Le schème

Nous introduisons le concept de *schème* comme une entité qui transcrit de manière simple et précise la structure d'une classe ou d'une instance. Nous l'empruntons aux Algorithmes Génétiques dans le cadre desquels un schème est une entité ayant la même structure génétique que la population qu'il représente.

Il permet de spécifier à la fois un sous-ensemble de gènes du patrimoine génétique d'une population ainsi que leur présence ou non. Ainsi, le schème 1 1 # # 0 permet de représenter à la fois tous les individus d'une population ayant :

- les deux premiers gènes du patrimoine génétique obligatoirement présents ;
- les deux suivants indifférents (ils peuvent être séparément ou en même temps présents ou absents) ;
- le dernier gène obligatoirement absent.

Le schème est un moyen à la fois simple et puissant pour modéliser en même temps un ensemble d'individus répondant à des critères donnés.

Nous le considérons comme l'expression d'une contrainte génétique à laquelle sont soumis les individus et qui permettra d'effectuer sans ambiguïté des choix sur des classes et des instances.

Nous considérons pour cela deux types de schèmes : les *schèmes permanents* et les *schèmes temporaires*. Les premiers sont spécifiés en même temps que les classes. Les seconds sont déclarés et utilisés avec les premiers lors des diverses étapes des processus d'évolution.

3.5.4.1. Schème permanent

A chaque classe lui est associé un schème de manière permanente. Il a une structure identique à la sienne car il exprime le code génétique de ses instances. Un schème permanent comporte par conséquent trois parties : *génotype fondamental*, *génotype hérité* et *génotype spécifique*. A chaque fois qu'une classe est spécifiée, le type de ses propriétés (1 : obligatoirement présente ; 0 : obligatoirement absente ; # : optionnel) est traduit au

sein d'un schème permanent. Il représente une transcription simple de la structure d'une classe et de celle de ses instances. Une instance ne se voit donc pas attribuer de schème permanent, étant donné qu'elle respecte celui de sa classe. Le seul cas où l'instance se voit associer un schème, c'est lorsqu'elle évolue et se démarque de sa classe. Cette situation se présente dans le cadre d'un processus d'évolution, et le schème qui lui est alors associé est dit *schème temporaire*.

3.5.4.2. Schème temporaire

Un schème temporaire a la même structure qu'un schème permanent. C'est une unité de sélection d'un ensemble d'entités (instances ou classes) ayant des caractéristiques communes. Un schème temporaire est surtout utile lors de l'évolution d'une instance. Il est utilisé comme un filtre permettant la sélection d'entités adaptées au problème traité. Il permet également de manipuler rapidement et sans confusion classes et instances. Il constitue la base sur laquelle opèrent les processus d'évolution (chapitre 4). Nous verrons par la suite comment le schème est l'entité impliquée dans une compétition avec d'autres entités du même type et comment se feront la sélection et la propagation de gènes.

3.5.4.3. Représentation d'un schème

Nous décrivons un schème à l'aide de l'alphabet minimal suivant {0, 1, #}. A chaque fois qu'une classe est spécifiée, le type obligatoire ou optionnel de ses propriétés (attribut/méthode) est traduit au sein d'un schème permanent. La correspondance entre le type d'une propriété et son gène représentant dans le schème est comme suit :

- une propriété obligatoirement présente est représentée par 1 ;
- une propriété dont la présence est optionnelle est représentée par # (ce qui introduit une souplesse dans les spécifications des instances d'une classe) ;
- une propriété obligatoirement absente est représentée par 0 (ce dernier point est particulièrement utile lors des étapes transitoires dans les processus d'évolution).

De par les concepts et les définitions introduites, nous pouvons déduire et donc présenter le schème de la classe *Chercheur* qui est représenté par :

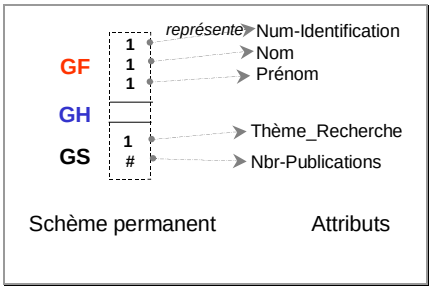


Figure 3-3. Schème permanent de la classe Chercheur

Tout chercheur doit donc toujours respecter le Génotype Fondamental en valuant obligatoirement les attributs le constituant (*Num-Identification* ; *Nom* ; *Prénom*) ainsi que l'attribut *Thème-Recherche* qui est défini dans le Génotype Spécifique. Par contre, la valuation de l'attribut *Nbr-Publications* n'est pas obligatoire (en effet, un *Eleve-Chercheur* qui débute n'a pas de publications).

Nous notons le cas particulier suivant : il n'y a pas de gènes définis dans le Génotype Hérité de la classe *Chercheur*. En effet, même si sa super-classe est *Membre-Université*, la classe Chercheur hérite uniquement du Génotype Fondamental. Elle n'a pas de Génotype Hérité car la classe *Membre-Université* ne définit pas de Génotype Spécifique. Ce cas ne contredit en rien la manière dont est constitué le Génotype Hérité de toute classe, telle que défini dans la Figure 3-2.

3.5.5. Le lien d'évolution

Une fois la structure conceptuelle émergente créée et insérée dans le modèle de classes, les objets ayant évolué sont rattachés aux classes correspondantes. Un lien est créé entre la classe source (initiale) de l'instance vers sa classe cible (nouvelle). Ce lien exprime en fait l'un de nos objectifs, celui de permettre au modèle d'apprendre un des ses comportements futurs. Le modèle apprend un nouveau comportement évolutif et une voie possible d'évolution non seulement de ses instances, mais également de son modèle de classes. Le lien d'évolution est abordé de manière plus détaillée dans le chapitre suivant.

3.6 - L'exemple et sa représentation

Le modèle de l'exemple présenté dans la Figure 3-1 est décrit dans GENOME comme suit :

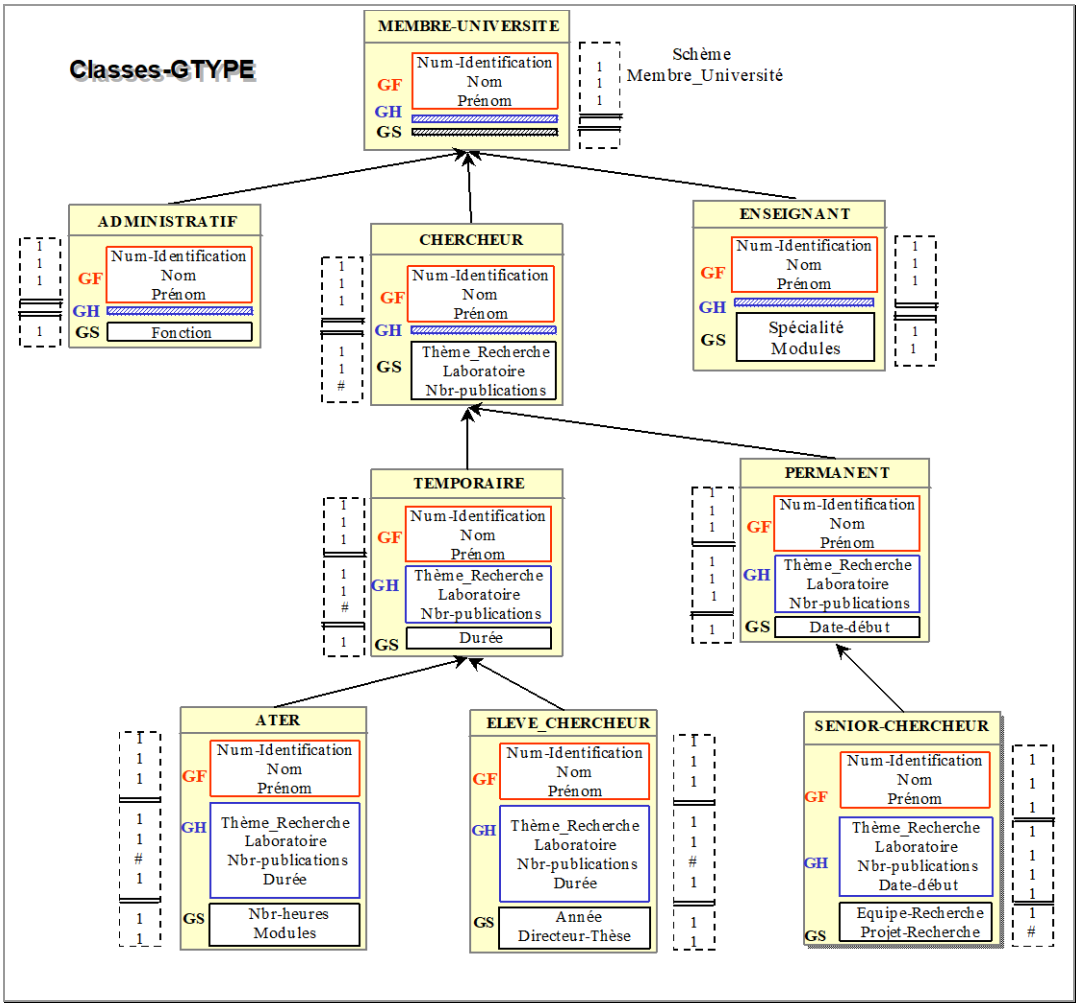


Figure 3-4: le même exemple sous GENOME

La classe Membre-Universite définit uniquement le GF de la population. Elle n'a pas de GH car elle n'hérite d'aucune classe, et ne définit pas de gènes pouvant être hérités par les sous-classe. Le modèle de la Figure 3-4 met en évidence les différents génotypes ainsi que les schèmes permanents. Ainsi :

$GH_{(Eleve\_Chercheur)} = GH_{(Temporaire)} + GS_{(Temporaire)}$ et $GH_{(Temporaire)} = GH_{(Chercheur)} + GS_{(Chercheur)}$,

avec l'éventualité de renommage et de surcharge des génotypes hérités $GH_{(Eleve\_Chercheur)}$ et $GH_{(Temporaire)}$.

De plus, le schème permanent de *Senior-Chercheur* montre que l'attribut *Nbr-Publications* est obligatoirement présent (1) dans toutes ses instances, alors qu'il est optionnel (#) pour la classe *Temporaire* et ses sous-classes.

3.7 - Le méta-modèle

Après avoir introduit et défini les concepts du modèle, nous pouvons exprimer les spécifications de ce dernier. Tous les concepts sont réifiés. En effet, nous nécessitons d'exprimer et de manipuler aussi bien les classes que les instances, les attributs et les méthodes.

Nous présentons la structure du modèle dans ce qui suit. Nous présentons en premier lieu le formalisme utilisé.

3.7.1.1. Formalisme utilisé

Pour décrire les principaux concepts de GENOME, nous empruntons le formalisme d'UML [Muller 97]. Aussi, les concepts clés de classe, d'héritage, de composition et d'associations seront utilisés, ainsi que les notions de rôles et de multiplicité (cardinalités) sur les relations.

3.7.1.2. Synoptique du modèle

Le méta-modèle met en évidence les concepts présentés. Aussi, la présentation qui en est faite se focalise principalement sur la manière dont sont organisés les concepts et présente leurs caractéristiques structurelles. Les seuls attributs présentés dans le méta-modèle sont ceux qui reflètent la sémantique et les caractéristiques des concepts présentés au niveau du chapitre. Les méthodes de chaque classe ne sont pas présentées afin d'avoir un modèle de lecture aisée. Les méthodes ainsi que d'autres attributs nécessaires à l'implémentation du modèle sont définis et utilisés dans la phase d'implémentation durant la tâche de développement du prototype (chapitre 6).

Le méta-modèle est décrit par le modèle de classes UML suivant :

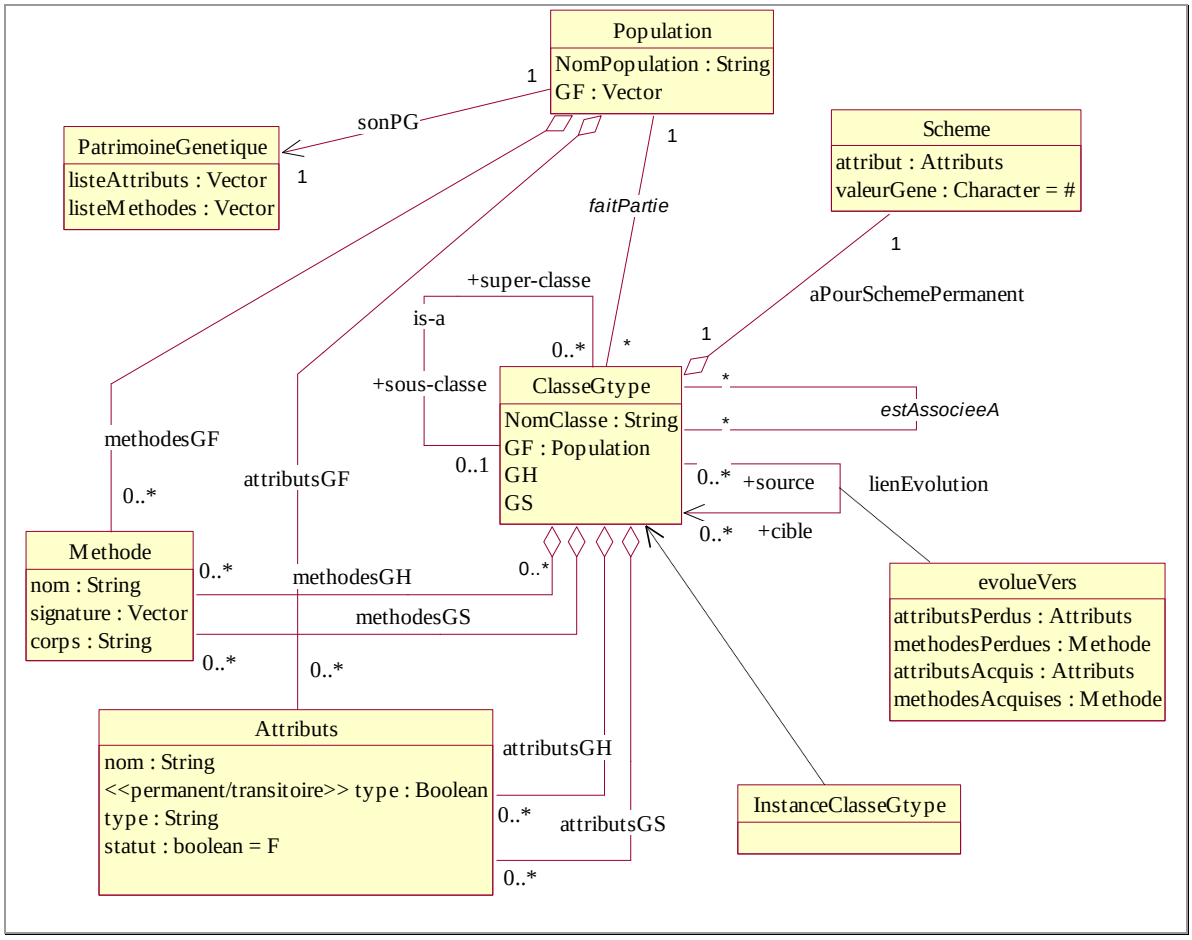


Figure 3-5. Architecture du modèle GENOME selon le formalisme d'UML

Les classes et les liens décrits au sein du modèle sont la représentation des concepts présentés tout au long de ce chapitre. Nous définissons séparément les classes et les liens :

3.7.1.3. Définition et rôle des classes

1) - Population

Cette classe modélise les populations. Chaque population est caractérisée par un nom et le Génotype Fondamental. Ce dernier est constitué d'un certain nombre de méthodes et d'attributs. Il est le même pour toutes les classes de la même population. Le génotype fondamental est fixé lors de la création de la population et il ne peut pas être modifié, sauf spécification explicite de la part du concepteur.

2) - PatrimoineGenetique

La classe PatrimoineGenetique permet de définir l'ensemble des propriétés (attributs et méthodes) présentes dans toute la population. Une instance de PatrimoineGenetique

est rattachée à une instance et une seule de Population. Cette dernière a, en plus de ce qui a été décrit dans la classe Population, son patrimoine génétique. Une instance de PatrimoineGenetique est constituée d'un ensemble de références d'instances d'Attributs et de Methode. Elle peut donc être constituée et complétée selon les besoins, en manipulant les différentes classes de la même population.

De manière plus précise, toute instance de PatrimoineGenetique est définie pour une instance de Population et est définie par l'ensemble des références (sans redondance) des gènes particuliers constituant le génotype fondamental commun à toutes les classes, et des gènes constituants le GH et le GS de chaque classe de la population.

3) - ClasseGtype

Toute classe détient un ensemble d'attributs et de méthodes constituant ses gènes. Toute classe est décrite par trois blocs de gènes distincts : le *génotype fondamental*, le *génotype hérité* et le *génotype spécifique*.

La classe ClasseGtype représente toute classe décrite dans notre modèle. Elle est caractérisée par un nom. L'attribut GF de ClasseGtype est une référence vers une instance de Population. Ainsi, le GF est défini dans chaque classe et est hérité par ses éventuelles sous-classes. Ceci permet également d'assurer l'appartenance de ces sous-classes à la même population.

Le GH définit les attributs et méthodes hérités qui peuvent être redéfinis ou surchargés. Le GS définit les attributs et méthodes définis localement. Le GH de toute classe est en fait le GS et le GH de sa super-classe (Figure 3-2), éventuellement redéfinis ou surchargés.

4) - Attributs

Dans un état stable, tous les attributs sont des composants de classes et font partie intégrante des patrimoines génétiques existants dans le modèle. Un attribut peut faire partie des structures de différentes classes, voire de différentes populations.

Durant un processus d'évolution, certains attributs peuvent ne pas appartenir pleinement à un patrimoine génétique donné. Ces attributs sont ceux nouvellement introduits par un objet ayant évolué. Tant que le processus n'a pas abouti, ces nouveaux attributs ne sont référencés par aucune classe ni par aucun patrimoine génétique. Ils ont un statut particulier, le statut *transitoire*, contrairement aux attributs participant aux spécifications de classes, et par conséquent de patrimoines génétiques, ont un statut *permanent*. C'est à la fin de la phase d'exploitation que leur statut peut changer. Si le concepteur décide, au vu des résultats de l'exploitation, de garder des attributs de ce type, ils intègrent alors définitivement le modèle. Leur statut passe alors de transitoire vers permanent. Le patrimoine génétique de la population à laquelle appartient la classe est mis à jour pour l'intégrer. Les éventuelles sous-classes, de la classe détenant ces nouveaux attributs, sont également modifiées pour le prendre en compte dans leur GH (confère processus de développement-chapitre 5). Par contre si le concepteur décide de ne pas les intégrer, ils sont alors rejetés et donc détruits sans incidence aucune sur le modèle existant.

L'intégration d'un nouvel attribut au sein d'une classe influence également les schèmes permanents concernés. En effet, cela implique automatiquement la mise à jour de tous les schèmes concernés de la population : le schème de la classe concernée aura la valeur à 1 ou # selon le cas ; les schèmes de ses sous-classes également, sauf restriction explicite du concepteur.

5) - Methode

Dans un état stable, toutes les méthodes sont des composants de classes et font partie intégrante des patrimoines génétiques existants dans le modèle. Une méthode peut faire partie des spécifications de différentes classes, voire de différentes populations.

6) - Scheme

Un schème est constitué d'un ensemble de couplets (*Attribut* ; *ValeurGène*), *Attribut* est la référence de l'attribut représenté et *ValeurGène* est la valeur de ce gène dans le schème. Chacune de ces valeurs est prise dans l'ensemble {0 , 1 , #} pour exprimer respectivement l'absence, la présence ou l'indifférence de l'attribut référencé.

Un schème (donc une instance de la classe Scheme) peut être permanent ou temporaire. Un schème permanent est complètement dépendant de la classe à laquelle il est associé. Pour un schème permanent, les attributs référencés sont ceux de sa classe.

Il est à rappeler que la valuation des attributs du GF au niveau des schèmes sont identiques pour tous les schèmes permanents de la même population.

7) - InstanceClasseGtype

Cette classe est représentée dans le modèle pour exprimer le fait que les instances de la classe ClasseGtype soient également des classes, même si elles sont représentées dans un premier temps par des instances (cet aspect ne fait pas partie des objectifs de ce chapitre. Nous proposons au lecteur de se référer au chapitre implémentation pour de plus amples éclaircissements).

3.7.1.4. Définition des différents liens

Les liens entre concepts qui sont porteurs d'une sémantique particulière pour la compréhension du modèle sont définis ci-après. Chaque lien est défini par son nom et les deux classes qu'il relie :

1) - 'faitPartie' entre classe ClasseGtype et classe Population

'*faitpartie*' est une association entre ClasseGtype et Population. Ce lien est en fait véhiculé par l'attribut GF de ClasseGtype, et qui réfère à une instance de Population qui définit le GF. Ce lien est explicitement nommé pour exprimer cette appartenance d'une classe à une population. Cette appartenance peut être directe si une classe hérite directement de sa population ou indirecte si elle hérite d'une autre classe de la même population.

2) - 'son PG' entre classe Population et classe PatrimoineGenetique

La relation '*sonPG*' est une association entre Population et PatrimoineGenetique. Il exprime qu'une population a un et un seul patrimoine génétique, et qu'un patrimoine génétique revient à une et une seule population.

3) - 'AttributsGF' entre classe Population et classe Attributs

'*AttributsGF*' est un lien de composition. Une population définit un GF. Un GF est constitué d'un ensemble d'attributs.

4) - 'MethodesGF' entre classe Population et classe Methode

'*MethodesGF*' est également un lien de composition. Une population définit un GF. Un GF est constitué également d'un ensemble de méthodes définies sur les attributs constituants le GF.

5) - 'is-a' entre classe ClasseGtype

Le lien '*is-a*' est une association réflexive sur la classe ClasseGtype. Elle représente le lien d'héritage qui peut exister entre une classe et sa super-classe. Les rôles expriment un peu plus la sémantique du lien d'héritage :

- *sous-classe :* cardinalité : 0..1. Une classe peut au plus être sous-classe d'une autre classe.
- *super-classe :* cardinalités : 0..*. Une classe peut avoir aucune ou plusieurs sous-classes.

6) - 'MethodesGH' entre classe ClasseGtype et classe Methode

Le lien '*MethodesGH*' est un lien de composition. Une classe a son GH composé de méthodes définies sur les attributs constituants le GH de la classe. Ces méthodes sont celles constituant les méthodes du GS et celles du GH de la super-classe. Une méthode peut participer plusieurs fois à la constitution de GH différents.

7) - 'MethodesGS' entre classe ClasseGtype et classe Methode

Le lien '*MethodesGS*' est un lien de composition. Une classe a son GS composé de méthodes définies sur les attributs constituants le GS de la classe. Ces méthodes sont définies localement au sein de la classe. Une méthode peut participer plusieurs fois à la constitution de GS différents.

8) - 'AttributsGH' entre classe ClasseGtype et classe Attributs

Le lien '*AttributsGH*' est un lien de composition. Une classe a son GH composé d'attributs du GS et d'attributs du GH de sa super-classe. Un attribut peut participer plusieurs fois à la constitution de GH différents.

9) - 'AttributsGS' entre classe ClasseGtype et classe Attributs

Le lien '*AttributsGS*' est un lien de composition. Une classe a son GS composé d'attributs définis par la classe. Un attribut peut participer plusieurs fois à la constitution de GS différents.

10) - 'aPourSchemePermanent' entre classe ClasseGtype et classe Scheme

Le lien '*aPourSchemePermanent*' est un lien de composition. Une classe est également constituée d'un schème permanent qui permet de compléter sa structure et, par la suite, celle de ses instances. Une classe n'a qu'un seul schème permanent et un schème permanent ne peut être associé qu'à une seule classe.

11) - 'estAssocieA' entre classe ClasseGtype

L'association '*estAssocieA*' est une association réflexive définie sur ClasseGtype. Elle permet d'exprimer une association que peut avoir une classe avec une autre classe.

12) - 'lienEvolution' entre classe ClasseGtype

Ce lien est une association porteuse de propriétés, et elle est représentée par une classe nommée '*evolueVers*', ce qui permet d'exprimer le lien d'évolution. Une instance peut évoluer d'une classe vers une autre en suivant un lien d'évolution. Pour exprimer cette sémantique, le lien '*lienEvolution*' définit deux rôles :

- *source :* représente quelle classe est la source du lien d'évolution. Une classe peut ne pas être une classe source d'un lien d'évolution ou peut l'être pour différents liens d'évolutions ;
- *cible :* représente quelle classe est la cible du lien d'évolution. Une classe peut ne pas être une classe cible d'un lien d'évolution ou peut l'être pour différents liens d'évolutions.

Cette association est navigable, puisque orientée vers la classe cible. Cette association est instanciée lorsqu'un objet de la classe (classe source) a eu à évoluer, comme résultat du processus d'évolution, et à être rattaché à une nouvelle classe ou une classe existante (classe cible).

La classe '*evolueVers*' a pour attributs :

- *attributsPerdus :* l'ensemble des références des attributs perdus par une instance de la classe source ;
- *methodesPerdus :* l'ensemble des références des méthodes perdues, c'est-à-dire celles ayant ces attributs présents dans la signature et perdus par une instance (c'est-à-dire qui ne se retrouveront pas dans la classe cible), et par conséquent l'ensemble des méthodes auxquelles elle ne réagira plus ;
- *attributsAcquis :* l'ensemble des références des attributs de la classe cible à acquérir par l'instance qui évolue vers cette classe cible ;
- *methodesAcquises :* l'ensemble des références des méthodes acquises, c'est-à-dire celles ayant ces attributs présents dans la signature et acquis par l'instance (c'est-à-dire qui se retrouveront dans la classe cible), et par conséquent l'ensemble des méthodes auxquelles elle devra réagir.

3.8 - Conclusion

Ce chapitre est le premier des trois chapitres qui présentent le modèle GENOME : ses concepts. Ses processus d'évolution et son modèle de métriques pour le contrôle de l'émergence sont définis dans les chapitres suivants. Ils sont définis pour permettre d'appréhender l'évolution d'un modèle à objet à partir de l'évolution structurelle dynamique d'instances. Ces changements sont exprimés directement sur les instances par le concepteur. Ces processus sont itératifs et auto-adaptatifs : lorsqu'une classe évolue et provoque des impacts, il s'agit d'un processus de *développement* (section 3.3.1) ; lorsqu'une instance évolue dans sa structure et peut influencer les classes, il s'agit d'un processus d'*émergence* (section 3.3.2). Cela fait l'objet des deux prochains chapitres.

Les concepts de GENOME font donc l'objet du présent chapitre. Les concepts sont définis de manière à pouvoir manipuler aussi bien les classes et leurs spécifications et liens, que les instances. Pour cela les concepts sont tous réifiés.

Pour les mêmes raisons, nous définissons pour le modèle des concepts de base et des concepts avancés. Les concepts de base (section 3.4 -) sont plus une définition des concepts objet de base sous l'optique de l'Évolution Artificielle : une classe-GTYPE (section 3.4.2) détient le génotype de ses instances alors qu'une instance-PTYPE (section 3.4.3) représente le phénotype de sa classe, c'est-à-dire sa réalisation dans un environnement donné. Un gène (section 3.4.1) est la partie d'information que sont l'attribut et la méthode. Les classes qui déclinent les différentes possibilités d'une abstraction complexe du monde réel constituent une population (section 3.4.4). L'ensemble des gènes définis par ses classes constitue le patrimoine génétique (section 3.4.4) de la population.

Les concepts avancés (section 3.5 -) sont spécifiques à GENOME. Une classe est définie au travers de trois groupes de gènes : le GF (section 3.5.1) qui défini l'essence de toute population et qui est immuable pour toutes ses classes, le GH (section 3.5.2) qui définit l'héritage des ancêtres (les super-classes directe et indirectes) et qui peut être redéfini et enfin le GS (section 3.5.3) qui définit les spécificités de chaque classe. Le schème (section 3.5.4) quant à lui est une entité qui transcrit la structure de classes : c'est le schème permanent (section 3.5.4.1), ou la structure d'une instance : c'est le schème temporaire (section 3.5.4.2). Il est décrit de façon simple au travers de trois valeurs possibles : 1, 0 ou # (section 3.5.4.3). L'indéniable avantage du schème est de considérer de la même manière classes et instances dans des processus d'évolution complètement automatisés, et de pouvoir faire du calcul symbolique.

Tous ces concepts et toutes ces notions sont représentés dans le méta-modèle de GENOME (Section 3.7 -).

Chapitre 4 : GENOME : LES PROCESSUS D'EVOLUTION

4.1 - Introduction

Après avoir abordé et défini les concepts du modèle ainsi que son contexte, nous abordons dans le présent chapitre les *processus d'évolution* du modèle GENOME.

Les processus d'évolution sont définis dans le cadre de l'émergence et du développement. Comment ?

4.1.1. L'évolution gérée au travers de processus d'évolution

Un processus d'évolution est généralement déclenché pour faire face et répondre à une situation de changement. Un processus d'évolution est le reflet opératoire de la politique d'évolution d'une stratégie donnée.

Dans le cadre de notre travail, et toujours au vu de nos objectifs, nous nous intéressons à la gestion de l'évolution lorsqu'une ou plusieurs instances sont modifiées dans leur structure. Nous définissons des processus d'évolution qui sont déclenchés lorsqu'un changement, prévu ou non, survient sur des instances.

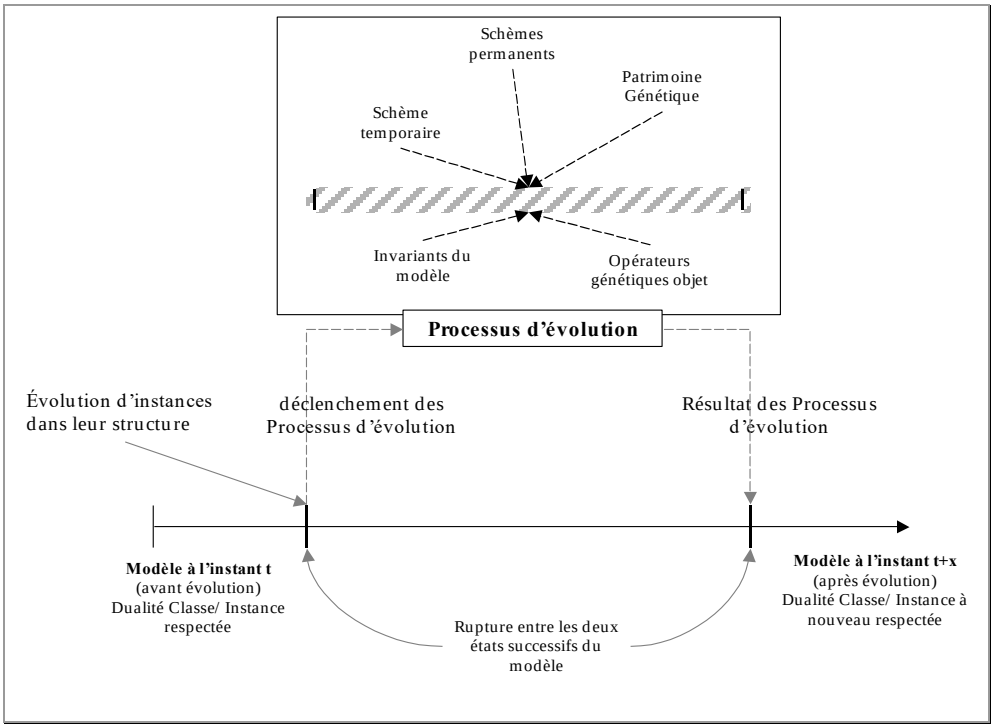


Figure 4-1: les processus d'évolution entre deux états stables du modèle.

Le processus d'évolution doit être en mesure de détecter ce changement ainsi que son type, de retrouver les entités impliquées dans l'évolution et de répercuter ce changement de manière adéquate sur les classes existantes.

4.1.2. Formalisme utilisé

Tout au long de ce chapitre, nous présentons les différentes étapes d'un processus d'évolution, après l'avoir défini et avoir expliqué son utilité. Nous optons pour une représentation graphique des différentes étapes de chaque processus afin d'en donner une vision plus claire. Cette représentation est accompagnée d'explications. Le formalisme graphique que nous choisissons d'utiliser est celui des diagrammes d'états-transitions d'UML [Muller 97] pour spécifier les différentes étapes de chaque phase. Le formalisme est décrit en Figure 4-2 :

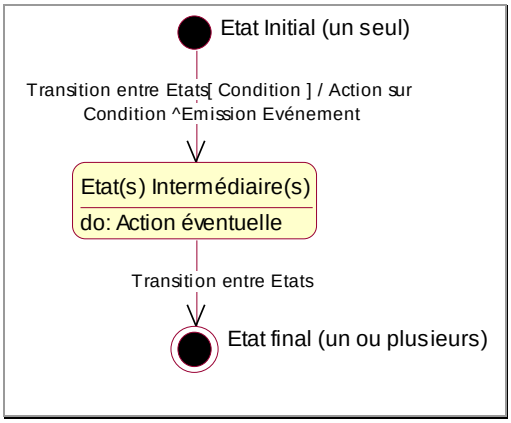


Figure 4-2. Formalisme du diagramme Etats-Transitions d'UML

Ce diagramme s'inspire des statecharts [Harel 87, Harel 96]. Il est adapté à nos besoins car il permet de représenter :

– les différents états par lesquels peut passer une entité. Il existe trois différents types d'états, chacun ayant un formalisme graphique précis :

- un seul état initial
- un ou plusieurs états intermédiaires avec des actions
- un ou plusieurs états finaux

– les transitions entre deux états. Une transition est représentée par une flèche entre deux états successifs. Cette flèche est orientée de l'état précédent vers l'état suivant. Outre le nom de la transition, le formalisme offre également la possibilité d'exprimer des conditions sur l'exécution de ces transitions ainsi que les actions menées et les événements émis :

- `[condition]`
- `/Action sur Condition`
- `^Émission Événements`

4.1.3. Niveau d'abstraction considéré

Les diagrammes d'états-transitions sont définis pour être utilisés avec des classes. En effet, selon UML, un diagramme d'états-transitions, dit également *automate*, est attaché à une classe pour décrire les différents états, ainsi que leur séquence, par lesquels passent les objets de cette classe. Concernant GENOME, nous avons besoin de décrire les différents états par lequel passe un modèle donné, donc un schéma de classes. Aussi, les automates que nous présentons ne décrivent pas les états des objets *d'une* classe donnée mais les états des objets *des* classes définies au sein d'un modèle de classes. Les processus d'évolution sont donc représentés par des automates et peuvent impliquer toutes les entités (classe, instance, lien, attribut) présentes dans le modèle de classes.

4.2 - Les trois phases d'un processus d'évolution : définition

Nous considérons que tout processus d'évolution d'une instance s'exécute en trois phases :

- une phase d'*extraction ;*
- une phase d'*exploration ;*
- une phase d'*exploitation*.

4.2.1. Présentation

Chacune de ces trois phases est déclenchée par un événement particulier et nouveau. Ces trois phases se succèdent. Comment est-ce que le début et la fin d'une phase sont matérialisés ?

Le diagramme d'états-transitions suivant (Figure 4-3) met en évidence l'enchaînement des trois phases :

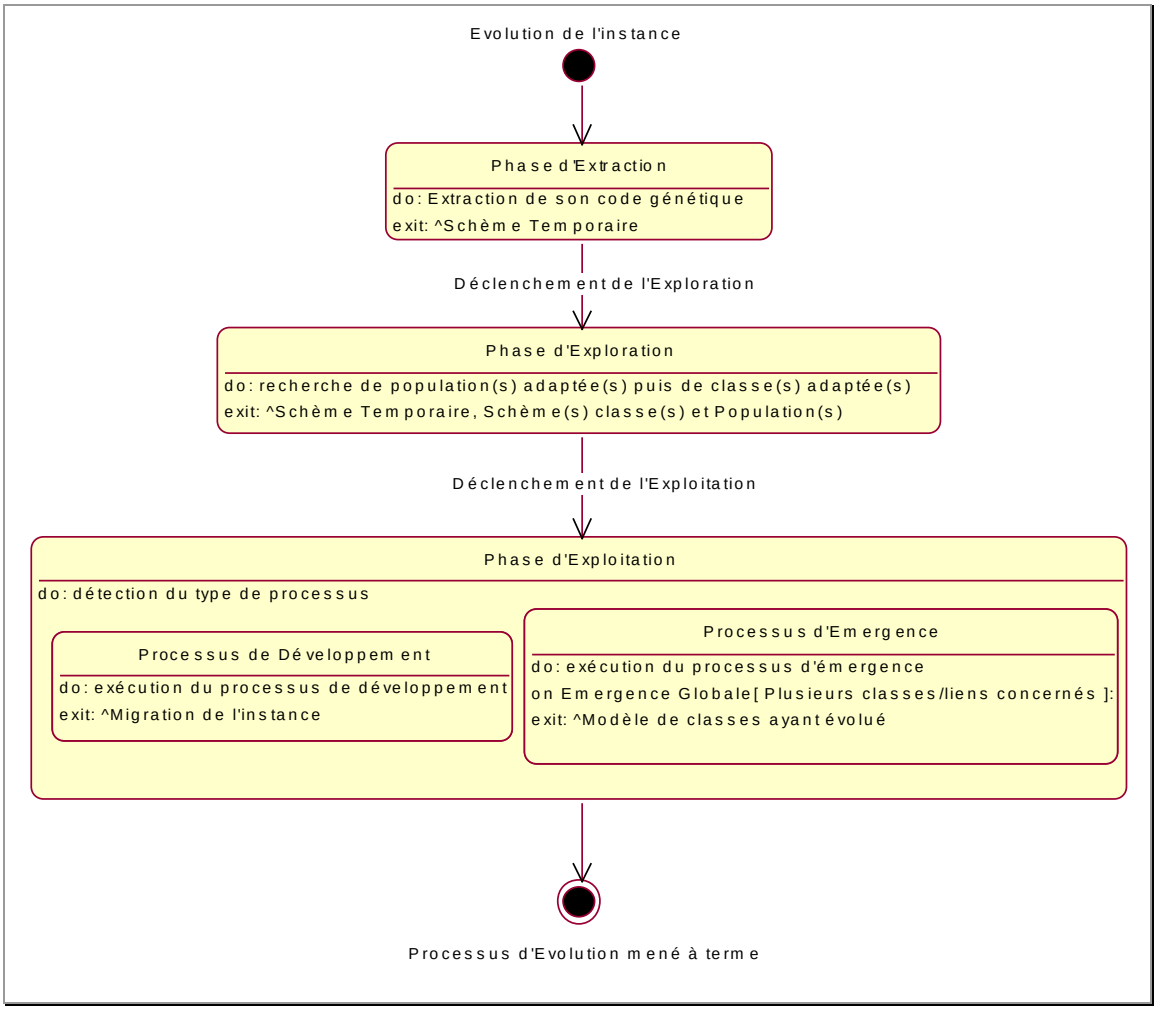


Figure 4-3: Les trois phases d'un processus d'évolution

A ce stade de description, le diagramme est volontairement peu détaillé. L'objectif est de mettre en évidence ce que fait globalement chaque phase, préfixé par un '`do :`', et ce qui ressort de l'exécution de chacune, préfixé par un '`exit :`'.

D'une manière globale :

– L'évènement déclencheur de la phase d'extraction est l'évolution d'un objet dans sa structure initiale. L'événement en sortie est le schème temporaire associé ;

– L'évènement déclencheur de la phase d'exploration est la spécification du schème temporaire. L'événement en sortie est l'ensemble des classes sélectionnées ;

– L'évènement déclencheur de la phase d'extraction est l'ensemble des classes sélectionnées. L'évènement en sortie est l'achèvement de l'évolution.

