CHAPITRE 5

IMPLÉMENTATION DU PROTOTYPE

L'implémentation d'un prototype a pour but de clarifier davantage comment le logiciel envisagé pourrait fonctionner et aussi et surtout de servir de preuve de concept. Le prototype présenté ne couvre par toutes les étapes de conception, mais représente une des plus importantes dans la conception d'un bâtiment soit celle de la conception des sous-systèmes horizontaux. Celle-ci sera faite en fonction de baies dont les dimensions sont établies. Une partie seulement des connaissances seront donc modélisées.

L'implémentation générale a été faite en collaboration avec le finissant en génie logiciel, Dominic Fortin. L'auteur de ce mémoire a collaboré à la définition des besoins et à la mise en place des éléments du prototype. L'auteur a aussi modélisé et implémenté les connaissances de conception préliminaire. Pour davantage d'informations sur l'implémentation générale, veuillez consulter Fortin (2006).

Dans ce chapitre, l'architecture générale du prototype global est expliquée, ce dernier incluant le module de raisonnement géométrique, la base de données et le DKM. Le module de gestion des connaissances (DKM) est expliqué en détail. De plus, son architecture y est décrite ainsi que chacun des *paquetages* utilisés. Par la suite, l'utilisation des nœuds décisionnels par le DKM est expliquée. Pour finir, l'interface utilisée est présentée et l'annexe 7 présentant des extraits du code du prototype est introduite.

5.1. Architecture générale du prototype complet

Le module de gestion des connaissances (DKM) implémenté au cours de cette recherche doit communiquer avec un module de raisonnement géométrique (StAr) afin de permettre des analyses en fonction du modèle de bâtiment (c.-à-d. une analyse en fonction des relations entre chacune des entités). De plus, il est pertinent de pouvoir raisonner géométriquement tôt dans le processus.

Plusieurs problèmes devaient être résolus pour permettre l'interaction entre StAr et le DKM :

1. Un modèle doit pouvoir être partagé par StAr et le DKM;

2. Étant donné que chacune des applications est appelée à modifier les données de manière concurrente, il est nécessaire d'implémenter un suivi de ces modifications et préserver le modèle dans son état le plus récent.

3. StAr n'a pas un modèle de bâtiment persistant, c'est-à-dire que tout le bâtiment est généré à chaque fois que l'on compile le programme. Toutefois, le processus de conception se fait habituellement sur plusieurs sessions de conception. Ainsi, le modèle de bâtiment doit pouvoir persister d'une session de conception à l'autre.

4. Les deux prototypes utilisent un langage différent. En effet, StAr utilise le C++ tandis que le DKM est programmé en Java. Un élément devait donc faire le pont entre ces deux langages.

La solution adoptée est l'utilisation de la base de données « Caché » (Alcade, 2006). Cette base de données permet :

1. L'utilisation du modèle objet, accélérant le développement d'applications;
2. Les données y sont persistantes
3. La communication avec les deux langages utilisés ici, soit le C++ et le Java.

Ainsi, le prototype complet utilisant les deux modules (StAr et DKM) comporte un troisième élément soit une base de données commune maintenue par « Caché ». Les deux modules communiquent avec Caché afin d'accéder aux données du bâtiment (Voir figure 28). Cette façon de faire facilite aussi l'implémentation individuelle des deux modules puisqu'ils peuvent être développés en parallèle.

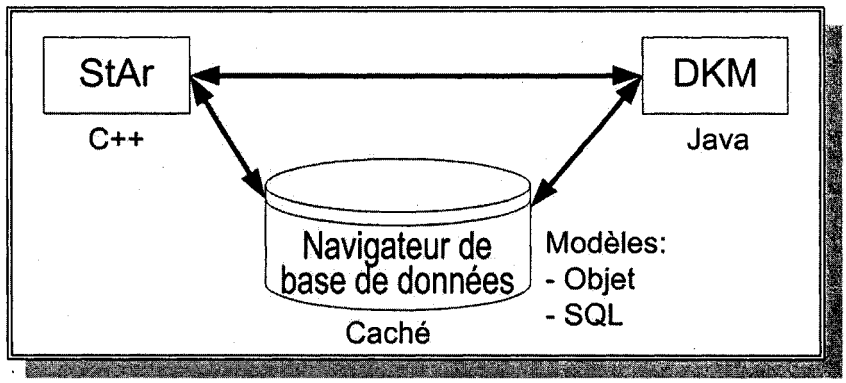


Figure 28 Architecture du prototype complet assistant la conception préliminaire de structures de bâtiments

L'implémentation du prototype du module de gestion des connaissances, réalisée à ce jour, s'est limitée à l'utilisation des éléments présents dans la base de données sans requérir d'analyses géométriques réalisées par StAr. La façon de traiter ces interactions n'a donc pas été abordée. Toutefois, pour ce genre d'interactions, il sera peut-être nécessaire de faire un lien direct entre les modules StAr et DKM.

StAr est déjà implémenté tel qu'expliqué à la sous-section 2.2.1 et continue à être amélioré. Ce module n'est pas l'objet de la présente recherche. Pour de plus amples informations sur l'implémentation de ce module, veuillez consulter Mora (2005). Le module DKM est expliqué en détail dans les sections suivantes faisant l'objet de cette recherche.

5.2. Structure des données

Les données sont structurées selon l'*approche descendante*. Ainsi, chacune des classes hérite de la classe générale **`BuildingEntity`** pour être subdivisée ensuite en classes plus spécialisées telles que la classe **`HorizontalSubsystem`**. C'est de cette dernière qu'héritent la classe **`SlabElement`** et les autres classes de sous-systèmes horizontaux spécifiques tels que ceux qui ont été implémentés dans le présent prototype (Voir figure 29).

