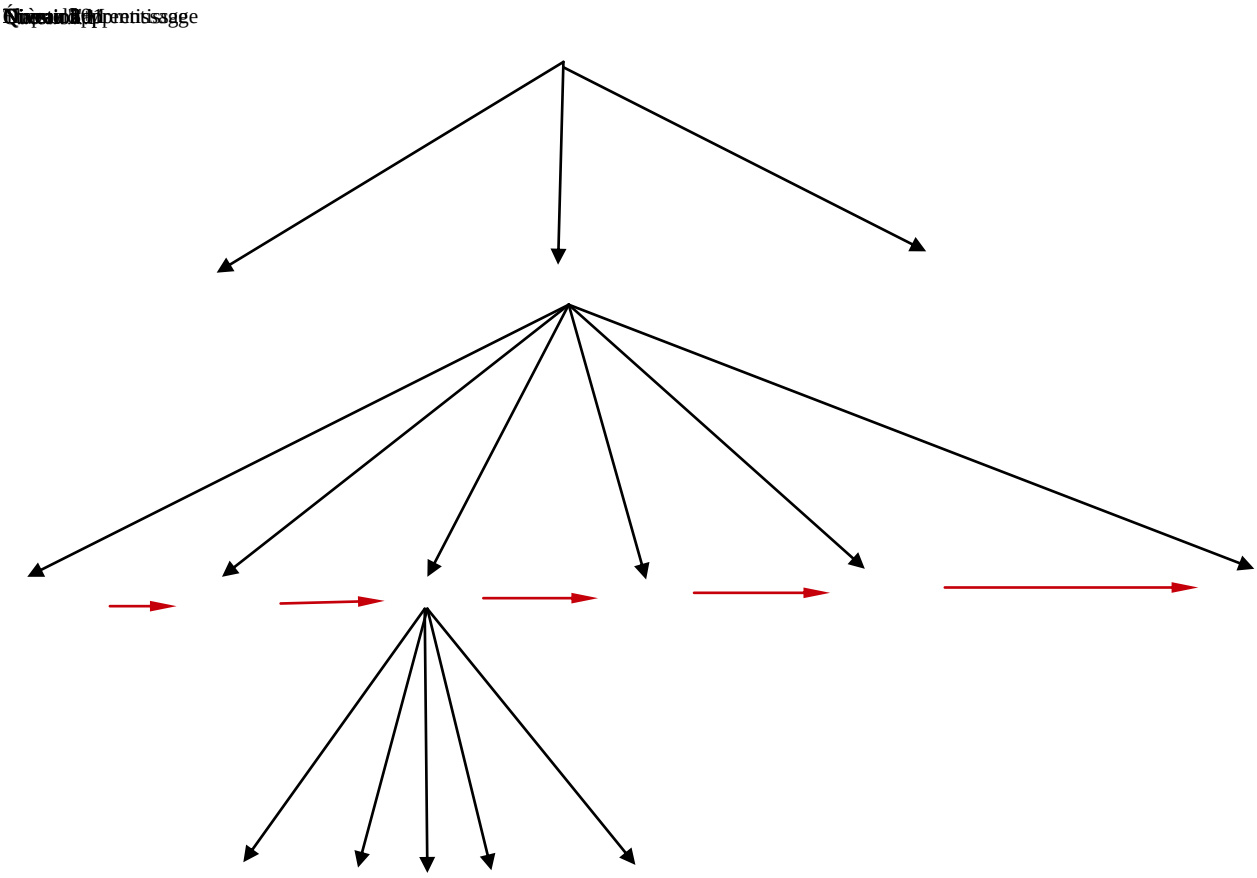
Projet ALSO
Prototype d'adaptation automatique de contenus pédagogiques

1- Introduction

Le but est de développer un outil d'apprentissage de la lecture et de l'écriture en N thèmes d'apprentissage. Chaque thème est composé d'étapes (ou activités) d'apprentissage. Il y a environ 70 étapes. Chaque étape contient I niveaux (30 environ, de difficulté croissante), et chaque niveau contient environ M questions. Le choix d'une question est aléatoire, il peut donc arriver que la même question soit posée plusieurs fois, mais cela permet de toujours avoir une possibilité de question à poser.

En cas de mauvaise réponse d'un élève à une question, une ou plusieurs adaptations peuvent être réalisées afin d'aider ce dernier. Les adaptations peuvent concerner la taille du texte, l'espace entre les mots, les couleurs, la police de caractères, les graphiques,… Il y a environ 60 adaptations possibles.



This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

2- Stratégie de base de gestion de l'apprentissage adaptatif

Il s'agit d'une stratégie fixée, propre à l'application. A chaque niveau correspond un ensemble de questions :

- Si l'élève répond correctement à une question d'un niveau i, alors le but est atteint, et on passe au niveau i+1. On pourra aussi parfois baisser l'intensité d'une adaptation passée et toujours active.
- Si l'élève répond incorrectement à une question, alors :
  - on continue sur la même question jusqu'à ce que l'enfant trouve la bonne réponse (question à choix multiple), sinon le système donne la réponse après M essais (notamment dans le cas où la réponse n'est pas proposée).
  - Sachant qu'une mauvaise réponse peut être une étourderie, on pose une autre question du même niveau pour s'en assurer. Si la seconde réponse est exact alors c'était bien une étourderie et on poursuit le processus avec une question plus difficile, sinon on propose une adaptation, s'il en reste, tant que le niveau n'est pas réussi. Pour résumer, on ne propose pas d'adaptation à la 1ère erreur, mais s'il y a 2 erreurs de suite.
  - on propose au plus K adaptations successives pour que l'élève réponde correctement et passe le niveau i : après avoir proposé K adaptations non pertinentes à un niveau i (donc à chaque fois 2 mauvaises réponses successives suite à une adaptation), on redescend provisoirement vers le niveau i-1 (si i=1, on reste au niveau 1) avec l'adaptation en cours, afin de remettre l'élève en confiance, puis on remonte au niveau i avec une nouvelle adaptation.
  - A chaque étape d'apprentissage, lorsque l'on propose une adaptation, on va en général avoir plusieurs choix possibles. Les points évoqués dans les chapitres 4.1 et 4.2 permettent de faire un choix.
  - Entre différents niveaux ou étapes, il est possible de cumuler des adaptations.

3- Les adaptations

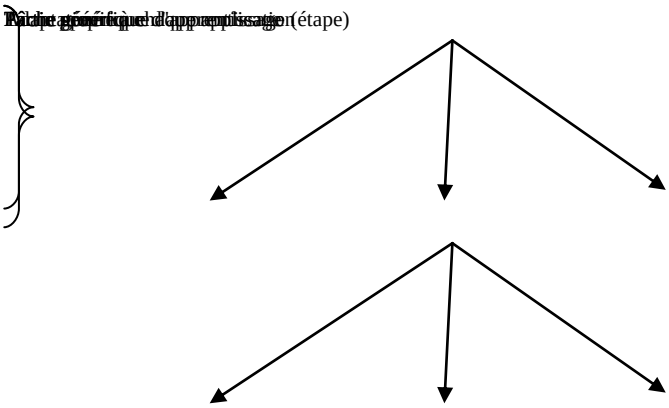
3.1- Modélisation

Chaque étape d'apprentissage est associée à des adaptations propres.
Plus précisément, une étape (ou activité) est relative à une tâche d'apprentissage spécifique (par ex « faire la différence entre texte et image »).
Cette tâche spécifique est en fait composée de tâches génériques d'apprentissage, et chaque tâche générique peut éventuellement poser des problèmes si l'élève a certains troubles, donc nécessiter des adaptations (non pas toutes les adaptations liées aux troubles concernées, mais seulement celles qui sont également liées à la tâche générique d'apprentissage).

Deux modélisations sont possibles (sous la forme de taxonomie) et diffèrent selon le niveau de détail.

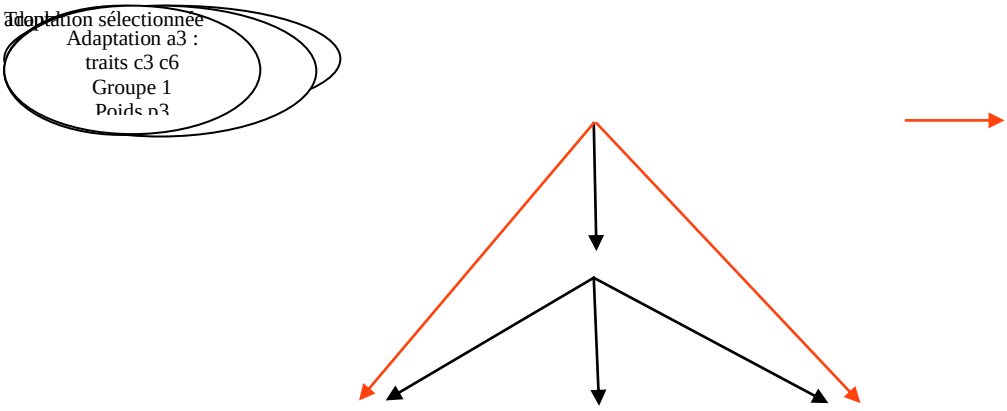
This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

La première modélisation relie directement une tâche générique d'apprentissage à des adaptations :



La seconde modélisation est plus précise car elle passe par le niveau intermédiaire des troubles, mais nécessite plus de travail.

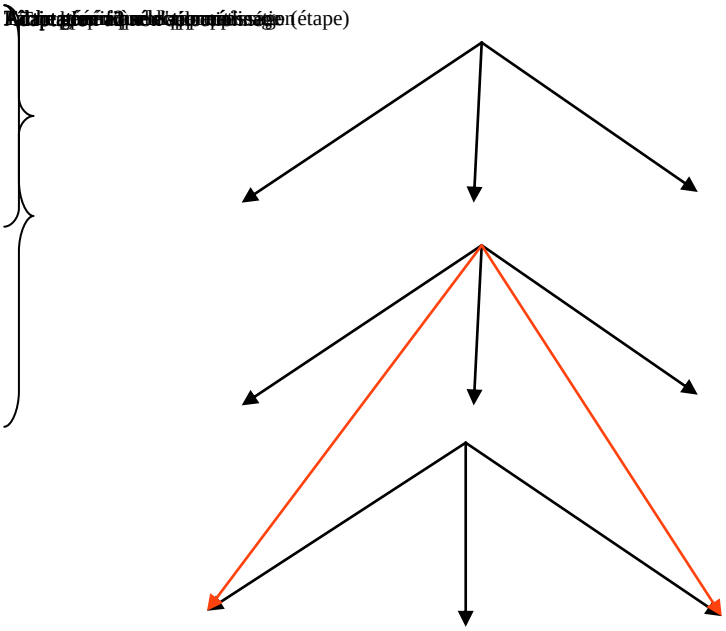
Une tâche générique d'apprentissage est liée à certains troubles, dans le sens où ces troubles peuvent poser des problèmes pour réaliser cette tâche d'apprentissage.
Par ailleurs, un trouble est liée à un ensemble d'adaptations possibles (ce travail a déjà été réalisé).
Mais dans le cadre d'une tâche générique d'apprentissage, seules certaines adaptations liées à ce trouble sont pertinentes. Une possibilité est de caractériser en termes de traits les tâches génériques d'apprentissage et également les adaptations, de façon à pouvoir rapprocher une tâche générique d'apprentissage et une adaptation, et donc pouvoir sélectionner les adaptations en rapport avec une tâche générique (c'est-à-dire avec au moins un trait en commun) :



This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Le travail à faire est donc de définir les traits caractéristiques de chaque tâche générique et de chaque adaptation.

