Université Joseph Fourier

U.F.R Informatique & Mathématiques Appliquées

Institut National Polytechnique de Grenoble

ENSIMAG

I.M.A.G

DEA D'INFORMATIQUE :

SYSTEMES ET COMMUNICATIONS

Projet présenté par :

Claire LECOURTIER

REVISION COLLABORATIVE D'UN SERVEUR DE CONNAISSANCES

Effectué au laboratoire :

INRIA Rhônes-Alpes, projet SHERPA

Date : 22 Juin 1998

Jury : Jacques Briat
Joëlle Coutaz
Jérôme Euzenat
Maria-Antonietta Grasso
Brigitte Plateau

REMERCIEMENTS

Je remercie Jérôme Euzenat pour m'avoir fait découvrir ce qu'était un travail de recherche et m'avoir guidée et soutenue pendant toute cette année.

Je remercie Loïc Tricand De La Goutte, pour le support technique qu'il m'a fourni.

Je remercie l'ensemble de mon jury d'avoir examiné mon travail.

Enfin, je remercie toues les personnes que j'ai rencontrées et côtoyées à l'INRIA pour tous les bons moments de détente que nous avons pu partager.

SOMMAIRE

Résumé

Depuis longtemps, le problème de la représentation de la connaissance se pose aux informaticiens. Une des solutions actuelles est l'utilisation de langages de type « langage à objets ». C'est l'approche qui a été retenue dans le système de gestion de bases de connaissances « Troeps ». Lorsque l'on ajoute une nouvelle assertion dans une base de connaissance, celle-ci peut s'avérer inconsistante. Il faut donc réviser la base. Lorsque la base est partagée par un ensemble de spécialistes de différents sujets, l'auteur de la connaissance conflictuelle peut ne pas comprendre pourquoi la nouvelle assertion déclenche un conflit. L'objectif de ce travail est de développer un protocole permettant à l'auteur de la connaissance conflictuelle de prendre contact avec les spécialistes compétents pouvant l'aider à résoudre le conflit. Ce protocole gère l'argumentation développée par l'ensemble des intervenants pour choisir la meilleure modification à apporter à la base pour résoudre le conflit, et ceci de façon structurée en s'appuyant sur la structure logique de la base. Il a été implémenté sur le système de gestion de bases de connaissances « Troeps ».

Mots-Clés

Argumentation, Travail collaboratif, CSCW, Révision de bases de connaissances

I. Introduction

Le travail présenté ici a été réalisé à l'INRIA Rhônes-Alpes, au sein du projet SHERPA. L'un des objectifs de ce projet est le développement d'un environnement de construction de bases de connaissances. Cet environnement est appliqué à la constitution coopérative de bases de connaissances scientifiques ainsi que de mémoires techniques. Pour cela ont été développés un système de gestion de bases de connaissances par objets, un protocole de soumission de connaissance et un éditeur de bases de connaissances sur le World-wide web.

L'un des problèmes majeurs lors de la constitution d'une base de connaissance est de conserver la consistance de l'ensemble des informations qu'elle contient. En effet, lorsqu'un utilisateur introduit une nouvelle connaissance dans la base, celle-ci peut entrer en conflit avec les éléments déjà contenus dans cette dernière. Il faut alors réviser la base en supprimant une partie minimale de la connaissance qu'elle contient de façon à pouvoir introduire la nouvelle connaissance tout en conservant la cohérence de l'ensemble. Il y a souvent plusieurs solutions pour résoudre le conflit. Un des problèmes qui se posent lors de la révision est que l'auteur de la connaissance conflictuelle ne maîtrise pas forcément tous les domaines concernés par le conflit. Le but de ce travail est donc d'aider l'utilisateur à résoudre son problème en dirigeant le dialogue entre les différents spécialistes (généralement les auteurs des connaissances conflictuelles) et lui-même.

Dans un premier temps, nous présenterons la problématique de la représentation de connaissance par objets en nous appuyant sur l'exemple de Troeps qui est le système à bases de connaissances développé au sein du projet SHERPA et à partir duquel ce travail a été réalisé. Nous développerons ensuite la problématique de la révision sur les bases de connaissance, et verrons plus en détail ce qu'il en est dans le cas de l'utilisation des objets. Dans un troisième temps, nous aborderons la problématique posée par le travail collaboratif, ainsi que les différents modèles d'argumentation existants. Dans une quatrième partie, nous introduirons le modèle d'argumentation développé dans le cas particulier de la révision sur les bases de connaissance à objet. Enfin, nous parlerons de l'implémentation réalisée et des difficultés rencontrées pour celle-ci.

II. Bases de connaissances à objets

Comme nous venons de le voir, l'objectif de ce travail est de développer un système permettant d'aider les utilisateurs de bases de connaissance à résoudre les conflits qu'ils génèrent lors de l'assertion d'une connaissance inconsistante dans leur base. Nous allons donc, dans ce chapitre, aborder la problématique générale des bases de connaissance.

Dans un premier temps, nous parlerons de représentation de connaissances [Masini91], et plus particulièrement des modèles à objets. Nous introduirons ensuite le système « Troeps », qui est le système sur lequel nous avons travaillé. Enfin, nous verrons plus en détail le problème de la révision et présenterons le protocole spécifique aux représentations par objets sur lequel le travail présenté ici s'est appuyé.

II.1. Les bases de connaissances

Le but de la représentation de connaissance est de rendre compte d'un « domaine » particulier, de telle sorte que cette représentation soit manipulable par une machine. Elle sert en particulier dans les systèmes à base de connaissances. Ceux-ci offrent des langages permettant de modéliser le « domaine ». Les constructions de ces langages ayant un sens particulier, elles peuvent être manipulées par un ordinateur en respectant ce sens, et ainsi permettre de préserver la cohérence interne de la modélisation du domaine.

Avec chaque langage de représentation de connaissance, on définit une sémantique de ce langage. Celle-ci permet de justifier la validité des opérations du système de représentation de connaissance par rapport au sens des expressions introduites dans ce système.

Le système « Troeps », sur lequel ce travail a été réalisé, s'appuie sur un modèle de représentation de connaissance par objets. Nous allons donc dans un premier temps voir brièvement l'histoire des représentations de connaissance par objets. Ceci nous permettra d'introduire dans un deuxième temps les principes de telles représentations.

II.1.1. Histoire

La représentation de connaissance par objets s'attache à définir des modèles généraux de représentation de connaissance, c'est-à-dire des modèles pouvant être appliqués à toutes sortes d'entités (objets physiques, idées, actions, relations...) par opposition aux modèles qui se concentrent sur un domaine particulier (relations temporelles, tâches...). Historiquement, ceux-ci ont été nombreux.

L'un des premiers systèmes de représentation générique de connaissance est le modèle des réseaux sémantiques. Il se caractérise par une organisation de la connaissance sous forme de graphe orientés dont les noeuds et les arcs sont étiquetés. Il est doté d'un mécanisme d'inférence : la propagation de marqueurs, qui consiste à parcourir le graphe dans le sens des arcs à partir d'un noeud précis (ou de deux noeuds). Il permet donc de déduire un ensemble de noeuds accessibles à partir d'un ou de plusieurs noeuds. Il est limité car il ne distingue pas a priori d’étiquettes particulières alors qu’il serait utile de distinguer certains arcs (comme par exemple les arcs indiquant les relations de spécificité) ou certains noeuds (par exemple, pour distinguer les individus des espèces). De plus, la signification des arcs et des noeuds n’est pas précisée clairement.

Le modèle ultérieur est celui des schémas, ou frames, de Minsky introduit dans le cadre de la représentation de connaissance acquise par la vision. Ce modèle organise la connaissance autour de la notion de schéma, un schéma étant un nom auquel est associé un ensemble d’attributs ayant chacun un ensemble de facettes permettant de les caractériser et contenant des valeurs. Cette notion de facettes, inspirée de la métaphore des différentes faces sous lesquelles il est possible de voir un objet, a été utilisée à des fins multiples et variées telles que le codage de la position de l’objet par rapport à un observateur, d’un rôle joué par celui-ci, ou de différentes acceptations du nom de l’attribut. Le mécanisme d’inférence privilégié qui lui est associé est la mise en correspondance (ou appariement, ou matching), qui consiste à comparer deux schémas en fonction de leurs facettes et valeurs pour vérifier si celles-ci correspondent ou non.

Avec ces deux modèles, certains problèmes qu’il faut absolument trancher persistent :

- conceptuellement, rien ne distingue les individus des ensembles d’individus ; il faut donc faire attention à ne pas considérer la même entité tantôt comme l’un, et tantôt comme l’autre ;
- de même, certaines propriétés sont descriptives (les oiseaux doivent avoir des plumes), alors que d’autres sont typiques (typiquement, les oiseaux volent) ;
- enfin, il faut distinguer les entités pour lesquelles on donne une définition de celles dont on donne une description et des prototypes.

Un troisième modèle, inspiré des deux précédents, mais aussi des langages de programmation à objets voit le jour vers la fin des années 1970 : c’est celui des systèmes de représentation de connaissance par objets. Ceux-ci se caractérisent par l’adoption de la notion de classe d’objets, ce qui permet de différencier les instances (un objet particulier) des ensembles d’objets, par la définition d’un lien de généricité-spécificité fort (c’est-à-dire dont le sens est inclus dans le système), ainsi que par une rationalisation de l’utilisation des facettes, celles-ci étant conservées en nombre restreint (mais parfois extensible) à des fins de typage ou d’inférence des valeurs d’attributs.

Maintenant que nous avons vu les systèmes ayant mené à l’idée de la représentation de connaissance par objets, voyons comment celle-ci peut être réalisée.

II.1.2. Principe des représentations de connaissance par objets

Il n’existe pas à proprement parler de principes uniques, généraux et inflexibles des représentations de connaissance par objets. Certaines caractéristiques typiques sont cependant communes à tous ces systèmes. C’est ce que nous allons présenter ici.

La fonction d’un système de représentation de connaissance par objets est double :

- stocker et organiser la connaissance autour de la notion d’objet ;
- fournir des services inférentiels de bas niveau destinés à compléter l’information disponible.

On s'intéresse ici à la première partie, c'est-à-dire à la structure sur laquelle se fonde un système de représentation de connaissance par objets.

Celle-ci est très proche de celle d'un langage de programmation à objets. A l'instar des langages post-frames, les objets sont décrits à l'aide de classes organisées dans une hiérarchie de spécialisation. La connaissance est organisée autour de la notion d'objet.

Les objets sont des ensembles de couples attributs-valeurs associés à un identifiant. En général, cette identification se fait à l'aide d'un nom, celui-ci étant soit extérieur aux couples attributs-valeurs, soit la valeur d'un attribut particulier. La valeur d'un attribut peut être soit un objet, soit une valeur d'un type primitif du langage (par exemple, une chaîne de caractères ou un entier). Elle peut être connue ou inconnue. Souvent, les attributs peuvent contenir une collection (par exemple un ensemble) de valeurs.

Les objets sont regroupés en classes qui, à l'instar de celles des langages de programmation par objets, permettent de regrouper des traits communs à ces objets. Ces classes sont organisées par la relation de spécialisation (parfois nommée relation d'héritage ou de généricité). La représentation de connaissance par objets privilégie donc au moins ce lien particulier entre les objets. La spécialisation est une relation d'ordre. Dans certains systèmes, le graphe de la réduction transitive de cette relation peut être restreint à un arbre, contraint à être connexe ou à disposer d'une unique source (racine dans le cas d'un arbre). La sémantique de cette relation est celle de l'inclusion ensembliste : les individus appartenant à l'interprétation d'une classe doivent appartenir à celle de ses super-classes. On dispose donc d'un ensemble de sous-ensembles imbriqués (voir Figure 1 et Figure 2).

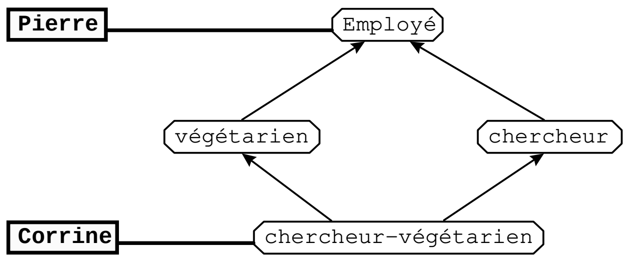


Figure 1 : Exemple de hiérarchie de classes instanciées

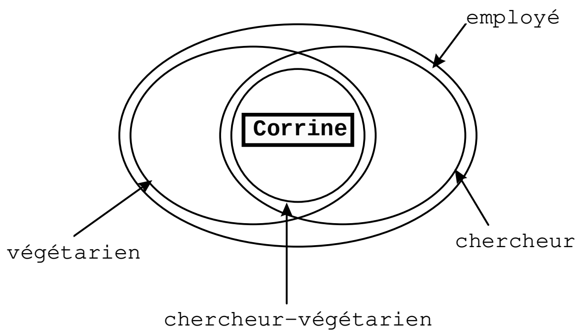


Figure 2 : Représentation ensembliste de la hiérarchie de classes de la Figure 1

De cette interprétation de la relation de spécialisation, tout découle :

- plus une classe est spécifique, plus les domaines des ses attributs doivent être restreints ;

- si une classe est sous-classe de deux classes, les domaines de ses attributs sont forcément inclus dans (ou égaux à) l'intersection des domaines de ces deux classes ;
- si un objet est instance d'une classe, il l'est aussi de toutes ses super-classes ;
- etc.

Ceci permet de faciliter l'expression du domaine de valeurs des attributs (que l'on qualifiera d'effectif) en ne précisant dans les sous-classes que les restrictions par rapport à la super-classe (on parlera alors de domaine exprimé). Par exemple, une contrainte sur le salaire des chercheurs s'appliquerait aussi au salaire des chercheurs-végétariens : elle ferait partie du domaine effectif de l'attribut salaire de la classe `chercheur-végétarien` mais il serait inutile de l'exprimer.

Dans ce cadre (celui de la structure), certains problèmes n'apparaissent pas. C'est le cas des problèmes de conflits dus à la multi-généralisation (aussi nommé héritage multiple ou multi-spécialisation), c'est-à-dire le fait d'avoir plusieurs sur-classes incomparables par la relation de spécialisation. C'est aussi le cas des problèmes de non monotonie traités par l'intelligence artificielle. Ainsi, si l'utilisateur exprime que les `végétariens` sont (c'est-à-dire ont pour valeur de l'attribut `moyens`) `pauvres`, que les `chercheurs` sont `riches` et qu'il existe des individus à la fois `chercheurs` et `végétariens`, il s'est tout simplement trompé quelque part.

Après cet exposé général sur les représentations de connaissance à objets, nous allons nous intéresser à un exemple spécifique : celui de Troeps, qui est le système à partir duquel le travail présenté ici a été réalisé.

II.2. Troeps

Troeps [Mariño90] est donc un système de représentation de connaissance par objets. Par rapport à d'autres systèmes de ce type, il peut se caractériser en quatre points :

- l'ensemble des objets est partitionné en concepts ;
- un concept peut être vu sous plusieurs points de vue ;
- chaque point de vue détermine une hiérarchie de classes ;
- les rapports d'inclusion entre classes de différents points de vue peuvent être exprimés à l'aide de passerelles entre ces classes.

Dans ce contexte, un objet est instance d'un seul concept et est visible sous plusieurs points de vue où il est attaché à une seule classe plus spécifique. La Figure 3 montre un exemple d'organisation de la connaissance dans ce système.

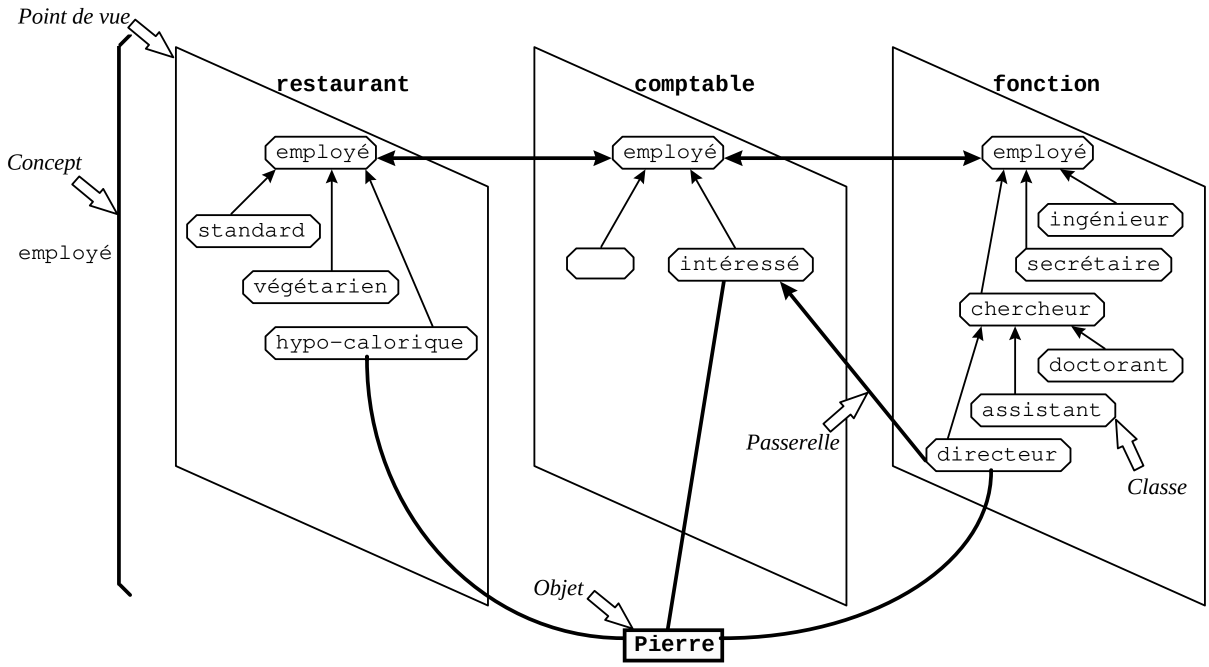


Figure 3 : Organisation de la connaissance dans Troeps

Nous allons voir maintenant les différentes structures de la représentation de la connaissance dans Troeps plus en détail [Sherpa95].

II.2.1. Le formalisme

Un des principes nouveaux dans Troeps est le concept. L’ensemble des objets d’une base est partitionné en un ensemble de concepts permettant de regrouper les objets ayant des caractéristiques communes (un objet fait partie d’un et un seul concept). Un concept rassemble donc l’aspect ontologique d’un ensemble d’objets, c’est-à-dire qu’il définit :

- la structure des objets : l’ensemble des attributs permettant de décrire les caractéristiques d’un objet sont définis comme des attributs de concept ; ces attributs sont typés,
- l’identité et l’intégrité des objets : un sous-ensemble de ces attributs forme la clé qui doit être unique et permettre l’identification de l’objet,
- l’espace des noms d’attributs, d’objets et de points de vue.

Ainsi, un employé est doté des attributs `nom`, `prénom`, `date-naissance`, `salaire`, `projet`, `diplômes`, etc. Il est identifié par ses `nom`, `prénom` et `date-naissance` (voir Figure 4).

```
<employé
     cle = {
          <nom
               type = chaine ;
               constructeur = un ; >
          <prénom
               type = chaine ;
               constructeur = liste ; >
          <date-naissance
               type = date ;
               constructeur = un ; >
```

```
        } ;
        attributs = {
            <salaire
                type = entier ;
                constructeur = un ; >
            <diplômes
                type = diplôme ;
                constructeur = liste ; >
            <projet
                type = projet ;
                constructeur = un ; >
            <chef
                type = employé ;
                constructeur = un ;
            <subordonnés
                type = employé ;
                constructeur = ensemble ; >
            <bureau
                type = bureau ;
                constructeur = un ; >
        } ;
        contraintes = {
            !.chef = !.projet.chef
        } ;
    >
```

Figure 4 : Exemple de définition de concept

L'ensemble des attributs applicables aux instances du concept sont définis dans celui-ci. Il est possible d'ajouter des attributs de concept à n'importe quel moment. Les attributs définis dans le concept sont applicables à n'importe quel objet du concept, même si celui-ci est attaché à des classes qui ne les mentionnent pas. Chaque attribut est décrit au niveau du concept par son type et son constructeur (voir Figure 4). Le type d'un attribut est un concept ou un type externe au système Troeps. Le constructeur d'un attribut permet de préciser si la valeur doit être une valeur unique ou une collection de valeurs (un ensemble ou une liste par exemple). Ainsi, les quatre premiers attributs de la Figure 4 sont typés respectivement par une chaîne de caractères, une liste de chaînes de caractères, une date et un entier. Les deux attributs suivants sont typés respectivement par un objet du concept `projet` et une liste d'objets du concept `diplôme`.

Des contraintes peuvent aussi être attachées aux concepts de manière à exprimer les relations entre attributs. Ces relations sont exprimées par un prédicat (ici =) entre objets atteignables à partir de l'objet courant, celui-ci étant désigné par « ! ».

Un concept peut-être vu sous plusieurs points de vue. Un point de vue est une structure essentiellement destinée à accueillir une taxonomie de classes sur les objets du concept. Il décrit donc cette taxonomie en commençant par sa classe racine puis en y ajoutant des sous-classes. Les taxonomies sont de strictes hiérarchies (c'est-à-dire qu'elles n'autorisent pas la multi-généralisation). La présence de plusieurs taxonomies dans un même concept rend compte plus simplement de nombreuses situations dans lesquelles les langages de programmation à objets utilisent la multi-généralisation. Par exemple, la taxonomie décrite par la Figure 1 peut être modélisée dans Troeps sans multi-généralisation comme le montre la Figure 5.

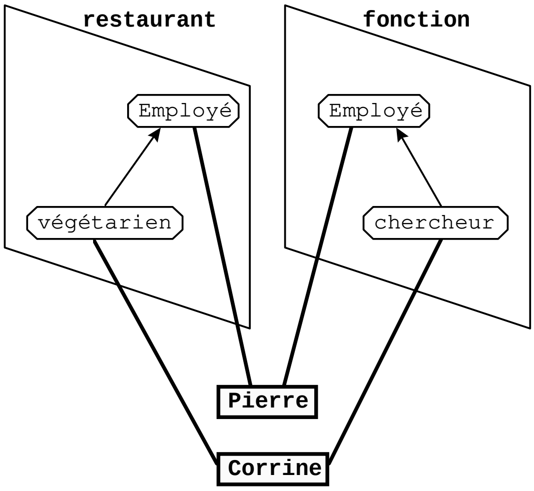


Figure 5 : La modélisation sans multi-généralisation grâce aux points de vue

Les attributs pertinents pour un point de vue sont ceux dont il est fait mention au sein des classes du point de vue. Un objet est attaché à exactement une classe par point de vue. Par conséquent, les taxonomies sont supposées exclusives (c'est-à-dire qu'un objet ne peut appartenir à des classes incomparables) et tous les objets du concept appartiennent à la racine de chaque point de vue, ce qui justifie que toutes les racines portent le nom du concept.