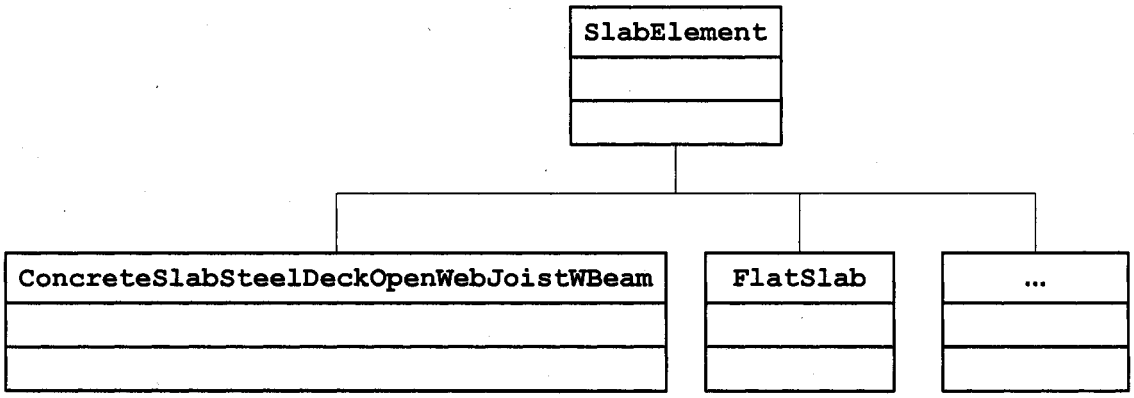


Figure 29 Hiérarchie des classes des entités structurales

Les classes de décisions héritent toutes de la classe abstraite **AbstractDecisionNode**. On y retrouve les méthodes abstraites **IsApplicable** et **Apply** ainsi que toutes les autres méthodes qui sont utilisées par tous les nœuds tels que la méthode **rollBack.** Ces dernières seront expliquées plus loin. Les classes de décisions spécifiques à chacun des sous-systèmes structuraux héritent de cette classe, chacune des méthodes abstraites y étant redéfinies. **Dec_FlatSlab** et **Dec_ConcreteSlabSteelDeckOpenWebJoistWBeam** sont deux exemples de ces classes spécifiques (Voir figure 30).

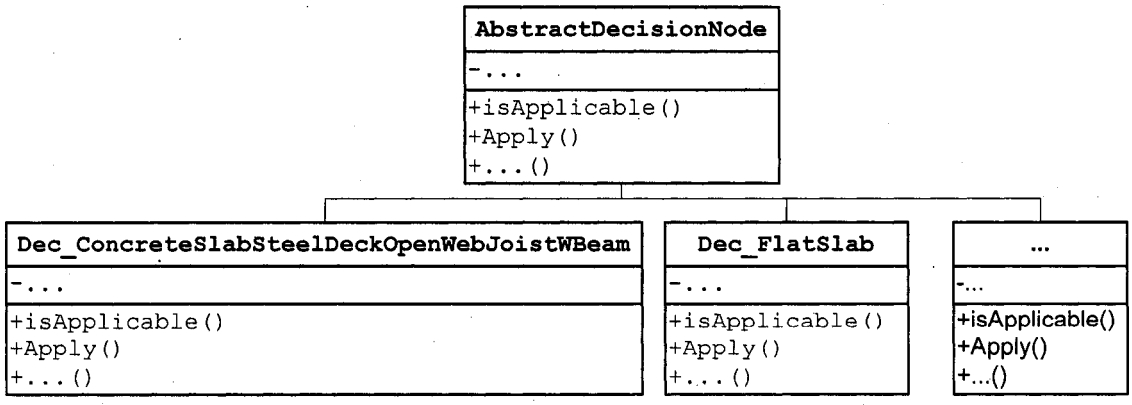


Figure 30 Hiérarchie des classes de décision

5.3. Module de gestion des connaissances « DKM »

Ce module est le principal élément dont traitera le présent chapitre. En effet, l'implémentation du prototype visé par cette recherche concerne particulièrement le module de gestion des connaissances.

5.3.1. Utilisation du Java comme langage de programmation

La programmation Java est un langage de haut niveau qui utilise uniquement la programmation orientée objet. La programmation orientée objet est importante, car elle permet à la programmation de coller à la réalité.

> « La conception orientée objet modélise les objets du monde réel, en d'autres termes crée un modèle à partir de la réalité. [...] et offre une manière plus naturelle et plus intuitive de visualiser le processus de conceptualisation, à savoir : la modélisation d'objets réels, de leurs attributs et de leurs comportements. » (Deitel & Deitel, 2002)

Le Java facilite l'interaction avec l'utilisateur notamment par l'exploitation de graphismes, d'images, d'animation, de l'audio et de la vidéo (Deitel & Deitel, 2002). Comme il a été expliqué au cours du chapitre 3, le volet de l'interaction avec l'utilisateur est très important dans ce projet de recherche. Le Java peut faciliter cette interaction.

Les logiciels conçus à partir de ce langage « s'accompagnent d'une portabilité complète permettant aux applications de fonctionner sans modification sur [...] différents types d'ordinateurs équipés de systèmes d'exploitation différents » (Deitel & Deitel, 2002). Cette qualité des programmes Java facilitera probablement certaines étapes futures de notre recherche. En effet, fonctionnant sous l'environnement Windows de Microsoft, il pourrait être plus facile de communiquer avec EsQUIsE (Voir sous-section 2.2.2), l'interpréteur d'esquisses architecturales, fonctionnant sous l'environnement Mac OS de Apple.

« Ce langage est certain de devenir le langage de premier choix dans l'implantation des applications destinées à l'Internet, l'intranet et aux logiciels pour appareils devant communiquer par l'entremise d'un réseau » (Deitel & Deitel, 2002). Il est envisageable

que dans une autre phase de développement, le prototype puisse fonctionner à partir d'Internet et être accessible par les bureaux voulant en faire l'utilisation.

Finalement, le Java est disponible gratuitement à partir du site Internet de la compagnie « Sun ». Ceci représente un attrait majeur pour les projets universitaires.

5.3.2. Architecture spécifique au DKM

L'architecture spécifique au DKM est composée d'une architecture en couche, c'est-à-dire que l'utilisateur n'a pas à interagir directement avec la base de données. Il interagit avec l'interface et le visualiseur seulement. Ces derniers activent les contrôles qui commandent les actions à prendre sur les entités. Ces entités sont ensuite conservées dans la base de données « Caché » (Voir figure 31).

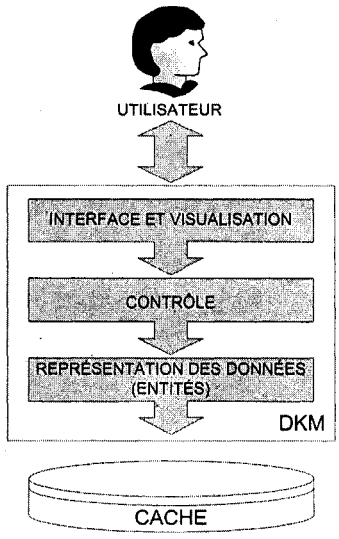


Figure 31 Architecture du module DKM

5.3.3. Paquetages utilisés

Trois *paquetages*[11] principaux et cinq autres spécifiques aux sous-systèmes structuraux horizontaux sont utilisés pour l'implémentation du prototype DKM. Il s'agit

---

[11] En programmation, conteneur qui, au moyen d'un mot-clé commun, regroupe des classes partageant des relations logiques (Office de la langue française, 2005).

d'un premier *paquetage*, appelé **CacheEntities** qui sert à interagir avec les entités dans « Caché ». Le deuxième, appelé **dkmApplication**, contient toutes les fonctions destinées au traitement de l'interface ainsi qu'une classe globale (**Globals**) qui permet d'y insérer des fonctions qui peuvent être utilisées par plusieurs autres classes. Finalement, les *paquetages* dont le nom débute par **knowledge** contiennent les classes spécifiques à chacun des nœuds des arbres de nœuds décisionnels. On peut ainsi retrouver le *paquetage* principal **knowledge** qui contient les connaissances pour les nœuds appartenant à tous les sous-systèmes horizontaux et d'autres *paquetages* spécifiques tels que **knowledge_ConcreteSlabSteelDeckOpenWebJoistWBeam** (Sous-système horizontal défini par une dalle de béton sur tablier métallique supporté par des poutrelles d'acier ajourées et poutre en W) contiennent les connaissances spécifiques pour un sous-système en particulier.