Nous définissons de manière plus précise chaque phase dans ce qui suit, avant de définir leur automate respectif dans la section 4.4 - .

4.2.2. Phase d'extraction

La phase d'*extraction* survient dès que l'évolution d'un objet (voire plusieurs objets) dans sa structure est effectuée et détectée. Étant dans un cadre de simulation de l'évolution, nous laissons le soin au concepteur de choisir de déclencher lui-même le processus d'évolution. S'il ne le souhaite pas, le déclenchement sera automatique dès qu'une instance est modifiée dans sa structure. L'objectif de cette phase est d'extraire le code génétique de cet ou ces objets. Le code génétique de tout objet ayant évolué est extrais au sein d'un schème temporaire (défini au chapitre 4). Il a la même structure que l'objet ayant évolué car il spécifie la présence ou non d'un attribut dans la structure de l'instance. En effet, si un attribut du patrimoine génétique de la population est présent dans sa structure, il est référencé par le schème. La façon dont il est valué est alors spécifiée : à 1 (donc obligatoirement présent), à 0 (donc obligatoirement absent) ou à # (donc optionnel).

Ces schèmes temporaires (un par instance ayant évolué) sont les entités sur lesquelles s'appliquent les deux autres phases du processus d'évolution. Son expression est simple. Il peut donc être aisément manipulé de façon automatisée, ce qui est beaucoup moins évident si l'on travaillait directement sur les instances.

4.2.3. Phase d'exploration

La phase d'*exploration* se déclenche lorsque les schèmes temporaires sont spécifiés. Comme son nom l'indique, l'objectif de cette phase est d'explorer l'ensemble des structures conceptuelles présentes dans le modèle, dans le but de trouver celles qui sont complètement ou partiellement adaptées à l'instance ayant évolué. En effet, les différentes structures conceptuelles constituent un espace de solutions potentielles en terme d'adéquation entre les schèmes temporaires et les schèmes permanents (en d'autres termes, l'adéquation entre une instance et une classe).

L'exploration s'exécute sur les schèmes temporaires et les schèmes permanents. Elle se fait en deux principales étapes :

– La première étape consiste à explorer l'ensemble des populations existantes pour détecter la population (voire les populations dans le cas d'objets complexes) à laquelle l'instance peut potentiellement appartenir ;

– La seconde étape consiste à explorer l'ensemble des classes définies au sein de la population sélectionnée afin de déterminer les classes qui détiennent une partie ou la totalité des gènes de l'instance.

4.2.4. Phase d'exploitation

La phase d'*exploitation* se déclenche lorsque la population et les classes plus ou moins adaptées à l'instance sont sélectionnées. L'objectif de cette phase est d'exploiter les différentes entités : les classes sélectionnées, représentées par les schèmes permanents,

et les instances ayant évoluées, représentées par les schèmes temporaires ainsi que les différentes autres données présentes dans le modèle, telles que par exemples des contraintes. Le but ultime de l'exploitation est de tenter de tirer le meilleur parti possible des toutes les informations présentes dans le modèle et celles introduites par les instances.

L'exploitation permet de mettre en évidence la façon la plus appropriée de faire évoluer le modèle au vu des spécifications de classes et d'instances. L'exploitation peut mener à terme l'un des deux principaux processus d'évolution, le *développement* ou l'*émergence* :

4.2.4.1. Processus de développement

Un processus de développement est tout processus d'évolution de classes aussi bien en intention qu'en extension :

– *L'évolution en intention :* cela consiste à faire évoluer les propriétés (attributs et méthodes) au sein d'une classe. Nous incluons également dans l'évolution en intention, les impacts de l'évolution des spécifications d'une classe vers les autres classes concernées par le lien d'héritage, en l'occurrence les sous-classes, ou par les liens de composition, ainsi que les différents impacts sur les classes d'une hiérarchie lorsque cette dernière est réorganisée ;

– *L'évolution en extension :* cela consiste à faire évoluer les instances d'une classe. Il existe différentes possibilités :

- Dans le cadre de l'évolution en intention d'une classe, plus précisément des attributs, dits également variables d'instances, les instances d'une classe peuvent nécessiter d'être adaptées au nouvel état de leur classe. L'évolution en extension de la classe vient comme conséquence de son évolution en intention ;
- Dans le cadre de l'évolution des valeurs des instances dans leurs variables d'instances définies dans leur classe ;
- Dans le cadre de l'évolution d'une instance à partir d'une classe vers une autre (migration d'instances). Dans ce cas, il s'agit d'évolution en extension des deux classes source et cible de la migration.

4.2.4.2. Processus d'émergence

Un processus d'émergence est tout processus d'évolution d'instances avec des impacts sur les spécifications de classes. Il concerne toute émergence d'une nouvelle structure conceptuelle à partir d'instances. Il y a deux issues possibles à cette évolution :

– *Émergence locale :* concerne l'émergence de nouvelles informations au sein de classe(s) existante(s). Le code génétique de l'objet a été modifié et peut faire modifier celui de sa classe ou celui d'une classe sémantiquement voisine.

– *Émergence globale :* concerne l'émergence d'une nouvelle structure conceptuelle (donc de classe(s)) soit par projection directe à partir de l'objet ayant évolué, soit par croisement de structures existantes.

Avant d'expliquer de manière plus détaillée le fonctionnement de chaque phase au sein d'automates (section 4.4 -), nous définissons et présentons les opérateurs de base qui sont nécessaires à l'exécution de ces différentes phases dans la section suivante.

4.3 - Les Opérateurs Génétiques Objets

Nous définissons les opérateurs de GENOME qui s'appliquent et manipulent les concepts du modèle (population, classe-GTYPE, instance-PTYPE, schème permanent, schème temporaire, gène et patrimoine génétique). Ces opérateurs sont utilisés lors des trois phases d'évolution.

Les opérateurs utilisés sont ceux de *sélection*, de *reproduction*, de *croisement* et de *mutation*. Ces opérateurs s'appliquent différemment. Nous faisons cette distinction selon leur action sur l'entité ou les entités concernées.

4.3.1. Introduction

L'idée principale des opérateurs que nous définissons est de pouvoir manipuler indifféremment les spécifications de classes et les spécifications d'instances. Cette indifférence est rendue possible grâce au concept de schème. Les opérateurs s'exécutent en effet uniquement sur les schèmes permanents et/ou temporaires, donc sur les gènes ; ils diffèrent selon leur application : sur un schème, sur deux schèmes ou sur un gène.

Les opérateurs que nous proposons sont inspirés des opérateurs des Algorithmes Génétiques. De la même manière que nous avons procédé pour les concepts, certains opérateurs sont une interprétation d'actions sous une transcription Évolution Artificielle, alors que d'autres sont nouvellement introduits (tels que la *sélection* et principalement le *croisement*).

Leur exécution est ainsi aisément automatisée durant tout le processus d'évolution.

Nous rappelons que les processus d'évolution sont définis à des fins de simulation d'évolution. Les entrées sont les besoins qu'exprime le concepteur et les résultats en réponse à ces besoins sont donnés au concepteur. Ce dernier aura alors la charge des choix conceptuels définitifs.

4.3.2. Sélection

De manière générale, la *sélection* nécessite de travailler sur l'adaptation d'une entité à un problème ou une application donnés. Dans GENOME, l'opérateur de sélection est défini pour déterminer, après l'évolution structurelle d'une instance, quelles sont les classes qui détiennent une partie ou la totalité de ses gènes. Cet opérateur est inspiré des Algorithmes Génétiques. L'opérateur de sélection que nous proposons revient donc à choisir des classes en comparant la structure de la classe avec celle de l'instance, au travers des schèmes permanents et du schème temporaire. Cette sélection se base sur leur adaptation par rapport à un critère d'adéquation partielle ou totale de deux schèmes. Cette adéquation se calcule au travers de la *valeur d'adaptation* (confère section 4.3.6), grâce à une *fonction d'adaptation*.

4.3.3. Reproduction

La *reproduction* revient à la création d'un individu par rapport à un code génétique. Au vu des concepts de classe-GTYPE et instance-PTYPE et de leur définition, nous considérons que dans un modèle à objet, la reproduction revient au processus d'instanciation. L'instance a donc le code génétique de sa classe qu'elle value. Cette valuation est différente d'une instance à une autre et donne ainsi des individus différents constitutifs du PTYPE de la classe.

4.3.4. Croisement

L'opérateur de *croisement* est défini dans les Algorithmes Génétiques (chapitre 2). Il reprend le principe du croisement de codes génétiques en biologie. Le croisement s'applique sur deux schèmes. Il permet d'interchanger des portions de codes génétiques. Le résultat du croisement est défini dans deux nouveaux schèmes, généralement appelés *enfants*. Les deux schèmes ayant ainsi été croisé sont généralement appelés *parents*. Les schèmes enfants ont la même longueur de code génétique que leurs parents.

Dans le cadre de GENOME, nous définissons l'opérateur de croisement pour permettre à deux schèmes d'interchanger leur code génétique. Le croisement s'applique sur les schèmes permanents de l'ensemble des classes sélectionnées lors de la phase d'exploration, et qui sont partiellement adaptées à une instance ayant évolué. Le principal objectif est de tenter de parvenir, par cet interchangement de gènes, à constituer d'autres structures conceptuelles plus adaptées aux instances ayant évolué, donc plus adaptées au nouveau contexte. Ces structures conceptuelles résultantes sont également représentées sous forme de schèmes.

Le but recherché n'est pas d'arriver en définitive à un schème représentant une instance, mais de dégager un ou plusieurs schèmes, dits *émergents*, pouvant englober l'instance, et par conséquent répondant aux nouveaux besoins. Ces schèmes émergents

seront des structures nouvelles plus riches que celles portées par l'instance. Ils permettent de faire apparaître et de mettre en évidence des structures conceptuelles nouvelles qui étaient peut être noyées dans les structures existantes ou qui étaient peut être complètement méconnues.

Le détail du fonctionnement de l'opérateur de croisement est défini dans la section 4.5.2.2 au travers de l'automate qui lui est associé.

4.3.5. Mutation

Nous considérons comme mutation toute apparition d'une nouvelle information, au sein d'un objet ou d'une classe, auparavant inexistante, et qui provoque le changement du patrimoine génétique de la population.

L'opérateur de *mutation* est donc défini comme toute action pouvant modifier les caractéristiques des instances et des classes. La mutation revient à activer ou à désactiver la présence d'un gène au sein d'un schème, c'est-à-dire mettre à 1 ou à 0, voire même à #, un gène dans un schème. L'opérateur de croisement seul ne peut pas toujours empêcher la perte d'informations prometteuses pour l'évolution. L'opérateur de mutation est utilisé pour introduire ou modifier des gènes nécessaires lors de processus d'évolution, voire même lorsque la sélection et le croisement ne donnent pas de résultats intéressants. L'objectif est alors double :

- cerner une autre partie du modèle pour effectuer des recherches ;
- tenter de restaurer éventuellement des structures génétiques perdues ou nécessaires.

Cet opérateur n'est pas systématiquement utilisé et ne s'applique pas de manière automatique. Son application est ponctuelle et est généralement déclenchée par le concepteur pour modifier spécifiquement les gènes qu'il juge bon de faire muter.

4.3.6. Valeur d'Adaptation

La *valeur d'adaptation Va* permet de calculer l'adaptation de classes à un objet. Le calcul de cette valeur se fait grâce à une fonction dite *fonction d'adaptation* et ayant comme paramètres d'entrée deux schèmes de même longueur. Le premier schème représente l'instance ayant évoluée et le deuxième schème en paramètre représente soit un schème permanent (donc une classe de la population sélectionnée), soit un schème intermédiaire. En notant :

- le schème de l'objet ayant évolué Sch_{obj}
- et le schème en paramètre par Sch_{param},

nous définissons la fonction d'adaptation utilisant l'opérateur $\wedge$ (ou_logique dont la table de vérité est le Tableau 4-1) comme suit :

$$Va\ (Sch_{param}) = \sum_{(i = 1 \rightarrow n)} \{ Sch_{obj}[i] \wedge Sch_{param}[i] \} / n$$

Où :

- n est le nombre de gènes spécifiés dans l'objet ayant évolué ;
- i est l'indice variant de 1 à n définissant à chaque passage la position des deux gènes respectifs des schèmes analysés.

Le principe est qu'à chaque fois que deux gènes sont identiques, la valeur d'adaptation de Sch_{param}, initialement à zéro, augmente de 1. Si deux gènes sont contradictoires, la valeur n'augmente pas. A la fin, la valeur d'adaptation est divisée par le nombre de gènes spécifiés, c'est-à-dire valués à 1, dans le schème de l'objet ayant évolué. Elle est ainsi toujours comprise dans l'intervalle [0, 1]. Lorsqu'un schème Sch_{param} a une valeur d'adaptation égale à 1, cela implique qu'il est complètement adapté à Sch_{obj}. Il est alors dit *Schème Émergent*.

Les gènes spécifiés sont tous les gènes ayant leur présence obligatoire, donc valués à 1. Les gènes obligatoirement absents ne sont pas pris en compte, et les gènes optionnels dans Sch_{obj} ne sont pas déterminants dans les choix. Ils ne sont par conséquent pas considérés, contrairement aux gènes optionnels dans Sch_{param}. Les différentes combinaisons des valeurs pour un gène à la position i dans chacun des deux schèmes, et leur interprétation sont récapitulées dans la table suivante :

$Sch_{param}[i]$ \ $Sch_{obj}[i]$	0	1	#
0	0	0	0
1	0	1	0
#	0	1	0

Tableau 4-1: table de vérité de l'opérateur ∧.

Pour que deux schèmes puissent être comparés en terme d'adaptation, ils doivent avoir la même structure. En effet, étant donné que le but est de calculer si un schème Sch_1 est adapté et dans quelle mesure il l'est à un autre schème Sch_2, il faut pouvoir comparer si un gène est spécifié ou non dans les deux schèmes. Pour cela, une première phase d'uniformisation des deux schèmes est effectuée par comparaison deux à deux des gènes. Cette uniformisation se base sur la concordance de la position du même gène à la même position dans les deux schèmes (indépendamment de la façon dont il est évalué). Nous imposons qu'un gène soit à la même position lorsque deux schèmes sont comparés.

Par exemple, les deux schèmes Sch_2 et Sch_1 suivants sont à comparer. On désire connaître l'adaptation de Sch_2 à Sch_1. Sch_1 est constitué des gènes (attributs) a_1, a_4, a_7, a_9. Sch_2 est constitué des gènes (attributs) a_1, a_7, a_{10}.

	a_1	a_4	a_7	a_9
Sch_1 :	1	#	1	1

	a_1	a_7	a_{10}
Sch_2 :	#	1	1

Tableau 4-2: deux schèmes à comparer.

Pour que ces deux schèmes puissent être comparés et donc avoir la valeur d'adaptation de Sch_2 à Sch_1, ils doivent avoir la même structure. Les deux schèmes sont donc reconstruits au vu de cette contrainte, sans pour autant perdre une information déjà présente ni gagner une information déjà absente. Ils deviennent :

	a_1	a_4	a_7	a_9	a_{10}
Sch_1 :	1	#	1	1	0

Sch_2 :	#	0	1	0	1

Tableau 4-3: les deux schèmes uniformisés.

Le '0' permet de dire '*attention ! Ce gène n'est pas présent dans le schème !*'. La valeur d'adaptation de Sch_2 est : $Va(Sch_2)= 2/3$. L'interprétation de cette valeur est :

- 2 : représente le nombre de gènes présent dans Sch_1 et présent également dans Sch_2. Il s'agit des gènes a_1 et a_7.
- 3 : représente le nombre de gènes réellement spécifiés dans Sch_1. Il s'agit des gènes a_1, a_7 et a_9.

Il est également important de noter qu'au vu de l'importance du GF, la comparaison de ce dernier diffère de celles des GH et GS présents dans les schèmes permanents. En effet, pour que deux entités puissent être comparées, il faut qu'il y ait une concordance entre leur GF respectif. Donc deux schèmes voient d'abord leurs GF comparés et l'adaptation doit être totale. Les autres gènes sont ensuite comparés grâce à la fonction d'adaptation présentée ci-dessus.

4.3.7. Distance Sémantique

La *distance sémantique d.s* permet de calculer l'écart qu'il y a entre un schème émergent et les schèmes permanents sélectionnés. La d.s est utile pour trouver l'endroit où insérer la structure conceptuelle véhiculée par le schème émergent au sein d'une hiérarchie de classes d'une population. La distance sémantique est calculée avec la même fonction d'adaptation que celle utilisée pour calculer la valeur d'adaptation.

Plus un schème permanent est proche du schème émergent, plus la classe qui lui est associée a des chances de devenir une super-classe de la structure conceptuelle véhiculée par le schème émergent.

Nous définissons la distance sémantique comme une première mesure structurell simple. Elle peut être évaluée rapidement et automatiquement. Elle présente également l'avantage de s'appliquer directement sur la structure des entités comparées. Nous la considérons comme une mesure de base nécessaire aux processus d'évolution.

Nous abordons maintenant la classification et l'utilisation des opérateurs :

4.3.8. Classification et utilisation des opérateurs

4.3.8.1. Classification des opérateurs

Nous classifions les opérateurs selon leur action :

Action \ Produit / Opérateur		Schème	Gène
Pas de modification	Sélection	S'appuie sur l'adaptation d'un schème à un autre par calcul de la Va	-
	Reproduction	Agit sur une classe pour en créer une instance	-
Modification	Croisement	Agit sur deux schèmes pour les croiser et transmettre leurs gènes dans deux schèmes enfants	Transmission des gènes
	Mutation	Agit sur un gène pour changer sa valuation	Modifie un gène

Tableau 4-4 : Classification des opérateurs objets

Ce tableau met en évidence l'action de chaque opérateur selon qu'il modifie ou non l'entité concernée. Il apparaît que la sélection et la reproduction ne modifient pas le code génétique, alors que le croisement et la mutation le font. Le croisement modifie la population de croisement en créant deux nouveaux schèmes à partir de deux schèmes parents, qu'ils remplacent dans la population. La mutation agit sur un gène pour permettre d'acquérir (1) ou de perdre (0) un gène spécifique. Cela modifie donc la structure du schème contenant ce gène.

4.3.8.2. Utilisation des opérateurs

Les opérateurs définis sont utilisés durant les processus d'évolution. L'illustration de leur utilisation respective est abordée dans la section 4.4 - qui présente les automates associés à chaque phase ainsi que les différentes étapes, le tout ponctué par des illustrations de leur déroulement sur des exemples simples. Avant cela, et afin d'en donner un aperçu général, nous situons dans le tableau suivant l'utilisation des opérateurs pour chaque phase :

Phase / Opérateur	Extraction	Exploration	Exploitation	
			Développement	Émergence
Sélection		x	x	x
Reproduction			x	x
Croisement				x
Mutation			x	x
Valeur d'Adaptation		x	x	x
Distance Sémantique			x	x

Tableau 4-5: Utilisation des opérateurs durant les phases des processus d'évolution.

Les opérateurs ne sont pas tous utilisés en même temps dans une même phase, mais au fur et à mesure que leur besoin est ressenti.

4.4 - Les trois phases de tout processus d'évolution : Les automates

Nous présentons en premier l'exemple qui est utilisé à titre illustratif tout au long des processus d'évolution. Il permet de mettre en exergue le déroulement des étapes de chacune des trois phases. Ensuite, chaque phase est présentée au travers de ce qu'elle prend en entrée, de ce qu'elle produit en sortie, ainsi que son automate qui montre les différentes étapes. Les automates sont représentés grâce au formalisme des états-transitions présenté en section 4.1.2.

4.4.1. L'exemple

Considérons les instances suivantes, de la population Membre-Université déjà introduite dans le chapitre 3, dans leur état initial :

	Patrimoine Génétique	Instances de Eleve-Chercheur			Instances d'Ater	
		O_1	O_2	O_3	O_4	O_5
GF	*Num-Identification*	#3	#8	#4	#7	#1
	Nom	N1	N2	N3	N4	N5
	Prénom	P1	P2	P3	P4	P5
	Thème-Recherche	Auto	Math	Info	Contraintes	EIAO
	Laboratoire	L1	L2	L3	L4	L5
	Nbr-Publications		1	4	3	
	Spécialité					
	Modules				RO	GL
	Durée	4	3	2	1	1
	Date-début					
	Nbr-heures				88	96
	Année	1ère	2ème	3ème		
	Directeur-Thèse	Dupont	Durant	Dupond		
	Equipe-Recherche					
	Projet-Recherche					

Tableau 4-6: l'état initial des instances de l'exemple.

Chacune de ces instances a la structure définie par sa classe d'appartenance (O_1, O_2 et O_3 sont des instances de la classe Eleve-Chercheur ; O_4 et O_5 sont des instances de la classe Ater). Le Tableau 4-6 met en évidence les attributs du patrimoine génétique valués par ces instances. Ces dernières non seulement évoluent dans leur structure initiale mais introduisent en plus de nouveaux attributs :

	Patrimoine Génétique	Les instances deviennent				
		O_1	**O_2**	**O_3**	**O_4**	**O_5**
GF	*Num-Identification*	#3	#8	#4	#7	#1
	Nom	N1	N2	N3	N4	N5
	Prénom	P1	P2	P3	P4	P5
	Thème-Recherche	Auto	- Math	Objet	Contraintes	Objet
	Laboratoire	L1	- L2	L3	L1	L5
	Nbr-Publications		- 1	4	6	
	Spécialité					
	Modules	**+Segmentation**				CSI
	Durée	- 4	- 3	- 2	1	- 1
	Date-début	**+ 01-10-2000**		**+ 01-10-2000**		**+01-09-2000**
	Nbr-heures				- 88	- 96
	Année	- 1ère	- 2ème	- 3ème		
	Directeur-Thèse	- Dupont	- Durant	- Dupond		
	Equipe-Recherche	**+ Vision**		**+ Objet**		
	Projet-Recherche					
Nouveaux attributs	*Grade*	**++ Professeur**				
	Poste		**++Ingénieur**			
	Responsabilité			**++Encadrant**		
	Responsable				++ **Mr X**	

Tableau 4-7: les instances dans leur nouvel état.

Avec :

Attribut	: attribut existant
- attribut	: attribut perdu
+ **attribut**	: attribut acquis
++ **attribut**	: nouvel attribut

Ainsi, les instances gardent certains attributs (représentés par attribut), en perdent d'autres (représentés par - attribut), acquièrent des attributs déjà existants dans le patrimoine génétique de la population (représentés par + **attribut**) et en introduisent de nouveaux (représentés par ++ **attribut**). Ces derniers ont donc le statut transitoire (confère chapitre 3).

4.4.2. Notion d'invariant

Lorsque le concepteur fait évoluer les instances, il peut agir sur deux types d'attributs : les attributs existants et les nouveaux attributs. Pour chaque attribut existant, il peut exprimer une des possibilités suivantes :

- Un attribut existant et obligatoire est représenté par un 1 dans le schème associé à l'instance ;
- Un attribut non désiré est représenté par un 0 dans le schème associé à l'instance ;
- Un attribut indifférent n'est pas représenté ou est représenté par un # dans le schème associé à l'instance.

Ainsi, les deux premières éventualités (présence obligatoire ou absence obligatoire) sont des *contraintes* ou des *invariants* qu'exprime le concepteur sur ce qu'il veut ou ne veut pas voir apparaître sur son instance, donc dans la structure conceptuelle finale. Par exemple, l'attribut *Date-début* et l'attribut *Durée* sont considérés comme contradictoires. Le premier exprime la date d'embauche d'un permanent, alors que le second exprime la durée du contrat d'un temporaire. Ces attributs caractérisent respectivement un permanent et un temporaire. Ainsi, le concepteur a exprimé que les objets O_1, O_3 et O_5 ont évolué et valuent, entre autre, l'attribut *Date-début*. Ces objets ne peuvent plus valuer l'attribut *Durée*, étant donné qu'ils sont contradictoires. Autrement une exception est levée.

Cet invariant peut avoir été exprimé dans le modèle ou, s'il en est absent, il peut être exprimé sur l'instance par le concepteur. Le processus d'évolution doit respecter cet invariant.

Nous abordons maintenant en détails chaque phase :

4.4.3. Phase d'extraction

La phase d'extraction est définie en section 4.2.2. Son automate est décrit ci-après :

4.4.3.1. Automate

L'automate représentant les différentes étapes de la phase présente en entrée et en sortie les évènements suivants :

- **En entrée :** l'instance ayant évolué dans son nouvel état.
- **En sortie :** le schème temporaire associé ; les attributs nouveaux créés avec un statut transitoire au sein du patrimoine génétique.
- **Corps de l'automate :** le cas d'une simple modification de valeurs de l'objet qui respecte toujours la définition de sa classe ne nécessite aucun traitement. Le cas qui nous intéresse est celui de modification de la structure de l'objet (ajout ou retrait d'attributs). L'automate se déroule pour chaque instance ayant évolué :

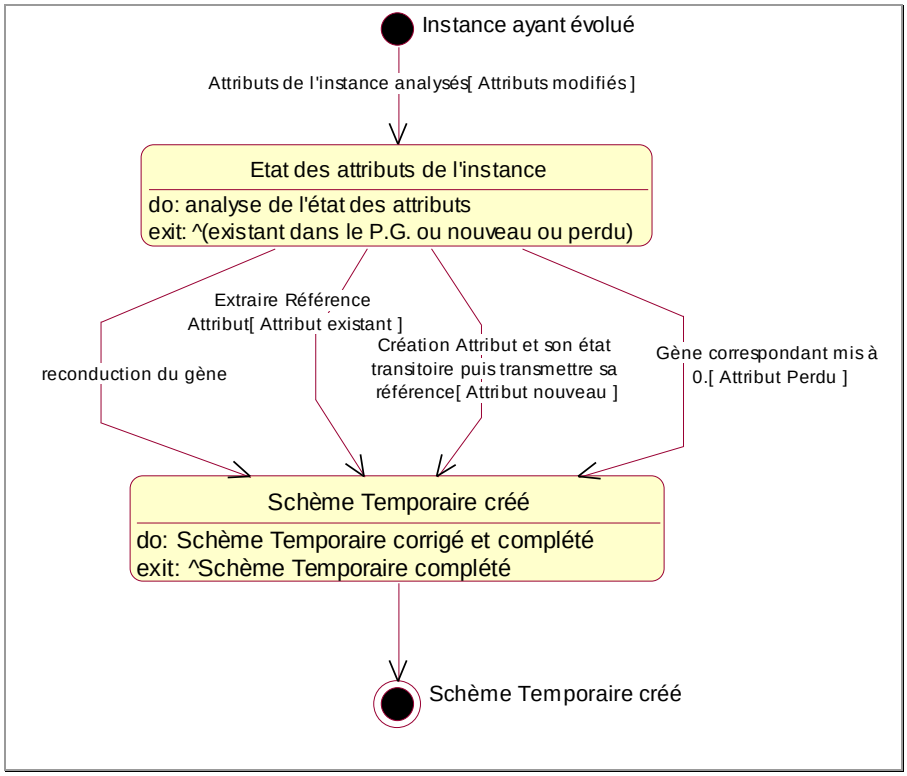


Figure 4-4. Étapes de la phase d'Extraction

Les attributs rajoutés au sein de l'objet peuvent être :

- déjà définis par d'autres classes du modèle, donc déjà présents dans le patrimoine génétique ;
- être nouvellement introduits par l'objet modifié ou encore ce dernier perd quelques attributs (il ne les value donc plus).

Dans le cas de nouveaux attributs introduits, le concepteur provoque en fait la création de ces attributs lorsqu'il modifie l'objet. Lorsqu'un attribut est créé par le concepteur durant la modification de la structure d'une instance, le système crée l'attribut en positionnant son statut à l'état transitoire.

Les attributs sont donc analysés, et le schème temporaire est créé au fur et à mesure. Pour chaque attribut, quatre cas de figures sont possibles :

1. L'attribut existe déjà et reste dans l'instance : le gène correspondant est défini dans le schème temporaire de la même manière qu'il l'est dans le schème permanent de la classe d'appartenance initiale de l'instance ;
2. L'attribut est acquis et est déjà existant dans le P.G (Patrimoine Génétique) de la population, mais n'est pas défini dans sa classe d'appartenance initiale : le processus en extrait la référence dans le gène correspondant. Sa valuation est par défaut à #. Elle est soumise à la validation ou à la modification du concepteur (qui peut éventuellement la faire muter) ;

3. L'attribut est nouvellement introduit par l'instance (par le biais du concepteur). Cet attribut doit donc être créé. Il aura un statut transitoire. Sa valuation est par défaut égale à 1. Son intégration complète dans le P.G. au même titre que les autres attributs (ou son rejet) devient effective (effectif) à la fin du processus d'évolution ;
4. L'attribut n'est plus valué par l'instance : le gène correspondant est mis à 0, mais il n'est en aucun cas supprimé du P.G.

Ainsi, au fur et à mesure que l'état des attributs de l'instance est détecté, le schème temporaire est construit. Chaque gène détient la référence de l'attribut représenté du patrimoine génétique et sa valuation (1, 0 ou #). A la fin de la phase, le schème temporaire est créé. Les éventuels nouveaux attributs également l'auront été.

Une fois le schème temporaire ainsi construit, il est soumis à la validation du concepteur. Ce dernier peut changer, au vu de ses besoins, les situations 2 et 3 car les gènes présentent des valuations par défaut.

4.4.3.2. Exemple

Les schèmes temporaires des instances ayant évolué sont :

	Patrimoine Génétique	Les schèmes temporaires sont				
		O_1	**O_2**	**O_3**	**O_4**	**O_5**
GF	*Num-Identification*	**1**	**1**	**1**	**1**	**1**
GF	*Nom*	**1**	**1**	**1**	**1**	**1**
GF	*Prénom*	**1**	**1**	**1**	**1**	**1**
	Thème-Recherche	1	0	1	1	1
	Laboratoire	1	0	1	1	1
	Nbr-Publications	#	0	1	1	#
	Spécialité	#	0	0	#	#
	Modules	1	0	0	0	1
	Durée	0	0	0	1	0
	Date-début	1	0	1	0	1
	Nbr-heures	0	0	#	0	0
	Année	0	0	0	#	0
	Directeur-Thèse	0	0	0	0	0
	Equipe-Recherche	1	0	1	#	#
	Projet-Recherche	#	0	#	#	#
Nouveaux attributs	*Grade*	1				
Nouveaux attributs	*Poste*		1			
Nouveaux attributs	*Responsabilité*			1		
Nouveaux attributs	*Responsable*				1	

Tableau 4-8: résultat de la phase d'extraction: les schèmes temporaires

Nous remarquons que chaque schème temporaire représentant un objet diffère du schème permanent de la classe d'appartenance initiale du même objet.

Une fois les schèmes temporaires spécifiés, la phase d'exploration est déclenchée.

4.4.4. Phase d'exploration

La phase d'exploration est définie en section 4.2.3. La spécification du schème temporaire obtenu à la fin de la phase d'extraction déclenche la phase d'exploration. La phase se déroule sur chaque schème temporaire :

4.4.4.1. Automate

L'automate représentant les différentes étapes de la phase présente en entrée et en sortie les évènements suivants :

- **En entrée :** le schème temporaire issu de la phase d'extraction.
- **En sortie :** le schème temporaire, les classes sélectionnées (y compris leur schème permanent).
- **Corps de l'automate :**

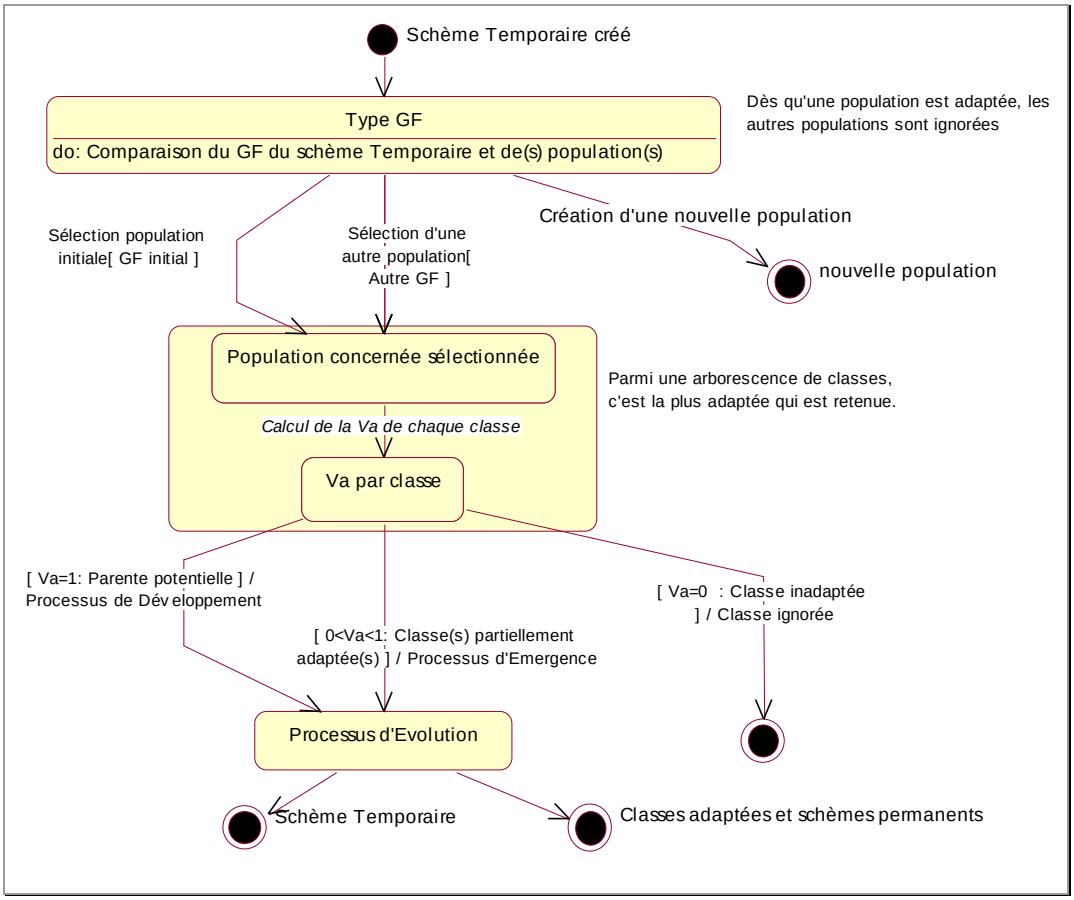


Figure 4-5. Étapes de la phase d'Exploration

L'exploration se déroule en deux principales étapes :

– L'exploration commence par explorer l'ensemble des populations spécifiées. Le but est de rechercher à quelle population le schème temporaire pourrait appartenir. La comparaison se fait donc entre le Génotype Fondamental du schème temporaire et le Génotype Fondamental représentant une population. Chaque population est passée en revue, en commençant par la population initiale. L'adaptation entre les GF doit être totale. Une fois la population sélectionnée (initiale ou autre), la seconde étape est déroulée. Nous soulignons à ce niveau, que le concepteur peut amorcer la création d'une nouvelle population s'il définit un GF totalement nouveau sur l'instance. Cela ne fait pas partie du cadre actuel de notre étude.

– L'exploration se poursuit en explorant les classes appartenant à cette population, en recherchant cette fois l'adaptation des différentes classes. La valeur d'adaptation Va de chacune est calculée. Il est important de noter que les nouveaux attributs (marqués transitoires) ne sont pas concernés dans la sélection étant donné qu'aucune classe ne les détient. De plus, le GF n'intervient pas dans le calcul de l'adaptation des classes étant donné qu'il est identique pour chacune d'entre elles. Il permet, lors de l'exploration, uniquement de sélectionner les populations, et il n'est pas utilisé au sein d'une même population.

Une fois l'exploration menée à terme, les Va (valeurs d'adaptation) sont analysées afin de détecter quel type d'évolution doit être exécutée. Cela déclenche la phase d'exploitation.

4.4.4.2. Exemples

Nous remarquons (Tableau 4-8) que pour chaque objet, le schème temporaire associé détient toujours le même GF que la population d'appartenance initiale. La première conclusion est que tous les objets doivent a priori rester dans la même population de Membre-Université. L'exploration est alors menée pour chaque schème temporaire dans cette même population. Nous rappelons qu'à cette étape, le GF et les nouveaux attributs ne sont pas considérés. Les seuls gènes considérés sont généralement définis de façon plus ou moins différentes au sein des Génotypes Hérités (GH) et des Génotypes Spécifiques (GS) des classes.

Nous présentons pour chaque objet ayant évolué, l'adaptation, donc la Va, de chaque classe de la population Membre-Université au schème permanent.

1) - Objet O_1

- Exploration

Entité	Schèmes Temporaire et Permanents												Va
Objet O_1	1	1	#	#	1	0	1	0	0	0	1	#	
Membre-Université	0	0	0	0	0	0	0	0	0	0	0	0	0/5
Chercheur	1	1	#	0	0	0	0	0	0	0	0	0	2/5
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	**1/5**
Temporaire	1	1	#	0	0	1	0	0	0	0	0	0	2/5
Permanent	1	1	1	0	0	0	1	0	0	0	0	0	3/5
Ater	1	1	#	0	1	1	0	1	0	0	0	0	**3/5**
Eleve-Chercheur	1	1	#	0	0	1	0	0	1	1	0	0	**2/5**
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	**4/5**

Tableau 4-9 : exploration de classes pour O_1.

Ainsi, les différents blocs de gènes sont représentés graphiquement par :

Génotype Hérité [ ] et Génotype Spécifique [ ]

L'objet O_1 a :

- 5 gènes spécifiés (gènes à 1),
- 3 gènes indifférents (gènes à #),
- 4 gènes obligatoirement absents (gènes à 0).

L'adaptation nulle de Membre-Université est logique étant donné qu'elle représente la population et ne détient que le GF qui est ignoré à ce niveau. L'adaptation de cette classe est la même pour tous les autres objets.

- L'adaptation de chaque classe de la hiérarchie est mise en évidence par :

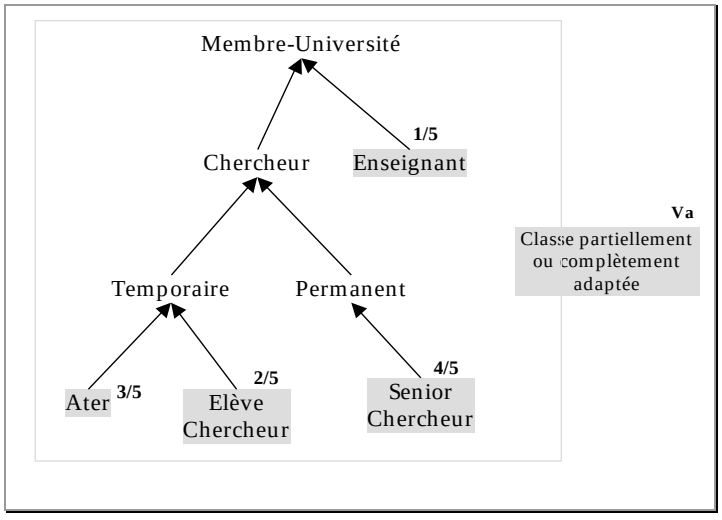


Figure 4-6: adaptation des classes à O_1.

Conclusion : quatre classes sont partiellement adaptées.

