La relation d'association dans les approches dirigées par les modèles : besoins et usages pratiques

Sylvain Vauttier, Christelle Urtado, Eric Mendizabal

▶ **To cite this version:**

Sylvain Vauttier, Christelle Urtado, Eric Mendizabal. La relation d'association dans les approches dirigées par les modèles : besoins et usages pratiques. ALCAA 2003 (Agents logiciels, coopération, apprentissage et activité humaine), Sep 2003, Bayonne, France. pp.151-163. hal-00365133

HAL Id: hal-00365133
https://hal.science/hal-00365133

Submitted on 2 Mar 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

La relation d'association dans les approches dirigées par les modèles : besoins et usages pratiques

Sylvain Vauttier, Christelle Urtado, Eric Mendizabal

LGI2P / Ecole des Mines d'Alès,
Site EERIE - Parc Scientifique G. Besse - F30035 Nîmes - France
{Sylvain.Vauttier, Christelle.Urtado, Eric.Mendizabal}@site-eerie.ema.fr

RÉSUMÉ. Après avoir essentiellement porté sur la conception d'infrastructures d'intégration de composants logiciels, la pratique du développement à base de composants s'oriente vers des approches dirigées par les modèles. Ces approches consistent à changer le niveau d'abstraction de la description des composants : les composants logiciels, décrits par leur codage dans un langage de programmation donné et pour une infrastructure donnée, sont remplacés par des composants conceptuels, décrits par des modèles technologiquement neutres dans un langage de modélisation standard. Le développement à base de composants peut alors se définir comme la réutilisation et l'intégration des modèles décrivant les composants. La relation d'association est le concept clé de la mise en œuvre de l'assemblage de tels composants conceptuels.

ABSTRACT. Component-based software engineering practices head towards model-driven approaches, after having been for long essentially focused on the design of middleware frameworks to manage the integration of software components. These new approaches consist in using more abstract component descriptions : software components - described by code expressed in a given programming language and for a given middleware framework - are replaced by conceptual components - described by platform-independent models expressed in a standard modeling language. Component-based software development can therefore be defined as the reuse and the integration of the models that describe components. The association relationship is the key concept to assemble such conceptual components.

MOTS-CLÉS : approches dirigées par les modèles, composants conceptuels, réutilisation de composants, assemblage de composants, relations d'association, méta-modélisation, UML, MDA.

KEYWORDS : model-driven approaches, conceptual components, component reuse, component assembling, association relationships, metamodeling, UML, MDA.

1. Introduction

L'ingénierie à base de composants des logiciels et des systèmes d'information est à l'aube d'une mutation. Après avoir essentiellement consisté à intégrer, au niveau implémentation, des composants logiciels interopérables, à l'aide d'infrastructures telles qu'OMA (CORBA) [OMG97, OMG01], J2EE [Sun01] et plus récemment les web services [W3C03], l'ingénierie à base de composants s'enrichit de démarches dirigées par les modèles [Béz01, Dso01]. Ces démarches sont fondées sur un constat simple : les composants logiciels ne tiennent pas leurs promesses en termes de réutilisation. Les infrastructures qui permettent l'intégration des composants logiciels ne sont pas matures : elles subissent des évolutions rapides qui obligent à faire migrer régulièrement le code des composants vers ces infrastructures. Seuls les modèles conceptuels de ces composants, produits lors des étapes de spécification ou de conception, constituent des productions pérennes et capitalisables. Ce sont donc ces modèles conceptuels qui doivent constituer les véritables éléments réutilisables d'un logiciel.

Ainsi, les approches dirigées par les modèles proposent-elles de réaliser l'intégration des logiciels non plus au niveau implémentation, grâce à des composants logiciels interopérables, mais au niveau conceptuel, par l'intégration de composants conceptuels, décrits dans un langage de modélisation standard et technologiquement neutre tel qu'UML. Dans ce type d'approche, les composants logiciels implémentant ces composants conceptuels sont obtenus à l'aide de générateurs de code spécifiques à l'infrastructure cible, ce qui permet de faire migrer à moindre coût les composants.

Dans ce contexte, nous avons étudié la création de composants conceptuels pour et par la réutilisation [Cau99]. La relation d'association est l'un des concepts clés de la mise en œuvre pratique de cette réutilisation : elle permet lier des classes provenant de différentes modèles, en exprimant les rôles relatifs qu'elles jouent entre-elles, et de créer ainsi de nouveaux modèles, par assemblage de modèles existants. Dans cet article, nous examinons certaines propriétés des relations d'association [Civ98, Gra97, Hen98] sous l'angle de leur impact sur les mécanismes de définition puis d'utilisation de composants conceptuels. Nous en déduisons les caractéristiques nécessaires à un modèle de relations d'association pour faciliter le découplage des composants (ce qui permet de produire des composants réutilisables) puis leur assemblage (ce qui permet de réutiliser effectivement des composants). UML [OMG01a], en tant que standard pour les langages de modélisation, et MDA, en tant que futur standard de démarche dirigée par les modèles proposé par l'OMG [Béz01,Dso01], servent de cadre pratique pour illustrer nos travaux. Nous présentons en conclusion des perspectives d'étude et d'application de tels modèles de relations d'association.

2. Problématique de la réutilisation de composants conceptuels

Nos travaux portent sur la réutilisation de composants conceptuels et s'inscrivent dans la tendance actuelle des approches de développement de logiciels et de systèmes d'information dirigées par les modèles. Nous présentons brièvement ci-dessous MDA, l'approche de développement dirigée par les modèles proposée par l'OMG, autour de ses standards de modélisation.

2.1. MDA : l'approche de développement dirigée par les modèles de l'OMG

L'OMG a lancé une initiative pour bâtir une approche dirigée par les modèles appelée MDA [Béz01, DSo01]. Elle s'appuie sur le standard de modélisation orientée objets à quatre niveaux de l'OMG [OMG01c] :

- le niveau méta-méta, qui définit le MOF (Meta Object Facility), un méta-méta-modèle permettant de décrire de façon standard un méta-modèle et d'étendre ou de modifier les méta-modèles existants.
- le niveau méta, où les méta-modèles sont tous exprimés comme des instances du MOF, afin de permettre leur interopérabilité. Ces méta-modèles définissent les « langages » de modélisation qui sont utilisés pour décrire les modèles. L'OMG a ainsi défini, à ce niveau, le langage UML [OMG01a] par un méta-modèle instance du MOF.
- le niveau classe, où les modèles sont exprimés comme des instances de méta-modèles définissant le langage de modélisation choisi. Par exemple, un modèle UML, correspondant à la spécification/conception d'un composant logiciel est défini à ce niveau comme une instance du méta-modèle d'UML.
- le niveau instance, où sont produites des instances particulières (celles existant dans les applications) par instanciation des modèles de niveau classe.

