

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES  
DE SOPHIA ANTIPOLIS  
UMR 6070

# VERS LA NÉGOCIATION DE CONTRATS DANS LES COMPOSANTS LOGICIELS HIÉRARCHIQUES

*Hervé Chang, Philippe Collet*

*Projet OCL*

Rapport de recherche  
ISRN I3S/RR-2004-33-FR

Octobre 2004

---

# Vers la négociation de contrats dans les composants logiciels hiérarchiques <sup>1 2</sup>

**Hervé Chang — Philippe Collet**

*Équipe OCL, Objets et Composants Logiciels*

*I3S – CNRS – Université de Nice - Sophia Antipolis  
Les Algorithmes, Bât. Euclide, 2000 route des Lucioles  
BP 121, F-06390 Sophia Antipolis Cedex*

*{Herve.Chang,Philippe.Collet}@unice.fr*

---

*RÉSUMÉ. L'approche contractuelle se révèle bien adaptée aux besoins de spécification et vérification dans les systèmes à composants logiciels. Cependant, les contrats sont fréquemment remis en cause par des reconfigurations dynamiques et par la fluctuation des aspects non fonctionnels. Pour résoudre ce problème, nous présentons, dans cet article, un premier modèle de négociation de contrats qui permet d'automatiser le rétablissement de contrats valides. Nous décrivons aussi une politique de négociation par relâchement, qui convient bien à des contrats comportementaux exprimés par des assertions exécutables. Ce modèle s'intègre à **ConFract**, un système de contractualisation pour les composants logiciels hiérarchiques **Fractal**.*

*ABSTRACT. The contractual approach turns out to be well-suited to specification and verification needs in component-based software systems. However, contracts are frequently challenged by dynamic re-configurations and fluctuations of non functional aspects. To solve this problem, we propose, in this article, a first negotiation model which aims at automatically restoring the validity of contracts. We also describe a concession-based negotiation policy, which is well-suited to behavioral contracts with executable assertions. This model is integrated into **ConFract**, a contracting system for the **Fractal** hierarchical component model.*

*MOTS-CLÉS : génie logiciel orienté composant, négociation, contrat, composants hiérarchiques, systèmes multi-agents, Contract-Net Protocol, ConFract, Fractal.*

*KEYWORDS: Component-Based Software Engineering, Negotiation, Contract, Hierarchical Components, Multi-Agent Systems, Contract-Net Protocol, ConFract, Fractal.*

---

1. Cette recherche a été en partie financée par le contrat de recherche externe n° 422721832-I3S avec France Télécom R&D.

2. Cette soumission est une version étendue et améliorée d'un article soumis à JMAC'04 (Journée Multi-Agents et Composants, Workshop sans acte publié).

## 1. Introduction

Dans le domaine du génie logiciel, l'approche par composants est actuellement l'objet d'un intérêt croissant de la part de la communauté scientifique et des industriels. Cette approche présente notamment les avantages de mieux séparer interface et implémentation et de rendre l'architecture des applications explicite. Ainsi, un composant logiciel ne montre que ses interfaces requises et fournies, et des *contrats* basiques s'établissent lors des assemblages de composants. Il est maintenant reconnu que des contrats doivent prendre en compte des propriétés fonctionnelles plus finement décrites, ainsi que des propriétés non fonctionnelles (synchronisation, qualité de services, etc.) [BAC 00, SZY 02]. Cette notion de contrat devient primordiale lorsque les composants sont hiérarchiques. En effet, comme un composant peut être créé par assemblage d'autres composants, des propriétés de ces assemblages et du composite résultant doivent aussi être exprimées et contractualisées.

Pour répondre à ce besoin, nous avons conçu *ConFract*, un système de contractualisation pour composants logiciels [COL 04b, COL 04a]. Son objectif est d'établir et de vérifier des propriétés fonctionnelles et non fonctionnelles sur des composants hiérarchiques en s'appuyant sur des *contrats d'interface*, établis entre chaque connexion, et des *contrats de composition*, qui supervisent le contenu des composants. La mise en oeuvre de *ConFract* est basée sur la plate-forme *Fractal* [BRU 03], qui fournit notamment des possibilités de reconfiguration dynamique. Comme *ConFract* se doit aussi de prendre en compte des aspects non fonctionnels qui peuvent être très fluctuants, des modifications fréquentes et importantes interviennent sur les contrats et entraînent leur échec.

Pour rétablir la validité de ces contrats, nous proposons des mécanismes de négociation inspirés de ceux élaborés dans les systèmes multi-agents. Nous définissons ainsi un modèle de négociation, dans lequel une négociation atomique est associée à chaque disposition de contrat en échec. Cette négociation atomique se base sur une version adaptée du *Contract-Net Protocol* [SMI 88, COM 02] dans laquelle les composants impliqués dans les dispositions peuvent agir. Différentes politiques de négociation sont potentiellement applicables et nous décrivons ici une politique par relâchement, qui est bien adaptée à des contrats exprimés par des assertions exécutables.

La suite de cet article s'organise de la manière suivante. La section 2 décrit brièvement le système de contractualisation *ConFract* et définit notre problématique sur un exemple de lecteur multimédia. Dans la section 3, nous présentons les composantes de notre modèle de négociation ainsi que son fonctionnement. Nous discutons différents points en relation avec le modèle dans la section 4. La section 5 présente les travaux connexes et la section 6 conclut cet article.