2) - Objet O_2

- Exploration

Entité	Schèmes Temporaire et Permanents												Va
Objet O_2	0	0	0	0	0	0	0	0	0	0	0	0	
Membre-Université	0	0	0	0	0	0	0	0	0	0	0	0	0
Chercheur	1	1	#	0	0	0	0	0	0	0	0	0	0
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	0
Temporaire	1	1	#	0	0	1	0	0	0	0	0	0	0
Permanent	1	1	1	0	0	0	1	0	0	0	0	0	0
Ater	1	1	#	0	1	1	0	1	0	0	0	0	0
Eleve-Chercheur	1	1	#	0	0	1	0	0	1	1	0	0	0
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	0

Tableau 4-10: exploration de classes pour O_2.

- L'adaptation de chaque classe de la hiérarchie est mise en évidence par :

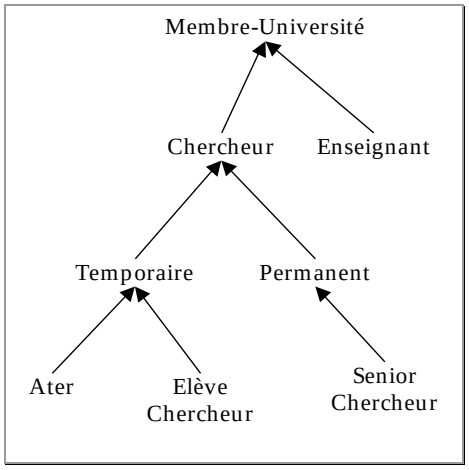


Figure 4-7: adaptation des classes à O_2.

Conclusion : aucune classe ne présente d'adaptation.

3) - Objet O_3

- Exploration

Entité	Schèmes Temporaire et Permanents												Va
Objet O_3	1	1	1	0	0	0	1	#	0	0	1	#	
Membre-Université	0	0	0	0	0	0	0	0	0	0	0	0	0/5
Chercheur	1	1	#	0	0	0	0	0	0	0	0	0	3/5
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	0/5
Temporaire	1	1	#	0	0	1	0	0	0	0	0	0	4/5
Permanent	1	1	1	0	0	0	1	0	0	0	0	0	4/5
Ater	1	1	#	0	1	1	0	1	0	0	0	0	3/5
Eleve-Chercheur	1	1	#	0	0	1	0	0	1	1	0	0	3/5
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	**5/5**

Tableau 4-11: exploration de classes pour O_3.

- L'adaptation de chaque classe de la hiérarchie est mise en évidence par :

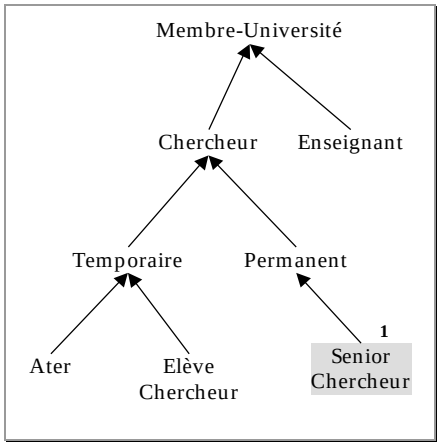


Figure 4-8: adaptation des classes à O_3.

Conclusion : une seule classe pleinement adaptée.

4) - Objet O_4

- Exploration

Entité	Schèmes Temporaire et Permanents												Va
Objet O_4	1	1	1	#	0	1	0	0	#	0	#	#	
Membre-Université	0	0	0	0	0	0	0	0	0	0	0	0	0/4
Chercheur	1	1	#	0	0	0	0	0	0	0	0	0	3/4
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	0/4
Temporaire	1	1	#	0	0	1	0	0	0	0	0	0	**4/4**
Permanent	1	1	1	0	0	0	1	0	0	0	0	0	3/4
Ater	1	1	#	0	1	1	0	1	0	0	0	0	**4/4**
Eleve-Chercheur	1	1	#	0	0	1	0	0	1	1	0	0	**4/4**
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	3/4

Tableau 4-12: exploration de classes pour O_4.

- L'adaptation de chaque classe de la hiérarchie est mise en évidence par :

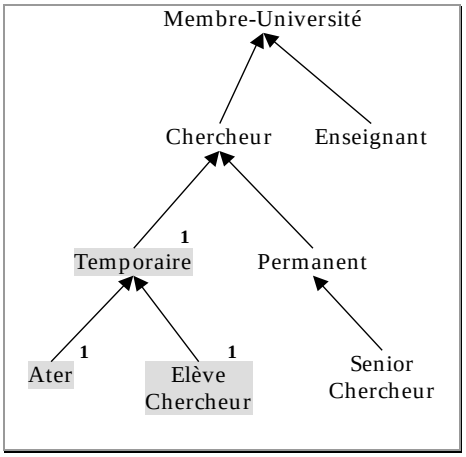


Tableau 4-13: adaptation des classes à O_4.

Conclusion : trois classes sont pleinement adaptées.

5) - Objet O_5

- Exploration

Entité	Schèmes Temporaire et Permanents												Va
Objet O_5	1	1	#	#	1	0	1	0	0	0	#	#	
Membre-Université	0	0	0	0	0	0	0	0	0	0	0	0	0/4
Chercheur	1	1	#	0	0	0	0	0	0	0	0	0	2/4
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	**1/4**
Temporaire	1	1	#	0	0	1	0	0	0	0	0	0	2/4
Permanent	1	1	1	0	0	0	1	0	0	0	0	0	3/4
Ater	1	1	#	0	1	1	0	1	0	0	0	0	**3/4**
Eleve-Chercheur	1	1	#	0	0	1	0	0	1	1	0	0	**2/4**
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	**3/4**

Tableau 4-14: exploration de classes pour O_5.

- L'adaptation de chaque classe de la hiérarchie est mise en évidence par :

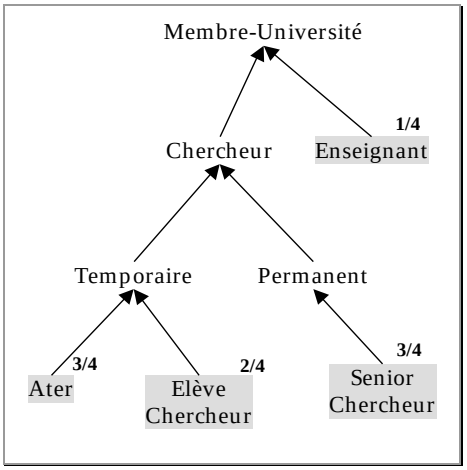


Tableau 4-15: adaptation des classes à O_5.

Conclusion : quatre classes sont partiellement adaptées.

6) - Conclusion

En résumé, nous obtenons pour chaque objet :

Objet	Résultat de l'exploration	Classes complètement ou partiellement adaptées
O_1	Quatre classes partiellement adaptées	Enseignant, Ater, Eleve-Chercheur, Senior-Chercheur
O_2	Aucune classe adaptée	-
O_3	Une seule classe pleinement adaptée	Senior-Chercheur

O_4	Trois classes pleinement adaptées	Temporaire, Ater, Eleve-Chercheur
O_5	Quatre classes partiellement adaptées	Enseignant, Ater, Eleve-Chercheur, Senior-Chercheur

Tableau 4-16: résultats de l'exploration.

Ces résultats permettent de mener à terme la troisième et dernière phase : l'exploitation.

4.4.5. Phase d'exploitation

La phase d'exploitation est définie en section 4.2.4. La phase d'exploitation permet de déclencher le processus de développement ou d'émergence. Pour savoir lequel des processus est mis en marche, elle s'appuie sur les résultats de la phase d'exploration. L'algorithme présenté ci-après précise les différentes étapes suivies et celles qui suivront au vu des résultats obtenus jusque là :

Si GF(instance) = GF(Population i)

Alors les classes de la population i sont considérées. Le GF n'est plus pris en compte dans la suite du processus

Si un schème permanent a une Va=1

Alors émergence locale dans sa classe :

+ intégration de l'instance dans son nouvel état

+ adaptation de cette instance à sa nouvelle classe par éventuel ajout de la différence dans les attributs

+ adaptation si nécessaire des instances existantes

+ répercussion sur les sous-classes et leurs instances

Sinon

Si plusieurs schèmes ont une Va=1

Alors

- *1er cas :* la Va de la racine d'une sous-branche de la hiérarchie est égale à 1 ainsi que ses sous-classes : l'instance exprime une variante de cette même racine. Cette dernière est généralement abstraite. La structure émergeant de l'instance est proposée comme une sous-classe de cette racine

- *2ème cas :* des classes disjointes ayant une Va=1 porte à penser qu'il y a un problème de conception.

Sinon des Va sont strictement comprises entre 0 et 1 : croisement :

Constitution de la population de départ avec tous les schèmes partiellement adaptés et significatifs.

- *Adapté :* 0<Va<1

- *Significatif :* dans une même sous-branche de la hiérarchie, la classe la plus adaptée est sélectionnée. Pourquoi ? Étant donné que se sont des spécialisations d'une même classe, elles détiennent une partie commune. Il est inutile et redondant de la dupliquer, mais il est préférable de sélectionner la sous-classe la plus adaptée.

Tableau 4-17: l'algorithme permettant d'exploiter, dans un premier temps, les résultats de la phase d'exploration.

4.4.5.1. Automates

Les différents automates que nous définissons pour les processus d'évolution ont tous les mêmes entités en entrée et l'achèvement de l'évolution en sortie :

– **En entrée :** les schèmes émergents, en plus des entités déjà présentes : les schèmes temporaires, les schèmes permanents, le patrimoine génétique.

– **En sortie :** l'évolution achevée, c'est-à-dire :

La classe à modifier en cas d'émergence locale ;

La ou les nouvelles structures conceptuelles à insérer et son ou leur emplacement dans la hiérarchie en cas d'émergence globale.

– **Corps de l'automate :** nous définissons un automate par type de processus d'évolution. Ils sont explicités séparément dans ce qui suit.

Nous présentons en premier lieu le constat des résultats obtenus jusque là suite à l'exécution des deux premières phases :

4.4.5.2. Exemple

– *Conclusion O_1 :* des classes partiellement adaptées. Le processus d'évolution adéquat est *l'émergence globale par croisement*.

– *Conclusion O_2 :* aucune classe ne présente d'adaptation. Le processus d'évolution adéquat est *l'émergence globale par création directe de la structure conceptuelle*.

– *Conclusion O_3 :* une seule classe pleinement adaptée et l'objet O_3 introduit de nouveaux gènes. Le processus d'évolution adéquat est :

- *l'émergence locale* au sein de la classe qui est pleinement adaptée si cette dernière n'est pas une classe abstraite ;
- *l'émergence globale par création directe* de la structure conceptuelle comme sous-classe de la classe pleinement adaptée si cette dernière est abstraite.

– *Conclusion O_4 :* trois classes pleinement adaptées et l'objet O_4 introduit de nouveaux gènes. Le processus d'évolution adéquat est :

- *l'émergence locale* au sein de la classe qui est pleinement adaptée si cette dernière n'est pas une classe abstraite ;
- *l'émergence globale par création directe* de la structure conceptuelle comme sous-classe de la classe pleinement adaptée si cette dernière est abstraite.

– *Conclusion O_5 :* des classes partiellement adaptées. Le processus d'évolution adéquat est *l'émergence globale par croisement*.

En résumé :

Objet	Résultat de l'exploration	Processus d'évolution adéquat
O_1	Quatre classes partiellement adaptées	*Émergence globale par croisement* des classes partiellement adaptées
O_2	Aucune classe adaptée	*Émergence globale par création directe de la structure conceptuelle*
O_3	Une seule classe pleinement adaptée	*Émergence locale* ou *Émergence Globale directe*
O_4	Trois classes pleinement adaptées	*Émergence locale* ou *Émergence Globale directe*
O_5	Quatre classes partiellement adaptées	*Émergence globale par croisement* des classes partiellement adaptées

Tableau 4-18: Processus d'évolution pour chaque objet.

Ces différents processus sont présentés ci-après. Nous présentons la politique adoptée pour le processus de développement en section 4.6 - .

4.5 - L'émergence

Nous distinguons les automates en définissant un automate par type d'émergence : l'émergence locale et l'émergence globale. Cette dernière distingue deux automates : celui de l'émergence globale directe et celui de l'émergence globale par croisement.

4.5.1. L'émergence locale

L'émergence locale permet de faire émerger des gènes nouvellement introduits dans une ou plusieurs classes pleinement adaptées. En effet, aussi bien pour l'objet O_3 que O_4, des classes ont leur Va=1, mais cette adaptation est valable uniquement sur les gènes déjà existants. Elle ne concerne pas les nouveaux gènes. Ces derniers, après que le concepteur accepte de mener une émergence locale, doivent émerger au sein de ces classes, avant que l'objet correspondant ne lui soit rattaché.

4.5.1.1. L'automate

L'automate représentant les différentes étapes de la phase :

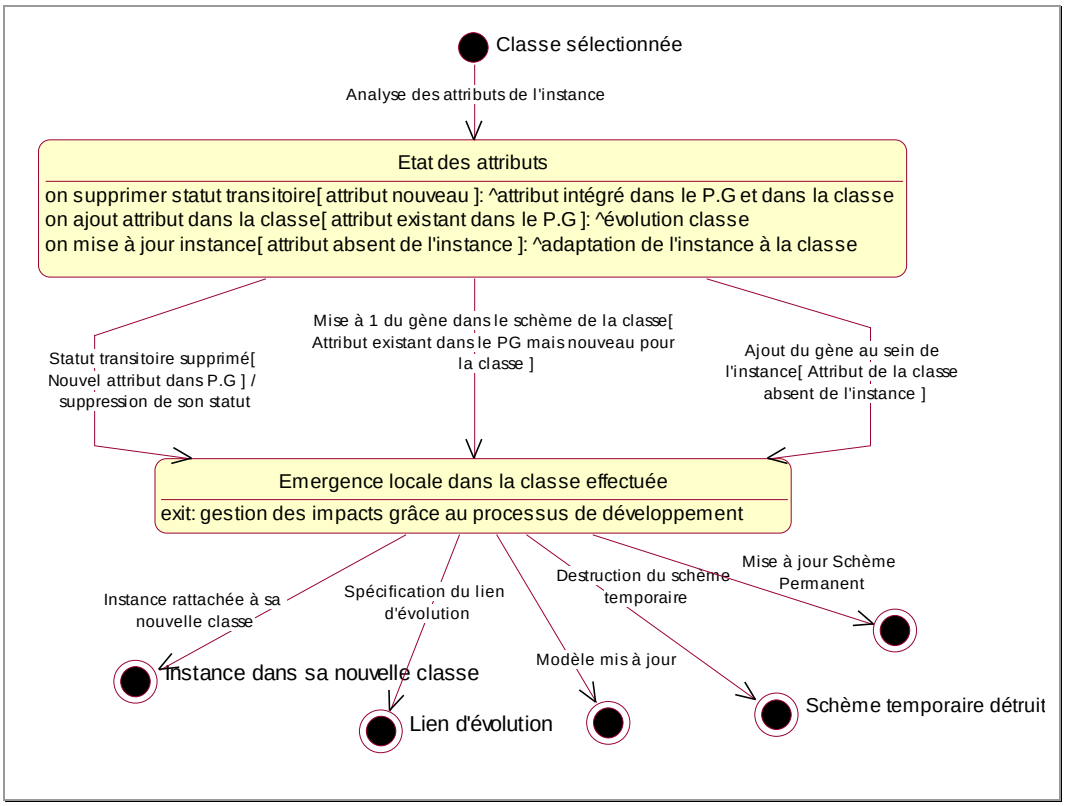


Figure 4-9. Étapes de l'Émergence Locale

L'émergence locale permet de faire évoluer la classe existante dans le modèle et qui est adaptée à l'instance pour lui permettre de prendre en compte les nouveaux attributs présents dans l'instance. Pour cela, la classe doit évoluer pour s'adapter pleinement. L'instance doit également s'adapter à sa nouvelle classe le cas échéant. Ainsi, outre les attributs qui émergent dans la classe, l'instance doit prendre les attributs de la classe et qu'elle ne détient pas.

Le concepteur doit également s'assurer de la complétude du schème permanent. En effet, ce dernier prend par défaut les valuations mentionnées dans le schème temporaire pour les nouveaux attributs. Le schème permanent est soumis à la validation du concepteur. Les instances de la classe doivent s'adapter. Cela se fera au travers du processus de développement (section 4.6 -). La dualité classe/instance est à nouveau respectée dans un état stable du modèle (Figure 4-1).

4.5.1.2. L'exemple

Pour les objets O_3 et O_4, le processus d'émergence peut a priori être soit de l'émergence locale soit de l'émergence globale par création directe de la structure conceptuelle. Analysons les deux cas :

- *Objet O_3 :* Senior-Chercheur est la seule classe ayant une Va adaptée à l'objet O_3. De plus, cette classe peut avoir des instances. Il s'agit donc d'*émergence locale*. L'objet O_3 est proposé comme instance de Senior-Chercheur, moyennant l'émergence locale du nouvel attribut *Responsabilité* introduit par l'instance, dans cette classe. Le lien d'évolution (introduit dans le chapitre 4 et traité en détails dans la section 4.7 -) est créé de la classe initiale de l'objet O_3, à savoir Eleve-Chercheur vers la nouvelle classe de l'objet O_3, à savoir Senior-Chercheur. L'objet lui est rattaché.
- *Objet O_4 :* trois classes ont une Va égale à 1. Les classes correspondantes sont donc en mesure d'être ou de contenir la structure conceptuelle sous-jacente à l'instance, donc l'instance à proprement parler. Laquelle choisir ? Des trois classes, la classe `Temporaire` est la plus éligible car elle représente la racine de la sous-population des temporaires. C'est une classe abstraite et elle est utilisée pour factoriser des caractéristiques communes à ses différentes sous-classes. Elle ne peut donc pas contenir l'instance O_4. Celle-ci représente une nouvelle catégorie de temporaire et introduit un nouvel attribut. Il ne s'agit donc pas d'émergence locale mais d'*émergence globale directe*. L'exemple doit donc être traité dans ce cadre (section 4.5.2.1).

4.5.2. L'émergence globale

L'émergence globale concerne l'émergence d'une nouvelle structure conceptuelle. Celle-ci peut être une simple structure conceptuelle représentée par une classe ou une association entre deux classes, ou une structure conceptuelle plus complexe, représentée par des classes liées entre elles. Cela est le cas des objets complexes. Nous nous intéresserons principalement à l'émergence de structures conceptuelles simples.

Il existe deux principaux types d'émergence : l'émergence globale par création directe de la structure conceptuelle, que nous qualifions également d'*émergence directe*, et l'émergence globale par croisement.

4.5.2.1. L'émergence directe

L'émergence globale directe revient à extraire la spécification d'une nouvelle structure conceptuelle directement d'une instance. Elle est adaptée pour les situations suivantes :

- Aucune classe n'est adaptée à une instance et cette dernière appartient tout de même à la population ;
- La classe qui est adaptée est une classe abstraite. Cette situation porte à croire que l'instance exprime une nouvelle structure conceptuelle de cette classe.

1) - L'automate

La phase s'applique pour une instance. L'automate représentant les différentes étapes de la phase est :

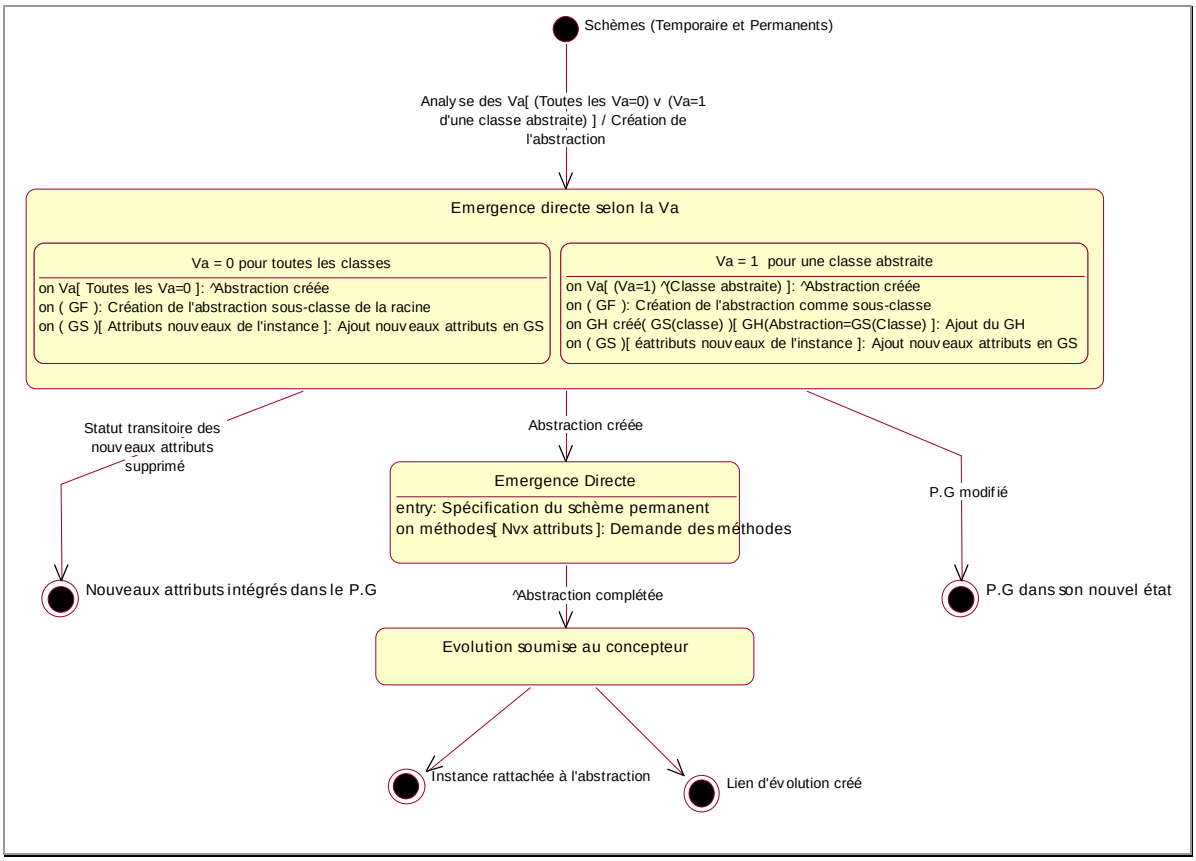


Figure 4-10. Étapes de l'Émergence Globale Directe.

L'objectif de cet automate est de détecter l'emplacement de la nouvelle structure conceptuelle dans la hiérarchie de la population :

Soit aucune classe n'est adaptée, auquel cas cela revient à créer la structure conceptuelle comme une sous-classe de la classe racine de la hiérarchie. Dans ce cas, outre le GF qui reste inchangé, les nouveaux attributs sont intégrés dans le P.G en perdant leur statut transitoire. Ils constituent le GS de la structure conceptuelle. Il n'y a pas de GH dans la structure.

Soit la classe adaptée est une classe abstraite, auquel cas cela revient à créer la structure conceptuelle comme une sous-classe de la classe abstraite adaptée. Dans ce cas, outre le GF qui reste inchangé, les nouveaux attributs sont intégrés dans le P.G en perdant leur statut transitoire. Ils constituent le GS de la structure conceptuelle. Le GS de la super-classe est hérité par la classe nouvellement créée et constitue ainsi son GH.

Le résultat final est soumis au concepteur pour validation. Il lui est également demandé de compléter la structure conceptuelle en spécifiant les méthodes qui invoquent ces attributs (section 4.5.3). Celui-ci doit valider et compléter selon ses besoins les spécifications de la structure conceptuelle.

2) - Les exemples

- *Objet O_2 : émergence globale par création directe de la structure conceptuelle – lieu d'insertion ?* Dans ce cas, la structure n'a rien de commun avec les autres classes sauf le GF. Elle appartient donc à la même population. Elle sera donc créée comme sous-classe de Membre-Université, car le GF est le seul point commun avec la population. L'attribut *Poste* introduit par l'instance fait maintenant partie intégrante du PG. Cet attribut constitue le GS de la nouvelle classe. Il est valué par la valeur 'Ingénieur'. Cette structure conceptuelle représente une nouvelle catégorie de Membre-Université, à savoir celle d'Ingénieur. L'objet O_2 lui est rattaché. Un lien d'évolution (confère section 4.7 -) est créé de sa classe d'appartenance initiale, à savoir Eleve-Chercheur, vers sa nouvelle classe. Le résultat de l'évolution, à savoir la classe Ingénieur, est proposé au concepteur, y compris le schème permanent qui n'est autre que le schème temporaire. Il est également demandé au concepteur de spécifier les méthodes nécessaires (aussi bien dans la signature que le corps).
- *Objet O_4 : émergence globale par création directe de la structure conceptuelle – lieu d'insertion ?* Trois classes ont leur Va égale à 1. Les classes correspondantes sont donc en mesure d'être ou de contenir la structure conceptuelle sous-jacente à l'instance, donc l'instance à proprement parler. Des trois classes, la classe `Temporaire` est la plus éligible car elle représente la racine de la sous-population des temporaires. C'est une classe abstraite et elle est utilisée pour factoriser des caractéristiques communes à ses sous-classes. Elle détient les attributs de l'instance O_4, sauf les nouveaux attributs qu'elle introduit. Elle ne peut pas contenir l'instance O_4 mais peut généraliser la structure conceptuelle représentée. Celle-ci représente une nouvelle catégorie de temporaire et introduit un nouvel attribut *Responsable*. La classe correspondante est créée comme sous-classe de `Temporaire`.

  L'attribut *Responsable* introduit par l'instance fait maintenant partie intégrante du PG. Cet attribut constitue le GS de la nouvelle classe. Il est valué par la valeur 'Mr X' dans l'exemple. Le GS de Temporaire est hérité par la nouvelle classe. Il constitue son GH. Cette structure conceptuelle représente une nouvelle catégorie de Membre-Université, à savoir celle de Post-Doc. L'objet O_4 lui est rattaché. Un lien d'évolution (confère section 4.7 -) est créé de sa classe d'appartenance initiale, à savoir Ater, vers sa nouvelle classe. Le résultat de l'évolution est proposé au concepteur, à savoir la classe Post-Doc, y compris le schème permanent qui n'est autre que le schème temporaire. Il est également demandé au concepteur de spécifier les méthodes nécessaires (aussi bien dans la signature que le corps).

4.5.2.2. L'émergence globale par croisement

L'émergence globale par croisement est exécutée lorsque plusieurs classes sont partiellement adaptées à l'instance. Le croisement permet alors de tirer profit au maximum des propriétés définies au sein de ces classes partiellement adaptées et de l'instance. Le croisement se fait au vu des propriétés (ou gènes) présentes dans l'instance, et en respectant les contraintes et invariants présents dans le modèle.

Les classes sélectionnées voient leurs schèmes permanents constituer la population de départ pour le croisement. La population est de taille fixe tout au long du processus d'évolution. Les nouveaux schèmes remplaceront leurs parents, permettant ainsi de garder la taille de la population fixe tout en variant et changeant sa composition.

1) - Définition détaillée du croisement

Deux schèmes sont choisis aléatoirement dans la population de croisement pour être croisés. Le choix de points de croisement est important. Il existe les croisements classiques dits à n-points (chapitre 3), mais l'aspect de scinder en groupes des gènes ne nous satisfait pas, car il n'y a pas de sémantique particulière dans la séparation des gènes. Nous sommes plus intéressés par une sélection aléatoire des gènes. Pour ces raisons, nous nous sommes plutôt inspirés du *croisement uniforme* des Algorithmes Génétiques [Syswerda 89]. Car ce type de croisement permet d'introduire de l'aléa dans le choix du gène à transmettre au descendant. Ce type de croisement permet de ne pas se confiner trop rapidement dans un même espace de solutions et permet plutôt de diversifier les descendants, donc les structures conceptuelles. Quant à l'aspect groupe de gènes, nous préférons appliquer les groupes de gènes du modèle, à savoir les GH et les GS.

Le principe du croisement uniforme est d'accorder un poids relatif à chacun des deux parents pour la transmission de gènes dans un des enfants. Nous déterminons et calculons ce poids en fonction de la valeur d'adaptation de chaque schème parent (cf. l'exemple). Afin d'uniformiser ces Va, nous pondérons ces deux Va entre elles de manière à ce que leur somme fasse 1. Nous les dénommons *Vap* pour *Valeur d'Adaptation Pondérée*. Ainsi, nous savons sans ambiguïté quel schème aura plus de chances d'influencer les générations futures. Nous considérons également que le schème parent 1 influence le schème enfant 1 et que le schème parent 2 influence le schème enfant 2.

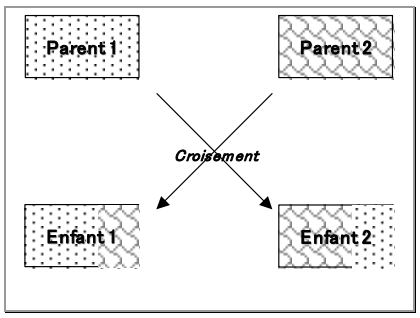


Figure 4-11: prédominance d'un parent sur un enfant.

Ensuite, un nombre aléatoire compris entre 0 et 1 est généré pour chaque position de gène. Ce nombre est comparé au poids du premier parent. S'il lui est inférieur, le gène du schème parent 1 est transmis à l'enfant 1 et le gène du schème parent 2 est transmis à l'enfant 2. Dans le cas contraire, la permutation de gènes est faite : c'est-à-dire que le gène du parent 1 est transmis à l'enfant 2 et le gène du parent 2 est transmis à l'enfant 1.

En notant Vap_1 la Va pondérée du schème parent 1 et Vap_2 la Va pondérée du schème parent 2, nous illustrons la procédure précédente de la manière suivante :

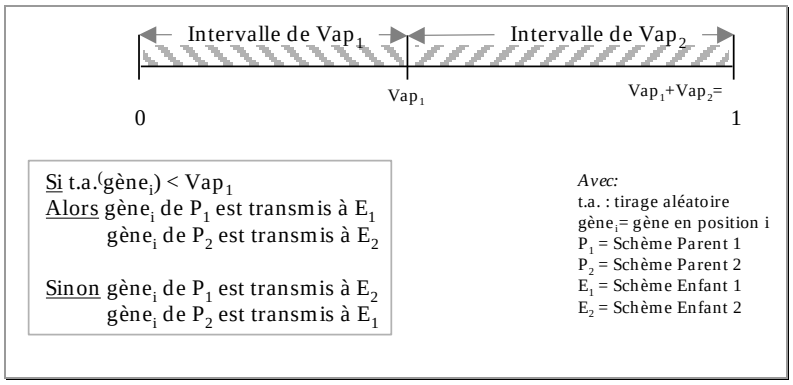


Figure 4-12: Transmission des gènes lors du croisement

Cette opération de tirage aléatoire se répète jusqu'à ce que tous les gènes soient passés en revue.

Les schèmes enfants remplacent respectivement leurs parents dans la population de croisement. Une nouvelle génération est ainsi définie. Le croisement est relancé depuis le début sur cette nouvelle génération. Le critère d'arrêt est l'émergence de schèmes pleinement adaptés ou la dégradation des performances du croisement.

2) - La contrainte de blocs de gènes dans le croisement

Une fois le croisement fait, nous imposons une contrainte que nous qualifions de contrainte de *blocs de gènes*. En effet, la définition d'un attribut au sein d'une classe est plus riche sémantiquement qu'en étant défini avec le groupe d'attributs auquel il appartient. Ainsi, l'attribut *Nbr-Publications* défini dans la classe Chercheur, présente une sémantique restreinte, alors que s'il est accompagné de tous les attributs définis avec lui dans le GS, à savoir *Thème-Recherche* et *Laboratoire*, sa sémantique est plus riche. Nous imposons pour cela que les blocs de GH et de GS doivent être conservés lors d'un croisement. Cette contrainte évite surtout de retrouver au bout d'un certain nombre de croisements, des schèmes avec des structures complètement disloquées.

Pour qu'un bloc GH ou GS soit conservé, la contrainte est que si un attribut est présent dans un schème résultant d'un croisement et que ce même attribut appartient à un bloc GH ou GS de son schème parent, alors tous les attributs du bloc doivent être présents dans le schème résultant du croisement. Par contre, si aucun gène du bloc GH ou GS n'est présent dans le schème issu du croisement, les gènes du bloc ne lui sont pas imposés, mais la structure du bloc est toujours transmise aux schèmes enfants.

Comme le schème parent 1 influence le schème enfant 1 et que le schème parent 2 influence le schème enfant 2, le schème enfant 1 devra respecter les blocs GH et GS du schème parent 1 et le schème enfant 2 devra respecter les blocs GH et GS du schème parent 2.

3) - L'automate

L'automate représentant les différentes étapes de la phase :

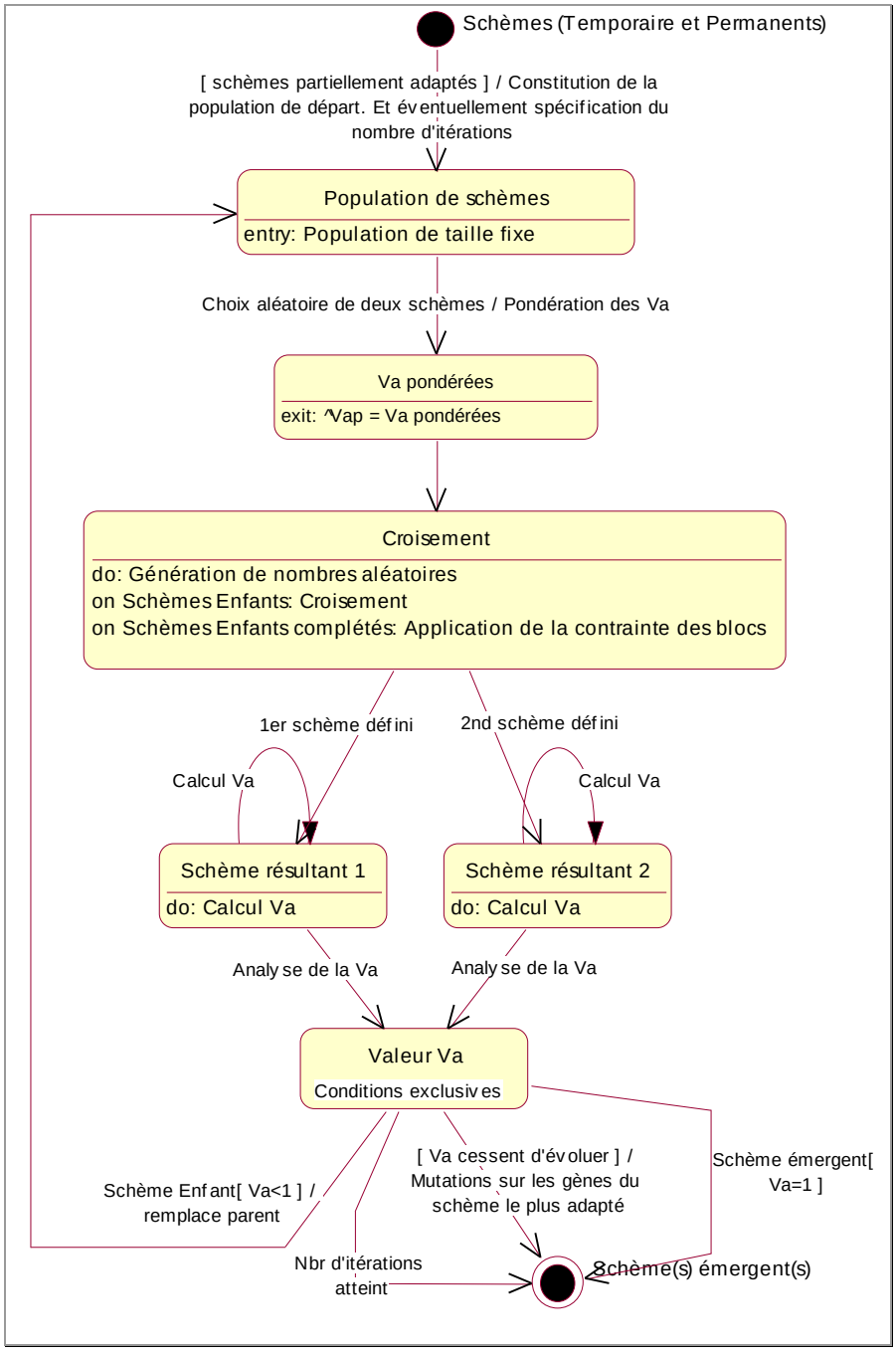


Figure 4-13. Le croisement : cœur du processus d'émergence globale

Tant qu'aucun des critères d'arrêts n'est atteint, les schèmes enfants sont introduits dans la population de croisement en remplacement de leurs parents. Une nouvelle génération est ainsi définie. Le processus de croisement reprend sur cette population.

Le critère d'arrêt est l'émergence de schèmes pleinement adaptés ou la dégradation des performances du croisement. En effet, le croisement peut aboutir à des schèmes émergents (Va=1). Le croisement se sera donc déroulé avec succès. Cependant, le processus ne peut pas durer indéfiniment si aucun schème émergeant n'apparaît. Le

critère d'arrêt peut donc être une dégradation constatée des résultats du croisement : les schèmes obtenus sont de moins en moins adaptés.

D'autres critères d'arrêt peuvent dépendre du concepteur. L'arrêt du croisement peut être du à l'intervention du concepteur pour arrêter explicitement le croisement. L'arrêt du croisement peut également arriver si un certain nombre d'itérations autorisé est défini par le concepteur. Dans le cas où aucun schème émergeant n'apparaîtrait, les schèmes les plus adaptés sont proposés au concepteur. Celui-ci peut alors décider de faire muter quelques gènes des schèmes de façon à obtenir des schèmes pleinement adaptés. Cette mutation est entièrement laissée à la charge du concepteur.

A la fin, le schème de la nouvelle structure conceptuelle a ainsi émergé. Les parents « géniteurs » permettent de construire par blocs la nouvelle structure conceptuelle dont le code génétique est détenu par le schème émergent.

Le croisement est illustré sur deux exemples :

4) - Les exemples

Le croisement est déroulé sur les objets nécessitant de déclencher une émergence globale par croisement, en l'occurrence O_1 et O_5 :

- *Objet O_1 : croisement :* la population de départ pour le croisement est constituée des schèmes permanents des classes partiellement adaptées que sont `Enseignant`, `Ater` et `Senior-Chercheur`. Le schème permanent de la classe `Elève-Chercheur` n'est pas considéré car elle est moins adaptée que sa consœur `Ater` dans la même sous-population `Temporaire` (au vu de ce qui est expliqué dans la section 4.4.5).

  - Le croisement est effectué sur autant de générations de population que nécessaire. Nous présentons le croisement pour chaque génération séparément, ce qui nous permet d'expliquer chaque étape, la première génération est constituée des schèmes permanents sus-cités.
  - Les schèmes sont préalablement uniformisés lors de la phase d'exploration. Nous remarquons que deux gènes du patrimoine génétique sont absents aussi bien du schème temporaire de O_1 que des schèmes permanents de la population de croisement. Il s'agit du 9ème et du 10ème gènes (*Année* et *Directeur-Thèse*). Il est inutile d'appliquer le croisement sur ces gènes. Ils ne sont donc pas considérés. Ils sont représentés comme suit :

0	Gène absent de tous les schèmes. Il n'est pas pris en compte dans le processus de croisement car il n'a aucune signification

  - Le tirage aléatoire est fait. Il est toujours compris entre 0 et 1. Il y a un tirage aléatoire par gène.