Au sein de MDA, les modèles entretiennent principalement deux types de relations [Béz01, Dso01] :

- des **relations de transformation**, décrivant la traduction d'un modèle, instance d'un méta-modèle, en un modèle équivalent, instance d'un autre méta-modèle. L'expression des « règles de transformation » entre modèles représente un champ de recherche important. Ces règles permettront, par exemple, de décrire la génération automatique du code d'implantation d'un composant conceptuel pour une plate-forme donnée (règles de transformation entre le modèle de composants d'un langage de modélisation et le modèle de composants spécifique à une plate-forme d'implémentation).
- des **relations d'intégration**, décrivant comment les modèles de différents composants conceptuels, instances d'un même méta-modèle, sont assemblés en un modèle de granularité supérieure pour définir un nouveau composant. Nos travaux portent sur ce deuxième type de relations, directement liées aux mécanismes de réutilisation par assemblage de composants. Concrètement, ces relations d'intégration se traduisent, dans un premier temps, par des relations d'« import » entre modèles, permettant à un modèle de réutiliser les éléments d'un autre modèle. Puis les éléments provenant des différents modèles réutilisés sont

assemblés, grâce aux différents mécanismes permettant de lier « physiquement » deux graphes d'objets : l'identification, la fusion, et l'association de classes d'objets [Cla00]. L'identification consiste à définir comme identiques (à des renommages près) deux classes d'objets. Les graphes correspondant aux deux modèles sont alors intégrés par le partage d'une classe d'objets commune. La fusion consiste à créer une nouvelle classe par spécialisation de deux classes appartenant à deux modèles à intégrer : cette sous-classe hérite des propriétés des deux classes et les fusionne. Les graphes correspondant aux deux modèles sont alors intégrés grâce à la création d'une classe commune. L'association consiste à définir les rôles que jouent les classes les unes par rapport aux autres au sein des modèles à l'aide de relations d'association [Wir90, Hol92, Ree95, Dso98]. L'intégration de deux modèles peut être obtenue en définissant une association entre deux classes d'objets, créant ainsi un arc reliant les graphes des deux modèles.

Dans cet article, nous nous intéressons à l'assemblage de composants conceptuels par association de classes d'objets et à son support : la relation d'association. En effet, les relations d'association sont un concept clé de l'assemblage de composants car elles permettent, en définissant le rôle joué par les différentes classes d'un modèle, les unes par rapport aux autres, de spécifier les collaborations qui assurent le fonctionnement global de l'assemblage [Vau99a].

2.2. La relation d'association

La sémantique et la modélisation des relations d'association a été, et continue à être, au centre de nombreux travaux [Civ98, Gra97, Hen98, Ste01]. Il est cependant possible d'en retenir une sémantique consensuelle en se retreignant aux propriétés influant sur la réutilisation de composants conceptuels (cf. Section 2.3) : la dépendance et la réciprocité.

De manière générale [OMG01a], une **relation d'association** est une relation n-aire liant un ensemble de classes d'objets et définissant les rôles relatifs joués par les instances de ces classes, lorsqu'elles sont mises en relation par des instances de cette relation d'association. Une instance d'une relation d'association est appelée un **lien d'association**. Un lien d'association lie n objets, instances des n classes liées par la relation d'association dont il est une instance.

Dans cet article, par soucis de simplification, l'étude sera restreinte aux relations d'association binaires[1].

2.2.1. La propriété de réciprocité

Il convient, dans un premier temps, de distinguer les relations d'association unidirectionnelles des relations d'association bidirectionnelles [Fir98, Hen98], c'est-à-dire la propriété qu'ont les relations d'exprimer, ou non, des rôles réciproques.

Une **relation d'association unidirectionnelle,** est une relation d'association qui ne possède qu'un seul sens de lecture : elle définit le rôle de la classe cible vis à vis de la classe source de la relation, mais ne définit par de rôle réciproque à la classe source vis à vis de la classe cible. Un lien unidirectionnel, associant un objet source *O1* à un objet cible *O2* définit le rôle que joue *O2* pour *O1*, et uniquement ce rôle.

Par opposition, une **relation d'association bidirectionnelle** définit les rôles réciproques des deux classes qu'elle lie et possède ainsi deux sens de lecture. Lorsqu'un objet *O1* est associé à un objet *O2* par un lien bidirectionnel, ce lien définit le rôle joué par *O2* vis-à-vis de *O1*, dans cette association, et réciproquement le rôle joué par *O1* vis-à-vis de *O2*.

Le point de vue des méthodologistes diverge sur les mérites comparés de ces deux types de relations d'association. Certains, par économie de concepts, proposent de n'utiliser que des relations unidirectionnelles.

[1] Les relations binaires sont les plus couramment utilisées. Peu de démarches encouragent l'utilisation de relations d'association d'arité supérieure, s'attachant avant tout à définir la relation qu'entretient, en particulier, chaque couple de classes d'un modèle.

L'expression de relations réciproques est alors assurée par un couple de relations d'associations unidirectionnelles assorties de contraintes pour gérer leur réciprocité [Gra97, Hen98]. On peut reprocher à ce type de modèle de relation d'association, d'une part, d'alourdir la spécification des relations d'associations réciproques et, d'autre part, de poser des problèmes de respect des principes objet dans la gestion de la représentation des relations d'association (à quelles entités doit on associer les règles ou les contraintes gérant la réciprocité des deux relations unidirectionnelles pour respecter le principe d'encapsulation ?).

Inversement, d'autres ne proposent que des relations d'association bidirectionnelles, ce qui impose de définir systématiquement un rôle réciproque à toute définition d'un rôle. C'est le cas du modèle de relations d'association d'UML [OMG01a], qui associe systématiquement à chaque extrémité d'une relation d'association la définition d'un rôle. En théorie, il est impossible d'exprimer, à l'aide d'un tel modèle, des relations d'association non-réciproques. Dans la pratique, cette obstacle est contourné en utilisant des rôles « factices », vides de sémantique, qui ne servent pas à exprimer des collaboration entre classes, mais uniquement à respecter la contrainte d'expression de relations uniquement bidirectionnelles. Au prix de rôles superflus, les relations bidirectionnelles peuvent donc servir à exprimer des relations « conceptuellement » unidirectionnelles. Mais les relations véritablement bidirectionnelles sont exprimées dans ces modèles à l'aide d'un concept adapté qui peut encapsuler toute la sémantique de la relation, ce qui simplifie la gestion du métamodèle et assure le respect des principes objet.

Ne proposer que des relations bidirectionnelles est, sans doute, ainsi, la meilleure solution de compromis entre l'expressivité du modèle de relations et efficacité du métamodèle. Mais la meilleure des solutions, et plus particulièrement pour la gestion de composants conceptuels, est celle d'un métamodèle sans compromis qui propose ces deux types de relations d'associations [Civ98].

2.2.2.La propriété de dépendance

Il est à distinguer, dans un deuxième temps, les relations d'associations que nous désignerons comme intrinsèques des relations que nous désignerons comme extrinsèques, selon qu'elles ont, ou non, la propriété d'exprimer un rôle dont dépend la définition ou le fonctionnement d'une classe.

- Les **relations d'association intrinsèques** sont des relations d'association qui participent directement à la description ou au fonctionnement des classes. Elles traduisent les besoins existentiels (description d'une partie d'un objet par un autre objet) ou fonctionnels (appels aux méthodes fournies par un autre objet pour réaliser ses propres méthodes) d'une classe d'objets vis à vis d'une autre classe d'objets. Le premier cas correspond à la définition d'une relation de composition[2] [Win78, Boc98, OMG01a, Bar03], qui est un cas particulier de relation d'association. Le second cas correspond à la définition d'une « simple » relation d'association, dont la sémantique n'exprime que la nécessaire collaboration entre deux classes.

