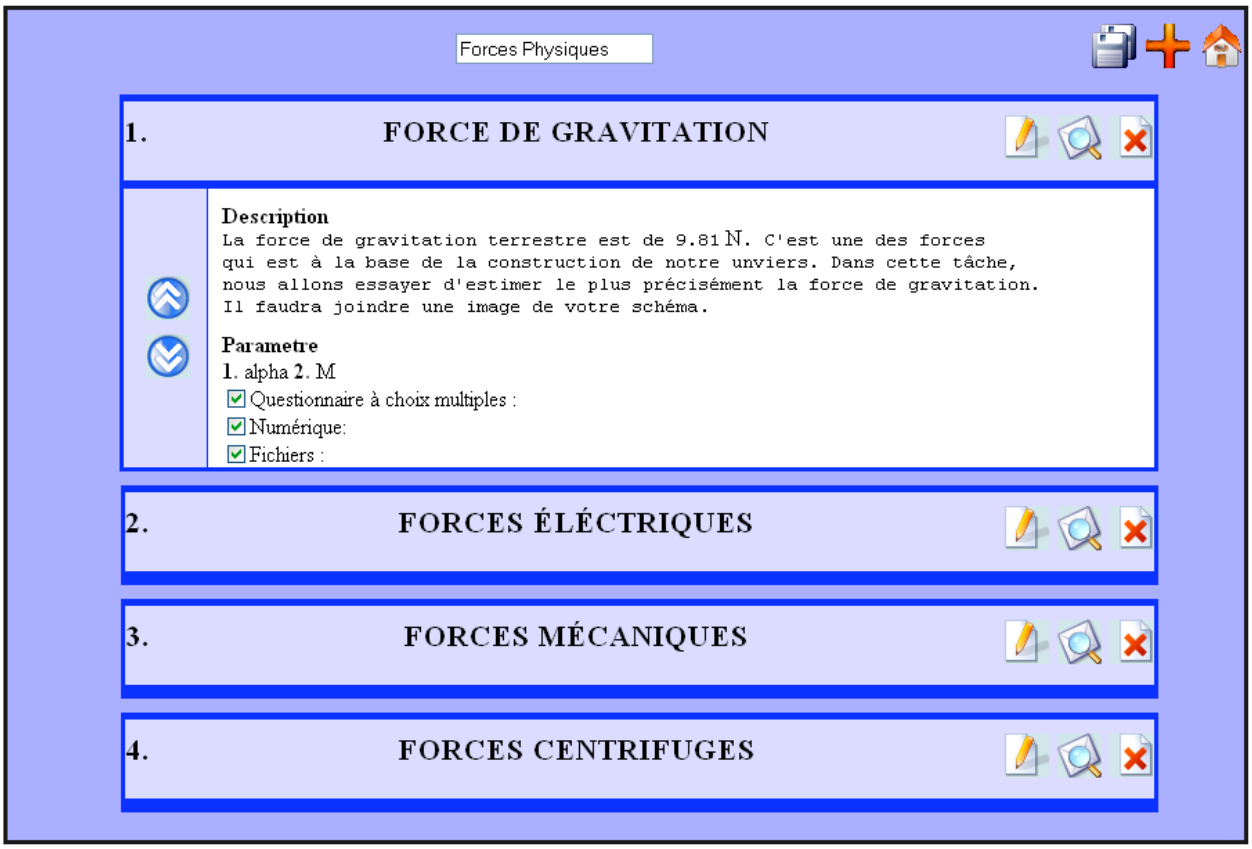
Correction automatique de TP d’automatique

Projet de Semestre de Luc Dubouloz

Hiver 2004



Professeur Denis Gillet

Assistant Anh vu Nguyen Ngoc

Table des matières

Résumé

Le projet correction automatique de TP d’automatique, fait partie du programme e-mersion, plus précisément de la partie eJournal. Le but de ce projet, est de fournir deux interfaces permettant la création et l’utilisation d’un protocole. En effet, pour chaque expérience physique, le professeur doit pouvoir générer une liste de taches regroupées dans un protocole, que l’étudiant doit remplir lors de la réalisation de l’expérience. Chaque tache est composée d’une description, de paramètres et de déliverables : questionnaire à choix multiples, scripts, numériques, fragments.