Tous les attributs utilisés pour l'implémentation sont présentés en annexe 5. Chacun des attributs pour un élément de dalle ainsi que ceux des classes de connaissances spécialisées des sous-systèmes horizontaux y est détaillé en français avec sa traduction anglophone.

5.3.4. Utilisation des nœuds décisionnels

Les nœuds décisionnels sont inspirés du modèle BENT (Fenves et coll., 2000; Gomez, 1998). Un nœud décisionnel représente les connaissances requises pour implémenter une étape de conception. L'application d'un nœud décisionnel peut être vue comme étant une décision menant à une solution de conception. Ces nœuds utilisent les heuristiques, tels que les systèmes experts (Voir les sous-sections 1.4.2 et 1.5.2) mais sans l'utilisation du moteur d'inférence. De plus, les nœuds décisionnels permettent le raisonnement à différents niveaux d'abstraction. Différentes fonctionnalités sont nécessaires pour satisfaire toutes les décisions à prendre. Certains nœuds décisionnels n'utilisent que les heuristiques tandis que d'autres nécessitent aussi

l'utilisation de dialogue, de tableaux (nombreux dans les connaissances recueillies), d'*infobulles*[12] et de pages HTML.

Tous les nœuds utilisent le principe de règle avec *partie gauche* et *droite* provenant des systèmes basés sur les règles utilisés entre autres par les systèmes experts. Afin de déterminer l'applicabilité d'un nœud, la fonction **`IsApplicable`** (Voir exemple dans la prochaine sous-section) vérifie si le nœud est applicable (équivalent de la *partie gauche* de la règle). Si le nœud est applicable, le nœud est montré en vert. Ensuite, si l'utilisateur décide d'appliquer la décision, en cliquant deux fois sur le nœud lorsqu'il est vert, la fonction **`Apply`** (équivalent de la règle droite) détermine les changements à réaliser sur l'entité et peut commander aussi différentes fonctionnalités qui seront expliquées dans les prochains paragraphes. Il est à noter que l'utilisateur remplace le moteur d'inférence utilisé en système expert. Une fois appliqué, le nœud devient rouge. Voici un exemple d'une règle **`Apply`** (voir autres exemples à l'annexe 7), en pseudo-code, appartenant au nœud #8 (sous-système horizontal du type dalle de béton sur tablier métallique supporté par des poutrelles d'acier ajourées, voir figure 39) :

```
Apply Dec_ConcreteSlabSteelDeckOpenWebJoistWBeam
{
   Si l'entité de bâtiment sélectionnée est un élément de dalle
      Conserver la version actuelle de l'entité
      Créer une nouvelle entité de dalle qui sera la version
      actualisée;
      Copier les valeurs de l'ancienne entité dans la nouvelle;
      Définir le type de l'entité à :
            "ConcreteSlabSteelDeckOpenWebJoistWBeam"
      Actualiser l'interface avec la nouvelle entité
Retourner l'élément de bâtiment modifié en fonction de la
décision}
```

La fonction **`Apply`** conserve l'ancienne version de l'entité pour permettre le retour en arrière sur les décisions prises. Ainsi, si l'utilisateur désire retourner en arrière sur une

---

[12] Élément d'un système d'aide contextuelle qui, à la demande de l'utilisateur, affiche de l'information sur les différentes parties d'une fenêtre, sous une forme qui rappelle un peu celle des bulles des bandes dessinées (Office de la langue française, 2005).

décision prise, il clique deux fois sur le nœud appliqué et la fonction **rollback** est appelée ramenant ainsi l'ancienne version de l'entité à l'état de version actuelle. Suivra dans les prochains paragraphes, une description de chacune des particularités des nœuds.

Les heuristiques sont très présentes dans la littérature sous forme de règles du pouce. Il est possible de décortiquer ces heuristiques sous la forme d'une *partie gauche* (**IsApplicable**) et d'une *partie droite* (**Apply**) tel que décrit précédemment.

Un exemple d'heuristique recueillie à partir des tables de Schodek (2004) (Voir annexe 2) permet de déterminer si les portées à couvrir peuvent l'être et ce de façon économique avec le sous-système horizontal inclus dans le nœud. Voici un exemple de pseudo-code représentant la fonction **IsApplicable** pour le nœud #8 (sous-système horizontal du type dalle de béton sur tablier métallique supporté par des poutrelles d'acier ajourées, voir figure 39):

```
IsApplicable Dec_ConcreteSlabSteelDeckOpenWebJoistWBeam
{
    Si l'entité de bâtiment sélectionnée est un élément de dalle;
        Si le cheminement des charges est « Unidirectionnel »;
            Si la longue portée est entre 3000 mm et 36000mm;
                Si la courte portée est entre 3000mm et 21000mm;
                        Retourner « Vrai »;
}
```

La fonction **IsApplicable** retourne « vrai » si toutes les conditions préalables sont respectées. Ceci rend alors le nœud applicable et donc sélectionnable par l'utilisateur.

Il n'est pas nécessaire d'indiquer dans la fonction **IsApplicable** que lorsqu'un sous-système est déjà sélectionné (groupe #4) les autres sous-systèmes ne sont plus applicables. Cette vérification est faite au niveau de la gestion de l'interface. Deux classes provenant du *paquetage* **dkmApplication** s'occupent de faire cette vérification. La classe **EntityColorFunction** vérifie s'il y a un nœud du groupe qui

est rouge (nœud appliqué) et si c'est le cas, colore les autres nœuds du groupe en gris. Ensuite, la classe **`EntityPickPlugin`** s'assure que les autres nœuds, qui ne sont plus applicables, ne sont pas sélectionnables par l'utilisateur.