- Deux schèmes sont choisis aléatoirement de la population de croisement : Enseignant et Ater. Ils sont mis en évidence dans la population de croisement par un soulignement.
- Ils ont pour Va respectivement 1/5 et 3/5. Leurs Va sont pondérées entre elles. Leur Vap respective est : 1/4 et 3/4. Leur somme fait bien 1.
- Les deux schèmes choisis sont mis en place pour effectuer le croisement au vu des différents tirages aléatoires. Les schèmes enfants (donc issus du croisement) sont notés E_i, i étant le numéro séquentiel de l'apparition du schème enfant :

Population de départ (1ère génération) : Croisement

1ère étape : croisement

O_1	1	1	#	#	1	0	1	0	0	0	1	#	
Tirage aléatoire	*0.34*	*0.59*	*0.42*	*0.70*	*0.15*	*0.60*	*0.50*	*0.43*			*0.10*	*0.90*	*Vap*
Enseignant	0	0	0	1	1	0	0	0			0	0	1/4
Ater	1	1	#	0	1	1	0	1			0	0	3/4
E1	1	1	#	0	1	1	0	1			0	0	
E2	0	0	0	1	1	0	0	0			0	0	

2ème étape : contrainte des blocs

O_1	1	1	#	#	1	0	1	0	0	0	1	#	
Tirage aléatoire	*0.34*	*0.59*	*0.42*	*0.70*	*0.15*	*0.60*	*0.50*	*0.43*			*0.10*	*0.90*	*Va*
Enseignant	0	0	0	1	1	0	0	0			0	0	
Ater	1	1	#	0	1	1	0	1			0	0	
E1	1	1	#	1	1	1	0	1			0	0	3/5
E2	0	0	0	1	1	0	0	1			0	0	1/5

Tableau 4-19: Les deux étapes de toute itération d'un croisement au travers d'un exemple

- La première étape consiste à effectuer le croisement gène à gène en fonction du tirage aléatoire et des deux Vap.
- La seconde étape consiste à faire appliquer les contraintes de blocs. Ainsi, les gènes modifiés suite à l'application de la contrainte des blocs sont ceux pointés par une flèche. Ils ont été modifiés de façon à avoir les blocs GS des deux parents respectifs (Enseignant pour E1 et Ater pour E2). Par contre, le GH de E1 n'est pas modifié puisqu'il est entièrement spécifié. Par contre, les gènes du GH de E2 restent à zéro étant donné qu'aucun gène du GH de Ater n'y est spécifié, mais sa structure est transmise.

La Va des deux schèmes enfants ainsi obtenus est calculée. Elles sont toutes les deux inférieures à 1. Le croisement continue. Les schèmes enfants remplacent leurs parents respectifs. Nous remarquons que la taille de la population est toujours la même. La

deuxième génération est prête pour une itération du croisement. Nous regroupons ensemble dans la suite de l'illustration du croisement, le croisement à proprement parler et la contrainte de blocs. Deux schèmes sont choisis aléatoirement : E2 et Senior-Chercheur.

– La seconde génération connaît le croisement suivant :

2ème génération	Croisement

Schèmes sélectionnés

E1

E2

Senior-Chercheur

O_1	1	1	#	#	1	0	1	0	0	0	1	#	
Tirage aléatoire	*0.05*	*0.12*	*0.20*	*0.10*	*0.49*	*0.79*	*0.37*	*0.25*			*0.85*	*0.12*	*Vap*
E2	0	0	0	1	1	0	0	1			0	0	1/5
SeniorChercheur	1	1	1	0	0	0	1	0			1	#	4/5
E3	0	0	0	1	0	0	1	0			1	0	2/5
E4	1	1	1	0	1	0	1	0			1	#	5/5

Tableau 4-20: la seconde génération est croisée.

À la fin de cette deuxième itération de croisement, il y a un schème émergent : E4. Le croisement continue de façon à donner également des chances à d'autres éventuels schèmes émergents. Les schèmes enfants remplacent leurs parents respectifs dans la population. La troisième génération est prête pour une autre itération du croisement.

– La troisième génération connaît le croisement suivant :

3ème génération	Croisement

Schèmes sélectionnés

E1

E3

E4

O_1	1	1	#	#	1	0	1	0	0	0	1	#	
Tirage aléatoire	*0.34*	*0.19*	*0.02*	*0.70*	*0.15*	*0.60*	*0.50*	*0.33*			*0.80*	*0.90*	*Vap*
E1	1	1	#	1	1	1	0	1			0	0	3/5
E3	0	0	0	1	0	0	1	0			1	0	2/5
E5	1	1	#	1	1	0	1	1			1	0	5/5
E6	1	1	1	1	0	1	0	0			0	0	2/5

Tableau 4-21: troisième et dernière itération du croisement

Le processus de croisement s'achève car tous les schèmes ont participé au croisement et au moins un schème émergent est apparu. Il y a en fait deux schèmes émergents : E4 et E5.

Le même processus est appliqué sur le schème temporaire de l'objet O_5 :

- *Objet O_5 : croisement :* la population de départ pour le croisement est constituée de : `Enseignant`, `Ater` et `Senior-Chercheur`. La classe `Eleve-Chercheur` n'est pas considérée car elle est moins adaptée que sa consœur `Ater` dans la même sous-population `Temporaire`.

N°Population	Croisement
1 : Enseignant Ater Senior-Chercheur	O_5 : 1 1 # # 1 0 1 0 0 0 # # *Tirage aléatoire* : *0.23 0.68 0.54 0.95 0.32 0.71 0.48 0.78 0.48 0.74* Enseignant : 0 0 0 1 1 0 0 0 0 0 — *Vap* 1/4 Ater : 1 1 # 0 1 1 0 1 0 0 — *Vap* 3/4 E1 : 0 1 # 0 0 0 0 1 0 0 — *Vap* 1/4 E2 : 1 1 # 1 1 0 0 0 0 0 — *Vap* 3/4
2 : E1 E2 Senior-Chercheur	O_5 : 1 1 # # 1 0 1 0 0 0 # # *Tirage aléatoire* : *0.54 0.28 0.15 0.78 0.46 0.51 0.91 0.85 0.71 0.15* E1 : 0 1 # 0 0 0 0 1 0 0 — *Vap* 1/4 S[r]Chercheur : 1 1 1 0 0 0 1 0 1 # — *Vap* 3/4 E3 : 1 1 # 0 0 0 1 0 1 0 — *Vap* 3/4 E4 : 1 1 1 0 0 0 1 1 1 # — *Vap* 3/4
3 : E3 E2 E4	O_5 : 1 1 # # 1 0 1 0 0 0 # # *Tirage aléatoire* : *0.42 0.62 0.13 0.19 0.45 0.18 0.73 0.61 0.85 0.12* E2 : 1 1 # 1 1 0 0 0 0 0 — *Vap* 1/2 E4 : 1 1 1 0 0 0 1 1 1 # — *Vap* 1/2 E5 : 1 1 # 1 1 0 1 1 1 0 — *Vap* **4/4=1** E6 : 1 1 1 0 0 0 1 0 1 # — *Vap* 3/4
4 : E3 E5 E6	O_5 : 1 1 # # 1 0 1 0 0 0 # # *Tirage aléatoire* : *0.12 0.56 0.47 0.81 0.41 0.52 0.23 0.74 0.21 0.51* E3 : 1 1 # 0 0 0 1 0 1 0 — *Vap* 1/2 E6 : 1 1 1 0 0 0 1 0 1 # — *Vap* 1/2 E7 : 1 1 # 0 0 0 1 0 1 # — *Vap* 3/4 E8 : 1 1 1 0 0 0 1 0 1 # — *Vap* 3/4

Tableau 4-22: les étapes du croisement sur le schème temporaire représentant O_5

Le croisement s'achève sur la quatrième population. Ce dernier croisement ne présente pas de schème émergent, mais le croisement précédent a donné lieu au schème émergent E5.

4.5.3. Discussion sur l'émergence

Nous avons vu que nous déclinons l'émergence en :

- émergence locale,
- émergence globale
  - par création directe de la structure conceptuelle,

- par croisement.

Ces différents types d'émergence ont été illustrés au travers des exemples traités. Le type d'émergence à appliquer, en déclenchant le processus d'évolution correspondant, est déterminé au vu des résultats de la phase d'exploration. Dans le cadre de l'émergence locale d'un attribut et de l'émergence directe d'une structure conceptuelle, l'endroit d'insertion (au sein d'une classe ou de la hiérarchie) est généralement aisément détectable. Dans le cadre de l'émergence globale par croisement, la détection de l'endroit de la modification, c'est-à-dire le lieu d'insertion dans la hiérarchie, n'est pas triviale.

Ainsi, pour chaque objet ayant évolué, le type d'émergence a été détecté en s'appuyant sur les valeurs d'adaptation :

- l'émergence locale pour l'objet O_3,
- l'émergence globale par création directe de la structure conceptuelle pour les objets O_2 et O_4,
- l'émergence globale par croisement pour les objets O_1 et O_5.

Les dernières étapes pour finaliser l'évolution dans chaque cas sont présentées ci-après au travers toujours des mêmes exemples :

4.5.3.1. L'émergence locale

L'objet O_3 doit être rattaché à sa nouvelle classe *Senior-Chercheur* moyennant l'émergence locale du nouvel attribut *Responsabilité*. Une fois cette émergence validée par le concepteur, le processus lui recommande de préciser les méthodes pouvant manipuler cet attribut (section 4.5.3.3). Une fois la structure conceptuelle complétée, son évolution déclenchera l'adaptation de ses éventuelles sous-classes et de leurs instances. La gestion de ce type d'impacts rentre dans le cadre d'un processus de développement (section 4.6 -).

4.5.3.2. L'émergence globale

Le choix quant à l'endroit d'insertion d'une nouvelle structure conceptuelle est important. Il doit refléter la part d'influence de ses parents mais également la sémantique portée par la structure conceptuelle émergente :

1) - L'émergence globale par création directe de la structure conceptuelle

Les objets O_2 et O_4 provoquent chacun l'émergence d'une nouvelle structure conceptuelle. La première sera sous-classe de *Membre-Université* alors que la seconde sera sous-classe de *Temporaire*. Plusieurs stratégies peuvent prendre en compte ce type d'insertion et gérer les différents impacts. Cela est pris en charge dans le cadre d'un

processus de développement (section 4.6 -). Ces structures conceptuelles doivent également être complétées en termes de méthodes afin de spécifier le comportement de leurs instances (section 4.5.3.3).

2) - L'émergence globale par croisement

Concernant les objets O_1 et O_5, un croisement a été déclenché pour chacun des schèmes temporaires les représentant. Dans le cas de O_1, le processus a permis l'émergence de deux schèmes émergents : E4 et E5.

Dans le cas de O_2, le processus a permis l'émergence d'un schème émergent : E5 (qui est différent du schème émergent E5 précédemment cité, étant donné que les deux processus de croisement sont complètement distincts).

Le processus d'émergence globale par croisement se poursuit en recherchant l'endroit d'insertion de la nouvelle structure conceptuelle représentée par le schème émergent. Pour déterminer l'endroit d'insertion, il faut rechercher la classe existant dans la hiérarchie qui se rapproche le plus du schème émergent et qui présente le plus de similitudes en terme de constitution du GH. Une telle classe présenterait les caractéristiques nécessaires et voire même suffisantes pour représenter la super-classe de la structure conceptuelle émergente.

L'espace de recherche d'une telle classe n'est en aucun cas celui de la population. Il se cantonnera uniquement à l'ensemble des classes sélectionnées par la phase d'exploration et utilisées dans le croisement en phase d'exploitation.

En effet, elles sont les seules à avoir transmis un maximum de gènes aux schèmes émergents, il est logique qu'elles peuvent présenter des caractéristiques recherchées, contrairement aux classes qui ne sont pas adaptées au schème temporaire.

L'indéniable avantage est celui d'avoir un espace de recherche déjà cerné, et qui de surcroît est restreint aux seules classes potentiellement candidates. Dans le cas de l'objet O_1 et de l'objet O_5, il s'agit des classes `Enseignant`, `Ater` et `Senior-Chercheur`.

Ensuite, pour choisir parmi ces classes laquelle est plus adaptée à être la super-classe de la structure conceptuelle émergente, nous utilisons la *distance sémantique d.s*, définie en section 4.3.7. La d.s permet de détecter les classes les plus proches sémantiquement du schème émergent parmi la population initiale du croisement, et de déterminer par conséquent l'emplacement de la nouvelle structure conceptuelle dans la hiérarchie de classes.

3) - L'exemple de l'objet O_1

Nous devons calculer la d.s pour chacun des deux schèmes émergents :

– *Les d.s du schème émergent E4 :*

Le nombre de gènes spécifiés dans E4 est de 6 (gènes à '1').

	Schèmes												
Schème Émergent E4	1	1	1	0	1	0	1	0			1	#	d.s
Enseignant	0	0	0	1	1	0	0	0			0	0	1/6
Ater	1	1	#	0	1	1	0	1			0	0	4/6
Senior-Chercheur	1	1	1	0	0	0	1	0			1	#	5/6

Tableau 4-23: calcul de la d.s pour chaque schème permanent ayant participé au croisement pour E4

En conclusion, la classe Senior-Chercheur a la plus grande distance sémantique. De plus, le bloc de GH de E4 est le même que celui de Senior-Chercheur. Senior-Chercheur est donc candidate pour être la super-classe de la structure conceptuelle émergente véhiculée par E4. Elle exprime ainsi une catégorie de Senior-Chercheur.

Remarque : nous mettons en évidence à ce niveau une seconde utilité à la contrainte de blocs de gènes. Elle nous permet en effet de remarquer la similitude, la différence et voire même l'exclusion dans la constitution d'un schème émergent avec un schème permanent.

– *Les d.s du schème émergent E5 :*

Le nombre de gènes spécifiés dans E5 est de 7 (gènes à '1').

	Schèmes												
Schème Émergent E5	1	1	#	1	1	0	1	1			1	0	d.s
Enseignant	0	0	0	1	1	0	0	0			0	0	2/7
Ater	1	1	#	0	1	1	0	1			0	0	4/7
Senior-Chercheur	1	1	1	0	0	0	1	0			1	#	4/7

Tableau 4-24: calcul de la d.s pour chaque schème permanent ayant participé au croisement pour E5

La classe Ater et la classe Senior-Chercheur présentent la même distance sémantique. La classe Enseignant a la plus faible d.s, pourtant elle détient le même GS que le schème émergent E5.

Conclusion :

Avant d'aller plus loin, une question plus importante s'impose : lequel des deux schèmes émergents E4 et E5 sera le schème véhiculant la structure conceptuelle émergente ? En effet, l'émergence de la structure conceptuelle se fera sur un seul schème émergent.

Pour le schème émergent E4, la classe Senoir-Chercheur a la plus grande d.s. La classe Ater vient en seconde position car elle présente également une forte d.s. La structure conceptuelle serait donc a priori dans la catégorie des chercheurs.

Pour le schème émergent E5, les deux classes Ater et Senior-Chercheur ont la même d.s. La seule conclusion à laquelle nous pouvons arriver est que la structure conceptuelle émergente est une sous-classe, directe ou indirecte, de la classe Chercheur.

En l'état actuel, nous ne disposons pas encore des outils nécessaires pour faire un tel choix. Cela fait l'objet du chapitre suivant (chapitre 6).

4) - L'exemple de l'objet O_5

La distance sémantique est calculée entre le schème émergent et la population initiale du croisement. Le nombre de gènes spécifiés dans E5 est de 7 :

	Schème												
Schème Émergent E5	1	1	#	1	1	0	1	1	0	0	1	0	d.s
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	2/7
Ater	1	1	#	0	1	1	0	1	0	0	0	0	4/7
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	4/7

Tableau 4-25: calcul de la d.s pour chaque schème permanent ayant participé au croisement pour E5

Conclusions :

- `Enseignant` a la plus faible distance sémantique. Elle ne peut donc pas être une super-classe du schème émergent.
- `Ater` et `Senior-Chercheur` ont la même distance sémantique, et elle est supérieure à celle d'`Enseignant`. De plus, ces deux classes sont des sous-classes de `Chercheur`.

Les deux classes Ater et Senior-Chercheur ont la même d.s. La seule conclusion à laquelle nous pouvons arriver est que la structure conceptuelle émergente sera une sous-classe, directe ou indirecte, de la classe Chercheur, moyennant des mutations sur les attributs tels que *Modules*.

En l'état actuel, nous ne disposons pas encore des outils nécessaires pour faire un tel choix. Cela fait l'objet du chapitre suivant (chapitre 6).

5) - L'héritage multiple et l'émergence globale par croisement

Nous avons soulevé le fait que le choix quant à l'endroit d'insertion d'une nouvelle structure conceptuelle soit important. Il doit refléter la part d'influence de ses parents mais également la sémantique portée par la structure conceptuelle émergente. Il est plus logique qu'une structure conceptuelle qui prend la principale part de ses gènes d'une classe, soit dans la même branche dans la hiérarchie qu'elle.

Pour inférer une telle information, nous avons recours à l'exploitation des valeurs d'adaptation déjà calculé lors des phases d'exploration. Ainsi, lors d'un croisement, le parent ayant la plus forte Va va influer majoritairement le schème final. A plus forte

raison, si le schème émergent contient le GH du schème parent. Aussi, dans le cas d'une émergence globale, la structure conceptuelle émergente sera rattachée à sa branche.

Le cas particulier où les deux parents ont chacun une Va avoisinant les 50% (une Va comprise en 40% et 60%) au schème temporaire, peut voir la structure conceptuelle émergente créée par héritage multiple. Un autre critère permettrait d'avoir recours à l'héritage multiple sans ambiguïté est lorsque le GH de chaque schème parent se retrouve dans le schème émergent.

6) - Conclusions

Cependant, dans le cadre de l'émergence globale par croisement, le processus d'émergence tel qu'il a été défini gagnerait à être enrichi pour mener à bien les cas suivants :

- lorsqu'il y a le choix à faire entre plusieurs schèmes émergents car les Va étant identiques. Particulièrement si les schèmes émergents se retrouvent avec des blocs différents de GH, c'est-à-dire que différentes branches de la hiérarchie sont éligibles, tel le cas de O_1 ;
- lorsqu'il y a un choix à faire entre des classes pertinentes pour l'insertion d'une structure conceptuelle émergente, tel le cas de O_5. Les Va et les d.s obtenues à elles seules ne permettent pas de distinguer et de choisir la configuration finale de la structure conceptuelle émergente et de son lieu d'insertion dans la hiérarchie.

Ces limites rencontrées lors de l'analyse des résultats du croisement font l'objet du chapitre 6. Ces limites sont levées en ayant recours à des métriques.

4.5.3.3. Compléter le comportement spécifié au sein des structures conceptuelles

Les résultats de l'émergence sont proposés au concepteur. Celui-ci peut valider en choisissant la solution qui lui convient le mieux, s'il existe plusieurs solutions. Au sein de la solution choisie, il peut activer des mutations selon ses besoins. Le processus lui réclame en outre, de spécifier des méthodes pour pouvoir manipuler les attributs nouvellement introduits. Dans le cas de l'émergence de structures conceptuelles, en plus de la nécessité de définir ces méthodes, le processus crée automatiquement le constructeur et le destructeur par défaut.

Dans le cas ou le concepteur décline cette proposition de venir compléter les structures conceptuelles, le système lui rappelle, sous forme de messages d'alerte, le fait que les méthodes ne soient pas spécifiées, et ce dès que le système accédera à une de ces structures conceptuelles ou l'une de leurs instances. Ceci est valable pour toute émergence d'un nouvel attribut et d'une nouvelle structure conceptuelle.

Une autre situation se pose lorsque l'émergence globale par croisement est effectuée. Une nouvelle structure conceptuelle émergeant par croisement est, en gros, constituée d'attributs existants (et donc préalablement spécifiés au sein d'autres classes) et de

nouveaux attributs. Elle nécessite également d'être complétée en terme de spécification du comportement de ses instances (actuelles et à venir).

Pour les nouveaux attributs, la procédure est celle sus-citée. Pour les attributs existants, le processus détecte automatiquement, pour chacun de ces attributs, toutes les méthodes dans la classe qui le référencent dans leur signature. L'ensemble de ces méthodes est alors référencé par la nouvelle structure conceptuelle.

Une fois ainsi complétée 'par défaut', la structure conceptuelle est soumise à la validation du concepteur.

4.6 - Le développement

Nous avons introduit la notion de processus de développement dans le cadre de l'état de l'art (chapitre 3) et dans le cadre des concepts et notions du modèle (chapitre 4). Nous l'avons également défini de manière plus précise dans la section 4.2.4.1 du présent chapitre.

Nous considérons en effet qu'un processus de développement représente toute évolution en intention et toute évolution en extension d'une classe.

Dans le cadre de l'étude de l'évolution d'objets faite dans le chapitre 3, plusieurs stratégies ont été présentées. Nous en avions conclu, au vu des définitions d'un processus de développement et d'un processus d'émergence, que la majeure partie de ces stratégies d'évolution font du développement.

Dans le cadre des processus d'évolution que nous proposons dans notre modèle, nous définissons des processus d'évolution pour gérer l'émergence. Pour ce qui est des processus de développement (définition et différents cas présentés en section 4.2.4.1), nous considérons que nous pouvons utiliser l'une des stratégies d'évolution étudiées dans le chapitre 3. Elles sont en effet adaptées pour gérer le développement car elles assurent l'évolution par modification de classes. Nos choisissons pour notre étude une de ces stratégies, à savoir la correction de classes [Banerjee& 87] car elle présente une taxonomie complète d'opérations d'évolution.

Rappelons le fonctionnement général des processus de développement et d'émergence. La figure suivante est légèrement plus détaillée que celle présentée dans le chapitre 3 :

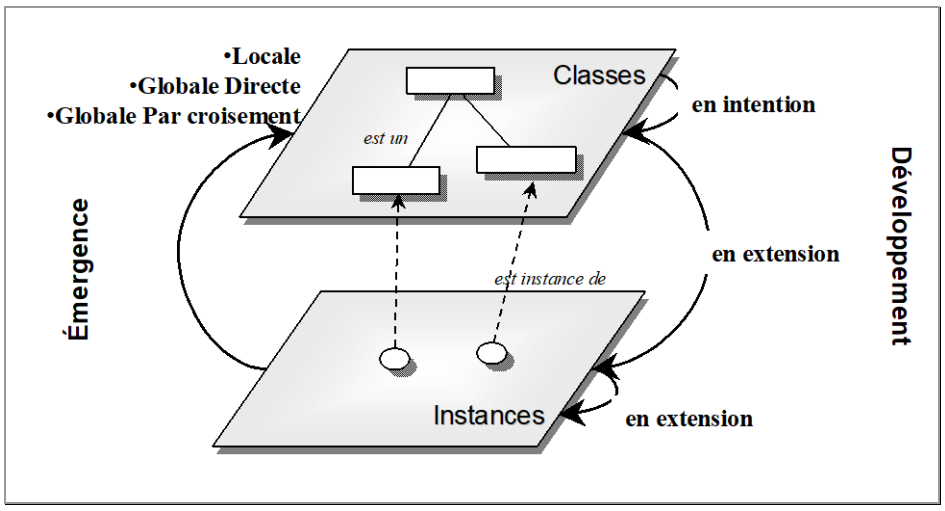


Figure 4-14: le Développement et l'Émergence : les deux processus d'évolution.

Outre sa définition propre, nous considérons qu'un processus de développement est déclenché de manière *directe* et/ou *indirecte* :

– de manière *directe* : lorsque l'évolution en intention ou en extension d'une classe survient directement sur l'entité concernée (classe ou instance) ;

– de manière *indirecte* : lorsque l'évolution en intention et en extension d'une classe survient pour gérer les conséquences d'un processus d'émergence. Lorsqu'un processus d'émergence locale, globale directe ou globale par croisement aboutit, les modifications doivent être répercutées au niveau de la hiérarchie des classes. La gestion des conséquences sur les autres entités concernées se fait dans le cadre du développement.

Dans ce cadre, nous proposons de laisser le choix au concepteur de choisir la stratégie de développement qui lui convient pour gérer les conséquences de l'émergence. Notre objectif est double :

– permettre l'intégration de notre modèle, moyennant la transcription de la hiérarchie de classes en prenant en compte les différents concepts proposés, principalement les GF, GH, GS et schème, au sein d'applications déjà existantes ;

– permettre au concepteur de continuer à travailler avec son environnement habituel et les stratégies d'évolution déjà présentes dans l'application.

4.7 - Le lien d'évolution

Le *lien d'évolution* a été introduit dans le chapitre 4. Il en a également été question en section 4.5.1.2 qui traite des exemples d'émergence locale, et en section 4.5.2.1.2, qui

traite des exemples de l'émergence globale par création directe de structures conceptuelles.

Ces premières introductions permettent de donner une idée de ce qu'est le lien d'évolution. Cependant, le lien d'évolution est défini et présenté en détails dans la présente section.

4.7.1. Introduction

L'émergence peut être menée à son terme :

– De manière *locale* : de nouvelles propriétés apparaissent au sein d'une classe existante ;

– De manière *globale* : une ou plusieurs structures conceptuelles émergent et sont insérées dans la hiérarchie de classes.

Mais quel que soit l'issue de l'émergence, tant qu'elle est concluante, les objets ayant provoqué ces processus d'émergence évoluent de leur classe initiale, dont ils sont détachés, vers les classes correspondantes, modifiées ou créées par émergence, auxquelles ils sont rattachés.

4.7.2. Définition du lien d'évolution

Le lien d'évolution est un lien sémantique représentée par une association réflexive dans le méta-modèle de GENOME, et porteur de propriétés. Lorsqu'un objet évolue d'une classe source vers une classe cible, un lien est créé de la classe source vers la classe cible. Ce lien est dit *lien d'évolution*. Le lien d'évolution est toujours orienté de la classe source vers la classe cible.

Une classe peut être la source d'aucun, d'un ou de plusieurs liens d'évolution. Une classe peut également être la cible d'aucun, d'un ou de plusieurs liens d'évolution. C'est ce qui est exprimé dans la portion suivante du méta-modèle de GENOME défini dans le chapitre 3 :

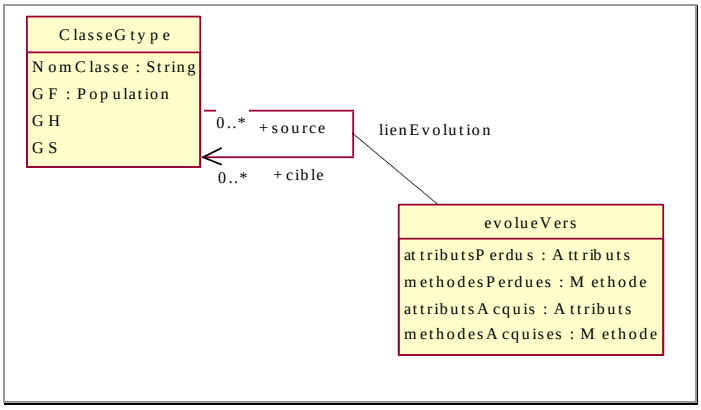


Figure 4-15: Lien d'évolution

Un lien d'évolution, en plus de la spécification de ses classes source et cible, porte la différence des ensembles d'attributs et de méthodes des deux classes. Il détient les références des attributs et des méthodes de la classe source perdus par l'objet, et les références des attributs et des méthodes de la classe cible acquis par ce même objet.

4.7.3. Utilité du lien d'évolution

Le lien d'évolution est le concept qui concrétise un de nos principaux objectifs, celui de permettre à un modèle d'apprendre un de ses comportements futurs et nouveaux : **permettre la capitalisation de nouveaux comportements et des nouveaux besoins**.

L'objectif du lien d'évolution est de matérialiser le fait que le modèle ait appris un nouveau comportement et une voie possible et connue d'évolution non seulement de ses instances, mais également de son modèle de classes.

La structure et la composition du lien d'évolution sont utilisées comme suit :

Lorsqu'un objet d'une classe évolue dynamiquement dans sa structure, au lieu de déclencher directement un processus d'évolution, il est préférable d'exploiter en premier lieu les différents liens d'évolution émanant de sa classe d'appartenance (c'est-à-dire qu'elle en est la source).

En effet, étant donné que le système a acquis auparavant la connaissance de comportements évolutifs des instances de cette classe, il est plus judicieux de l'exploiter. Cela se fait en analysant les informations portées par le lien d'évolution. Dans le cas de résultats infructueux, il est possible de conclure que la situation de changements est nouvelle, et par conséquent inconnue. Un processus d'évolution est alors déclenché pour tenter d'y répondre.

Comment analyser les liens d'évolution ?

Il s'agit de comparer tous les liens d'évolution émanant de la classe d'appartenance de l'objet, c'est-à-dire dont elle est la source. Ensuite, pour chaque lien, il faut comparer la nouvelle structure de l'instance avec les attributs référencés par le lien, pour détecter si l'instance a perdu les mêmes attributs ou un sous-ensemble des attributs référencés par

le lien comme perdus, et si elle a acquis les mêmes attributs ou un sous-ensemble des attributs de la classe cible et référencés par le lien comme attributs à acquérir.

La comparaison avec le lien d'évolution évite d'avoir à parcourir l'ensemble des classes à la recherche de leur adaptation. Cela permet d'optimiser l'espace et le temps de recherche.

Dans le cas où l'instance serait conforme à la définition du lien d'évolution, le processus d'évolution se charge de faire évoluer l'instance de sa classe d'appartenance vers la classe cible. L'instance *emprunte* une voie d'évolution déjà connue, à laquelle le modèle sait répondre.

Dans le cas où l'instance ne serait conforme à aucun lien d'évolution émanant de sa classe d'appartenance initiale, un processus d'évolution est déclenché dans sa première phase qu'est la phase d'extraction, pour se poursuivre sur la phase d'exploration et la phase d'exploitation (section 4.4 -).

4.7.4. Les graphes d'évolution

Le lien d'évolution est un lien porteur d'une sémantique particulière. Il n'a pas le même rôle qu'une simple association. Une association entre deux classes exprime une information statique qui existe entre les instances de ces deux classes. Le lien d'évolution est porteur d'une information d'évolution entre une classe et une autre. Il se situe dans une dimension différente de celle des autres types de liens que sont l'héritage, la composition et l'association. L'historique des liens d'évolution donne une information sur la manière dont le modèle a évolué. Ils ne sont consultés et activés que face à de nouvelles situations de changements. Pour ces raisons, les liens d'évolution sont organisés au sein de *graphes*. Chaque classe peut potentiellement être la racine d'un graphe de liens d'évolutions. Elle représenterait leur classe source. Donc, une population peut avoir plusieurs graphes de liens d'évolution. Ces graphes sont représentés, pour des raisons de lisibilité et pour éviter toute confusion avec la hiérarchie de classes, dans un plan parallèle au modèle de classes. Dans un graphe d'évolution, les nœuds sont les classes. Les arcs sont les liens d'évolution porteurs des gènes perdus et acquis lorsqu'une instance évolue de la classe source d'un arc vers la classe cible de ce même arc. Les arcs sont orientés pour exprimer dans quel sens se fait l'évolution. Les graphes des liens d'évolution sont illustrés dans la Figure 4-16.

4.7.5. Les exemples

L'émergence a pu être menée à terme pour les objets O_2, O_3 et O_4. Le résultat de l'émergence pour ces objets sont :

- *Pour O_2 :* l'émergence globale par création directe de la classe Ingénieur comme sous-classe de la classe Membre-Université ;
- *Pour O_3 :* l'émergence locale de l'attribut *Responsabilité* au sein du GS de la classe Senior-Chercheur ;

- *Pour O_4 :* l'émergence globale par création directe de la classe Post-Doc comme sous-classe de la classe Temporaire.

L'émergence n'a pas été complètement menée à terme pour les objets O_1 et O_5. Pour l'objet O_1, le croisement a mis en évidence deux schèmes émergents. Le choix doit être fait entre ces deux schèmes. Par contre pour l'objet O_5, le croisement a mis en évidence un seul schème émergent. Cependant, en appliquant la distance sémantique, deux classes présentent la même distance sémantique. Le choix doit être fait entre ces deux classes. Aussi, ces deux exemples seront traités dans le chapitre suivant (chapitre 6), y compris pour le lien d'évolution.

Dans la présente section, nous mettons en évidence les liens d'évolution pour les objets O_2, O_3 et O_4 :

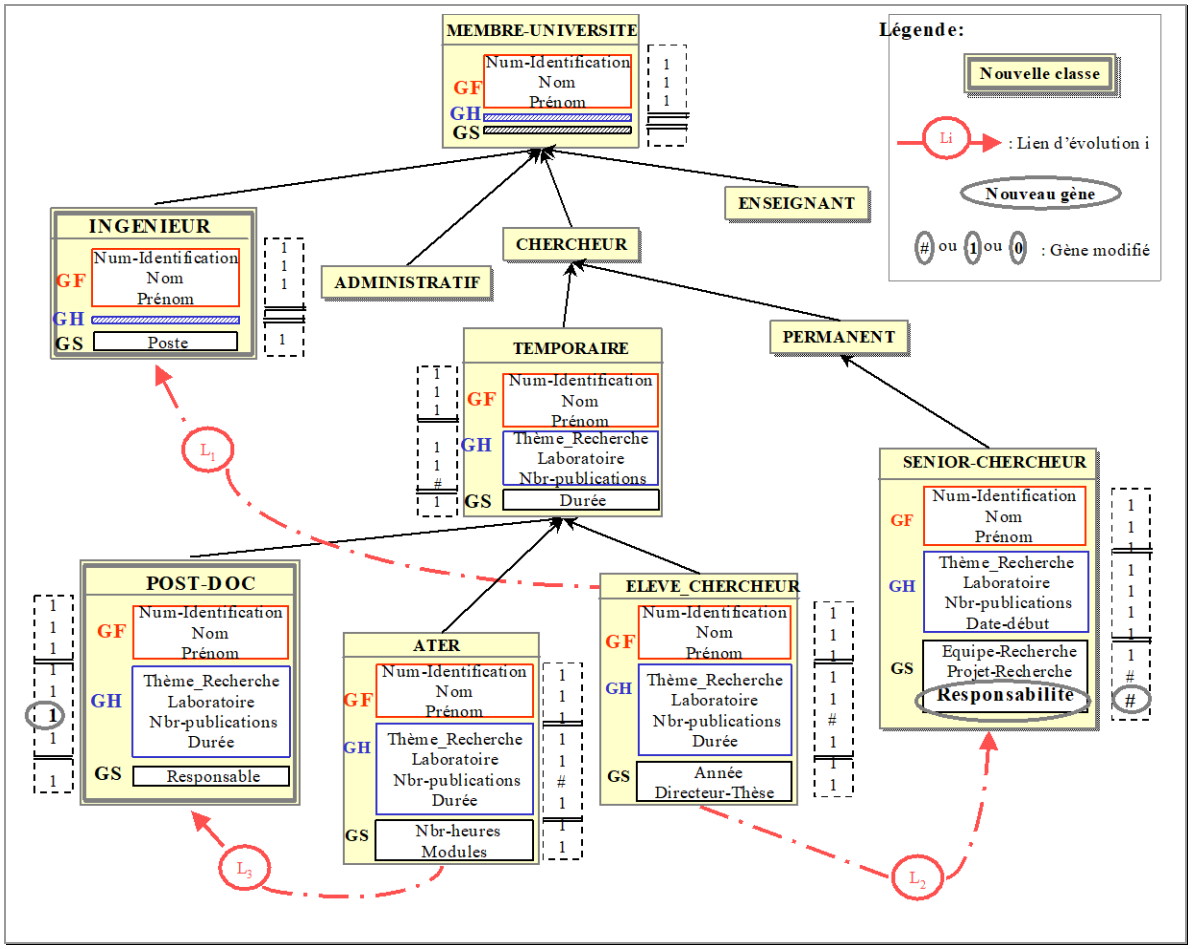


Figure 4-16: les liens d'évolution

La figure ci-dessus met en évidence la prise en compte des résultats de l'émergence issus de l'évolution des objets O_2, O_3 et O_4.

1) - Modifications apportées à la hiérarchie de classes

Les modifications au sein de la population Membre-Université sont comme suit :

– *Pour O_2 :* l'émergence globale par création directe de la classe Ingénieur comme sous-classe de la classe Membre-Université. La classe Ingénieur présente le même GF que sa population, ne présente pas de GH, et présente un GS constitué du nouvel attribut introduit par O_2, l'attribut *Poste*. Le schème permanent associé à Ingénieur a la même structure que la classe Ingénieur : la valuation des gènes du GF et la valuation du gène du GS. La première est imposée par la population. La seconde est proposée par défaut à 1 au concepteur. Dans ce cas, le concepteur décide de la laisser à 1.

– *Pour O_3 :* l'émergence locale du nouvel attribut *Responsabilité*, introduit par l'objet O_3, au sein du GS de la classe Senior-Chercheur. Le schème permanent associé à la classe Senior-Chercheur est modifié en conséquence : il s'agit d'introduire un nouveau gène à la position du nouvel attribut. Par défaut, il est proposé à 1 au concepteur. Celui-ci le modifie à # car il estime qu'il n'est pas obligatoire que tous les senior-chercheurs aient une responsabilité.

– *Pour O_4 :* l'émergence globale par création directe de la classe Post-Doc comme sous-classe de la classe Temporaire. La classe Post-Doc présente donc le même GF que toute la population. Elle présente également le même GH que les autres sous-classes de Temporaire. La classe Post-Doc présente également un GS constitué du nouvel attribut introduit par O_4, l'attribut *Responsable*. Le schème permanent associé à Post-Doc a la même structure que la classe Post-Doc : la valuation des gènes du GF, la valuation des gènes du GH et la valuation du gène du GS. La première est imposée par la population. La seconde subit une modification apportée par le concepteur sur le gène représentant le gène *Nbr-Publications*. Il le fait muter de # à 1. La troisième est proposée par défaut à 1 au concepteur. Dans ce cas, le concepteur décide de la laisser à 1.

2) - Spécifications des liens d'évolution

Trois liens d'évolution sont définis :

- L_1 : de Eleve-Chercheur vers Ingénieur,
- L_2 : de Eleve-Chercheur vers Senior-Chercheur,
- L_3 : de Ater vers Post-Doc.