Les classes décrivent un ensemble d'objets particuliers. Pour cela, elles définissent l'ensemble des attributs du concept pertinents pour ces objets, et un ensemble de restrictions sur les types de ces attributs. Cela ne signifie pas que les attributs non pertinents ne sont pas associés aux objets, mais que la classe est indépendante de ces attributs (par exemple, sous le point de vue `restaurant`, l'attribut `projet` n'est pas pertinent, ce qui ne signifie pas qu'un employé ne fasse pas aussi partie d'un projet).

Les classes sont définies en référence à leur super-classe. Toute classe peut préciser les types des attributs en introduisant des attributs de classes. Ceux-ci sont décrits par un ensemble de descripteurs (correspondant aux facettes des représentations de connaissance par objets). Les descripteurs de Troeps sont :

- classes (pour les objets) qui introduit un ensemble de classes auxquelles les valeurs de l'attribut doivent appartenir (ces classes obéissent à la syntaxe `<classe>@<point de vue>`) ;
- `intervalles` (pour les types ordonnés) qui introduit une réunion d'intervalles de valeurs acceptables ;
- `domaine` qui introduit un ensemble de valeurs acceptables ;
- `sauf` qui introduit un ensemble de valeurs non acceptables pour l'attribut ;
- `cardinal` (pour les collections) qui introduit un intervalle de cardinaux acceptables pour la valeur de l'attribut.

Un exemple de description de classes est donné à la Figure 6.

```
<employé
     attributs = {
          <nom>,
          <prénom>,
          <date-naissance>,
```

```
            <poste
                 intervalles = {[1 266]} ; >
       } ;
  >

  <cadre
       sorte-de = employé@fonctionnel ;
       attributs = {
            <diplômes
                 cardinal = [2 +inf[ ; >,
            <chef
                 type = {cadre@fonction} ; >,
            <poste
                 intervalles = {[1 72]} ;
                 sauf = {1, 42, 53, 54} ; >,
            <salaire
                 intervalles = {[10000 20000]} ; >
       } ;
  >
```

Figure 6 : Exemple de définition de classes

Les passerelles permettent d'exprimer des contraintes entre les classes de points de vue différents d'un même concept. Ainsi, si les points de vue sont théoriquement indépendants, il est possible que l'appartenance à une classe sous un point de vue implique l'appartenance à une autre classe sous un autre point de vue. Par exemple, la passerelle entre `directeur` et `intéressé` dans la Figure 3 page 11 indique que les employés qui sont directeurs de recherche sont forcément intéressés aux bénéfices.

Enfin, les objets sont semblables aux instances des langages de programmation par objets. Ce sont des ensembles de couples attribut-valeur. Ils sont identifiés par le concept dont ils sont instance et un sous-ensemble de leurs attributs (défini pour chaque concept) formant la clé. Par ailleurs, les objets de Troeps, s'ils appartiennent à un et un seul concept, sont attachés à une et une seule classe par point de vue. Les contraintes qui pèsent sur un objet sont donc celles du concept et celles qui concernent toutes les classes auxquelles l'objet appartient. Enfin, tous les objets peuvent être manipulés pour changer leurs valeurs d'attributs ou leurs classes d'appartenance.

Nous avons donc vu comment la connaissance pouvait être modélisée avec le formalisme défini dans « Troeps ». Le problème qui se pose maintenant est celui de la gestion des bases de connaissances décrites avec ce formalisme. C'est l'objet du paragraphe suivant.

II.2.2. Architecture

Un des buts des bases de connaissance est de modéliser la connaissance d'un domaine particulier assez large qui ne peut être dominé par une seule personne. Dans ce cas, l'utilisation des bases de connaissance est donc partagée. Le problème est de permettre le partage de cette connaissance tout en faisant en sorte que chacun puisse confronter ses hypothèses avec le contenu de la base. En effet, prenons l'exemple d'une base de connaissance partagée par un ensemble de chercheurs en génétique. Un utilisateur de la base peut avoir mené une série d'expériences à partir desquelles il formule une hypothèse. Il peut vouloir introduire ces nouveaux éléments dans la base de connaissance pour les confronter au contenu de la base, même s'il a besoin de mener des expériences complémentaires pour faire partager cette nouvelle connaissance. Pour gérer ce problème, une organisation particulière des bases de connaissance a été définie. Elle est gérée par un protocole appelé Co4 (pour « collaborative construction of consensual knowledge »). C'est cette organisation que nous présentons brièvement ici.

Chaque utilisateur a sa propre base de connaissance, sur laquelle il est libre de faire les modifications qu'il veut. Il adhère à une base globale consensuelle, sur laquelle il peut proposer des modifications (ajout d'une nouvelle connaissance, avec une éventuelle révision). Toutes les modifications sur la base partagée doivent être acceptées à l'unanimité. Cette structure est récursive (voir Figure 7).

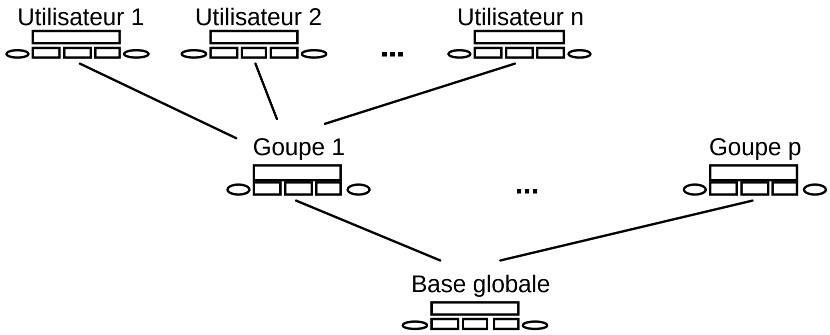


Figure 7 : Architecture de Troeps

II.2.3. Un éditeur de bases de connaissances

Un autre problème fondamental des systèmes à base de connaissance est l'édition. Dans Troeps un éditeur de bases de connaissances sur le World-wide Web gérant la manipulation des bases de connaissances a été développé [Alemany98]. Il permet la navigation dans la base, la classification, ainsi que la recherche de connaissances dans la base grâce à des filtres. Il rend la connaissance accessible à tous à moindre coût grâce au World-wide Web. De plus, la connaissance exprimée dans la base peut ainsi être reliée à toutes les autres formes de connaissance (bibliographie, schémas, pages HTML, ...).

Avec cet outil, se posent des problèmes liés à l'édition, dont le maintien de la consistance de la base, les problèmes d'accès concurrents et la gestion des copies multiples sont les principaux. Le problème de la consistance sera traité au paragraphe suivant. Pour les problèmes d'accès concurrents, comme on l'a vu précédemment, chaque utilisateur a sa base personnelle. Il est le seul à y avoir accès. Il peut donc réaliser l'ensemble des modifications qu'il souhaite sur celle-ci. Pour ce qui concerne l'ajout d'une connaissance dans une base partagée, un protocole a été développé [Euzenat96]. Celui-ci demande à l'ensemble des utilisateurs partageant la base à modifier s'ils sont d'accord pour introduire la nouvelle connaissance dans la base. C'est à cette seule condition que la connaissance est ajoutée. La base globale reste donc consensuelle. De plus, le protocole implémente une gestion des copies multiples classique en systèmes distribués.

Nous avons donc vu le principe général des représentations de connaissance. Nous avons ensuite présenté les caractéristiques du système sur lequel l'ensemble de ce travail a été réalisé. Nous allons maintenant aborder une partie plus centrale dans le problème que nous avons traité, c'est-à-dire le problème de la consistance d'une base de connaissance. Nous présenterons de plus les différents algorithmes de révision.

II.3. Consistance et révision

Une base de connaissance permet de stocker de manière structurée des informations disponibles dans un domaine donné. En tant que reflet d'une réalité, elle doit être consistante, ce qui est une condition nécessaire pour utiliser en toute confiance son contenu.

Cependant, la construction d'une base de connaissance, comme celle de beaucoup d'artefacts, n'est ni simple ni immédiate. Elle requiert un processus par essais et erreurs qui nécessite des remises en cause d'une partie du contenu de la base. La construction est donc incrémentale : de la connaissance s'ajoute à la base au fur et à mesure de son obtention (par exemple à la suite de séries d'observations ou d'expériences, ou après de nouvelles formalisations). Le problème de l'inconsistance de la connaissance apparaît dans divers contextes : lorsque la connaissance du domaine évolue ou lorsque différents acteurs interviennent, successivement ou simultanément, dans la construction de la base. C'est le cas en particulier lorsque des intervenants différents cherchent à mettre au point de manière coopérative une base de connaissance consensuelle (voir paragraphe précédent) [Euzenat95].

Ainsi, lorsque l'inconsistance est détectée, l'utilisateur se trouve confronté à deux interrogations :

- quelles sont les informations contradictoires ?
- que faudrait-il modifier afin de pouvoir réaliser l'assertion ?

Ces questions restent sans réponse pour un utilisateur ne pouvant appréhender la totalité de la base. Un outil permettant de guider l'utilisateur dans sa correction d'erreurs a donc été développé [Crampé97]. Cet outil lui permet de trouver les diverses modifications possibles garantissant la consistance de la base dont la connaissance a augmenté (voir Figure 8).

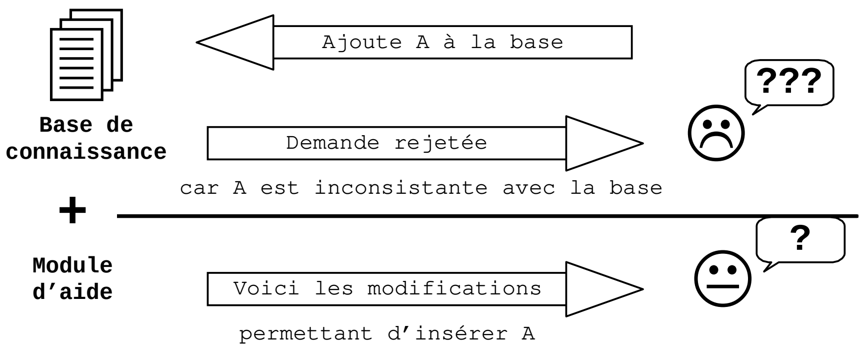


Figure 8 : Principe du module de révision

Ceci nécessite trois étapes :

- détecter l'inconsistance éventuelle de la base résultant de l'ajout de la nouvelle connaissance ;
- trouver les causes possibles de l'erreur ;
- trouver les solutions permettant d'y remédier, c'est-à-dire les modifications qui conduisent à une base consistante tout en réalisant la dernière assertion.

Historiquement, ce problème de révision a été d'abord traité en logique. Nous allons donc voir ce qu'est la révision logique, puis nous aborderons la révision spécifique aux objets.

II.3.1. La révision logique

Le travail de révision en logique s'inscrit dans une perspective plus large de modification de théories. La modification la plus connue est l'expansion, qui consiste en l'ajout d'une nouvelle proposition, consistante, dans la théorie. Cependant, d'autres types de modifications sont possibles, en particulier la contraction, qui consiste en le rejet d'une proposition de la théorie, et la révision, qui consiste en l'ajout d'une nouvelle proposition à une théorie A, après modification de cette dernière lorsque la nouvelle proposition est inconsistante avec le contenu de A. Le problème posé par la révision logique est donc de modifier automatiquement la théorie de façon à insérer la nouvelle proposition tout en gardant la consistance de celle-ci.

Pour cela, Gärdenfors [Alchourrón83] a développé des postulats permettant d'exprimer les propriétés de ces processus. Les notations qu'il utilise sont présentées maintenant. L'addition de la propriété x à la théorie A est notée « A + x ». L'opération sera donc réalisée à chaque fois que x est inconsistant avec A. On définit de plus une conséquence Cn comme une opération de l'ensemble des propositions vers l'ensemble des propositions qui satisfasse les trois conditions suivantes :

- pour tout ensemble X de propositions, $X \subseteq Cn(X)$
- pour tout ensemble X de propositions, $Cn(X) = Cn\ (Cn(X))$
- pour tout ensemble X et Y de propositions, $X \subseteq Y \Rightarrow Cn(X) \subseteq Cn(Y)$.

Pour simplifier la notation, on écrira Cn(x) à la place de Cn({x}) quand x est une proposition. De même, $y \in Cn(X)$ pourra remplacer $X \vdash y$. Une théorie est un ensemble A de propositions qui est fermé pour Cn, c'est-à-dire que A = Cn(A), ou il existe B sous-ensemble des proposition de A tel que A = Cn(B). On admet que Cn comprend les tautologies classiques, est compact (c'est-à-dire que si $y \in Cn(X)$, alors il existe X' fini tel que $X' \subseteq X$ et $y \in Cn(X')$), et satisfait la règle d'introduction de disjonctions dans les prémisses (c'est-à-dire si $y \in Cn(X \cup \{x1\})$ et $y \in Cn(X \cup \{x2\})$, alors $y \in Cn(X \cup \{x1 \vee x2\})$). On dit qu'un ensemble X de propositions est consistant (modulo Cn) si pour aucune proposition y, $y \in Cn(X)$ et $\neg y \in Cn(X)$.

On définit d'abord l'opération de contraction (-) d'une théorie. Soit $A \perp x$ l'ensemble des sous-ensembles maximaux B de A tels que x n'est pas conséquence logique de B. Soit A un ensemble de propositions et γ une fonction telle que, pour toute proposition x, $\gamma(A \perp x)$ est un sous-ensemble non vide de $A \perp x$ si celui-ci n'est pas vide, et $\gamma(A \perp x) = \{A\}$ si $A \perp x$ est vide. γ est appelée fonction de sélection pour A. On définit alors l'opération de contraction par $A - X = \cap\ \gamma(A \perp x)$ sur l'ensemble des x. L'idée intuitive est que la fonction de sélection γ retiens les éléments de $A \perp x$ les « plus importants ». Ensuite, la contraction A - x contient les propositions qui sont communes à tous les éléments sélectionnés par $A \perp x$. La révision est définie par rapport à la contraction par $A + x = Cn((A - \neg x) \cup \{x\})$. Il est à noter que les opérations A - x et A + x dépendent toutes les deux du choix de la fonction γ.Il en est de même pour l'opérateur conséquence Cn.

Les postulats de la contraction sont les suivants :

1. si a est une théorie, alors A - x est une théorie (clôture)
2. A - x $\subseteq$ A (inclusion)
3. si x $\notin$ Cn(A), alors A - x = A (vacuité)
4. si x $\notin$ Cn($\emptyset$), alors x $\notin$ Cn(A - x) (succès)
5. si Cn(x) = Cn(y), alors A - x = A - y (préservation)
6. si A est une théorie, alors A $\subseteq$ Cn((A - x) $\cup$ {x})

La signification de ces postulats est la suivante. La théorie contractée doit être fermée par déduction (1). La théorie contractée est la théorie de départ à laquelle on a retiré des propositions (2). Si l'information à retirer est inconsistante, alors elle ne se trouve pas dans la théorie, et donc la théorie contractée est la théorie de départ elle-même (3). Si la proposition à supprimer est inconsistante, la contraction se réduit au retrait de la proposition et à la fermeture par déduction (4). La contraction de la théorie doit être indépendante de la forme syntaxique de la suppression (5). L'opération de contraction n'est pas réversible.

Les postulats de la révision sont les suivants :

1. A + x est toujours une théorie
2. x $\in$ A + x
3. si $\neg$x $\notin$ Cn(A), alors A + x = Cn(A $\cup$ {x})
4. si $\neg$x $\notin$ Cn($\emptyset$), alors A + x est consistant sous Cn
5. si Cn(x) = Cn(y), alors A + x = A + y
6. si A est une théorie, alors(A + x) $\cap$ A = A - $\neg$x

La signification des postulats pour la révision est la suivante. La théorie révisée doit être fermée par déduction (1). La nouvelle information est prioritaire : elle doit appartenir à la théorie révisée (2). Si l'ajout n'est pas inconsistant, réviser se réduit à ajouter et à fermer par déduction (3). La condition suffisante pour obtenir une théorie révisée est que l'ajout ne soit pas faux en lui-même (4). La révision de la théorie doit être indépendante de la forme syntaxique de l'ajout (5). Les éléments retirés dans la théorie révisée sont exactement la théorie contractée avec la négation de l'assertion (6).

Les définitions données précédemment satisfont ces postulats (démonstration dans [Alchourrón83]).

Certains travaux ont essayé d'adapter ces principes dans le cadre des logiques terminologiques [Nebel90] [Nebel94], mais leurs techniques n'ont pas pu être implémentées. Nous allons maintenant voir comment ce modèle est adaptable dans le cadre de la révision sur les bases de connaissances à objets.

II.3.2. Révision sur les bases de connaissances à objets

Le problème de la révision sur les bases de connaissance à objets a beaucoup de points communs avec celui de la révision logique. Cependant, son cadre est plus contraint :

- les bases de connaissances sont exprimées dans un formalisme de représentation par objets, ce qui va permettre de diriger la recherche des solutions possibles ;

- le système se borne à aider l'utilisateur en lui proposant les diverses solutions, mais ne cherche pas à résoudre automatiquement les problèmes.

Pour définir un algorithme de révision sur les représentations de connaissance par objets, il faut d'abord définir un modèle de ces représentations. Nous allons donc maintenant étudier ce modèle. Ceci permet d'une part de définir formellement ce que sont les objets, et d'autre part de faciliter la transposition de l'algorithme de la révision logique [Crampé96] [Crampé98].

Dans un premier temps, un langage de représentation de connaissance par objets a été défini. Ce langage est une abstraction de Troeps. De plus, il peut sembler simple. Cependant, il recouvre une partie importante des systèmes de représentation par objets existant. Ce langage est présenté maintenant.

Ce langage contient évidemment des objets et des classes. Il a des attributs, dont les types sont définis dans les classes et les valeurs dans les objets. Ces valeurs d'attributs peuvent être soit des objets, soit des valeurs de types de base (des entiers par exemple). De plus, on a une relation d'ordre sur les classes : c'est la notion de sous-classe définie dans le paragraphe II.1.2. La grammaire de ce langage est la suivante :

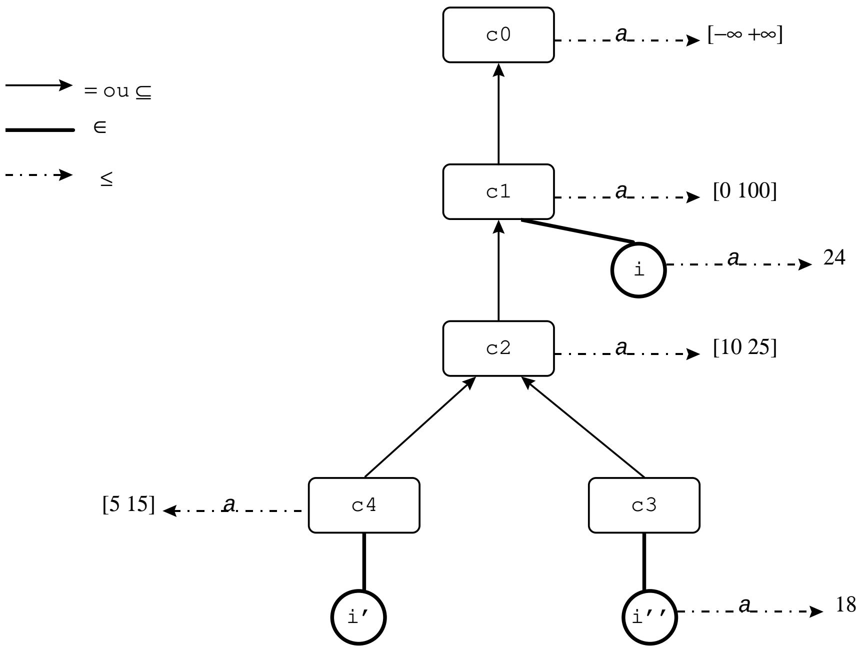
```
<BC>          ::=  {<assertion>*}

<assertion>   ::=  <spec>
                |  <attachment>
                |  <class_slot>
                |  <inst_slot>

<spec>        ::=  <class_name> ≤ <class_name>

<attachment>  ::=  <inst_name> ∈ <class_name>

<class_slot>  ::=  <class_name>.<slot_name> ⊆ <domain>
                |  <class_name>.<slot_name> ⊆ <class_name>

<inst_slot>   ::=  <inst_name>.<slot_name> = <value>
                |  <inst_name>.<slot_name> = <inst_name>

<class_name>  ::=  <identifier>

<inst_name>   ::=  <identifier>

<slot_name>   ::=  <identifier>

<identifier> sont des chaînes de caractères alphanumériques

<value> sont des entiers

<domain> sont des intervalles
```

Figure 9 : Grammaire du langage de représentation de connaissance par objets défini pour la révision

En plus de cette définition, il existe des contraintes méta-syntaxiques :

- une base de connaissance est un ensemble fini d'assertions ;
- le graphe de la relation $\subseteq$, appelé restriction d'attributs (et composé avec la relation $\leq$), est acyclique.

Enfin, les classes et les instances peuvent être introduites plus d'une fois dans le cadre de la multi-spécialisation (plusieurs super-classes pour une seule classe) et de la multiinstanciation (un objet est attaché à plusieurs classes). Un exemple de base de connaissance définie avec ce langage est présenté à la Figure 10.



B = {c1≤c0 ; c2≤c1 ; c3≤c2 ; c4≤c2 ; i∈c1 ; i'∈c3 ; i''∈c4 ;
c0⊆[−∞ +∞] ; c1.a⊆[0 100] ; c2.a⊆[10 25] ; c4.a⊆[5 15] ;
i.a=24 ; i'.a=18}

Figure 10 : Exemple de base de connaissance exprimée avec le langage de représentation de connaissance par objets défini pour la révision

Il faut maintenant définir la sémantique du langage qui vient d'être introduit. C'est ce que nous présentons maintenant. Cette sémantique est donnée en référence au domaine d'interprétation. Les objets sont interprétés comme des éléments de ce domaine. Les classes sont des sous-ensembles de celui-ci. La relation de spécialisation est interprétée comme la relation de sous-ensemble, et l'attachement à une classe comme l'appartenance à un ensemble.

A partir de cette syntaxe et de cette sémantique, on définit une interprétation des bases. Soient B une base de connaissance, O son ensemble d'objets, C son ensemble de classes, A son ensemble de noms d'attributs, τ l'ensemble possible des valeurs d'attributs, et D un domaine. Une interprétation est une paire <D, I> dans laquelle I est appelée fonction d'interprétation et est définie de la manière suivante :

I :	$O \rightarrow D$	injectivité
	$C \rightarrow 2^D$	
	$A \rightarrow (D \rightarrow \tau \cup D)$	I(a) est totale

L'injectivité de I sur O assure que les objets ont un nom unique.

On définit ensuite la satisfaction (notée |=) d'une assertion δ dans une interprétation <D, I> de la manière suivante :

$\models_{<D,I>}\delta$ si

$\models_{<D,I>} c \leq c'$	$\Leftrightarrow$	$I(c) \subseteq I(c')$
$\models_{<D,I>} c.a \subseteq d$	$\Leftrightarrow$	$I(a \mid c) \subseteq I(d)$
$\models_{<D,I>} o \in c$	$\Leftrightarrow$	$I(o) \in I(c)$
$\models_{<D,I>} o.a = v$	$\Leftrightarrow$	$I(a)(I(o)) = I(v)$
$\models_{<D,I>} c.a \subseteq c'$	$\Leftrightarrow$	$I(a \mid c) \subseteq I(c')$
$\models_{<D,I>} o.a = o'$	$\Leftrightarrow$	$I(a)(I(o)) = I(o')$

où $I(a \mid c)$ est le domaine de l'interprétation de l'attribut « a » quand le co-domaine est restreint à c.