Les relations intrinsèques sont définies en même temps que les classes à la définition desquelles elles participent. Ces relations d'associations intrinsèques sont assimilables, d'un point de vue fonctionnel, à des attributs et ont d'ailleurs parfois été désignées comme telles [Gra95]. Les liens intrinsèques qui associent un objet à d'autres objets sont connus de cet objet et sont utilisés comme autant de « références » dans le codage de ses méthodes. A ce titre, les liens intrinsèques sont généralement implémentés à l'aide d'attributs [Ste01], à défaut de disposer de modèles de relations réifiés [Urt98b, Vau99b]. Ce type d'implantation permet de traduire la dépendance fonctionnelle ou existentielle exprimées dans la relation d'association mais ne permet pas de traduire toute la variété de sa sémantique opérationnelle. En effet, s'il est possible, conceptuellement, de représenter tous les attributs d'une classe d'objets à l'aide de relations d'associations intrinsèques, la réciproque est fausse : toutes les relations d'association intrinsèques ne représentent pas nécessairement un attribut d'une classe. Les attributs d'un objet servent à décrire l'objet lui-même. Les relations d'association servent plus généralement à traduire les collaborations qu'entretiennent les objets entre eux [Wir90, Dso98, Ste01]. Ainsi, les attributs peuvent être représentés par une sous-classe de relation d'association intrinsèque, et pour être plus précis comme une relation de composition exclusive (un composant n'appartient qu'à un seul objet composite) et dépendante (la destruction de l'objet composite entraîne la destruction de ses composants) [Vau99a]. En résumé, dans tous les cas, les

[2] Dans cet article, nous utilisons le terme relation de composition pour faire référence à toutes les formes de composition (ce qui englobe, sans distinction, les relations d'agrégation et de composition au sens d'UML).

relations d'association intrinsèques participent à la définition de la structure ou du comportement d'une classe d'objets : elles sont, à ce titre, définies a priori[3] et ne peuvent être modifiées ou supprimées indépendamment de la définition de la classe d'objets qui en est dépendante. Des exemples concrets de ces relations d'association intrinsèques sont les relations d'association navigables d'UML [OMG01a].

- Les **relations d'associations extrinsèques** sont, au contraire, des relations d'association qui ne participent pas à l'expression des besoins fonctionnels ou existentiels des classes d'objets qu'elles associent. En d'autres termes, ces relations ne participent pas à la description des classes d'objets, mais uniquement à la description du contexte qui les environne dans le modèle. Les relations extrinsèques ne sont pas connues ou utilisées par les classes d'objets qu'elles lient : il est possible d'ajouter ou de retirer de telles relations d'association indépendamment de la définition des classes d'objets. Un exemple concret de telles relations d'association sont les relations d'associations non-navigables d'UML [OMG01a]. Ce type de relations d'association permet, par exemple, de structurer l'ensemble des objets d'un modèle, indépendamment des structures établies par les objets eux-mêmes, pour faciliter la navigation ou optimiser certains calculs « ensemblistes » au sein du modèle (utilisation du modèle comme structure de données), ou d'exprimer de la connaissance (la sémantique des rôles des associations) qui n'intervient pas dans la définition des classes. Ce type de relation permet également de définir, a posteriori[3], des collaborations entre classes d'objets, lorsqu'elles sont utilisées dans un nouveau contexte, sans impliquer de modification des classes (afin de préserver leur potentiel de réutilisabilité et de ne pas compromettre leurs utilisations courantes). Les attributs et méthodes nécessaires à l'implémentation de ces collaborations sont définis au niveau de la relation d'association extrinsèque, qui devient alors un concept de première classe que nous appelons une **classe d'objets relationnels.** Un objet relationnel permet ainsi de représenter non seulement le lien structurel existant entre deux objets et mais également de gérer leur collaboration dans cette relation [DSo98]. UML propose concept comparable, appelé classe-association [OMG01b], sans lui donner une définition et un usage aussi précis. Bien que décrié par certains, qui le jugent inutile [Gra97, Hen98], le concept d'objet relationnel nous paraît essentiel dans ce contexte de l'assemblage de composants conceptuels. Il permet en effet de définir des relations d'association indépendamment des classes associées, tout en gérant leurs collaborations.

Nous utilisons les notations UML suivantes pour exprimer les relations d'association qui définissent les composants conceptuels (*cf.* Figure 1.) :

- Les rôles sont représentés par des étiquettes textuelles. Cette étiquette définit le nom du rôle et spécifie le type d'objets qui peut remplir le rôle. Sur la figure 1, par exemple, l'étiquette *R : I* signifie que le rôle *R* est destiné à être rempli par des objets de type *I*. *I* peut être une interface ou une classe d'objets. La classe d'objets qui remplit effectivement le rôle *R* dans un modèle (qui est liée à cette extrémité de la relation d'association) doit être « conforme » au type *I* (donc concrètement réaliser *I* ou être une sous-classe de *I*, suivant le cas). Dans le cadre de cet article, les autres éléments associés à la définition d'un rôle (par exemple sa multiplicité) ne sont pas représentés et discutés (par soucis de simplification, car ces éléments n'interviennent pas dans les mécanismes étudiés).

- l'absence de rôle, à l'une des extrémités d'une relation d'association, est signalée par la présence d'un cercle contenant une croix. Cette notation est un ajout au standard d'UML pour pallier l'absence de relations d'association unidirectionnelles [4],

- la dépendance, exprimée par la navigabilité d'un rôle, est représentée par une flèche ouverte à l'extrémité de l'association.

- Les classes d'objet relationnels sont représentées sous la forme d'une classe attachée par un ligne pointillée à la ligne matérialisant la relation d'association.

---

[3] A priori et a posteriori font référence ici au processus de réutilisation. Une relation définie a priori doit l'être au même moment que les classes aux définitions desquelles elle participe. Elle ne peut en aucun cas y être ajoutée a posteriori, c'est-à-dire après l'achèvement de la conception des classes (par exemple, pour être adaptées à un autre contexte au cours d'un processus de réutilisation) sauf modifications radicales des classes mêmes (notamment de leurs implémentations).

[4] Nous travaillons conjointement à l'adaptation du métamodèle d'UML mais le parti pris de cet article est de ne présenter que les aspects conceptuels de l'utilisation des relations d'association dans la gestion des composants conceptuels.

2.3. Conception pour et par la réutilisation de composants conceptuels

Le développement à base de composants repose sur deux mécanismes : la production de composants *pour la réutilisation* et la production de composants *par réutilisation* [Cau99, Fro99].