Des dialogues sont souvent nécessaires afin de proposer des valeurs à utiliser à l'utilisateur ou tout simplement pour lui poser des questions nécessaires à la poursuite des décisions. Ces dialogues ont été implémentés tel que prévu dans le chapitre 3. Par exemple, pour déterminer le sens des poutrelles, il faut demander à l'utilisateur dans quel sens il désire les disposer. Il est toutefois possible de lui donner des conseils basés sur les connaissances à même cette fenêtre de dialogue. Pour ce genre d'interaction, une boîte de dialogue de type **`JOptionPane.showInputDialog`** est utilisée (Voir figure 32).

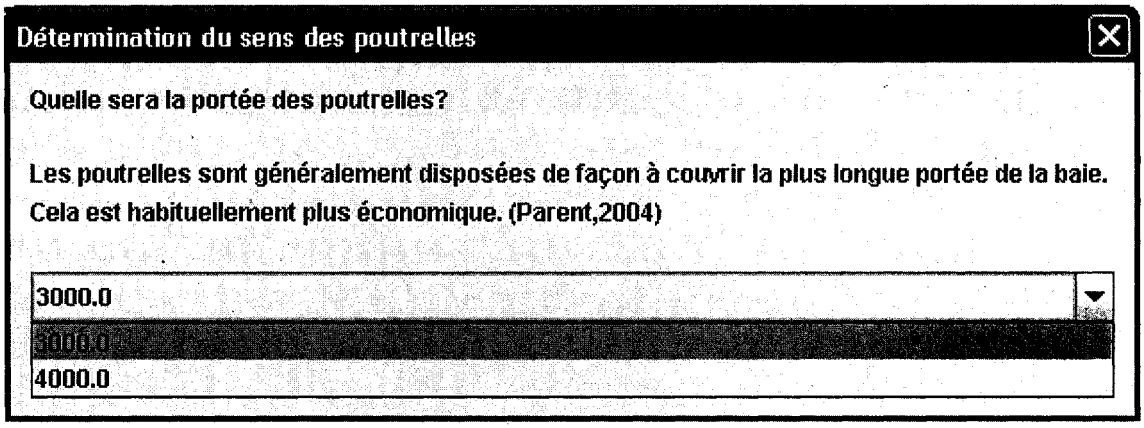


Figure 32 Exemple d'une boîte de dialogue utilisée pour déterminer le sens des poutrelles

Énormément de tableaux sont présents dans les connaissances en conception préliminaire de structures de bâtiments. Ces tableaux se trouvent dans différentes sources. Il est intéressant de pouvoir fournir à l'utilisateur, au temps opportun, les tableaux pertinents aux décisions à prendre. De plus, il est possible de faire un traitement de ces tableaux et de ne montrer que ce qui est pertinent en fonction du cas à résoudre. Un exemple de tableau pouvant être utilisé pour la conception préliminaire est montré au Tableau IV.

Tableau IV

Masse des poutrelles (en kg/m) en fonction de la charge pondérée et de la charge de service pour les poutrelles d'une portée de 4 mètres

Portée (m)	Profondeur de la poutrelle (mm)	Charge pondérée (kN/m) / Charge de service (kN/m)												
		4,5 / 3,0	6,0 / 4,0	7,5 / 5,0	9,0 / 6,0	10,5 / 7,0	12,0 / 8,0	13,5 / 9,0	15,0 / 10,0	16,5 / 11,0	18,0 / 12,0	19,5 / 13,0	21,0 / 14,0	22,5 / 15,0
4	200	7,8 / 105	7,8 / 79	8,4 / 73	8,8 / 64	10,3 / 65	11,5 / 64	12,8 / 65	14,4 / 65	15,8 / 65	17,3 / 64	18,8 / 64	20,4 / 65	22,1 / 64
	250	8,0 / 170	8,0 / 128	8,0 / 102	8,0 / 85	8,2 / 73	8,8 / 74	9,7 / 68	11,3 / 75	12,0 / 72	12,6 / 69	13,5 / 67	13,9 / 66	14,4 / 66
	300	9,6 / 200	9,6 / 200	9,6 / 183	9,6 / 153	9,6 / 131	9,6 / 115	9,6 / 102	10,3 / 96	10,6 / 90	12,4 / 98	13,4 / 95	13,4 / 88	13,7 / 86
	350	9,8 / 200	9,8 / 200	9,8 / 200	9,8 / 200	9,8 / 181	9,8 / 159	10,1 / 141	10,1 / 127	10,5 / 121	10,5 / 111	11,8 / 112	12,9 / 116	13,6 / 114
	400	9,9 / 200	9,9 / 200	9,9 / 200	9,9 / 200	9,9 / 200	9,9 / 200	10,3 / 187	10,3 / 168	10,3 / 153	10,4 / 140	10,9 / 135	10,9 / 126	12,0 / 128
	450	10,1 / 200	10,1 / 200	10,1 / 200	10,1 / 200	10,1 / 200	10,4 / 200	10,5 / 200	10,5 / 200	10,5 / 195	10,7 / 179	11,1 / 165	11,2 / 153	11,2 / 150
	500	10,3 / 200	10,3 / 200	10,3 / 200	10,3 / 200	10,6 / 200	10,6 / 200	10,7 / 200	10,7 / 200	10,9 / 200	10,9 / 200	11,3 / 200	11,3 / 191	11,6 / 178

(Canam - Solutions + Service, 2003)

Trois différentes solutions pour la présentation des tableaux à l'utilisateur ont été évaluées. Les exemples de traitement de tableaux seront faits à partir du Tableau IV.

1. Présenter les tableaux sous forme de nœuds; le meilleur choix pourrait être hachuré et les autres choix possibles en vert. Ici, pour le choix de la profondeur de la poutrelle, celle ayant la masse la moins élevée en fonction de la résistance de la poutrelle pourrait être celle qui serait hachurée (Voir figure 33).

2. Présenter à l'utilisateur des tableaux dynamiques avec lesquels il peut interagir; le tableau initial pourrait être traité de façon à ne pas montrer les parties de tableau inutiles. De plus, l'utilisateur pourrait prendre sa décision directement en sélectionnant son choix en cliquant sur la case appropriée dans le tableau. Le (ou les) meilleur choix pourrait être montré en rouge ou avec une autre façon mettant l'emphase sur les meilleurs choix (Voir figure 34).