Pour le reste, la seconde modélisation est semblable à la première :



La seconde modélisation permet aussi un choix d'adaptation basé sur un vote (cf chapitre 4.2).

A travers ces deux modélisations, on a d'une part une partie spécifique à chaque application, où une étape est définie en termes de tâches génériques d'apprentissage (cette partie spécifique est la même dans les deux modélisations), et d'autre part une partie générique (indépendante de l'application) où chaque tâche générique d'apprentissage est associée de façon directe (première modélisation) ou indirecte (seconde modélisation) à des adaptations.

Il est décidé d'utiliser la seconde modélisation.

Le stockage persistant de ces connaissances peut se faire de diverses manières :
- fichier XML ou JSON
- base de données relationnelle
- base de données orientée graphe (par ex Neo4J ou OrientDB)

Etant donné le caractère relationnel des données (cf graphes ci-dessus, relations entre le poids d'une

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

adaptation et une étape d'apprentissage, les influences possibles entre adaptations,…), il est décidé d'utiliser une base orientée graphe Neo4J.

3.2- Réduction du nombre d'adaptations possibles

Le nombre d'adaptations possibles pour chaque étape peut être non négligeable, et si tel est le cas il ne faut pas passer en revue toutes les adaptations possibles et les proposer successivement à un élève jusqu'à trouver celle qui est adéquate, car cela peut détourner l'élève de la tâche d'apprentissage. Il faut proposer rapidement à chaque élève une adaptation adéquate.

Il existe des possibilités pour réduire le nombre d'adaptations possibles :

- réaliser des tests de détection de certains troubles. Il est possible de faire passer à chaque élève quelques tests, préalables à la séance d'apprentissage, et destinés à détecter certains troubles et identifier des adaptations pour pallier ces troubles. Ces adaptations sont sures, et sont donc proposées à l'élève pour l'ensemble de l'activité pédagogique qui suit.
- cumuler certaines adaptations, et ne proposer que l'adaptation cumulée (par ex on propose « a1 et a2 », mais on ne propose ni a1, ni a2), ou bien proposer d'abord l'adaptation cumulée.
- on remplace un niveau (notamment le premier niveau d'une étape) comportant un nombre non négligeable d'adaptations par plusieurs niveaux comportant peu d'adaptations : par ex si on a 4 adaptations possibles sur un niveau, on peut remplacer ce niveau par deux autres (un niveau qui concerne 2 adaptations, et un autre niveau qui concerne les 2 autres adaptations). Il s'agit donc de créer des niveaux intermédiaires avec peu d'adaptations possibles.
- on ajoute un niveau (notamment avant le premier niveau d'une étape comportant un nombre non négligeable d'adaptations) comportant peu d'adaptations : par ex si on a 4 adaptations possibles sur un niveau, on peut ajouter devant ce dernier un autre niveau qui concerne 2 adaptations, ce qui va réduire le nombre d'adaptations possibles pour les niveaux suivants. Il s'agit donc de créer si nécessaire ce niveau avec peu d'adaptations possibles.
- ajouter une question (notamment avant la première question d'un niveau) avec un nombre réduit de réponses possibles (par rapport à la question d'après), et donc également avec un nombre réduit d'adaptations possibles.

4- Gestion des adaptations à partir de connaissances explicites

4.1- Calcul de l'ensemble des adaptations possibles

De manière générale, si on ne tient pas compte du cas "étourderie" (donc pas d'adaptation), et du cas "modification de l'intensité d'une adaptation précédente et pertinente" (voir 4.2), alors :

- si on a 2 adaptations de base a1 et a2 (des adaptations isolées), alors les choix possibles sont :
  - a1
  - a2
  - a1 et a2 (on peut proposer une adaptation cumulée au lieu d'une adaptation isolée. Une adaptation peut donc être simple ou cumulée. Par la suite, le terme adaptation est vu de cette

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

manière et englobe donc les 2 sous types)

- si on a 3 adaptations de base a1, a2 et a3, alors les choix possibles sont :
  - a1
  - a2
  - a3
  - a1 et a2
  - a1 et a3
  - a2 et a3
  - a1 et a2 et a3

Et ainsi de suite. Avec n adaptations de base, le nombre de choix possibles est S :

$$S = \sum_{k=1}^{k=n} C_n^k$$

Remarque : En théorie, si le nombre d'adaptation augmente, le nombre de choix possibles augmente dans des proportions encore plus fortes. On se limite à n = 3 adaptations de base. Au delà la combinatoire est trop élevée, il y a trop d'adaptations possibles. Il faut des moyens pour trouver la bonne adaptation rapidement, sans passer en revue toutes les possibilités, par ex via un calcul de poids pertinent et dynamique.

Par ailleurs, à chaque étape d'apprentissage, les adaptations peuvent être organisées en groupes et sous groupes. Par ex si on a 2 groupes {a1, a2} et {a'1, a'2, a'3}, avec les sous groupes {a1,a2} et {a'1,a'2} et {a'3}, alors les adaptations possibles sont :

- a1 et a2
- a'1 et a'2
- a'3
- a'1 et a'2 et a'3

Ceci si on ne cumule pas d'adaptation entre groupes. Si on cumule entre groupes, alors il faut ajouter :

- a1 et a2 et a'1 et a'2
- a1 et a2 et a'3
- a1 et a2 et a'1 et a'2 et a'3

Il a été décidé de ne pas cumuler d'adaptation entre groupes.

Pour la modélisation, il suffira d'ajouter 2 traits "groupe" et "sous groupe" pour chaque adaptation (cf figure chapitre 3.1)

Aspects dynamiques :

- Si une adaptation ne permet pas de passer un niveau, elle n'est donc pas adéquate, on ne doit donc pas la proposer à nouveau ni dans ce niveau ni dans l'étape associée (mais on peut proposer cette adaptation cumulée avec une autre), mais on peut la proposer à nouveau ultérieurement, lors d'une autre étape, et avec un poids plus faible
- Si une adaptation pertinente a été proposée, on peut ensuite progressivement la baisser, toutes les N bonnes réponses

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- Si une adaptation pertinente a été proposée, puis progressivement baissée jusqu'au min sans erreur de l'élève (donc l'adaptation devient inactive), on peut tout de même la proposer à nouveau lors d'une autre étape.
- Si une adaptation pertinente a été proposée, puis progressivement baissée jusqu'à une intensité optimale (cf 4.2), alors elle reste tout de même active, et on peut tenter d'abaisser à nouveau cette adaptation, toutes les P bonnes réponses (P > N) Mais on effectue ces baisses d'intensité optimale une fois que toutes les adaptations sont soient inactives soient à l'intensité optimale.
- Si une adaptation pertinente a été proposée, puis progressivement baissée, puis remontée avec une erreur (cas d'incohérence), alors elle reste active (à l'intensité remontée d'un cran) et on peut tenter de la baisser à nouveau toutes les N bonnes réponses
- s'il n'y a plus d'adaptation possible pour une question, alors cela signifie que l'exercice proposé n'est pas adapté à l'enfant, donc on sort de l'application.

Il faudra donc une procédure de calcul de l'ensemble des adaptations possibles prenant en compte toutes ces contraintes.

L'ensemble des adaptations possibles pour une baisse d'intensité est l'ensemble des adaptations pertinentes et encore actives.

4.2- Choix d'une adaptation

De manière générale, si on doit choisir une adaptation parmi un ensemble, on a 3 alternatives pour faire ce choix :

- choix fonction d'un poids préalablement associé à chaque adaptation (poids propre à chaque étape d'apprentissage, qui peut être liée par exemple à la fréquence d'utilisation d'une adaptation, et peut être fourni par un expert). Il faut aussi déterminer le poids d'une adaptation cumulée
- choix fonction d'un vote sur les adaptations liées à chaque tâche générique d'apprentissage au sein d'une même tâche spécifique d'apprentissage. Intuitivement, étant donné qu'une tâche spécifique d'apprentissage est composée de plusieurs tâches génériques d'apprentissage, elles-mêmes reliées à des troubles, si plusieurs de ces troubles sont liées à une même adaptation, alors on peut penser que cette adaptation est pertinente. Ce calcul est possible avec la seconde modélisation des adaptations (chapitre 3.1).
- tenir compte de l'historique des adaptations grâce à des connaissances métier, car il peut y avoir des influences entre adaptations, de sorte qu'une adaptation réalisée peut influencer le choix d'une prochaine adaptation : influence causale de sorte qu'une adaptation proposée implique qu'une autre adaptation peut ensuite être proposée, influence temporelle (il peut exister un ordre par défaut entre certaines adaptations).

Il serait donc utile de modéliser ces influences entre adaptations, si possible sans passer par les troubles, car la première modélisation (chapitre 3.1) n'utilise pas les troubles, et la seconde modélisation utilise les troubles mais il n'est pas toujours possible de savoir de quel trouble est issue une adaptation, car une étape d'apprentissage est liée à au moins une tâche générique d'apprentissage et un même tâche générique peut être associée à plusieurs troubles et certains de ces troubles peuvent avoir des adaptations en commun.

- Si une adaptation ne permet pas de passer un niveau, elle n'est donc pas adéquate, on ne doit

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

donc pas la proposer à nouveau ni dans ce niveau ni dans l'étape associée (mais on peut proposer cette adaptation cumulée avec une autre), mais on peut la proposer à nouveau ultérieurement, lors d'une autre étape, et avec un poids plus faible

- suite à une baisse de niveau, si on a une bonne réponse, alors on cumule l'adaptation courante avec une adaptation d'un autre groupe. Si on a une mauvaise réponse, on choisit une adaptation d'un autre groupe. Mais s'il n'y a pas d'autre groupe , on choisit une adaptation du même groupe.