## 2. Le système ConFract

*ConFract* se présente comme un système pour organiser sous forme de contrats la spécification et la vérification de propriétés fonctionnelles et non fonctionnelles sur des composants logiciels *Fractal*. A partir de spécifications, *ConFract* construit des

contrats lors des assemblages de composants. Ces contrats sont alors des objets de première classe pendant les phases de configuration et d'exécution des composants. Le système *ConFracta* a été conçu pour séparer clairement les mécanismes contractuels de l'expression des spécifications. Ainsi, différents formalismes peuvent être pris en compte, mais actuellement, seul un langage d'assertions exécutables, nommé *CCL-J* (*Component Constraint Language for Java*), est intégré au système. Ce langage, partiellement inspiré d'*OCL* [OBJ 97], est dédié à Java, car l'implémentation actuelle de *ConFracta* repose sur l'implémentation en Java de *Fractal* [BRU 04]. Les exemples de spécification présentés par la suite utilisent d'ailleurs le langage *CCL-J*. *ConFracta* introduit différentes formes de contrat pour tenir compte des spécificités du modèle *Fractal*. En effet, un composant *Fractal* peut exposer plusieurs interfaces, requises ou fournies. Ces interfaces sont composées d'un nom et d'une signature<sup>1</sup>. Des interfaces requises et fournies compatibles peuvent ainsi être connectées pour établir des communications entre composants. De plus, certains composants peuvent être composites et contenir d'autres composants. En *Fractal*, des contrôleurs permettent de gérer les aspects techniques des composants tout en appliquant une approche par séparation des préoccupations. Les plus utilisés gèrent le cycle de vie (LC sur la figure 1), les connexions (BC) et le contenu (CC). Le lecteur pourra se reporter à [COL 04a] pour une description plus détaillée.

Tout au long de cet article, nous nous appuyons sur un exemple de lecteur multimédia simplifié (cf. figure 1). Le composant de type *FractalPlayer* contient quatre sous-composants : *Player* qui fournit exclusivement le service de lecture vidéo (méthode *start*) et gère ses paramètres par des attributs, *GuiLauncher* qui gère l'interface graphique, *VideoConfigurator* qui fournit des services pour optimiser la lecture vidéo (la méthode *canPlay* détermine par exemple si une vidéo peut être visualisée entièrement avec une dimension donnée, en tenant compte des ressources disponibles comme la batterie et la mémoire) et enfin *Logger* permettant de gérer un historique des vidéos jouées (la méthode *lastUrl* permet d'obtenir l'url de la dernière vidéo).

## 2.1. Types de contrat

Dans le système *ConFracta*, deux grands types de contrat sont distingués. Un *contrat d'interface* est établi sur chaque connexion entre une interface requise et une interface fournie. Il regroupe les spécifications des deux interfaces, sachant que ces spécifications ne peuvent référencer que les méthodes de l'interface. Par exemple, une spécification *CCL-J* peut être associée à l'interface *MultimediaPlayer* (haut de la figure 1), pour décrire la précondition de la méthode *start* comme on le fait dans un système à objets. Un *contrat de composition*, quant à lui, est associé à la membrane d'un composant et regroupe toutes les spécifications définies sur cette membrane. On distingue d'une part un contrat de composition externe, dont les spécifications ne référencent que des interfaces visibles à l'extérieur de la membrane de

1. Cette signature peut être assimilée à une interface Java.

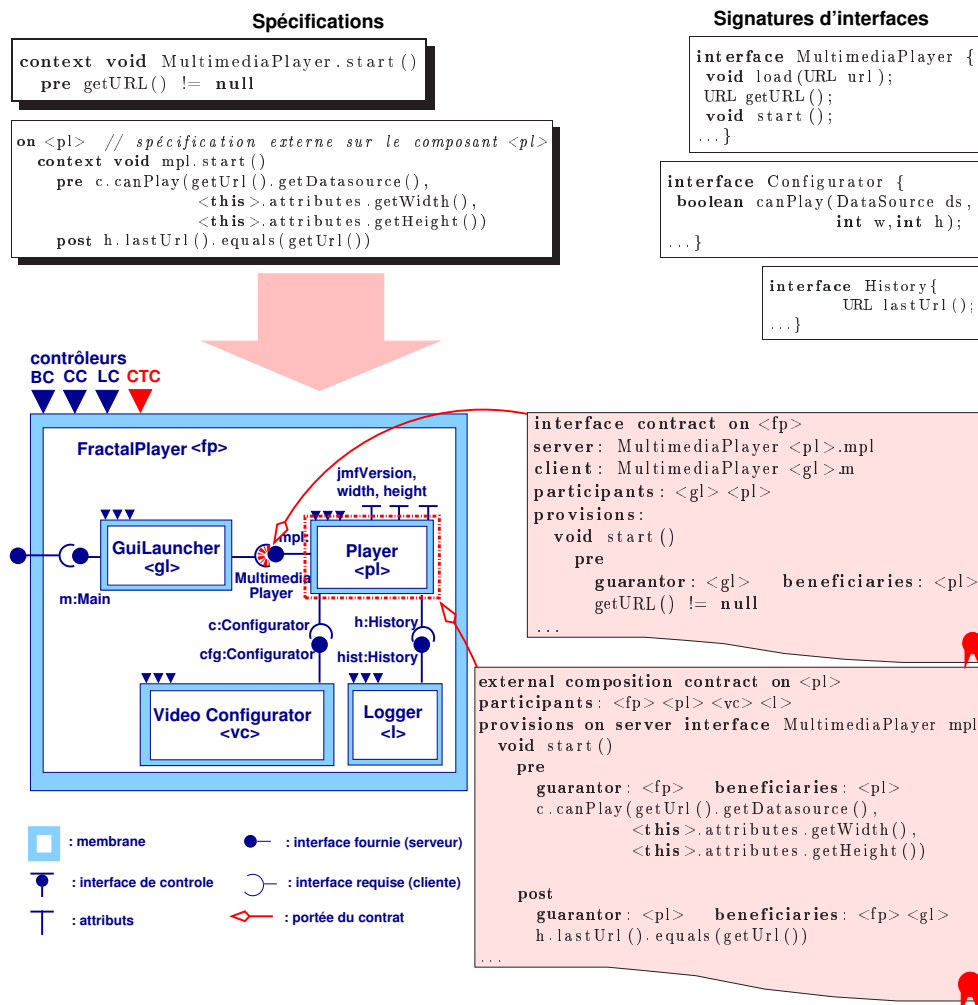


Figure 1 – Contractualisation du lecteur multimédia.