1. Introduction

i. Limite du protocole

Actuellement, dans le eJournal, il existe déjà un protocole pour chaque module. Cependant, il est statique et ne permet pas à l’étudiant de transmettre des informations au professeur, autre que les fichiers qu’il a créé durant l’expérience. Il contient une description de la tache, les paramètres à utiliser et le résultat. Pour avoir accès aux manipulations, l’étudiant doit envoyer des fragments (les pré-lab) à l’assistant qui les évalues, puis qui lui donne le droit d’accès. Ensuite, l’étudiant valide lui-même les taches via le protocole, en cliquant sur l’icône à cet effet (cf. fig. 1).

Pour connaître l’avancement du groupe, il faut donc voir quel est le dernier fragment envoyé.

Pour générer le protocole, le professeur doit générer un fichier XML. Il n’y a pas d’interface spécifique pour créer le protocole.

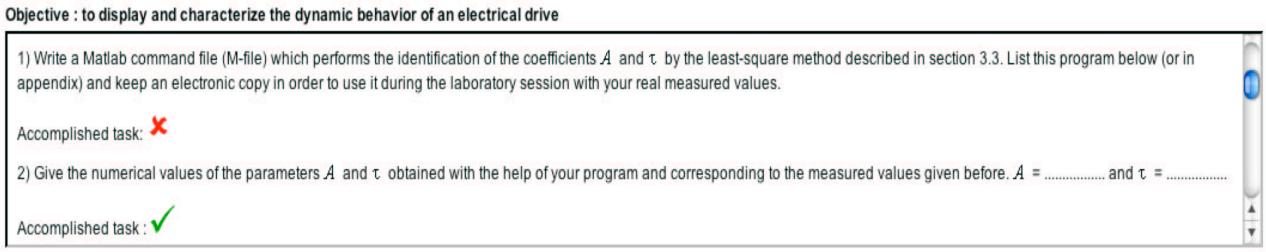


fig. 1 screen shoot du protocole existant

ii. Objectifs d’amélioration dans la version courante

Le projet, correction automatique de TP d’automatique a pour but de permettre un suivit précis et significatif de l’avancement de chaque groupe dans le protocole. De plus, il doit pouvoir faciliter la création de taches, leur édition et leur correction, grâce a quatre types de déliverables prédéfini : fragment, quiz, numérique et script.

On a donc une interface professeur. Cette interface permet l’édition d’un protocole. Tout d’abord, le professeur peut choisir de créer ou d’éditer un protocole existant. Ensuite, il peut éditer chaque une des taches du protocole séparément en remplissant la description, les paramètres s’il y en a, un questionnaire à choix multiples, des valeurs automatiquement corrigées ou un script et finalement, le type de fragment qu’il désire recevoir.

De plus, on a l’interface étudiant qui permet le remplissage du protocole et sa correction plus ou moins automatique. Il y a donc un résumé des taches, avec des liens sur chaque partie à remplir : questionnaire à choix multiples, auto corrigé (numérique et script), fragments que l’étudiant doit compléter pour chaque tache.

Nous verrons plus en détails le fonctionnement de ces deux interfaces dans les sections suivantes.

2. Choix des technologies

i. Introduction

Le choix des technologies a été fortement influencé par le fait que le projet est une application web. Pour cela, il faut des langages de programmation compatible avec n'importe quel browser, étant donné que l'application peut être lancée depuis n'importe quelle machine. De plus, il faut que la partie graphique soit suffisamment souple pour une adaptation facile au design du projet eJournal. C'est pour cela que nous avons choisit : - My SQL