2.3.1. Notions de contexte de définition et de contexte d'utilisation d'un composant conceptuel

La qualité principale attendue d'un composant est d'être réutilisable. Cette qualité est directement liée à sa généricité : la définition d'un composant doit être exempte de tout élément contextuel, lié à une utilisation

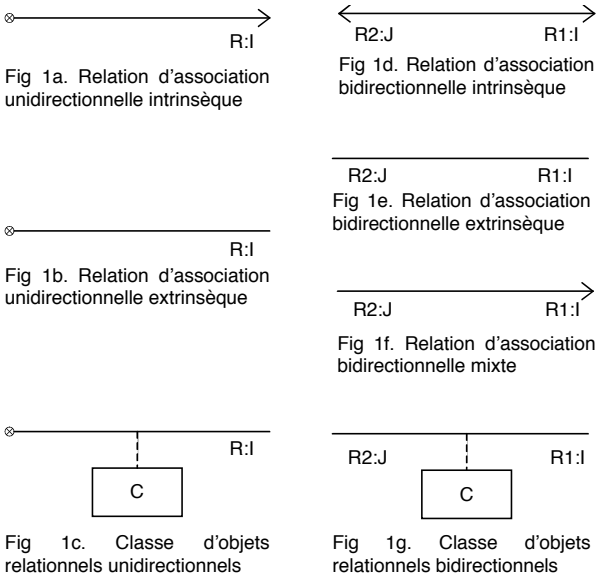

Fig 1a. Relation d'association unidirectionnelle intrinsèque

Fig 1d. Relation d'association bidirectionnelle intrinsèque

Fig 1e. Relation d'association bidirectionnelle extrinsèque

Fig 1b. Relation d'association unidirectionnelle extrinsèque

Fig 1f. Relation d'association bidirectionnelle mixte

Fig 1c. Classe d'objets relationnels unidirectionnels

Fig 1g. Classe d'objets relationnels bidirectionnels

Figure 1. Notation des différents types de relations d'association

particulière du composant, qui restreint sa capacité à être réutilisé pour d'autres utilisations. Le respect du typage dans les systèmes à objets (relation de spécialisation / généralisation existant entre une classe et ses sous-classes) fait qu'il est possible d'ajouter ou de spécialiser un élément dans la définition d'un composant mais non d'en supprimer. Ainsi, lorsqu'un composant est sélectionné pour être réutilisé, il est possible de l'adapter à un usage particulier en le spécialisant ou en l'encapsulant dans un adapteur. Si certains éléments du composant sélectionné sont inutiles, au mieux ils alourdissent la définition et la représentation du nouveau composant, au pire, ils produisent des effets de bord qui rendent incohérents la définition du nouveau composant. Il faut donc éviter de « polluer » la définition d'un composant avec des éléments qui ne sont pas essentiels à sa définition [Vau99a]. L'idéal est de proposer une palette de composants comportant des composants génériques, qui pourront être adaptés à des utilisations inédites, et des composants plus spécialisés, sous-type des précédents, qui minimisent le coût de l'adaptation pour des utilisations classiques d'un type de composants (compromis entre généricité / utilisabilité / coût d'adaptation d'un composant).

Nous formalisons cette notion par le concept de **contexte de définition d'un composant**. Le contexte de définition d'un composant consiste en un modèle *minimal* ne comportant que les éléments qui sont nécessaires à la définition du composant. C'est le modèle de référence qui est, par exemple, stocké dans une bibliothèque et sur la base duquel le composant est réutilisé. Les contextes de définition des composants sont les « briques » du processus de développement par réutilisation de composants. Un composant ne possède qu'un contexte de définition. Nous parlons de « contexte » plutôt que de « modèle » de définition d'un composant pour être plus générique et moins ambigu. Malgré les inconvénients présentés ci-dessus, nous n'interdisons pas la définition conjointe de plusieurs composants au sein d'un même contexte de définition. Ce cas correspond à une décomposition non achevée de composants en composants plus élémentaires, dans un processus de production de composants réutilisables à partir de composants existants.

Par opposition, nous définissons le **contexte d'utilisation d'un composant** comme un modèle décrivant l'intégration d'un composant conceptuel à d'autres composants pour produire un composant d'une granularité plus élevée. Ce modèle réutilise les éléments du contexte de définition du composant et y ajoute les éléments spécifiques à son utilisation dans ce contexte donné. Un composant possède ainsi potentiellement une infinité de contextes d'utilisation.

Les notions de contexte de définition et d'utilisation sont relatives : si un composant C_1 est composé d'un composant de granularité plus faible C_2, le contexte de définition de C_1 constitue l'un des contextes d'utilisation de son composant C_2.

2.3.2. Conception de composants conceptuels pour la réutilisation

Pour être utile au processus de réutilisation, le contexte de définition d'un composant doit comporter :

- la définition des classes qui constituent le composant,

- les relations structurelles (relations de composition) qu'entretiennent les classes d'objets du composant entre-elles,

- les relations fonctionnelles (collaborations définies par des relations d'association) qu'entretiennent les classes d'objets du composant entre-elles.

Ces différents éléments constituent la définition « interne » du composant conceptuel (à comparer avec l'implémentation d'une classe d'objets). Mais le contexte de définition du composant doit comporter une définition « externe ». Cette définition doit comporter :

- l'ensemble des fonctionnalités qui sont proposées par le composant aux autres composants, appelé interface fournie par le composant [Col03]. Ces fonctionnalités sont spécifiées sous la forme d'interfaces et sont concrètement implémentées par les différentes classes du composant.

- l'ensemble des fonctionnalités qui doivent être fournies par d'autres composants pour permettre le fonctionnement du composant, appelé interface requise par le composant [Col03].

Ces éléments sont comparables à la définition de l'interface d'une classe d'objets, dans le sens où elle spécifie les collaborations que le composant pourra avoir avec d'autres composants. Elle en diffère cependant en spécifiant à la fois les responsabilités du composant (collaborations offertes par le composant) et ses besoins (collaborations requises par un composant). Ces concepts, introduits et développés par [Dso98, Hol92, Ree95, Wir90] dans des méthodes de conception «collaboration-based » ou « role-based » , se sont généralisés dans les modèles de composants. Cette « interface » définit les possibilités d'assemblage du composant avec d'autres composants :

- sa capacité à être associé à un autre composant pour jouer un certain rôle, grâce aux fonctionnalités qu'il implémente,

- son besoin d'être associé à d'autres composants, afin qu'ils lui apportent leur collaboration, en jouant le rôle qui leur est défini dans l'association.

Comme décrit plus haut, la réutilisabilité d'un composant dépend de la généricité de son contexte de définition. Cette généricité dépend à son tour de la capacité du langage de modélisation (de son métamodèle) à isoler la définition d'un composant de tout contexte d'utilisation. Cette isolation requiert un certain nombre de concepts :

- le concept d'*interface* (au sens de définition abstraite d'une classe d'objets). La définition d'un rôle, dans une relation d'association, est assortie de la définition du type des objets qui sont capables de remplir ce rôle [Hol92, Ree95, Wir90]. Pour assurer un bon découplage du contexte de définition d'une classe d'objets vis-à-vis des autres classes d'objets, il convient de pouvoir typer ce rôle à l'aide d'une interface plutôt qu'avec

une classe d'objets. Une interface est la spécification d'un type d'objets : elle définit le protocole, c'est-à-dire l'ensemble des fonctionnalités qu'une classe d'objets doit implémenter pour se conformer au type défini par l'interface. Une classe d'objets est une réalisation concrète, particulière d'un ou plusieurs types d'objets. Ainsi, spécifier le type d'un rôle à l'aide d'une interface laisse un libre choix quant aux classes d'objets dont les instances joueront le rôle dans un contexte d'utilisation donné. Spécifier le type d'un rôle à l'aide d'une classe d'objets contraint le type des objets aux seules instances de cette classe et de ses sous-classes. Une telle modélisation peut-être utilisée à dessein afin d'imposer le choix d'une classe d'objets, c'est-à-dire d'une réalisation particulière d'un type d'objets. Les interfaces sont les « composants » de granularité la plus faible et de réutilisabilité la plus élevée. Ce sont également les « composants » d'utilisabilité la plus faible (ce ne sont que des éléments abstraits, qui ne peuvent être utilisés tels quels). A un stade de désassemblage et de généralisation le plus élevé, l'ensemble de ces interfaces constitue un « référentiel de types » [Col03] fournissant une base commune pour exprimer les capacités des composants puis d'assurer, au minimum, leur compatibilité syntaxique, lors des assemblages. Une classe qui réalise une interface décrit une solution d'implémentation concrète. Cette solution fait potentiellement apparaître des besoins de collaboration entre la classe et d'autres classes, exprimés par des relations d'association.