Suivant les définitions classiques en logique, un modèle de base de connaissance est une interprétation qui satisfait toutes les assertions, une conséquence de la base est une assertion qui satisfait tous les modèles et une base est consistante si elle admet au moins un modèle (et inconsistante si elle n'en admet pas).

De plus on définit une procédure de déduction sur une base par un ensemble de règles d'inférence. Celle-ci est utile pour la révision dans la mesure où une inconsistance peut être levée par une inférence sur une base et où un ensemble de règles de déduction peut aider la révision par utilisation de l'abduction. La clôture Cn(B) d'une base de connaissances B par les règles de déduction est l'ensemble des assertions que l'on peut obtenir en appliquant les règles de déduction. Une assertion δ est alors dite déductible d'une base B (notée $B \vdash \delta$) si et seulement si $\delta \in Cn(B)$. On a les propriétés suivantes :

- $B \vdash \delta \Rightarrow B \models \delta$
- si B est consistante, $B \models \delta \Rightarrow B \vdash \delta$

On peut donc maintenant définir syntaxiquement une inconsistance. Une base de connaissance B sera dite inconsistante si et seulement si une des trois propositions suivantes est vérifiée :

1. $(o.a = x) \in B$ et $(o.a = x') \in B$, avec$x \neq x'$
2. $B \vdash o \in c$, $B \vdash c.a \subseteq d$ et $(o.a = v) \in B$, avec $\neg v{:}d$
3. il existe I tel que $\forall i \in I$, $B \vdash o \in c_i$, $(c_i.a_1.a_2....a_n \subseteq^* d_i) \in B$ et $\wedge_{i \in I}\, d_i = \emptyset$,
   où $c.a_1.a_2....a_n \subseteq^* d$ signifie « il existe $c_1, ..., c_n, c'_1, ..., c'_n$ telles que :
   $\forall\ 1 \leq j \leq n$, $(c_j \leq c'_j) \in B$ et $(c'_j.a_j \subseteq c'_{j+1}) \in B$
   avec $c = c_1$ et $(c'_n.a_n \subseteq d) \in B$

La complexité de détection des inconsistances dans une base de connaissance est polynomiale.

Comme en logique, on définit les notions de base de connaissance contractée et de base de connaissance révisée. Soient B une base de connaissance et δ une assertion telle que $B \cup \{\delta\}$ soit inconsistante. B' est une base de connaissance contractée de (B, δ) si et seulement si $Cn(B') \subset Cn(B)$ (i.e. $B' \not\models B$ et $B \models B'$) et $B' \cup \{\delta\}$ est consistante. B' est une base révisée de (B, δ) si et seulement si $B' = B'' \cup \{\delta\}$ avec B'' une base contractée de (B, δ).

Nous allons maintenant voir un exemple de révision. Soit B la base de connaissance suivante : $B = \{c1 \leq c0$; $c2 \leq c1$; $c3 \leq c2$; $c4 \leq c2$; $i \in c3$; $c0.a \subseteq [-\infty\ +\infty]$; $c1.a \subseteq [0\ 100]$;

c2.a⊆[10 25] ; c4.a⊆[5 15] ; i.a=50}. Cette base de connaissance est représentée à la Figure 11.

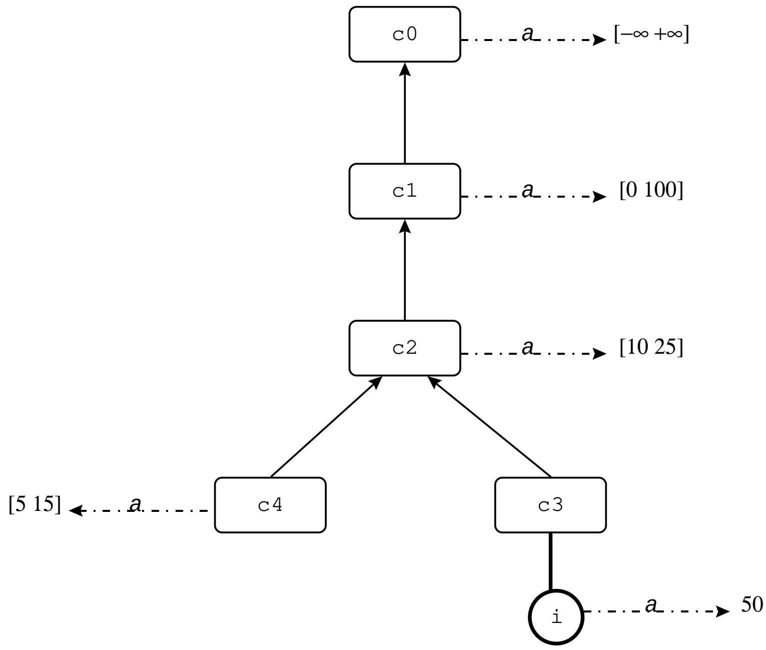


Figure 11 : Exemple de base de connaissance inconsistante

La base B est inconsistante car l'attribut « a » de l'objet « i » est affecté à la valeur 50 alors que cet objet est rattaché à la classe « c3 » pour laquelle l'attribut « a » doit être compris dans l'intervalle [10 25] (par héritage de la classe c2). Ceci correspond au deuxième cas d'inconsistance présenté précédemment car :

- $B \vdash i \in c3$
- $B \vdash c3.a \subseteq [10\ 25]$
- $(i.a = 50) \in B$
- $\neg(50 \in [10\ 25])$

Il existe plusieurs façons de réviser la base (car il existe plusieurs bases contractées possibles). En effet, on peut :

- supprimer la valeur de l'attribut « a » de l'objet « i » ;
- déplacer l'instance « i » sous la classe « c0 » ;
- déplacer l'instance « i » sous la classe « c1 » ;
- déplacer la classe « c3 » sous la classe « c0 » ;
- déplacer la classe « c3 » sous la classe « c1 »
- agrandir le domaine de l'attribut « a » dans la classe « c3 » en agrandissant ce domaine dans la classe « c2 » ;
- déplacer l'instance « i » sous la classe « c2 » et agrandir le domaine de l'attribut « a » de la classe « c2 » ;
- déplacer la classe « c2 » sous la classe « c0 » et agrandir le domaine de l'attribut « a » de celle-ci.

La Figure 12 montre quelques bases révisées possibles.

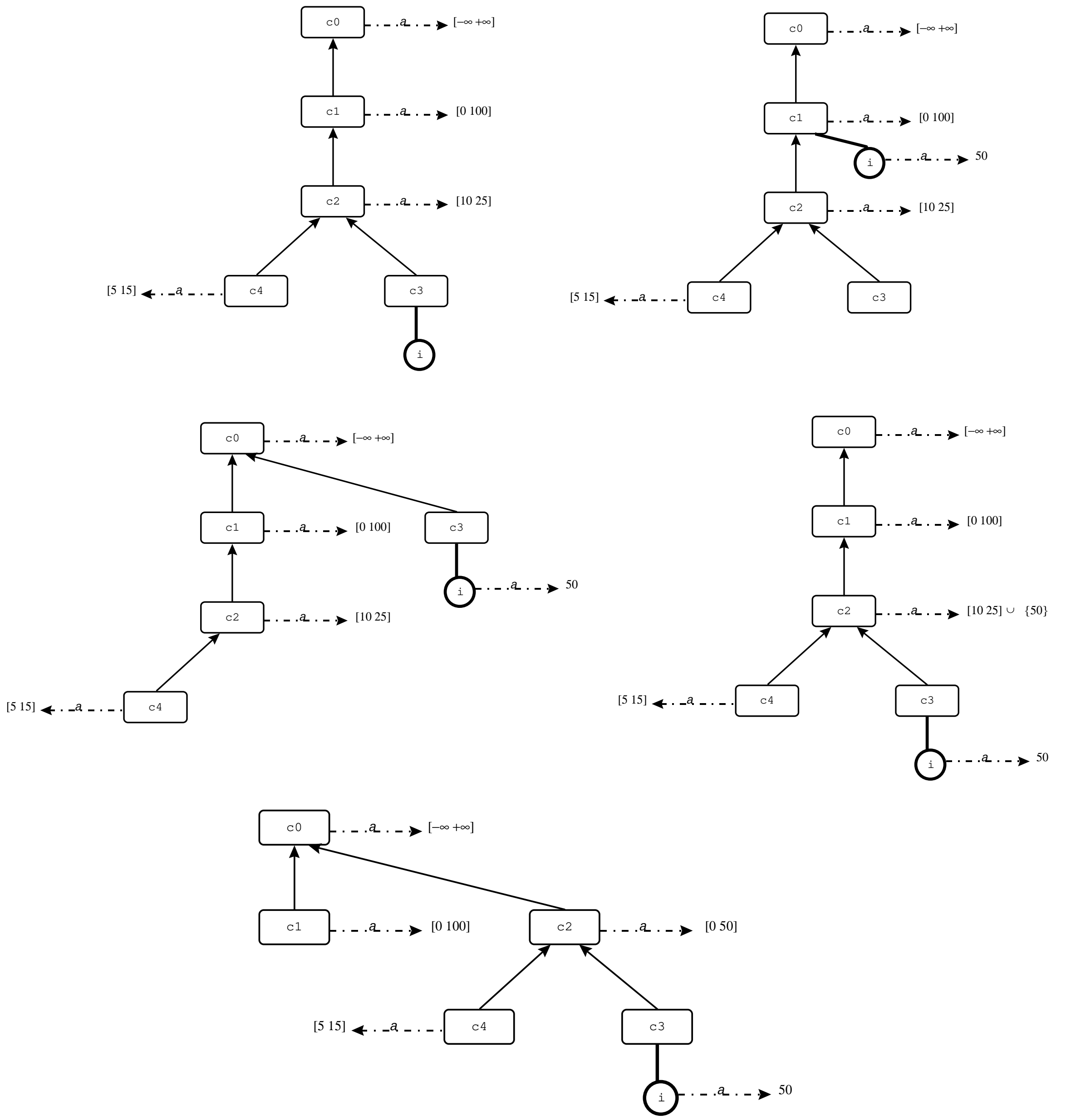


Figure 12 : Bases révisées possibles

En fait, la révision peut être interprétée en termes d’objets et non en termes logiques. Ainsi, il n’existe que quatre modifications élémentaires possibles pour résoudre une inconsistance :

1. déplacer un objet dans la hiérarchie des classes
2. supprimer la valeur de l’attribut d’un objet
3. élargir les restrictions d’un attribut (en élargissant le domaine de celui-ci ou en utilisant une classe plus générale)
4. déplacer une classe vers le haut dans la hiérarchie.

Nous verrons plus en détail dans le paragraphe IV.1 comment fonctionne l’algorithme de révision pour Troeps.

Toutes les bases révisées ne sont pas équivalentes en ce sens qu’un ensemble de modifications peut être inclus dans un autre. Dans l’exemple précédent, pour agrandir le domaine de l'attribut « a » d'une classe, on peut soit ajouter seulement la valeur « 50 » qui pose problème, soit agrandir l'intervalle de façon à ce que « 50 » soit inclus dedans. La première modification est donc incluse dans la deuxième. Parmi les différentes bases révisées, on peut ainsi établir un ordre partiel (noté $\propto$) qui permet de fournir une (ou un ensemble de) base(s) révisée(s) la (les) plus proche(s) possible(s) de la base de départ. Cet ordre est basé sur la relation de conséquence (Cn) définie précédemment. Sa définition est la suivante : soient B' et B'' deux bases contractées de (B, δ) ; alors B'' $\propto$ B' si et seulement si une des deux propositions suivantes est satisfaite :

- $B' \models B''$
- $Cn(B'') \subseteq Cn(B')$

On peut ainsi parler de minimalité de la révision, cette notion étant exclusivement basée sur la conservation d'un maximum de déductions. La contraction minimale d'une base est donc définie comme suit : B' est la base contractée minimale de (B, δ) si et seulement si B' est une base contractée de (B, δ) et il n'existe pas de base B'' telle que B'' ≠ B', B'' est une base contractée de (B, δ) et B'' $\propto$ B'.

L'implémentation de la révision dans Troeps permet de proposer à l'utilisateur, selon les principes définis ci-dessus, l'ensemble des modifications minimales possibles sur la base inconsistante. Cependant, c'est l'utilisateur qui choisit quelles modifications sont effectivement réalisées car le choix dépend du domaine modélisé, et donc ne peut être automatisé.

II.4. Conclusion

Dans ce chapitre, nous avons donc abordé le problème de la représentation de connaissance. Nous avons ensuite décrit le système Troeps sur lequel est basé le travail présenté ici. Cela nous a amené à aborder plus en détail le problème de la révision.

Nous avons vu que l'algorithme développé sur les représentations de connaissance par objets proposait plusieurs solutions pour réviser une base de connaissance. Or notre but est d'aider les utilisateurs de bases de connaissance à choisir ensemble la meilleure de ces solutions. Nous allons donc aborder maintenant le problème du travail collaboratif. Ceci va nous permettre d'introduire et d'analyser les différents modèles permettant de gérer la discussion entre les différents utilisateurs d'une base de connaissance permettant de mener à ce choix.

III. Le travail collaboratif : CSCW

La communauté de chercheurs regroupée sous le terme « CSCW » (Computer Supported Collaborative Work) s'intéresse à la coopération entre personnes via une machine [Lubich95] [CSCW96]. Elle regroupe à la fois des informaticiens et des psychologues, des sociologues... Ses thèmes de recherche sont aussi variés que :

- l'utilisation de collecticiels,
- l'étude du comportement de l'homme en groupe ou au sein d'organisations,
- le développement de méthodes de développement et d'évaluation des collecticiels,
- le développement de nouvelles techniques de communication telles que le courrier électronique ou la visioconférence,
- la mise au point de nouvelles technologies au niveau hardware.

Une partie de cette communauté s'intéresse plus particulièrement aux problèmes liés à l'argumentation [Shum]. En effet, le développement de protocoles permettant de gérer l'argumentation ouvre à la fois des perspectives quant à l'éducation (comment apprendre aux élèves à penser de façon critique ? Quels sont les types d'arguments utilisés dans tel ou tel domaine ?...) et quant à la gestion des différentes connaissances lors de la résolution de conflit. C'est dans ce dernier type de perspective que se place le travail réalisé. Deux types de travaux nous ont paru intéressants pour celui-ci. Le premier consiste à modéliser la structure de l'argumentation elle-même. Nous présenterons donc dans une première partie trois modèles permettant cette structuration. Le second consiste à modéliser la structure des communications entre personnes lors de la réalisation d'une tâche précise. C'est ce que nous verrons dans une deuxième partie.

III.1. Structure de l'argumentation

La réflexion sur les systèmes d'argumentation a été essentiellement menée autour du problème de la capture du raisonnement pendant la conception d'un nouveau produit. Trois systèmes d'argumentation sont maintenant présentés, en suivant l'historique de leur mise au point.

III.1.1. gIBIS

Une première solution à ce type de problèmes a été apportée par gIBIS [Conklin88]. gIBIS est un système basé sur la méthode proposée dans IBIS (Issue-Based Information System) [Kunz70]. Celle-ci a été développée dans le but d'aider à la conception de systèmes complexes. Elle est basée sur le principe selon lequel ce processus de conception est fondamentalement une conversation entre les différents intervenants dans la conception, à savoir, les concepteurs eux-mêmes, les clients, et les codeurs, chacun exprimant son point de vue quant à la résolution des différents problèmes rencontrés. Chaque problème ou question rencontré(e) peut être considéré(e) comme une « issue », et demander discussion, si ce n'est consensus, pour que la conception puisse avancer. En fait, dans le modèle IBIS, c'est cette argumentation qui constitue le processus de conception. Ce modèle fut développé dans les années 70 et utilisé avec succès dans diverses situations de conception telles que la conception architecturale, celle de plannings pour villes, ou pour l'Organisation mondiale de la santé.

Le modèle IBIS se concentre sur l'articulation de problèmes clés dans la conception, appelés « Issues ». Chaque Issue peut avoir plusieurs « Positions », une Position étant une affirmation permettant de résoudre le problème posé par l'Issue à laquelle il est rattaché. Souvent, les différentes Positions sont mutuellement exclusives. Cependant, le modèle ne l'impose pas. A son tour, chaque Position de chaque Issue peut avoir un ou plusieurs « Arguments », qui la supportent ou lui font objection. Ainsi, chaque Issue est la racine d'un arbre éventuellement vide, les enfants des Issues étant des Positions et les enfants des Positions étant des Arguments.

Il y a neuf types de liens entre les différents types de noeuds du modèle IBIS:

- une Issue peut généraliser, spécialiser, remplacer, questionner ou être suggérée par une autre ;
- une Issue peut questionner ou être suggérée par une Position ;
- une Issue peut questionner ou être suggérée par un Argument ;
- une Position peut répondre à une Issue ;
- un Argument peut appuyer ou s'opposer à une Position.

Les trois types de noeuds, ainsi que les différents types de liens qui sont susceptibles de les unir sont représentés à la Figure 13.

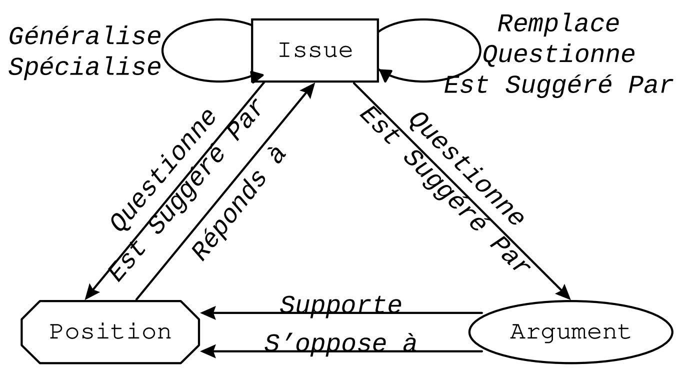


Figure 13 : Structure de l'argumentation utilisée dans gIBIS

Voyons maintenant un exemple de discussion utilisant le modèle IBIS. Imaginons le début d'une discussion concernant la conception d'un guichet bancaire. Un premier intervenant, A, se demande quels types de services le guichet devrait offrir. Il poste donc l'Issue « Combien de services offre-t-on ? ». Il pense que le mieux est d'avoir un ensemble de services. Il poste donc la Position « Tout un ensemble », ainsi que l'argument « Variété des services » supportant cette Position. Un deuxième intervenant, B, pense que ce qui est important est la rapidité avec laquelle l'opération de retrait d'argent liquide est effectuée. Pour lui, il vaut donc mieux que le guichet automatique ne permette que le retrait d'argent liquide. Il poste donc la Position « Seulement le retrait d'argent liquide » et l'Argument « rapidité » la supportant. Il ajoute ensuite un lien « S'oppose à » de son Argument vers la première Position. A peut alors ajouter un lien « S'oppose à » de son Argument vers la nouvelle Position. Un troisième intervenant, C, se demande s'il vaut mieux retirer l'argent et le ticket au même endroit ou avoir deux endroits différents. Il poste donc un nouvelle Issue « Où retire-t-on le liquide et le ticket ? », ainsi que deux Positions : « A des endroits différents » et « Au même endroit ». B peut alors indiquer que son Argument « Rapidité » « S'oppose à » la première et « Supporte » la seconde. Quant à A, il peut alors poster un nouvel Argument « Coût réduit » exprimant qu'il est plus facile de séparer les deux sorties. Cet Argument « Supporte » la première Position alors qu'elle « S'oppose » à la seconde. La discussion ainsi amorcée continue de la même façon. Un exemple de graphe résultant d'une telle discussion est donné à la Figure 14.

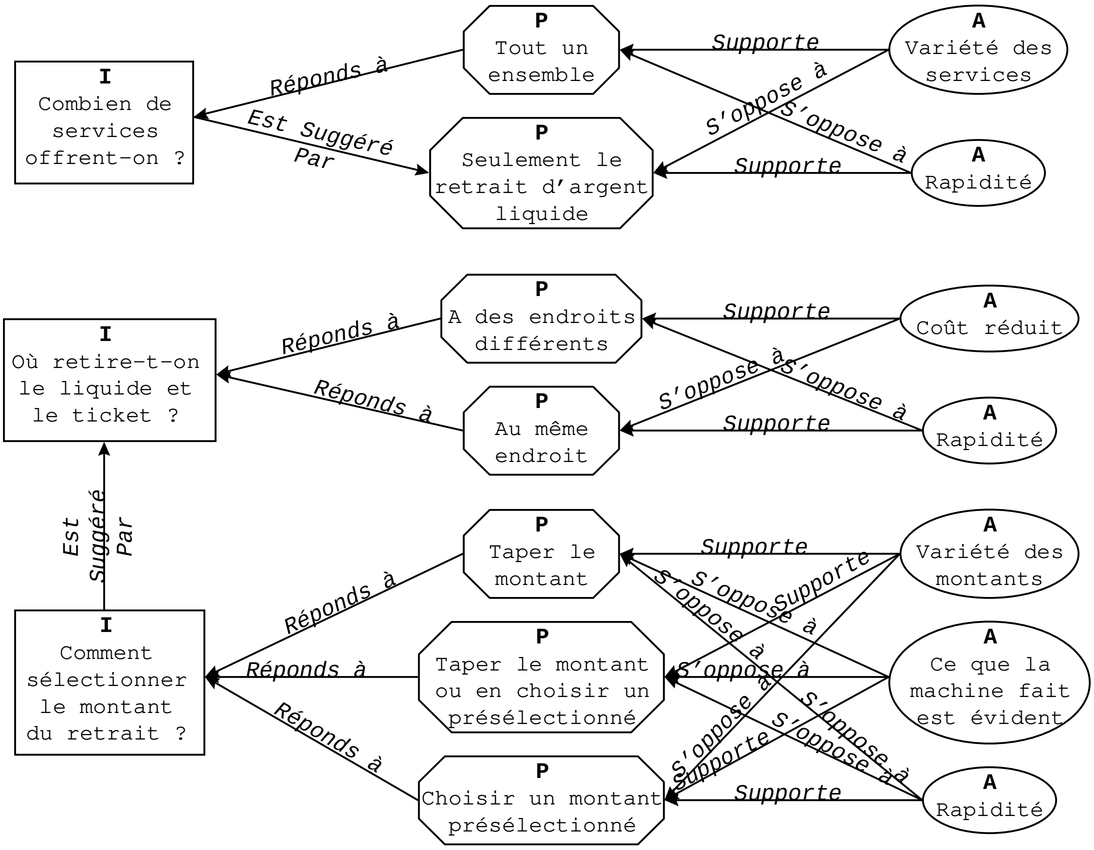


Figure 14 : Graphe d'argumentation obtenu avec le modèle IBIS

IBIS peut être considéré comme un système d'aide à la décision, mais d'un type inhabituel. En effet, on n'y trouve ni règle d'arrêt, ni moyen pour enregistrer qu'une Position particulière a été choisie pour une Issue donnée. Le but de la discussion est plutôt que chaque intervenant dans la discussion essaie de comprendre les éléments spécifiques de chaque proposition, et peut-être de persuader les autres que son point de vue est le meilleur. Cette méthode permet d'éviter aux intervenants les déviations vers des discussions trop rhétoriques, comme celles utilisant les « arguments par répétition », et les entraîne dans des discussions plus constructives, comme la recherche du problème central, la formulation de questions plutôt que de réponses. De plus, cette méthode oblige les intervenants à être précis dans la formulation des Arguments appuyant telle ou telle Position.

gIBIS (pour graphical IBIS) est une implémentation graphique du modèle IBIS. En plus des noeuds du modèle lui-même, gIBIS implémente un mécanisme d'échappement sous la forme de noeuds « Autre », qui peuvent être liés à n'importe quel noeud de l'argumentation par des liens de type « Autre ». Ce mécanisme permet aux utilisateurs n'arrivant pas à exprimer leur point de vue directement avec les noeuds du modèle de pouvoir quand même en faire part aux autres. De plus, il ajoute un type externe permettant de créer des noeuds qui ne contiennent pas des informations d'argumentation pure, mais des informations liées à celle-ci telles que des spécifications, des parties de code ou des tests. Enfin, des liens de généralisation ou de spécialisation entre Positions ou entre Arguments ont été ajoutés. Ces nouvelles fonctionnalités par rapport au modèle initial ont permis d'ajouter à ce dernier une flexibilité qui s'est avérée nécessaire. Cependant, comme le modèle avait déjà été utilisé pendant assez longtemps avant cette implémentation graphique, les changements ont été réduits au minimum.