Les liens d'évolution sont organisés dans un graphe qui leur est propre. Ainsi, la première représentation des liens d'évolution en Figure 4-16 est plus précise dans la Figure 4-17. Cette dernière met en évidence deux plans séparés : la hiérarchie de classes et le plan des graphes de liens d'évolution. Chaque nœud d'un graphe de liens d'évolution est connecté par une ligne en pointillés vers une classe de la hiérarchie de classes. En réalité, chacun de ces nœuds est juste une image de la classe afin de permettre de représenter séparément les deux plans. En fait, ces deux plans sont parallèles. Ils ont les mêmes nœuds (les classes), mais ces nœuds sont organisés différemment selon le plan : les arcs qui les relient dans la hiérarchie de classes sont des liens d'héritage, voire de références (association, composition) ; les arcs qui les relient dans le graphe d'évolution sont les liens d'évolutions :

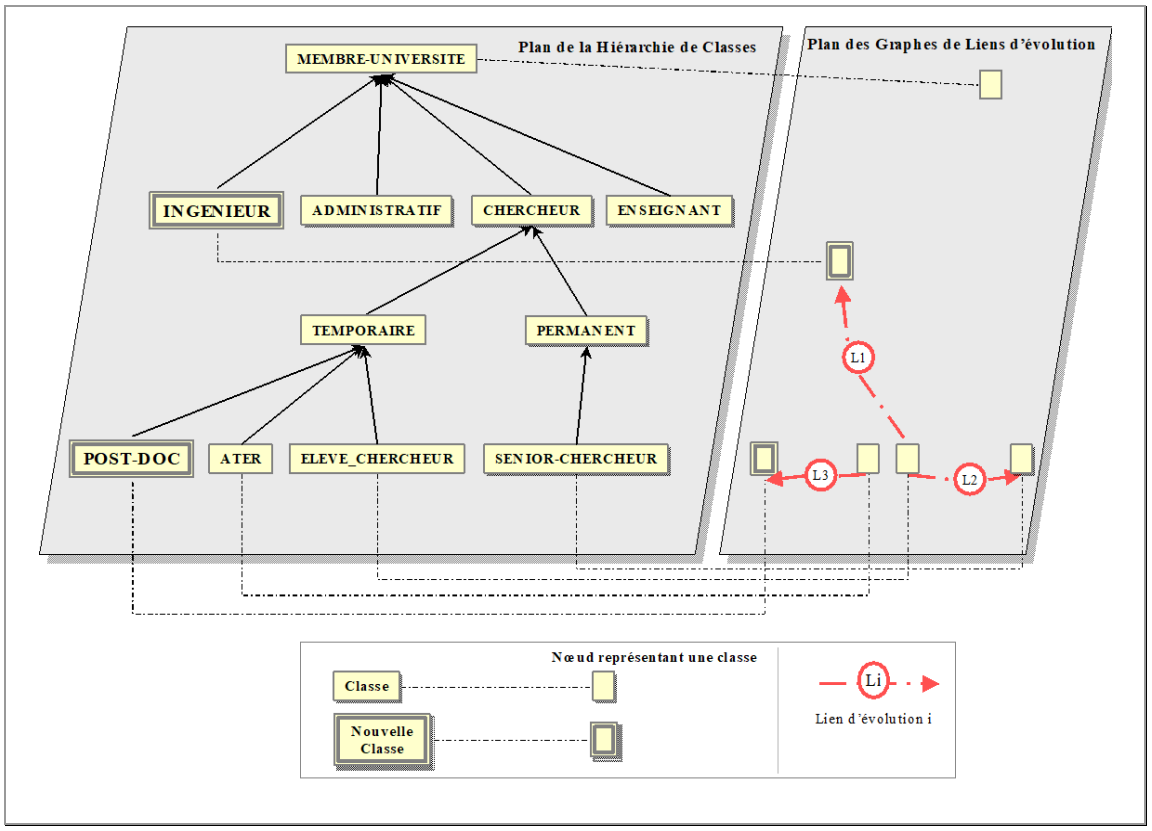


Figure 4-17: Hiérarchie de classes et Graphes de Liens d'évolution.

Ainsi, à une hiérarchie de classes représentant une population peut être associé un graphe de liens d'évolution.

Dans notre exemple, L_1, L_2 et L_3 sont les trois liens d'évolution qui ont été créés après avoir mené à bien les processus d'évolution de O_2, O_3 et O_4. Chacun des liens d'évolution est orienté de la source vers sa cible. Nous présentons ci-après le détail de chaque lien :

– L_1 est constitué des attributs suivants :

  - Theme-Recherche
  - Laboratoire
  - Nbr-Publications
  - Durée
  - Année
  - Directeur-Thèse
  + Poste

– L_2 est constitué des attributs suivants :

- Durée

- Année

- Directeur-Thèse

+ Date-début

+ Equipe-Recherche

+ Projet-Recherche

+ Responsabilité

– L_3 est constitué des attributs suivants :

- Nbr-heures

- Modules

+ Responsable

Les attributs précédés d'un signe - sont les attributs de la source qui sont perdus et les attributs précédés du signe + sont les attributs acquis de la cible. C'est la structure que respectera toute instance de la classe source qui évoluera de cette même classe source vers la classe cible.

4.8 - Conclusion

Tout au long de ce chapitre, les processus d'évolution de GENOME sont présentés et définis. Cette présentation a été abordée en plusieurs aspects :

– En premier lieu, nous précisons que les processus d'évolution sont déclenchés et opèrent entre deux états stables d'un modèle. Un état stable est un état où la dualité classe/instance est respectée, c'est-à-dire que toutes les modifications ont été apportées au niveau conceptuel ainsi que la gestion des impacts de ces modifications.

– Ensuite, les trois phases d'évolution de tout processus d'évolution sont introduites. Ce qui permet de définir plus clairement les processus d'évolution, étant donné que leurs principales étapes opératoires sont abordées. Ces phases sont les phases d'extraction, d'exploration et d'exploitation. Les processus de développement et d'émergence à proprement parler sont déclenchés lors de la dernière phase.

– Avant de préciser le détail opératoire de chaque phase, les opérateurs génétiques objets utilisés durant ces phases sont présentés et définis, ainsi que deux mesures. Les opérateurs sont la *sélection*, la *reproduction*, la *mutation* et le *croisement*, qui constitue le cœur du processus d'émergence globale. Les mesures sont la valeur d'adaptation Va et la distance sémantique d.s. Cette section récapitule à la fin l'utilisation des opérateurs dans chacune des trois phases.

– Le détail de chaque phase est abordé ensuite dans des automates. Nous associons à chaque phase un automate qui est décrit selon le formalisme des diagrammes d'états-transitions d'UML [Muller 97]. Chacun de ces automates est illustré par des exemples. Le processus d'émergence est ensuite détaillé au travers de ses différents types : émergence locale, émergence globale par création directe de la structure conceptuelle et émergence globale par croisement.

– Cette section est suivie d'une section consacrée à une discussion sur l'émergence, mettant en évidence les apports de l'émergence, mais également certaines limites. Ces dernières seront abordées et levées dans le chapitre suivant.

– Ce chapitre se termine par une présentation des liens d'évolution. Les liens d'évolution, organisés en graphes, permettent de représenter le comportement d'évolution des instances du modèle et permettent également d'apprendre un nouveau comportement évolutif des instances. Le système aura recours à ces liens pour toute situation de changement future. Si aucun lien ne le détient, alors un processus d'évolution est déclenché et ses trois phases sont ainsi exécutées.

Chapitre 5 : GENOME : LES METRIQUES POUR CONTROLER L'EMERGENCE

5.1 - Introduction

Nous traitons dans le présent chapitre les limites rencontrées lors de l'analyse des résultats du croisement dans le chapitre précédent. L'approche que nous adoptons pour pouvoir y répondre s'appuie sur l'utilisation de métriques. Notre objectif est de mesurer pour mieux contrôler ce qui émerge. Quelles sont donc ces limites et au vu de quoi pouvons-nous proposer des métriques ?

5.1.1. Les limites de l'émergence

Le chapitre précédent a permis de présenter les différents processus d'évolution. Ils ont été illustrés sur des exemples d'évolution d'objets. La plupart des exemples ont pu être menés à terme. Seules deux situations se sont retrouvées non résolues. Il s'agit de processus d'émergence globale par croisement. Le croisement a été mené à terme. Dans l'un des cas, il a permis l'émergence de deux schèmes émergeants mais aucun élément ne permet d'en préférer l'un à l'autre car les Va sont identiques. Dans l'autre cas, le croisement a permis l'émergence d'un schème émergent, mais l'application de la distance sémantique ne permet pas de dégager une super-classe potentielle pour l'abstraction véhiculée par le schème émergent. En effet, deux classes présentent la même d.s. La seule utilité de la d.s dans cet exemple a été d'éliminer une classe dont la d.s était trop faible pour être une super-classe, à savoir la classe Enseignant.

Nous rappelons ci-après les conclusions du précédent chapitre. Celles-ci mettent en évidence la nécessité d'enrichir l'émergence lors de l'analyse des résultats du croisement, et particulièrement dans les cas suivants :

- lorsqu'il y a un choix à faire entre plusieurs schèmes émergents. Particulièrement si les schèmes émergents se retrouvent avec des blocs

différents de GH, c'est-à-dire que différentes branches d'une hiérarchie sont éligibles, tel le cas de O_1 pour lequel le schème émergent E4 présente un bloc de gènes propre à la catégorie des enseignants, alors que le schème émergent E5 présente plutôt des blocs de gènes propres à la catégorie des chercheurs,

- lorsqu'il y a un choix à faire entre des classes pertinentes pour l'insertion d'une structure conceptuelle émergente, tel le cas de O_5, pour lesquelles au moins deux classes peuvent a priori être sa super-classe : Senior-Chercheur et Ater.

Nous en concluons que la Va et la d.s ne suffisent pas toujours pour mener à terme l'émergence. Dans les exemples précités, l'analyse des différentes Va et des différentes d.s permet de tirer quelques conclusions. La principale conclusion est que le schème émergent dans le cas de l'objet O_5 a des similitudes avec la catégorie des chercheurs. Cependant, les résultats de leur analyse ne permettent pas d'aller plus loin.

Nous en concluons que la Valeur d'adaptation, Va, et la distance sémantique, d.s, représentent des valeurs *nécessaires* pour l'exécution et l'achèvement de l'émergence mais, dans certaines situations, elles ne représentent pas des valeurs *suffisantes* principalement pour l'achèvement de l'émergence, tels que pour distinguer et choisir :

- parmi plusieurs schèmes émergents,
- parmi des classes identiquement proches du schème émergent.

Dans les deux cas, seule une analyse plus approfondie de la constitution des schèmes émergents et leur comparaison avec les classes éligibles peuvent permettre de statuer et de choisir parmi les solutions potentielles : le choix de schème émergent et/ou le choix de super-classe(s).

5.1.2. Des métriques au service de l'émergence

Les limites rencontrées pour l'achèvement de l'émergence globale par croisement nous incitent à vouloir analyser de manière plus approfondie les différents résultats d'un croisement : les schèmes émergents lorsqu'il y en a plusieurs pour un même croisement, et la comparaison du schème émergent avec les schèmes permanents ayant participé au croisement.

De manière plus précise, nous jugeons nécessaire d'avoir des informations sur la cohésion d'un schème émergent aussi bien de manière interne qu'externe, ainsi que la présence ou non de contradictions non seulement au sein d'un schème émergeant mais également par rapport aux schèmes permanents ayant participé au croisement ainsi que par rapport au modèle (au vu des contraintes et des invariants).

Nous optons pour l'utilisation de *métriques*. Ces dernières offrent l'avantage d'être calculées et évaluées de manière automatique et sont quantifiables. Ces caractéristiques

s'accordent avec notre principal objectif, à savoir celui de fournir un modèle de simulation d'évolution dynamique en cherchant et en proposant diverses voies d'évolution. Les caractéristiques précitées des métriques s'accordent avec notre approche pour deux principales raisons, qui constituent en fait des similitudes :

– Les métriques peuvent être évaluées automatiquement. Un processus d'évolution s'exécute automatiquement tout au long de ses trois phases. Une première utilisation de valeurs automatiquement calculées est celle de la Va et de la d.s. Cet automatisme permet de balayer rapidement l'espace des solutions, d'éliminer les moins pertinentes, et de proposer les plus intéressantes au concepteur ;

– Les métriques sont plus aisément applicables sur des données qui peuvent être évaluées sans ambiguïté. C'est justement la principale caractéristique et le principal rôle d'un schème, qu'il soit permanent, temporaire, enfant ou émergent. Sa structure constituée de gènes référençant des attributs et leurs valeurs restreintes (0, 1 ou #) permettent sans encombre d'effectuer des mesures principalement pour évaluer et relever aussi bien les similitudes, les rapprochements, les écarts et les contradictions.

Aussi, en mesurant de manière adéquate, nous pouvons mieux contrôler l'émergence et ses divers impacts. Pour pouvoir mesurer correctement, nous devons déterminer clairement ce que nous voulons faire et comment le faire. Nous devons également déterminer à quel endroit utiliser quelle métrique. Nous adoptons pour cela le principe général de la technique dite GQM pour Goal-Question-Metric [Basili& 84].

Nous considérons que la Va et la d.s sont des *métriques* d'un premier degré de mesure et d'analyse. Nous proposons, pour les raisons sus-citées, d'utiliser des métriques à un second degré de mesure et d'analyse pour étudier :

- la constitution de chaque schème de manière interne ;
- la cohésion du schème par rapport à la hiérarchie de classes existantes.

Nous nous inspirons de quelques métriques orientées objet standards [Coulange 98]. Nous en adoptons et adaptons quelques-unes unes à notre modèle. Nous en proposons également d'autres que nous introduisons pour prendre en compte les spécificités des concepts du modèle.

Nous présentons en premier lieu les métriques orientées objet dans leur définition, leur contexte d'utilisation ainsi que quelques métriques orientées objet standards.

5.2 - Les Métriques Orientées Objet

Les premières métriques ont été introduites en informatique pour la mesure de logiciels. Un exemple de métrique qui revient souvent dans le jargon d'un programmeur est la fameuse « X lignes de code ». L'effort de mesure ne s'y est pas uniquement cantonné. L'action de mesurer s'est appliquée à tous les niveaux

(application, développement, conception…) et à la plupart des composants (opérateurs, opérandes, fonction, flots de contrôle de programmes, effort de développement…) aussi bien en termes de quantité que de qualité. La métrologie est définie comme des « techniques qui permettent de vérifier de façon 'mathématique' des propriétés sur un logiciel » et elle s'est également étendue à l'approche objet [Coulange 98].

5.2.1. Définitions

Nous présentons une première définition générale de l'action de mesurer avant de parler de métrique dans l'approche objet :

– Définition générale : l'action de mesurer est le processus par lequel des nombres ou des symboles sont assignés à des *attributs* d'*entités* du monde réel d'une manière pouvant les décrire suivant des règles clairement définies.

– Métrique : une métrique est une mesure effectuée sur un logiciel ou sur un modèle d'analyse ou de conception. Une métrique est considérée comme un *identificateur* de l'entité mesurée.

5.2.2. Caractéristiques d'une métrique

Hormis des métriques objets de base (les premières sont les six métriques proposées par Chidamber et Kermer [Chidamber& 91] développées pour mesurer la qualité de la conception des logiciels orientés objet ; [Rocacher 88]) qui rassemblent plus ou moins un certain consensus, la définition d'une métrique est une tâche délicate. Elle doit en effet comporter et répondre à plusieurs éléments : sur quelle(s) entité(s) porte la métrique, comment la mesurer, comment interpréter la mesure, à quoi la comparer, quelles sont les limites qui permettent de dire que la mesure est 'bonne' ou 'mauvaise' ?…

Les caractéristiques auxquelles doit répondre une métrique sont rappelées dans [Coulange 98]. Parmi les plus significatives, une mesure doit :

– Suggérer une norme, une échelle de valeurs et des bornes. Il doit être possible de comparer deux mesures du même type.

– Avoir une signification.

– Se rapporter aux propriétés du processus, du modèle ou du système considéré.

– Suggérer une stratégie d'amélioration.

– Être simple.

Ces caractéristiques sont effectivement importantes, mais il n'y a pas de manière de les définir et de les assurer. La définition complète d'une métrique reste donc entièrement à l'appréciation de ceux qui la définissent.

Dans le cadre de logiciels orientés objet, les métriques se définissent selon un contexte précis d'utilisation :

5.2.3. Contextes d'utilisation d'une métrique et exemples

La signification d'une métrique dépend de son contexte d'utilisation. Généralement, trois contextes d'utilisation pour la qualité de la conception des logiciels orientés objet sont définis :

- contexte d'*application*,
- contexte de *classe*,
- contexte de *fonction*.

L'interprétation des métriques est souvent un indicateur et ne peut le plus souvent que renseigner et attirer l'attention du concepteur ou de l'utilisateur sur l'entité mesurée. Nous définissons succinctement chaque contexte d'utilisation de métriques. Nous présentons quelques exemples pour chaque contexte dans leur nom, leur définition et leur éventuelle utilité [Coulange 98]. La plupart de ces exemples sont des métriques proposées par [Chidamber& 96]. Elles ont également donné lieu à des variantes.

5.2.3.1. Contexte d'Application

Le contexte *Application* concerne tout le code source d'une application. Les métriques ont une portée globale. Voici quelques exemples :

Nom de la métrique	Signification	Utilité
Nombre de niveaux du graphe d'appel	Nombre de niveaux dans le graphe représentant les appels entre les méthodes au sein d'une application.	Un nombre élevé montre une hiérarchie trop forte
Nombre de niveaux dans le graphe d'héritage	Nombre de niveaux dans le graphe représentant l'héritage entre classes	Un nombre élevé rendra une application difficile à comprendre

Tableau 5-1: des exemples de métriques de contexte d'application.

5.2.3.2. Contexte de Classe

Le contexte *Classe* concerne les informations présentes au sein d'une classe. Voici quelques exemples [Chidamber& 91] :

Nom de la métrique	Signification	Utilité
Nombre pondéré de méthodes	Donne une idée de la complexité des méthodes d'une classe.	Elle peut être un indicateur de maintenabilité ou de réutilisabilité selon la formule de calcul adoptée. Plus elle est élevée, plus la classe est difficile à comprendre et donc à maintenir.
Manque de cohésion entre méthodes	Donne le nombre d'ensembles disjoints d'attributs dans une classe.	Une grande cohésion montre une bonne encapsulation. Une faible cohésion montre que la classe pourrait être divisée en plusieurs sous-classes.
Réponse pour une classe	Donne le nombre de méthodes pouvant être sollicitées en réponse à un message.	De mauvaises valeurs veulent peut être dire que les traitements ne sont pas dans les bonnes classes.
Couplage entre objets	Est un indicateur de dépendance d'une classe vis-à-vis des autres classes.	Plus elle est élevée, plus une classe est liée aux autres classes. Elle est alors difficile à réutiliser et à faire évoluer.

Tableau 5-2: des exemples de métriques de contexte de classe.

5.2.3.3. Contexte de Fonction

Le contexte *Fonction* concerne les informations présentes dans le corps d'une fonction. Voici quelques exemples :

Nom de la métrique	Signification	Utilité
Nombre d'instructions	Donne le nombre d'instructions exécutables situées entre la ligne de début et la ligne de fin d'un sous-programme.	C'est un indicateur de maintenabilité. Mais elle est difficile à apprécier.
Nombre cyclomatique	Donne le nombre de chemins linéaires indépendants dans un graphe fermé.	C'est un indicateur de l'effort de compréhension et de tests à faire.
Nombre de chemins	Donne le nombre de chemins d'exécution non cycliques.	Donne une idée précise du nombre de tests nécessaires.
Nombre de niveaux	Donne le nombre de niveaux d'imbrication des structures de contrôle dans une fonction.	Plus il est élevé, plus la compréhension de l'algorithme de la fonction est difficile.

Tableau 5-3: des exemples de métriques de contexte de fonction.

Il apparaît au vu des exemples cités, que les métriques ne permettent pas forcément de donner une information suffisamment précise sur l'élément mesuré qui permettrait d'agir directement. Mais elles peuvent représenter de précieux indicateurs et aider par conséquent à choisir ou à améliorer l'élément mesuré ainsi que ceux avec lesquels il peut interagir.

Dans la section suivante, nous abordons les métriques dans notre modèle. Nous expliquons comment nous procédons pour dégager des métriques nécessaires au contrôle de l'émergence, principalement les résultats du croisement, avant de présenter les différentes métriques et leur illustration.

5.3 - les métriques dans notre modèle

Nous introduisons des métriques pour compléter les processus d'évolution dans la dernière phase : la phase d'exploitation, et plus particulièrement pour l'émergence globale par croisement.

Nous avons expliqué l'utilité des métriques pour son achèvement. Nous savons que le schème émergent doit être analysé en détail dans sa constitution et par rapport aux autres schèmes. Nous devons donc déterminer exactement où et comment utiliser des métriques.

5.3.1. Détection des métriques nécessaires grâce à la technique GQM

Nous adoptons pour cela la technique GQM [Basili& 84] :

5.3.1.1. Définition de la technique GQM

La technique GQM propose une démarche pour déterminer et situer de manière précise à quelle étape dans un processus quelconque (conception, évolution, maintenance…), l'utilisation d'une métrique peut être utile. Cette approche est approuvée comme étant efficace pour sélectionner et implémenter des métriques. Son nom Goal-Question-Metric résume sa démarche. Celle-ci consiste à :

– Définir les *objectifs* (Goal) à atteindre ;

– Se poser les *questions* (Question) qui peuvent être décomposées en sous-questions jusqu'à obtenir des réponses suffisamment terminales pour être *quantifiées* (Metric).

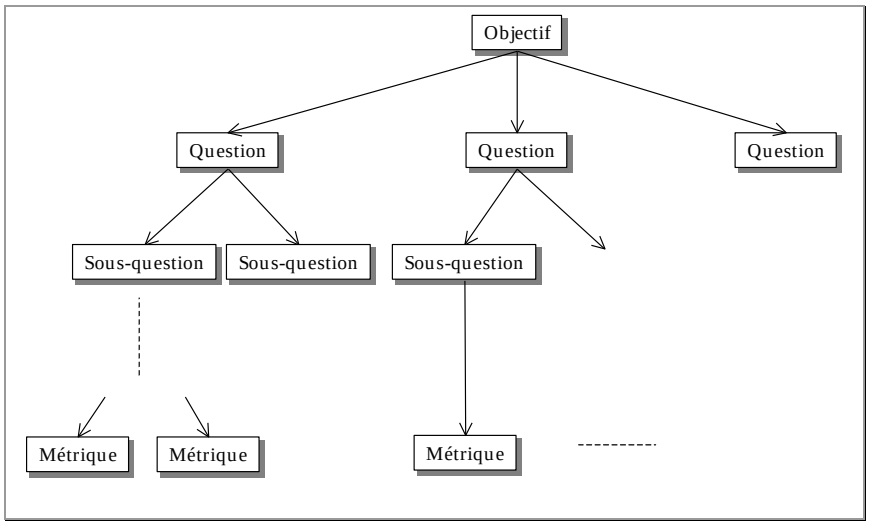


Figure 5-1: démarche descendante de la technique GQM.

Le résultat de l'application de la technique GQM est la spécification d'un ensemble de mesures en adéquation avec les objectifs. Pour que la technique soit efficace, il est nécessaire de déterminer avec précision pour chaque objectif :

– La caractéristique de la qualité associée à l'objectif ;

– L'objet (ou le processus) sur lequel porte l'objectif ;

– Le point de vue adopté ;

– Le but de l'objectif.

Certes, cette démarche ne dit pas comment déterminer et distinguer de manière détaillée l'application des étapes, mais nous estimons qu'en ayant une connaissance suffisante du système étudié, et en la combinant avec cette démarche, cela peut nous permettre de cerner les parties d'un processus d'évolution et de dégager les aspects susceptibles d'être mesurés.

5.3.1.2. Première application de la technique GQM pour un processus d'évolution de GENOME

Une première application de la technique sur un processus d'évolution nécessite une première décomposition. Nous appliquons cette décomposition naturellement sur les trois phases de tout processus d'évolution. Nous appliquons les étapes sur le déroulement d'un processus d'évolution tout en gardant en vue que le principal objectif est la simulation de l'évolution d'un modèle suite à une situation de changement imprévue sur une ou plusieurs instances :

Phase	Objectif(s)	Question	Nécessité de Mesurer (O/N) ?	Métrique
Extraction	Extraire le schème temporaire de l'instance ayant évolué	-	N	-
Exploration	trouver une population adaptée	son adaptation au schème temporaire	O	Va
	trouver des classes adaptées	leur adaptation au schème temporaire	O	Va
Exploitation	détecter le processus de développement	une classe adaptée au schème temporaire	O	Va
	détecter le processus d'émergence : - locale - globale	une classe adaptée au schème temporaire + de nouveaux attributs	O	Va
		plusieurs classes partiellement adaptées au schème temporaire + éventuellement de nouveaux attributs	O	Va + d.s

Tableau 5-4: une première application de la technique GQM.

Cette décomposition reste assez générale. Pour chaque phase, nous déterminons (en fait nous rappelons) le principal *objectif*, la principale *question* qui précise cet objectif. Avant de parler de métrique pour une question, nous estimons qu'il est en premier lieu nécessaire de s'enquérir de la pertinence d'utiliser ou non une métrique. Si elle est nécessaire, nous précisons alors la *métrique*.

Nous arrêtons la décomposition à ce stade pour introduire les métriques dans notre modèle.

5.3.2. Les métriques et GENOME

Nous nous inspirons de quelques métriques orientées objet et nous en proposons de nouvelles.

Les métriques orientées objet dont nous nous inspirons sont celles dont les contextes d'utilisation sont semblables aux nôtres. Comme nous nous intéressons à l'évolution de produit dans ses différentes composantes (classes, hiérarchies de classes, instances), nous nous intéressons aux métriques portant sur le produit : les métriques de contexte classe et les métriques de contexte application.

Quant aux métriques que nous proposons, nous déterminons en premier lieu les contextes d'utilisation pertinents pour les processus d'évolution et par conséquent pour le modèle via les concepts qui sont manipulés par les processus d'évolution. Étant donné que nous voulons analyser des structures conceptuelles émergentes[6] aussi bien dans leur constitution interne que dans leur positionnement par rapport à la hiérarchie de classes, nous dégageons deux contextes d'utilisation de métriques :

– le *contexte intra-structure* : définit des métriques qui donnent des indications sur la constitution d'un schème émergent ;

– le *contexte inter-structures* : définit des métriques qui donnent des indications sur la corrélation d'un schème émergent avec son environnement : hiérarchie de classes et applications.

Nous présentons alors la deuxième application de la technique GQM. Elle constitue une décomposition détaillée de la précédente.

5.3.2.1. Deuxième application de la technique GQM pour un processus d'évolution de GENOME

La seconde application que nous faisons suit une décomposition de chaque question posée. Cette décomposition se fait au vu des objectifs et des questions préalablement dégagés (Tableau 5-4) tout en prenant en compte les deux contextes *intra-structure* et *inter-structures* lorsque cela s'avère nécessaire. Lors de cette seconde décomposition, nous introduisons, en plus de la précédente décomposition, des sous-questions si nous le jugeons nécessaire ainsi que des commentaires.

Nous développons plus en détails les questions sur la nécessité de mesurer sur la **phase d'exploitation** dans le tableau suivant. La lecture de ce tableau est rattachée aux sections suivantes :

[6] On crée une structure conceptuelle émergente après avoir définitivement choisi un schème émergent.

Processus	Question	Sous-questions	Métriques	Commentaires
Développement	une classe adaptée au schème temporaire et aucun nouvel attribut	-	Va	Traité dans la littérature (cité dans chapitre 3).
Émergence Locale	une classe adaptée au schème temporaire et de nouveaux attributs introduits par l'instance	-	Va	Traité dans chapitre 5
	plusieurs classes adaptées au schème temporaire et de nouveaux attributs introduits par l'instance		Va+d.s	Traité dans chapitre 5.
Émergence Globale	*Contexte intra-structure :* par rapport à la constitution du schème émergent	Le schème émergent le plus *significatif* ?	S = écart entre un schème émergent et l'instance	Le schème le plus proche de l'instance (complète la d.s)
		Le schème émergent le moins *contradictoire* ?	C_t = nombre de contradictions entre attributs et blocs d'attributs	Le plus faible dénote le moins de contradictions
		Le schème émergent le plus *cohérent* ?	C_h = nombre d'attributs simples + nombre de contradictions entre attributs + nombre de blocs d'attributs + nombre de contradictions entre blocs d'attributs	Le plus faible dénote le moins d'incohérences
	Contexte inter-structures :	*Parents* du schème émergent dans la hiérarchie ?	Détection de super-classe(s)	C'est la distance sémantique

	Le schème émergent par rapport aux autres classes de la hiérarchie	*Contradiction* du schème émergent par rapport aux parents ?	C_d = Nombre d'attributs contradictoires + Nombre de blocs contradictoires ou contenant des contradictions	Le plus faible dénote le moins de contradictions
		Couplage du schème émergent avec les autres classes ?	C_p = somme des appels et références vers et des autres classes	Le plus faible dénote le minimum de couplage
		Profondeur du schème émergent dans le graphe ?	P_d = position dans le graphe d'héritage	Plus il est élevé, plus il y aura de réorganisation

Tableau 5-5: application approfondie de la technique GQM.

Nous remarquons que la notion de contexte est utile uniquement dans le cadre de l'émergence globale, et plus particulièrement pour l'émergence globale par croisement. Par conséquent, les métriques que nous proposons se départagent en :

- des métriques de contexte intra-structure

et

- des métriques de contexte inter-structures.

Elles se définissent comme suit :

5.3.2.2. Métriques de Contexte Intra-structure

1) - Définition

Les métriques de contexte intra-structure fournissent des indications sur la constitution interne d'un schème émergent. L'objectif est principalement de mesurer les écarts et les contradictions qu'il peut y avoir au sein d'un schème émergent.

Un schème émergent peut être constitué d'attributs simples et de blocs d'attributs. Un attribut simple est un attribut représenté par un gène présent dans un schème sans être rattaché à un bloc de gènes. Un bloc d'attributs est un bloc de gènes GH ou GS transmis au schème émergent tout au long du croisement par son schème parent dominant. Un gène appartenant à un bloc de gènes GH ou GS ne peut pas être dissocié des autres gènes constitutifs du même bloc (la contrainte de blocs de gènes).

Aussi, les métriques de contexte inter-structures considèrent les attributs simples et les blocs d'attributs de la même manière, c'est-à-dire comme un attribut indivisible. La seule distinction faite entre un attribut simple et un bloc d'attributs est que le premier n'est pas associé à d'autres attributs alors que le second peut être associé à un autre

bloc. En effet, un bloc de gènes GH peut être associé à un bloc GS et inversement, les deux formant une structure conceptuelle présente dans la population de classes. Par exemple, la classe Permanent est définie au travers du bloc GH constitué de *Thème-Recherche*, *Laboratoire*, *Nbr-Publications* **et** du bloc GS constitué de *Date-début* ; alors que pour la classe *Chercheur*, il y a uniquement le bloc GS.

Nous définissons trois métriques intra-structure :

2) - Métrique de Signification S

Cette métrique est définie pour privilégier un schème émergent qui serait le plus significatif par rapport à l'instance ayant évolué. En d'autres termes, le schème émergent le plus proche du schème temporaire.

Sa valeur est déterminée en faisant le décompte des gènes du schème émergent qui sont absents du schème temporaire. La métrique S est donc l'écart en nombre de gènes entre le schème émergent et le schème temporaire.

Elle est notée :

$$S(SE) = \sum_{i=1}^{n} S_i$$

avec : n : la longueur des deux schèmes temporaire et émergent

S_i : la signification locale sur le gène en position i définie par :

$$S_i = \begin{cases} 1 & si \quad \left((SE_i = 1) \vee (SE_i = \#)\right) \wedge (ST_i = \#) \\ 0 & \text{sinon} \end{cases}$$

avec : SE : Schème Émergeant

ST : Schème Temporaire

SE_i : gène à la position i de SE

ST_i : gène à la position i de ST

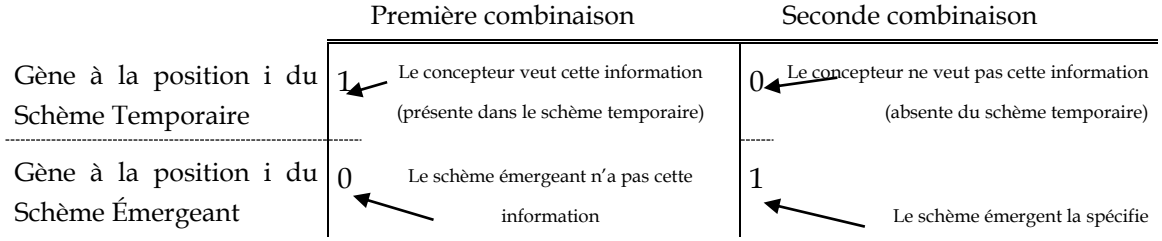
La métrique S est complémentaire de la d.s.

Interprétation : le schème émergent présentant la plus faible signification S est le plus proche du schème temporaire, donc de l'instance ayant évolué.

3) - Métrique de Contradiction C_t

Cette métrique est définie pour détecter d'éventuelles contradictions entre l'instance ayant évolué et un schème émergent, donc les éventuelles contradictions entre le schème temporaire et un schème émergent.

Une contradiction est détectée à chaque fois que pour le même gène, sa valeur dans le schème temporaire et sa valeur dans le schème émergent sont contradictoires, c'est-à-dire pour tout couple de valeurs (1,0) ou (0,1). En effet, une telle combinaison pose problème :



La métrique C_t est donc le nombre de contradictions entre le schème temporaire et le schème émergent.

Elle est notée :

$$C_t(SE) = \sum_{i=1}^{n} C_{t_i}$$

avec : n : la longueur des deux schèmes temporaire et émergaent

C_{t_i} : la contradiction locale sur le gène en position i définie par :

$$C_{t_i} = \begin{cases} 1 & si \quad ((ST_i = 1) \wedge (SE_i = 0)) \\ & \quad \vee ((ST_i = 0) \wedge (SE_i = 1)) \\ 0 & \sin on \end{cases}$$

avec : SE : Schème Émergeant

ST : Schème Temporaire

SE_i : gène à la position i de SE

ST_i : gène à la position i de ST

Interprétation : la plus faible C_t dénote le moins de contradictions. Plus elle est faible, plus le schème émergent est éligible.

4) - Métrique pour la Cohérence C_h

Cette métrique est définie pour la cohésion entre les attributs (donc les gènes) constitutifs d'un schème émergent. Les deux précédentes métriques prennent en compte le schème temporaire, contrairement à celle-ci.

Sa valeur est déterminée en faisant le décompte de gènes indépendants (nous entendons par gène indépendant un gène représentant un attribut simple ainsi qu'un bloc de gènes qui n'est associé à aucun autre), ainsi que le nombre de contradictions entre attributs (simples ou en blocs). Ces contradictions sont celles définies par le concepteur et présentes dans le modèle. Par exemple, les attributs *Date-début* et *Durée* sont considérés comme contradictoires par le concepteur. Ils ne doivent donc pas être en même temps présents dans une entité (structure conceptuelle, schème, instance), de la même manière que le bloc GS d'un permanent ne peut pas être présent en même temps que le bloc GS d'un temporaire.

La métrique C_h est donc le nombre d'attributs simples, le nombre de blocs d'attributs et le nombre de blocs d'attributs indépendants.

Elle est notée :

$$C_h(SE) = \sum \text{attributs simples} + \sum \text{contradictions entre attributs simples} + \sum \text{nombre de blocs d'attributs} + \sum \text{contradictions entre blocs d'attributs}$$

Interprétation : la plus faible C_h dénote le moins d'incohérences. Une trop forte valeur de C_h révélerait une constitution trop disloquée du schème émergent et donc peu de cohésion interne. Cela peut donc dénoter une mauvaise encapsulation. Il sera difficile de spécifier la structure conceptuelle au sein d'une classe, et encore plus difficile de l'insérer dans la hiérarchie de classes représentant sa population.

5) - Conclusion sur l'utilisation des métriques intra-structure

Les métriques de contexte intra-structure sont particulièrement utiles pour choisir un seul schème parmi plusieurs schèmes émergents. Elles donnent au concepteur suffisamment d'indicateurs. Elles sont efficaces du moment que pour une même métrique, plusieurs valeurs (une par schème émergent) peuvent être comparées. Il n'est donc pas nécessaire de les utiliser lorsque le croisement fournit un seul schème émergent. Le concepteur peut toutefois y faire appel s'il veut avoir quelques indications sur ce schème. Il doit interpréter lui-même les valeurs obtenues, étant donné qu'il n'a pas de valeurs de comparaison.

Les métriques suivantes permettent d'avoir des indicateurs pour le schème émergent par rapport aux autres classes du modèle, qu'il soit unique ou sélectionné après avoir utilisé les métriques intra-structure.

5.3.2.3. Métriques de Contexte Inter-structures

1) - Définition

Les métriques de contexte inter-structures fournissent des indications sur le schème émergent par rapport à sa population. Les différentes métriques portent sur la mesure des écarts et des contradictions que peut avoir un schème émergent avec les autres classes de la population, ainsi que les classes potentiellement parentes (c'est-à-dire des super-classes potentielles) et les interactions avec les autres classes.

Le principal objectif de ces métriques est de donner des indicateurs pour positionner une nouvelle structure conceptuelle, véhiculée par le schème émergent, par rapport aux classes de la même population. Pour cela, les métriques de contexte inter-structures s'appliquent sur deux schèmes à la fois : le schème émergent et un schème permanent.

Nous définissons quatre métriques de contexte inter-structures :

2) - Métrique Distance Sémantique d.s

Cette métrique a été préalablement définie et illustrée dans le précédent chapitre (Chapitre 4 – Section 4.3.7). Elle est évaluée en l'appliquant à chacune des classes ayant participé au croisement par comparaison avec le schème émergent.

Interprétation : la plus forte d.s présuppose une classe parente. Une d.s trop faible présuppose l'élimination de la classe en tant que classe parente potentielle. Nous rappelons que nous entendons par classe parente pour un schème émergent, une classe qui peut devenir la super-classe de la structure conceptuelle véhiculée par ce même schème émergent.

3) - Métrique de Contradiction C_d

Cette métrique est définie pour détecter les contradictions qui peuvent exister entre un schème émergent et un schème permanent. Ces contradictions peuvent exister entre des attributs simples et des blocs d'attributs. Le principe de cette métrique est similaire à celle de la métrique de contradiction C_t de contexte intra-structure.

La métrique de contradiction C_d revient à dénombrer le nombre d'attributs contradictoires, le nombre de blocs contradictoires, ainsi que la vérification des invariants du modèle de classes exprimés sur la branche de la hiérarchie de la classe du schème permanent qui est en comparaison avec le schème émergent.

Elle est notée :

$$C_d(SE, SP) = \sum_{i=1}^{n} C_{d_i}$$

avec : n : la longueur des deux schèmes permanent et émergeant

C_{d_i} : la contradiction locale sur le gène en position *i* définie par :

$$C_{d_i} = \begin{cases} 1 & si \quad ((SP_i = 1) \wedge (SE_i = 0)) \\ & \quad \vee ((SP_i = 0) \wedge (SE_i = 1)) \\ 0 & \text{sinon} \end{cases}$$

avec : *SE* : Schème Émergeant

SP : Schème Permanent

SE_i : gène à la position *i* de *SE*

SP_i : gène à la position *i* de *SP*

Sachant qu'un gène en position *i* peut représenter un attribut simple ou un bloc d'attributs.

Interprétation : la plus faible C_d dénote le moins de contradictions entre le schème émergent et le schème permanent. Plus elle est faible, plus la classe associée au schème permanent peut être considérée comme éligible comme classe parente (donc comme super-classe) de la structure conceptuelle véhiculée par le schème émergent.

Les deux dernières métriques sont des métriques orientées objet standards.

4) - Métrique de Couplage C_p

Cette métrique est définie pour déterminer le couplage de la nouvelle structure conceptuelle avec toutes les autres classes du modèle (de la même population et éventuellement d'autres populations existant dans l'application). Cette métrique s'inspire de la métrique de couplage de contexte classe (section 5.2.3.2).

La métrique de couplage C_p s'applique au niveau de la conception et doit donc prendre en compte les différentes manières dont une classe peut être couplée. Dans [Briand 97], ils présentent trois modes possibles : *Classe-Attribut* (*CA*) où un attribut de la classe a un paramètre qui a pour type une autre classe, *Classe-Méthode* (*CM*) où une méthode de la classe a un paramètre qui a pour type une autre classe, et *Méthode-Méthode* (*MM*) où une méthode de la classe a un paramètre qui a pour type une méthode d'une autre classe.