Modification de l'intensité de l'adaptation :

- Intensité de l'adaptation : Par défaut on proposera l'intensité la plus forte, et on diminuera progressivement cette intensité (toutes les N bonnes réponses) dans la suite du processus d'apprentissage afin de revenir vers des normes sociales. Mais il peut se produire le cas où, lorsque l'intensité devient trop basse, l'élève commet à nouveau des erreurs. Dans ce cas, il faut remonter l'intensité d'un cran : c'est l'intensité "optimale" pour l'élève.
- La stratégie est donc de modifier l'intensité d'une des adaptations précédentes pertinentes (issues des niveaux et étapes précédentes) de la façon suivante :
  - si on a N bonnes réponses successives (sans adaptation préalable), on peut alors baisser d'un cran l'intensité d'une adaptation précédente, ceci progressivement jusqu'au minimum. Dans le cas d'une adaptation cumulée, on abaisse progressivement l'intensité de chaque adaptation de base qui la compose. A priori on abaisse successivement chacune jusqu'à la norme (on abaisse une jusqu'à la norme, puis une autre jusqu'à la norme,...). Le choix fait est d'abaisser successivement chacune des adaptations jusqu'à la norme, par ordre inverse d'importance.
  - En cas de mauvaise réponse (suite à une baisse d'intensité), soit on a trop baissé l'intensité, soit il faut une nouvelle adaptation :
    - On commence par remonter d'un cran l'intensité d'une adaptation qui vient juste d'être baissée.
    - Si on a une bonne réponse, alors on fixe provisoirement l'intensité de cette adaptation pour la suite (il s'agit de l'intensité "optimale" qui est la limite pour que l'élève traite le problème), et on poursuit avec une question plus difficile. Sinon on choisit une nouvelle adaptation.
  - Quand une adaptation a été baissée et fixée provisoirement à une intensité optimale (au-dessus du min) alors on peut plus tard abaisser à nouveau l'intensité (toutes les P bonnes réponses). Mais on effectue ces baisses d'intensité optimale une fois que toutes les adaptations sont soient inactives soient à l'intensité optimale.
- Tant qu'on ne sait pas si une adaptation est pertinente, on n'abaisse pas l'intensité d'une autre adaptation
- Une adaptation pertinente non baissée jusqu'au min reste active (c'est-à-dire susceptible d'être encore baissée)
- Il faudra définir pour chaque adaptation l'intensité maximale, l'intensité minimale (correspondant à la norme sociale), et le pas de calcul (lorsque l'on baisse l'intensité).

Il faudra une fonction de calcul de poids qui prenne en compte toutes ces contraintes :

- poids fixée pour une adaptation selon l'étape d'apprentissage
- calcul du poids des adaptations cumulées : somme des poids des adaptations la composant
- prise en compte des influences : une adaptation (nœud sortant) reliée à une autre déjà

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

proposée et jugée pertinente, reçoit un poids plus important
- prise en compte des votes : plus le nombre de vote pour une adaptation est grand, plus on ajoute de poids
- le cas échéant, informations en début de session sur les troubles de l'enfant, donc des connaissances d'adaptations potentiellement pertinentes, à travers un calcul dynamique de poids
- prise en compte des aspects dynamiques : cf 4.1

Il faudra aussi une fonction de calcul du poids des adaptations dans le cas d'une baisse d'intensité. Notamment pour une adaptation cumulée, où on abaisse successivement chacune des adaptations jusqu'à la norme, par ordre inverse d'importance.

Prise en compte du temps de réponse (non implémenté en version1) :
- si pas de réponse (temps de réponse > Tmax) alors ???
- si bonne réponse mais avec temps de réponse long (> T0) plusieurs fois de suite, alors on propose une adaptation mais pas à l'intensité maxi (1/3 ou 1/2). Si temps de réponse diminue alors adaptation ok (on baisse l'intensité ???), sinon changement d'adaptation (ou on monte l'intensité ???)
- si mauvaise réponse et avec temps de réponse long (> T1), alors on propose une adaptation cumulée

4.3- Traitement informatique de gestion des adaptations

A chaque étape d'apprentissage, lorsque l'on propose une adaptation, on va en général avoir plusieurs choix possibles. Les points évoqués dans les chapitres 4.1 et 4.2 permettent de faire un choix.
On tient compte de toutes les contraintes exprimées dans les chapitres 2, 4.1 et 4.2.
On implémente des fonctions de calcul de l'ensemble des adaptations possibles, et calcul des poids des adaptations (dans le cas général et dans le cas d'une baisse d'intensité) ;

Typiquement, chaque utilisateur sera associée à une session, l'algorithme (un script C#) sera appelé après chaque validation de réponse d'un utilisateur.

4.3.1- Notion d'agent

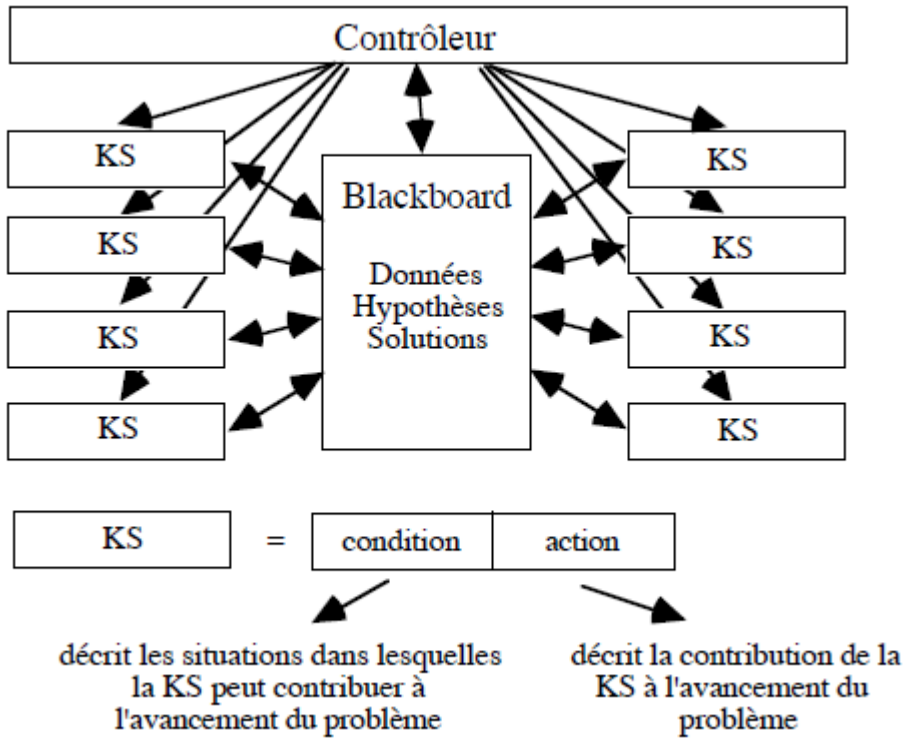
L'implémentation de la méthode de gestion des adaptations à partir de connaissances explicites peut se faire via un algorithme basé sur des agents.

En référence aux systèmes multi-agents utilisées en Intelligence Artificielle, ces agents ont accès à un "tableau noir" de données communes, partagées, dynamiques, et qui représentent l'état courant du système à un instant t. Le tableau noir est un espace de recherche partagé où s'inscrivent les résultats obtenus par les agents (ou sources de connaissances KS). L'accès au tableau noir est ici en lecture et écriture. Il s'agit donc d'un mode de communication indirecte entre agents.

Un agent est un module indépendant regroupant des connaissances homogènes, sous la forme d'une règle avec des conditions d'applications (des valeurs particulières de certaines variables du tableau noir, et des actions à réaliser en sortie (mises à jour de variables du tableau noir, poser une nouvelle

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

question au niveau supérieur, au niveau inférieur ou au même niveau, mettre en œuvre une adaptation, abaisser le niveau d'une adaptation active, remonter le niveau d'une adaptation active, annuler la précédente adaptation car non pertinente pour l'élève,…).
Les agents traquent les changements du tableau noir : quand une configuration est reconnue (partie condition satisfaite), la partie action est déclenchée et crée de nouveaux objets dans le tableau.
Mais contrairement à un système multi-agents, il n'y a pas ici de distribution des agents.
On peut mettre en œuvre un contrôleur qui gère les agents (choix d'un agent, priorité entre agents) à travers des règles générales, des métaconnaissances,... On peut aussi laisser le système évoluer sans contrôle : dès qu'un agent peut agir, il agit.



4.3.2- Base de données

On commence par construire la base orientée graphe (voir chapitres 1 et 3.1) sous Neo4J, à partir de

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

fichiers csv et de requêtes Cypher dans le shell Neo4J. On a 7 types de nœud et 7 types de relation :

- voir document "Neo4J.odt", chapitre "5.7.4- import des données depuis des fichiers csv avec un encodage utf-8 (sans BOM)".
- voir dossier de fichiers csv "csv import avec Cypher"

La base réelle sera créée à partir de fichiers csv, mais pour les tests unitaires, on peut créer une mini base, soit avec des fichiers csv, soit directement avec des requêtes.

4.3.3- Tests

Les tests unitaires servent à tester chaque bout de code séparément. Il s'agit de tester :

- tous les agents spécifiques (13 au total),
- toutes les requêtes dans la base (classe DAO)
- toutes les fonctions (calcul des adaptations possibles, calcul du poids d'une adaptation).

Ensuite, il y a les tests d'intégration, une fois que l'algorithme de gestion des adaptations est intégré dans l'application, afin de tester les enchaînements de réponses et d'appel aux divers agents.