- des *relations d'association unidirectionnelles*. Les relations d'association unidirectionnelles permettent d'exprimer les besoins de collaboration d'une classe d'un composant conceptuel, en spécifiant le rôle que doit jouer pour elle une classe d'un autre composant. Cette classe peut être spécifiée à l'aide d'une interface afin de laisser le choix de la classe qui remplira le rôle. Disposer de relations d'association unidirectionnelles prend ici toute son importance car celles-ci fournissent un moyen d'adopter le point du vue du composant à définir et de découpler sa définition de celle des autres composants. En effet, l'utilisation d'une relation d'association bidirectionnelle réclame, comme décrit plus tôt, la définition d'un rôle réciproque, à savoir la définition du rôle joué par l'une des classes du composant pour une classe d'un autre composant. La définition d'un tel rôle ne peut se justifier dans le contexte de définition que s'il exprime une exigence, posée par la classe du composant, sur les rôles qu'elle accepte de jouer pour des classes d'autres composants : non seulement cette classe spécifie comment elle veut utiliser une autre classe mais également comme elle doit être utilisée par cette autre classe. Une situation typique est la définition d'une relation de composition. Une relation de composition est nécessairement une relation bidirectionnelle : elle exprime non seulement, le rôle joué par la classe composante, mais également le rôle de la classe composite, pour pouvoir y spécifier la propriété d'agrégation (obligatoire pour définir la classe comme une classe d'objets composite) [OMG01a]. La figure 2a illustre différentes variantes de cette situation. La classe d'objets composites *C* y spécifie que ses instances ont besoin de composants de type *Q* pour jouer le rôle *R9*, à l'aide d'une relation de composition bidirectionnelle, qui spécifie que les instances de *C* doivent nécessairement jouer le rôle réciproque *R10* pour ces composants. Par ailleurs, la classe *C* spécifie que ses instances ont besoin de composants de type *N* pour jouer le rôle *R5*, à l'aide d'une relation de composition bidirectionnelle. Le rôle de la classe *C* dans cette relation, *R4*, est non navigable : il spécifie ainsi que les instances de *C* ne peuvent pas être utilisées par les composants qui leur seront associés dans cette relation (un rôle non navigable n'est pas fonctionnel ; il est purement informationnel). Dans ces deux cas, la classe *C* spécifie entièrement la collaboration qui doit s'établir entre ses instances et leurs composants : les composants sont entièrement subordonnés aux objets composites et ne peuvent pas définir les rôles qu'ils souhaitent leur voir jouer. Ces exigences peuvent être assouplies, pour « ouvrir » le composant. Au lieu d'« imposer » un rôle à la classe composante, la classe composite peut « suggérer » un rôle. C'est ce qu'illustre la relation de composition définissant le rôle *R7*, rôle de composants de type *O* pour les objets composites instances de *C*. Cette relation spécifie que la classe composite « peut » jouer, si la classe composante associée le requiert, le rôle réciproque *R6:J*. Aussi, ce rôle potentiel n'est pas défini comme un rôle effectif (étiquette textuelle associée à l'extrémité de la relation), mais sous la forme d'une contrainte (étiquette textuelle entourée d'accolades) portant sur la future définition du rôle. Une relation d'association, au contraire, définit une collaboration « d'égal à égal ». Pour ouvrir totalement le composant, les classes associées doivent pouvoir définir, a posteriori, de manière indépendante, le rôle réciproque qu'elles souhaitent voir jouer par les classes du composant, « en échange du rôle qu'elles jouent pour elles ». Les relations d'association unidirectionnelles issues des classes *C* et *D* spécifient qu'elles ont besoin de la collaboration d'objets de types *M* et *P* pour jouer les rôles *R3* et *R8*, mais n'ont aucune exigence quant à leurs rôles réciproques dans leur future association avec des classes répondant à leur besoin. Il sera alors possible d'assembler le composant avec d'autres, en associant les classes *C* et *D*, avec de quelconques classes implémentant les types *M* et *P*. Ainsi, en général, pour assurer sa généricité, le contexte de définition

d'un composant ne spécifie, hormis les relations d'association internes, que des relations d'associations unidirectionnelles, précisant le rôle attendu de classes externes.

Au delà de la description de nouveaux composants ex cathedra, le métamodèle du langage de modélisation doit permettre la production de composants conceptuels réutilisables par désassemblage de modèles. Ce processus réclame un mécanisme permettant de décomposer les relations bidirectionnelles en relations unidirectionnelles afin de pouvoir isoler le contexte de définition du composant du reste du modèle. Par exemple, le désassemblage du composant présenté Figure 2. a, en découpant la relation bidirectionnelle liant les classes *C* et *D* en un ensemble de deux relations unidirectionnelles, permet de produire deux nouveaux composants (*C-C* et *C-D*), qui permettent d'utiliser les classes *C* et *D* indépendamment l'une de l'autre (*cf.* Figure 2. b).

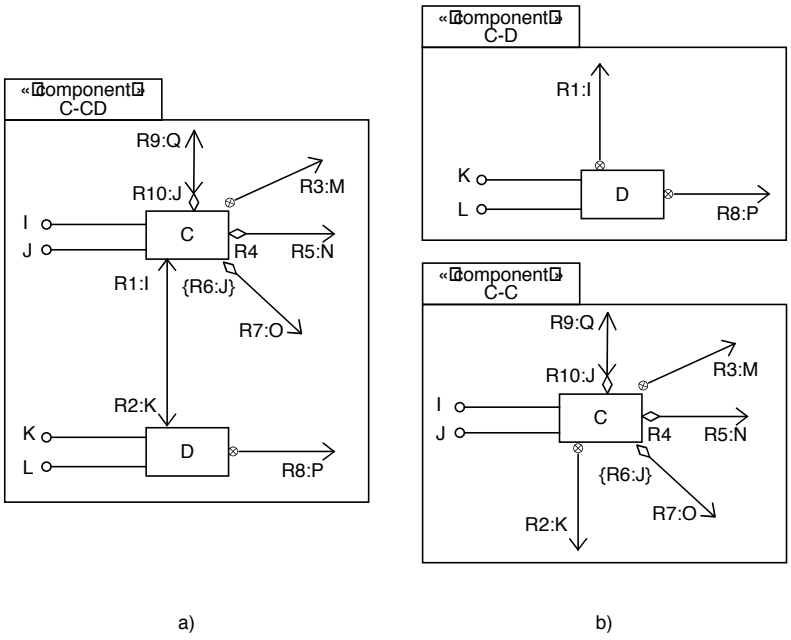

Figure 2. Exemples de contextes de définition de composants

2.3.3. Conception de composants conceptuels par la réutilisation

La production d'un composant conceptuel par la réutilisation de composants conceptuels, consiste à sélectionner un ensemble de composants conceptuels, décrits par leurs contextes de définition, à importer les éléments des modèles correspondant dans le modèle du contexte d'utilisation, à adapter les éléments au contexte (par spécialisation ou encapsulation dans un adapteur), puis à les assembler (par association, dans le contexte de ce papier). On désigne les modèles des contextes de définition comme les modèles composants du modèle du contexte d'utilisation, désigné quant à lui comme le modèle résultant de l'assemblage des modèles composants.