Le but de cette implémentation était triple :

- explorer le domaine de la capture du raisonnement dans l'histoire d'une conception, c'est-à-dire les décisions, les options rejetées et l'analyse ayant mené aux différents choix ;
- explorer le problème du travail en équipe supporté par l'ordinateur, et en particulier découvrir le type de conversations apparaissant au cours de la conception et pouvant être supportées par un réseau d'ordinateurs ;
- explorer le problème de la navigation à travers de très larges espaces d'informations ; pour ce dernier problème, un outil hypertexte a été développé ; cet outil permet la navigation dans le graphe d'argumentation.

Cet outil a été utilisé de façon expérimentale pendant 1 an. Les utilisateurs l'ont utilisé soit en groupe pour concevoir de nouveaux produits, soit seuls pour clarifier leurs idées.

Le premier résultat de l'expérimentation ayant été menée est que, malgré l'apparente complexité des différents noeuds et liens du modèle, les personnes ayant utilisé l'outil ont trouvé que c'était une excellente méthode pour aider le cheminement de la pensée pendant la phase de conception, ainsi que pour délibérer et prendre une décision. Les utilisateurs qui travaillaient seuls ont reconnu que l'articulation entre Issues, Positions et Arguments les avaient aidés à se concentrer sur les parties difficiles et critiques des problèmes qu'ils essayaient de résoudre, et ainsi à détecter les aspects incomplets ou inconsistants de leur pensée plus rapidement. Ceci s'explique par la structure de gIBIS, qui oblige à bien différencier les problèmes des solutions possibles et des différents arguments pour ou contre telle ou telle solution, tout en n'imposant aucune contrainte sur la façon d'exprimer un problème, une position ou un argument. En effet, chaque noeud du graphe est relié par un lien hypertexte à une fiche permettant de décrire le noeud. Ces fiches contiennent en particulier le sujet, l'auteur, la date de création et un ensemble de mots-clés permettant de décrire le noeud, mais aussi un texte complètement libre dans lequel l'auteur du noeud peut exprimer plus en détail son point de vue. L'auteur du noeud peut donc s'exprimer sans contraintes d'expression alors que le lecteur du graphe est guidé par la structure de celui-ci.

Cependant, certains inconvénients sont apparus durant cette expérience :

- il n'y a pas de type de noeud (ou de lien) permettant d'exprimer le but de la discussion ainsi que les différentes contraintes qui y sont attachées ;
- il n'existe aucun support particulier permettant de prendre une décision parmi les différentes solutions proposées ;
- de la plupart des décisions découlent l'adjonction d'éléments de solution dans la conception elle-même (comme du code, une structure de module), mais ces éléments ne peuvent pas être pris en compte par le modèle et doivent être stockés

de façon externe par rapport à l'outil ; ceci implique que le produit d'une décision ne peut être relié à la discussion ayant mené à celui-ci ;

- les utilisateurs ont parfois eu du mal, devant un problème difficile, à structurer leur pensée en Issues, Positions et Arguments ; par exemple, il arrive que l'on veuille essayer quelque chose, sans vraiment voir le problème caché derrière ; or la méthode IBIS impose de penser en terme de problèmes, auxquels on propose plusieurs solutions que l'on soutient par des arguments ; un mode « brainstorming », où il serait possible de lancer des idées de façon un peu moins structurée, a donc été étudié.

Ce premier modèle a donc été critiqué, et deux autres modèles ont vu le jour : PHI (pour Procedural Hierarchy of Issues) et QOC (pour Questions, Options and Criteria). C'est ce que nous allons voir maintenant.

III.1.2. La méthode PHI

Cette méthode est née d'une réflexion critique par rapport à gIBIS et est présentée dans [Fisher91]. Ces travaux se placent encore dans la perspective d'amélioration du travail de conception.

Documenter les travaux de conception d'un nouvel outil par une argumentation structurée peut permettre d'améliorer ceux-ci. Cela peut permettre d'améliorer la maintenance de l'outil, une fois que celui-ci est créé, de faciliter les changements de conception et de favoriser la réutilisation. De plus, cela permet de promouvoir la réflexion critique pendant la phase de conception elle-même. Cependant, l'expérience montre que l'argumentation ne sert pas naturellement la conception. Il faut donc la concevoir pour qu'elle remplisse ces attentes. Le but est qu'elle aide les concepteurs :

- à améliorer leur travail,
- à coopérer avec d'autres personnes,
- à comprendre les éléments déjà existants.

Le travail sur IBIS se place dans cette perspective : il rejette les efforts de certains pour automatiser le raisonnement utilisé pendant la conception, et développe, comme on l'a vu précédemment, une approche argumentative qui essaie d'améliorer le raisonnement sous-tendant la conception et a pour but d'aider le raisonnement humain des concepteurs au lieu d'essayer de le remplacer par un raisonnement automatique. Cependant, il manque, dans IBIS, deux types d'informations mais qui sont nécessaires pour une approche basée sur les problèmes. Ce sont :

- des relations de dépendance entre la résolution des différents problèmes, c'est-à-dire le fait que la réponse à un problème dépend souvent des réponses apportées à un ensemble d'autres problèmes ;
- l'ensemble des questions dont on n'a pas encore délibéré, c'est-à-dire les questions pour lesquelles le pour et le contre des différentes alternatives n'est pas examiné ; IBIS ignore donc les questions qui trouvent naturellement une réponse et s'attache à celles pour lesquelles il y a débat et controverse ; cependant, de telles questions, non délibérées, apparaissent souvent dans le travail de conception et peuvent influencer la résolution des différents problèmes ; de plus, la plupart de ces questions ont des réponses qui dépendent de la résolution des problèmes de conception.

PHI a été développé pour s'affranchir de ces limitations. Comme IBIS, PHI est une méthode de conception plus qu'un logiciel. Il diffère d'IBIS suivant deux points importants :

- il utilise une définition plus stricte des « Issues » ;
- il définit un nouveau principe pour lier les Issues entre elles.

Dans IBIS, les Issues sont les questions qui se posent et sur lesquelles les concepteurs réfléchissent et débattent. Dans PHI, tous les problèmes comptent, qu'il y ait besoin d'une délibération à leur sujet ou que leurs solutions soient immédiates et ne nécessitent pas d'argumentation. De plus, PHI abandonne les liens entre Issues développés dans le modèle IBIS, et les remplace par un unique type de relation appelé « Sert ». On dit qu'une Issue A « Sert » une Issue B si et seulement si la résolution de A influence celle de B. La relation « Sert » la plus courante est la relation « Est une sous-Issue de », qui indique que la résolution d'un problème est une sous-tâche de la résolution d'un autre. De fait, alors qu'un graphe de type IBIS est très foisonnant, et que les Issues sont reliées entre elles par un ensemble de types de liens assez différents les uns des autres, un graphe d'argumentation de type PHI est une structure simple quasiment hiérarchique, ayant une unique racine. Cette structure ressemble à un arbre mais n'en est pas un dans la mesure ou plusieurs Issues peuvent avoir la même sous-Issue. La racine d'un graphe d'argumentation de type PHI représente le problème dans son ensemble. La Figure 15 montre un graphe de type PHI sur le problème déjà traité précédemment avec IBIS. Seuls les Issues ont été représentées dans la mesure où c'est à leur niveau que le modèle PHI diffère du modèle IBIS.

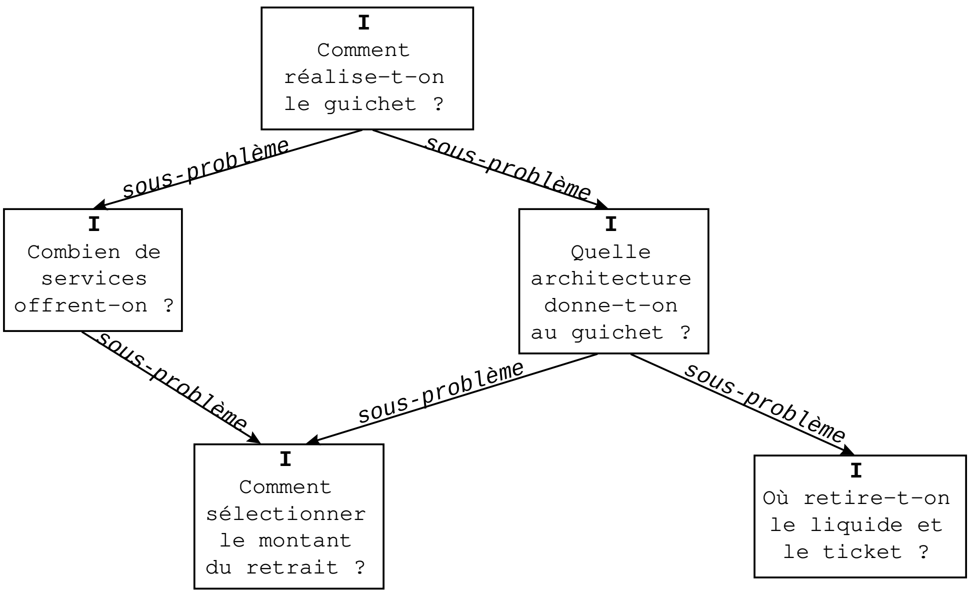


Figure 15 : Graphe d'argumentation obtenu avec le modèle PHI
(seules les Issues sont représentées)

Lors de tests par des utilisateurs, la méthode PHI s'est avérée très bonne pour la création d'un graphe d'argumentation utilisable. Par contre, en ce qui concerne le travail de conception en lui-même, cette méthode s'est avérée trop coûteuse par rapport aux avantages qu'elle pouvait donner sur le moment (voir par exemple [McCall91]).

Une troisième méthode d'argumentation, alternative aux deux précédentes, est présentée maintenant.

III.1.3. QOC

Alors que les deux précédents modèles étaient destinés à enregistrer l'historique des délibérations, QOC (pour « Questions, Options and Criteria ») [MacLean91] cherche plutôt à modéliser l'ensemble des possibilités de conception et à expliquer les raisons des choix qui ont été réalisés. Ainsi, une possibilité de conception est toujours placée dans le contexte des différentes possibilités offertes.

QOC est donc une notation très simple permettant de décrire les différentes possibilités s'offrant aux concepteurs. Elle définit trois types de noeuds principaux :

- les « Questions », qui représentent les problèmes clés permettant de structurer l'espace des diverses alternatives ;
- les « Options », qui constituent les différentes réponses que l'on peut apporter aux Questions ;
- les « Critères », qui sont les bases permettant d'évaluer et de faire un choix parmi les différentes Options.

L'argumentation développée par QOC doit être considérée comme un sous-produit du produit en cours de conception. En effet, les analyses ainsi documentées représentent elles-mêmes un produit dans la mesure où ce sont des représentations explicites qui doivent être créées par les concepteurs. De fait, la création de telles structures demande un effort supplémentaire aux concepteurs. Cependant, cet effort est payant dans la mesure où il permet d'expliquer les choix réalisés, et ainsi, d'avoir la possibilité d'opérer des évolutions après réalisation sans remettre en cause les choix fondamentaux de conception ayant été réalisés. De plus, pendant le travail de conception, un tel produit peut être utile pour aider le raisonnement, mais aussi la documentation du produit et la communication entre les différents intervenants dans la conception.

Nous allons maintenant voir comment, dans l'exemple déjà développé précédemment pour les modèles IBIS et PHI, la notation QOC peut être utilisée. Comme on l'a vu précédemment, les trois types de noeuds définis par QOC sont les « Questions », les « Options », et les « Critères ». Les Critères sont reliés aux Options par des liens du type de ceux développés par IBIS entre les Positions et les Arguments, c'est-à-dire qu'il y a deux types de liens :

- les liens positifs, entre un Critère et une Option qui satisfait celui-ci ;
- les liens négatifs, entre un Critère et une Option qui ne satisfait pas celui-ci.

Chaque Critère est relié à toutes les Options pour une Question donnée, mais des ensembles différents de Critères sont appliqués à chaque Question. Cependant, souvent, un certain nombre de critères sont valables pour un ensemble de Questions. De plus, lorsqu'une Option est choisie, elle est marquée comme telle sur le graphe d'argumentation. Cela permet, en utilisant des notations différentes, de marquer plusieurs Options suivant les Critères qui paraissent les plus importants, et ainsi de faire des comparaisons entre les différentes réalisations possibles. La Figure 16 montre le graphe résultant de l'analyse, avec QOC, de la conception d'un guichet bancaire. Deux ensembles d'Options peuvent être retenues suivant si le Critère le plus important est la rapidité du service que l'on veut offrir ou si c'est la variété des services que l'on veut offrir.

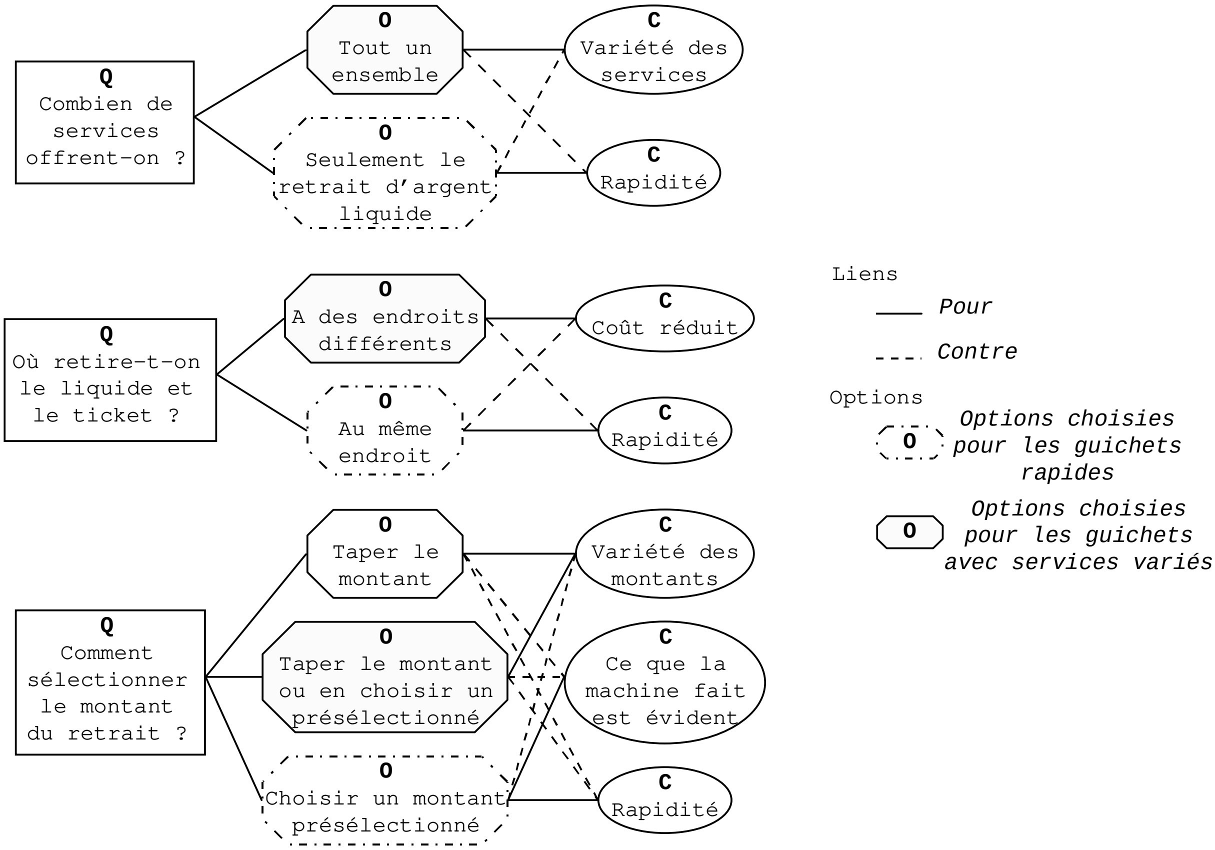


Figure 16 : Type de graphe obtenu avec la notation QOC

QOC permet donc de définir les différentes solutions et de proposer des critères d'évaluation de celles-ci. Cependant, telle quelle, elle ne prend pas vraiment en compte l'argumentation. Il est en fait possible d'ajouter à ce graphe un ensemble d'arguments permettant de débattre l'ensemble des Critères à retenir pour réaliser le choix des meilleures Options. Les noeuds « Arguments » sont reliés aux liens entre Critères et Options par des liens de type « Pour » ou « Contre ». Par exemple, dans la Figure 17, l'Argument 1 soutient le choix de l'Option 1 par rapport à l'Option 2 suivant le Critère 1. L'Argument 2 est contre le fait que l'Option 2 est meilleure suivant le Critère 1. De plus, les Arguments peuvent être reliés entre eux par les mêmes types de liens. Par exemple, dans la Figure 17, l'Argument 3 soutient l'Argument 1 alors que l'Argument 4 s'oppose à celui-ci.

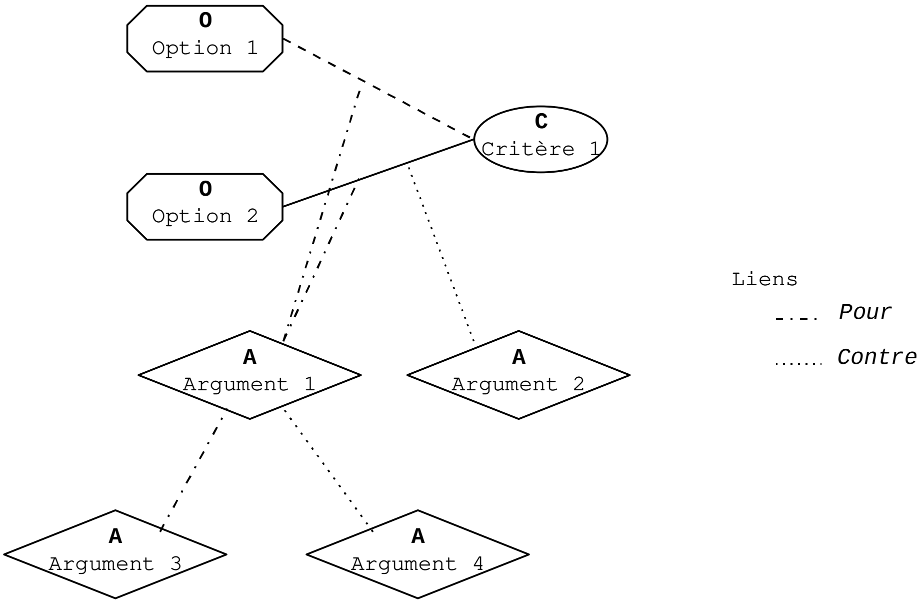


Figure 17 : L'utilisation d'arguments dans QOC

Cette méthode permet donc de clarifier les buts qui vont guider l'ensemble de la conception. Elle a été testée dans diverses conditions aux cours des années qui ont suivi sa définition [Shum&] [Shum91]. Cependant, elle n'a jamais été implémentée sur un système informatique du type de gIBIS.

III.1.4. Conclusion

Nous avons donc vu les différentes méthodes possibles pour définir une argumentation structurée face à un problème de conception. C'est en effet dans ce cadre-là que ce type de travaux a été effectué. Ces méthodes définissent un nombre restreint de noeuds et de liens entre ces noeuds permettant de décrire l'ensemble du raisonnement et/ou de l'argumentation mis en jeu lors de la conception d'un nouvel objet.

Nous allons maintenant voir une autre approche du travail collaboratif, l'approche « Langage/Action », qui s'appuie sur la modélisation de l'ensemble des communications entre les intervenants dans la réalisation d'une tâche particulière.

III.2. L'approche Langage/Action

Ce travail s'inscrit dans la réflexion concernant la conception de nouveaux systèmes informatiques. Il s'appuie sur l'hypothèse suivante : le langage est la première dimension de toute activité humaine coopérative. Dans un premier temps, nous allons voir en quoi consiste cette nouvelle approche, puis nous étudierons un exemple.

L'approche Langage/Action [Winograd87] consiste à dire que « les personnes agissent à travers le langage ». Cette approche est donc en contraste avec l'approche classique s'appuyant sur l'hypothèse selon laquelle « les personnes gèrent un ensemble d'informations et prennent des décisions ». Bien sûr, toute personne dans une organisation fait les deux, mais le fait de raisonner par rapport à l'une ou à l'autre montre une différence de point de vue. Si l'on considère par exemple une infirmière appelant une pharmacie, demandant les médicaments disponibles et en choisissant un pour un de ses patients. Avec la seconde approche, on se concentrerait sur la base de données contenant l'ensemble des informations sur les médicaments, ainsi que sur les règles permettant de choisir le médicament adéquat. Dans l'approche Langage/Action, on se concentre plutôt sur l'acte de commander les médicaments ainsi que sur la forme des interactions entre infirmière et pharmacien au cours des conversations liées à cette commande. Avec d'autres approches, on aurait pu se concentrer sur des points tels que les relations personnelles entre l'infirmière et le pharmacien ou le rapport coût/efficacité résultant du fait de passer la commande par téléphone. Suivant l'approche choisie, la modélisation s'appuiera sur des modèles physiques, économiques... Dans le cas de l'approche Langage/Action, celle-ci s'appuie sur des théories du langage, mais pas de linguistique très spécialisée. Le but n'est pas de s'occuper en détail des problèmes posés par l'utilisation de langues naturelles, mais des problèmes de forme, sens, et utilisation qui sont communs à toute communication humaine. L'utilisation du terme « Langage », et non celui de « Communication », permet d'accentuer l'importance des symboles et de l'interprétation.

La théorie linguistique distingue trois niveaux d'analyse d'un langage :

- la syntaxe, qui représente la structure des formes visibles (ou audibles) du langage,
- la sémantique, qui représente la relation systématique entre les structures d'un langage et l'ensemble de leurs significations possibles,
- la pragmatique, qui traite des problèmes d'utilisation du langage.