composant, et d'autre part, un contrat de composition interne, qui contient des spécifications qui référencent des interfaces dans le contenu du composant — soit des interfaces internes du composant, soit des interfaces externes de ses sous-composants —. Par exemple, la spécification concernant le composant <pl> donnée dans la figure 1 fera partie du contrat de composition externe de ce composant. Elle exprime en effet à la fois une précondition et une postcondition sur la méthode start de l'interface *Fractal* nommée *mpl*. La précondition référence une autre interface externe de <pl> — *c*, interface requise de type *Configurator* — pour exprimer la condition « la source vidéo peut-elle être entièrement jouée dans les dimensions configurées ? » La postcondition, quant à elle, référence une autre interface *h*, interface requise de type

History et vérifie le fait que la dernière entrée de l'historique correspond bien à l'ur1 de la vidéo venant d'être jouée. C'est bien cette référence à plusieurs interfaces sur le même composant qui définit le contrat de composition externe.

Les contrats construits par le système *ConFract* regroupent les spécifications suivant leur catégorie et chaque clause de spécification devient alors une *disposition* du contrat. Ces contrats contiennent aussi les références aux composants participants à l'évaluation de chacune des dispositions. Un métamodèle décrit en effet chaque type de spécification, par exemple les préconditions d'une méthode dans une spécification externe de composant en *CCL-J*, avec les responsabilités attribués aux composants impliqués. Ces composants impliqués peuvent ainsi être des garants (s'efforçant de rendre la spécification vraie), des bénéficiaires (s'appuyant sur la spécification vraie) ou des contributeurs (nécessaires à l'évaluation de la spécification). Ainsi, les responsabilités de garant et de bénéficiaire du contrat de composition externe du composant `<p1>` (fig. 1) sont données par la table suivante<sup>2</sup> :

Type d'interface	Construction	Garant	Bénéficiaires
serveur (mpl)	pre	<fp>	<p1>
serveur (mpl)	post	<p1>	<fp>, <g1>
cliente (c)	pre	<p1>	<fp>, <vc>
cliente (c)	post	<fp>	<vc>

Il est à noter que le garant de certaines conditions, comme la précondition sur une méthode serveur (interface `mpl`), est le composant englobant, ici `<fp>`, car il est le seul à voir cette précondition. En effet, le composant `<g1>`, utilisateur de l'interface `mpl`, n'est garant que du contrat d'interface sur la connexion. C'est bien le composant englobant qui est responsable de la bonne utilisation générale de ses sous-composants.

## 2.2. Gestion des contrats

Dans *ConFract*, les contrats sont gérés par des contrôleurs de contrats (CTC sur la figure 1) placés sur la membrane de chaque composant composite. Chaque contrôleur de contrats prend en charge le cycle de vie et l'évaluation (*i*) du contrat de composition interne du composite sur lequel il est placé, (*ii*) du contrat de composition externe de chacun des sous-composants et (*iii*) du contrat d'interface de chaque connexion dans son contenu. Le contrôleur de contrat utilise le mécanisme de mixin de *Fractal* pour effectuer des intercessions sur les contrôleurs de cycle de vie, de connexion et de contenu. Il construit et met à jour les contrats en fonction des événements d'assemblage qui surviennent sur les composants (insertion d'un sous-composant, connexion de deux interfaces, etc.).

Les dispositions des contrats de configuration qui définissent des invariants sur les composants sont vérifiées à la fin de la phase de configuration. Dans le cas du langage

2. Nous ne montrons que les responsabilités de ce type de contrat pour des pré et postconditions.

*CCL-J*, toutes les autres dispositions sont vérifiées dynamiquement. Lorsqu'une méthode est appelée sur une interface *Fractal*, les préconditions du contrat d'interface sont d'abord évaluées, puis celles du contrat de composition externe du composant recevant l'appel, et enfin celles du contrat de composition interne. Des vérifications, dans l'ordre inverse, sont effectuées au retour de la méthode pour les postconditions et les invariants.

### 2.3. Problématique

Le système *ConFracta* pour objectif de prendre en compte des aspects fonctionnels et non fonctionnels. Par exemple, dans la figure 1, la précondition du contrat de composition externe établit les bonnes conditions de lecture de la source vidéo. Le composant *VideoConfigurator* va ainsi établir le résultat de la méthode *canPlay* en fonction de différents paramètres du système (niveau de la batterie) et de la source vidéo (complexité du décodage). La fluctuation de certaines propriétés non fonctionnelles peut ainsi couramment entraîner la violation de bon nombre de contrats. Actuellement, le système *ConFract* gère ces violations en notifiant le composant garant de la disposition en échec et en lui fournissant toutes les informations sur le contexte de la violation. Ceci implique la multiplication de codes de rattrapage pour rétablir des contrats valides alors qu'intuitivement, un grand nombre d'entre eux semblent négociables. Dans notre exemple, on pourrait consommer moins d'énergie en diminuant la taille de l'affichage, voire retirer la condition tout en sachant que la lecture pourra s'interrompre.