- Tomcat & JSP
- Struts
- CSS

ii. Tomcat & JSP

Pour ce projet, la technologie JSP a été beaucoup utilisée pour sont côté dynamique. En effet, il y a beaucoup d'interactions entre l'utilisateur et le programme. Pour cette technologie le serveur Tomcat est la référence dans le domaine. Bien qu'il existe de nombreux serveur aux fonctionnalités semblables, il est le seul serveur respectant entièrement les dernières spécifications JSP et Servlet, de plus sont utilisation est totalement gratuite. Rappelons brièvement que la technologie JSP, réponse de SUN à ASP et PHP est actuellement une sinon la plus puissante façon de crée des applications internet. Pour le programmeur, la syntaxe du code est simple puisque c'est directement du langage JAVA enrichit de quelques fonctionnalités propres au JSP (cf. dernières spécifications JSP1.2 et servlet2.3). Le code est compilé lors de sa première exécution ce qui permet un traitement très rapide des requêtes. La puissance de JAVA alliée aux nombreux apis spécifiques que l'on peut se procurer sur le web permet de trouver très souvent des solutions simples à des problèmes qui semblaient complexes.
Voir plus de détails [Tomcat & JSP, Hunter 1999, Taylor 2002, Bergsten 2001]

iii. Mysql

Pour stocker les données, nous utilisons une base de donnée mysql parce qu’elle est gratuite pour des applications non commerciales, rapide, simple à installer et à configurer et s’intègre très bien dans une application codée en java puisqu’il existe des drivers(mm.mysql) pour accéder à cette base.

iv. CSS

Pour l’aspect purement graphique du projet, nous utilisons des **C**ascading **S**tyle **S**heet. Tout d’abord, il n’y a aucun droits à payer. De plus, c’est un langage très pratique qui permet de mettre toute les informations graphiques dans un fichier appart, permettant d’effectuer des changement rapides et simples sur le design du programme. En effet, il suffit de changer quelques options pour que le changement soit répercuté dans tout le projet. Ceci est très important pour l’uniformisation graphique du projet avec le eJournal.

[CSS]

v. Struts

Afin d’optimiser la qualité et la clarté du code, nous avons utilisé le framework Struts. En effet, il permet de séparer la partie présentation et la partie code en utilisant le modèle MVC (Modèle, Vue, Contrôleur ; cf. fig. 2).

Le contrôleur est chargé de faire le lien entre le modèle et la vue, en réceptionnant toutes les requêtes du client et en les redirigeant vers des contrôleurs spécifiques (décrit dans le fichier

struts-config.xml). Chaque action accède au modèle pour récupérer les données nécessaires au traitement de la requête du client.

La vue, est l’ensemble des pages JSP, utilisant Struts. Afin de faciliter la tache, il y a des libraires de tags prédéfinit (en gras) :

```
<html:form action="/doNewProtocol" method="GET">
 <div class="Section">
  <div class="ButtonFieldAction">
   <html:submit><bean:message key="button.submit"/></html:submit>
  </div>
  <bean:message key="title.newProtocol"/>
 </div>
 <div class="bodyNormal">
  <div>
   <bean:message key="label.nbTask"/>
   <html:text property="nbTask" maxlength="2" size="2" value="2"/>
  </div>
  <bean:message key="label.title"/>
   html:text property="moduleTitle" maxlength="50" size="55" value="Nouveau module"/>
</html:form>
```

Le modèle est indépendant du contrôleur, le choix des technologies est donc libre.