L'ensemble des règles syntaxiques (ou grammaire) d'un langage détermine donc les éléments de base de ce langage (lettres, mots...) et les différentes manières de les combiner. Ce niveau d'analyse se distingue donc des autres en ce sens qu'il ne prend pas du tout en compte l'interprétation ou le sens. La sémantique définit chaque élément individuel du langage (c'est-à-dire chaque mot), ainsi que le sens généré quand on combine ceux-ci (par exemple, le fait que « Jacques voit Jean » est différent de « Jean voit Jacques »). La pragmatique, de même que la sémantique, traite des sens possibles d'une expression, mais s'attache aux possibles sens cachés de celles-ci. L'exemple classique est celui du maître disant à son serviteur « Il fait froid ici ». Bien que le sens littéral de cette phrase concerne la température ambiante, l'intention est de provoquer une action de la part du serviteur, comme par exemple celle de fermer la porte. L'intérêt de l'approche Langage/Action concerne cette dernière partie du langage, c'est-à-dire son rôle dans l'évocation et l'interprétation d'actions. Les linguistes modernes ont adopté une approche cognitive en formalisant la structure de la connaissance et des procédés mentaux mis en jeu par chaque utilisateur du langage. L'approche Langage/Action s'appuie plutôt sur les trois niveaux d'analyse définis précédemment. Elle considère en effet que les problèmes de sens sont très liés à des considérations de contexte et d'action, et que la syntaxe présente l'intérêt particulier de refléter les petites distinctions pouvant modifier le sens d'une conversation. Cependant, elle s'appuie essentiellement sur l'aspect pragmatique de ces différents points de vue, c'est-à-dire non pas sur la forme du langage, mais sur ce que les gens en font. Plus précisément, le point de départ de la réflexion à ce sujet est la théorie des actes de langage. C'est ce que nous allons voir maintenant.

Toutes les phrases d'un langage ne sont pas soit justes, soit fausses. Par exemple, l'affirmation « Je vous déclare mari et femme » est une action, qui est appropriée ou non, mais qui n'est ni vraie ni fausse. De même les actes de langage concernant le commandement, les questions et les excuses ne sont pas des descriptions d’un monde non linguistique. On distingue généralement cinq types fondamentaux d’actes de langage :

- assertif, qui engage celui qui les dit sur la véracité de la proposition dite ;
- directif, qui essaie de faire faire quelque chose à celui qui écoute (avec différents degrés) ; ce type comprend à la fois les questions (qui peuvent pousser celui qui écoute à faire une déclaration de type assertif en réponse) et les commandements (qui poussent celui qui écoute à faire un acte, linguistique ou non) ;
- commissif, qui engage celui qui les dit pour une action future ;
- déclaratif, qui permet de faire la correspondance entre le contenu propositionnel d’un acte de langage et la réalité (c’est l’exemple du mariage) ;
- expressif, qui exprime un état psychologique à propos d’une situation (par exemple l’excuse ou la prière).

Il faut noter trois points importants à ce sujet.

1. Tous ces actes de langage sont interprétés par celui qui parle et celui qui écoute dans un certain contexte. Par exemple, un commissive n’a pas besoin de comprendre les mots « Je promets » ou « Je vais faire... », mais peuvent être « un franc » ou « Je pense » si la question précédente était « Pouvez-vous me donner quelque chose ? », ou même être réduits à un geste de la tête ou une expression du visage. La différenciation entre les différents types dépend à la fois de celui qui parle et de celui qui écoute, et ainsi la compréhension mutuelle est toujours sujette aux différences d’interprétation. Par exemple, la phrase « C’est l’heure du dîner » peut être une assertive ou une directive suivant qui la prononce à qui et dans quelles circonstances.
2. Les directives et les comissives parlent toutes les deux d’une action future. La seule différence entre les deux est que pour l’une, l’action future concerne celui qui parle, alors que pour l’autre, elle concerne celui qui écoute.
3. Les actes de langage prennent effet lorsqu’il y a une déclaration publique, c’est-à-dire lorsque ce lui qui parle et celui qui écoute savent tous deux que l’acte a été fait. Ceci est particulièrement évident pour les déclaratives et les expressives : par exemple, une excuse murmurée et non entendue n’a aucune valeur. Cependant, c’est aussi vrai pour toutes les autres.

Les actes de langage ne sont pas isolés mais participent d’une conversation plus grande. Un exemple important est la « conversation pour action » dans laquelle une personne A demande à une personne B de réaliser une certaine tâche. La demande est comprise par chacun comme ayant un certain degré de satisfaction qui caractérise le cours des futures actions de B. La personne A amorce le dialogue en formulant une requête envers la personne B. B peut alors :

- accepter la requête : il fera le travail demandé ;
- faire une contre-proposition à A ;
- décliner la proposition et refuser de faire le travail.

Si B accepte la requête de A, il doit lui envoyer un rapport une fois qu’il a terminé le travail pour le lui dire. A peut alors refuser le rapport (et B doit recommencer) ou l’accepter. Il déclare alors le travail fini. Si B fait une contre-proposition à A, celui-ci peut soit l’accepter, soit la refuser et terminer la communication, soit faire une autre contre-proposition. A tout moment, A ou B peuvent clore la discussion. Le diagramme des communications possibles de cet exemple est montré à la Figure 18.

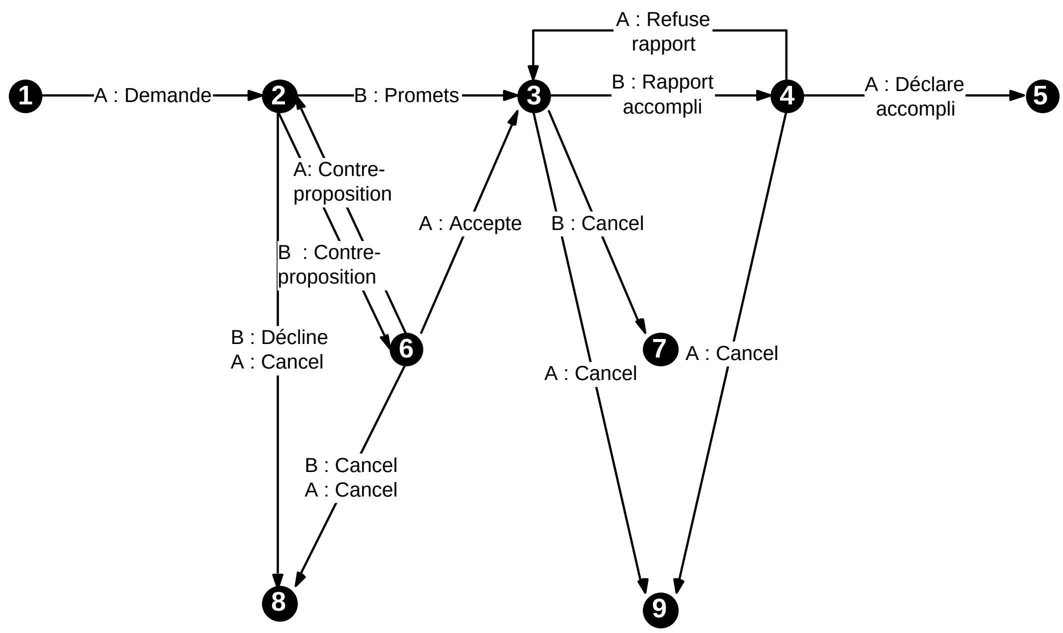


Figure 18 : Graphe Langage / Action initié par une requête

Ce diagramme n'est pas un modèle de l'état mental des intervenants dans la conversation. La logique présentée ne s'occupe que de la progression de l'action liée à la requête. D'autres possibilités, qui ne sont pas montrées sur ce diagramme, peuvent émerger dans la conversation. Par exemple, A peut dire « Pardon ? Je ne vous ai pas entendu », mais ceci n'a rien à voir avec l'action à réaliser.

Dans le diagramme présenté ici, la conversation est initiée par une demande. Un diagramme similaire peut être construit dans le cas d'une conversation initiée par une offre.

Quelques points intéressant de cette approche sont à souligner :

- Le mot « conversation » a été employé au sens large pour indiquer une séquence d'actes pouvant être interprétés comme ayant une signification. Ainsi, la conversation n'est pas forcément parlée. Par exemple, une personne remplissant une déclaration d'impôts est engagée dans une conversation avec l'administration fiscale, même s'il n'y a jamais de contact direct entre les deux. De même, certaines requêtes sont faites de manière implicite. Par exemple, un chef de service ne demande pas tous les matins à ses employés d'arriver à l'heure.
- A chaque point de la conversation, il existe un petit nombre de type d'actions possibles, l'ensemble de celles-ci étant déterminé par l'histoire de la conversation. Par contre, chaque type d'action a un nombre illimité de contenus détaillés possibles. Par exemple, une contre-proposition peut spécifier des conditions particulières de satisfaction.

Nous avons donc vu la perspective que cette approche ouvrait. Nous allons maintenant voir comment celle-ci a été utilisée pour implémenter « The Coordinator » [Flores88] qui est un système d'aide à la gestion d'une organisation.

Les deux approches précédemment suivies pour développer ce type de systèmes consistaient soit à définir une base de données accessible à tous, soit à définir des outils d’aide à la décision ou des systèmes experts. L’inconvénient de la première approche est qu’il est très difficile de ne récupérer que des documents pertinents. L’utilisateur devient très vite submergé par une masse d’informations secondaires. La deuxième approche se concentre sur l’évaluation et la recherche de solutions. Pour [Flores88], ce n’est pas la bonne solution, car l’action est gouvernée par le langage, et celui-ci fait référence à un contexte le plus souvent sous-entendu et que l’on ne peut pas implémenter. Il faut donc aider la communication mais laisser l’interprétation libre à l’utilisateur. Dans « The Coordinator », la structure de conversation pour proposer, accepter, refuser et/ou mener à terme une action, définie par l’approche Langage/Action et présentée à la Figure 18, est utilisée. Le diagramme est en fait compilé comme un protocole dans le logiciel. Les utilisateurs peuvent alors initier des conversations suivant ce protocole. L’interface ne leur permet alors que les actions autorisées par le protocole. Cette architecture a été implémentée en utilisant des mails préparés, ce qui structure la communication sans limiter le pouvoir d’expression des différents intervenants. Le logiciel connaît donc à tout moment l’ensemble des conversations en cours et peut dire à chaque utilisateur quelles sont les tâches sur lesquelles il s’est engagé, combien il a de requêtes qu’il n’a pas encore regardées... De plus, un système de datation a été implémenté, ce qui permet de limiter le temps d’attente d’une réponse, de mettre une échéance à la réalisation d’une tâche, et de gérer des systèmes d’alarmes.

III.3. Conclusion

Dans ce chapitre, nous avons abordé la problématique du travail collaboratif. Deux approches nous ont semblé intéressantes pour structurer la coopération entre les utilisateurs de bases de connaissance pendant le processus de révision :

- la modélisation de l’argumentation développée,
- la modélisation des communications mises en jeu dans le processus.

La première approche va nous permettre de structurer la réflexion des différents intervenants. Nous avons vu trois modèles pour réaliser cela. Nous analyserons donc les avantages et les inconvénients de chacun par rapport à notre problème, qui est la révision, et nous développerons un système hybride.

Notons ici que certains systèmes utilisent des règles pour pouvoir prendre des décisions de façon automatique. En particulier, une des idées consiste à donner une note aux différentes possibilités [Takayuki97]. Cependant, nous considérons d’une part que cette notation est très difficile, et d’autre part, qu’elle limite l’expression des différents intervenants. Nous avons donc préféré les approches de type IBIS ou Langage/Action. Il existe aussi des systèmes hybrides [Karacapilidis97].

La deuxième approche va nous permettre de structurer l’architecture de notre modèle.

Nous allons donc maintenant présenter le modèle que nous avons réalisé.

IV. Une argumentation basée sur la révision

Nous avons donc vu d'une part les modèles de connaissance à objets, ainsi qu'un système de révision propre à ceux-ci, et d'autre part les différents systèmes d'argumentation existants. Il s'agit maintenant d'analyser la structure de la révision du point de vue de l'argumentation, de choisir le système d'argumentation le plus pertinent et de l'adapter au problème de la révision.

IV.1. Le problème

Comme nous l'avons vu précédemment, la méthode de révision développée sur les objets permet de proposer à l'utilisateur les différentes solutions pour réviser la base. Cette révision peut être récursive. En effet, certaines actions de révision sont terminales. C'est-à-dire que la modification choisie suffit pour résoudre l'inconsistance. Par exemple, dans la Figure 19, la modification du domaine de l'attribut « a » de la classe « c1 » suffit pour résoudre l'inconsistance.

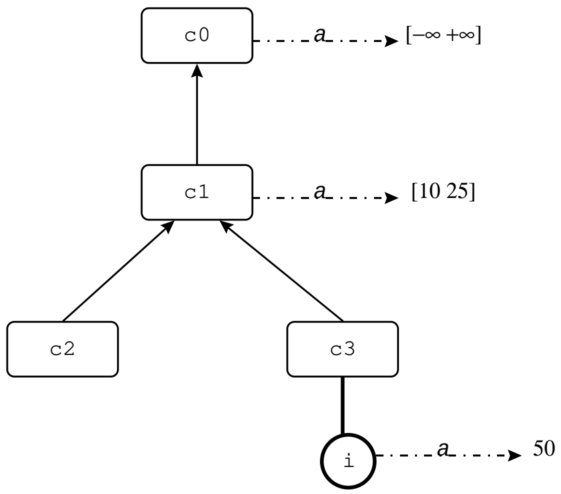


Figure 19 : Révision en une unique étape

Par contre, dans d'autres cas, il faut plusieurs modifications avant de reconstruire une base consistante. Par exemple, dans la Figure 20, l'inconsistance est liée au fait que la valeur de l'attribut « a0 » de l'objet « i'0 » est l'objet « i0 », qui est rattaché à la classe « c0 », alors que l'objet « i'0 » est rattaché à la classe « c'0 » pour laquelle la valeur de l'attribut « a0 » doit être un objet de la classe « c1 » (qui est une sous-classe de « c0 »). Pour lever l'inconsistance, on peut vouloir déplacer l'objet « i0 » sous la classe « c1 ». Cependant, pour pouvoir le faire, il faut d'abord élargir le domaine de l'attribut « a1 » de la classe « c1 » ou modifier la valeur de l'attribut « a1 » pour l'objet « i0 ». Il y a donc deux modifications à faire pour rendre la base consistante.

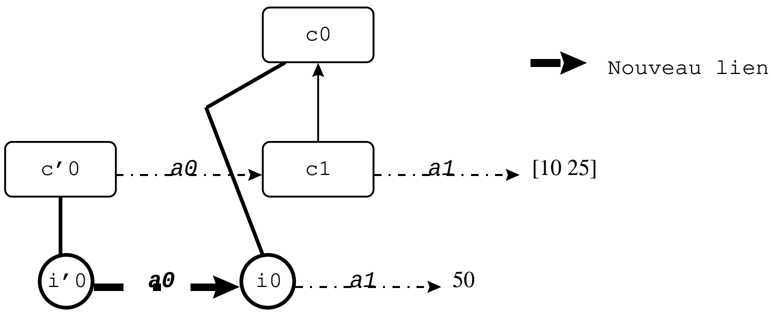


Figure 20 : Révision en plusieurs étapes

Le module de révision développé propose les modifications au fur et à mesure. Dans ce deuxième cas, par exemple, la révision propose, dans un premier temps, le choix entre :

- déplacer l'instance « i'0 »
- modifier chaque valeur d'attributs de l'instance « i'0 »
- faire des modifications sur les restrictions de domaine.

Si l'utilisateur choisit de modifier chaque valeur d'attribut, le module de révision propose alors de modifier l'objet « i0 » en modifiant la valeur de son attribut « a1 ». Il est alors seulement possible de proposer de déplacer « i0 » sous la classe « c1 ».

Plusieurs problèmes se posent pour l'utilisation de ce module. D'une part, l'utilisateur n'a pas une vue d'ensemble des solutions. En particulier, il ne sait pas ce qu'un choix implique : aura-t-il d'autres choix à faire après ? Cette modification est-elle la dernière ? Dans l'exemple précédent, lorsque les trois choix lui sont proposés, il ne sait pas si un de ceux-ci lui permet de revenir à une base consistante, ou si chacun impose une (ou plusieurs) modification(s) supplémentaire(s) après ce premier choix. D'autre part, ces bases de connaissance sont amenées à être partagées. Comme on l'a déjà souligné, cela implique qu'un utilisateur ne maîtrise pas l'ensemble de la connaissance contenue dans une base. Ainsi, il peut ne pas savoir quel choix faire. On cherche donc à le faire discuter de ce choix avec les autres utilisateurs de la base. Il va donc falloir que les différents intervenants confrontent leurs points de vue quant aux diverses possibilités pour réviser la base, c'est-à-dire parlent du contenu de la révision. Notre module d'argumentation doit donc à la fois s'appuyer sur le processus de révision en lui-même, et sur le contenu des idées qui y sont développées. Par rapport à ce problème, voyons comment utiliser le module de révision pour développer l'argumentation.

IV.2. Structure logique de la révision

Nous venons de voir que l'algorithme de révision proposait un ensemble de choix à chaque étape. Le déroulement de la révision peut donc être modélisé par un arbre dont les noeuds sont les possibilités offertes à chaque étape. La visualisation d'au moins une partie de cet arbre peut permettre à l'utilisateur d'avoir une vue d'ensemble du processus de révision, et ainsi faciliter son choix parmi les différentes solutions.

Nous avons donc choisi de nous appuyer sur cet arbre pour développer l'argumentation. En effet, cela permet aux utilisateurs d'avoir une vue d'ensemble du processus de révision en cours. De plus, cet arbre de révision peut être développé interactivement avec l’utilisateur. Ainsi, lorsque celui-ci sait qu’il ne fera pas telle ou telle modification proposée par le module de révision, ce n’est pas la peine de développer la branche de l’arbre correspondante. Il en résulte un gain de temps du fait de la complexité de la révision par rapport à afficher l’arbre complet, car cela reviendrait à explorer toutes les possibilités de révision au début de l’argumentation. De plus, cet arbre pouvant être assez complexe, le fait de ne développer que les parties utiles de l’arbre permet d’avoir plus de clarté que si l’arbre était complet, et de se concentrer sur les parties de la révision qui posent problème et dont il faut donc discuter. Ainsi, lorsque l’utilisateur hésite entre deux (ou plusieurs) possibilités, il peut avoir une vue d’ensemble des modifications impliquées par l’une ou par l’autre.

Un exemple d’arbre de révision sur la base inconsistante de la Figure 20 est donné à la Figure 21.

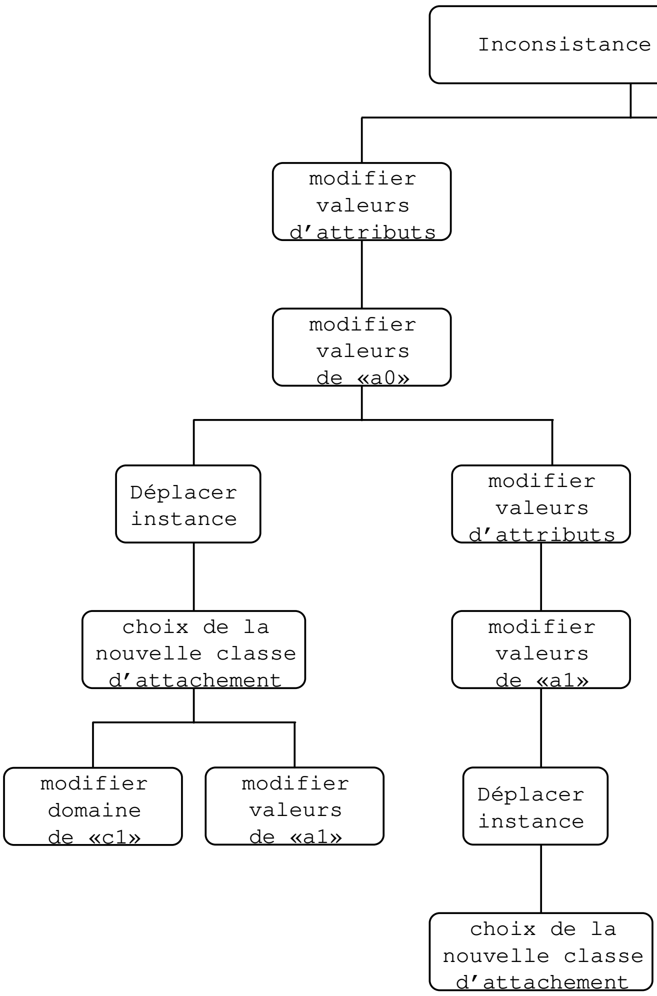


Figure 21 : Arbre de révision

Maintenant que nous avons vu comment nous appuyer sur la structure de la révision, nous allons voir quel système d’argumentation est le plus adapté à notre problème.

IV.3. Articulation Révision/Argumentation

Comme nous l'avons vu dans le paragraphe III.1, il existe trois modèles d'argumentation : IBIS, PHI et QOC. Ces modèles sont dédiés à la modélisation du raisonnement pendant la conception d'un nouvel objet (logiciel, maison, voiture...). Ils structurent tous les trois l'argumentation en problèmes, solutions possibles et arguments pour ou contre ces solutions. Dans les trois approches, les problèmes permettent de représenter les questions qui se posent pendant la conception et dont il faut délibérer. Dans le cas particulier de la révision, le problème à résoudre est toujours le même : « Quelle est la meilleure solution pour réviser la base parmi les différentes options proposées ? ». Ainsi, dans l'arbre de révision défini dans le paragraphe précédent, les problèmes sont déjà représentés par les lieux de séparation en différentes branches (voir Figure 22).

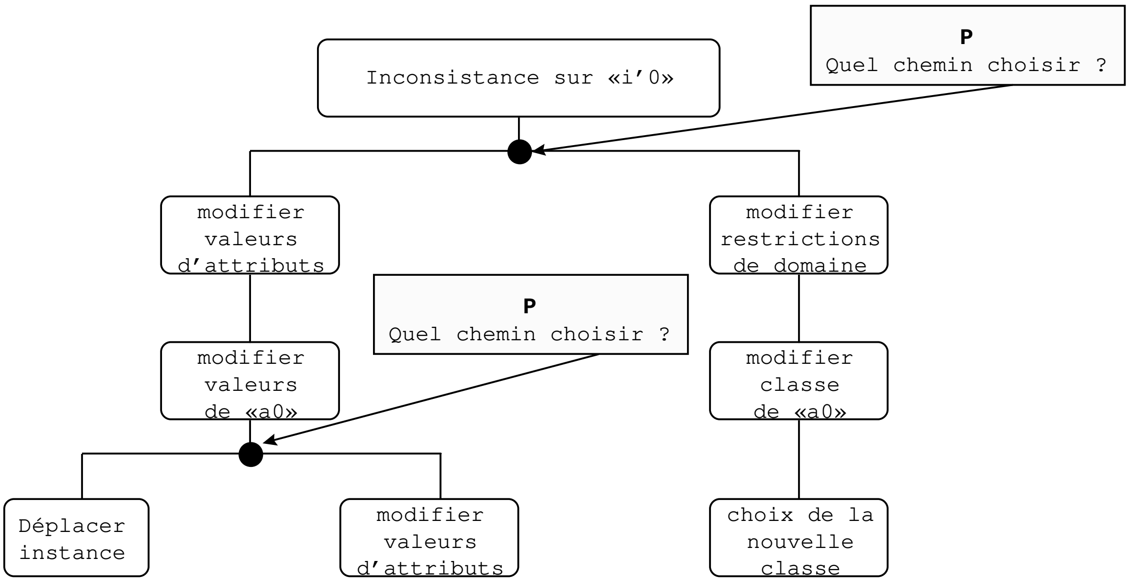


Figure 22 : Représentation des problèmes dans l'arbre de révision

De même, les options (ou positions) représentent les différentes possibilités de réponse au problème auquel elles sont rattachées. Dans le cas de la révision, ces différentes options sont fournies par le module de révision. Elles sont donc déjà présentes sur l'arbre de révision sous la forme des nœuds offrant les différentes possibilités de révision (voir Figure 23).