Approches de l'assemblage de composants

On peut distinguer deux approches dans l'assemblage des composants :

- *l'assemblage par « connecteurs »* [Dso98, Duc97]. Pour des composants conceptuels, ces connecteurs consistent en des relations d'association qui décrivent les liens existant, dans le contexte d'utilisation, entre les classes d'objets provenant des différents contextes de définition. Sous la forme de classes d'objets relationnels, ces relations d'association sont le support de la définition et du contrôle des collaborations qu'entretiennent les classes pour définir ce nouveau composant. Cette philosophie d'assemblage de composants produit des « systèmes » d'objets en interaction, par juxtaposition et connexion de classes d'objets.

- *l'assemblage par construction d'objets composites* [Vau99a]. La construction de classes d'objets composite offre l'avantage de produire une représentation concrète, abstraite et hiérarchisée, des systèmes d'objets qui résultent de l'assemblage. Dans cette approche, le composant qui est formé par assemblage d'autres composants est représenté explicitement dans le modèle sous la forme d'une classe d'objets composites. Cette classe d'objets composites décrit la structure interne du composant (relations de composition), définit et gère la coopération entre ses composants internes (définition du comportement global du composant au niveau de la classe) et définit, à l'aide de son interface, une interface abstraite du composant (fonctionnalités émergeantes ou résultantes du composant).

La construction d'une classe d'objets composites présente l'avantage de pouvoir manipuler le composant ainsi construit comme une classe d'objets. A contrario, dans l'approche par connecteurs, le système d'objets est la résultante de la présence d'éléments de connexion entre classes mais le système n'est aucunement formalisé sous la forme d'une entité explicite, directement manipulable. Cette situation provient de l'utilisation de langages orientés objets et non orientés composants. Même au niveau conceptuel, dans des langages de modélisation comme UML, le concept de composant est embryonnaire et ne concerne que la phase de déploiement. Ces langages ne proposent pas de concept véritablement adaptés à la représentation et à la gestion des composants. Les classes d'objets composites sont alors un moyen de représenter des composants comme des classes d'objets et de bénéficier alors des mécanismes classiques de l'orientation objet pour leur gestion (encapsulation, abstraction, héritage, instanciation, etc.). Cependant, il serait sans doute souhaitable de développer de véritables langages de programmation « orientés composants », intégrant directement la manipulation abstraite des composants.

La construction d'une classe composite représentant le composant est utilisée pour réduire la complexité apparente du composant, par encapsulation et abstraction : un objet composite, quelle que soit la complexité de sa structure interne et la profondeur de son graphe de composition, fournit une interface abstraite qui permet de manipuler l'ensemble des objets qui le composent comme s'il s'agissait d'un objet simple. La capacité de réduire la complexité d'un système d'objets en le représentant sous la forme d'un objet composite, puis à composer hiérarchiquement ces objets composites, permet d'envisager des assemblages complexes, à plusieurs niveaux d'imbrication. A défaut, en n'assemblant que par simple juxtaposition à l'aide de connecteurs, le concepteur est rapidement confronté à une explosion de la complexité des modèles, en nombre d'éléments.

Les deux approches diffèrent également par leur philosophie : décentralisée pour l'approche par connecteurs, centralisée pour l'approche par construction d'objets composites. L'objet composite représente un élément de contrôle central, qui définit et gère les collaborations définies a posteriori, nécessaires au fonctionnement global de l'assemblage de composants qui le constitue : c'est la notion d'objet dominant ou d'objet racine évoquée dans [Civ98, Ram96, Rum95]. Dans une approche par connecteurs, la définition et la gestion du comportement global du composant sont répartis sur un ensemble de classes d'objets (collaborations définies a priori) et des classes d'objets relationnels (collaborations a posteriori, nécessaires à l'assemblage).

Gestion de l'assemblage de composants conceptuels

Dans tous les cas, la construction d'un nouveau composant par réutilisation d'autres composants revient à définir les nouvelles relations d'associations qui lient les classes d'objets (issues des composants réutilisés ou crées pour les besoins de définition du nouveau composant) participant au contexte de définition du nouveau composant. Ces relations d'association ont différentes provenances :

- elles proviennent des contextes de définition des composants réutilisés,

- elles sont créées, entre de nouvelles classes, créées pour les besoins de définition du nouveau composant, dans le contexte de définition du nouveau composant (définition a priori),

- elles sont créées, a posteriori, entre des classes issues des composants réutilisés.

Ce troisième cas est la véritable situation d'assemblage de composants conceptuels préexistants. De façon plus précise, la création de relations d'associations intrinsèques bidirectionnelles entre classes existantes constitue le cœur de ce processus d'assemblage. En effet, la définition a posteriori des relations d'association ne

pose aucun problème dans le cas de relations d'associations extrinsèques. Par contre, la définition, a posteriori, de nouvelles relations d'association intrinsèques entre classes requiert le support de mécanismes spécifiques. Comme il n'est pas acceptable de modifier la définition des classes existantes[5] et que les relations d'association intrinsèques participent directement à la définition des classes, les relations d'associations intrinsèques définies a posteriori doivent nécessairement « provenir » des relations d'association intrinsèques définies a priori dans les composants. Ainsi, le langage de modélisation doit fournir des mécanismes permettant de construire, a posteriori, des relations d'association à partir des relations d'association définies a priori dans le contexte de définition des composants. Ces mécanismes sont typiquement la gestion de la spécialisation et de la composition entre relations d'association, mécanismes parfois évoqués mais peu étudiés et développés [Boc98].