## 3. Modèle de négociation

### 3.1. Principes

De manière générale, les contraintes exprimées dans les contrats portent sur des aspects fonctionnels et non fonctionnels. La négociation des propriétés fonctionnelles reste surtout valable lors des tests. En revanche, comme les aspects non fonctionnels spécifient la qualité de fonctionnement des composants (qualité des services) et leurs relations avec l'environnement (contraintes de déploiement), nous proposons de négocier des dispositions qui spécifient de tels aspects. Pour cela, comme les contrats sont découpés en dispositions, notre modèle de négociation permet de rétablir un contrat valide en déclenchant une *négociation atomique* pour chaque disposition en échec, en vue de la rétablir. Cette négociation peut intervenir lors de la phase de configuration pour une disposition vérifiable statiquement, ou à l'exécution pour une disposition vérifiable à l'exécution. La figure 2 décrit l'intégration de la négociation dans le processus de contractualisation des composants. De plus, comme le métamodèle de *ConFract* permet d'identifier précisément les responsabilités des participants de chaque disposition, notre modèle de négociation s'appuie sur la connaissance de leur rôle de garant, bénéficiaire ou contributeur pour modifier les dispositions. Enfin,

comme les contrats suivent la structure hiérarchique, la négociation n'opère qu'au niveau de hiérarchie concerné. Il est ainsi possible, d'après ces responsabilités, d'adopter différents types de *politique* qui définissent le raisonnement des participants et orientent la négociation atomique.

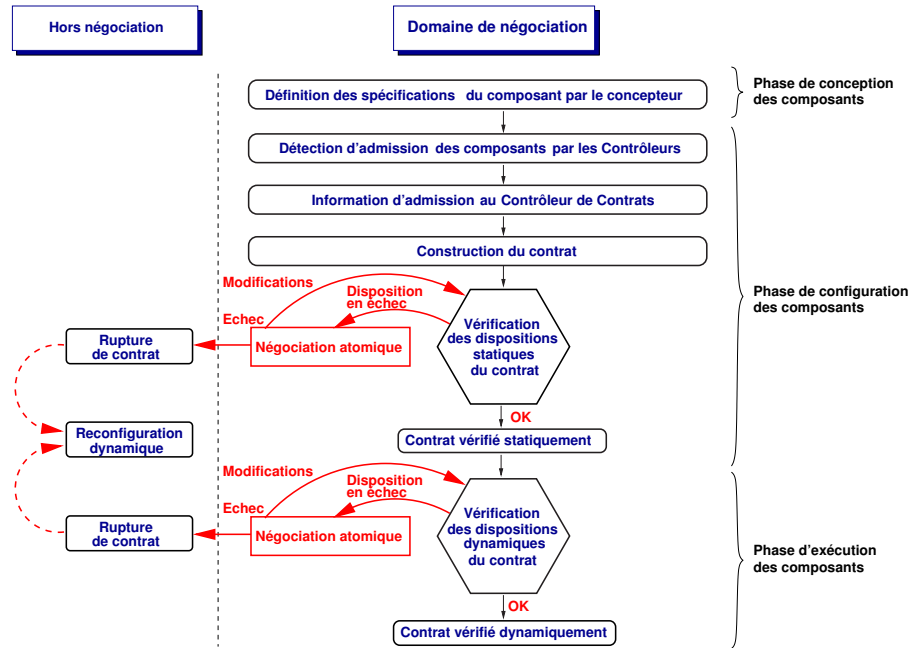


Figure 2 – Intégration de la négociation dans le processus de contractualisation.

### 3.2. Protocole et participants de la négociation atomique

La négociation atomique fait intervenir un initiateur et des participants, et se déroule selon un protocole inspiré du *Contract-Net Protocol* étendu (cf. section 5). Les parties négociantes sont (i) le contrôleur de contrat dans le rôle de l'initiateur, car c'est lui qui gère le cycle de vie du contrat et réalise les vérifications, (ii) les participants de la disposition en échec, et enfin (iii) un *participant externe* qui permet de représenter les intérêts d'une entité possédant une capacité de décision supérieure. Ainsi, par le biais de cette partie, il serait possible d'intégrer des contraintes portant sur le système global (contraintes de déploiement) et d'injecter des données externes au système pour paramétrer le processus de négociation (durée limite de la négociation). Dans l'exemple du contrat de composition, les parties qui négocient la précondition de la méthode `start` sont le contrôleur de contrat de `<fp>` en tant qu'initiateur, `<fp>` lui-même en tant que garant et `<p1>` au titre de bénéficiaire (cf. figure 3). Pour la post-condition, les parties sont le contrôleur de contrat de `<fp>` en tant qu'initiateur, `<p1>`

au titre de garant, et <fp> et <gl> en tant que bénéficiaires.

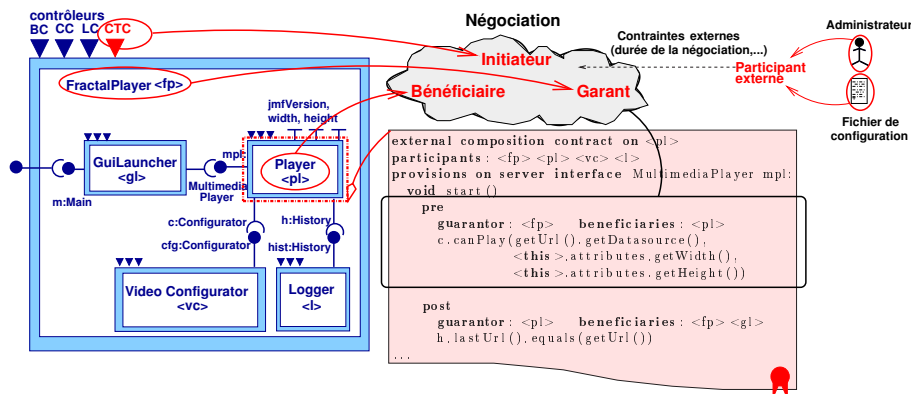


Figure 3 – Illustration des parties négociantes pour la précondition du contrat de composition externe.