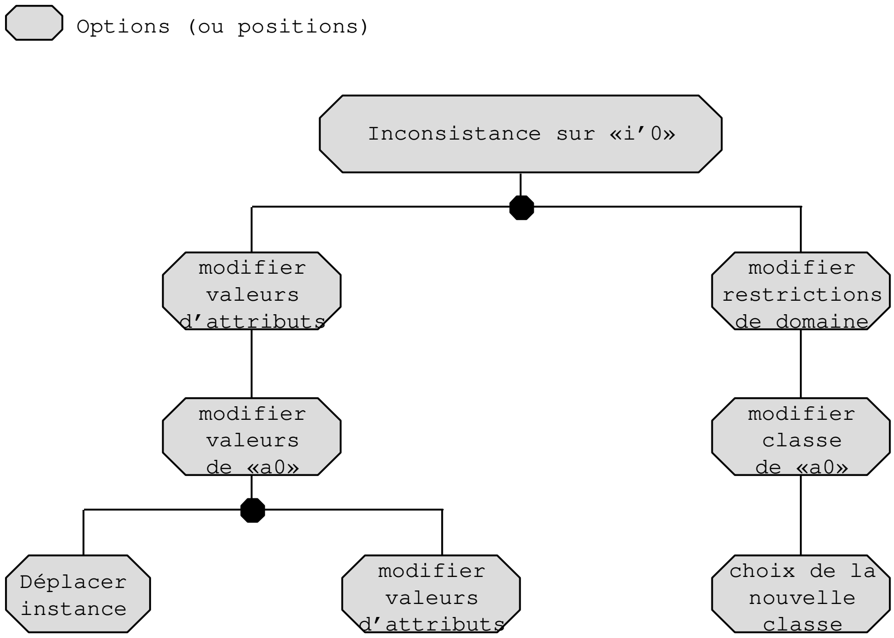


Figure 23 : Représentation des différentes options (ou positions) dans l'arbre de révision

Les problèmes, ainsi que les différentes possibilités de réponse à ceux-ci, apparaissent donc déjà sur l'arbre de révision. Il reste à fournir une façon pour que chacun puisse justifier le choix qui lui paraît le meilleur. Pour cela, nous avons décidé, en nous inspirant de ce qui a été fait pour les systèmes d'argumentation liés au problème de la conception, de créer un type de nœuds appelé "Argument". Ces noeuds contiendront chacun un argument pour ou contre la possibilité de révision représentée par le noeud auquel ils seront rattachés. Ainsi, un graphe d'argumentation adapté au problème de la révision aura l'allure présentée à la Figure 24.

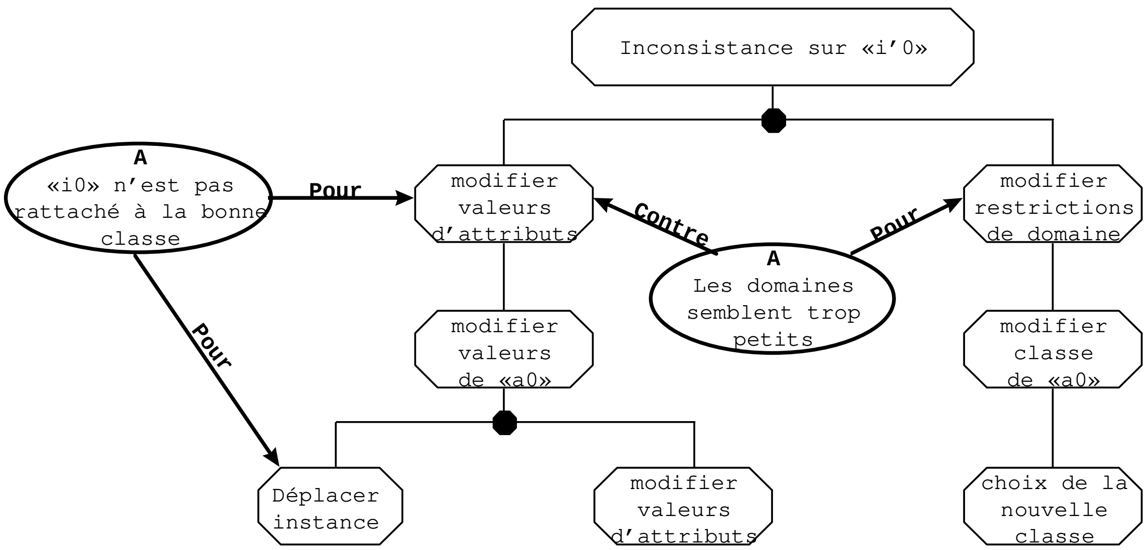


Figure 24 : Graphe d'argumentation pour le problème de la révision

Cependant, les problèmes et options qui apparaissent dans l'arbre de révision correspondent à ceux qui concernent directement la révision. Il pourrait aussi être intéressant, pour les différents intervenants, d'avoir un moyen d'engager des discussions connexes à celles concernant le choix de la meilleure révision. Par exemple, un intervenant pourrait se demander : "Quelle est la meilleure solution pour modéliser telle connaissance ?". Il s'agit là d'un problème de conception : la conception de la modélisation de la connaissance. Ce problème, qui peut être soulevé pendant la révision, pourrait être argumenté grâce à un des systèmes étudiés dans le paragraphe III.1. Il est donc intéressant de choisir un système d'argumentation permettant de modéliser ce type de discussion. Ce problème n'étant pas central dans le modèle d'argumentation pour la révision, nous avons décidé de réutiliser un modèle déjà existant. Nous avons choisi le modèle IBIS, et allons maintenant expliquer les raisons de ce choix.

IBIS essaie d'améliorer le travail des concepteurs en leur fournissant une méthode pour enregistrer l'ensemble de leurs délibérations. Il définit des liens relativement complexes mais complets entre les différents types de noeuds. PHI cherche à améliorer la solution proposée par IBIS en retenant deux inconvénients de ce dernier :

- les liens entre les noeuds d'IBIS sont trop complexes ;
- tous les problèmes rencontrés n'apparaissent pas dans le diagramme car seules les questions qui sont se sont effectivement posées sont représentées.

PHI permet donc de prendre en compte l'ensemble des problèmes, même ceux qui ont trouvé une réponse naturelle et immédiate. De plus, il introduit la notion de sous-problème et replace donc plus facilement un problème dans l'ensemble de la délibération. Ceci est important dans le cas de la conception, car il n'y a pas de base structurée sur laquelle développer la discussion. Cependant, ce n'est pas le cas pour la révision puisque, comme nous l'avons vu précédemment, l'arbre d'argumentation va fournir un support solide. De plus, il nous a paru important, dans la mesure où les discussions, dans le cadre de la révision, seront restreintes (en taille du graphe d'argumentation), de garder une structure d'expression relativement riche.

Contrairement à IBIS et PHI, qui cherchent à enregistrer l'historique des délibérations, QOC cherche à modéliser l'espace des possibilités pour le système à concevoir. Le but est d'aider la compréhension des choix de conception réalisés. Cependant, dans le cas de la révision, cet espace des possibilités est déjà complètement connu : c'est l'algorithme de révision qui le donne. Cette méthode ne nous a donc pas paru intéressante.

Nous avons donc choisi d'utiliser gIBIS pour mettre en place la possibilité d'une argumentation dérivée. Cette méthode a de plus l'avantage d'avoir été déjà implémentée, avec un système hypertexte pour la navigation, ce qui sera aussi le cas avec Troeps.

Nous allons maintenant voir sur un exemple comment l'arbre de révision ainsi que son argumentation associée peuvent être développés.

IV.4. Un exemple détaillé

Nous considérons dans ce paragraphe une base de connaissance appelée « Scolarité » décrivant le système scolaire français. Cette base contient deux hiérarchies de classes : une permettant de décrire les élèves, et l'autre permettant de décrire les établissements.

Un élève a :

- un nom, qui est une chaîne de caractères
- un prénom, qui est une chaîne de caractères
- un lieu d'étude, qui est un « établissement »,
- un nombre d'années d'étude, qui est un entier.

La hiérarchie de classes définie pour les élèves permet de décrire le niveau d'études. Cette hiérarchie est représentée à la Figure 25.

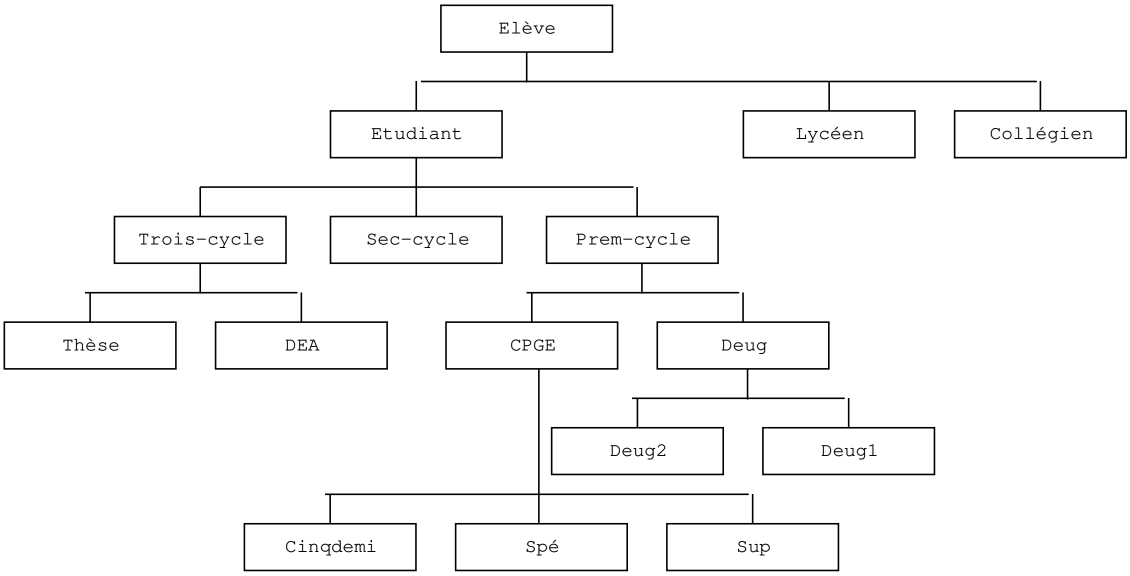


Figure 25 : Hiérarchie de classes définie pour les élèves

Un établissement a :

- un nom, qui est une chaîne de caractères,
- un nombre d'élèves qui est un entier

La hiérarchie de classes définie pour les établissements permet de décrire le type d'établissement auquel on a affaire (collège, lycée, université...). Cette hiérarchie est présentée à la Figure 26.

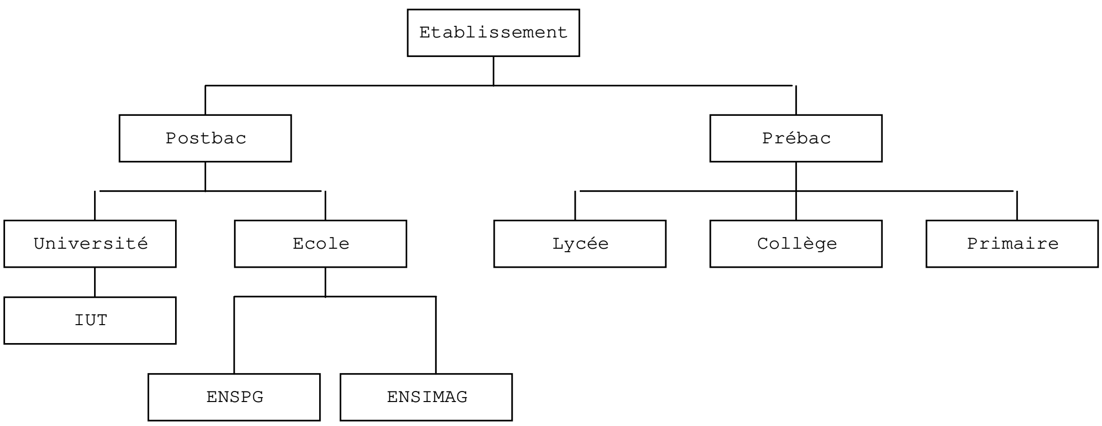


Figure 26 : Hiérarchie de classes définie pour les établissements

Les deux contraintes sur les classes sont les suivantes :

- dans la classe lycée, le nombre d'élèves est compris entre 2000 et 10000 ;
- le lieu d'étude d'un étudiant en « CPGE » est un « lycée » (il doit être rattaché à cette classe).

Les instances de cette base de connaissance sont les suivantes :

- l'établissement « lycée Condorcet » ; il est rattaché à la classe « Etablissement » et a 1000 élèves ;
- l'élève « Dupont » ; il est rattaché à la classe « CPGE ».

Supposons que l'élève « Dupont » travaille au « lycée Condorcet ». On veut donc donner à l'attribut « lieu-étude » de « Dupont » la valeur « lycée Condorcet ». Or, « Dupont » est rattaché à la classe « CPGE », pour laquelle l'attribut « lieu-étude » doit appartenir à la classe « Lycée », et « lycée Condorcet » est rattaché à la classe « Etablissement ». Il y a donc une inconsistance (voir Figure 27) : le module de révision est appelé.

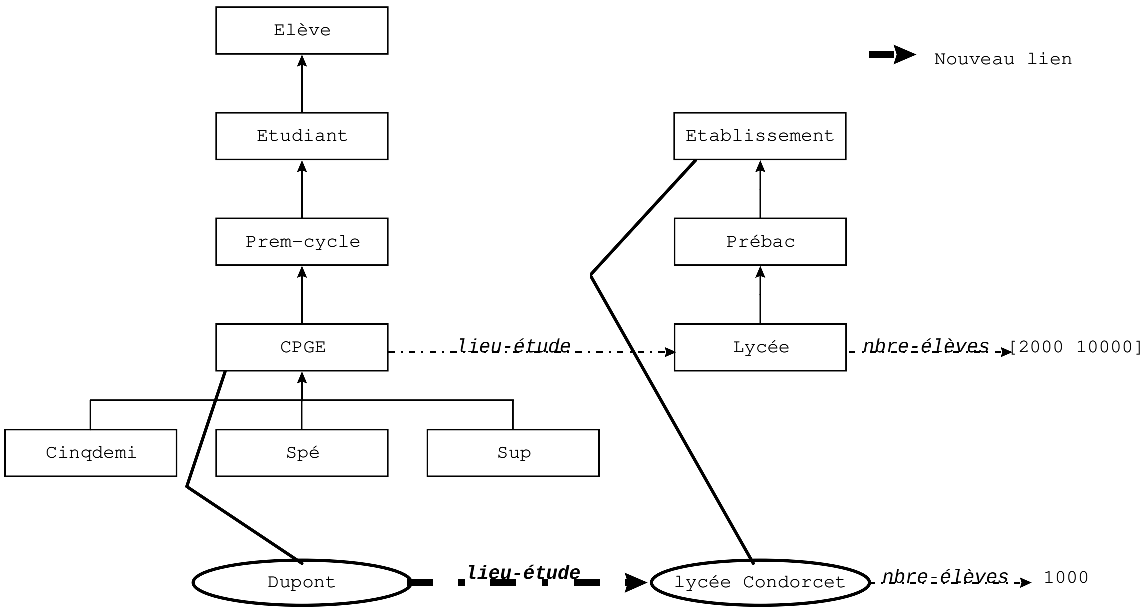


Figure 27 : Inconsistance dans la base Scolarité

Supposons maintenant que la base est partagée par deux utilisateurs : A et B, que l'un est spécialiste des élèves et l'autre des établissements. et que c'est A qui a introduit l'inconsistance. Lorsque A essaie de donner la valeur « lycée Condorcet » à l'attribut « lieu-étude » de l'instance « Dupont », le module de révision détecte une inconsistance au niveau de l'instance « Dupont ». Il propose alors trois solutions pour réviser la base :

- déplacer l'instance « Dupont »
- modifier chaque valeur d'attribut de l'instance « Dupont »
- modifier les restrictions des domaines des attributs de la classe « CPGE »

Supposons que A ne sache pas quelle solution choisir. Il décide d'utiliser le module d'argumentation. Celui-ci lui engendre le graphe présenté à la Figure 28.

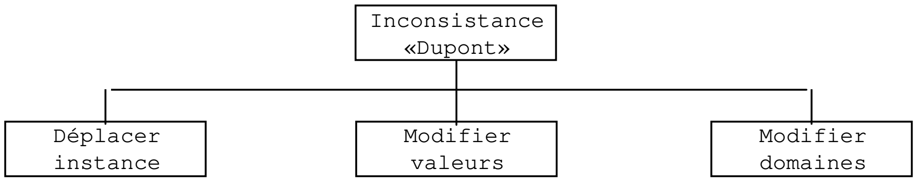


Figure 28 : Premier graphe d'argumentation

Supposons que A considère qu'il ne faut pas déplacer l'instance car l'élève « Dupont » est bien inscrit en classes préparatoires. Il va donc poster l'Argument « Ne pas déplacer l'instance ». Le graphe qu'il obtient après cette opération est présenté à la Figure 29.

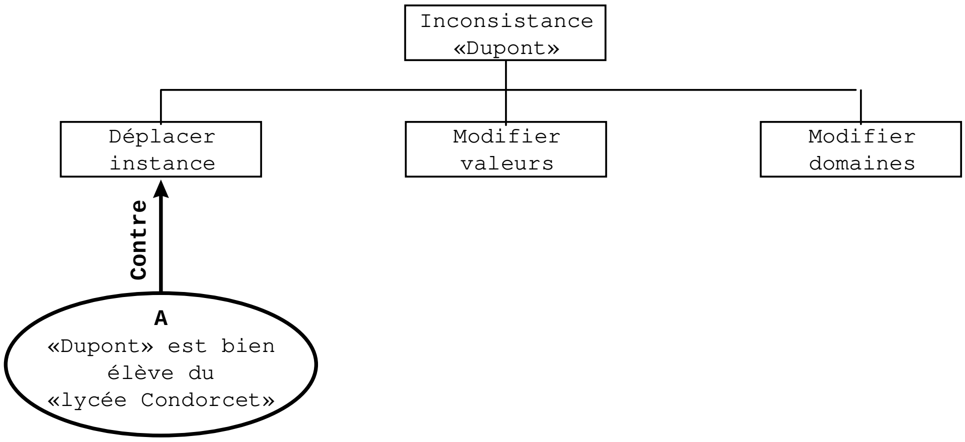


Figure 29 : Graphe d'argumentation après ajout d'un noeud « Argument »

Par contre, il hésite entre modifier des valeurs d'attributs ou modifier des domaines de classes. Il va donc développer l'arbre de révision pour ces deux noeuds-là. Le graphe qu'il obtient alors est représenté à la Figure 30.

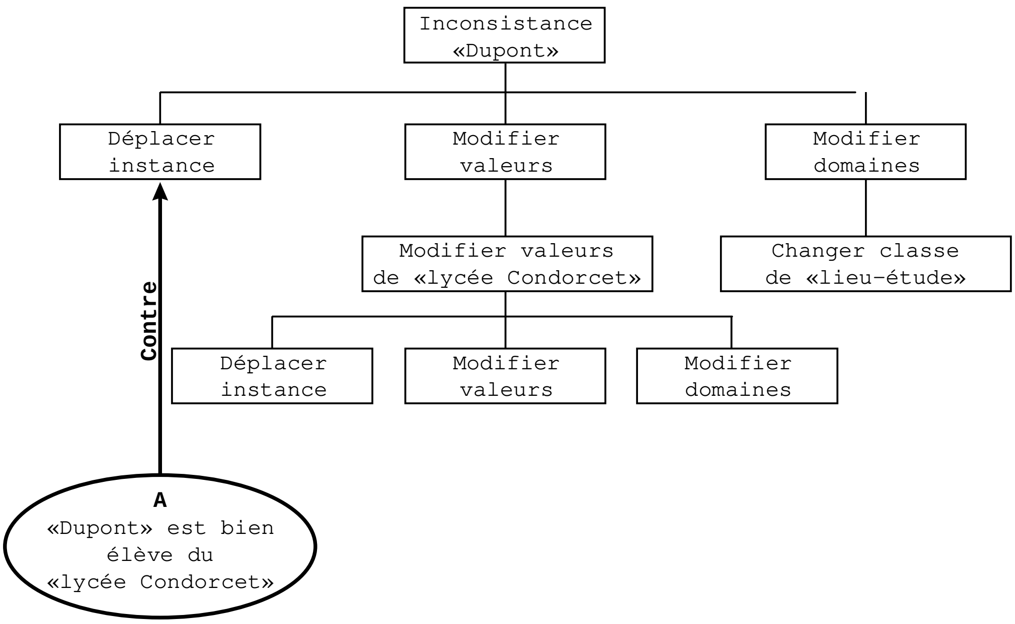


Figure 30 : Graphe d'argumentation développé par A

B peut alors indiquer par un argument, que le « lycée Condorcet » est bien un lycée. Cet argument sera pour la modification des domaines et le déplacement de l'instance. Le graphe résultant de cette modification est présenté à la

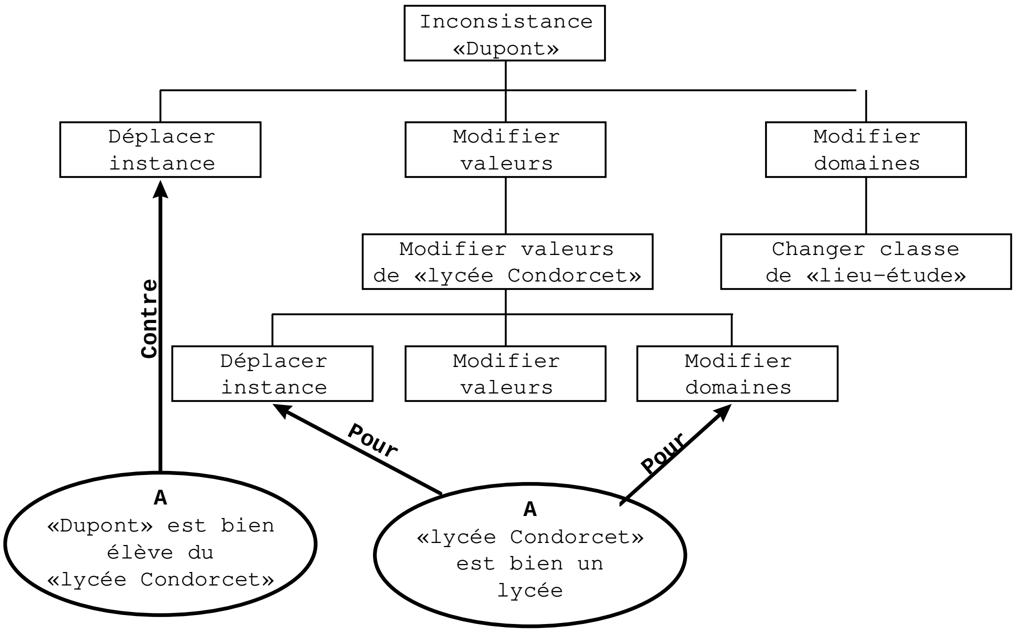


Figure 31 : Graphe d'argumentation modifié par B

Il faut noter que le fait que A et B modifient le même graphe pose des problèmes, classiques dans le domaine du travail coopératif, tels que l'accès concurrent, la mise à jour des copies multiples... Nous présenterons plus en détail ce problème dans le paragraphe IV.6.3.