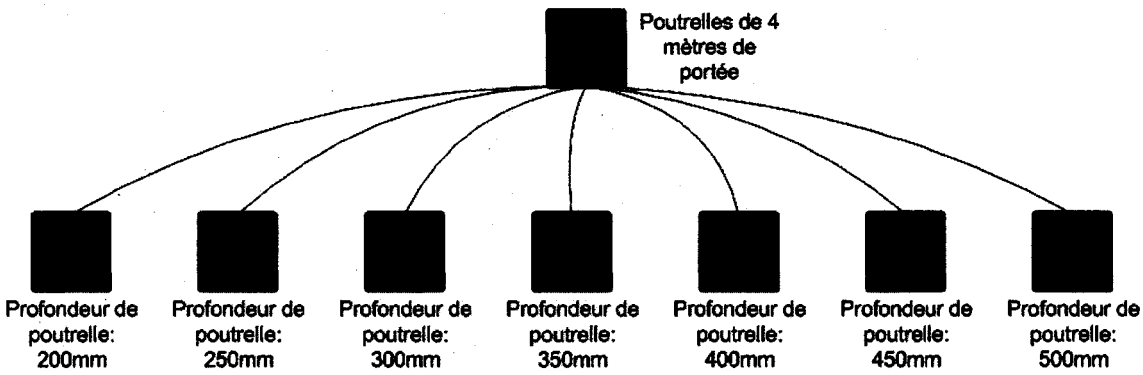


Figure 33 Présentation d'un tableau sous forme de nœuds

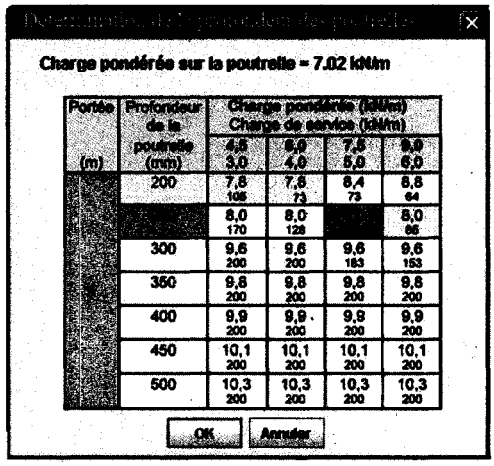


Figure 34 Exemple d'un tableau traitant dynamiquement les informations

3. Présenter à l'utilisateur les données tirées d'un tableau dans une boîte de dialogue avec un menu déroulant offrant les choix possibles. Les résultats peuvent être ordonnés de façon à rendre les choix les plus intéressants plus accessibles (Voir figure 35).

Chaque solution comporte des avantages et inconvénients. Il faut notamment considérer le temps d'implémentation nécessaire pour rendre possible l'utilisation des tableaux dans le prototype et la convivialité de l'interface dans la sélection des choix. Les avantages, inconvénients et temps approximatif pour l'implémentation de ces solutions sont détaillés dans le Tableau V.

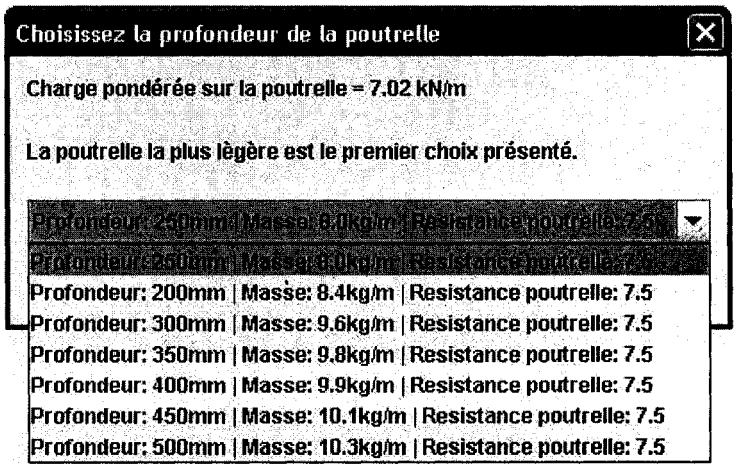


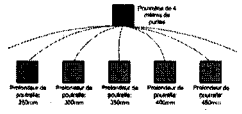
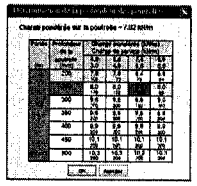
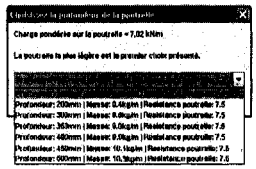
Figure 35 Dialogue présentant les éléments d'un tableau sous forme de menu déroulant

La troisième solution a été adoptée notamment pour son temps d'implémentation plus court que pour la solution #2. La solution #1 a été écartée étant donné le trop grand nombre d'options qui se présenteraient à l'utilisateur. De plus, il a été trouvé qu'il est possible d'implémenter les tableaux dans la base de données « Caché » et de faire par la suite des requêtes SQL sur ces tableaux. Cette façon de faire facilite grandement l'insertion des connaissances sous forme de tableaux puisqu'à partir d'un tableau en format Excel (ou autre), il est possible d'importer directement les données.

La personne qui modélise les connaissances a toutefois le choix entre la solution numéro 1 et 3. En effet, il est possible de créer autant de nœuds que souhaité pour un même niveau de décision ce qui correspondrait à tous les choix possibles présentés dans le menu déroulant de la solution numéro 3. Il faudrait noter cependant que le choix entre les deux solutions peut se baser sur le nombre de choix offerts à l'utilisateur. En effet, si le nombre de choix possibles est très élevé, étant donné que les nœuds non-applicables sont montrés en gris sur l'interface, trop de nœuds pourraient être montrés à l'utilisateur si la solution 1 est adoptée. Ainsi, tel que montré pour le choix des poutrelles, puisque 7 choix sont disponibles pour chaque longueur de poutrelles, il est préférable de présenter les poutrelles avec la solution 3.

Tableau V

Avantages, inconvénients et temps approximatif pour l'implémentation pour chacune des trois solutions pour le traitement des tableaux

IDENTIFICATION DE LA SOLUTION	- SOLUTION 1 -	- SOLUTION 2 -	- SOLUTION 3 -
			
TEMPS POUR IMPLÉMENTER	0 jour	Très long (durée indéterminée)	1.5 jours (approximatif)
AVANTAGES	Plus intuitif pour l'utilisateur du prototype étant donné que le reste du prototype utilise en majeure partie les nœuds.	Tableau bien organisé; Intuitif pour les ingénieurs puisque cela ressemble à leur méthode habituelle.	Facile à implémenter; Boîte de dialogue existante en Java (`OptionPane`, librairie swing, Java); Possibilité de classer les solutions.
INCONVÉNIENTS	Énormément de classes à implémenter; Pas intuitif pour les ingénieurs comparativement aux tableaux. Peut y avoir trop de nœuds; difficile à lire	Difficulté d'implémentation liée à la création d'une nouvelle interface qui permettrait l'interaction entre l'utilisateur et le tableau.	Moins bien organisé qu'un tableau; Les titres de colonnes doivent être répétés; Ce n'est pas possible de montrer le tableau lui-même.