Le protocole de négociation atomique, quant à lui, s'organise en trois étapes. L'initiateur *demande des modifications* aux parties sur la disposition en échec et les parties *soumettent des modifications*. Par la suite, l'initiateur effectue les modifications proposées et re-vérifie la disposition en vue de la rétablir. Ces étapes sont répétées et la négociation se termine, soit lorsqu'une solution satisfaisante est trouvée et la négociation finalisée, soit par la rupture du contrat après échec des modifications ou dépassement de la durée limite de la négociation. Ces étapes régissent ainsi le processus de négociation atomique, toutefois il est nécessaire de définir des politiques qui vont établir les stratégies de cette négociation.

### 3.3. Politique de négociation atomique

Nous présentons ici une politique par relâchement dont le principe consiste à s'orienter vers les bénéficiaires de la disposition pour qu'ils s'appuient sur une disposition moins contrainte en procédant au *relâchement des contraintes*. Ainsi, elle peut conduire au changement de la disposition en échec en conservant le contexte d'exécution du système ou à la modification des attributs des bénéficiaires à disposition constante. Cette politique est suffisamment générale pour être applicable du moment que le formalisme de spécification utilisé permette le découpage des expressions en dispositions. Nous présentons dans la partie 4 quelques pistes pour mettre en oeuvre d'autres politiques.

Dans la mise en oeuvre de la politique de relâchement, la notion de bénéficiaire est affinée en distinguant les bénéficiaires principaux des bénéficiaires auxiliaires. Les bénéficiaires principaux sont directement concernés par la disposition et ont la capacité de modifier la disposition. En revanche, les bénéficiaires auxiliaires ont un rôle plus

passif et ne peuvent pas faire avancer la négociation. Ainsi, pour la postcondition de la méthode `start` de l'interface `mp1`, `<fp>` est le bénéficiaire principal car responsable de la correcte utilisation du contrat (l'historique des vidéos jouées est correcte) et `<gl>` bénéficiaire auxiliaire, car simple client du service et plus indirectement concerné.

En cas d'échec d'une vérification, le processus de relâchement se décompose en trois phases décrites à la figure 4. Tout d'abord (phase 1), l'initiateur demande la négociabilité de la disposition en échec à tous les bénéficiaires et détermine le résultat par une fonction additive pondérée. Si la disposition n'est pas négociable, la négociation atomique échoue. Dans le cas contraire (phase 2), l'initiateur demande aux bénéficiaires principaux de *faire des concessions* en leur décrivant la disposition en échec, et les bénéficiaires principaux *proposent des concessions*. Les bénéficiaires principaux peuvent alors renvoyer soit une nouvelle disposition, soit une alternative qui décrit le relâchement à effectuer sur leurs attributs. Par la suite, l'initiateur effectue les relâchements proposés et réévalue la disposition. Si l'évaluation de la disposition est fautive, l'initiateur annule les modifications pour rétablir le contexte initial et redemande des concessions. Au contraire, si l'évaluation est vraie alors la négociation atomique est réussie, et l'initiateur finalise la négociation en demandant aux bénéficiaires d'effectuer les relâchements correspondants. Autrement (phase 3), si l'initiateur ne reçoit que des demandes de retrait venant des bénéficiaires principaux, il procède à une dernière consultation pour demander aux bénéficiaires principaux et auxiliaires l'autorisation de retirer la disposition.

Les bénéficiaires principaux peuvent raisonner en s'appuyant sur un ensemble d'alternatives qui définit les relâchements. Un composant  $C$  associe ainsi à la disposition  $\#disp$  un ensemble d'alternatives (formules ou valeurs) ordonnées  $\mathcal{A}_{\#disp,C} := \{A_{\#disp,C}^1, A_{\#disp,C}^2, \dots, A_{\#disp,C}^n, \text{STOP ou RELEASE}\}$ . Dans le protocole de négociation, à chaque fois que l'initiateur émet une demande de concession au composant, celui-ci renvoie un relâchement, correspondant à une nouvelle disposition ou une modification des attributs, nouvellement construit à partir de cet ensemble d'alternatives. La dernière alternative STOP, resp. RELEASE, permet de notifier la fin du relâchement en conservant la disposition, resp. en autorisant le retrait de la disposition.

Dans notre exemple, la disposition portant sur la capacité du lecteur à jouer une vidéo dans sa totalité est vérifiable à l'étape d'exécution des composants, car elle nécessite le contexte d'exécution et l'accès aux ressources. En cas d'échec, la négociation par relâchement fait intervenir le contrôleur de contrat de `<fp>`, et `<p1>` en tant que bénéficiaire principal. Il n'y a, dans ce cas, pas de bénéficiaire auxiliaire. Ainsi, le processus de négociation par relâchement conduirait à consommer moins de ressources batterie, par exemple, en modifiant successivement les paramètres d'affichage de la vidéo pour pouvoir la visualiser dans sa totalité. Si cela reste insuffisant, il est alors possible d'imaginer le retrait complet de la contrainte, et la vidéo pourra être interrompue en cours de lecture si la batterie s'avère être insuffisante. Dans un tel scénario, `<p1>` définit un ensemble d'alternatives  $\mathcal{A}_{pre,<p1>} := \{(width \leftarrow \frac{width}{\sqrt{2}}, height \leftarrow \frac{height}{\sqrt{2}}), \text{RELEASE}\}$  qui représente les relâchements à effectuer sur les attributs d'affichage de la vidéo. De cette manière, à la première demande de concession, `<p1>` renvoie une alternative qui décrit la modification à réaliser sur les attributs `width` et `height`. L'ini-