Pour notre part, étant donné que nous travaillons essentiellement sur la structure, la métrique de couplage C_p s'intéresse aux couplages de type *CA* et également de type *CM* si les méthodes sont spécifiées, ne serait-ce que dans leur signature. Sinon, elle s'applique uniquement sur les couplages du type *CA*.

La métrique de couplage C_p s'applique sur un schème émergeant. Elle permet de déterminer à quel point ce schème émergeant sollicite et peut être sollicité dans le modèle. Elle est notée :

$$C_p(SE) = \sum CA + \left[\sum CM\right]$$

Interprétation : un trop fort couplage peut être pénalisant en termes de coûts de réorganisation de la hiérarchie de classes lors de l'insertion de la nouvelle structure conceptuelle.

5) - Métrique de Profondeur P_d

Cette métrique est définie pour situer à quel niveau dans la hiérarchie sera insérée la nouvelle structure conceptuelle.

La métrique P_d est notée :

$P_d(SE)$ = Niveau (Classe Parent) + 1

Interprétation : cette métrique permet d'évaluer les impacts plus ou moins importants de l'insertion de la structure conceptuelle émergente dans la hiérarchie de classes. Pour être facilement interprétée, elle doit être comparée à la profondeur de l'arbre d'héritage, généralement nommée *DIT* (*Depth of Inheritance Tree*) qui représente la longueur du chemin le plus long de la hiérarchie depuis la classe racine jusqu'à la plus lointaine classe feuille qui en hérite [Chidamber& 91].

6) - Conclusion sur l'utilisation des métriques inter-structures

Les quatre métriques inter-structures s'appliquent sur le schème émergent. Chacune permet de donner des indications sur chaque schème permanent potentiellement éligible par rapport au schème émergent. Elles sont nécessaires pour déterminer la classe qui pourra être la super-classe de la structure conceptuelle émergente.

L'objectif principal étant de dégager une classe potentiellement parente, les métriques de contexte inter-structures peuvent par contre connaître une dépendance entre elles quant à leur application. En effet, la première métrique inter-structures à appliquer est la d.s. Elle permet d'emblée d'éliminer les classes qui ont une très faible distance sémantique, c'est-à-dire celles qui présentent très peu de similitudes avec le schème émergent. Elle permet également de mettre en évidence celles qui présentent une forte d.s. Dans le cas où l'application de la d.s. serait suffisante pour déterminer une seule classe qui présente une forte d.s, il n'est pas nécessaire de poursuivre l'application des autres métriques, car la classe potentiellement parente est trouvée avec la d.s. La d.s est alors une métrique qui est à la fois *nécessaire* et *suffisante* pour trouver la classe potentiellement parente. Les autres métriques inter-structures pourront par contre être utilisées si le concepteur veut des indications supplémentaires sur la structure conceptuelle émergente et les conséquences de son insertion dans la hiérarchie de classes.

Dans certains cas, la d.s à elle seule ne suffit plus. Elle est toujours une métrique *nécessaire* mais n'est plus *suffisante* pour trouver la classe potentiellement parente. Les autres métriques sont alors utilisées.

La métrique de contradiction C_d s'appuie sur une analyse des blocs de gènes et des gènes indépendants. La première métrique, la d.s, comme nous l'avons vu dans le précédent chapitre, s'appuie sur la comparaison des gènes du schème émergent avec ceux d'un schème permanent. En combinant les valeurs obtenues pour la métrique de contradiction C_d avec celles obtenues pour la d.s, l'interprétation peut être plus riche. En effet, une classe qui présente une d.s intéressante pour un schème émergeant mais qui, en même temps présente trop de contradictions avec ce même schème peut aboutir à son élimination comme classe potentiellement parente.

Les deux dernières métriques, celle de Couplage C_p et celle de Profondeur P_d s'inspirent directement de métriques orientées-objets [Chidamber& 91], [Sahraoui 99]. La métrique de profondeur P_d permet de connaître l'endroit d'insertion de la nouvelle structure conceptuelle. Cette insertion est gérée dans le cadre d'un processus de développement (Chapitre 5 – Section 4.6 -).

Après avoir présenté les deux catégories de métriques et après avoir expliqué l'utilisation des métriques de chacune des deux catégories, nous abordons leur utilisation au vu des besoins pour mener à terme le processus d'émergence et également au vu des invariants du système.

5.3.3. L'application des métriques et les invariants

La principale question qui se pose est :

Comment et surtout quand et dans quelle séquence (ou en parallèle) appliquer les différentes métriques sur les schèmes émergents ?

Étant donné que l'objectif premier est la simulation de l'évolution d'un modèle de classes au vu de changements dynamiques dans la structure des instances, le concepteur décide à la fois des changements qu'il veut exprimer sur des instances et du moment où il veut déclencher le processus de simulation.

De la même manière, nous offrons au concepteur la possibilité d'exprimer des invariants à respecter par le processus d'évolution à différents niveaux :

5.3.3.1. Invariants sur les instances

Outre les invariants déjà présents dans les spécifications du modèle, le concepteur peut exprimer des invariants sur les instances en précisant les attributs qu'il veut avoir dans la structure conceptuelle finale, par 1 pour le gène correspondant dans le schème, et les attributs qu'il ne veut pas voir apparaître dans l'instance, par un 0 dans le schème temporaire associée à l'instance ayant évoluée. Les attributs qu'il considère comme indifférents ou sur lesquels il a un doute, seront à #.

5.3.3.2. Invariants sur les résultats de l'émergence

Outre ce premier type d'invariants, le concepteur peut également préciser des invariants sur l'évolution. Cela revient à exprimer des invariants sur les résultats de l'émergence. Ces invariants mettent en jeu les métriques précédemment définies. Ils déterminent également la manière dont elles seront utilisées et si leur application est totale ou sélective.

Le concepteur peut par exemple préciser qu'il préfère une structure conceptuelle émergente qui provoquerait le moins possible de réorganisation. La simulation d'évolution devra prendre en compte cet invariant. L'invariant revient à la « *stabilité du modèle de classes* ». Parmi les métriques proposées, celle de couplage C_p est plus adaptée que les autres métriques. Elle est donc privilégiée. Le concepteur peut également préférer une structure conceptuelle émergente de plus haut niveau possible dans la hiérarchie.

Le tableau suivant résume selon l'invariant imposé par le concepteur, la métrique inter-structures qui lui est associée et qui est prépondérante dans le choix de la classe parente pour la nouvelle structure conceptuelle :

Métrique Inter-structures	Invariants associés	Conséquences
Métrique Distance Sémantique d.s	Elle est obligatoire. Il n'y a donc pas d'invariant associé.	-
Métrique de Contradiction C_d	Les contradictions ne sont pas prépondérantes	La classe sélectionnée peut présenter des contradictions. Son insertion nécessite alors un effort de révision dans ses spécifications. Cette métrique sera uniquement indicative.
	Une structure conceptuelle avec le minimum de contradictions	La classe sélectionnée sera la plus cohérente possible. Son insertion dans la hiérarchie peut se faire sans modifications majeures. Cette métrique sera prépondérante.
Métrique de Couplage C_p	Le degré de couplage n'est pas prépondérant	La structure conceptuelle émergente peut être fortement couplée. Cette métrique sera uniquement indicative.
	Le degré de couplage doit être minimal	La structure conceptuelle émergente doit être faiblement couplée. Cette métrique sera prépondérante.
Métrique de Profondeur P_d	Les efforts de réorganisation ne sont pas prépondérants	La structure conceptuelle émergente peut engendrer des coûts et des efforts pour gérer les impacts de son insertion. Cette métrique sera uniquement indicative.
	La classe doit engendrer un minimum de réorganisation	La structure conceptuelle émergente devra être sélectionnée de manière à éviter un minimum de réorganisation suite à son insertion. Cette métrique sera prépondérante.

Tableau 5-6: métriques inter-structures et invariants associés.

Ces invariants sont prépondérants dans le choix des métriques à appliquer sur les schèmes émergents. Ils peuvent par conséquent imposer l'ordre de leur application, voire même réduire les métriques à appliquer. Dans le cas où une classe répondrait à l'invariant imposé par le concepteur et que par ailleurs, elle présente des valeurs médiocres pour les autres métriques, elle n'est pas rejetée. Cette classe ainsi que les différentes valeurs qu'elle a obtenues sont présentées au concepteur qui a la charge d'accepter et de prendre en compte les différents résultats, ou de rejeter ses choix et de recommencer l'application des métriques, voire de la simulation de l'évolution.

Dans le cas où le concepteur ne préciserait pas d'invariants, seules les métriques nécessaires pour choisir un schème émergent sont appliquées. D'autres métriques

peuvent être utilisées dans le cas ou celles qui ont été appliquées ne suffisent pas à démarquer les schèmes émergents entre eux.

En effet, les métriques de contexte intra-structure se complètent avec les métriques inter-structures pour enrichir le processus d'émergence globale par croisement. Les premières sont utiles pour le choix d'un schème émergent, lorsqu'il y a un choix à faire entre plusieurs schèmes émergents. La seconde catégorie de métriques s'applique de manière indépendante de la première. Parmi les métriques inter-structures que nous appliquons par défaut, ce sont les deux premières métriques pour :

- Déterminer la ou les super-classes grâce à la distance sémantique d.s.
- Détecter d'éventuelles contradictions entre les schèmes émergents et chaque classe potentiellement parente. Cette métrique peut influencer le choix définitif de schèmes émergents et de super-classe(s).

Les deux autres métriques peuvent influencer le choix si le concepteur les activent comme étant importantes. De la même manière, elles peuvent influencer les choix finaux.

La section suivante permet d'illustrer l'utilisation des métriques. Nous revenons sur les deux exemples de croisement qui n'ont pas pu être menés à terme en utilisant uniquement la d.s.

5.4 - Exemples

Nous partons des résultats de l'émergence globale par croisement de la fin du chapitre 5 – Section 4.5.2.2, qui relèvent les manques de l'émergence et des exemples qui n'ont pas pu être traités complètement, notamment les objets O_1 et O_5. Le premier a eu deux schèmes émergents. Le second a eu un seul schème émergent, mais deux parents du croisement qui ont la même distance sémantique.

Nous appliquons les différentes métriques pour chacun des cas :

5.4.1. Exemple de l'objet O_1

Reprenons l'objet O_1 avec les deux schèmes émergents :

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	
O_1	1	1	#	#	1	0	1	0	0	0	1	#	Va
E4	1	1	1	0	1	0	1	0			1	#	5/5=1
E5	1	1	#	1	1	0	1	0			1	0	5/5=1

Tableau 5-7: les deux schèmes émergents E4 et E5 pour O_1.

Les gènes sont numérotés de 1 à 12, avec les gènes en positions 9 et 10 qui ne sont pas considérés étant donné qu'aucun des schèmes considérés ne les spécifient.

Ayant deux schèmes émergents, il est nécessaire d'en choisir un, avant de rechercher à quelle classe de la population rattacher la structure conceptuelle qu'il sous-tend. Les métriques de Contexte intra-structure sont nécessaires pour élire un schème émergent :

5.4.1.1. Métriques Contexte intra-structure

Les métriques intra-structure sont appliquées pour chaque schème émergent :

Métrique	**Valeur**		**Conclusions**
	Schème émergent E_4	**Schème émergent E_5**	
Signification S	$S(E_4) = 2$ les gènes 3 et 12	$S(E_5) = 2$ les gènes 3 et 4	*$S(E_4)$= $S(E_5)$=2 valeurs non significatives*
Contradiction C_t	$C_t(E_4) = 0$	$C_t(E_5) = 0$	*Aucune contradiction*
Cohérence C_h	$C_h(E_4)$ = nbr d'attributs simples(E_4) + nbr de blocs(E_4) + nbr de blocs indépendants(E_4) = 1 + 2 + 0 = 3	$C_h(E_5)$ = nbr d'attributs simples(E_5) + nbr contradiction entre attributs simples(E_5) + nbr de blocs(E_5) + nbr de blocs indépendants(E_5) = 5 + 0 + 1 + 0 = 6	*$C_h(E_4)$=3 ; $C_h(E_5)$=6 Le schème E_4 est plus cohérent que le schème E_5*

Tableau 5-8: applications des métriques intra-structure pour les schèmes émergents de l'objet O_1.

Conclusion :

- les deux schèmes émergents présentent la même signification S=2 ;
- les deux schèmes émergeants ne présentent aucune contradiction C_t=0 ;

- le schème E_4 présente une plus grande cohérence que le schème E_5, C_h est respectivement à 3 et à 6 :

Le schème émergent E_4 est élu comme étant le schème émergent.

5.4.1.2. Métriques Contexte inter-structures

Une fois le schème émergent élu, nous devons rechercher à quel endroit de la hiérarchie l'insérer. Nous devons appliquer les métriques inter-structures. La première métrique à appliquer est la d.s. Si elle n'offre pas de résultats permettant de choisir, les autres métriques doivent être appliquées pour pouvoir choisir une classe parente.

Pour calculer la d.s, il est nécessaire de comparer les schèmes permanents ayant participé au croisement avec le schème émergent :

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	
O_1	1	1	#	#	1	0	1	0	0	0	1	#	
Schème Émergeant	1	1	1	0	1	0	1	0	0	0	1	#	d.s
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	1/7
Ater	1	1	#	0	1	1	0	1	0	0	0	0	4/7
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	6/7

Tableau 5-9: application de la d.s sur le schème émergent de O_1.

Senior-Chercheur est la classe qui présente la plus grande distance sémantique : 6/7. Cette valeur est forte. La classe Senior-Chercheur est donc éligible comme classe parente de la structure conceptuelle émergente.

Conclusion : il est proposé au concepteur de créer la structure conceptuelle émergente véhiculée par le schème émergent comme sous-classe de Senior-Chercheur. Ce choix est d'autant plus cohérent que le schème émergent et Senior-Chercheur présentent le même bloc GH. Tout porte donc à considérer la structure conceptuelle émergente comme étant effectivement une spécialisation de Senior-Chercheur.

Étant donné que la d.s est suffisante pour cet exemple et étant donné qu'il n'y a pas d'invariants particuliers, les métriques de *couplage* et de *profondeur* ne sont pas utiles pour le choix final. Elles sont uniquement instructives. A titre d'exemple, en notant `ref` toutes les références du schème et `est-ref` toutes les fois où le schème est référencé, les valeurs de couplage pour le schème sont :

$$C_p(E_4) = \Sigma_{\texttt{ref}} + \Sigma_{\texttt{est-ref}} = 5 + 0 = 5$$

Le choix précédent peut être conforté par les valeurs des autres métriques inter-structures. Nous rappelons la d.s et nous appliquons les trois autres métriques :

Métrique	Valeur	Conclusions
Distance sémantique d.s	d.s(Enseignant) = 1/7	*Éliminée*
	d.s(Ater) = 4/7	*Moyennement éligible*
	d.s(Senior-Chercheur) = 6/7	*Fortement éligible*
Contradiction C_d	$C_d(E_4, \text{Ater}) = 1$	*Pénalisant*
	C_d (E_4, Senior-Chercheur) = 0	*Favorable*
Couplage C_p	$C_p(E_4) = 5$	-
Profondeur P_d	$P_d(E_4) = 5$	-

Tableau 5-10: Métriques inter-structures sur le schème émergent de O_1.

Conclusion : en appliquant la métrique de contradiction sur le schème émergent et chacune des deux classes qui présentent une d.s assez bonne, à savoir Ater et Senior-Chercheur, il apparaît que E_4 présente une contradiction avec Ater (E_4 spécifie l'attribut *Date-Début* alors que Ater spécifie l'attribut *Durée* ; ces deux attributs sont exclusifs) alors qu'il n'en présente aucune avec Senior-Chercheur. Ce résultat conforte le premier choix fait avec la d.s.

Les deux dernières métriques n'apportent aucune information qui puisse influencer d'une manière ou d'une autre le choix de la classe parente. Elles sont purement informatives.

Appliquons les métriques sur le schème émergent de l'objet O_5.

5.4.2. Exemple de l'objet O_5

Dans le cas de l'objet O_5, il y a un seul schème émergent. Il reste à déterminer ses parents. Les métriques de contexte intra-structure ne sont pas utiles. Elles peuvent par contre donner un peu plus d'indications au concepteur sur la constitution du schème :

5.4.2.1. Métriques Contexte intra-structure

L'application des métriques intra-structure pour le schème émergent E_5 donne :

Métrique	Valeur	Conclusions
Signification S	$S(E_5) = 2$	*valeur non significative*
Contradiction C_t	$C_t\ (E_5) = 0$	*aucune contradiction avec l'objet O_5*
Cohérence C_h	$C_h\ (E_5)$ = nbr d'attributs simples(E_5) + nbr de blocs(E_5) + nbr de blocs indépendants(E_5) = 3 + 0 + 2 + 0 = 5	*non significative toute seule, sauf qu'elle révèle l'absence de contradictions*

Tableau 5-11: Métriques intra-structures sur le schème émergent de O_5.

Conclusion: les métriques de contexte intra-structure n'apportent aucune information dans ce cas, sauf qu'il n'y a pas de contradiction au sein du schème émergent E_5.

L'application des métriques inter-structures est plus utile :

5.4.2.2. Métriques Contexte inter-structures

Nous devons rechercher à quel endroit de la hiérarchie insérer la structure conceptuelle émergente véhiculée par le schème émergent E_5. Pour calculer la d.s, il est nécessaire de comparer les schèmes permanents ayant participé au croisement avec les schème émergent. Le calcul de la d.s pour ce schème émergent a déjà été effectué (Chapitre 5 – Section 4.5.3.2). Nous le rappelons à ce niveau :

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	
O_5	1	1	#	#	1	0	1	0	0	0	0	#	
Schème Émergeant	1	1	#	1	1	0	1	1	0	0	1	0	d.s
Enseignant	0	0	0	1	1	0	0	0	0	0	0	0	2/7
Ater	1	1	#	0	1	1	0	1	0	0	0	0	5/7
Senior-Chercheur	1	1	1	0	0	0	1	0	0	0	1	#	5/7

Tableau 5-12: application de la d.s sur le schème émergent de O_5.

Conclusion : la classe Enseignant est éliminée car elle présente une faible distance sémantique. Par contre, les deux autres classes présentent chacune une forte d.s. Par contre, elles sont identiques : 5/7.

Nous devons appliquer les trois autres métriques, même si seules les deux premières métriques sont nécessaires pour établir un choix entre les deux classes Ater et Senior-Chercheur :

Métrique	Valeur	Conclusions
Distance sémantique d.s	d.s(Enseignant) = 2/7	*Éliminée*
	d.s(Ater) = 5/7	*Fortement éligible*
	d.s(Senior-Chercheur) = 5/7	*Fortement éligible*
Contradiction C_d	Cd(E_5, Ater) = 3	*Pénalisant*
	Cd (E_5, Senior-Chercheur) = 0	*Favorable*
Couplage C_p	Cp(E_5) = 5	-
Profondeur P_d	Pd(E_5) = 5	-

Tableau 5-13: Métriques inter-structures sur le schème émergent de O_5.

Conclusion: seules les deux premières métriques sont utiles dans ce cas. Le schème émergent E_5 donne lieu à une structure conceptuelle émergente qui sera potentiellement une sous-classe de Senior-Chercheur.

Une fois les schèmes émergeants choisis et les classes parentes trouvées, l'émergence globale peut être achevée. Nous l'illustrons sur l'exemple en présentant l'état final du modèle de classes dans la section suivante.

5.5 - l'exemple final

Dans le cas de notre exemple, les processus d'évolution ont dégagé deux structures conceptuelles émergentes. Ces structures conceptuelles seront créées toutes les deux en tant que sous-classes de Senior-Chercheur. Par conséquent, elles ont en commun le même bloc GH, hormis les changements explicites. Les schèmes émergents deviennent des structures conceptuelles émergentes qui prennent en compte l'analyse des métriques appliquées :

	Attributs existants												
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	
Schème Émergent(O1)	1	1	1	0	1	0	1	0	0	0	1	#	Nouveaux attributs
Schème Émergent(O2)	1	1	#	1	1	0	1	0	0	0	1	0	
Structure émergente(O1)	1	1	1	0	1	0	1	0	0	0	1	#	1 - -
Structure émergente(O2)	1	1	#	1	1	0	1	0	0	0	1	0	- - -

Tableau 5-14: les structures conceptuelles émergentes.

Une analyse des deux schèmes émergents met en évidence une similitude dans leur constitution. Seuls diffèrent le quatrième attribut, à savoir *Spécialité*, et le nouvel attribut introduit par O_1, à savoir *Grade* qu'il value avec '*Professeur*'. Cette constatation est soumise au concepteur. Il réalise en effet que les deux structures conceptuelles sont identiques, mis à part ces deux attributs.

Le concepteur ajuste la structure finale pour ne garder qu'une seule structure conceptuelle. Il peut choisir de garder le quatrième attribut (donc mettre le gène correspondant à l'attribut *Spécialité* à 1) et d'accepter le nouvel attribut *Grade*. Ce dernier est alors intégré dans le modèle et perd son statut transitoire.

Les deux structures conceptuelles mettent en évidence des spécificités d'un Senior-Chercheur, présentent des caractéristiques d'un Enseignant (principalement l'attribut *Module* présents dans les deux structures conceptuelles) et introduisent de nouvelles informations. Avec les choix faits, elles se sont donc complétées mutuellement. Elles sont donc représentées par une seule et même structure conceptuelle : celle d'*Enseignant-Chercheur*.

La classe est insérée dans la population Membre-Université comme sous-classe de Senior-Chercheur. La figure suivante met en évidence cette insertion :

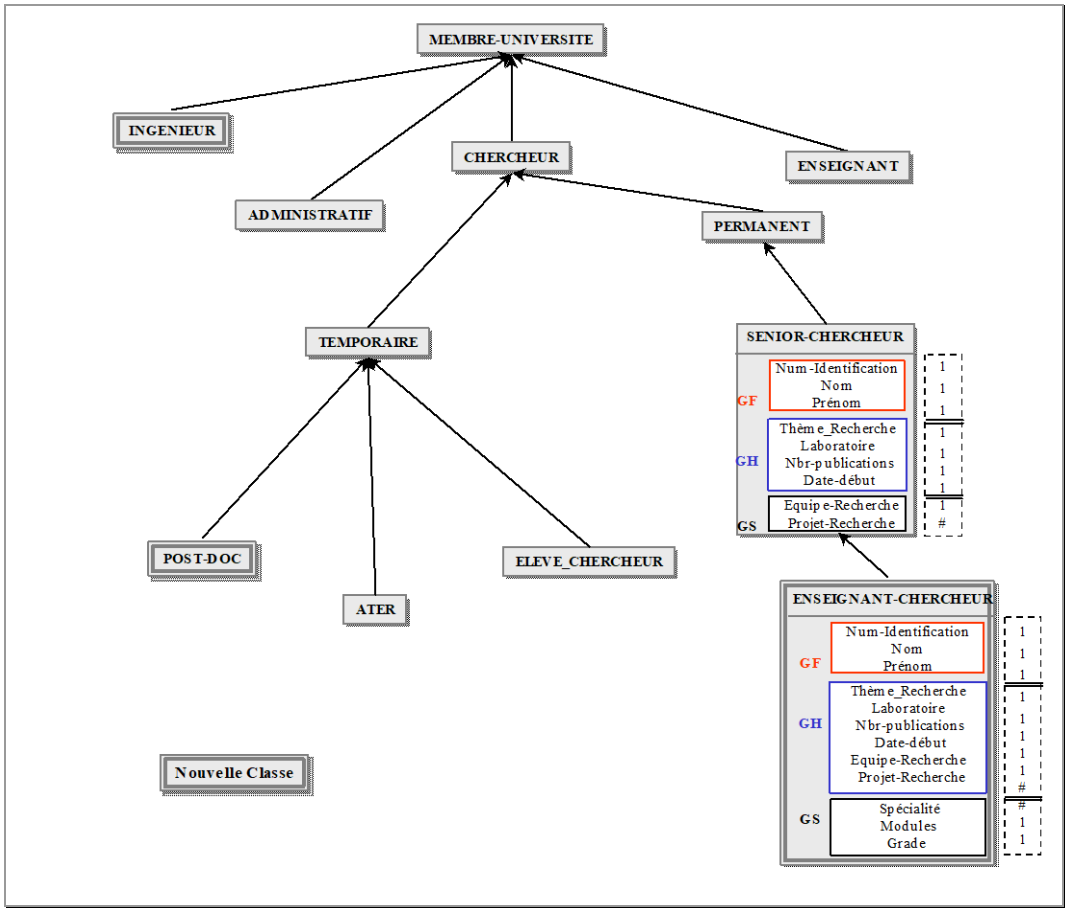


Figure 5-2: Insertion d'Enseignant-Chercheur dans la population de Membre-Université.

Les objets O_1 et O_5 sont ajustés à la nouvelle structure conceptuelle avant d'être rattachés à leur nouvelle classe. L'objet O_1, en plus de son évolution initiale, peut valuer (gène à # dans le schème permanent) en plus les attributs acquis *Spécialité* et *Projet-Recherche*. L'objet O_5, en plus de son évolution initiale, value obligatoirement (gène à 1 dans le schème permanent) les attributs *Projet-Recherche* et *Grade*.

Le lien d'évolution doit également être spécifié. Il existe en fait deux liens d'évolutions :

– un lien d'évolution ayant pour source Ater et pour cible Enseignant-Chercheur. Nous le notons L_4 ;

– un lien d'évolution ayant pour source Elève-Chercheur et pour cible Enseignant-Chercheur. Nous le notons L_5.

Le graphe d'évolution est complété. La figure suivante complète la figure 16 du chapitre 4 :

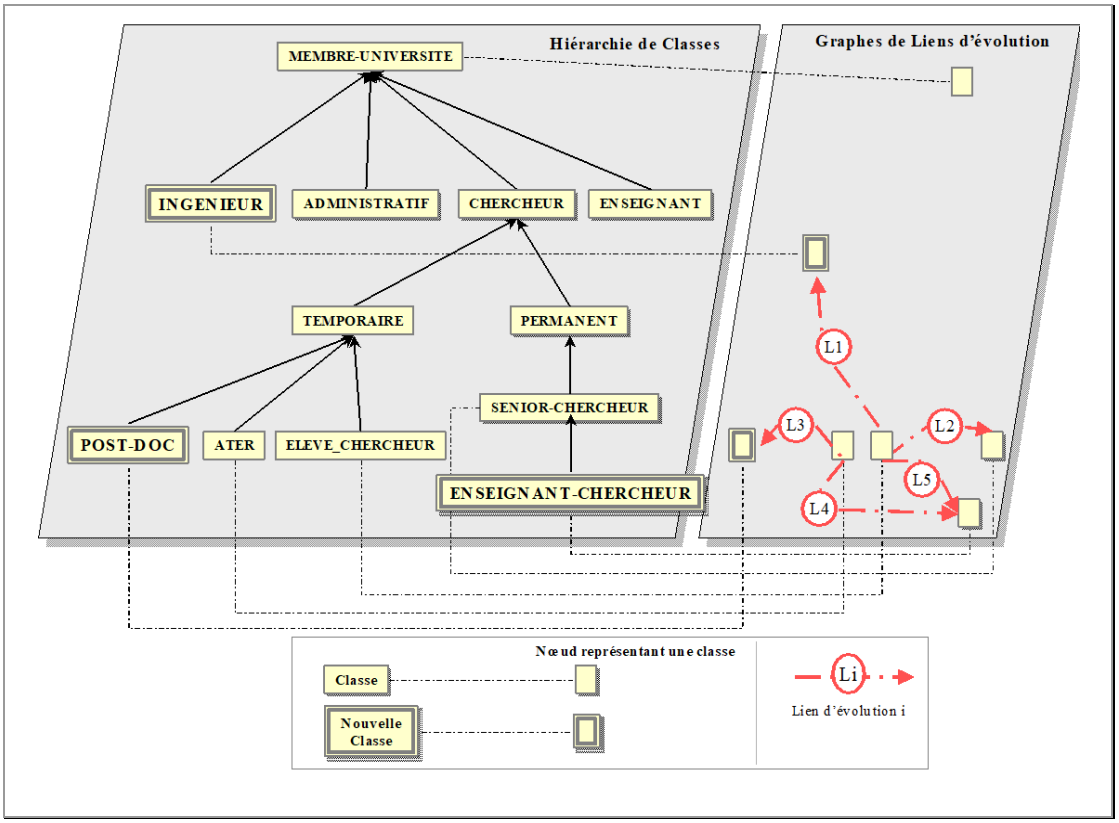


Figure 5-3: Hiérarchie de classes et Graphes de Liens d'évolution.

En plus des liens d'évolution L_1, L_2 et L_3 créés dans le cadre des processus d'évolution des objets O_2, O_3 et O_4 (Chapitre 4), les liens L_4 et L_5 sont les deux liens d'évolution qui sont créés après avoir mené à bien les processus d'évolution des objets O_1 et O_5.

L_4 est constitué des attributs suivants :

-durée

-nbr-heures

- année

-directeurthèse

+date-début

+Equipe-Recherche

+Projet-Recherche

+spécialité

+grade

L_5 est constitué des attributs suivants :

-durée

- nombre-heures

+date-début

+Equipe-Recherche

+Projet-Recherche

+spécialité

+grade

Chacun des liens d'évolution est orienté de la source vers sa cible. Aussi, les attributs précédés d'un signe - sont les attributs de la source qui sont perdus et les attributs précédés du signe + sont les attributs acquis de la cible. C'est la structure que respectera toute instance de la classe source qui évoluera vers la classe cible.

– Enseignant-Chercheur est une nouvelle structure conceptuelle qui a été créée par croisement. Elle est une sous-classe de Senior-Chercheur. Une instance de lien d'évolution a été créée entre chacune des classes dont une instance a évoluée vers cette nouvelle structure conceptuelle. Chaque lien détient comme propriétés les attributs perdus et les attributs acquis.

– Deux nouvelles classes ont été créées par émergence directe : la classe Ingénieur comme une sous-classe de Membre-Université et Post-doc comme une sous-classe de Temporaire.

5.6 - Conclusion

L'émergence, et plus particulièrement l'émergence globale par croisement, est le processus d'évolution qui constitue le noyau de notre modèle. Il est primordial d'assurer la vérification et l'analyse des résultats de l'émergence dans le but de mieux les contrôler, et par conséquent de mieux décider.

Nous considérons que l'émergence peut être mesurée et contrôlée. Les résultats obtenus nécessitent de prendre en compte un critère important : la cohésion de la structure conceptuelle émergeante vis-à-vis d'elle-même et du reste du modèle.

Aussi, nous proposons l'utilisation de métriques pour mieux comprendre et analyser la constitution d'une structure conceptuelle émergente. Ces métriques sont dites *Métriques de Contexte intra-structure*. Elles permettent de déceler d'éventuelles contradictions et incohérences internes.

D'autres métriques sont utilisées pour mieux comprendre et analyser l'impact de l'insertion d'une structure conceptuelle émergente dans une hiérarchie de classes. Ces métriques sont dites *Métriques de Contexte inter-structures*. Elles permettent de trouver les super-classes potentielles de la nouvelle structure conceptuelle, de détecter d'éventuelles contradictions avec elles, et de finalement choisir une seule classe parente. Elles permettent également de connaître la position de la structure

conceptuelle émergente dans la hiérarchie ainsi que le degré de couplage de cette structure conceptuelle avec les autres classes.

Les métriques ne sont pas toutes appliquées de manière systématique, principalement les secondes. Les premières sont utiles pour départager entre plusieurs schèmes émergents lorsque la situation se présente : il faut prendre le schème émergent le plus cohérent. Les secondes sont utiles pour attirer l'attention sur les impacts d'insertion (si les coûts ne représentent aucun souci, elles ne sont pas utiles). Aussi, étant dans le cadre d'un outil de simulation au service du concepteur pour l'évolutivité et la maintenabilité de systèmes dont il a la charge, le concepteur a la possibilité d'exprimer des *invariants* soit sur des instances, soit sur les schèmes émergents. Ces invariants devant être respectés, les métriques associées ont alors toute leur utilité.

Il est essentiel de préciser que le modèle que nous proposons ne répond pas de manière absolue et sans appel à la prise en compte de tous les nouveaux besoins. Il reste un outil de simulation pour l'aide à l'évolution et à la maintenance. La manière dont les besoins et les invariants sont exprimés influencent grandement les résultats. De la même manière, si des erreurs de conception existent, l'objectif n'est pas un objectif de correction de conception mais si les résultats apparaissent trop incohérents, cela peut dénoter une mauvaise conception ou une mauvaise spécification initiales des besoins, et permet aussi d'attirer l'attention du concepteur.

Chapitre 6 : IMPLEMENTATION

6.1 - Introduction

Nous abordons dans le présent chapitre une vue générale de la réalisation et de l'implémentation du modèle GENOME. Ce chapitre aborde en premier lieu le contexte et l'environnement déjà existants et par rapport auxquels GENOME doit s'intégrer. Il s'agit de l'environnement de Présage. Nous décrivons pour cela l'infrastructure de Présage. Nous présentons ensuite les choix d'implémentation de GENOME et son positionnement par rapport à l'infrastructure de Présage.

La seconde partie de ce chapitre est consacré au modèle de représentation de GENOME. L'architecture du modèle dont une première ébauche a été présentée dans le chapitre 3, est complétée en précisant les méthodes de chaque classe ainsi qu'une brève description. Nous avons décrit les processus d'évolution par des automates dans le chapitre 4. Nous présentons plus de détails dans ce chapitre, principalement les classes (via leurs méthodes) mises en jeu pour mener ces processus d'évolution, principalement le croisement. Les présentations sont faites sur la base de la notation d'UML [Muller 97].

Nous terminons enfin par un exemple de Présage auquel peut répondre GENOME, ainsi qu'un bref descriptif des parties implémentées et des résultats.

6.2 - L'environnement Présage et GENOME

6.2.1. L'infrastructure de Présage

Présage est un logiciel de modélisation et de planification de réseaux de télécommunications, qui consiste à étudier l'évolution de ces réseaux à court, moyen et long termes, à savoir dans un délai pouvant aller de 1 an à 10 ans. Un environnement orienté objet à été développé pour l'implantation de l'infrastructure de Présage [Caminada 92]. Le modèle est basé sur une modélisation hiérarchisée/multi-vues [Oussalah 88] : un réseau peut être considéré suivant ses différentes vues. Par exemple, une vue peut correspondre à une demande en transmission, ou à son implantation physique (en anneaux, en étoile, point-à-point…). Une vue peut être associée à différents niveaux d'abstractions.

Présage s'adresse à deux types d'utilisateurs : le *Constructeur d'Applications* qui modélise un réseau, et l'*Utilisateur Final* qui manipule une instance de ce réseau. Cet environnement a été implémenté dans le langage C++. Une de ses principales caractéristiques est la dynamicité des classes. En effet, la dynamicité des classes au niveau du Constructeur d'Applications CA est simulée grâce à l'implantation d'une sur-couche du langage C++ [Puig 97].

Toute classe créée dynamiquement par le CA est représentée par une instance C++, qui lors du passage en mode Utilisateur Final UF, sera transformée en classe C++, grâce à une étape de génération automatique de code C++ :

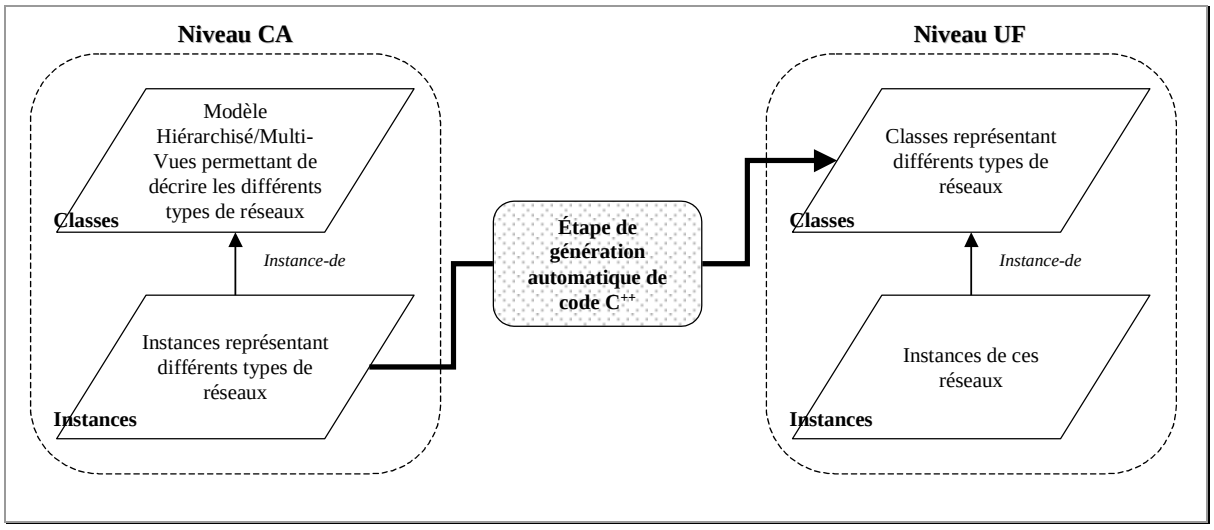


Figure 6-1: Présage: prise en charge de la dynamicité des classes dans C++.

Ainsi, les classes niveau CA représentent le modèle hiérarchisé/multi-vues de planification de réseaux de télécommunications. Par exemple, la planification de réseaux urbains selon différentes vues (zone, commutations, dimension du réseau, transmission), et pour chaque vue selon plusieurs abstractions (niveaux système, demande de transmission, support physique …). Ce modèle est instancié pour définir différents types de réseaux urbains.

Ces instances, grâce à la **génération automatique de code C++**, sont représentées par des classes ainsi générées, au niveau UF. Ces classes sont ensuite instanciées pour définir des réseaux urbains. C'est ainsi que les attributs sont réifiés en mode CA et deviennent de vrais attributs contenus dans les classes en mode UF. L'ajout et la modification d'attributs sont simulés. C'est également ainsi que la relation d'héritage entre ces classes dynamiques a été ré-implémentée grâce aux attributs *isa* qui référence la super-classe, et *sub* qui référence les sous-classes.

6.2.2. GENOME et les choix d'implémentation

6.2.2.1. Positionnement de GENOME par rapport à Présage

Nous considérons GENOME comme étant indépendant. Il peut en effet parfaitement être considéré comme un module indépendant, voire une application indépendante.

Dans le cadre de son intégration dans Présage, nous le considérons comme un module à part, avec des points d'entrée et des points de sortie. Étant un outil de simulation d'évolution, le module de GENOME travaille à part de l'application Présage qui s'exécute dans le cadre de ses activités de planification.