4.3.4- Les Classes

Ce script fera appel à :

- Une classe "Adaptation" définie dans le modèle de données,
- Une classe par agent spécifique, qui réalisent un certain nombre d'actions s'ils sont activés. Un agent est activé si un certain nombre de conditions sont vérifiées (valeurs de variables globales, réponse bonne ou pas). Il y a 13 agents spécifiques :
  - Agent en cas de 1ère réponse fausse : on attend car il peut s'agir étourderie possible
  - Agent en cas de 2ième réponse fausse : on propose une adaptation, et on annule l'adaptation qui vient d'avoir lieu (s'il y en a eu une que ce n'est pas une remontée d'intensité car il s'agit d'une adaptation pertinente)
  - Agent en cas de réponse fausse après une baisse de niveau : on cherche une adaptation qui contienne la précédente, sinon une autre (de poids maxi)
  - Agent en cas de réponse fausse entraînant une baisse de niveau (qui fait suite à 3 adaptations proposées mais non pertinentes) : on baisse de niveau, on pose une nouvelle question, et on place l'adaptation dans la liste des adaptations non pertinentes
  - Agent en cas de réponse fausse entraînant une remontée d'intensité d'une adaptation (qui fait suite à une réponse fausse après une baisse d'intensité) : on remonte d'un cran l'intensité de la dernière adaptation qui vient d'être baissée
  - Agent en cas de réponse juste après une baisse de niveau : on monte d'un niveau, et on pose une nouvelle question
  - Agent en cas de réponse juste après une baisse d'intensité d'une adaptation : si elle atteint son min, alors elle devient inactive. On monte d'un niveau, et on pose une nouvelle question
  - Agent en cas de réponse juste après une remontée d'intensité : on est à l'intensité optimale pour cette adaptation, on regarde si toutes les adaptations actives sont à l'optimum ou pas. On monte

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

d'un niveau et on pose une nouvelle question

- Agent en cas de réponse juste après une adaptation (hors baisse de niveau, baisse d'intensité, et hausse d'intensité) : on enregistre l'adaptation dans la liste des adaptations pertinentes et la liste des adaptations actives
- Agent en cas de réponse juste et qu'il n'y a pas eu d'adaptation proposée (et pas de baisse de niveau) : on monte d'un niveau et on pose une nouvelle question
- Agent qui compte le nombre de réponses justes avant de proposer une baisse d'intensité (dans un 1er temps on baisse les intensités des adaptations actives d'un cran toutes les N bonnes réponses, puis dans un 2ième temps, lorsque toutes les adaptations actives ont été baissées au min donc sont devenues inactives, ou ont été baissées puis remontées à l'intensité optimale, on baisse les adaptations à l'intensité optimale)
- Agent en cas de réponse juste entraînant une baisse d'intensité
- Agent en cas de fin d'étape

- Ces classes d'agent spécifique héritent d'une même classe "Agent"
- Une classe BlackBoard contenant uniquement les variables d'état du système, dynamiques et accessibles à tous les agents, qui peuvent agir en fonction des valeurs de ces variables.
- Une classe "DAO" pour l'accès aux données de la base

Il y aura donc des variables globales, de session, accessibles depuis le script, et mises à jour dynamiquement dans le script. Ces variables globales pourraient être des variables static d'une classe static BlackBoard :

- `int nbExemples`

- `int nbMauvaisesReponses // utilisé pour tester si on a une étourderie ou pas`

- `int k // nombre d'adaptations proposées`
- `int kMax = 3`

- `int n  // nombre de bonnes réponses successives avant baisse d'intensité`
- `int nMax = 3`

- `int p // nombre de bonnes réponses successives avant baisse d'intensité optimale`
- `int pMax = 4`

- `bool toutEstOptimal // vrai si toutes les adaptations actives sont à leur intensité optimale ou min (inactives), avec au moins une à l'optimal`
- `bool baisseDeNiveau // cas où on baisse d'un niveau suite à des mauvaises réponses`
- `bool baisseIntensite  // vrai s'il vient d'y avoir une baisse d'intensité`
- `bool hausseIntensite  // vrai s'il vient d'y avoir une hausse d'intensité`

- `Adaptation derniereAdaptation // dernière adaptation proposée, et on ne sait pas encore si elle est pertinente`

- `List<Adaptation> ListeAdaptationsActives // liste des adaptations pertinentes et actives`

- `List<AdapatationInstanciee> ListeAdaptationsNonPertinentes // liste des adaptations non pertinentes deja proposées + niveau, étape et thème où elles ont été proposées`
`Adaptation instanciée = Adaptation + niveau + étape + thème`

- `List<Adaptation> ListeAdaptationsIntensiteOptimale // liste des adaptations baissées`

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

jusqu'à l'intensité optimale

- List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdapatation // séquence des niveaux (et l'étape et le thème) où il y a eu des questions mal répondues avec les adaptations pertinentes associées
- List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdaptationBaissee // séquence des niveaux (et l'étape et le thème) où il y a eu des baisses d'intensité
- List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdaptationInactive // séquence des niveaux (et l'étape et le thème) où des adaptations ont été baissées jusqu'au min sans problème (=> inactive)
- List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdaptationMontee // séquence des niveaux (et l'étape et le thème) où il y a eu des remontées d'intensité

La classe "Agent" contient :

- les variables communes, notamment les sorties d'un agent, donc toutes les actions à réaliser.

On a donc une liste de résultats en sortie d'un agent :

```
// resultats[0]
Liste outputListAdaptations // liste des adaptations à réaliser

// resultats[1]
booléen adaptationABaisser // true => baisser l'intensité d'un cran

// resultats[2]
booléen adaptationAMonter // true => monter l'intensité d'un cran

// resultats[3]
booléen continuerQuestionCourante // true =>  continuer sur la question qui vient d'être posée jusqu'à ce que l'enfant trouve la bonne réponse (processus automatisé)

// resultats[4]
Liste nouvelleQuestion

// nouvelleQuestion[0]
string outputTheme // thème de la nouvelle question à poser

// nouvelleQuestion[1]
string outputEtape // étape de la nouvelle question à poser

// nouvelleQuestion[2]
string outputNiveau // niveau de la nouvelle question à poser

// nouvelleQuestion[3]
booléen outputAutreQuestion // true => poser une autre question

// resultats[5]
booléen exitFinAppli
```

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- `// resultats[6]`
- `booléen exitPlusDAdaptation`

- `// resultats[7]`
- `booléen annulerAdaptation`

- les méthodes communes :
- `Liste calculEnsembleAdaptations(booléen reponse, string theme, string etape)`
- `int calculPoids(Adaptation a, string theme, string etape)`

Enfin, il y a la classe principale "Program" qui contient la fonction :

- `List gestionAdaptations(string niveau, string etape, string theme, bool reponse, BlackBoardToStore bb)`

Cette fonction commence par récupérer les données globales de BlackBoardToStore (une classe non statique, copie de la classe statique BlackBoard, et qui fournit les valeurs des variables d'état du système à l'instant courant, et qui est un paramètre de la fonction), puis instancie tous les agents, calcule quels agents sont applicables dans l'état courant du système, et calcule un score pour chaque agent applicable, fonction du nombre de conditions pour que l'agent soit applicable. Donc plus un agent a de conditions, plus il est spécifique, plus son score est élevé. A noter qu'un seul agent est sélectionné et appliqué à chaque appel de la fonction (celui avec le score maxi), mais l'agent Fin d'Etape, s'il est applicable, est appliqué en série avec l'agent déjà sélectionné. Puis les valeurs des variables d'état du système (classe BlackBoard) sont copiées dans la classe BlackBoardToStore et renvoyées au système (retour de la fonction gestionAdaptations())

Le script communique avec une base Neo4J, avec une connexion à la base en début de session, et une fermeture de la base en fin de session. La classe "DAO" comporte :

- une fonction par requête
  - requêtes pour interroger la base durant le déroulement de l'algorithme,
  - requêtes en fin de session pour stocker un nouvel exemple
- une fonction pour se connecter à la base : il s'agit du constructeur qui est programmé de façon à n'être instancié qu'une seule fois par session. La fonction récupère ensuite le nombre d'exemples dans la base
- une fonction pour fermer la base.

4.3.5- Les entrées et sorties

En entrée du script, on aura :

- la question, le niveau, l'activité et le thème courants (pour savoir où on est dans le processus d'apprentissage),
- la réponse associée (booléen)

Ce sont les paramètres de la fonction `gestionAdaptations()` du programme principal

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

En sortie :
Les sorties sont celles de l'agent appliqué (voir ci-dessus), c'est-à-dire une (ou plusieurs) action à réaliser : un concept dans le programme, donc à relier à une (ou plusieurs) action réelle. L'action peut être :

- une liste d'adaptations à faire, avec chacune une intensité associée.
- continuer sur la question qui vient d'être posée jusqu'à ce que l'enfant trouve la bonne réponse. Il s'agit d'un processus automatisé.
- poser une autre question (tirage aléatoire, mais ne pas poser la même question) à un certain niveau d'une étape donnée d'un thème donné
- annuler l'adaptation précédente
- sortir de l'application : soit parce que le processus est terminé et on est allé au bout, soit parce qu'à un niveau donné, on n'a plus aucune adaptation à proposer.

4.3.6- Intégration du programme C# dans l'application Unity3D

Une suite d'actions sont à réaliser :

Un programme qui fait le lien entre le programme C# qui interroge Neo4J (version récente de .net), c'est-à-dire le programme de gestion des adaptations, et le programme C# sous Unity3D (version antérieure de .net), c'est-à-dire le programme de toute l'application avec les interfaces graphiques, via le protocole TCP (programmation client-serveur).

Dans l'application, recopier le code de la classe BlackBoardToStore et des classes qu'elle utilise (Adaptation et AdaptationInstanciee), car l'application doit stocker les valeurs des variables globales d'état du système (retour de la fonction gestionAdaptations()) et fournir ces valeurs de variables d'état pour gérer les adaptations (paramètre d'entrée de la fonction gestionAdaptations())

Il faut installer Neo4J sur un serveur (et non sur mon poste local pour les tests unitaires), et l'application devra se connecter au serveur et à Neo4J.

Dans l'application, à chaque réponse de l'utilisateur, il faut lancer la fonction `gestionAdaptations(string niveau, string etape, string theme, bool reponse, BlackBoardToStore bb)`, à une exception : si on doit continuer sur la question qui vient d'être posée (paramètre `continuerQuestionCourante = true`) jusqu'à ce que l'enfant trouve la bonne réponse : il s'agit d'un processus automatisé.

Après le retour de la fonction `gestionAdaptations()`, il faut ajouter un code pour analyser les sorties de la fonction, qui est une liste "resultats" qui contient :

- la liste resultats des sorties de l'agent sélectionné : resultats[0] à resultats[7]. Ces données indiquent les actions à réaliser.
- la structure BlackBoardToStore : resultats[8]. Il s'agit de récupérer cette structure, puis de la

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

passer en paramètre de la fonction `gestionAdaptations()` lors de l'appel suivant.

4.3.7- Stockage d'exemple

En fin de session, on stockera en base des informations (questions mal répondues et adaptations associées, évolution de l'intensité des adaptations) qui permettront ensuite de réaliser des calculs statistiques sur de nombreux utilisateurs via des algorithmes de data-mining :

- clustering des utilisateurs,
- découverte d'associations (entre des adaptations, entre une adaptation et une question,...),
- choix d'une adaptation par apprentissage statistique ou via un système CBR (voir chapitre 5).