Nous allons voir maintenant ce que l'approche Langage/Action peut apporter à notre modèle.

IV.5. L'approche Langage/Action

L'approche Langage/Action, comme nous l'avons vu dans le paragraphe III.2, permet de déterminer l'ensemble des communications possibles entre les différents intervenants dans le système. Nous allons voir maintenant quel est le graphe correspondant au modèle d'argumentation pour la révision. Pour la mise en place de ce graphe, nous nous limitons au cas où deux personnes seulement sont impliquées dans la construction du graphe d'argumentation. Soient A et B ces deux personnes.

Chacune de ces deux personnes peut modifier le graphe. Pour cela, il y a plusieurs possibilités :

- ajouter des arguments pour ou contre une des possibilités de révision apparaissant déjà sur le graphe ;
- développer les branches de l'arbre de révision, pour savoir ce que le choix de cette branche implique comme modifications dans la base.

De plus, il faut modéliser les communications entre ces deux personnes. Supposons que c'est A qui a introduit la nouvelle connaissance inconsistante. Il met donc en place le graphe d'argumentation lié à celle-ci et le développe tant que cela lui paraît nécessaire. La création et les diverses modifications du graphe sont notifiées à B au fur et à mesure. Celui-ci peut donc aussi développer le graphe si cela lui paraît nécessaire. Lorsque B reçoit le graphe, il le développe à son tour, et le renvoie à A. Cet échange peut se répéter jusqu'à ce que le graphe soit assez complet pour qu'une décision puisse être prise.

Le problème qui se pose alors est de savoir qui prend la décision, c'est-à-dire dans notre cas particulier, qui choisit la meilleure révision. Dans la mesure où ce système devait être implémenté sur Troeps, qui propose une architecture où chacun a sa propre base de connaissance, nous avons décidé que c'est la personne à qui le problème s'est posé qui devait décider de la solution qu'il considérait comme la meilleure. En effet, une fois les modifications faites sur sa propre base, il peut, grâce au système de gestion de base de connaissance, proposer sa modification à l'ensemble du groupe. C'est alors ce système qui gérera la modification dans la base partagée.

Le graphe Langage/Action correspondant à la solution décrite ci-dessus est donc celui de la Figure 32.

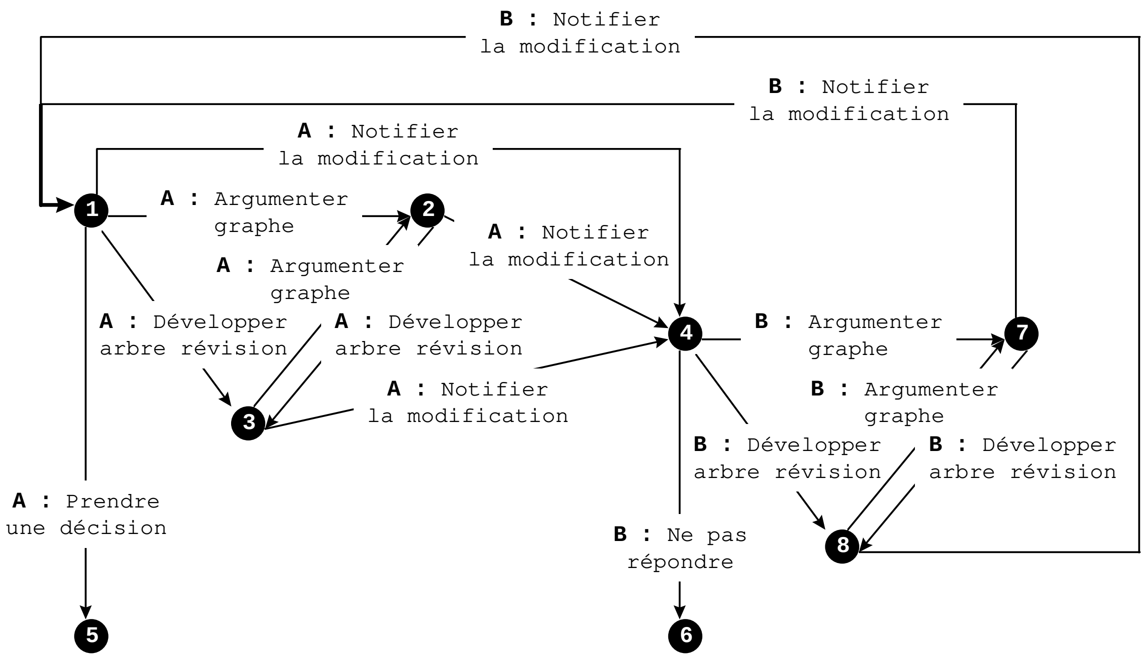


Figure 32 : Graphe Langage/Action pour l'argumentation liée à la révision

Ce graphe est relativement simple : les communications sont peu nombreuses. Nous avons remarqué que toutes ces communications se déroulent autour du graphe d'argumentation : il n'y a pas de communication indépendante de celui-ci. L'interface de Troeps étant gérée par un système hypertexte, nous avons donc décidé de gérer l'argumentation autour d'une page centrale qui contient le graphe d'argumentation ainsi que des boutons permettant d'accéder les différentes fonctionnalités, telles que l'ajout d'un noeud d'argumentation ou le développement de l'arbre de révision.

Nous allons maintenant voir le modèle construit dans son ensemble.

IV.6. Le modèle construit

IV.6.1. Gestion de l'argumentation

Comme nous venons de le voir, le modèle construit est basé sur un graphe d'argumentation. Celui-ci est donc le centre du modèle. Ce graphe est composé de trois parties distinctes :

- l'arbre représentant les différentes révisions possibles ;
- le graphe des arguments directement liés au problème du choix de la meilleure révision ;
- le (ou les) graphe(s) complémentaires(s) permettant d'ouvrir des discussions plus larges au sujet de la modélisation du domaine représenté, et qui sont soulevées par les propositions de révision ; ces dernières discussions sont basées sur le modèle IBIS.

Cette dernière partie n'est pas indispensable.

Chaque noeud du graphe qui concerne l'argumentation (les noeuds « Argument » directement liés aux noeuds de révision, mais aussi les noeuds « Issue », « Position » et « Argument » des discussions dérivées) contient un ensemble d'informations sur le noeud qui sont :

- l'auteur du noeud,
- la date de création du noeud,
- le sujet du noeud,
- un ensemble de mots-clés permettant de cerner le sujet du noeud
- un texte libre décrivant le but du noeud ; il s'agit :
    - ⇨ de la description du problème pour les noeuds de type « Issue »
    - ⇨ de la description de la solution proposée pour les noeuds de type « Position »
    - ⇨ du développement d'une raison pour ou contre une Position pour les noeuds de type « Argument »
- l'ensemble des liens entre ce noeud et les autres noeuds du graphe avec leurs types ; les types des liens retenus sont ceux proposés par le modèle IBIS (voir Figure 13) ; nous avons ajouté les types « Supporte » et « Fait objection à » pour les liens entre un noeud « Argument » et un noeud de révision, ce qui correspond au fait que les noeuds de révision représentent les Positions dans la problématique de choix de la meilleure solution posée par la révision.

Ces informations ne peuvent bien sûr pas toutes apparaître sur le graphe, ce qui nuirait à la vision d'ensemble que nous cherchons à donner de la problématique. Elles sont donc accessibles via des liens hypertexte à partir de chaque noeud.

Nous avons donc vu la structure générale des composants du modèle. Voyons maintenant l'architecture choisie pour ce modèle.

IV.6.2. Architecture

Le modèle que nous avons construit se base, au niveau de l'interface avec l'utilisateur, sur un système hypertexte. Ce choix a été guidé par deux éléments :

- c'est le choix qui a été réalisé pour l'implémentation de gIBIS ;
- Troeps, sur lequel l'implémentation va être réalisée, a une interface sur le web.

L'avantage de tels systèmes est qu'ils permettent de donner une vue d'ensemble des informations gérées par le système, tout en laissant les différents niveaux de détail très accessibles.

Nous avons donc décidé que le graphe d'argumentation apparaîtrait sur une page centrale, sur laquelle se trouveraient aussi les différents boutons donnant accès aux fonctions permettant d'augmenter ce graphe. Chaque bouton donnera accès à une page contenant un formulaire à remplir pour créer l'objet voulu. Une fois cette page remplie, on reviendra sur la page centrale, et le nouveau graphe sera mis en place.

Les différents boutons nécessaires sont les suivants :

- un bouton permettant d'augmenter l'arbre de révision ;
- un bouton permettant d'ajouter un noeud argument au graphe ;
- un bouton permettant d'ajouter un lien entre un noeud Argument déjà présent sur le graphe, et un noeud de l'arbre de révision ; cela est nécessaire dans la mesure ou un même noeud Argument peut être lié à plusieurs noeuds de l'arbre de révision ;

Dans un deuxième temps, pour permettre la mise en place de discussions connexes (comme discuté au paragraphe IV.3), il sera nécessaire d'ajouter les boutons suivants :

- un bouton permettant d'ajouter un noeud Issue au graphe ;
- un bouton permettant d'ajouter un noeud Position au graphe ;
- des boutons permettant d'ajouter les liens nécessaires entre Issues, Positions et Arguments.

L'architecture est donc celle représentée à la Figure 33.

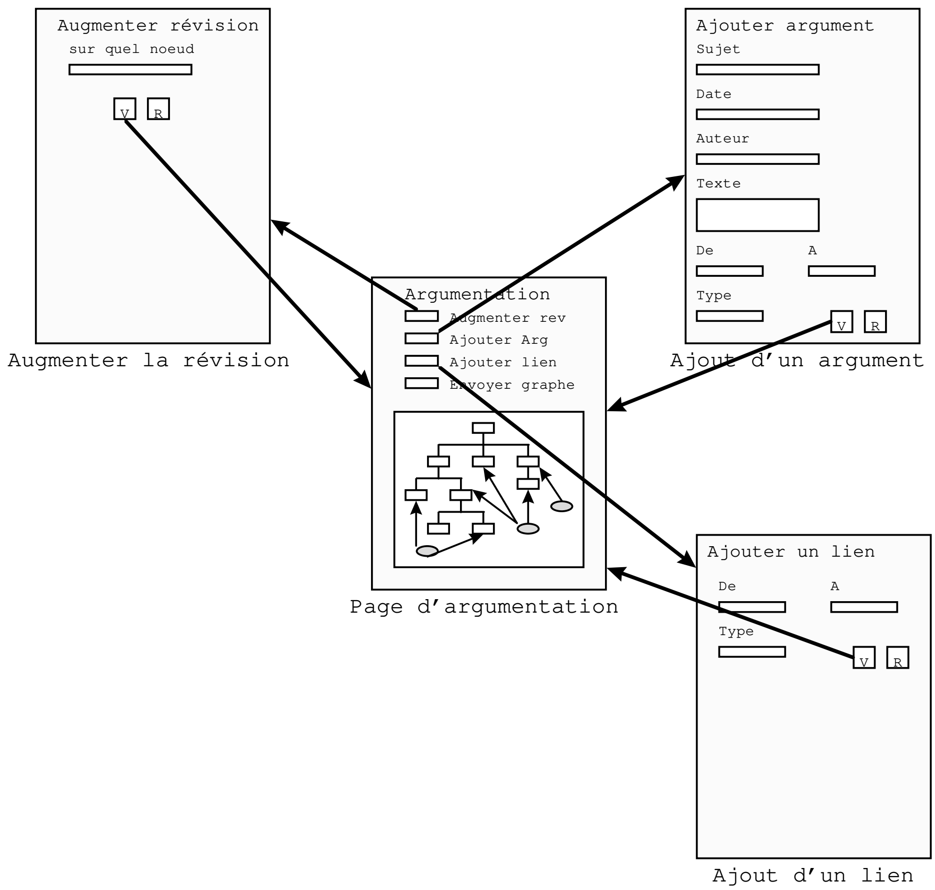


Figure 33 : Architecture du modèle construit

IV.6.3. Le problème des bases partagées

Deux problèmes se posent du fait du travail collaboratif. D'une part, plusieurs utilisateurs travaillant sur le même graphe, il faut décider si ce graphe est dupliqué, ce qui va poser des problèmes de mise à jour, ou s'il ne l'est pas, ce qui va poser des problèmes d'accès concurrent. Nous n'avons pas tranché cette question. Il s'agit là de problèmes bien connus en système. De plus, il existe déjà un protocole qui s'occupe de la gestion du partage des bases et de la mises à jour de celles-ci. Nous avons donc décidé que ce serait ce protocole qui gérerait aussi les problèmes posés par l'argumentation.

De plus, chacun est libre de réaliser les modifications qu'il souhaite sur sa propre base de connaissance. Imaginons qu'une personne A notifie une modification sur un graphe de révision à une personne B concernant une inconsistance sur un objet « i ». B a très certainement une base un peu différente de celle de A. Les inconsistances ne sont donc pas exactement les mêmes. Si c'est le cas sur l'exemple que A lui a envoyé, il ne pourra pas tester et aider A de façon autonome puisqu'il aura besoin de la base de A pour obtenir les mêmes résultats que lui. Une solution à ce problème consiste à créer, au moment de l'argumentation, une nouvelle base, copie de la base de A, accessible par tous, et dédiée au travail d'argumentation. Les accès à cette base devront être exclusifs, pour les raisons habituelles lors d'un travail concurrent.

Nous avons vu le modèle qui a été construit. Nous allons maintenant voir les résultats de l'implémentation de ce modèle.

V. Réalisation

Le modèle a été implémenté sur le système à bases de connaissances Troeps. Pour cette implémentation, quelques restrictions ont été apportées. C'est ce que nous verrons dans une première partie. Dans un deuxième temps, nous verrons les structures de données mises en place pour l'implémentation. Enfin, nous verrons le résultat de cette implémentation grâce à l'interface réalisé.

V.1. Restrictions

Nous avons souhaité réaliser une implémentation du modèle proposé. Cela a posé plusieurs problèmes, qui ont été déjà partiellement soulevés, et que nous allons voir maintenant.

Pour commencer, le modèle proposé donne la possibilité aux divers utilisateurs d'avoir des discussions connexes au problème de la révision. Ces conversations sont structurées suivant le modèle IBIS. Dans la mesure cette partie du modèle créé n'est pas indispensable, nous ne l'avons pas implémentée. Par contre, comme nous le verrons dans le paragraphe suivant, nous avons tenu compte de cette possible évolution dans l'implémentation que nous avons réalisée.

Un autre problème que nous avons déjà abordé concerne le partage des bases. De même, nous n'avons pas traité ce problème et nous sommes limités, pour l'implémentation, au cas d'une base partagée et accédée séquentiellement par les utilisateurs.

Nous allons voir maintenant les structures de données que nous avons mises en place pour implémenter ce module.

V.2. Structures de données

Comme nous l'avons vu dans le chapitre précédent, notre modèle est basé sur le graphe d'argumentation. Ce graphe peut se diviser en deux parties distinctes :

- l'arbre d'argumentation,
- l'ensemble des noeuds « Argument » qui y sont rattachés.

Dans ce paragraphe, nous allons voir comment ce graphe a été représenté. De plus, nous parlerons de la façon dont cet arbre a été stocké dans les structures de Troeps.

V.2.1. Représentation du graphe d'argumentation

Dans cette première partie, nous abordons plus particulièrement le problème du choix des structures de données permettant de représenter le graphe d'argumentation.

La structure centrale du graphe d'argumentation est l'arbre de révision. Nous avons donc implémenté une structure d'arbre classique : chaque arbre a une « racine », qui contient l'information nécessaire pour ce noeud, et une liste de fils, qui sont eux-mêmes des arbres. De plus, à chaque noeud peuvent être rattachés des noeuds « Argument ». Un champ contenant la liste de ces noeuds a donc été ajouté à la structure de données des arbres de révision. Voyons la structure de ces noeuds « Argument ».

Pour ce qui concerne le dispositif d'argumentation proprement dit, nous avons essayé d'avoir une structure évolutive. En effet, même si ce n'est pas implémenté dans ce premier prototype, nous souhaitons fournir une structure complète de type IBIS aux utilisateurs. Or, les noeuds de type « Issue », « Position » et « Argument » ont tous une structure de base commune contenant les informations générales suivantes :

- le nom du noeud, permettant une identification unique du noeud et un affichage simple ; ce nom est géré directement par le système, ce qui permet d'en garantir l'unicité ;
- l'auteur du noeud ;
- la date de création du noeud ;
- le sujet du noeud ;
- les mots-clés permettant de cerner très rapidement le développement de l'argumentation introduite dans le noeud ;
- le texte expliquant plus en détail l'argumentation développée (le problème posé, une solution pour y répondre, ou un argument détaillé appuyant une des Positions).

Suivant les types de noeuds, il y a plusieurs types de liens possibles vers différents types de noeuds. Nous avons donc créé une structure de type « noeud d'argumentation », de laquelle est dérivée la structure d'Argument, et pourront être dérivées celles d'Issue et de Position.

La structure d'Argument contient, en plus des champs définis dans la structure générale de noeuds d'argumentation, un champ contenant la liste des liens vers les noeuds révision. Dans une perspective d'évolution vers un système IBIS complet, il faudrait rajouter un champ contenant la liste des liens vers des noeuds « Position ». Voyons donc maintenant la structure d'un lien.

Un lien doit contenir les informations suivantes :

- l'auteur du lien ;
- la date de création du lien ;
- le type du lien ; ce type est une chaîne de caractères ; les chaînes possibles sont « SUPPORTS » et « OBJECTS-TO » ; cet ensemble sera augmenté des autres types de liens définis dans le modèle IBIS lors d'une éventuelle évolution ;
- le noeud vers lequel l'argument pointe ; ce noeud peut être de n'importe quel type, et ceci une fois de plus pour des problèmes d'évolution.

Ces informations sont celles définies dans le modèle IBIS.

Nous avons donc défini une structure centrée sur l'arbre de révision. Le problème qui se pose maintenant est de relier cette structure à Troeps.

V.2.2. *Stockage du graphe d'argumentation*

Une fois les structures de données définies, nous nous sommes posé le problème du stockage de ces structures. Le graphe d'argumentation, et plus précisément l'arbre de révision qui en est la base, est très étroitement lié aux structures de Troeps. En effet, la révision se fait sur les structures de la base de connaissance. Nous avons donc cherché à rattacher le graphe d'argumentation aux structures de Troeps.

Dans Troeps, il existe un mécanisme d'annotations sur chaque entité (concept, classe, objet...). Ce mécanisme permet de stocker n'importe quel type et quantité d'informations, Nous avons donc choisi d'utiliser ce mécanisme pour stocker le graphe d'argumentation. Le problème a alors été de savoir sur quel structure nous devions déclarer l'annotation. Nous avons donc analysé les cas de révision possibles.

Comme nous l'avons vu dans le paragraphe II.3.2, il y a quatre actions élémentaires possibles pour réviser une base de connaissance à objets :

1. déplacer un objet dans la hiérarchie des classes
2. supprimer la valeur de l'attribut d'un objet
3. élargir les restrictions d'un attribut (en élargissant le domaine de celui-ci ou en utilisant une classe plus générale)
4. déplacer une classe vers le haut dans la hiérarchie.

La révision peut donc s'opérer sur une classe ou sur un objet. Il est donc possible de créer une annotation contenant ce graphe sur l'objet ou la classe concernée par le processus de révision. C'est ce que nous avons choisi de faire.

Les avantages de cette méthode sont, d'une part, que le graphe de révision est directement accessible : il se trouve là où on est en train de travailler. D'autre part, cela donne un moyen très simple de vérifier si la structure sur laquelle on travaille est impliquée par ailleurs dans un processus de révision et d'argumentation : il suffit de vérifier s'il y a une annotation du type de celles définies pour le stockage de la structure d'argumentation. Il est donc facile d'avertir n'importe quel utilisateur que la structure qu'il est en train d'utiliser est impliquée dans un processus d'argumentation.

Nous venons de voir les structures de données que nous avons définies pour implémenter notre module d'argumentation. Nous allons maintenant nous intéresser au fonctionnement du logiciel en présentant l'interface mise en place.

V.3. Interface

Comme nous l'avons vu au paragraphe II.2.3, Troeps possède un éditeur de bases de connaissances sur le World-wide Web. L'interface que nous avons développée est donc aussi sur le Web.

Pour présenter l'implémentation réalisée, nous nous basons sur l'exemple développé dans le paragraphe IV.4. Nous disposons donc d'une base de connaissance modélisant le système scolaire français. Lorsqu'on essaie de donner la valeur « Lycée Condorcet » à l'attribut « lieu-étude » de l'élève « Dupont », une inconsistance est levée. Le module de révision est alors appelé. Il affiche la page présentée à la Figure 34.

Figure 34 : Première page de révision

Comme on peut le remarquer, un bouton « Argumentation » a été mis en place sur cette page. Ce bouton permet d’arriver à la page centrale du module d’argumentation présentée à la Figure 35.