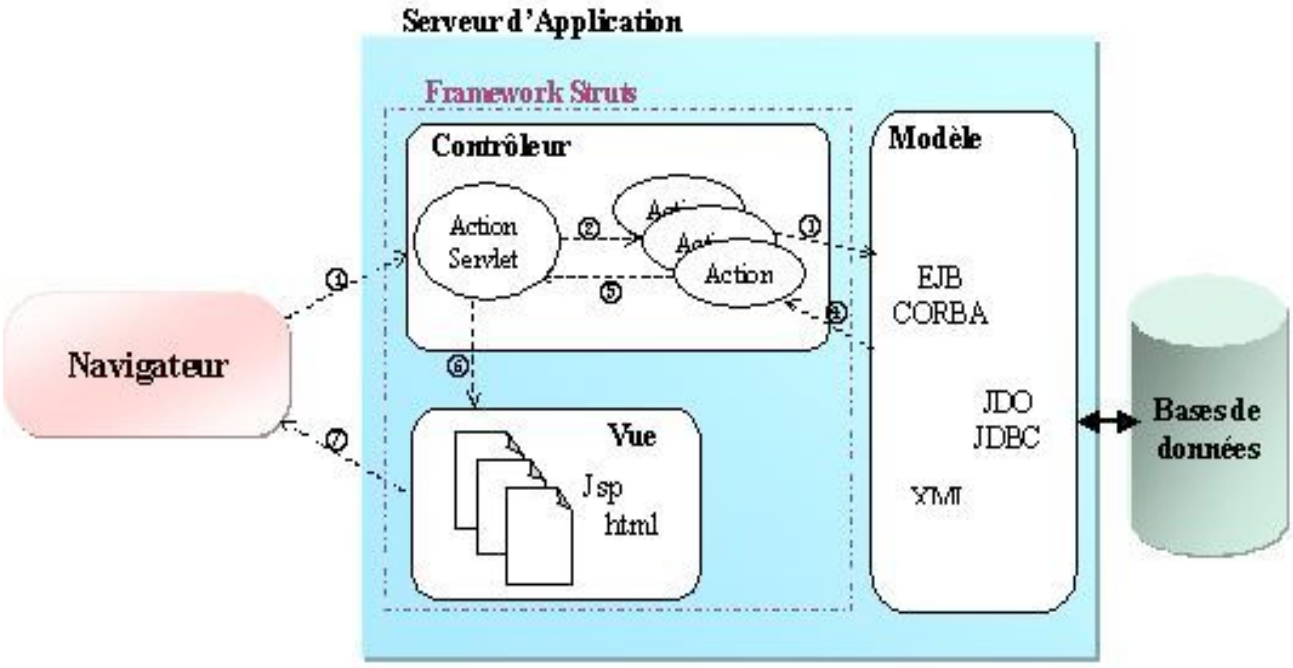


fig. 2. Model Vue Controle

adapté de [http://www.objectiva.fr/dossiers.2003-04-15_struts.php]

1. Le client envoie sa requête HTTP à l'application. Cette requête est prise en charge par le contrôleur principal, en l'occurrence ActionServlet;
2. La requête est redirigée vers le contrôleur adéquat;
3. Le contrôleur choisi prend en charge le traitement de la requête. Un dialogue avec les services métiers est entamé si nécessaire.;
4. Le modèle fournit les données demandées;
5. Le contrôleur principal est notifié du résultat du traitement. En cas de succès, les données sont encapsulées dans des JavaBean (ActionForm) et transmis à la JSP sélectionnée par le contrôleur;
6. La JSP construit la réponse suivant les données transmises;
7. La réponse est envoyée au navigateur.

[Struts & Goodwill 2003]

3. Module du professeur

i. Introduction

Le module du professeur a pour but de simplifier la création de protocoles dans une présentation de style formulaire. De cette manière, on a un protocole type, que le professeur peut éditer de manière simple. Ce protocole est en fait un prototype qu’il doit ensuite personnaliser pour chaque groupe d’étudiants. En effet, il doit être possible de choisir des paramètres différents pour chaque groupe, des questions différentes, etc. Cependant, il lui suffit de générer une donnée et en suite d’assigner les valeurs de son choix pour chaque étudiant.

ii. Architecture du module

Le fonctionnement du module du professeur est le suivant (cf. fig. 3) :

(1) Le professeur ouvre une session. Dans son browser s’affiche une page qui lui permet de charger ou de supprimer un protocole existant, parmi une liste de ceux qu’il a déjà créé. Sinon il peut décider de créer un nouveau protocole en introduisant le nom et le nombre de taches pour ce protocole. Ceci le conduit sur la page « résumé du protocole »

(2) La page « résumé du protocole » affiche toutes les taches créées, si l’on a chargé un protocole, ou crée un nombre de tache défini dans la page précédente, si c’est un nouveau protocole. De plus, on peut changer le nom du protocole, le sauver ou ajouter une tache. Par défaut, il n’y a que le nom des taches qui s’affiche. En effet, ceci permet de voir plus de taches dans la fenêtre et d’économiser de la place. En cliquant sur le nom, le résumé de la tache s’affiche.