On enregistre pour chaque utilisateur la suite des questions, réponses, et adaptations. 2 cas se présentent :

- il y a un professeur qui propose toujours une adaptation pertinente, c'est la meilleure solution, car l'algorithme de Machine Learning se basera sur les recommandations du professeur, il aura (ou sera censé avoir) un comportement similaire au professeur : cas idéal mais utopique !
- S'il n'y a pas de professeur (donc c'est un algorithme qui détermine les adaptations), les adaptations proposées peuvent ne pas être pertinentes. Dans ce cas, il faudra un petit prétraitement pour supprimer les adaptations non pertinentes (une adaptation ayant entraîné 2 mauvaises réponses de suite sur un même niveau de questions, car on traite l'étourderie), et les questions-réponses associées. Je pense que si l'on ne supprime pas ces adaptations non pertinentes, le modèle appris soit de mauvaise qualité, car il aura appris à proposer des adaptations non pertinentes. Par ailleurs il s'agit aussi de stocker la baisse progressive d'intensité, et les éventuelles remontées. Ceci étant je me pose la question concernant les modifications d'intensité : je pense qu'une approche par apprentissage serait plus pertinente pour proposer une adaptation, la modification d'intensité est procédurale.

On peut alors stocker l'exemple dans une base Neo4J. 3 possibilités :

- on stocke les exemples dans la base existante contenant les connaissances génériques (adaptations, questions,…), mais on aura des données génériques et des données spécifiques dans la même base, ce qui n'est pas homogène.
  - Soit on réalise une copie de la base globale (connaissances génériques et exemples), puis on effectue des suppressions de nœuds et/ou relations des bases, afin de séparer la base globale en 2 bases.
  - Soit on utilise un label spécifique pour séparer les exemples des connaissances génériques, ou un nœud racine propre à chaque base, afin de les séparer (chaque base est un graphe distinct). **Par ex on pourrait créer pour chaque exemple un nœud de la classe (label) Exemple relié à la séquence de questions, adaptations, modifications d'intensité.**
- ou on crée une base spécifique pour les exemples, distincte de la base pour les connaissances génériques. Mais il n'est pas possible avec Neo4J d'ouvrir 2 bases en même temps.
  - soit on fait plusieurs installations de Neo4J avec une configuration de port différente : ainsi chaque installation pourra utiliser sa propre base
  - soit on utilise une virtualisation ou un conteneur tel que http//docker.io
  - soit en fin de session utilisateur, on ferme la base de connaissances génériques, on échange

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

de place les 2 bases car Neo4J utilise un dossier spécifique où se trouvent les fichiers liées à la base. Donc l'une est dans le dossier spécifique qu'utilise Neo4J et l'autre est dans un dossier quelconque. Après ce déplacement de fichiers, on ouvre la base d'exemples pour stocker le nouvel exemple.

- On a juste la base générique, et **on stocke les exemples dans des fichiers csv**. Lorsque l'on voudra utiliser les exemples, alors on créera une base Neo4J d'exemples à partir des fichiers csv.

Méthode retenue pour de stockage des exemples :
On stocke les exemples dans la même base que celle contenant les données génériques. Mais les données sont séparées.

On crée un nœud de type "nbExemples" dont la propriété "valeur" est le nombre d'exemples stockés dans la base. La requête Cypher est : CREATE (:nbExemples {valeur: 0})

Lorsqu'une session utilisateur va jusqu'à son terme, alors on stocke l'exemple dans la base :

- la valeur du noeud "nbExemples" est incrémentée de 1
- un nœud de type "Exemple" est créé
- on stocke 4 listes d'adaptations instanciées (adaptation + niveau + étape + thème) :
  - `List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdapatation`
  - `List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdaptationBaissee`
  - `List<AdapatationInstanciee>SequenceQuestionNiveauEtapeThemeAvecAdaptationInactive`
  - `List<AdapatationInstanciee> SequenceQuestionNiveauEtapeThemeAvecAdaptationMontee`

Pour chaque liste, on crée un lien du nœud "Exemple" vers la 1ère adaptation de la liste (cette adaptation est elle-même reliée au niveau, étape et thème où elle a été proposée), puis des liens entre la j ième adaptation et la j+1 ième (toujours en reliant chaque adaptation au niveau, étape et thème).

5- Autres approches possible pour la gestion des adaptations

5.1- Apprentissage vectoriel

5.1.1- Généralités sur l'apprentissage vectoriel

Une approche classique en Intelligence Artificielle est l'apprentissage supervisé (Supervised Machine Learning) : il s'agit d'algorithmes capables d'apprendre un modèle mathématique à partir d'exemples d'une classe, afin de pouvoir classer de nouveaux exemples inconnus du système

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

comme étant des instances de la classe, ou non.

Typiquement un exemple est représenté par un vecteur dont les dimensions sont les valeurs de variables explicatives (qu'il convient de définir préalablement). La plupart des algorithmes utilisent des valeurs de variable explicative de nature numérique / quantitative car ils sont basés sur des calculs mathématiques (par ex des calculs de distance entre vecteurs ou des calculs de probabilité ou des statistiques). Certains sont capables de traiter des valeurs qualitatives.

On fournit donc en entrée de l'algorithme des couples (exemple - classe associée), à partir desquels l'algorithme élabore un modèle mathématique. Étant donné que ces algorithmes n'utilisent aucune connaissance a priori mais se basent uniquement sur des calculs mathématiques (souvent statistiques), il faut fournir un grand nombre d'exemples afin que l'algorithme puisse espérer induire un modèle correct.

Il existe de nombreux algorithmes de Machine Learning (Support Vector Machines - SVM, k plus proches voisins, réseaux de neurones, arbres de décision probabilistes, régression logistique, modèle bayésien naïf, espace des versions, algorithmes génétiques, Case Based Reasoning – CBR, …).

Remarque : il existe un autre mode d'apprentissage, l'apprentissage non supervisé, où on fournit en entrée uniquement des exemples (pas la classe associée), et il s'agit de séparer les exemples en classes homogènes et bien séparées, inconnues au départ. On parle aussi de clustering.

5.1.2- Application de l'apprentissage vectoriel à notre problème

On se place en mode apprentissage supervisé, et il s'agit de modéliser les exemples, et les classes à apprendre par l'algorithme de Machine Learning :

Les classes à apprendre sont les adaptations. Chaque classe est une adaptation. Le nombre de classes est donc le nombre d'adaptations possibles, et il y a donc environ 60 classes à apprendre.

Pour la représentation des exemples, partons de l'analyse suivante :

- Première mauvaise réponse à une question : un espace vectoriel de taille 1 est possible, avec la question qui a posé problème. La classe est l'adaptation associée. On voit donc qu'il faut des exemples pour chaque question. Ce système serait utilisé pour traiter la première mauvaise réponse à une question.
- Seconde mauvaise réponse à une question : un espace vectoriel de taille 3 est possible, avec la première question ayant posé problème, l'adaptation associée, et la seconde question ayant posé problème. La classe est l'adaptation associée à cette seconde question. On voit donc qu'il faut de très nombreux exemples correspondant à tous les cas possibles de couples de mauvaises réponses (on peut se retreindre aux couples qui peuvent réellement se produire, mais il y a en tout de même beaucoup). Ce système serait utilisé pour traiter la seconde mauvaise réponse à une question.
- N ième mauvaise réponse à une question : Dans le même esprit, on augmente la taille de l'espace vectoriel (taille 2N-1) afin de prendre en compte (N-1) mauvaises réponses associées à des adaptations et la N ième mauvaise réponse dont on cherche l'adaptation, ce qui augmente encore le nombre d'exemples à fournir en entrée de l'algorithme. Ce système serait utilisé pour traiter la N

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

ième mauvaise réponse à une question.

Si on modélise l'espace vectoriel de manière qualitative, alors on utilise les noms des questions et des adaptations.
Si on modélise l'espace vectoriel de manière quantitative, alors il faut remplacer les valeurs qualitatives (noms des questions et des adaptations) par des nombres, et cela peut poser problème car il n'est pas évident que les calculs de distance entre deux vecteurs soient pertinents.
Il semble donc préférable que les dimensions de l'espace vectoriel soient associées à des variables qualitatives (nom d'une question, nom d'une adaptation). Il existe des algorithmes adaptés à ce type de variable, par ex l'algorithme des arbres de décision probabiliste.

L'apprentissage supervisé pose tout de même certains problèmes, car d'une part cela nécessite beaucoup d'exemples (peut être trop, ce qui rendrait l'approche irréaliste), d'autre part il n'y a pas de garanti de résultat satisfaisant, surtout concernant les adaptations liées aux premières mauvaises réponses (notamment sur la première mauvaise réponse, l'algorithme n'a pas les moyens de choisir la bonne adaptation parmi un ensemble, il fait un choix basé sur des statistiques).

Mais peut-être que les adaptations liées aux mauvaises réponses ultérieures pourraient être plus pertinentes, car on tient compte d'un certain historique relatif à des premières mauvaises réponses et des adaptations associées.

Intuitivement, si 2 élèves ont un historique commun de couples (mauvaise réponse - adaptation), et s'ils ont en commun également une mauvaise réponse ultérieure, alors on peut se baser sur l'adaptation (associée à cette mauvaise réponse) associée à un des élèves, et la proposer à l'autre élève.

Reste à déterminer la valeur minimale de N à partir de laquelle la valeur de la prédiction d'adaptation est pertinente : N = 1 ou 2 ou 3 ou plus ?
Reste également à déterminer la valeur maximale de N.

Ensuite, une fois choisi l'algorithme d'apprentissage, il faut exécuter l'apprentissage (Nmax – Nmin + 1) x (Nombre d'adaptations) fois. Pour cela, il faut une base d'exemples pour chaque apprentissage.

Pour construire les bases d'exemples (des profils utilisateur, ou élèves), il faut récupérer, pour chaque élève qui utilise l'outil actuel d'apprentissage, l'ensemble des questions qui ont entraîné des mauvaises réponses, et les adaptations qui ont été associées. Il faut un grand nombre d'élèves (selon moi plusieurs milliers, voire plusieurs dizaines de milliers).
Le problème est que les conditions de déroulement d'une séance d'apprentissage ne permettent pas de garantir qu'une adaptation est adéquate. On peut le supposer si on voit que le niveau est réussi juste après l'adaptation, mais là aussi, il n'y a pas de certitude. Il faut espérer que dans la plupart des cas, l'adaptation est adéquate. Des adaptations inadéquates pourraient polluer l'apprentissage si elles sont en nombre important. S'il s'agit d'adaptations « presque adéquates » qui sont en nombre important, alors seuls des tests permettront de savoir si le modèle appris est correct ou non.

Un point important est de savoir :
- si on veut que 2 élèves aient exactement le même historique afin de proposer l'adaptation

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

d'un des élèves à l'autre. Dans ce cas, il n'y a pas besoin d'algorithme d'apprentissage, juste stocker des exemples et en trouver un qui corresponde exactement au cas à traiter. Le problème est ici de trouver exactement le même profil, ce qui nécessite énormément d'exemples.