Les *infobulles* servent à dispenser des évaluations rapides en fonction des décisions prises. Elles sont consultables en passant la souris par-dessus le nœud, cela appelle

alors la classe **KnowledgeTooltip**, présente dans le *paquetage* dkmApplication qui demande à la classe du nœud les informations à écrire dans l'*infobulle*. Si aucune information n'a été entrée dans la classe du nœud, la méthode de la classe **AbstractDecisionNode** donne des informations de base sur le nœud tel son nom (Voir exemple d'*infobulle* à la figure 36).

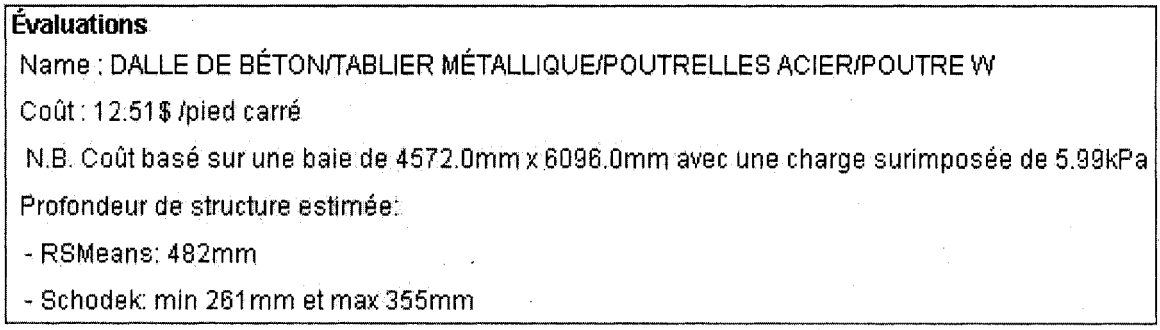


Figure 36 Exemple d'*infobulle* utilisée pour fournir une évaluation

Des pages HTML, équivalentes aux fenêtres pop-up présentées au chapitre 4, sont utilisées pour donner des conseils d'une façon plus complète que ceux qui peuvent être donnés par des *infobulles*. Ces pages HTML sont consultables en cliquant à droite avec la souris sur le nœud. Si une page HTML y est attachée, la classe **KnowledgePickPlugin** appelle la méthode **getInfoURL**. La page HTML s'ouvre alors. Plusieurs exemples de ces pages HTML sont présentés en annexe 8.

L'implémentation des nœuds sous forme d'arbres a été faite avec la librairie logicielle JUNG (JUNG: Java Universal Network/Graph Framework, 2006), une librairie existante. Cette librairie est extensible et disponible en *code source libre* pour la modélisation, l'analyse et la visualisation de données qui peuvent être représentées graphiquement sous forme d'arbres ou de graphes. Écrite en Java, la librairie JUNG augmente les capacités d'interface utilisateur du langage Java (Java API). Deux exemples d'utilisation de la librairie JUNG sont présentés aux figures 37 et 38. Le premier exemple est le logiciel Grape aidant à visualiser et explorer le graphique d'importation d'un programme. Le deuxième exemple est une application pour la gestion et l'analyse

de réseaux sociaux (contacts, relations, ressources, compétence et autres informations).

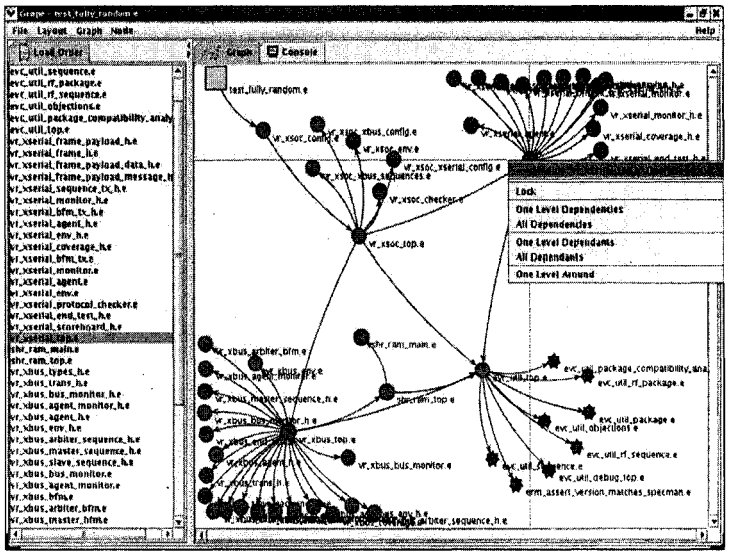


Figure 37 Utilisation de la librairie JUNG pour le logiciel « Grape » (AMIQ Consulting, 2005)

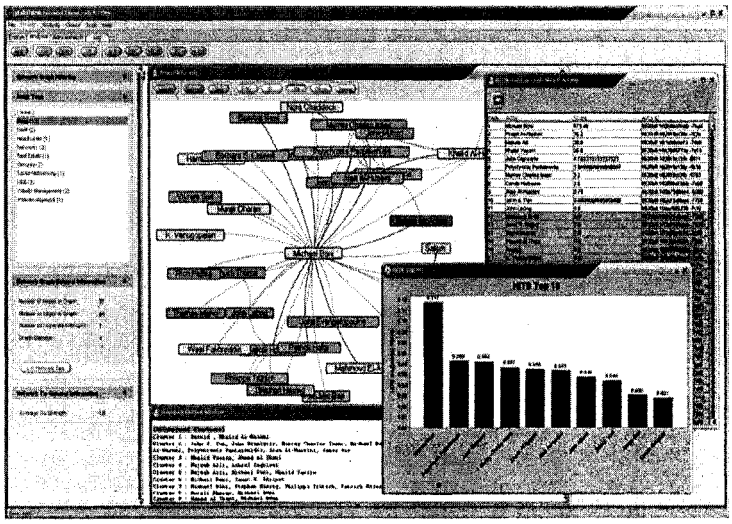


Figure 38 Utilisation de la librairie JUNG pour l'application « shiftTHINK Connect »
(shiftTHINK, 2006)

Pour ajouter une connaissance, une classe d'un nœud standard existe recueillant les principales composantes qui peuvent être nécessaires à l'implémentation du nœud. En copiant le nœud standard, on obtient une nouvelle classe contenant les méthodes **`Apply`** et **`isApplicable`** déjà amorcées. Il faut y ajouter les conditions d'applications du nœud dans la méthode **`isApplicable`** et les impacts de la décision (changements aux attributs, demande d'informations à l'utilisateur et autres) dans la méthode **`Apply`**.