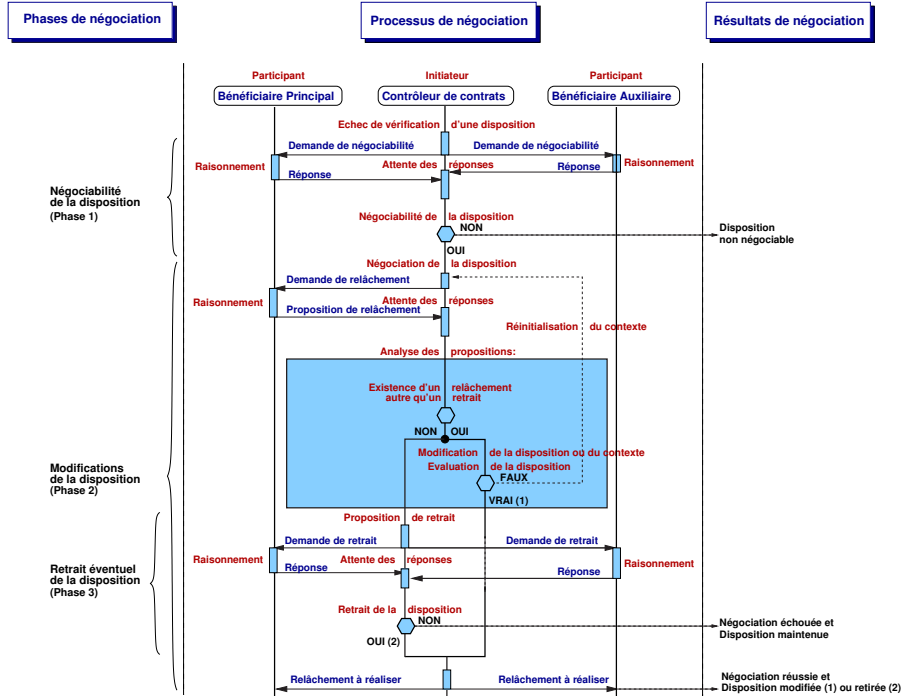


Figure 4 – Processus de négociation atomique par relâchement.

tiateur modifie alors les attributs d’affichage de  $\langle p1 \rangle$  et si la disposition est satisfaite, la négociation atomique s’achève par un succès. Sinon, le contrôleur de contrat annule la modification faite et redemande une concession à laquelle  $\langle p1 \rangle$  répond par RELEASE pour proposer le retrait de la disposition. Comme il est l’unique bénéficiaire, la disposition est retirée et la négociation réussit.

La disposition portant sur la cohérence des vidéos jouées est aussi vérifiable à l’étape d’exécution des composants. La négociation de cette disposition ferait intervenir le contrôleur de contrat de  $\langle fp \rangle$  en tant qu’initiateur,  $\langle p1 \rangle$  au titre de garant, ainsi que  $\langle fp \rangle$  et  $\langle g1 \rangle$  en tant que bénéficiaires principal et auxiliaire respectivement. Dans le processus de négociation,  $\langle fp \rangle$ , étant bénéficiaire principal, proposerait par exemple le retrait de la disposition ( $\mathcal{A}_{post, \langle p1 \rangle} := \{RELEASE\}$ ) à l’initiateur, lequel consultera  $\langle fp \rangle$  et  $\langle g1 \rangle$  pour décider du retrait définitif.

### 3.4. Autres exemples de mise en oeuvre

Il est aussi possible spécifier la composition interne des composants, par exemple, en contraignant  $\langle fp \rangle$  de façon à ce que le sous-composant  $\langle p1 \rangle$  utilise une version de l’API JMF (Java Media Framework) au moins plus récente que 2. 1. La figure 5

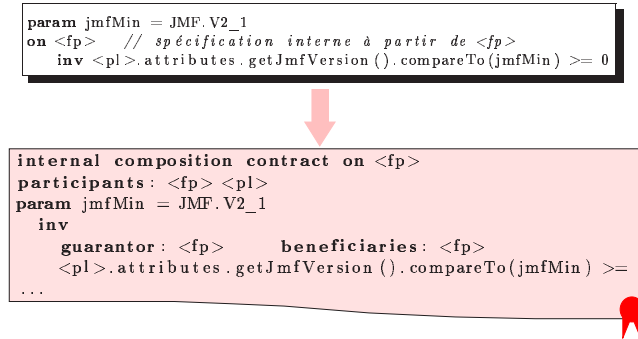


Figure 5 – Spécification de la JMF et contrat de composition interne résultant.

présente une spécification possible de cette contrainte en *CCL-Jet* le contrat de composition interne résultant. Dans ce cas, la disposition du contrat exprime un invariant de configuration et est vérifié à l'étape de configuration des composants. Ainsi, si un composant  $\langle p1 \rangle$  venait à utiliser une version du JMF moins récente que 2.1, alors la négociation ferait intervenir le contrôleur de contrat de  $\langle fp \rangle$ , et  $\langle fp \rangle$  lui-même au titre de garant et bénéficiaire.  $\langle fp \rangle$  pourrait alors relâcher sa contrainte en proposant, d'après sa liste d'alternative  $\mathcal{A}_{inv, \langle fp \rangle} := \{jmfMin \leftarrow 2.0, STOP\}$ , la version 2.0 de JMF. Si le relâchement reste insuffisant alors  $\langle fp \rangle$  signifierait l'arrêt de ses relâchements avec STOP. La contrainte serait alors maintenue et la négociation s'achèverait à ce stade par la rupture du contrat. Il faudrait alors employer d'autres politiques ou réaliser des traitements particuliers (mise à jour, reconfigurations) sur  $\langle fp \rangle$  ou  $\langle p1 \rangle$ .