- si on autorise des différences au niveau de l'historique. Mais dans ce cas, il faut examiner s'il y a des contraintes sur les différences acceptables ou non entre 2 profils (plus précisément entre les dimensions des vecteurs, donc entre 2 questions, ou entre 2 adaptations), donc s'il faut définir une mesure de similarité *ad hoc* entre 2 profils
  - s'il n'y a pas de contraintes, alors on peut utiliser tout type d'algorithme d'apprentissage supervisé (basé sur une distance, ou des probabilités).
  - sinon, il faut définir une mesure de similarité (inverse d'une distance) *ad hoc* et utiliser un algorithme basé sur un calcul de distance.

On peut également se placer **en mode apprentissage non supervisé**, et le but est de séparer des exemples (profils d'élève ayant réalisé une séance d'apprentissage) en groupes homogènes.

5.2- Apprentissage de séquences

Dans les chapitres précédents (5.1 et 5.2), on représente un exemple sous la forme d'un vecteur, et donc on ne tient pas compte d'un éventuel ordre entre les dimensions du vecteur.

Il existe des algorithmes de Machine Learning séquentiel conçus pour traiter des exemples sous la forme de séquence :

Soit une séquence d'observations X = (x1, x2, …, xN)
Et une séquence de classes Y = (y1, y2, …, yN)

Le but est de trouver la meilleure séquence de classe Y connaissant la séquence d'observations X, c'est-à-dire trouver une séquence Y de façon à maximiser la probabilité que la séquence X soit étiquetée par la séquence Y.

Un algorithme très utilisé est la méthode des Conditional Random Fields (CRF).

Pour notre problème, on crée une base d'exemples constituée de séquences de mauvaises réponses à des questions (les observations), et les séquences de classes associées sont les adaptations à chaque mauvaise réponse.

Comme précédemment, pour construire les bases d'exemples, il faut récupérer, pour chaque élève qui utilise l'outil actuel d'apprentissage, l'ensemble des questions qui ont entraîné des mauvaises réponses, et les adaptations qui ont été associées.

On exécute l'apprentissage pour $N_{min} \leq N \leq N_{max}$

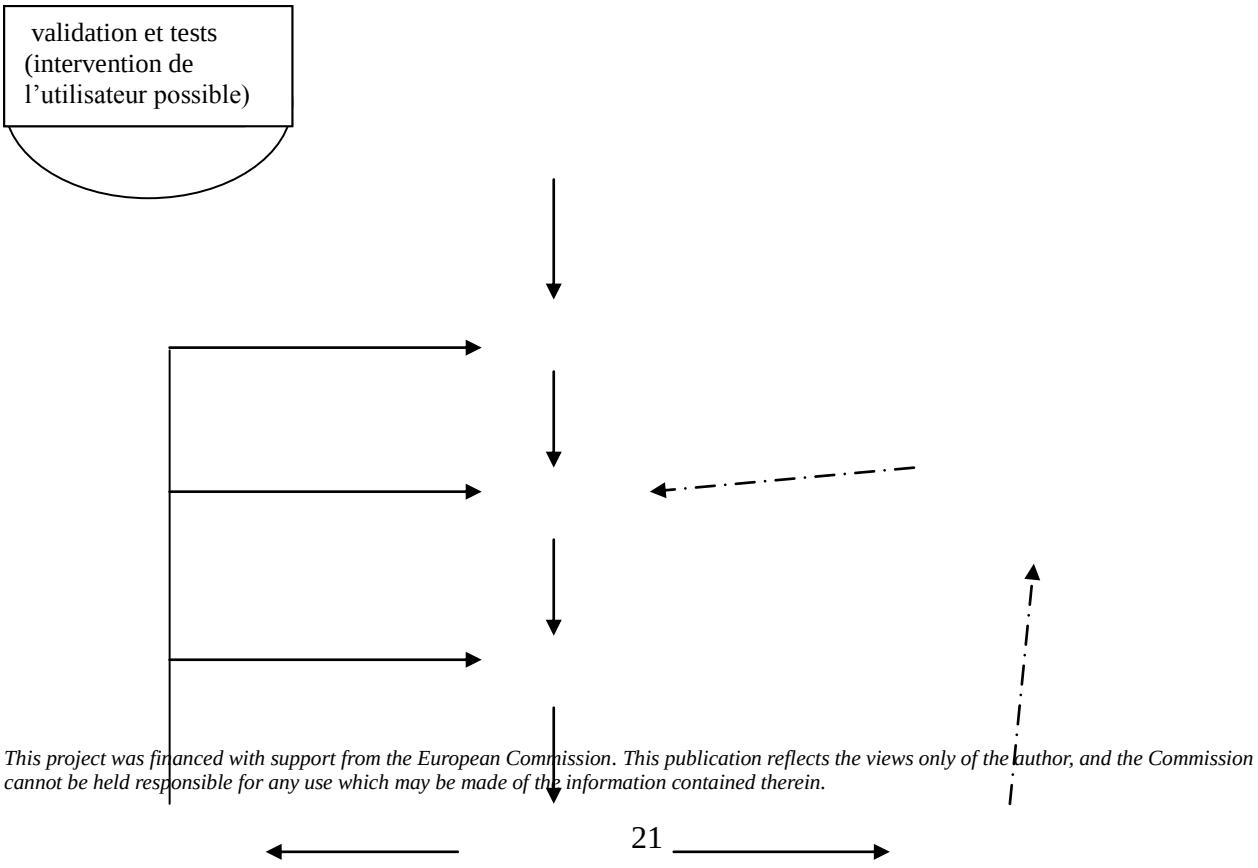
This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

5.3- Case-Based Reasoning - CBR

5.3.1- Introduction

Un humain utilise très souvent ses expériences passées, qu'il s'agisse de réussites ou bien d'échecs, afin de résoudre un problème courant, puis apprend à partir de ce problème résolu. Cela consiste généralement à rechercher en mémoire une expérience similaire survenue dans le passé, à adapter cette expérience au contexte présent, à la valider, la ré-utiliser, puis enfin à la stocker en vue d'une ré-utilisation future.
Le raisonnement par analogie consiste, pour résoudre un problème, à aller rechercher en mémoire, sur la base d'indices (les mots d'un texte, les formes visuelles, …), les expériences passées qui permettront de comprendre puis résoudre ce problème. L'intérêt de cette approche est d'être proche d'une certaine forme du raisonnement humain. Nous allons examiner une approche possible de ce type de raisonnement : le raisonnement à base de cas (CBR *: Case-Based Reasoning*).

Un système de raisonnement à base de cas intègre deux aspects essentiels : il est susceptible d'apprendre à partir de ses expériences (un problème résolu correspond à un "cas" et est conservé en mémoire), et il utilise ses expériences passées pour résoudre des problèmes (le système recherche en mémoire un cas passé jugé similaire et le réutilise après l'avoir adapté à la nouvelle situation en recherchant les différences entre les deux situations). Le terme résolution de problèmes est vu ici au sens large : il peut s'agir de justifier un raisonnement ou un argument, de critiquer ou expliquer une solution fournie par l'utilisateur, d'interpréter une situation. Un système CBR opère en plusieurs phases bien distinctes. Il y a un ordre à respecter entre ces phases, ainsi que des cycles possibles à partir de la phase de validation vers la phase d'indexation, ou de recherche, ou d'adaptation. En outre, une intervention de l'utilisateur peut être nécessaire si le système n'a pas les connaissances suffisantes pour résoudre un problème nouveau



This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Un cas est représenté par les données du problème (état initial), la solution (état final), les contraintes et le chemin (les pas intermédiaires) permettant de passer de l'état initial à l'état final.
Les cas peuvent être regroupés en concepts, ces concepts étant organisés sous forme d'une liste ou d'une hiérarchie. Chaque concept est défini en extension (par l'ensemble de ses membres) et éventuellement par intention (par l'ensemble de ses propriétés représentant des conditions nécessaires et suffisantes).
Des modifications peuvent avoir lieu lors de l'apprentissage, soit par ré-organisation de la hiérarchie des index lors de l'intégration du nouveau cas, soit par abstraction de deux cas similaires (fusionnement d'un cas ancien et du cas nouveau).

1- Indexation

Le but principal d'un index est de prédire l'utilité d'un cas et d'aider lors de la phase de recherche de cas similaires. Il doit représenter les buts que le cas peut atteindre ou aider à atteindre, ainsi que les circonstances, le contexte dans lequel le cas est utile pour chacun de ces buts. Un cas peut posséder plusieurs index, s'il existe par exemple plusieurs contextes à partir desquels le cas peut être utilisé. Il s'agit ensuite de généraliser cette description pour qu'elle soit applicable dans diverses situations futures.
L'indexation consiste à déterminer les descripteurs importants du problème à résoudre et éliminer les descripteurs non pertinents à la vue du contexte, et éventuellement inférer des descripteurs non présents en entrée grâce aux connaissances générales du domaine, afin notamment que la situation nouvelle soit décrite dans le même vocabulaire que les cas en mémoire

2- Recherche de cas similaires

Elle se fait à partir des index, et impose de définir la notion de similarité.
deux types de procédures sont intégrés dans cette phase :

- la procédure de mise en correspondance et de classement : il s'agit ici de comparer deux cas et de juger de leur degré d'adéquation (ou de similarité). Cet algorithme prend donc en entrée la description et le but à atteindre du problème nouveau.
- la procédure de recherche de cas similaires dans la librairie de cas (ou d'index). Cet algorithme dépend donc de l'organisation de la base de cas/index (liste, arbre, ou graphe de concepts).

La similarité est souvent basée sur des critères superficiels, et est définie soit sous la forme d'une distance (entre deux vecteurs), soit par le biais d'une classification dans une hiérarchie d'index (notion de plus petit généralisant : [Salotti *et al.* 99]), soit encore sous la forme d'une procédure de mise en correspondance.

Il existe de nombreuses façons de calculer la similarité :
Généralement, on définit la similarité par :

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- *SIM (x, y)* = propriétés communes à *x* et *y*. Il peut être utile, voire nécessaire, de définir une mesure de similarité pour chaque type de propriété.

Et la dis-similarité par :
- *DISS (x, y)* = propriétés de *x* qui ne subsument pas une propriété de *y* [Salotti *et al.* 99] (opération non symétrique)

Plus spécifiquement, des exemples de mesure de similarité sont :
- *SIM (x, y)* = nombre de propriétés (ou une pondération) communes à *x* et *y*.
- *SIM (x, y) = f (x ∩ y, x – y, y – x)* qui combine similarité et dis-similarité,

ou bien, dans le cas où les deux individus *x* et *y* sont décrits par des ensembles d'attributs *A* et *B* respectivement :
- *SIM (x, y) = f (A ∩ B) / (f (A ∪ B) + α.f (A – B) + β.f (B – A))* avec *α* et *β* positifs.
- *SIM (x, y)* = longueur (nombre d'arêtes) entre les deux concepts *x* et *y* au sein d'une hiérarchie.