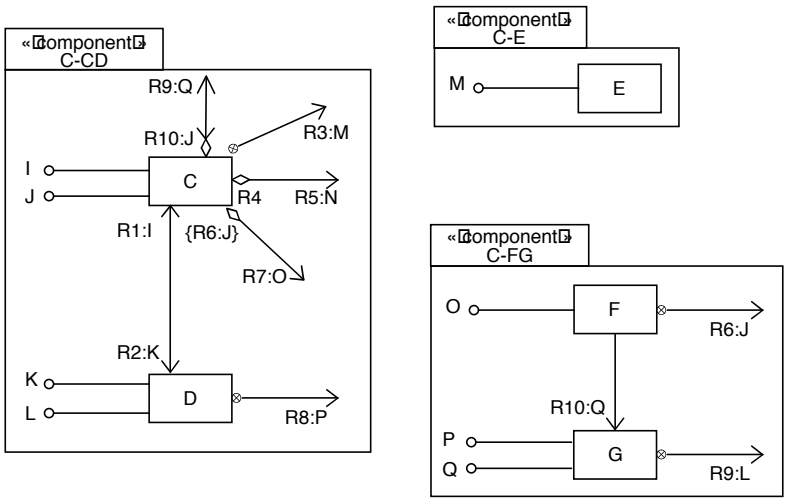

Figure 3. Exemple d'assemblage de composants

L'intérêt pratique de ces mécanismes est plus particulièrement de pouvoir assembler deux relations unidirectionnelles intrinsèques, exprimant les besoins de deux classes, en une relation d'association bidirectionnelle intrinsèque liant ces deux classes. Cette nouvelle relation exprime alors que, dans ce contexte

[5] La définition des classes réutilisées est préservée. Elle ne doit pas être modifiée par son utilisation dans un contexte donné. Si une nouvelle définition d'une classe d'objets réutilisée est requise, elle est introduite dans le modèle sous la forme soit d'une nouvelle classe par spécialisation ou composition (adaptation) des classes existantes, soit d'une version de classe. Il est possible que la modification d'une classe au sein d'un composant ait des impacts sur les autres classes du composant. [Urt98a, Urt98b] étudient ces impacts dans le cadre du versionnement.

d'utilisation, les deux classes d'objets collaborent pour satisfaire leurs besoins respectifs en jouant un rôle réciproque. La Figure 3. illustre l'intérêt de ce mécanisme pour l'assemblage de composants. Par exemple, les classes *D* et *G* sont connectées, dans le composant *C-CDEFG* résultant de l'assemblage des composants *C-CD*, *C-E* et *C-FG*, par une relation d'association bidirectionnelle intrinsèque exprimant le fait que ces deux classes jouent, l'une pour l'autre, un rôle réciproque. Cette relation est construite à partir des relations unidirectionnelles définissant les rôles *R8* et *R9*.

La spécialisation ou la composition de deux relations d'association unidirectionnelles en une relation d'association bidirectionnelle ne pose pas de problème sémantique. Chaque relation d'association unidirectionnelle ne définit la sémantique que d'une seule extrémité (un rôle), la sémantique de l'autre extrémité étant indéfinie. La sémantique de l'association bidirectionnelle résultante est obtenue en « affectant » à chaque extrémité le rôle spécifié par l'une des relations unidirectionnelles. Implicitement, l'association bidirectionnelle ajoute à cette sémantique la propriété de réciprocité (propriété qui, dans les modèles ne fournissant pas de relations bidirectionnelles, doit être représentée à l'extérieur des relations grâce, par exemple, à des contraintes [Gra97]). Cette construction peut être à la fois interprétée comme une spécialisation (on spécialise l'extrémité d'une relation sans sémantique précise en précisant son rôle) ou comme une composition (une relation est composée d'extrémités ; une relation peut alors être construite en réutilisant les extrémités d'une autre relation en tant que composants) de relations. D'autres formes de constructions de relations sont envisageables, telles que la construction de relations n-aires à partir d'un ensemble de n relations unidirectionnelles, ou bien des constructions basées sur la spécialisation ou la composition des rôles eux-mêmes, afin de pouvoir combiner des extrémités porteuses d'un rôle entre-elles. Ce type de construction permet, par exemple, de combiner deux relations bidirectionnelles en une nouvelle relation bidirectionnelle. Un autre exemple est celui présenté sur la Figure 3. où la relation de composition bidirectionnelle intrinsèque efférente à la classe *C* dans le composant *C-CD* est combinée à la relation d'association unidirectionnelle intrinsèque efférente à la classe *F* dans le composant *C-FG* pour définir la relation de composition bidirectionnelle intrinsèque liant les classes *C* et *F* dans le composant *C-CDEFG*. L'étude exhaustive des différentes formes et règles de construction de relations d'association, par spécialisation ou composition d'autres règles d'association, est hors des limites de cet article. C'est une perspective à nos travaux qui ne concernent véritablement, pour l'instant, que la construction de relations bidirectionnelles à partir de couples de relations unidirectionnelles.

La composition offre un cadre d'assemblage de relations plus souple que celui la spécialisation [Joh93]. En effet, la spécialisation de deux relations d'association unidirectionnelles en une relation d'association bidirectionnelle produit une relation d'association monolithique, définie de manière statique par héritage des propriétés des relations unidirectionnelles dont elle hérite[6]. La composition, au contraire, préserve la sémantique des relations unidirectionnelles. Une relation bidirectionnelle construite par composition est représentée comme une entité composite, explicitement composée des relations unidirectionnelles dont elle est issue. Un lien d'association bidirectionnel, instance d'une telle relation, est conséquemment représenté comme une entité composite, composée de deux liens d'association unidirectionnels. La Figure 3. présente le diagramme de composition d'une relation d'association bidirectionnelle, composée à partir de deux relations d'association unidirectionnelles. Il correspond à une vue détaillée de la relation d'association liant les classes *C* et *D* dans le composant *C-CD* (cf. Figure 3.). Sur la Figure 4. , les rôles de la relation bidirectionnelles sont présentés comme des rôles dérivés (précédés d'une barre oblique et en italique), pour signifier que les extrémités de la relation bidirectionnelles reprennent les rôles des relations unidirectionnelles dont elle est composée. La composition permet ainsi de conserver une « trace » explicite de la méthode de construction d'une relation d'association, qui peut servir de support à une certaine dynamique de déconstruction / reconstruction de relations, par réutilisation des relations qui les composent.

[6] Certains langages orientés objets, tels qu'UML, proposent un héritage dit « dynamique ». Cela signifie qu'il est possible de modifier la hiérarchie d'héritage en ajoutant ou retirant une relation de spécialisation. La définition de la classe s'en trouve alors modifiée. Pour un ensemble de relations de spécialisation donné, la définition d'une classe est figée : toutes les instances de la classe ont la même définition au même instant.

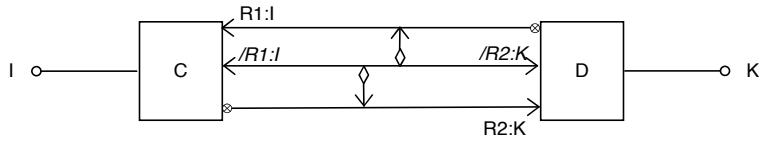

Figure 4. Diagramme de composition d'une relation d'association bidirectionnelle composée de deux relations d'association unidirectionnelles