Lorsque le concepteur nécessite de simuler l'évolution au vu de certaines données, il transmet à GENOME comme entrée la base de données ou la partie de la base de données concernée par l'évolution. GENOME donne en retour comme sortie les résultats de la simulation de l'évolution, et donc de changements proposés en termes d'émergence de nouvelles structures conceptuelles. Ces changements sont proposés au concepteur. Lui seul a le pouvoir de décider d'accepter les changements, voire de les ajuster avant de les intégrer définitivement dans la base de Présage :

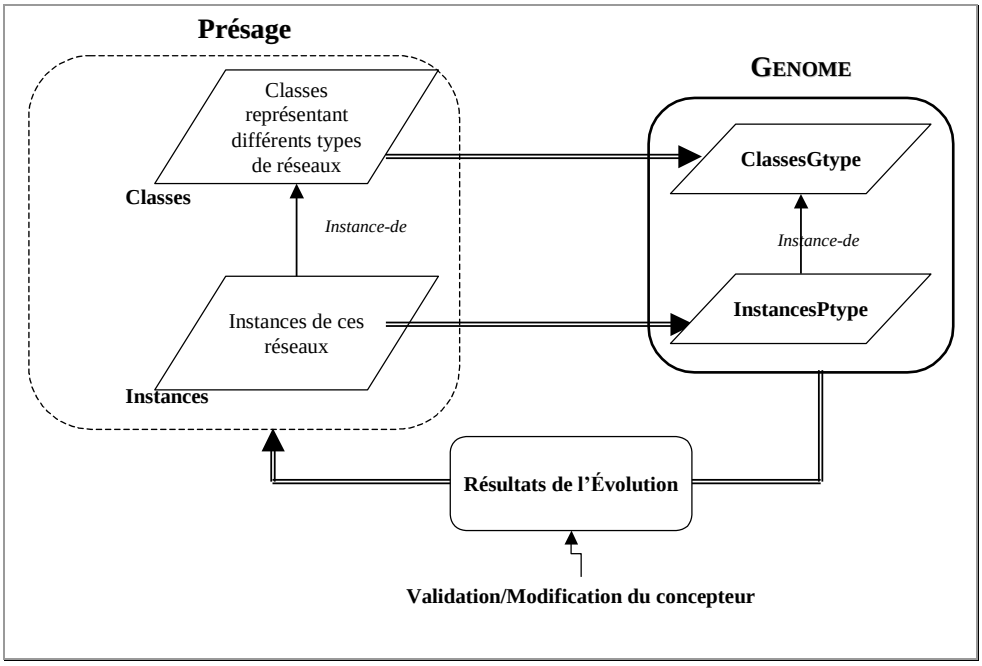


Figure 6-2: Interactions Présage/Genome.

Ce passage entre les deux modèles implique le passage de classes, liens et instances Présage et qui sont nécessaire à GENOME pour mener l'évolution. Ces classes, liens et instances sont donc momentanément décrits en GENOME, le temps de la simulation de l'évolution. La principale information qui doit être dégagée par le concepteur pour transcrire la (portion de) base de données Présage en GENOME est de distinguer la population, donc de cerner le Génotype Fondamental, ainsi que la valuation des schèmes permanents. Les GH et GS se déduisent sans encombre grâce au lien d'héritage.

Une fois les résultats de l'évolution obtenus, validés et/ou corrigés par le concepteur, les données sont à nouveau transmises et sont intégrées à la base Présage dans leur nouvel état.

6.2.2.2. Choix d'implémentation de GENOME

La nouvelle politique adoptée par le CNET dans le cadre du développement de nouvelles applications et de l'extension d'applications existantes, est de favoriser le développement en Java. Pour cette raison, GENOME ainsi que les parties développées du prototype sont spécifiés en Java.

Les similitudes de GENOME avec Présage est la réification des différents concepts. Cependant, la différence des langages d'implémentation nécessite de développer une passerelle entre les deux applications pour prendre en compte le passage C++/Java des informations transmises entre Présage et GENOME. Cette partie a fait l'objet de propositions en cours d'études et de validation. Cette étape nécessite le développement d'un noyau permettant un interfaçage automatique entre Présage et GENOME.

Nous présentons dans la section suivante le modèle d'implémentation de GENOME :

6.3 - Le modèle d'implémentation de GENOME

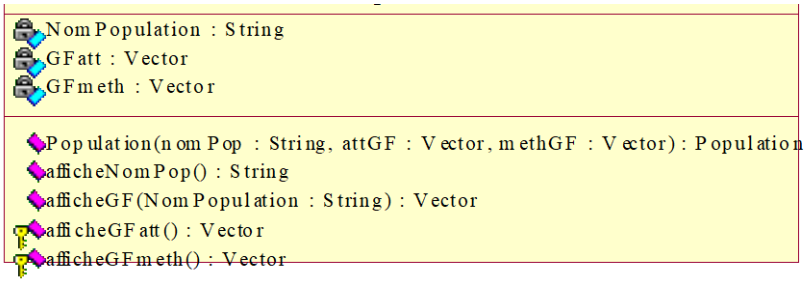
6.3.1. Le modèle GENOME

Nous reprenons les classes du modèle de GENOME présenté au chapitre 3, et nous détaillons chacune d'entre elles, principalement en termes de méthodes car les principaux attributs ont déjà été présentés, ainsi qu'un bref descriptif.

Certains attributs diffèrent de ceux présentés dans la synoptique du modèle. Il s'agit principalement des trois groupes de gènes. En effet, dans la partie implémentation, nous distinguons les attributs et les méthodes constituants chaque groupe de gènes : GF, GH et GS.

– **Classe Population :**

public class Population extends Object : cette classe représente les caractéristiques d'une population.



Attributs

NomPopulation : est de type chaîne de caractères.

GFatt : est l'ensemble des références des attributs (instances de la classe **Attributs**) qui constituent les attributs du GF.

GFmeth : est l'ensemble des références des méthodes (instances de la classe **Methode**) qui constituent les méthodes du GF.

Remarque : un GF d'une population est réellement constitué de *GFatt* et de *GFmeth*.

Méthodes

public Population(String nomPop, Vector attGF, Vector methGF) : est le constructeur.

public String afficheNom() : méthode qui affiche le nom de la population courante.

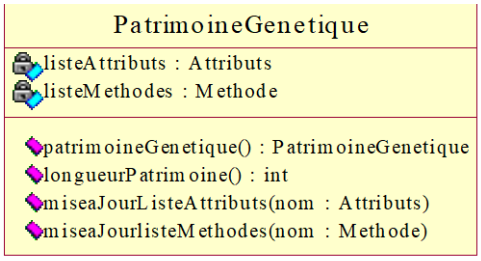
protected Vector afficheGF(String nomPopulation) : méthode qui affiche le GF de la population courante. Elle fait appel dans son corps aux méthodes protected *afficheGFatt* et *afficheGFmeth*.

protected Vector afficheGFatt(String nomPopulation) : méthode qui affiche les attributs du GF de la population courante.

public Vector afficheGFmeth(String nomPopulation) : méthode qui affiche les méthode du GF de la population courante.

– **Classe PatrimoineGenetique :**

public class PatrimoineGenetique extends Object : cette classe stocke l'ensemble des références de gènes (attributs et méthodes) constituant le patrimoine génétique d'une population.



Attributs :

listeAttributs : c'est l'ensemble des références d'attributs (classe **Attributs**) constituant le patrimoine génétique de la population associée.

listeMethodes : c'est l'ensemble des références de méthodes (classe **Methode**) constituant le patrimoine génétique de la population associée.

Méthodes :

public patrimoineGenetique() : constructeur.

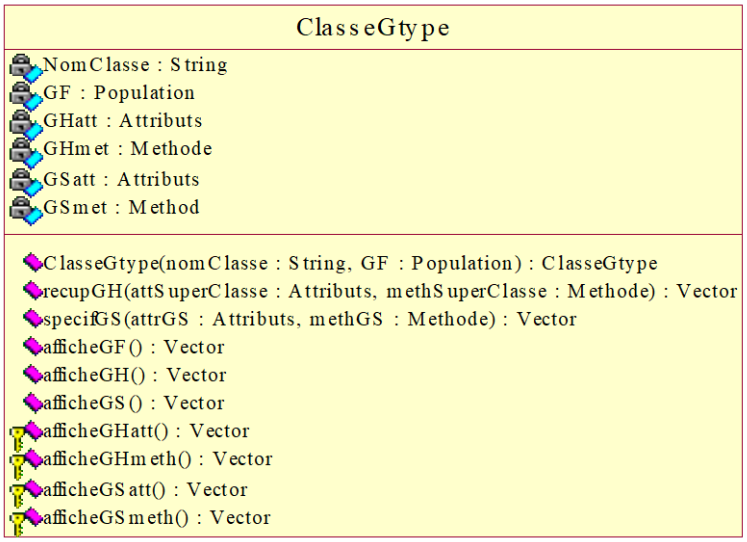
public int longueurPatrimoine() : méthode qui renvoie la taille du patrimoine génétique.

public void miseaJourlisteAttributs(Attributs nom) : méthode qui met à jour le patrimoine génétique avec les nouveaux attributs/méthodes de la classe courante de la population.

public void miseaJourlisteMethode(Methode nom) : méthode qui met à jour le patrimoine génétique avec les nouveaux attributs/méthodes de la classe courante de la population.

– **Classe ClasseGtype :**

public class ClasseGtype extends Object : c'est la classe qui permet de représenter toute classe sous GENOME. La principale distinction est les trois groupes de gènes.



Attributs :

NomClasse : est de type chaîne de caractères et permet de nommer une classe.

GF : est une référence vers une instance de la classe **Population**. De cette sorte, une classe est toujours liée à une population et respecte par conséquent le GF. Cet attribut est transmis à toutes ses sous-classes.

GHatt : représente l'ensemble des attributs constituant le GH de la classe courante.

GHmet : représente l'ensemble des méthodes constituant le GH de la classe courante.

GSatt : représente l'ensemble des attributs constituant le GS de la classe courante.

GHmet : représente l'ensemble des méthodes constituant le GS de la classe courante.

Méthodes :

public ClasseGtype(String NomClasse, Population : GF) : constructeur.

public Vector recupGH(Attributs : attSuperClasse, Methode : methSuperClasse) : constitue le GH de la classe courante en récupérant le GH et le GS de la super-classe.

public Vector specifGS(Attributs : attSuperClasse, Methode : methSuperClasse) : permet de spécifier le GS de la classe courante.

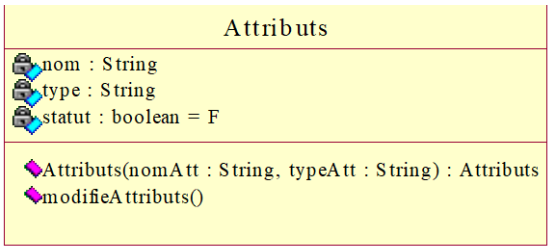
public afficheGF() : fait appel à la méthode afficheGF de la classe **Population**.

public afficheGH() : affiche le GH de la classe en cours en faisant appel aux méthodes afficheGHatt et afficheGHmeth.

Public afficheGS() : affiche le GS de la classe en cours en faisant appel aux méthodes afficheGSatt et afficheGSmeth.

– **Classe Attributs :**

public class Attributs extends Object : cette classe Attributs permet de réifier les attributs.



Attributs :

nom : de type String qui définit le nom de l'attribut.

type : détermine le type de l'attribut. Pour l'instant, nous le considérons comme une chaîne de caractères.

statut : qui est de type booléen pour exprimer si l'attribut est permanent ou transitoire.

valeur : sous forme de chaîne de caractères pour contenir la valeur de l'attribut.

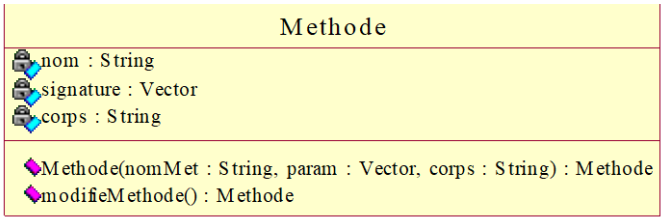
Méthodes :

public Attributs (String nomAtt, String typeAtt) : le constructeur, qui met à défaut le statut à vrai pour exprimer qu'il est permanent, et la valeur à chaîne nulle.

public modifieAttributs() : pour modifier un attribut. Cette méthode est redéfinie et surchargée de manière à gérer toutes les modifications possibles.

– **Classe Methode :**

public class Methode extends Object : cette classe Methode permet de réifier les méthodes.



Attributs :

nom : le nom de la méthode sous forme de chaîne de caractères.

signature : donne la liste de paramètres.

corps : chaîne de caractères.

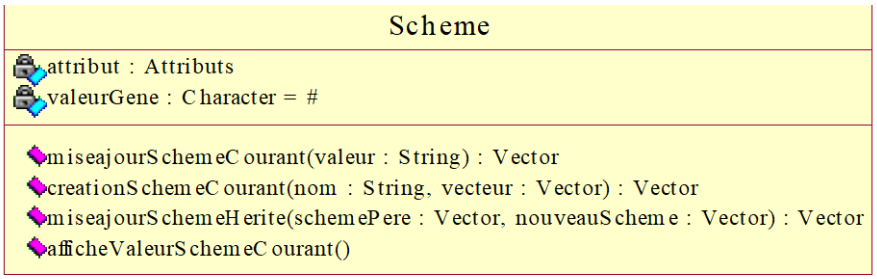
Méthodes :

public Methode(String nomMet, Vector param, String corps) : le constructeur qui permet de créer une méthode en spécifiant son nom, la liste de ses paramètres et le corps considéré comme une chaîne de caractères.

public modifieMethode() : pour modifier une méthode. Cette méthode est redéfinie et surchargée de manière à gérer toutes les modifications possibles.

– **Classe Scheme :**

public class Scheme extends Object : cette classe représente tout schème et permet donc de créer le schème courant et le stocke dans un vecteur.



Attributs :

attribut : est un ensemble d'attributs que référencent le schème. Il référence des instances de la classe **Attributs**.

valeurGene : est de type caractère. Il représente la valeur associé à l'attribut référencé. Il est initialisé par défaut à #. Il peut prendre également la valeur 0 ou 1.

Méthodes :

Scheme() : constructeur

public void afficheValeurSchemeCourant() : affiche le schème de la classe courante

public Vector creationSchemeCourant(String nom, Vector vecteur) : méthode qui crée le vecteur contenant le schème courant

Vector miseajourSchemeCourant(String valeur) : méthode qui crée le schème de la classe courante

public Vector miseajourSchemeHerite(Vector schemePere, Vector nouveauScheme) : méthode qui met à jour le schème de la classe courante en prenant en compte l'héritage.

– **Classe InstanceClasseGtype :**

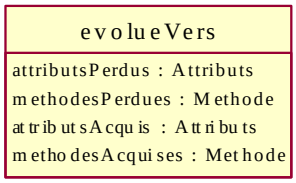
public class PatrimoineGenetique extends Object :

Méthodes :

public void afficheSaisie(Vector l) : méthode qui affiche l'interface de saisie d'une nouvelle classe et qui stocke la nouvelle classe avec ses valeurs dans le vecteur des classes. Elle est représentée par une instance de ClasseGtype.

– **Classe evolueVers :**

public class PatrimoineGenetique extends Object : c'est la classe qui représente un lien d'évolution.



6.3.2. Les processus d'évolution

Nous citons dans cette section quelques classes notables de l'application ainsi que les principales parties de l'application des processus d'évolution :

- **Class MiseaJourScheme :**

public class MiseaJourScheme extends Object : cette classe stocke le schème courant dans le vecteur des schèmes et met à jour le schème des autres classes. Elle définit entre autres les méthodes suivantes :

public void miseaJourSchemeToutesAutresClasses(Vector v) : méthode qui met à jour tous les autres schèmes.

public void StockageSchemeCourant(Vector v, String pere) : méthode qui stocke le schème courant dans le vecteur des schèmes

- **Class Genome :**

public class Genome extends Object implements ActionListener : cette classe instancie un objet Genome1. Elle permet de gérer l'interface graphique et définit la méthode *afficheSaisie(Vector)* qui affiche l'interface de saisie d'une nouvelle classe et qui stocke la nouvelle classe avec ses valeurs dans le vecteur des classes.

- **Class AfficheClasse :**

public class AfficheClasse extends Object implements ActionListener, WindowListener : cette classe crée l'IHM de visualisation des classes qui affiche tous les attributs et méthodes d'une classe.

- **Class AfficheMenu :**

public class AfficheMenu extends Object implements ActionListener : cette classe est l'IHM dans laquelle on peut choisir de créer une nouvelle classe et de visualiser n'importe quelle classe déjà créée. Elle détient le main (point de lancement).

– **Class AfficheVisu :**

public class AfficheVisu extends Object implements ActionListener : cette classe affiche la liste des classes déjà créées.

Méthodes intéressantes :

public void creeListedesClasses(Vector v) : méthode qui crée la liste des classes à afficher dans cette interface.

public void methodeAfficheVisu() : méthode qui affiche la liste des classes déjà créés.

– **Les processus d'évolution :** nous présentons non pas le détail du programme, mais uniquement l'intitulé des différentes étapes :

- préparation du tableau de population en le mettant à zéro
- saisie ou extraction du schème temporaire
- Recherche des parents et chargement de la population via les schèmes permanents

corps du programme

- affichage des parents
- génération de la population
- fin de la génération

Croisement

- choix des géniteurs dans la population
- pondération des VA
- /* création des enfants */

```
for(i=0;i<longueur_chem;i++)
       {
       valeatoir=Math.random();

       if(valeatoir>va_ponde)
              {
              e1[0][i]=tablo_population[p1][i];
              e2[0][i]=tablo_population[p2][i];
              }
       else
              {
              e1[0][i]=tablo_population[p2][i];
              e2[0][i]=tablo_population[p1][i];
              }
       e1[1][i]=-1;
       e2[1][i]=-1;
       }
```

étude des contraintes de blocs

- mise à jour des schèmes des enfants en appliquant les contraintes de blocs de gènes
- entrée des enfants i dans la population à la place des parents
- calcul des va de chaque enfant i
- fin de calcul des va des ei
- enregistrement de va max
- affichage de population
- affichage des va_max

6.3.3. Un exemple de Présage

Nous reprenons l'exemple de Présage d'un réseau urbain, représenté par une instance, et les diverses évolutions qu'il peut connaître. Cet exemple a été introduit dans le chapitre 1.

L'état initial de l'instance représente un réseau de télécommunications qui a été créé en utilisant les éléments contenus dans la classe **Cl-Réseau Urbain.2**. Cette instance représente les voies de communications d'un réseau urbain. Avec cette représentation, le concepteur réalise rapidement qu'il obtient une saturation au niveau de la demande pour les deux anneaux de type '**STM1**' (à droite de la Figure 1-2). Il faut donc remédier à ce problème. Une solution serait de remplacer les deux anneaux de type '**STM1**' par un anneau de type '**STM4**' qui lui permet de faire passer quatre fois plus d'informations qu'un anneau de type '**STM1**'. Voici comment le concepteur voudrait pouvoir manipuler les réseaux existants. Mais en même temps, il faut également tenir compte de la nature expérimentale de la construction du modèle, comme c'est le cas pour la plupart des modèles. Avec GENOME, le concepteur est en mesure d'exprimer les différents nouveaux besoins :

Nous remarquons dans cet exemple que le concepteur apporte les changements suivants à la structure, aux composants et aux types de composants du graphe du réseau urbain modélisé :

– les arêtes existantes **rstm1.4** et **rstm1.3** du type existant d'arêtes **RSTM1** ont été détruites ;

– de nouvelles arêtes, **système.1** et **système.2**, du type existant d'arêtes **système** ont été ajoutées ;

– une nouvelle arête, **rstm4.1**, d'un nouveau type d'arêtes **RSTM4** ont été ajoutés (le type et l'arête ont été ajoutés) ;

– les nœuds existants **stm1.6** à **stm1.8** du type de nœuds existant **STM1** ont été remplacées ;

– de nouveaux nœuds, **stm4.1** à **stm4.4**, d'un nouveau type de nœuds **STM4** ont été ajoutés.

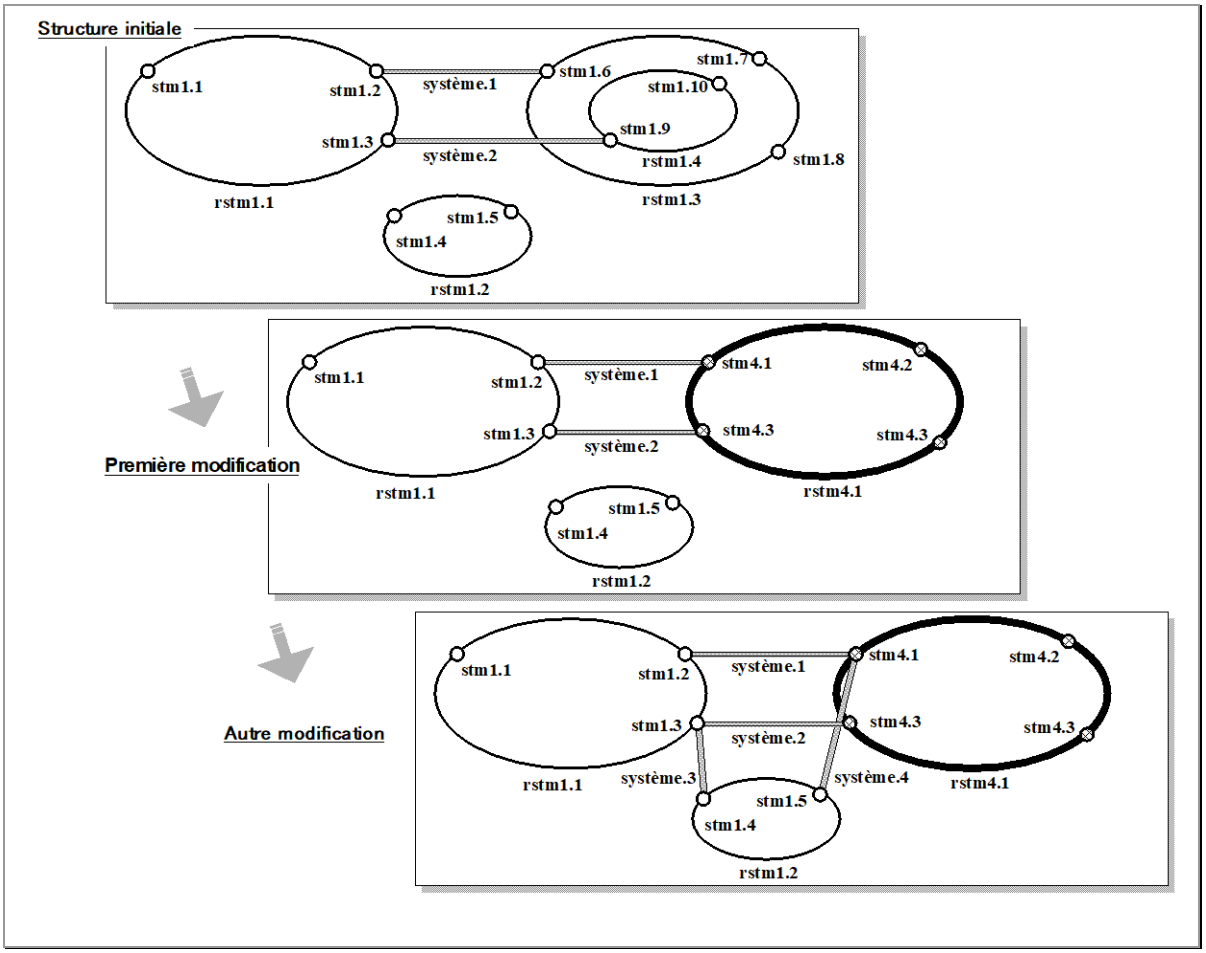


Figure 6-3: un exemple d'une instance, donc d'un réseau de télécommunications, et les changements que veut y apporter le concepteur.

Le concepteur a ainsi exprimé ses besoins d'évolution sur une instance existante, dans le cadre de sa tâche de planification de réseaux de télécommunications. Le processus d'évolution permet l'émergence d'une nouvelle structure conceptuelle représentée par la classe **Cl-Réseau Urbain.3** :

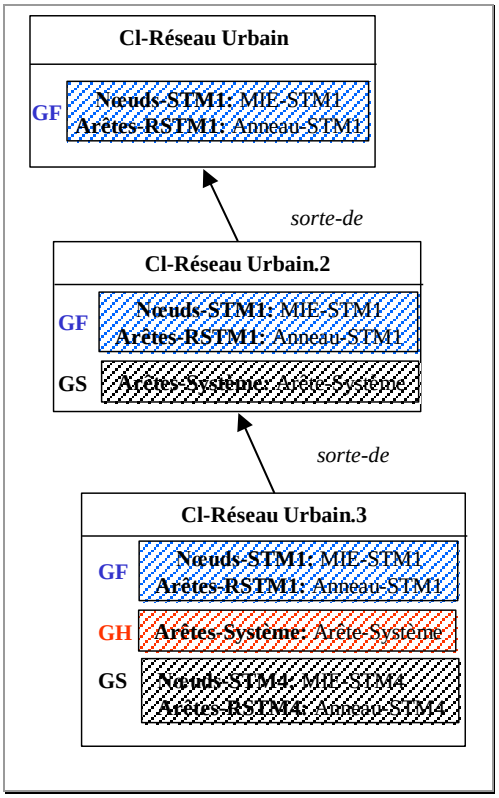


Figure 6-4: résultat de l'évolution.

Sur cet exemple, nous mettons en évidence le GF caractérisant tous les réseaux urbains, ainsi que la nouvelle classe émergente, définie au travers du GF, de son GH et du GS qu'elle introduit.

6.4 - Descriptif et état d'avancement

Les deux principales parties de GENOME ont été développées : l'aspect des processus d'évolution a été développé avec le JDK 1.0 et l'aspect de l'implémentation du méta-modèle a été développé avec le JDK 1.1.5 sur Unix. L'implémentation du modèle GENOME, a été menée dans le cadre de l'extension de Présage, et dans le cadre de projets avec des étudiants. Dans un souci d'uniformité, elle doit prendre en compte les deux niveaux CA et UF, car elle doit offrir de nouvelles fonctionnalités pour Présage, dans le cadre de la manipulation et l'évolution des réseaux de télécommunications.

De plus, le langage Java ne présentant pas de méta-niveau, le modèle est donc implémenté selon le modèle de Présage, à savoir avec des classes (représentant en fait le niveau CA) dont les instances seront par la suite transformées pour définir des classes (représentant en fait le niveau UF) qui seront à leur tour instanciées de manière terminale.

Nous nous sommes attelés à la première partie. La seconde devrait se poursuivre dans le cadre de l'extension de Présage avec GENOME.

L'implémentation s'est faite chronologiquement en deux phases :

1- Le processus d'émergence globale par croisement ;

2- Le modèle de GENOME.

Les raisons de ces choix de développement sont explicitées ci-après :

6.4.1. Processus d'émergence globale par croisement

En effet, la première partie à avoir été développée a été celle de l'émergence globale par croisement. Il nous était effectivement nécessaire de développer le croisement à des fins de validation et de vérification du croisement tel que nous l'avions défini.

Dans le cadre de la validation du croisement, les caractéristiques qui nous ont intéressées sont résumées comme suit :

– *La manipulation directe d'un codage des paramètres, et non des paramètres eux-mêmes :* le codage des paramètres étant le schème et les paramètres étant les attributs ;

– *L'exploration au moyen d'une population de solutions, et non d'une solution unique :* la population de solutions étant la population de classes, ou plus précisément la population des schèmes permanents des classes partiellement adaptées ; la solution unique étant le schème temporaire, donc l'abstraction véhiculée par l'instance ayant évolué ;

– *L'exploration 'aveugle' à partir uniquement des données de la situation :* l'exploration 'aveugle' est un qualificatif utilisé par les Algorithmes Génétiques. Il revient pour nous à l'exploration de l'ensemble de la population de solutions et le croisement effectué en introduisant de l'aléa dans le croisement et donc dans la transmission des gènes manipulés. Les données de la situation sont : les schèmes permanents, les contraintes de blocs de GH et de GS de ces schèmes à respecter, les invariants exprimés par le concepteur dans le modèle ainsi que ceux, éventuellement, exprimés directement sur les instances ayant évolué.

– *L'utilisation pour l'optimisation de la solution finale, d'opérateurs stochastiques :* et non de règles purement déterministes.

6.4.1.1. Validation et amélioration du croisement

L'implémentation du croisement nous a été particulièrement utile. Grâce à elle, des améliorations importantes ont été apportées au croisement, principalement l'uniformisation de deux schèmes avant le croisement, l'utilisation de Valeurs d'Adaptation pondérées Vap et l'application des contraintes de blocs de gènes.

Ce dernier point a été initialement introduit uniquement pour des raisons sémantiques. Lors de son implémentation, nous avons réalisé que les résultats du croisement s'étaient

nettement améliorés. Avant cela, les résultats obtenus du croisement, donc les Va, tendaient à s'améliorer, mais restaient généralement en deçà de la solution idéale, c'est-à-dire celle ayant une Va à 1. Sur un nombre d'itérations prédéfini (i=100), les meilleures des résultats obtenus variaient autour d'une Va = 0.6 ±{0.1 , 0.2}.

L'introduction de la contrainte de blocs a permis de converger les résultats plus rapidement (une amélioration de l'ordre de plus de 25%) et la Va idéale était plus souvent atteinte (Va=1), sans pour autant diminuer de la diversité des schèmes émergents obtenus.

Le principe d'exécution est d'introduire un schème temporaire et de le comparer avec la population des 20 schèmes. Ces jeux d'essais sont appliqués plusieurs fois sur cette population fixe de 20 schèmes différents, une population fixe de 20 individus, une longueur de schème fixée arbitrairement à 8 gènes, et le GF codé sur un gène (il n'intervient pas dans le croisement). Les résultats constatés montrent l'apparition d'individus dans la population : c'est *l'émergence d'une solution non prévue.*

La constatation finale que nous avons faite de l'évolution des Va maximales est représentée dans le graphe suivant :

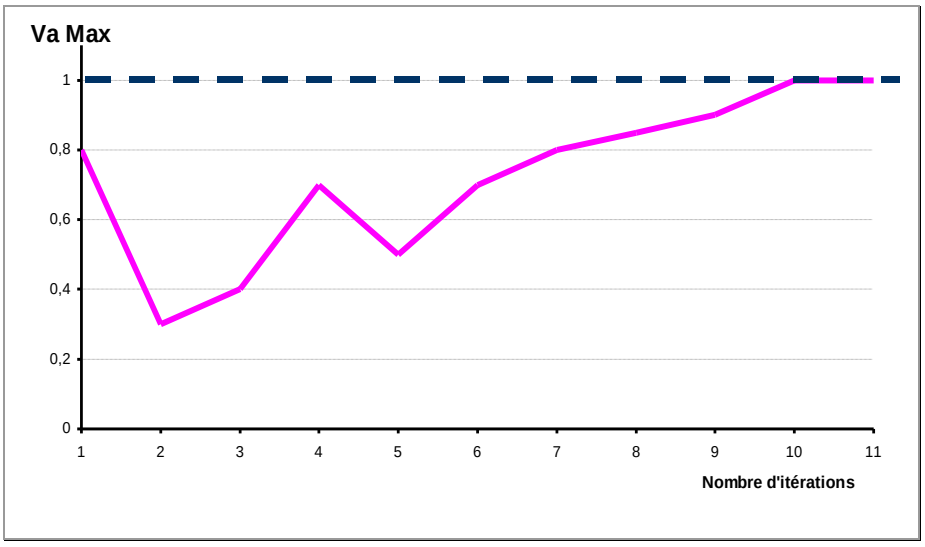


Figure 6-5: Évolution de la Va Max au cours des itérations du croisement.

La Va maximale (la plus grande Va de chaque génération de croisement) évolue au cours des générations du croisement. Comme l'indique le graphique, lors des premières itérations, la Va maximale baisse sensiblement, avant de remonter asymptotiquement, en rattrapant d'abord le niveau initial (donc les Va initiales, celles des schèmes permanents) puis de le dépasser et se rapprocher de la valeur maximale 1, atteinte après un nombre d'itérations du croisement. Le nombre d'itérations avant la convergence de la Va vers 1 varie selon les schèmes permanents dont on dispose.

Cette dégradation des performances avant une amélioration est connue dans le cadre des Algorithmes Génétiques.

6.4.1.2. Deuxième étape dans l'implémentation du croisement

Les limites de cette première étape de l'implémentation du croisement était le codage figé des schèmes permanents (nous considérions que nous travaillions sur une même population, d'où les schèmes permanents figés). Mais cette étape était nécessaire pour valider et même améliorer le croisement.

La seconde étape consiste à rendre le processus plus générique en le rendant modulaire et paramétrable, le rendre donc plus générique. Il est donc devenu nécessaire de :

- travailler sur les schèmes permanents de population de classes, donc de schèmes de longueur variables (d'où l'introduction de l'uniformisation de schèmes deux à deux avant de les comparer, ainsi que des Vap) ;
- manipuler pour ajout, modification, destruction de classes, d'instances et d'attributs ;
- extraire des schèmes temporaires des instances ayant évolué.

Cela revient à l'implémentation des deux phases d'extraction et d'exploration.

Des améliorations ont été également nécessaires à apporter au croisement :

- le croisement doit pouvoir être déclenché à la demande du concepteur, soit après la modification d'une instance, soit après avoir modifié plusieurs instances ;
- une autre option est de faire un croisement 'à la carte', c'est-à-dire uniquement sur deux schèmes, pour des besoins de simulations ponctuelles et d'évaluation.

Aussi, le croisement tel qu'obtenu en premier a été intégré comme un noyau dans un autre module qui permet de travailler via une interface graphique sur la simulation d'un modèle de classes avec des attributs mais aussi la sélection d'une instance et de provoquer son évolution structurelle de manière dynamique. Il s'ensuit une extraction des schèmes temporaires et de l'accomplissement du croisement.

Ainsi, une fois que le cœur du modèle a été vérifié et validé, puis complété avec les deux premières phases, la seconde étape de l'implémentation a été celle des concepts du modèle, tel qu'il est présenté en section 6.3 - .

6.4.2. État d'avancement du prototype

L'avancement et les manques de l'implémentation de GENOME sont abordés dans cette section, tout en précisant les extensions nécessaires à apporter à court terme :

6.4.2.1. Le modèle de GENOME

Les concepts du modèle de GENOME sont spécifiés : le concept de Classe-Gtype, la distinction entre les trois différents génotypes ainsi que la transmission du GF et du GH entre les classes d'une population, la simulation du lien d'héritage ainsi que la spécification du schème permanent lors de la définition d'une classe et la possibilité de définir les liens d'évolution.

Cependant, il n'est pas encore possible de redéfinir le GH et l'interface graphique ne permet pas encore de représenter tout le modèle (tel que le schème qui n'est pas encore manipulé graphiquement mais ses états apparaissent sur la ligne de commande uniquement). Il n'y a pas encore la transformation du modèle des instances en classes pour qu'elles puissent être à leur tout instanciées.

Nous jugeons que plusieurs aspects doivent être étendus :

– Approfondir la simulation du lien d'héritage en permettant la redéfinition des gènes hérités ainsi que la modification du schème permanent en conséquence ;

– Spécifier le lien d'évolution (il ne pourra être utilisé qu'en développant la jonction entre le modèle statique et le modèle dynamique) ;

– Simuler l'instanciation d'une classe et permettre la persistance des données (classes et instances), ce qui permettrait de récupérer une base de données existante sur laquelle peuvent être simulées des évolutions. Cela doit se poursuivre dans le cadre de Présage ;

6.4.2.2. Les Processus d'Évolution de GENOME

Dans le cadre des processus d'évolution, l'effort s'est porté essentiellement sur le processus d'émergence par croisement. De plus, le croisement peut aussi bien se faire sur une instance ayant évolué par rapport à des classes existantes, qu'entre deux instances au choix pour un besoin de simulation. Nous avons également simulé la phase d'extraction de schèmes temporaires sur des exemples simples.

Cependant, la phase d'extraction doit être plus générique. Les suites de la phase d'exploitation, après avoir mené le croisement, n'a pas encore été développée, principalement l'analyse et l'insertion d'abstractions émergentes dans le modèle de classes, la gestion des différents impacts, ainsi que le modèle de métriques.

Pour répondre à ces manques, il faut développer la phase d'exploitation, principalement :

- La mesure et l'analyse de l'émergence ;
- Le modèle de métriques ;
- L'insertion des abstractions émergentes ;
- La gestion des impacts de cette insertion ;
- Les spécification du lien d'évolution.

6.4.2.3. Complexité du croisement

Étant donné que le croisement de GENOME se base sur le croisement des Algorithmes Génétiques, ses résultats et ses performances sont semblables à ces derniers. Il est connu dans le domaine des Algorithmes Génétiques, que la convergence du croisement ne peut être prouvée de manière théorique, et que les résultats dépendent plutôt des cas concrets de leur utilisation. Dans le cadre de nos travaux, et principalement dans le cadre de la simulation de l'évolution par émergence globale, nous avons défini au niveau du croisement trois critères d'arrêts exclusifs :

- lorsqu'au moins un schème émergent (Va=1) apparaît ;
- lorsque le concepteur décide à l'avance d'un certain nombre d'itérations pour le croisement d'une population ;
- lorsque les résultats du croisement se détériorent de manière très sensible.

De plus, au vu des différents paramètres que nous utilisons lors du croisement, principalement la contrainte de blocs de gènes, les résultats des tests du croisement permettaient toujours d'obtenir une bonne convergence du croisement pour une même population.

6.5 - Conclusion

Ce chapitre met en évidence d'une part l'infrastructure de Présage, dans le cadre duquel le modèle GENOME doit s'intégrer, et d'autre part, l'implémentation du modèle de GENOME et de ses processus d'évolution. En l'état actuel, les deux parties sont indépendantes. La jonction doit permettre de spécifier le modèle de manière complète et d'assurer une cohérence respective :

– Les processus d'évolution se baseront alors sur le modèle de classes et d'instances pour s'exécuter ;

– Les résultats du processus de développement et d'émergence seront intégrés et répercutés sur le modèle de classes.

CONCLUSION : CONTRIBUTIONS ET PERSPECTIVES

7.1 - Contributions de nos travaux

La problématique ayant motivé cette thèse s'inscrit dans le cadre plus général qu'est l'évolution d'objets [Oussalah& 99], [Dony& 97], [Nguyen& 89], [Lerner& 90]… Nos travaux se sont intéressés à l'évolution d'objets sous un angle particulier, celui de l'évolution initiée à partir de l'évolution dynamique de la structure d'instances avec impacts sur les classes, par modification de ces dernières au sein de leur hiérarchie. Ce dernier point constitue le point commun avec les stratégies d'évolution d'objets par modification de classes, telles que leur *réorganisation* [Casais 91], leur *classification* [Capponi 94], leur *catégorisation* [Napoli 92], ou leur *correction* [Banerjee& 87].

Comme nous l'avions préalablement souligné dans l'état de l'art (Chapitre 2), la plupart des stratégies d'évolution s'intéresse à l'évolution de classes et de schémas de classes. Notre approche est complémentaire. A notre connaissance, peu de travaux ([Li& 94]) ont traité l'évolution d'objets sous cet angle.

En effet, ces principales stratégies (présentées en chapitre 2) sont nécessaires et utiles pour mener à bien l'évolution à partir des classes, chacune selon sa politique et donc son adéquation à la situation. Mais elles ne peuvent être utiles que s'il est possible au concepteur de définir a priori l'évolution. Or, au vu de l'expérience acquise, suite à une maturation et un recul certains, le besoin de pouvoir manipuler, dans une certaine mesure les spécifications existantes à partir d'instances, se fait pesamment ressentir. Ainsi, un concepteur ayant la charge d'applications complexes et volumineuses rencontre souvent ce type de problèmes.

Nous avons proposé GENOME, un modèle objet pour la simulation de l'évolution d'objets à partir de l'évolution dynamique de la structure d'instances. Ce modèle trouve toute son utilité sur des applications objets complexes et volumineuses. Nous l'avons proposé comme un outil d'aide à la maintenance et à l'évolution, pour un utilisateur particulier : le *concepteur* ayant la charge de ce type d'applications.