A. Mille et B. Fuchs [Mille *et al.* 99], définissent deux mesures de similarité et de dis-similarité dans le cadre de la supervision industrielle, où un cas est représenté. Leur définition de la dis-similarité entre deux séquences d'événements temporels est assez générale et représente un intérêt pour notre étude.

3- Adaptation

Il s'agit de réutiliser le ou les cas sélectionnés et proposer une solution. Pour cela, il faut tenir compte des différences entre le cas remémoré et le problème à résoudre. Il faut agir sur une ancienne solution, c'est-à-dire insérer un élément nouveau, effacer un élément, substituer un élément par un autre. En outre, il s'agit éventuellement de savoir quelle(s) partie(s) du cas retrouvé sera transférée au cas à résoudre. Cette phase a lieu lors de la formulation d'une solution, et éventuellement après un *feedback* de l'utilisateur (phase de révision) ayant pointé des problèmes.

On distingue, de manière non exhaustive, quatre approches : substitution, opérateurs de transformation (en fonction de différences), dérivation, heuristiques.

4- Validation

L'évaluation de la solution proposée détermine si la solution est correcte. Cela peut être fait par l'utilisateur, ou par un simulateur, ou en exécutant la solution dans le cadre réel. Si cette évaluation prend un certain temps, alors le cas peut être enregistré dans la base, mais il doit être marqué comme non évalué.
Si l'évaluation est réussie, on passe à la phase d'apprentissage.
Sinon, nous sommes alors dans le cadre d'un apprentissage à partir d'échec : il faut *réparer la solution*, c'est-à-dire :
- détecter les erreurs. Cette phase peut nécessiter l'intervention de l'utilisateur s'il faut modifier certaines connaissances du système ou lui fournir de nouvelles connaissances.
- corriger la solution et inclure des connaissances de façon à ne plus commettre ces erreurs. Cette phase peut également nécessiter l'intervention de l'utilisateur. Mais on peut parfois automatiser cette phase (retour sur la phase d'adaptation, ou la phase d'indexation, ou sur la

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

définition de la similarité,…) :

◦ si le bon cas a été retrouvé, alors c'est l'adaptation qui est en cause. On boucle sur la phase d'adaptation (les informations éventuellement fournies par l'utilisateur vont être utilisées), puis on modifie, par apprentissage, les stratégies d'adaptation [Leake 96 (a)] (annexe 5).

◦ si le système a utilisé un cas inopportun alors qu'il existe dans la base un cas qui permettrait de résoudre le problème, alors on boucle sur la phase de recherche pour tester un autre cas. Mais il peut être utile, si le système possède suffisamment de connaissances, de ré-indexer le cas nouveau (le problème à résoudre) ainsi que certains cas passés (ceux qui ont été injustement jugés similaires et ceux qui ont été injustement jugés non similaires), puis de modifier la hiérarchie des cas/index en conséquence. Enfin, il peut être utile de modifier les connaissances liées aux notions de similarité et d'indexation.

◦ si le système a utilisé un cas inopportun et qu'il n'existe pas dans la base un cas qui permettrait de résoudre le problème, alors on peut soit tenter d'adapter le cas inopportun, ce qui peut nécessiter un certain travail si le problème à résoudre et la cas retrouvé sont dis-similaires, soit enregistrer le problème et sa solution (grâce à l'intervention de l'utilisateur) car il s'agit là d'un problème vraiment nouveau que le système ne pouvait résoudre avec sa base de cas.

5- Apprentissage

Le but est de rendre le système plus efficace (si un cas a été adapté d'une nouvelle façon, a été résolu par une nouvelle méthode ou grâce à la combinaison de solutions de plusieurs cas, alors lors d'un rappel futur le travail réalisé n'aura pas à être répété) et plus compétent (l'expérience permet au système d'anticiper et d'éviter les erreurs commises dans le passé).

Le premier problème est de savoir si l'on doit effectivement enregistrer le cas nouveau : le système a-t-il appris quelque chose d'utile ? La notion d'utilité doit donc être définie, notamment par rapport à l'atteinte de buts, mais également en terme de contexte (si un problème connu a été résolu dans un contexte très différent, alors il est utile de généraliser le contexte). Selon [Kolodner et Leake 96], si ce qui est différent dans une nouvelle situation nous enseigne quelque chose qui n'aurait pas pu être *facilement* inféré à partir des cas existants, alors ce nouveau cas est utile.

Un problème potentiel lié à l'apprentissage par accumulation est qu'en enregistrant un nouveau cas après chaque résolution d'un problème, la base de cas peut rapidement devenir ingérable du fait de sa taille incontrôlée. Il faut donc enregistrer un cas de manière sélective, c'est-à-dire juger de l'intérêt du cas nouveau par rapport au contenu de la base de cas, comme cela vient d'être souligné précédemment. On peut également, à l'occasion, supprimer des cas dont la fréquence d'utilisation est devenue très faible, ce qui équivaut à un oubli. La généralisation de cas (fusionner le cas appris avec un autre cas similaire en mémoire) est aussi un moyen d'éviter d'enregistrer tous les cas. On peut citer à ce sujet les opérateurs de généralisation à propos de la méthode "espace des versions" [Cornuejols et Miclet 02] : pour un attribut numérique, on généralise à l'aide d'un intervalle englobant les différentes valeurs, ou à l'aide d'une valeur moyenne avec un écart-type, et pour un attribut qualitatif hiérarchique, on peut généraliser en utilisant l'attribut "père".

L'apprentissage au sein d'un système CBR peut être mis en œuvre de multiples manières :

• Apprentissage par accumulation (sélective) d'expériences (stockage de cas).

• Apprentissage des index (l'index d'un cas stocké peut être modifié dans le futur en fonction des nouvelles expériences), ou des connaissances d'indexation (pour rechercher les indices pertinents du problème à résoudre - phase de modélisation).

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- Apprentissage de l'organisation de la mémoire [Leake 95], afin notamment d'améliorer la phase de recherche. Il s'agit également, en utilisant les informations issues des différentes phases, de réorganiser la mémoire de cas et d'index :
  - trouver la place du cas résolu dans la hiérarchie, et éventuellement réorganiser cette hiérarchie (voir par exemple le chapitre sur la classification dans [Haton 91]),
  - généraliser le nouveau cas avec des cas passés,
  - créer de nouveaux index regroupant des cas similaires.
- Apprentissage de stratégies d'adaptation, sous la forme de règles de substitution ou de transformation, ou encore grâce à la génération de procédures de recherche en mémoire [Leake 93].

• Apprentissage de connaissances négatives afin de mieux définir la similarité, l'indexation ainsi que l'adaptation.

- Apprentissage de la notion de similarité, par exemple en modifiant les poids associés à une distance, ou encore en estimant un coût lié à l'obtention ou à l'adaptation de la solution [Leake 96] : en effet, lorsque plusieurs cas similaires ont été retrouvés pour résoudre un problème nouveau, il est judicieux de choisir le cas dont la solution est la plus facile à mettre en œuvre, c'est-à-dire nécessitant le moins d'adaptation.

• Améliorer le modèle des connaissances générales du domaine, utilisé pour définir la similarité, l'indexation, l'adaptation.

le CBR utilise donc un algorithme générique et des cas spécifiques. Un changement de stratégie entraîne donc juste la modification de certains cas. La maintenance est plus rapide qu'avec un algorithme ad hoc.

5.3.2- Application

1- Cas génériques ou cas spécifiques :

Dès lors il paraît intéressant de développer le raisonnement à partir de cas : l'idée est d'utiliser une librairie de cas stockés pour résoudre un problème nouveau par analogie, en se basant sur les notions de similarité, d'indexation et de mise en correspondance. Un cas représente les caractéristiques du problème (état initial, contraintes), la solution (état final), et éventuellement les pas intermédiaires menant de l'état initial à l'état final. A chaque cas est associé un index comportant les indices descriptifs importants et discriminants.

Dans le cadre de notre étude, on aurait une base de cas représentant des situations modélisées par un certain nombre de questions avec les réponses et les adaptations, par ex sous la forme d'un graphe sous Neo4J, ou bien sous la forme d'instances de classes (si besoin de connaissances procédurales), avec des caractéristiques / index et une action.
Chaque cas est associé à une action qui permet d'atteindre un but (par ex proposer une adaptation du contenu pédagogique afin de réussir un niveau d'apprentissage).

Il y a 2 façons de mettre en œuvre le CBR :

- **avec des connaissances expertes, donc quelques cas assez génériques, qui sont donc des problèmes génériques résolus**
- **avec de nombreux cas qui sont des exemples de problèmes spécifiques résolus (comme**

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

pour l'approche Machine Learning)

L'algorithme basé sur des connaissances explicites et génériques (chapitre 4) est basé sur des agents qui ont la forme de cas, avec des conditions initiales, et des actions à réaliser. Mais il n'y a pas toutes les phases d'un système CBR :

- indexation : non
- recherche de cas similaire : très simplifié ici, basé sur des conditions à vérifier
- adaptation : non
- validation : non
- apprentissage : non

L'algorithme présenté au chapitre 4 est donc assez éloigné d'un système à base de cas. Donc l'approche CBR générique n'est pas adaptée ici.

Par contre, si on stocke les exemples issus des sessions d'utilisateurs, par exemple en utilisant l'approche générique détaillée au chapitre 4, alors les exemples pourront alimenter ensuite un système de Machine Learning ou un système CBR spécifiques (basés sur de nombreux cas spécifiques, c'est-à-dire les exemples).
Etant donné que les exemples ont une structure assez complexe, et qu'il faudra probablement développer une procédure spécifique de recherche de cas similaire (intégrer les connaissances permettant de juger que l'historique d'un élève est similaire à un cas en mémoire), il apparaît qu'un système CBR sera peut être plus approprié qu'un système de Machine Learning.

2-Modélisation d'une base de cas :
Un cas est un historique (une séquence de questions mal répondues, les adaptations pertinentes associées, et éventuellement les baisses d'intensité d'adaptations), la question suivant cet historique et mal répondue, et l'action associé (une adaptation).
Si aucun cas n'est applicable, alors on ne fait rien.