#### 4. Discussion

*Autres politiques de négociation.* Pour améliorer nos mécanismes de négociation, d'autres politiques sont envisageables et pourront être employées à la suite de la politique par relâchement ou combinée à celle-ci. Une politique par effort consisterait ainsi à orienter la négociation vers les garants des dispositions. Toutefois, pour la développer, il est nécessaire de pouvoir s'appuyer sur des informations, exprimées ou déduites des dispositions. Cela implique d'étudier l'expression des contraintes non fonctionnelles mais aussi de propager la négociation vers des composants de hiérarchies inférieures en s'appuyant sur un modèle compositionnel. Un composant garant pourrait typiquement orienter la négociation vers les contributeurs pour que ces derniers puissent *faire un effort* sur une propriété non fonctionnelle si celle-ci était déterminée de manière compositionnelle et que ce composant puisse *reconfigurer* son contenu. Par ailleurs, à partir des connexions entre composants, déduites du raisonnement compositionnel, d'autres stratégies de négociation pourraient être développées, par exemple une stratégie de donnant-donnant pourrait être appliquée sur plusieurs dispositions en même temps en combinant effort et relâchement.

*Négociation des contrats d'interfaces.* Cette négociation peut être vue comme une forme simplifiée de celle relative aux contrats de composition. Elle fait en effet intervenir simplement le contrôleur de contrat du composant englobant (initiateur), et les composants client et serveur (garant et bénéficiaire). De plus, comme dans *ConFract*, les contrats d'interfaces spécifient des aspects fonctionnels, la seule politique cohérente consisterait vraisemblablement à retirer complètement la disposition en échec si le bénéficiaire l'autorise.

*Mise en oeuvre dans ConFract.* Comme les contrats sont gérés dans *ConFract* par des contrôleurs spécifiques, nous intégrons simplement les mécanismes de négociation en enrichissant chaque contrôleur de contrats afin (i) qu'il pilote la négociation du contrat de composition interne de son composant et les négociations des contrats d'interface du contenu de son composant, et (ii) qu'il réponde aux interrogations (négociabilité, propositions de modification, validation de relâchement) du contrôleur englobant dans les négociations du contrat de composition externe de son composant et de ses contrats d'interfaces.

## 5. Travaux connexes

*Négociation dans les systèmes multi-agents.* Dans les systèmes multi-agents, la négociation permet aux agents de se coordonner, de se partager des ressources limitées ou de résoudre un conflit. Le protocole de négociation le plus souvent utilisé se base sur le *Contract Net Protocol* [SMI 88] ou sur une version étendue [COM 02] qui définit un processus général de négociation en faisant intervenir un initiateur et des participants et en se basant sur des itérations de primitives simples (proposer, refuser, accepter) organisées en trois étapes (l'appel d'offres, la réception et l'analyse des offres, et la contractualisation). Les capacités de raisonnement sont, quant à elles, souvent adaptées en fonction des applications cibles. Ainsi, dans les systèmes de négociation qui modélisent des places de marché virtuelles, le raisonnement des agents se base fréquemment sur des stratégies prédéfinies (fonctions de type linéaire, quadratique ou additive pondérée) alors que d'autres travaux [FAR 99] détaillent des stratégies plus sophistiquées qui distinguent clairement l'évaluation des propositions de leur formulation ; elles ont été appliquées par Jennings et al. pour la négociation de ressources dans les réseaux. Bien que notre modèle de négociation soit inspiré de protocoles équivalents dans les systèmes multi-agents, l'autonomie résultante ne concerne que le rétablissement des contrats par la détermination dynamique d'une nouvelle configuration valide des composants. Ceci est donc plus proche du concept d'*autonomic computing* [IBM 03] que des notions d'autonomies sociales et environnementales des agents.

*Négociation de contrats logiciels.* De nombreux travaux s'appuient sur le formalisme QML (*QoS Modeling Language*) [FRØ 98] pour contractualiser les propriétés de Qualité de Service (QoS). QML permet de décrire de tels contrats en spécifiant les niveaux des qualités attendus, mais ne permet pas, au contraire de *CCL-J*, de combiner des aspects fonctionnels et non fonctionnels dans une même spécification. Ainsi, cer-

tains travaux utilisent QML pour spécifier les qualités des services dans les systèmes distribués [BEC 99, KOI 98] ou les plate-formes à composants [RIT 03, LOQ 04]. Dans [BEC 99, RIT 03, KOI 98], la négociation se déroule entre un client et un serveur afin d'établir un contrat initial de QdS. Ainsi, le client établit un dialogue avec le serveur pour choisir, éventuellement par une fonction d'utilité [KOI 98], le contrat de QdS le plus approprié parmi ceux exposés par le serveur. En revanche, en cas de variations de QdS, les adaptations nécessaires pour obtenir un nouveau contrat (renégociation automatisée, mise en oeuvre de politiques particulières) ne sont pas explicitées. Dans [LOQ 04], les adaptations sont exprimées statiquement dans le contrat sous la forme de service alternatif à essayer successivement. Dans notre modèle, les cas de fluctuation de QdS sont traités en négociant, de manière plus fine, sur chacune des dispositions du contrat en échec. De plus, notre négociation atomique est déclenchée avec la connaissance des composants responsables de la disposition en échec, et comme le système est ouvert, son déroulement dépend des propres capacités des composants.