3. Conclusion et perspectives

Nous avons étudié dans cet article les propriétés de la relation d'association qui influent sur leur capacité à être un support à la définition de composants conceptuels pour et par la réutilisation : la réciprocité, exprimée par les relations bidirectionnelles et la dépendance, exprimée par les relations intrinsèques. Nous en avons déduit les caractéristiques d'un modèle de relation d'association permettant de mettre en pratique les principes du développement à base de composants dirigés par les modèles : support de relations d'association unidirectionnelles pour exprimer, de manière isolée, les besoins de chaque composant ; support de mécanismes permettant de construire des relations d'associations bidirectionnelles à partir des relations unidirectionnelles, afin de ne pas intégrer les composants par simple juxtaposition mais par la définition de véritables collaborations ; support de mécanismes permettant, au contraire, de décomposer des relations d'association bidirectionnelles en associations unidirectionnelles pour produire, à partir d'un composant conceptuel, des composants conceptuels plus élémentaires. Une première perspective de ces travaux est la définition, au niveau du métamodèle d'UML, d'un modèle de relations d'association conforme à ces principes (le modèle de relations d'UML ne permettant pas, par exemple, l'expression de relations unidirectionnelles ou la composition entre relations) puis son implémentation, à fin d'expérimentation dans un AGL. Une deuxième perspective est l'étude exhaustive des différentes formes de construction d'une relation d'association à partir d'autres relations d'association, pour en évaluer les utilisations potentielles dans la gestion de la réutilisation de composants conceptuels. Une troisième perspective est une étude approfondie des possibilités offertes par la gestion dynamique des relations d'associations en tant qu'entités composites. Au niveau instance, cette dynamique pourrait permettre, par exemple, de gérer des mécanismes d'évolution du rôle d'un objet [Urt98b, Vau99b], au fil d'une collaboration. Au niveau classe, cette dynamique peut servir de base à la définition d'un « service de trading » de composants conceptuels, tel qu'il est prévu d'en outiller les futurs bus de connaissances MDA [Béz01]. Un tel service est chargé de référencer les composants existants, afin de pouvoir suggérer, pour un composant donné, des assemblages qui permettent de le mettre en œuvre en satisfaisant ses besoins de collaboration. Il s'agit, dans notre approche, de suggérer ces assemblages par la construction de nouvelles relations d'association, en composant les relations d'association existant dans les composants conceptuels référencés. Enfin, nous envisageons d'étendre la sémantique des rôles, pour adjoindre à la simple spécification du type d'objets attendu (définition des signatures des fonctionnalités requises), des informations assertionnelles (pré- et post- conditions) et comportementales (scénarios d'appels de fonctionnalités devant être supportés), afin que les contrôles de validité des collaborations établies entre composants ne se limitent pas à de simples vérifications syntaxiques.

4. Références bibliographiques

[Bar03] BARBIER F., HENDERSON-SELLERS B., LEPARC-LACAYRELLE A., BRUEL J.M., « Formalization of the Whole-Part Relationship in the UML », *IEEE Transactions on Software Engineering*, 29(5), 2003.

[Béz01] BEZIVIN J., « From Object Composition to Model Transformation with the MDA », *ToolsUSA,* 2001.

[Boc98] BOCK C., ODELL J., « A More Complete Model of Relations and Their Implementation: Aggregation », *Journal of Object-Oriented Programming*, Sept. 98.

[Cau99] CAUVET C., SEMMAK F., « La réutilisation dans l'ingénierie des systèmes d'information », *Génie Objet – Analyse et Conception de l'Evolution*, C. OUSSALAH et alii, Hermès, 1999.

[Civ98] CIVELLO F., « Rooted Class Diagrams : A notation for Context-Independent Models », *Journal of Object-Oriented Programming*, May 1998.

[Cla00] CLARKE S., « Composing design models: An extension to the UML », *UML'00*, Octobre 2000.

[Col03] COLLET P., ROUSSEAU R., « Contrôle d'admission de composants avec des contrats comportementaux », *LMO'03*, *L'Objet 1(2)*, 2003.

[Dso98] DSOUZA F., WILLS A.C., « Objects, Components and Framework with UML – The Catalysis Approach », Addison-Wesley, 1998.

[Dso01] DSOUZA D., « Omg's Mda, An Architecture for Modelling », www.omg.org , 2001.

[Duc97] DUCASSE S., « Intégration réflexive de dépendances dans un modèle à classes », Thèse de doctorat, Université de Nice - Sophia Antipolis., 1997.

[Fro99] FRONT-CONTE A., GIRAUDIN J.P., RIEU D., SAINT-MARCEL C., « Réutilisation et patrons d'ingénierie », *Génie Objet – Analyse et Conception de l'Evolution*, C. OUSSALAH et alii, Hermès, 1999.

[Fir98] FIRESMITH D.G., Henserson-Sellers B., « Clarifying Specialized Forms of Association in UML and OML », *Journal of Object-Oriented Programming*, May 1998.

[Gra95] GRAHAM I., « Migrating to Object Technology », Addison-Wesley, 1995.

[Gra97] GRAHAM I., BISCHOF J., HENDERSON-SELLERS B., « Associations considered a bad thing », *Journal of Object-Oriented Programming*, Février 1997.

[Hen98] HENDERSON-SELLERS B., « Open Relationships – Associations, Mappings, Dependencies, and Uses », *Journal of Object-Oriented Programming*, Février 1998.

[Hol92] HOLLAND I., « Specifying reusable components using contracts », *ECOOP'92*, 1992.

[Joh93] JOHNSON R. E., « Refactoring and Aggregation », Object Technologies for Advanced Software, *1st JSSST International Symposium*, Lecture Notes in Computer Science, 1993.

[OMG97] OMG, « A Discussion of the Object Management Architecture (OMA) », www.omg.org, Janvier 1997.

[OMG01a] OMG, « UML Specification 1.4 », www.omg.org, Septembre 2001.

[OMG01b] OMG, « Mof Specification 1.3.1 », www.omg.org, Novembre 2001.

[OMG01c] OMG, « The Common Object Request Broker : Architecture and Specification rev. 2.6 », www.omg.org, Décembre 2001.

[Ram96] RAMAZANI D., VON BOCHMANN G., « Extending OMT for the Specification of Composite Objects », *TOOLS USA'96*, 1996.

[Ree95] REENSKAUG T., WOLD P., LEHNE O., « Working with Objects », *The OORam Software Engineering Method*, Manning Publications Co., 1995.

[Rum95] RUMBAUGH J., « Taking things in context : Using composite objects to build models », *Journal of Object-Oriented Programming*, Novembre-Decembre 95.

[Ste01] STEVENS P., « On associations in the Unified Modelling Language », *UML'01,* 2001.

[Sun01] SUN MICROSYSTEMS INC., « Enterprise JavaBeans Specification, version 1.1 », java.sun.com, Janvier 2001.

[Urt98a] URTADO C., OUSSALAH C. « Complex entity versioning at two granularity levels », *Information Systems*, 23(2/3):197-216, Pergamon, Elsevier Science, 1998.

[Urt98b] URTADO C. « Versions d'entités complexes - Approches microscopique et macroscopique ». Thèse de doctorat, Université de Montpellier II, Oct. 1998.

[Vau99a] VAUTTIER S., MAGNAN M., OUSSALAH C., « Extended Specification of Composite Objects in UML », *Journal of Object-Oriented Programming*, Mai 1999.

[Vau99b] VAUTTIER S., « Une Etude de la Modélisation du Comportement des Objets Composites », Thèse de doctorat, Université de Montpellier II, Déc. 1999.

[W3C03] World Wide Web Consortium, "Web services architecture", www.w3.org/2002/ws/arch/, 2003.

[Win78] WINSTON M.E., CHAFFIN R., HERRMANN D., « A taxonomy of Part-Whole Relations », *Cognitive Science*, 11:417-444, 1978.

[Wir90] WIRFS-BROCK R., WILKERSON L., WIENER L., « Designing Object-Oriented Software », Prentice-Hall, 1990.