Ce résumé, affiche la description de la tache, ainsi que le type de paramètres utilisés pour les calculs demandés (les valeurs seront associées pour chaque groupe individuellement, dans un autre module). En dessous, il y a trois cases à cocher. La première permet de créer un questionnaire à choix multiples. La seconde, de créer des modules automatiquement corrigés (script ou numérique). Finalement, on peut choisir si l’on veut que l’étudiant nous envoie un délivrable. Depuis là, on peut décider d’éditer la tache, voir le détail de celle-ci ou la supprimer. Si l’on veut changer l’ordre des taches, il suffit de cliquer sur les flèches dans la marge de gauche.

(3) Une fois rentré sur la page pour éditer la tache, on peut changer son nom, remplir sa description et ajouter des paramètres. Ces derniers sont en fait le nom du paramètre en symbole ou en toute lettre (il ne faut pas mettre de signe égal, il est automatique rajouté).
Ensuite, en fonction des cases cochées dans la page précédente, les différentes options sont présentes. Tout d’abord, le questionnaire à choix multiples, dont on voit seulement le nom, mais sur lequel on peut cliquer et voire toutes les questions. Ceci dans le même souci d’économie de place que pour les taches dans la page « résumé du protocole ». Par défaut, il n’y a qu’une seule question avec deux réponses possibles. On peut en rajouter une en cliquant sur le plus à coté du titre « Quiz ». De même pour les réponses avec le plus à coté de la question choisie. Après avoir rempli les champs, on peut décider de changer l’ordre des questions grâce aux flèches dans la marge. Les cases à cocher à coté des réponses permettent de spécifier si c’est la bonne réponse, il peut y en avoir plusieurs.
Ensuite, il y a la partie numérique et script. Par défaut, il y a un numérique et un script, mais on peut tout à fait décider de supprimer l’un ou l’autre. Pour ce qui est du numérique, le premier champ à remplir est l’url de la page de correction. C'est-à-dire, le lien sur lequel on va envoyer la réponse de l’étudiant pour qu’elle soit corrigée. Le nombre de réponses demandées, est le second champ. La page de correction va donc recevoir le nombre de champs et retourner une valeur correcte ou non.
Finalement, il y a les fragments. Comme pour les autres parties, il y a un fragment par défaut et l’on peut décider d’en ajouter ou d’en supprimer. Les fragments, permettent de choisir le type de fichier que l’étudiant doit envoyer au professeur. Ces types de fichiers sont définit dans la classe « Constant.java ». Voici quelques exemples :

- SnapshotFragment
- ParameterFragment
- ResultFragment
- ExternalFragment
- QuizFragment
- ScriptFragment
- SysquakeFragment
- BinaryFragment

A tout moment, on peut décider de sauver la tache éditée, ce qui l’enregistrera dans la base de données. De plus, on peut choisir de changer de tache directement depuis le menu des taches en haut. Si l’on ne clique pas sur « sauvegarder », les champs rempli seront simplement stocké dans la session et seront effacé lorsqu’on la quitte ou que l’on se déconnecte. Pour revenir au résumé, il suffit de cliquer sur la petite maison (home).

(4) La page « détail », permet une vue synthétique de la tache que l’on a créé. Chaque champ rempli est affiché de manière à avoir une idée globale de la tache. Aucun champ ne peut être modifié. De même que pour l’édition d’une tache, on peut naviguer grâce au menu en haut d’une tache à l’autre.