## 6. Conclusion

Dans cet article, nous avons proposé des mécanismes de négociation adaptés à des contrats placés sur des composants logiciels hiérarchiques. Ces mécanismes ont été développés pour *ConFract*, un système de contractualisation pour les composants *Fractal*, qui permet de poser des contrats à la fois sur les connexions entre les composants et sur les composants eux-mêmes. Notre modèle de négociation s'inspire en partie des protocoles de négociation des systèmes multi-agents. Il reprend le principe du *Contract-Net Protocol* en l'appliquant à chaque niveau de hiérarchie : le contrôleur de contrats est alors l'initiateur de la négociation et les participants du contrat sont consultés selon une politique qui peut exploiter les notions de garant et bénéficiaire de chaque disposition de contrat. Nous avons aussi décrit une politique de négociation par relâchement qui est bien adaptée au langage d'assertions exécutables *CCL-J*, actuellement utilisé dans *ConFract*. Dans cette politique, l'initiateur consulte les bénéficiaires des dispositions négociées, pour essayer de réduire les contraintes posées. Nous obtenons ainsi de l'autonomie dans le rétablissement des contrats par la détermination dynamique d'une nouvelle configuration valide des contrats et des composants.

Pour généraliser notre approche, il nous faut maintenant développer de nouvelles politiques de négociation et prendre en compte des reconfigurations plus importantes des composants. Afin de raisonner de manière compositionnelle sur les propriétés, nous comptons élaborer un modèle compositionnel qui permettrait d'exprimer des propriétés non fonctionnelles dans la hiérarchie des composants. Il sera alors possible de proposer des politiques de négociation par effort ou de donnant-donnant, et de piloter la reconfiguration des composants logiciels en les remplaçant ou en modifiant leur contenu.

## 7. Bibliographie

- [BAC 00] BACHMAN F., BASS L., BUHMAN C., COMELLA-DORDA S., LONG F., ROBERT J., SEACORD R., WALLNAU K., « Technical Concepts of Component-Based Software Engineering », rapport n° CMU/SEI-2000-TR-008, mai 2000, Carnegie Mellon Software Engineering Institute, Volume 2.
- [BEC 99] BECKER C., GEIHS K., « Generic QoS Specification for CORBA », *Kommunikation in Verteilten Systemen (KiVS'99)*, Darmstadt, Germany, März 2-5 1999.
- [BRU 03] BRUNETON E., COUPAYE T., STEFANI J.-B., « The Fractal Component Model », Specification, Technical Report n° v1, v2, 2002,2003, The ObjectWeb Consortium, <http://fractal.objectweb.org>.
- [BRU 04] BRUNETON E., COUPAYE T., LECLERCQ M., QUÉMA V., STEFANI J.-B., « An Open Component Model and Its Support in Java », *ICSE 2004 - CBSE7*, vol. 3054 de LNCS, Springer Verlag, mai 2004.
- [COL 04a] COLLET P., ROUSSEAU R., « ConFract : un système pour contractualiser des composants logiciels hiérarchiques », rapport, octobre 2004, n° ISRN I3S/RR-2004-32-FR I3S.
- [COL 04b] COLLET P., ROUSSEAU R., « Contracting Hierarchical Components », rapport, mars 2004, n° ISRN I3S/RR-2004-09-FR I3S.
- [COM 02] COMMUNICATION F. T., « FIPA Contract Net Interaction Protocol Specification », rapport, 2002, FIPA Organization.
- [FAR 99] FARATIN P., SIERRA C., JENNINGS N., BUCKLE P., « Designing Flexible Automated Negotiators : Concessions, Trade-Offs and Issue Changes », rapport n° RP-99-03, 1999, Institut d'Investigacio en Intel.ligencia Artificial Technical Report.
- [FRØ 98] FRØLUND S., KOISTINEN J., « QML : a language for Quality of Service Specification », rapport n° HPL-98-10, février 1998, Software Technology Laboratory, Hewlett-Packard.
- [IBM 03] IBM RESEARCH, « Autonomic Computing », 2003, <http://www.research.ibm.com/autonomic/index.html>.
- [KOI 98] KOISTINEN J., SEETHARAMAN A., « Worth-Based Multi-Category Quality-of-Service Negotiation in Distributed Object Infrastructures », rapport, 1998, HP Labs.
- [LOQ 04] LOQUES O., SZTAJNBERG A., « Customizing Component-Based Architectures by Contract », *Proceedings of Component Deployment (CD 2004)*, Edinburgh, UK, May 2004.
- [OBJ 97] OBJECT MANAGEMENT GROUP I., « Object Constraint Language Specification », rapport n° version 1.1, ad/97-08-08, septembre 1997, IBM [www.software.ibm.com/ad/ocl](http://www.software.ibm.com/ad/ocl).
- [RIT 03] RITTER T., BORN M., UNTERSCHUTZ T., WEIS T., « A QoS Metamodel and its Realization in a CORBA Component Infrastructure », *Proceedings of the 36' HICSS Hawaii*, janvier 2003.
- [SMI 88] SMITH R. G., « The contract net protocol : high-level communication and control in a distributed problem solver », *Distributed Artificial Intelligence*, , 1988, p. 357–366, Morgan Kaufmann Publishers Inc.
- [SZY 02] SZYPERSKI C., *Component Software — Beyond Object-Oriented Programming*, Addison-Wesley Publishing Co. (Reading, MA), 2nd édition, 2002.