En résumé, les étapes logicielles pour la génération d'un nœud décisionnel sont les suivantes :

1. Créer un nœud "JUNG" vide;
2. Implémenter les méthodes nécessaires aux nœuds (**`Apply`**, **`isApplicable`** et autres)
3. Encapsuler le nœud décisionnel qu'on désire représenter à l'intérieur du nœud "JUNG";
4. Répéter l'opération pour chacun des nœuds décisionnels.

Une fois que chacun des nœuds est créé, pour constituer l'arbre décisionnel, il faut lier chacun des nœuds à leurs prédécesseurs et successeurs. Afin de faciliter cette opération, chacun des niveaux de décisions compose un groupe de nœuds, ou plusieurs sous-groupes. Par exemple, tous les sous-systèmes horizontaux unidirectionnels (représentés par plusieurs nœuds), font partie du groupe numéro 4 (Voir figure 39), tandis que les sous-systèmes horizontaux bidirectionnels composent le groupe numéro 5. Ainsi, s'il faut créer un lien entre un nœud et un groupe de nœuds, cela peut se faire beaucoup plus rapidement que s'il avait fallu créer les liens individuellement. Chacun des numéros de groupe utilisés pour l'implémentation des éléments de base est défini à l'annexe 4.

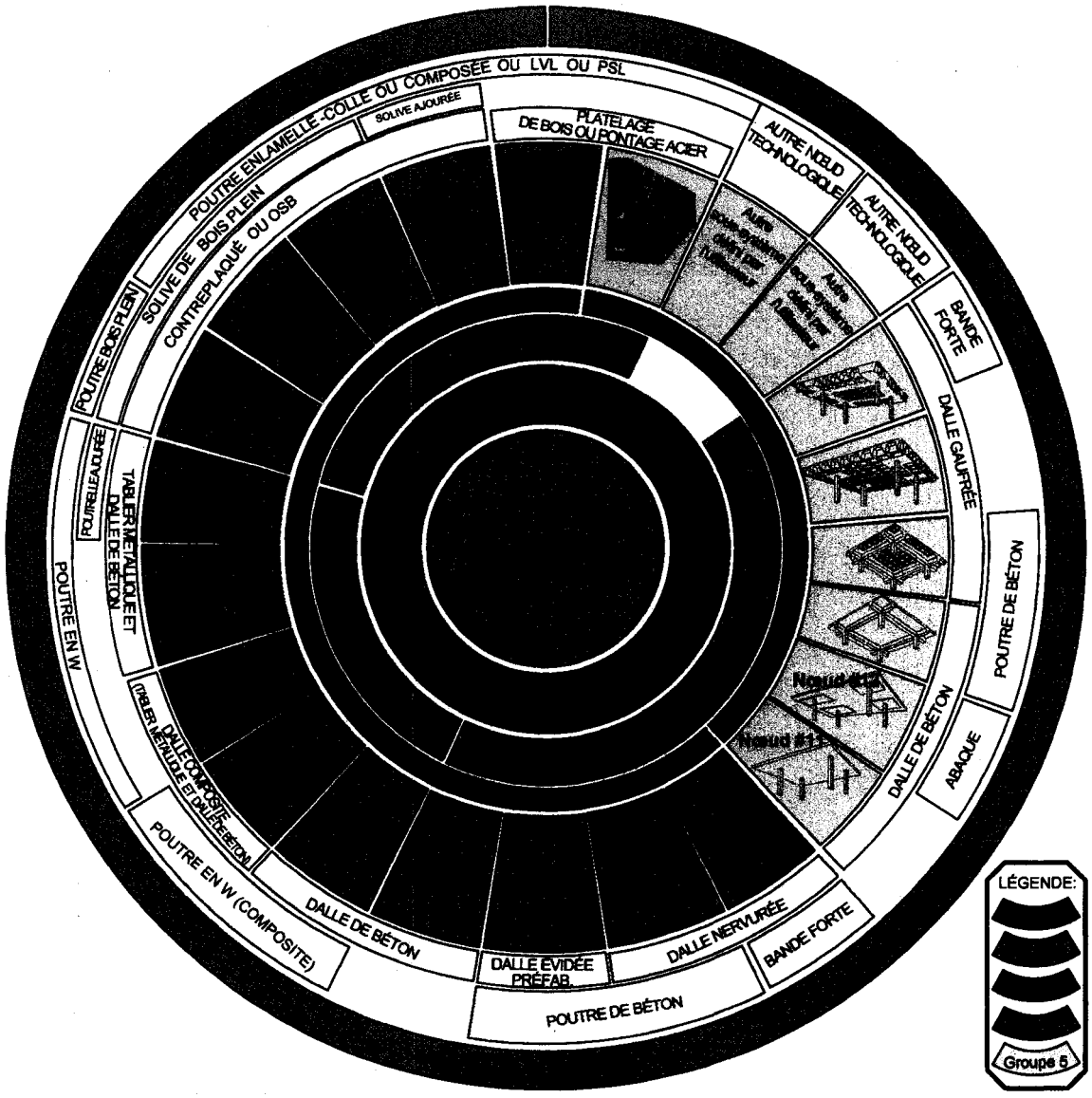


Figure 39 Définition des groupes et numéros de nœuds utilisés pour l'implémentation des liens entre les nœuds

En plus de faciliter la définition des liens entre les nœuds, cela facilite aussi la vérification de l'applicabilité des nœuds lorsqu'il n'y a qu'un groupe pour un niveau de décision. Cette notion sera expliquée davantage plus loin.

Quatre différents types de liens peuvent être définis de façon à pouvoir pallier à toutes les situations que l'on peut rencontrer dans les arbres de nœuds décisionnels développés soit :

1. Un nœud vers un nœud :

   Par exemple: Acier (nœud #3) à Unidirectionnel (nœud #6);

2. Un nœud vers un groupe de nœuds :

   Par exemple : Unidirectionnel (nœud #6) aux sous-systèmes unidirectionnels (gr.#4) ;

3. Un groupe de nœuds vers un nœud :

   Par exemple : Épaisseur de dalle de béton de 115 à 200mm (Groupe #15) vers « Détermination du sens des poutrelles » (nœud # 124);

4. Un groupe de nœuds vers un groupe de nœuds :

   Par exemple : Profondeur de tablier métallique de 38 à 76 mm (groupe #10) aux types de tablier métallique 22 à 18 (groupe#12).

Les liens peuvent être définis à l'aide des fonctions « **`addFollowingNode`** » pour indiquer quel nœud suit un autre nœud ou groupe de nœuds et similairement, « **`addFollowingGroup`** » pour indiquer quel groupe suit le nœud ou le groupe de nœuds.