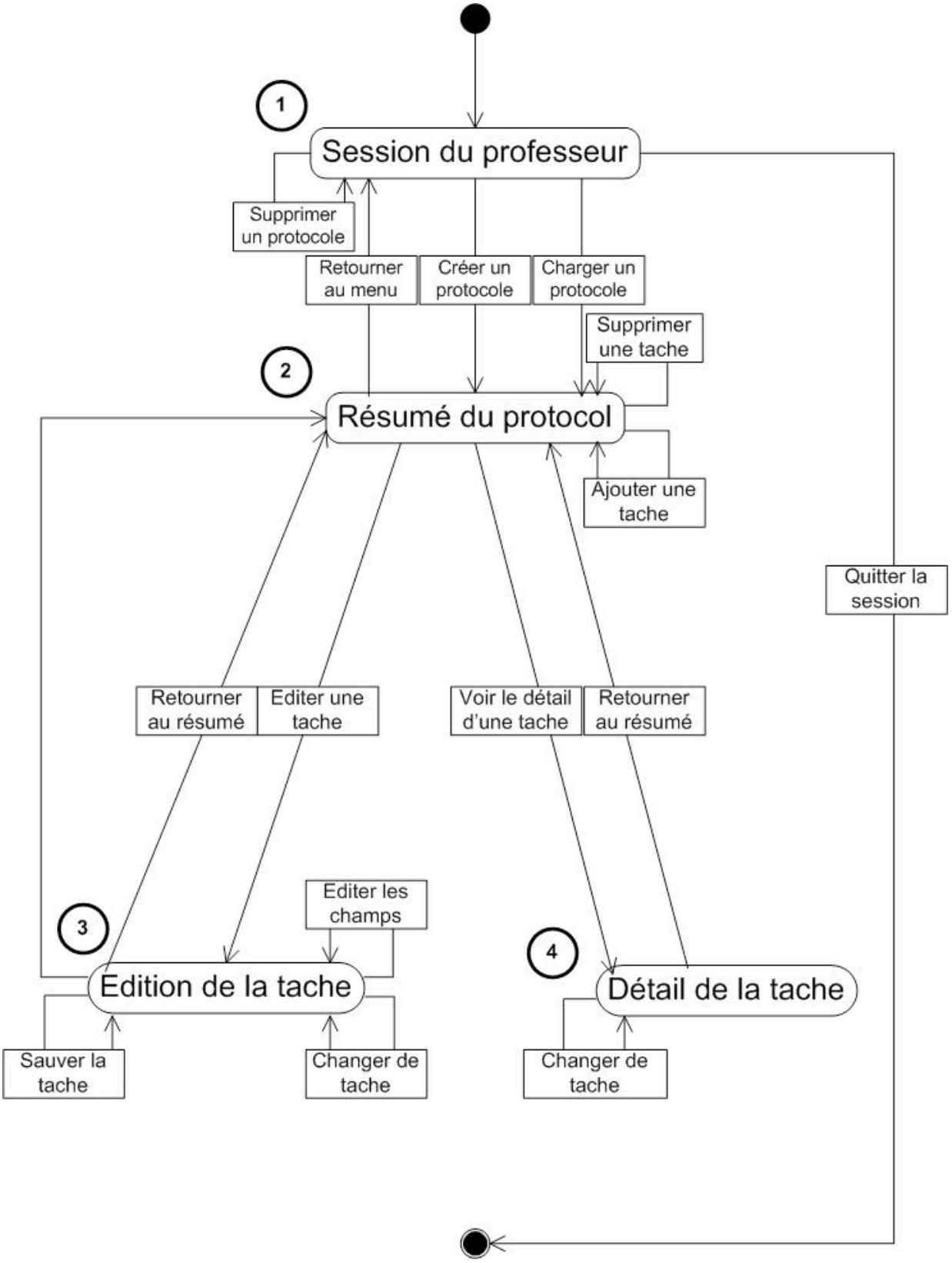


fig. 3. Diagramme d’activité du module du professeur.

iii. Conception de la base de donnée

La base de données est un « mapping » du schéma conceptuel du protocole. Elle a donc une structure assez simple, permettant une gestion facile et flexible des données. Ceci nous permet de créer et d’assigner dynamiquement les objets pour le protocole. Ici, on ne stockera que les informations générales, valables pour tous les groupes. L’architecture est la suivante (cf. fig.4) :