Indexation des cas :
Les cas en mémoire sont indexés, et le problème à résoudre (la situation d'apprentissage vécue par l'utilisateur) est lui aussi indexé.
Les index sont les éléments importants et discriminants décrivant une situation :

- questions et niveaux où il y a eu un échec, caractéristiques des adaptations pertinentes proposées
- classe Static pour toutes les variables globales qui décrivent les caractéristiques du problème courant)

Recherche de cas similaires :
Le système cherche à reconnaître l'application d'au moins un cas en mémoire. La procédure de recherche de cas sera basée sur une (ou plusieurs) procédure de mise en correspondance :

- au niveau des index, avec des pénalités lorsque des différences apparaissent (éventuellement pénalité infinie => rejet du cas)
- entre les 2 graphes, afin de mettre en correspondance le graphe Neo4J modélisant la situation d'apprentissage vécue par un utilisateur (toute la suite des questions, réponses et

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

adaptations jusqu'à l'instant courant), et les graphes neo4J modélisant des cas en mémoire. Chaque différence donne lieu à une pénalisation numérique (éventuellement pénalité infinie => rejet du cas)

- choix du cas avec pénalité globale mini.

Procédure d'adaptation :
Le cas choisi est ensuite adapté le cas échéant.
L'adaptation consisterait à prendre en compte les différences entre le cas choisi et la situation réelle afin de proposer une action adaptée.
A étudier de plus près, mais a priori, pas besoin d'adapter les cas pour ce problème.

Procédure de validation/réparation :
Le cas choisi est ensuite mis en œuvre (on réalise l'action associée au cas choisi)
Il n'est pas évident de mettre en place une validation par l'utilisateur de l'action proposée par le système CBR.
Mais on peut décider si l'action est pertinente en fonction de la réponse à la question juste après avoir réalisé l'action.

Procédure d'apprentissage de cas :

- juger de l'utilité d'enregistrer le problème résolu (fonction de la difficulté rencontrée par le système pour résoudre ce problème)
- repérer les actions pertinentes, et analyse des caractéristiques permettant de décider du choix de l'action en fonction du contexte (création d'un cas plus précis),
- modifier le calcul des poids
- modifier le calcul des pénalités.

5.4- Système à base de règles

Introduction :
Un système expert se compose de 3 parties :

- une base de faits ;
- une base de règles ;
- un moteur d'inférence.

Le moteur d'inférence est capable d'utiliser faits et règles pour produire de nouveaux faits, jusqu'à parvenir à la réponse à la question posée.

La plupart des systèmes experts existants reposent sur des mécanismes de logique formelle (logique aristotélicienne) et utilisent le raisonnement déductif. Pour l'essentiel, ils utilisent la règle d'inférence suivante (syllogisme) :

- si P est vrai (*fait* ou *prémisse*) et si on sait que P implique Q *(règle)*, alors Q est vrai (*nouveau fait* ou *conclusion*).

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Les plus simples des systèmes experts s'appuient sur la logique des propositions (dite aussi « *logique d'ordre 0* »). Dans cette logique, on n'utilise que des propositions, qui sont vraies, ou fausses. D'autres systèmes s'appuient sur la logique des prédicats du premier ordre (dite aussi « *logique d'ordre 1* »), que des algorithmes permettent de manipuler aisément.

Enfin, pour faciliter la description de problèmes réels sous forme de règles logiques, on a recours à des opérateurs ou des valeurs supplémentaires (notions de nécessité/possibilité, coefficients de plausibilité, etc.).

Il existe de nombreux types de moteurs, capables de traiter différentes formes de règles logiques pour déduire de nouveaux faits à partir de la base de connaissance.

On distingue souvent trois catégories, basées sur la manière dont les problèmes sont résolus :

- les moteurs - dits à « *chaînage avant* » - qui partent des faits et règles de la base de connaissance, et tentent de s'approcher des faits recherchés par le problème.
- les moteurs - dits à « *chaînage arrière* » - qui partent des faits recherchés par le problème, et tentent par l'intermédiaire des règles, de « remonter » à des faits connus,
- les moteurs - dits à « *chaînage mixte* » - qui utilisent une combinaison de ces deux approches *chaînage avant* et *chaînage arrière*.

Des exemples de moteur d'inférence sont :

- CLIPS : Chaînage avant, Contrôle irrévocable, Moteur d'ordre 1, Logique non monotone, Monde fermé
- PROLOG : moteur d'ordre 1 en chaînage arrière
- Drools (ou JBoss Rules) est un logiciel informatique gérant les règles métier (SGRM) utilisant un raisonnement déductif se basant sur des prémisses définies par l'utilisateur, et basé sur l'algorithme de Rete. Drools suit la norme JSR-94. C'est un logiciel libre distribué par Red Hat selon les termes de la licence Apache.

Certains moteurs d'inférence peuvent être partiellement pilotés ou contrôlés par des méta-règles qui modifient leur fonctionnement et leurs modalités de raisonnement.

Si les algorithmes de manipulation de faits et de règles sont nombreux et connus, la détermination de l'ensemble des faits et règles qui vont composer la base de connaissances est un problème délicat. Comment décrire le comportement d'un expert face à un problème particulier, et sa manière de le résoudre, là est la question. Car ce que l'on souhaite obtenir n'est ni plus ni moins que l'expérience, la connaissance pratique de l'expert, et non la théorie que l'on peut trouver dans les livres ni exclusivement les règles logiques d'inférence. Equivalents des méthodes d'analyse de l'informatique traditionnelle, des méthodes d'acquisition des connaissances sont développées.

En pratique, dès que l'on dépasse la centaine de règles, il devient difficile de suivre comment le système expert « raisonne » (manipule faits et règles en temps réel), et donc d'en assurer la mise au point finale, puis la maintenance.

Aujourd'hui les systèmes experts sont courants, notamment dans la finance et le secteur médical

Conclusion :

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Comme pour le CBR, on a un algorithme générique (ici le moteur d'inférence) et des cas spécifiques (ici les règles).
Un changement de stratégie entraîne donc la modification de certaines règles.

L'algorithme basé sur des connaissances explicites (chapitre 4) est basé sur des agents qui ont la forme de règles.
Nous avons développer un moteur d'inférences, où à chaque pas on cherche une nouvelle règle à appliquer en calculant un score pour chaque agent et en appliquant l'agent de score maxi.
A noter que les conclusions des règles sont complexes, procédurales (divers types d'action, mises à jour de variables globales,…) et donc difficilement modélisables avec un moteur d'inférences existant. Il faut donc ici développer un algorithme spécifique capable de traiter ces règles, c'est ce qui a été fait.
L'algorithme présenté au chapitre 4 peut donc être vu comme un système multi-agents de type "tableau noir" mais les agents ne sont pas distribués, ou bien il peut être vu comme un système à base de règles avec des conclusions complexes, et un choix de règle basé sur un calcul simple de score de chaque règle.

6- Méthode hybride

6.1- Fusion d'algorithmes de classification

Pour un problème donné, on peut tester plusieurs algorithmes de classification. Si plusieurs algorithmes donnent d'assez bons résultats, ou tout du moins semblent complémentaires en terme de résultat (quand l'un se trompe, un autre ne se trompe pas), alors il est possible de fusionner ces algorithmes afin d'améliorer les résultats :

- vote sur les décisions prises par chaque algorithme
- réalisation d'un nouvel apprentissage (avec un algorithme à choisir) qui prend en entrée les décisions intermédiaires des algorithmes à fusionner, et fournit en sortie la bonne décision. Il s'agit donc d'une méta classification, qui utilise un apprentissage vectoriel.
- création de règles spécifiques qui indiquent dans quels cas utiliser tel ou tel algorithme (plus précisément la décision associée à tel ou tel algorithme)
- création d'une fonction de décision
- méthode de boosting probabiliste (par ex la méthode Adaboost) : il est possible d'améliorer l'apprentissage d'un algorithme donné en relançant plusieurs fois l'apprentissage de cet algorithme et en se concentrant à chaque étape sur les exemples mal classés. Au lieu d'utiliser un seul algorithme, on peut utiliser plusieurs algorithmes que l'on combine selon la même philosophie.

6.2- Hybridation entre des méthodes de nature différente

Il semble possible de combiner la méthode basée sur des connaissances explicites (chapitre 4) et l'une des méthodes basées sur des exemples (chapitre 5.1 sur l'apprentissage vectoriel, chapitre 5.2 sur l'apprentissage de séquences, et chapitre 5.3 sur le CBR).

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Plus précisément, une méthode basée sur des exemples pourrait être utilisée pour le choix d'une adaptation au sein de la méthode basée sur un algorithme ad hoc et des connaissances explicites (chapitre 4.2) :

- soit on remplace la méthode décrite au chapitre 4.2 par une méthode basée sur des exemples (ou plusieurs méthodes que l'on fusionne). Mais on impose que la décision de la méthode basée sur des exemples soit dans l'ensemble des adaptations possibles (chapitre 4.1), si ce n'est pas le cas on procède comme indiqué au chapitre 4.2.
- soit on utilise la méthode décrite au chapitre 4.2 et une (ou plusieurs) méthode basée sur des exemples, en concurrence, auquel cas on doit combiner ces approches afin d'obtenir une décision finale à partir des décisions intermédiaires.

La combinaison peut se faire via une méthode de fusion (chapitre 6.1).

- On utilise la méthode 4.2 au début, pour les 1ères adaptations, lorsque l'historique est trop petit. Ensuite on utilise une méthode basée sur des exemples (ou plusieurs méthodes que l'on fusionne) car il y a un historique sur lequel on peut se baser pour trouver un exemple similaire, et donc proposer une action plus pertinente.
- Un exemple pourrait être sous la forme d'une séquence des questions mal répondues (et le niveau) avec les adaptations pertinentes associées, avec aussi la séquence des modifications d'intensité (question/niveau, et adaptation).

Mais je pense qu'une approche basée sur des exemples (CBR ou Machine Learning supervisé) serait plus pertinente pour proposer uniquement une adaptation, pas de modification d'intensité.
En effet, la modification d'intensité est très procédurale, et serait probablement mieux traitée avec l'algorithme ad hoc (chapitre 4).

7- Conclusion

L'algorithme présenté au chapitre 4 est basé sur des agents qui ont la forme de règles complexes et sur un tableau noir qui modélise les variables d'état du système, partagées et dynamiques.
On stocke également des exemples pour les élèves qui ont passé la session jusqu'au bout.

Une fois récoltés de nombreux exemples d'apprentissage (quelques milliers), et si on souhaite améliorer encore les résultats, on pourra tester :

- des méthodes de data mining afin d'extraire des régularités, associations, statistiques,…
- des algorithmes de clustering afin de regrouper des élèves
- des méthodes d'apprentissage (Machine Learning vectoriel, ou Machine Learning basé sur des séquences type CRF), et éventuellement une méthode hybride.
- La méthode CBR où les cas sont les exemples, et éventuellement une méthode hybride.

This project was financed with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.