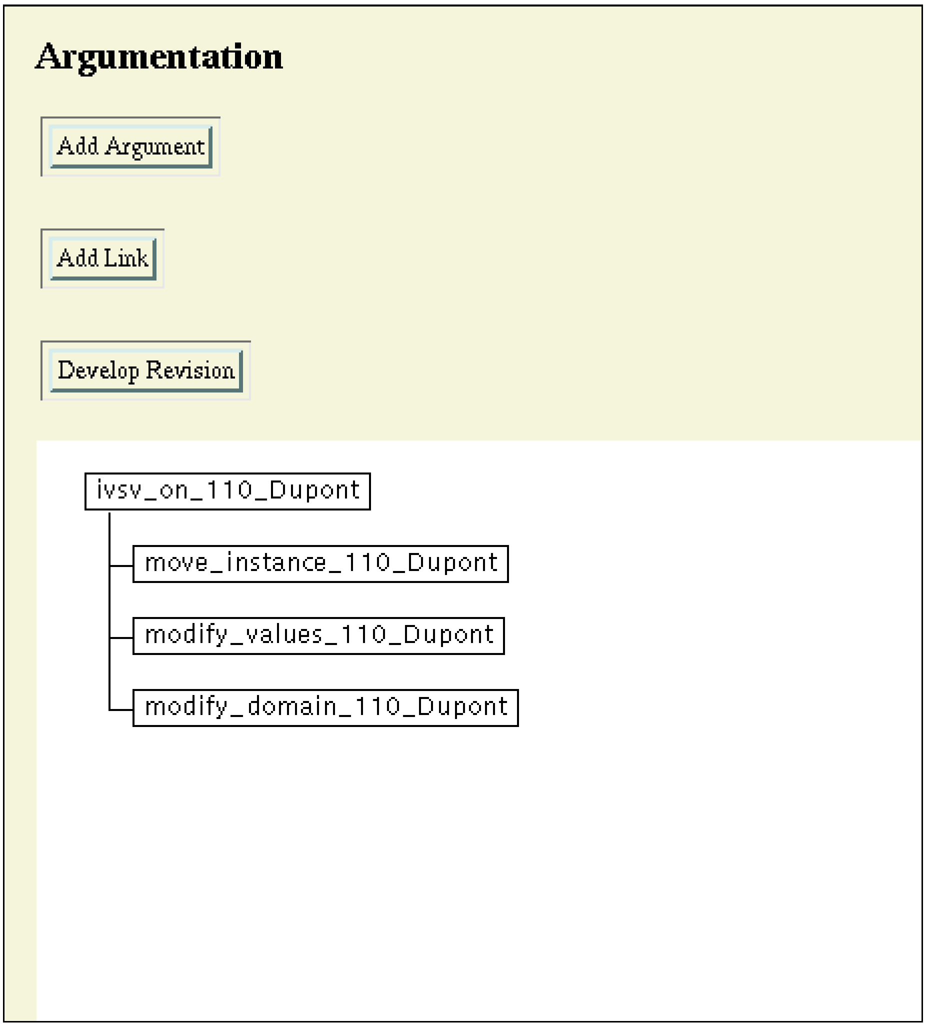


Figure 35 : Première page d'argumentation

A partir de cette page, il est possible de modifier le graphe en :

- développant l'arbre de révision,
- ajoutant un noeud Argument,
- ajoutant un nouveau lien entre un noeud Argument et un noeud révision déjà existant.

Si l'on reprend l'exemple du paragraphe IV.4, on veut maintenant ajouter un argument contre le déplacement de l'instance « Dupont ». Il faut donc cliquer sur le bouton « Add Argument » de la Figure 35. Cela mène à la page présentée à la Figure 36.

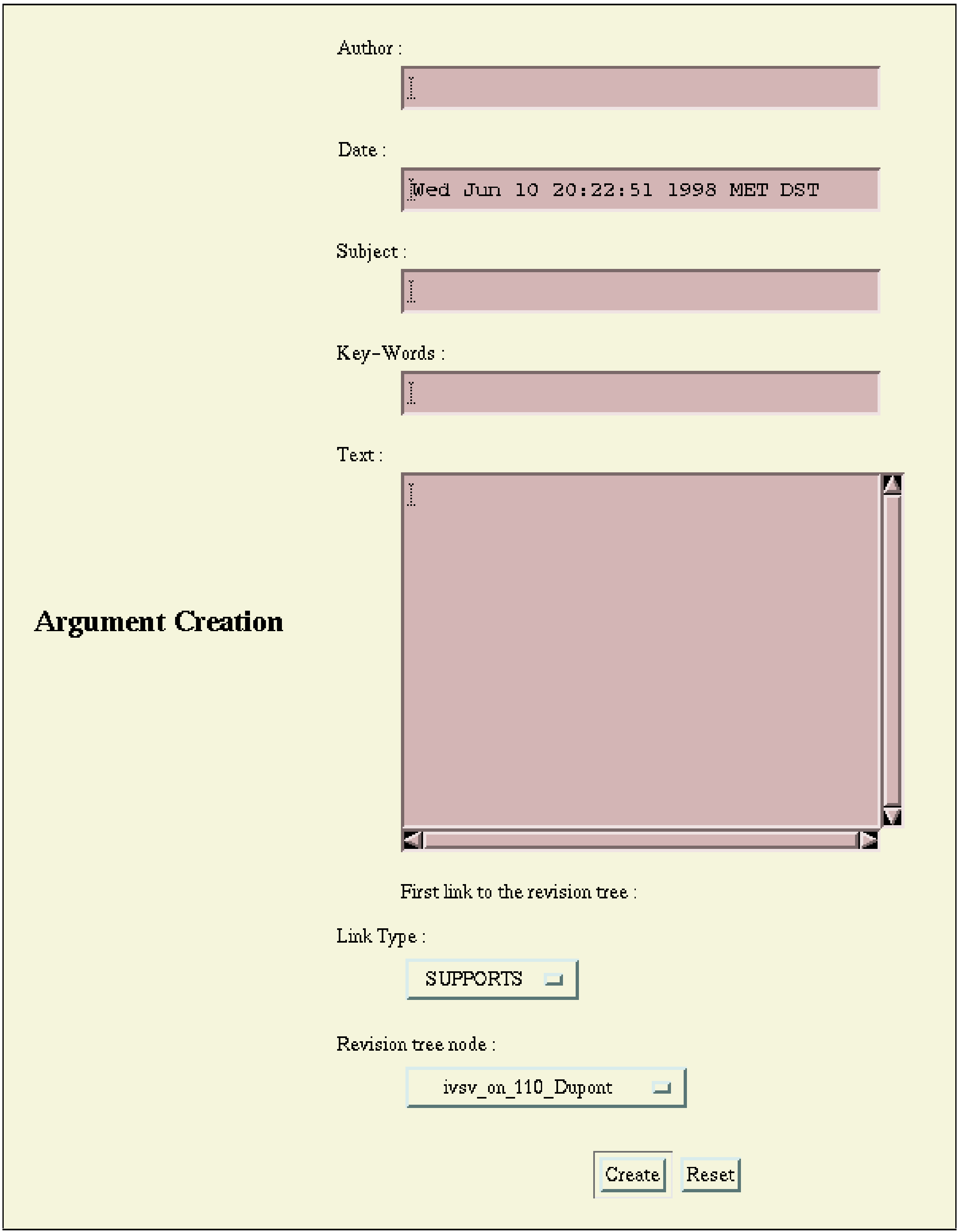


Figure 36 : Page d'ajout d'un Argument

Cette page propose à l'utilisateur de donner les informations contenues dans un noeud Argument, c'est-à-dire :

- l'auteur,
- la date (ce champ est en fait renseigné automatiquement),
- le sujet,
- les mots-clés,

- le texte détaillant l’argumentation,
- le type du lien entre ce noeud Argument et le noeud révision auquel il doit être rattaché,
- le noeud révision auquel l’argument doit être rattaché.

Ces deux derniers champs sont en fait renseignés grâce à une sélection dans la liste des liens possibles, pour le premier, et des noeuds révision possibles, pour le second. Cela permet d’une part de faciliter le choix à l’utilisateur dans la mesure où il n’a pas à se souvenir des noeuds révision qui existent dans le graphe, et d’autre part de limiter les risques d’erreur.

Une fois que les champs nécessaires sont remplis, l’utilisateur valide la création de l’argument en cliquant sur le bouton « Create ». Il faut noter qu’il n’est pas nécessaire de renseigner tous les champs. Les seuls champs nécessaires sont le type de lien, ainsi que le noeud révision auquel l’argument doit être rattaché. Cependant, la création d’un argument n’ayant ni sujet, ni texte, ne présente aucun intérêt.

L’opération de validation ramène sur la page précédente. Le graphe est modifié, comme le montre la Figure 37.

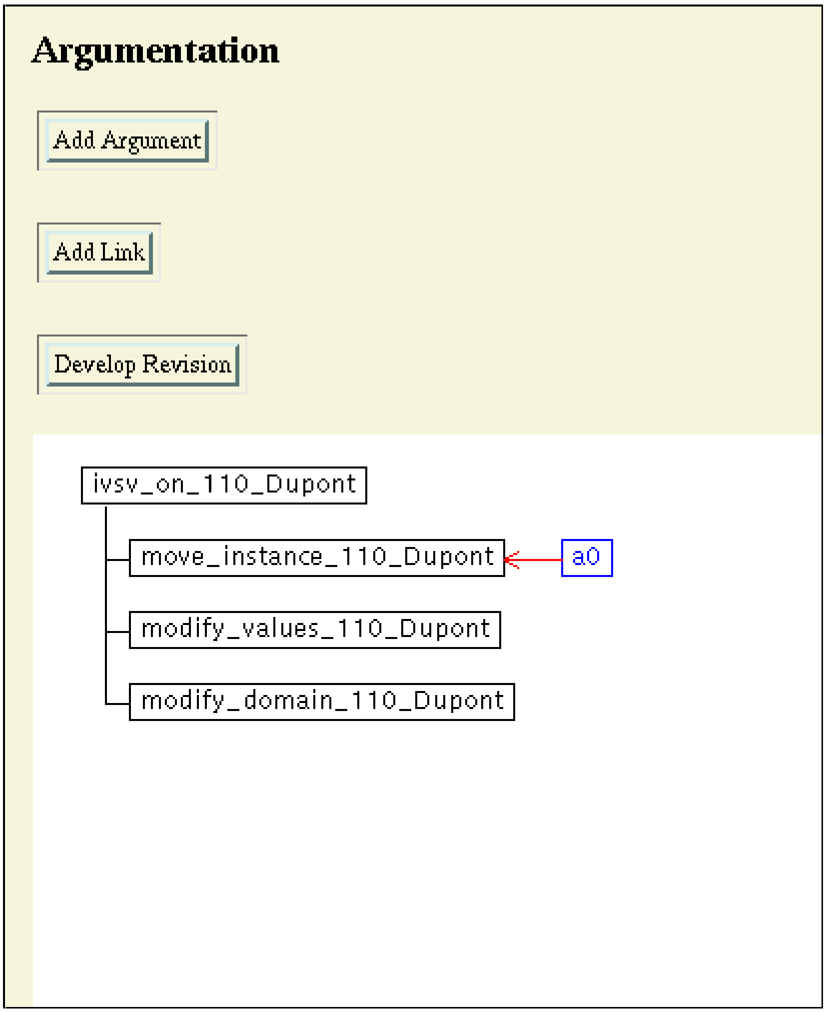


Figure 37 : Page d’argumentation après ajout d’un argument

Sur cette figure, l'argument ajouté s'oppose au noeud « déplacer instance ». C'est pourquoi le lien entre ces deux noeuds est représenté en rouge. Les liens « supporte » sont représentés en vert. Ce code de couleur est très classique. Il permet de ne pas surcharger le graphe tout en ne laissant aucune ambiguïté quant au sens de celui-ci.

A partir de la page d'argumentation, il est possible d'ajouter un lien entre un noeud Argument déjà existant et un noeud révision. Pour cela, il faut cliquer sur le bouton « Add Link » représenté à la Figure 35 ou à la Figure 37. Cette opération mène à la page représentée à la Figure 38.

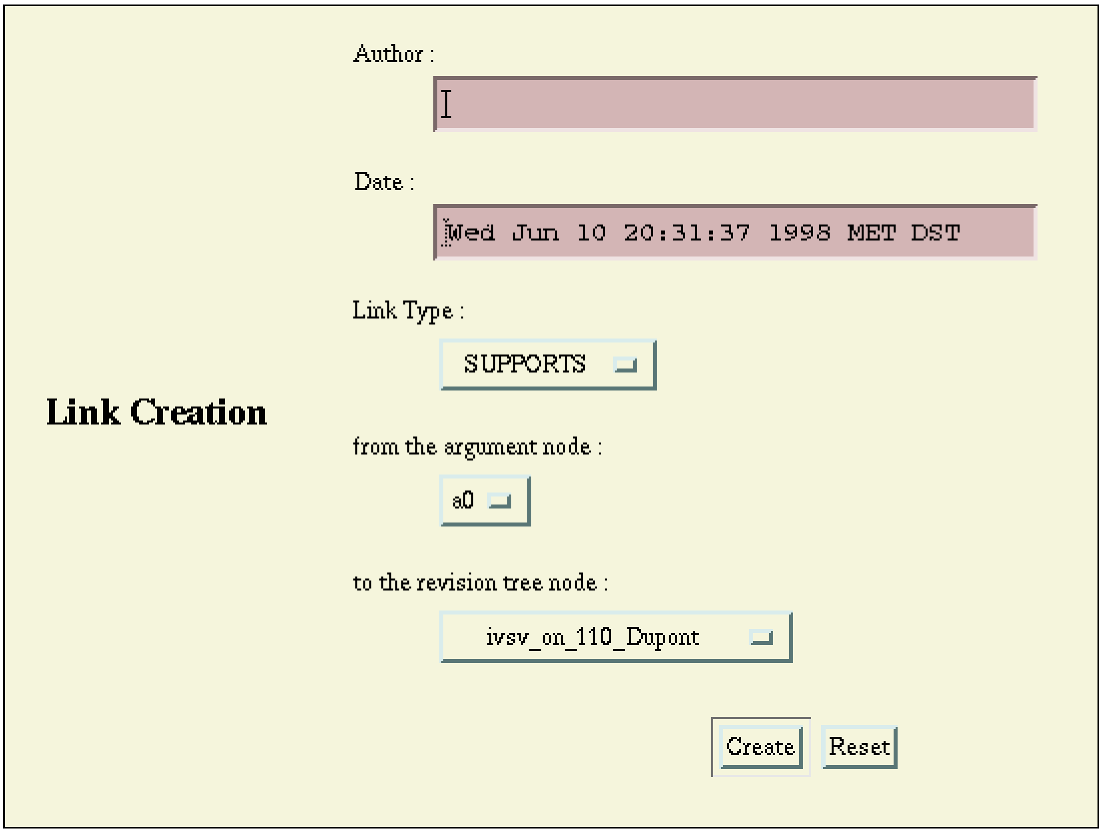


Figure 38 : Page de création d'un lien

De même que pour la page de création d'un argument, cette page propose un formulaire à remplir. Il faut indiquer sur celui-ci le type de lien à créer, ainsi que de quel noeud Argument vers quel noeud révision ce lien est.

Pour modifier un argument, ou seulement pour le consulter, il suffit de cliquer dessus sur le graphe. On revient alors à une page du type de celle de la Figure 36.

A partir de la page d'argumentation, on peut aussi développer la révision. Cela se fait en cliquant sur le bouton « Develop Revision ». Cette opération mène à la page représentée à la Figure 39.

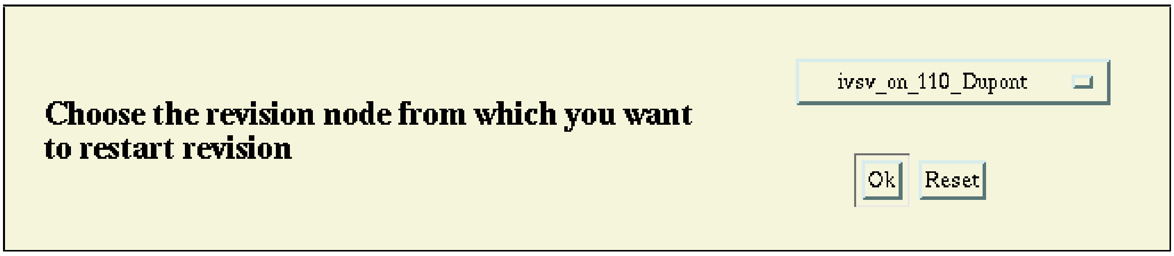


Figure 39 : Page de développement de la révision

Cette page demande à l’utilisateur à partir de quel noeud il veut développer la révision. Seules les feuilles de l’arbre de révision sont proposées dans les choix possibles. Cela permet une fois de plus de limiter le nombre d’erreurs.

A partir de ces pages, les utilisateurs peuvent développer le graphe d’argumentation et ainsi aboutir plus facilement à une décision.

VI. Conclusion

Lors de l’introduction dans la base d’une connaissance conflictuelle, l’utilisateur peut ne pas maîtriser tous les domaines concernés par le conflit. Il faut donc l'aider à choisir la meilleure solution pour résoudre celui-ci. Pour cela, un module de révision de la base permet de proposer l’ensemble des solutions qui rendent celle-ci consistante. Cependant, les modifications possibles sont proposées l’une après l’autre, ce qui ne permet d’avoir une vue d’ensemble de l’opération.

Nous avons donc développé un protocole permettant d’aider l’auteur de la connaissance conflictuelle à choisir la meilleure solution. Les bases de connaissances sont le plus souvent partagées. Nous avons donc eu l’idée d’essayer de faire communiquer les différents utilisateurs de la base en leur donnant les moyens d’avoir un discussion structurée autour du problème posé par la révision. Pour cela, nous nous sommes inspirés des modèles d’argumentation développés pour l’aide à la conception.

Le modèle que nous avons construit permet de développer, à partir de la structure de la révision, un graphe autour duquel la discussion peut s’instaurer. Ce graphe représente les choix pertinents possibles de révision. On peut y lier des arguments plaidant en faveur de l’une ou l’autre des solutions. De plus, il est possible d’avoir des discussions connexes au problème de la révision grâce à un système d’argumentation plus complet basé sur le modèle IBIS.

L’intérêt d’un tel système est qu’il dépasse certains problèmes d’IBIS et permet de renforcer la focalisation de la discussion car :

- le but de la discussion est fixé,
- les différentes options sont restreintes.

On peut donc se concentrer sur les arguments proprement dits.

Dans une perspective plus large, il serait intéressant de savoir comment, en détail, les différents utilisateurs du système peuvent effectivement communiquer.

VII. Bibliographie

[Alchourrón83] Carlos E. Alchourrón, Peter Gärdenfors, David Makinson
On the logic of theory change : partial meet contraction and revision functions
The Journal of Symbolic Logic, Volume 50, Number 2, p. 510-530, June 1985

[Alemany98] Christophe Alémany
Etude et réalisation d'une interface d'édition de bases de connaissances au travers du World Wide Web
Mémoire CNAM, Grenoble (FR), 1998

[Conklin88] Jeff Conklin, Michael L. Bergman
gIBIS : A Hypertext Tool for Exploratory Policy Discussion
ACM Transactions on Office Information Systems, Vol. 6, Octobre 88, p. 303-331

[Crampé96] Isabelle Crampé, Jérôme Euzenat
Fondements de la révision dans un langage d'objets simple
Proc. 3ième journées « langages et modèles à objets », 1996

[Crampé97] Isabelle Crampé
Révision interactive dans une base de connaissance par objets
Thèse d'informatique, université Joseph-Fourier, Grenoble (FR), (14 Octobre) 1997

[Crampé98] Isabelle Crampé, Jérôme Euzenat
Object knowledge base revision
ECAI 98, 13th European Conference on Artificial Intelligence

[CSCW96] ACM conference on Computer Supported Cooperative Work. November 16-20, 1996.
Actes du congrès

[Euzenat96] Jérôme Euzenat
Corporate memory through cooperative creation of knowledge bases and hyper-documents
Actes 10th knowledge acquisition workshop, Banff (CA)pp(36)1-18, 9-14 novembre 1996
http://www.inrialpes.fr/sherpa/papers/euzenat96b/euzenat96b.html
ftp://ftp.inrialpes.fr/pub/sherpa/publications/euzenat96b.ps.gz

[Fisher91] Gerhard Fischer, Andreas C. Lemke, Raymond McCall, Anders I. Morch
Making Argumentation Serve Design
Human Computer Interaction, Vol. 6, Iss. 3-4, pp. 393-419

[Flores88] Fernando Flores, Michael Graves, Brad Hartfield, Terry Winograd
Computer Systems and the Design of Organisational Interaction
ACM Transactions on Office Information Systems, Vol. 6, no 2, Avril 1988, pp. 153-172

[Karacapilidis97] N.I. Karacapilidis, B. Trousse
Computer-Supported Argumentation for Cooperative Design on the World-Wide Web
P. Siriruchatapong, Z. Lin & J.P. barthes (eds), Proceedings of the 2nd International Workshop on CSCW in Design (CSCWD'97), Bangkok, Thailand, November 26-28, 1997, International Academic Publishers, Beijing, pp 96-103.

[Kunz70] W. Kunz, H.Rittel
Issues as elements of information systems
Working Paper n^{o} 31
Berkeley, University of California, Center for Planning and Development Research

[Lubich95] Hannes P. Lubich
Towards a CSCW Framework for Scientific Cooperation in Europe
Lecture notes on computer science, Vol 889

[McCall91] Raymond McCall
PHI : A conceptual foundation for design hypermedia
Design Studies, 12, 30-40

[MacLean91] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, Thomas P. Moran
Questions, Options, and Criteria : Elements of Design Space Analysis
Human-Computer Interaction, Vol. 6, Iss. 3-4, pp. 201-250

[Mariño90] Olga Mariño, François Rechenmann, Patrice Uvietta
Multiple perspectives and classification mechanism in Object-oriented Representation
Actes 9th ECAI, Stockholm (SE), pp. 425-430 (Pitman Publishing, London (GB)), (8-10 août) 1990

[Masini91] Gérald Masini, Amedeo Napoli, Dominique Colnet, Daniel Léonard, Karl Tombre
Les langages à objets
InterEditions, Collection iia, Mai 1991

[Nebel90] Bernhard Nebel
Reasoning and revision in hybrid representation systems
LNCS 422, 1990

[Nebel94] Bernhard Nebel
Base revision operations and schemes : semantics, representation and complexity
Actes 11th ECAI, Amsterdam, pp.341-345, Août 1994

[Sherpa95] Projet Sherpa
Tropes 1.0 reference manual
Rapport interne, INRIA Rhône-Alpes, Grenoble (FR), juin 1995 (rev. Tropes 1.1 reference manual, novembre 1997, 120p.), 85p.
http://hytropes.inrialpes.fr/docs/tropes-manual.html
ftp://ftp.inrialpes.fr/pub/sherpa/rapports/troeps-manual.ps.gz

[Shum] Simon Buckingham Shum
http://kmi.open.ac.uk/~simonb/

[Shum91] Simon Buckingham Shum
QOC Design Rationale Retrieval : A Cognitive Task Analysis, & Design Implications
Thèse de doctorat, soutenue à Rank Xerox EuroPARC, Esprit Basic Research Action 3066 (AMODEUS Project), et SERC CASE Award 88504176 ; Rapport technique

[Shum&] Simon Buckingham Shum, Allan MacLean, Victoria M.E. Bellotti, Nick V. Hammond
Graphical Argumentation and Design Cognition
A paraître dans Human-Computer Interaction

[Takayuki97] Ito Takayuki, Shintani Toramatsu
Persuasion among Agents : An Approach to Implementing a Group Decision Support System on Multi-Agent Negociation
Proc. 15th IJCAI, Nagoya (Japon)

[Winograd87] Terry Winograd
A Language/Action Perspective on the Design of Cooperative Work
Human-Computer Interaction, Vol. 3, pp. 3-30

[Yakemovic90] K.C. Burgess Yakemovic, E. Jeffrey Conklin
Report on a Development Project Use of an Issue-Based Information System
Proc. CSCW90, pp. 105-118