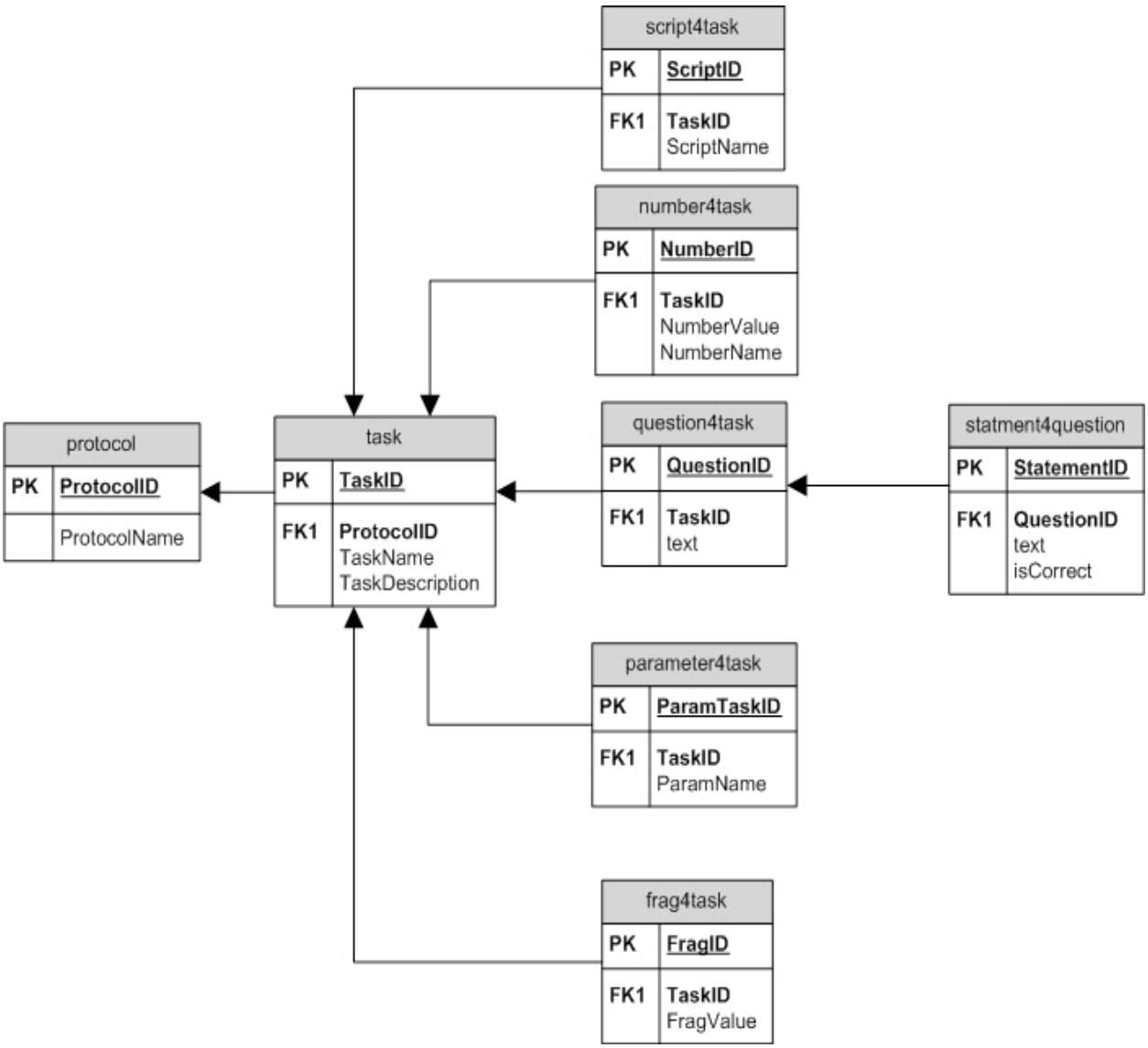


fig. 4. Diagramme de la base de donnée du module du professeur.

Table	Primary key (Int)	Foreign key (Int)	Champs 1 (text)	Champs2
Protocol	ProtocolID	/	protocolName	/
Task	TaskID	ProtocolID	TaskDescription	TaskName (text)
Parameter4task	ParamTaskID	TaskID	ParamName	/
script4task	ScriptID	TaskID	ScriptName	/
number4task	NumberID	TaskID	NumberName	NumberValue (int)
frag4task	FragID	TaskID	FragValue	/
question4task	QuestionID	TaskID	text	/
statement4question	StatementID	QuestionID	text	correct (tinyint)

ProtocolName : le nom du protocole

TaskDescription : la description de la tache

TaskName : le nom de la tache

ParamName : le nom du paramètre

ScriptName : l'url du script. C'est l'adresse ou va être envoyé le script pour être corrigé.