Un concepteur rencontre, dans l'accomplissement de sa tâche, de nouveaux besoins qui sont partiellement ou mal spécifiés, tout au long du cycle de vie des applications. Or, dans le cadre d'applications complexes et volumineuses (Réseaux de télécommunications [Caminada 92], CAO [Rieu& 92]…), il est parfois très difficile de cerner à quel endroit de l'application il est nécessaire d'apporter des modifications au niveau des spécifications conceptuelles existantes. De plus, ces besoins apparaissent généralement lors du déroulement des applications, donc de la manipulation des instances. Il nous est apparu fort utile de pouvoir manipuler ces mêmes instances dans leur structure pour refléter l'évolution et amorcer les impacts sur le niveau des classes.

Il n'a nullement été question dans le cadre de nos travaux d'accorder aux instances les mêmes prérogatives ni la même importance qu'aux classes. Nous nous sommes bien situés dans un cadre de modélisation et d'évolution d'objets, et nous considérons que dans un état stable, la dualité classe/instance, telle que définie dans l'approche objet [Masini 89], est et reste respectée dans un modèle. Nous considérons toutefois qu'au vu de certains de ses aspects rigides, la modélisation objet est contraignante dans certaines situations. Nous avons pour cela pensé que l'évolution d'objets gagnerait en richesse et en intérêt en permettant, dans ces situations, d'amorcer et de simuler l'évolution et ses conséquences par le biais de l'évolution dynamique de la structure d'instances.

Nous pouvons situer la contribution de nos travaux sur deux plans :

- l'état de l'art ;
- une approche pour l'évolution d'objets basée sur les instances.

7.1.1. D'un point de vue de l'État de l'Art

Nous avons proposé une classification de l'évolution d'objets basée sur trois *facettes* :

1- L'*Objet de l'évolution* : l'évolution peut porter sur un *produit* ou un *processus*.

2- Le *Type de l'évolution* : lorsque les besoins d'évolution sont pris en compte lors de l'analyse et spécifiés lors de la conception, l'évolution est dite *préventive*. Lorsque de nouveaux besoins apparaissent ou qu'ils aient été mal ou partiellement définis, l'évolution est dite *curative*.

3- Le *Processus d'évolution* : nous distinguons deux types de processus d'évolution : le *développement* et l'*émergence*. Le premier concerne l'évolution des classes et leurs impacts sur les sous-classes et les instances correspondantes. Le second concerne l'évolution d'instances avec des impacts sur les classes.

Nous avons mis en évidence que toute stratégie d'évolution peut être positionnée dans cette représentation triptyque en répondant pour chacune d'entre elles aux questions suivantes :

1- *Quel est l'objet de sa politique d'évolution :* ***produit*** *ou* ***processus*** *?*

2- *Quel est le type d'évolution qu'elle propose : **curative** ou **préventive** ?*

3- *Quel type de processus d'évolution permet-elle : **développement** ou **émergence** ?*

Nous avons souligné que cette classification offre un double avantage :

- celui de situer une stratégie d'évolution par rapport à l'évolution et non par rapport à une stratégie donnée, ce qui permet une comparaison uniforme ;
- celui de situer les aspects de l'évolution d'objets qui ont largement été traités, peu traités ou jamais traités.

7.1.2. D'un point de vue de l'Évolution d'objets

Notre approche, comme nous l'avons précédemment rappelé, se base sur l'évolution initiée à partir de celles d'instances existantes.

Comme nous considérons que les instances sont des entités détenant leur propre connaissance (via leurs classes), elles peuvent faire évoluer leur connaissance, au vu des perturbations émanant de l'environnement changeant et instable dans lequel elles évoluent. Afin de prendre en compte ces informations nouvellement présentes dans certaines instances, et afin d'assurer une évolution harmonieuse et sans rupture majeure, tout en veillant à tirer profit au maximum des informations préalablement existantes, nous avons défini les processus d'évolution et les concepts du modèle GENOME. En amont, nous avons situé notre contexte et nos hypothèses de travail.

Nous nous sommes inspirés de quelques principes, notions et concepts des travaux de la Vie Artificielle et des Algorithmes Génétiques, car les individus qu'ils définissent peuvent s'adapter et adapter leur propre connaissance dans un environnement instable. Les nouveaux individus qui apparaissent sont beaucoup mieux adaptés que leurs prédécesseurs.

Le principal nouvel aspect que nous introduisons dans nos travaux est le croisement d'entités (qu'elles soient instances ou classes) dans leurs spécifications. Il permet l'interchangement de spécifications, en introduisant de l'aléa mais tout en garantissant un minimum de cohérence (la contrainte des blocs). Le but est de dégager de nouvelles structures conceptuelles, auparavant inexistantes, mieux adaptées aux nouveaux besoins.

7.1.2.1. Le principe de l'évolution d'objets sous GENOME

Suite à l'évolution de la structure d'une instance, nous considérons que tout processus d'évolution s'exécute en trois phases : une phase d'*extraction*, une phase d'*exploration*, et enfin une phase d'*exploitation*. Nous rappelons le principe de chaque phase :

– **Phase d'extraction :** elle consiste en la détection de l'évolution de l'objet et l'extraction de sa constitution, que nous qualifions de son *code génétique*.

– **Phase d'exploration :** elle consiste à explorer l'ensemble des classes du modèle par comparaison du code génétique de l'instance ayant évolué et celui de chaque classe. L'objectif est de repérer les classes qui détiennent, ne serait ce que partiellement, les caractéristiques présentes dans l'objet, après que celui-ci aura évolué.

– **Phase d'exploitation :** elle permet l'aboutissement de l'évolution en gérant les différents impacts par un développement ou une émergence :

- *Processus de développement :* ils représentent les impacts d'évolution des classes sur les instances.
- *Processus d'émergence locale et d'émergence globale :* ils représentent les impacts d'évolution d'objets vers les classes. Ils concernent toute émergence d'une nouvelle information conceptuelle à partir d'instances. Il y a deux issues possibles à cette évolution :
  - *Émergence locale :* elle concerne l'émergence de nouvelles informations au sein de classe(s) existante(s). Le code génétique de l'objet a évolué et peut faire évoluer celui de sa classe ou d'une classe sémantiquement voisine.
  - *Émergence globale :* elle concerne l'émergence d'une nouvelle structure conceptuelle (donc de classe(s)) soit par projection directe au sein de l'objet ayant évolué, soit par *croisement* d'entités existantes.

7.1.2.2. Le principe de la modélisation sous GENOME

Nous avons défini les concepts de GENOME pour modéliser les classes et les instances ainsi que leurs caractéristiques pour pouvoir mener à bien les processus d'évolution, et principalement l'émergence. Nous avons notamment mis en évidence les notions de :

– *Population :* est l'ensemble des classes qui définissent les différentes abstractions d'une même entité de la situation réelle modélisée ;

– *Génotype Fondamental :* est cet ensemble particuliers d'attributs et de méthodes qui représente le noyau commun et indivisible à toutes les classes formant une population ;

– *Le Génotype Hérité et le Génotype Spécifique :* représentent respectivement les caractéristiques héritées de la super-classe et les caractéristiques propres à la classe ;

– *Le schème :* qu'il soit *permanent* (associé à une classe) ou *temporaire* (associé à une instance), le schème est cette entité exprimé de manière très précise et pour ainsi dire binaire. Il représente le code génétique d'une entité manipulé durant les étapes de tout processus d'évolution.

Nous abordons en premier lieu dans la section suivante une réflexion générale sur la problématique de l'évolution d'objets, avant de poursuivre avec les limites et perspectives de notre modèle.

7.2 - Limites et Perspectives

Comme nous l'avons cité, chaque stratégie d'évolution est bien adaptée pour mener à bien l'évolution dans un contexte et pour un besoin précis, mais chacune est assez souvent moins efficace en dehors de ce contexte. Cependant, la plupart de ces stratégies sont capables de faire face à des besoins connus et les intègrent durant les phases d'analyse et de conception, le tout en accord avec la politique d'évolution de la stratégie adoptée. Nous pensons que dans le cadre de la problématique générale de l'évolution d'objets, il est avant tout nécessaire d'élaborer des méthodologies pour gérer l'évolution principalement en :

- Combinant des approches d'évolution existantes ;
- S'appuyant sur des méta-modèles d'évolution ;
- Utilisant des techniques pour contrôler et mener à bien l'évolution ;
- Permettant, finalement, l'évolution d'objets durant le cycle de vie. Cela peut être assuré en prenant en compte les nouveaux besoins lorsqu'ils apparaissent sans provoquer de ruptures préjudiciables ni de perturbations notables.

Nous présentons les limites et perspectives par chapitre du manuscrit. Ensuite, nous les présentons par ordre de priorité jugé le plus approprié pour la suite des travaux :

6.5.1. Limites et perspectives par chapitre

– **Chapitre 3 : l'état de l'art :** nous avons défini dans ce chapitre une classification de l'évolution d'objets basée sur ses trois facettes. Nous avons ensuite étudié les principales stratégies d'évolution basées sur la modification de classes et de schémas de classes. Cette étude nous a permis de distinguer une taxonomie d'opérations d'évolution en opérations de base et opérations avancées. Ces dernières utilisent les premières au vu de la politique d'évolution adoptée. Nous pensons que cette étude gagnerait à :

– d'une part, être étendue aux autres facettes,

– et d'autre part d'établir des liens entre les différents éléments ainsi énumérés de chacune de ces facettes.

Ainsi, non seulement l'évolution peut être définie en situant quelle partie de chacune des facettes est concernée, mais également quel élément de chaque facette peut être relié à quel autre élément de chacune des deux autres facettes. Nous pourrions alors définir des triplets pour définir pleinement toutes les phases d'évolution. Ainsi, la connaissance d'un élément peut mener à la connaissance des autres éléments concernés

via justement ce triplet qui les réunis. Nous pensons qu'une telle démarche peut largement contribuer à l'analyse et la conception de l'évolution ainsi qu'à sa maîtrise et sa maintenance.

– **Chapitre 4 : les concepts :** nous avons présenté les concepts du modèle GENOME. Nous les avons illustrés par un exemple d'une population de classes. Nous avons manipulé, à titre d'exemple, quelques-unes unes des instances. GENOME est défini pour faire face à l'émergence de structures conceptuelles suite à l'évolution d'instances simples … vers d'autres instances simples. Nous estimons utile d'étendre le modèle de manière à prendre en compte des **objets composites** et des **objets complexes**. Un cas intéressant peut en effet survenir : celui de l'évolution d'une instance simple en une **instance composite** ou **complexe**. Actuellement, notre modèle ne peut pas répondre à une telle situation. En effet, une telle instance est constituée d'instances reliées entre elles par des liens. Ces instances peuvent ne pas appartenir à la même population. Nous aurions alors une instance composée d'instances de plusieurs populations, donc invoquant plusieurs Génotypes Fondamentaux. Ainsi, le processus d'évolution se trouvera dans une situation embarrassante puisqu'il ne détectera aucune population pleinement adaptée avec le (les ?) GF de l'instance.

Pour cela, nous estimons qu'il est intéressant et utile d'étendre les trois phases d'évolution de manière à intégrer et répondre à l'évolution d'objets composites et complexes. Nous devons pour cela intégrer les relations sémantiques dans notre modèle et de prévoir une gestion dynamique de ces relations. Nous pourrions pour cela utiliser des travaux déjà existants, tels ceux présentés dans [Rashid& 99] qui s'intéressent à l'évolution dynamique des relations dans le cadre d'applications de BDOO volumineuses.

Comme conséquence, si nous pouvons exprimer un lien de référence dans les spécifications d'une classe, nous considérons qu'il est nécessaire d'étendre le concept de schème en distinguant dans sa structure les attributs simples des attributs composites et complexes. Nous avons déjà abordé cette éventualité dans [Tamzalit& 98a] : nous avons proposé, dans le cadre d'émergence globale d'objets complexes, d'appliquer un *filtrage*. Un attribut composé revient à exprimer un lien entre l'objet ayant évolué et d'autres objets. Nous considérons les trois grandes catégories de liens que sont l'héritage, la composition et l'association.

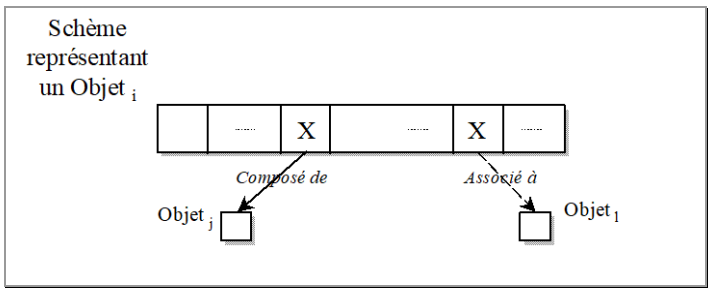


Figure 6-6 : Composition et association entre objets

Un autre aspect nous semble intéressant : celui de pouvoir définir des sous-populations au sein d'une même population. Si un concepteur doit modéliser une situation très complexe, il peut avoir besoin d'organiser les classes et les hiérarchies de classes en populations et sous-populations. Cela revient essentiellement à la manière de définir et de considérer les GF ainsi que les liens entre les GF de sous-populations entre elles et avec la population 'mère'.

Un autre aspect nous paraît également très prometteur : celui de la spécification de méthodes et l'extension plus marquée du croisement sur les méthodes. A terme, la simulation de l'évolution pourra, selon le choix et les besoins, être amorcée sur l'évolution de spécifications de méthodes ou d'attributs, voire une combinaison des deux. Cet aspect pourra s'appuyer sur la proposition d'extension de la taxonomie des facettes de l'évolution précitée. Nous considérons également que cette extension pour prendre en charge les méthodes au même titre que les attributs est un bon point d'ouverture vers les travaux en Génie Logiciel, principalement ceux de la rétro-ingenierie.

– **Chapitre 5 : les processus d'évolution**

Quelques pistes nous semblent intéressant pour dépasser les limites des processus d'évolution tels que nous les avons définis et assurer leur amélioration et leur performances :

- Pour l'instant nous avons présenté les processus d'évolution comme étant déclenchés dès qu'une instance évolue dans sa structure. En terme d'ouverture, l'idée serait de pouvoir établir un modèle de déclenchement des processus d'évolution : des règles à respecter, des priorités qui s'imposent, des quotas, de l'importance ou non des changements, du nombre d'instances modifiées, des états dans lesquels le modèle ne doit pas être modifié, des états où il pourra être modifié… Nous pourrions ainsi en partie automatiser le choix du moment propice au déclenchement de l'évolution. Pour l'instant, étant dans un cadre de simulation d'évolution, c'est le concepteur qui déclenche le processus une fois qu'il a exprimé les changements sur les instances.
- Nous proposons également d'assainir le modèle en l'optimisant dès qu'un certain nombre de changements sont effectués. On pourrait alors utiliser les techniques existantes pour l'optimisation et la factorisation des hiérarchies de classes ([Casais 91], [Napoli 92]…).
- Dans le cadre de l'utilisation du développement (soit directement, soit indirectement pour gérer les impacts de l'émergence), il serait intéressant d'étendre la possibilité d'utiliser n'importe quelle stratégie d'évolution et non pas nous intéresser principalement à celles qui font de l'évolution par modification de classes. Nous proposons d'étendre également aux stratégies d'évolution par dérivation et par versionnement. Ainsi, lorsque le

concepteur paramètre l'outil de simulation sur son application, il pourra entre autres choisir quelle stratégie il voudrait voir appliquer dans la phase de développement pour gérer les impacts de l'émergence. Si par exemple il veut garder un historique, il pourra choisir le versionnement. Ce choix de stratégies à appliquer pour la gestion des impacts de l'émergence dans le cadre du développement constitue un des invariants que pourra imposer le concepteur. Il pourrait donc influencer le choix des métriques et le choix des résultats de l'émergence. Il permet également de garder éventuellement des stratégies d'évolution déjà existantes dans son environnement et/ou auxquelles il serait habitué.

Nous présentons également quelques perspectives pour deux aspects particuliers du modèle :

- Nous proposons d'accorder un poids par lien d'évolution. Le poids d'un lien augmentera à chaque fois qu'une instance de sa classe source l'empruntera pour évoluer vers sa classe cible. Ces poids nous donneront de bons indicateurs sur la tendance générale d'évolution des instances du modèle, et ce sur un intervalle de temps déterminé.
- Dans le cadre de l'émergence locale, cela ne peut être possible que sur une classe concrète (étant donné que l'émergence de nouveaux attributs vient d'une instance qui devra être rattachée à cette classe, cette dernière ne peut être que concrète). Par contre, et ce en terme de perspective, si toutes les sous-classes d'une classe abstraite voient émerger les mêmes nouveaux attributs, l'émergence locale se fera alors sur la classe abstraite. Cela reste dans le cadre de l'optimisation du modèle au bout d'un certain temps d'utilisation, et en ayant recours à des stratégies d'évolution existantes, principalement pour la factorisation de propriétés communes.

6.5.2. Priorité des perspectives

Nous présentons les perspectives selon l'ordre de priorité qui nous semble le plus approprié. Nous prenons également en compte l'implémentation :

1. **Implémentation:**

Dans le cadre des prévisions du projet, nous prévoyons de compléter le prototype de en poursuivant par l'implémentation :

1. du modèle de métriques de contexte intra-structure et de contexte inter-structures ;
2. de la gestion de l'insertion des structures émergeantes dans la hiérarchie de classes. Cette gestion implique également celle des impacts de cette insertion

dans le cadre du développement, et ce en utilisant des stratégies d'évolution déjà existantes et bien adaptées à la tâche ;

3. du lien d'évolution et des graphes d'évolution ;
4. enfin, du processus d'évolution par émergence locale.

2. Extension du modèle GENOME : quant à l'extension du modèle, nous récapitulons la suite de son extension et de son enrichissement par :

1. la prise en compte de la modélisation et de l'évolution d'objets composites et complexes, ce qui implique la prise en compte également de la modélisation et de l'évolution des relations sémantiques ;
2. la combinaison de diverses stratégies d'évolution pour gérer le développement ;
3. l'extension des règles et des situations de déclenchement des processus d'évolution ainsi que l'optimisation des processus d'évolution. Dans le cadre de l'optimisation des processus d'évolution, nous reprendront les automates définis pour les décrire avec des diagrammes d'activités ;
4. la spécification de l'évolution dynamique également pour les méthodes

3. Méthodologie : cet aspect méthodologique est une carence récurrente dans toutes les stratégies d'évolution (versionnement, catégorisation…). Il en est de même pour notre modèle d'évolution. Nous devons définir une méthodologie claire et qui est destinée au concepteur pour lui permettre d'utiliser les fonctionnalités de GENOME, et pour lui permettre d'intervenir durant tout le processus d'évolution.

Après ces quelques perspectives, qui mettent en évidence en même temps les limites que nous recensons au modèle proposé, nous achevons cette conclusion en rappelant que nous nous sommes intéressés à l'évolution dans la structure d'instances et de classes, en tentant de dégager des structures conceptuelles à partir d'instances. Nous visons en effet, à tirer profit d'informations structurelles présentes aussi bien au niveau des instances (niveau implémentation) qu'au niveau des classes (niveau conceptuel). Le résultat obtenu, ou plutôt les résultats obtenus (étant donné que plusieurs voies possibles d'évolution peuvent être obtenues après simulation), ne sont pas une transcription conceptuelle d'informations structurelles nouvellement présentes dans les instances, mais représentent des spécifications, voire des structures conceptuelles plus génériques et plus riches. Ces spécifications permettront de faire émerger des structures conceptuelles qui étaient plus ou moins noyées et qui s'avèrent plus prometteuses par rapport aux changements survenus dans l'environnement.

BIBLIOGRAPHIE

REFERENCES BIBLIOGRAPHIQUES RELATIVES A CE TRAVAIL DE RECHERCHE

Chapitre dans un ouvrage

[Oussalah& 99] Oussalah C., Tamzalit D., Chapitre 4 « *l'émergence comme processus d'évolution d'objets* » dans l'ouvrage « Génie Objet, Analyse et Conception de l'évolution », Éditions Hermes, septembre 1999.

Revue Internationale

[Tamzalit& 00] Tamzalit D., Oussalah C. « Allowing Conceptual Emergence through Instance Evolutionary Processes » D. Tamzalit & C. Oussalah - International Journal of Engineering Intelligent Systems. Pp 177-192. Vol 8 N° 3, Septembre 2000. ISSN 0969 1170.

Publications à des Conférences Internationales avec comité de lecture

[Oussalah& 00] Oussalah C., Tamzalit D., « Emergence Measurement and Analyzes of Conceptual Abstractions during Evolution Simulation in OOD » *ISMIS'2000 (twelfth international symposium on Methodologies for intelligent systems),* Charlotte, USA, 11-14 Octobre 2000.

[Tamzalit& 00b] Tamzalit D., Oussalah C. « How to Manage Evolution in OODB? », 2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems 5 – 8, Prague, République Tchèque, Septembre 2000.

[Tamzalit& 99b] Tamzalit D., Oussalah C. "From Object Evolution to Object Emergence", 8th ACM International Conference on Information and Knowledge Management, p. 514-521, Kansas-City, USA. November 1999.

[Oussalah& 99a] Oussalah C., Tamzalit D.« Emergence as Object Evolution Process » *4th International Symposium on Programming and Systems (ISPS'99)* Alger, 18-20 Octobre 1999.

[Tamzalit& 99a] Tamzalit D., Oussalah C., « Instances Evolution Vs Classes Evolution » *DEXA'99*, Florence (Italie), 30 août – 3 septembre 1999.

[Tamzalit& 98b] Tamzalit D., Oussalah C., Magnan M. "How to Introduce Emergence in Object Evolution", Proceedings of International Conference on Object-Oriented Information Systems, Pp. 293-310, OOIS98, Paris, 1998.

[Oussalah& 98] Oussalah C., Tamzalit D., Magnan M., "Genetic Evolution as Object Evolution", *ICSIS'98*, Amman, Jordanie, Juin 1998.

[Tamzalit& 98a] Tamzalit D., Oussalah C., Magnan M., "GENOME : A Genetic EvolutioN Object ModEl", SEKE'98, San Francisco, U.S.A, Juin 1998.

Publications à un Workshop International avec comité de lecture

[Tamzalit& 00a] Tamzalit D., Oussalah C. « A Model for Object Structures Emergence », Workshop Objects and classification : a natural convergence, ECOOP 2000 Sophia Antipolis, France, 12 juin 2000.

Publication à une Conférence Nationale avec comité de lecture

[Rieu 97] Rieu D., Bounaas F., Morat P., Tamzalit D. « La métacircularité au service de la métamodélisation », Congrès INFORSID, Toulouse, France, juin 1997.

BIBLIOGRAPHIE

[Ahmed-Nacer 94] Ahmed-Nacer M. “Un modèle de gestion et d’évolution de schéma pour les bases de données de génie logiciel” Thèse de doctorat. Institut National Polytechnique de Grenoble, Grenoble, France, Juillet 1994.

[Astudillo 97] Astudillo H. "Maximizing Object Reuse with a Biological Metaphor", Wiley-Interscience Publication, John Wiley & Sons, vol. 3 n°4, 1997.

[Banerjee& 87] Banerjee J., Kim W., Kim H., Korth H.F. “Semantics Implementation of Schema Evolution in Object-Oriented Databases” ACM 1987.

[Bancilhon& 88] Bancilhon F., Barbedette G., Benzaken V., Delobel C., Gamerman S., Lécluse C., Pfeffer P., Richard P., Velez F. “The Design and Implementation of O2 an Object-Oriented Database System” Rapport Technique Altaïr 20-88. 13 avril 1988.

[Basili& 84] Basili, V.R., Weiss, D.,“A methodology for collecting valid software engineering data”, IEEE Transactions on Software Engineering. Pp 728-738. November 1984.

[Benatallah& 97] Benatallah B., Fauvet M.C. “Le point sur l’évolution du schéma d’une base de données” in L’Objet : Logiciel, Base de Données, Réseaux. Vol 3. N°3- Septembre 97. pp 277-297. Edition Hermes.

[Bertino 92] Bertino E. “A View Mechanism for Object-Oriented DataBases” 3rd International Conference on Extending Databases Technology, mars 23-24, Vienna (Austria), 1992

[Borgida 88] Borgida A. “Modelling Class Hierarchies With Contradictions”, SIGMOD Record (special issue on SIGMOD’88), vol. 17, No 3. Pp. 434-443, ACM, September 1988.

[Bounaas 95] Bounaas F. “Gestion de l’évolution dans les bases de connaissances”, Thèse de doctorat Informatique INPG, Grenoble, France, Octobre 1995

[Bratsberg 93] Bratsberg E. “Evolution and Integration of Classes in Object-Oriented Databases” PhD thesis, Norwegian Institute of Technology, jun. 1993.

[Briand 97] Briand L. C., Devanbu P., Melo W. L., “An Investigation into Coupling Measure fot C^{++}”, in proceedings of ICSE’97, pp. 412-421, 1997.

[Caminada 92] Caminada A., « Présage, un environnement de modélisation multi-vue/multi-niveau pour la construction et l’utilisation d’applications de planification des réseaux de télécommunications », Thèse de doctorat, Université de Montpellier II, Novembre 1992.

[Capponi 94] Capponi C., « Interactive class classification using types », dans Edwin Diday, Yves Lechevallier, Martin Schader, Patrice Bertrand, Bernard Burtschy (éds.), New approaches in classification and data analysis, Springer-Verlag, Berlin (DE), pp204-211, 1994

[Casais 91] Casais E. “Managing Evolution in Object Oriented Environnements : An Algorithmic Approach” Thèse - Université de Genève. 1991.

[Cauvet& 99] Cauvet C., Semmak F. dans [Oussalah& 99], chapitre 1, Pp 25-26, 1999.

[Chidamber& 91] Chidamber S.R., Kemerer C.F. “Towards a Metric Suite for Object-Oriented Design” Conference proceedings OOPSLA 1991.

[Chidamber& 94] Chidamber S.R., Kemerer C.F, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, vol. 20, 6 – 1994, p. 476-493.

[Clamen 94] Clamen S.M. “Schema Evolution and Integration ” in Proceedings of the Distibuted and Parallel Databases conference, volume 2, p 101-126. Kluwer Academic Publishers, Boston, 1994.

[Coad 90] Coad P. "Object-Oriented Patterns", *CACM*, vol. 35, n°9, 1990.

[Coulange 98] Coulange B. « Métrologie et Objets » L’Objet : Logiciel, Base de Données, Réseaux. Vol 4. N°4- Décembre 1998. Edition Hermes.

[Dekker 94] Dekker L. “FROME : Représentation multiple et classification d’objets avec points de vue” Thèse de doctorat informatique, USTL, Lille, France, Juin 1994.

[Demeyer& 99] Demeyer S., Ducasse S. “Metrics, Do They Really Help?” LMO 1999, Pp 69-82, VilleFranche-sur-Mer, France.

[Dony& 97] Dony C., Huchard M., Libourel T., “Automatic hierarchie reorganization: an algorithm and case studies with overloading”. Rapport technique 97279, Laboratoire d’Informatique, de Robotique et de Microéléctronique de Montpellier, 1997.

[Goldberg 94] Goldberg D.E. “Algorithmes Génétiques ” Intelligence Artificielle - Edition Addison-Wesley. 1994.

[Guetari 95] R. Guetari “An Object Model for the Real-Time Systems Design” in Actes 1995 IEEE International Conference on Systems, Man and Cybernetics. Vancouver, British Colombia, Canada, October 22-25, 1995.

[Harani 97] Harani Y. « Une Approche Multi-Modèles pour la Capitalisation des Connaissances dans le Domaine de la Conception », Thèse de Doctorat, Université de l’IPG, Grenoble, France, Novembre 1997.

[Harel 87] D. Harel – Statecharts : a visual formalism for complex systems, Science of Computer Programming pp. 231-274 – 1987.

[Harel 96] D. Harel, E. Gery - Executable object modeling with statecharts, Proceedings 18 th Int. Conf. Soft. Eng. IEEE Press 1996 pp.246-257.

[Haton& 91] Haton J.P., Bouzid N., Charpillet F., Haton M.C., Lâasri B., Lâasri H., Maquis P., Mondot T., Napoli A. « Le raisonnement en Intelligence Artificielle : modèles, techniques et architectures pour les systèmes à base de connaissances », InterEditions, 1991.

[Heudin 94] Heudin J.C. “La Vie Artificielle ” Edition Hermes, 1994.

[Holland 75] Holland J. “Adaptation in Natural and Artificial Systems”, University of Michigan Press, Ann Arbor, Mich., 1975.

[Jacobson& 91] Jacobson I., Lindtröm F. “Re-engineering of Old Systems to an Object-Oriented Architecture” Conference proceedings OOPSLA 1991.

[Jiang& 96] Jiang H., Castellani X. “Migration of Characteristics of Classes, a New Concept to Specify an Object System Evolution” 7th International Conference of the Information Resources Management Association (IRMA), May 1996, Washington, USA.

[Katz& 87] Katz R.H., Chang E. “Managing change in a computer-aided design database” In Proceedings of the 13th conference on VLDB, pp 455-462, Brighton, England, September 1987.

[Keen 89] Keen S.E., « Object-Oriented Programming in Common Lisp : A Programmer’s Guide to CLOS », Addison-Wesley, Reading, MA, 1989.

[Kim& 88] Kim W., Chou H.T. “Versions of schema for object oriented databases” In Proceedings of the 14th VLDB Conference, Los Angeles, Californie, 1988.

[Kim& 90] Kim W. "Introduction to Object-Oriented Databases", MIT Press, Cambridge Massachussetts, 1990.

[Langton& 90] Langton C., Taylor C., Farmer J.D., Rasmussen S. "Artificial Life II", Proceedings volume in the Santa Fe Institute studies in the sciences of complexity, 1990.

[Lerner& 90] Lerner B.S., Habermann A.N. “Beyond Schema Evolution to Database Reorganization” in Proc. ACM Conf. OOPSLA and Proc. ECOOP, Ottawa, Canada. Published as ACM SIGPLAN Notices 25(10), pp. 67-76. October 1990.

[Li& 94] Li Q., McLeod D. “Conceptual Database Evolution Through Learning in Object Databases”, IEEE Transactions on Knowledge and Data Engineering’ volume 6 – n°2 – Pp: 205-224. avril 1994.

[Li& 98] Li Wei, Talburt J. “Empirically Analyzing Object-Oriented Software Evolution”, JOOP, September 1998, Pp. 15-19.

[Masini 89] Masini G., Napoli A., Colnet D., & all « Les langages à Objets », Interéditions, 1989.

[Meslati& 97] Meslati D., Ghoul S. "Semantic Classification : A genetic approach to classification in object-oriented models", JOOP, 1997.

[Meyer 88] Meyer B. “Object-Oriented Software Construction” International Series in Computer Science. Prenctice Hall, 1988.

[Muller 97] Muller P.A. Modélisation objet avec UML, édition Eyrolles, 1997.

[Napoli 92] Napoli A. “Représentation à objets et raisonnement par classification en I.A.” Thèse de doctorat, Nancy 1992.

[Nguyen& 89] Nguyen G.T., Rieu D. "Schema Evolution in Object-Oriented Database Systems", Data and Knowledge Engineering, Vol. 4, 1989.

[Nguyen& 97] Nguyen G.T. dans [Oussalah& 97], Chapitre 6, Pp 205-232, 1997.

[Orel& 95] Orel T., Trousse B. "A view on the Philosophical Meanings of the Concept of Design", Journal of Design Sciences and Technology, vol. 4, 1995.

[Oussalah 88] Oussalah C., « Modèles hiérarchisés multi-vues pour le support de raisonnement dans les domaines techniques », Thèse de doctorat, université de droit, d’économie et de sciences d’Aix-Marseille, Juin 1988.

[Oussalah& 97] Oussalah C. et alii « Ingénierie Objets – Concepts et Techniques », InterEditions, 1997.

[Oussalah& 99] Oussalah C. & alii « Génie Objet : analyse et conception de l’évolution », Editions Hermes, 1999.

[Penney& 87] Penney D.J., Stein J. “Class modification in the GemStone object-oriented DBMS” SIGPLAN Notices (Proc OOPSLA’87) Vol. 22, No. 12, pp. 111-117. 1987.

[Pernici 91] Pernici B. “Objects with Roles” IEEE Conference on Office Information Systems, vol 16, n°3, pp 417-438, september 1991.

[Peyrade 87] Peyrade M., « Computer-aided Telecommunications Network Planning », communication & Transmission, n°2, 1987.

[Price 97] Price M.W., Dumerjian S.A. “Analyzing and Measuring Reusability in Object-Oriented Designs” Conference proceedings OOPSLA 1997.

[Puig 97] Puig V., « CHIC: Un Modèle de Contraintes dans les Hiérarchies de Composition d’Objets, » Thèse de Doctorat, Université de Montpellier II, Montpellier, France, 1997.

[Rashid& 99] Rashid A., Sawyer P. “Dynamic Relationships in Object Oriented Databases: A Uniform Approach” In proceedings of DEXA’99, Pp 26-35; Florence, August/September 1999.

[Rational 97] Rational Corporation. UML Notation Guide 1.1. Septembre 1997.

[Rechenman 89] Rechenman F., Uvietta P. “SHIRKA : an object-centered knowldege bases management system. ” In A. Pavé and G. Vansteenkiste ED. Artificial Intelligence in numerical and symbolic simulation, ALEAS, Lyon, France, pp 9-23.

[Rieu& 92] Rieu D., Nguyen G.T, Escamilla J. “SHOOD : an object model for design applications”, Proc. 7th Intl. Conf. Artificial Intelligence, Expert Systems & Natural Language, Avignon, France, juin 1992.

[Rieu 99] Rieu D. « Ingénierie des Systèmes d'Informations : Bases de Données, Bases de Connaissances et Méthodes de Conception. » Mémoire d'Habilitation à diriger des Recherches. LSR IMAG – INPG, 1999.

[Rocacher 88] Rocacher D. "Metrics definitions for Smalltalk" Projet ESPRIT 1257, MUSE WP9A, 1988.

[Sahraoui 99] Sahraoui A.H., Lounis H., Mao Y. « Impact du couplage sur la réutilisabilité des classes : une étude de cas. » LMO 1999, pp 83-98, VilleFranche-sur-Mer, France.

[Segapeli& 99] Segapeli J.L, Cavarero A. « Comment construire un schéma de classes à partir d'exemples ». LMO 1999, pp 213-228, VilleFranche-sur-Mer, France.

[Skarra& 87] Skarra A.H., Zdonik S.B. "Type Evolution in an Object-Oriented Databases" in Research Directions in Object-Oriented Programming, MIT Press Series in Computer Systems, MIT Press, Cambridge, MA, 1987, pp. 393-415. 1987.

[Stroustrup 96] Stroustrup B. « Le langage C^{++} », International Thomson Publishing France, Deuxième édition, Paris, 1996.

[Sun 98] SUN MICROSYSTEMS. Programmation Java – LJ20 – Solaris 2.x, Windows NT, Guide étudiant, Janvier 1998.

[Syswerda 89] Syswerda G. "Uniform Crossover in Genetic Algorithms" dans les actes de Intl. Conf. On Genetic Algorithms, pages 2-9, 1989.

[Talens 93] Talens G., Oussalah C., Colinas M.F. "Versions of Simple and Composite Objects" VLDB'93 : 19th International Conference on VLDB, Dublin- Ireland. Août 1993.

[Talens 94] Talnes G., « Gestion de versions d'objets simples et composites », Thèse de Doctorat, Université de Montpellier II, Montpellier, France, Février 1994.

[Urtado 98] Urtado C, « Versions d'Entités Complexes – Approches Microscopique et Macroscopique », Thèse de Doctorat, Université de Montpellier II, Montpellier, France, Octobre 1998.

[Weinberg 85] Weinberg GM "Natural Selection as Applied to Computers and Programs" Chapitre dans 'Processes of Software Changes' – Academic Press – London. Editeur : Lehman MM and Belady LA. Côte : R3127, 1985.

[Zdonik 90] Zdonik S.B. "Object-Oriented Type Evolution" in Advances in Database Programming Languages, François Bancilhon and Peter Buneman (eds.), ACM Press, New York, NY, pp. 277-288, 1990.

[Zicari 91] Zicari R., "A framework for schema updates in an object oriented database system", in proc. Of the 7th Conf. On Data Engineering, Japan, 1991.

Résumé : nous proposons le modèle d'évolution GENOME basé sur l'évolution dynamique de structure d'objets. Dans le cadre d'applications d'ingénieries complexes et volumineuses, nous voulons permettre à un concepteur de faire face à des besoins d'évolution imprévus ou mal spécifiés. Comme nous considérons l'évolution comme un processus interactif et itératif d'adaptation entre classes et objets, l'objectif est de rechercher et de dégager diverses voies d'évolution des structures conceptuelles à partir de l'évolution dynamique de la structure d'objets. Pour cela, GENOME permet au concepteur d'exprimer directement les nouveaux besoins sur les objets concernés. Ensuite, il recherche diverses voies possibles d'évolution pour permettre l'adaptation aux besoins. La recherche passe par la simulation de plusieurs voies d'évolution. L'adaptation passe par l'émergence de nouvelles structures conceptuelles. Celles-ci sont mieux adaptées et plus complètes, et émergent en tirant profit de l'expression des besoins ponctuels et des informations présentes dans la base.

Cette simulation d'évolution se fait au travers de *processus d'évolution*. Tout processus peut aboutir à deux situations : les *processus de développement* qui représentent toute extension de classes existantes par création, modification ou migration d'instances ; les *processus d'émergence* qui représentent toute émergence de nouvelles propriétés et/ou structures conceptuelles. Ces derniers peuvent être de *l'émergence locale* de nouvelles propriétés au sein de classes existantes, ou de *l'émergence globale* d'une nouvelle structure conceptuelle à partir de l'objet ayant évolué, ou en croisant plusieurs structures conceptuelles grâce au mécanisme de *croisement* proposé. Pour contrôler l'émergence, nous définissons des métriques objets de contextes *intra-structure* et *inter-structures*. Leur utilisation permet de trouver la structure émergente la plus cohérente, la plus significative et qui répond au mieux aux *invariants* du concepteur.

Mots-clés : Évolution dynamique d'objets - Émergence - Développement - Génotype - Phénotype - Génotype Fondamental - Génotype Hérité - Génotype Spécifique - Sélection - Croisement - Adaptation - Métriques.

Abstract: we propose the GENOME evolution model based on the dynamic evolution of objects structure. Within complex and bulky engineering applications, we want to allow a designer to face unpredicted and badly specified evolution needs. As we consider object evolution as an iterative and adaptive process between classes and objects, our objective is to search and extract several ways of evolution of conceptual structures from the dynamic evolution of objects in their structure. For that, GENOME allows the designer to express directly his new needs on the concerned objects. Then, it explores to find several ways of evolution, in order to achieve the adaptation to these new needs. The exploration is done through the simulation of these evolution ways. The adaptation is done through the emergence of new conceptual structures. These lasts are well-adapted and more complete, and they emerge from the expression of the needs and existing information within the database.

This simulation of evolution is done within *Evolutionary Processes*. Each process can lead to two situations: *Development processes* which represent classes extension by creating or modifying objects, or by migration of objects; *Emergence processes* which represent emergence of new properties or/and new conceptual structures. These processes can lead to *local emergence* of new properties within existing classes, or to *global emergence* of new conceptual structures from the evolved objects, or by crossing existing conceptual structures thanks to the proposed *crossing-over* mechanism. In order to control the emergence, we define some object metrics of *intra-structure* context and *inter-structures* context. Their use permit to find the most coherent, the most significant emergent structure and which well respond to the designer *invariants*.

Key-Words: dynamic object evolution – emergence – development – genotype – phenotype – fundamental genotype – inherited genotype – specific genotype – selection – crossing-over – adaptation – metrics.