5.3.5. Présentation de l'interface du prototype

L'interface graphique des arbres de connaissances tels que représentés dans le chapitre 4 n'a pas été possible pour ce prototype, car l'implémentation de l'interaction entre l'utilisateur et ces graphiques aurait été trop longue et compliquée à réaliser. De plus, il était souhaité de ne montrer à l'utilisateur que les nœuds applicables, afin de clarifier les choix possibles. Ainsi, il devenait encore plus complexe de traiter l'interaction avec l'utilisateur.

5.3.5.1. Apparence de l'Interface

L'interface utilisée pour le prototype est basée sur l'interface envisagée. À ce niveau, le seul grand changement apporté a été le repositionnement des fenêtres « Historique de design » et « Informations sur l'élément » à l'horizontale, dans le bas de l'interface

(Voir figure 40). Cela été fait pour laisser davantage de place à l'arbre de décisions qui a tendance à s'étendre horizontalement.

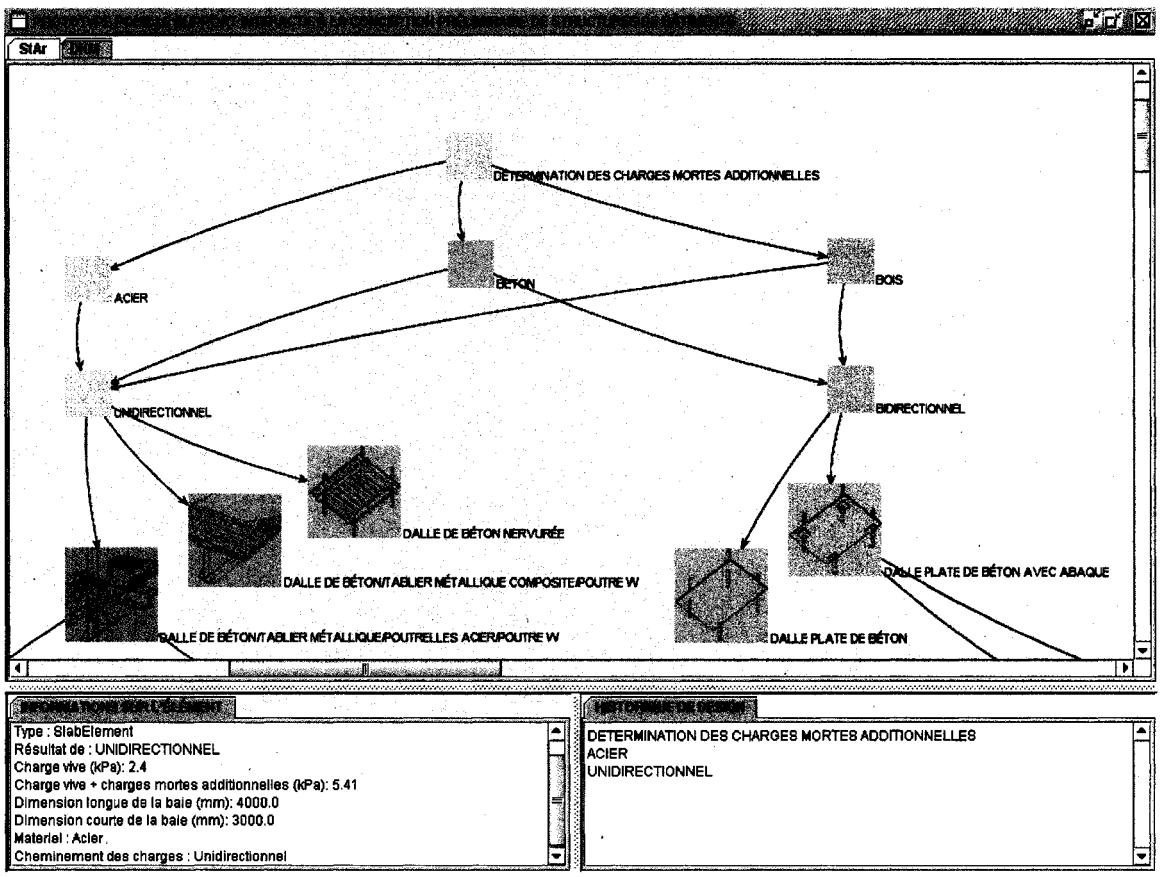


Figure 40 Interface générale implémentée de l'espace de travail en mode DKM

5.3.6. Détails de l'implémentation partielle du prototype

Les attributs qui ont été utilisés pour l'implémentation des sous-systèmes se retrouvent à l'annexe 5. Chacun des attributs y est indiqué en français et en anglais. L'implémentation s'est toutefois faite en anglais.

Deux exemples de classes qui ont été implémentées sont montrés en annexe 7. Le code présent dans ces deux exemples montre l'utilisation des méthodes **`isApplicable`** et **`Apply`**, l'utilisation de dialogues et des requêtes SQL.

5.3.7. Changements effectués au fonctionnement envisagé

Plusieurs fonctionnalités ont dû être simplifiées afin d'avoir un prototype fonctionnel dans le temps voulu :

1. Les arbres de connaissances développés n'ont pu être directement utilisés étant donné la difficulté d'implémentation de l'interaction avec l'utilisateur avec une telle interface;
2. Il était souhaité que les arbres de connaissances ne montrent que les nœuds décisionnels qui sont applicables pour ainsi permettre de montrer plus clairement toutes les possibilités qui s'offrent à l'utilisateur. Ainsi, les nœuds qui deviennent inapplicables au cas de conception, sont tout simplement supprimés de l'interface. Ce n'est pas le cas dans cette version du prototype encore une fois pour permettre d'avoir un prototype fonctionnel dans le temps voulu, mais il pourrait être possible d'ajouter cette fonctionnalité dans une itération subséquente.
3. Il était envisagé que l'utilisateur puisse voir les conséquences d'une décision sur tous les autres niveaux de décision de l'arbre de nœuds décisionnels. Ainsi, tous les nœuds qui peuvent être appliqués en fonction des décisions déjà prises, peu importe les décisions futures de l'ingénieur, sont montrés en vert. Cela pourrait accélérer l'exploration de solutions.
4. Les conseils fournis à l'ingénieur soit par l'intermédiaire d'*infobulles* ou par la consultation de pages HTML ne sont pas reportés dans l'historique de design de manière à ce que l'ingénieur puisse reconstituer la raison pour laquelle il aurait pris certaines décisions. Cela pourrait être une fonctionnalité ajoutée subséquemment.
5. Pour le retour en arrière sur des décisions déjà prises, il avait été prévu que cela puisse se faire directement en cliquant au niveau de la décision à partir de laquelle on voulait changer le cours des décisions. Toutefois, cela s'est avéré trop complexe pour cette première version du prototype.