NumberName : l'url du numérique. C'est l'adresse ou va être envoyé le numérique pour être corrigé.

NumberValue : le nombre de résultat demandé à l'étudiant.

FragValue : le nom du type de fichier que l'étudiant devra envoyer.

Text (question4task) : l'intitulé de la question.

Text (statment4question) : l'intitulé de la réponse.

Correct : la valeur de la réponse. 1 pour bonne réponse et 0 pour mauvaise réponse.

4. Module de l’étudiant

iv. Introduction

Le module étudiant est le résultat du protocole créer par le professeur et de l’assignation de taches particulières et des paramètres. En effet, le professeur peut choisir les taches et les valeurs pour les calculs en fonction du groupe auquel appartient l’étudiant. Ce module doit permettre une vision de la progression, claire et efficiente, aussi bien à l’étudiant qu’au professeur. Finalement, l’étudiant peut être appelé à répondre au questionnaire, script, numérique ou fragment depuis n’importe quel endroit du eJournal. Ces sous sections doivent donc êtres sous forme de modules et simple afin que l’accès puisse se faire facilement sans trop de contraintes.

v. Architecture du module

Le fonctionnement du module de l’étudiant est le suivant (cf. fig. 5) :

(1) L’étudiant ouvre sa session et rentre sur la page résumé des taches. Sur cette page, s’affiche le nom des taches qu’il doit faire et le pourcentage d’avancement de chaque tache. Le calcul du pourcentage se fait de la manière suivante ; chaque question, script, numérique ou fragment rempli correctement lui ajoute des points. Les points sont répartis de manière égale pour chaque déliverable.
En cliquant sur le nom, de la même manière que pour les taches du professeur, le résumé de l’avancement s’affiche de manière plus précise. Chaque partie à remplir s’affiche sous forme de lien à cliquer pour accéder au formulaire. Si la partie questionnaire à choix

multiples, autocorrection ou fragment est déjà remplie et correcte, un petit vu s’affiche à coté et le lien n’est plus utilisable. Sinon, une croix accompagne par défaut le lien.
Lorsque l’étudiant clique sur un lien pour remplir le formulaire annexe, ce dernier est ouvert dans une nouvelle fenêtre. Ceci afin de permettre l’accès au formulaire de n’importe quelle partie du programme eMersion.

(2) Sur la page édition d’un script, l’étudiant doit écrire son script ou code, qui sera envoyé à l’algorithme de correction donné par l’adresse URL. S’il y a plusieurs scripts à remplir pour la même tache, chaque fois qu’il en aura fait un correctement, le champs ne s’affiche pas et un petit vu accompagne le titre.

(3) Pour la page édition d’un questionnaire, les questions assignées par le professeur s’affichent avec leurs réponses possibles. Seul une question totalement correcte est validée, c'est-à-dire que l’étudiant aura répondu le bon nombre de réponses et au bon endroit. La correction se fait instantanément et les questions justes ne sont plus réaffichées.

(4) La fenêtre de l’édition d’un numérique permet à l’étudiant de saisir le nombre de valeurs définies par le professeur. Chacune de ces valeurs sera passée en paramètre à l’algorithme de correction. De la même manière que pour le script, si les valeurs sont correctes, le numérique ne s’affiche plus.

(5) Finalement, la page fragment, est toujours accessible, même si elle est considérée comme correcte. En effet, pour la valider, il suffit de joindre un fichier de l’extension souhaitée par le professeur. Comme il n’y a pas de correction automatique, l’étudiant doit pouvoir avoir le choix à tout instant de changer le fichier qu’il veut envoyer. L’affichage se fait de la manière suivante ; pour chaque fragment demandé une liste déroulante, avec tous les fragments de l’étudiant correspondant au type demandé par le professeur, s’affiche. L’étudiant a le choix d’envoyer un seul fragment à la fois ou aucun.

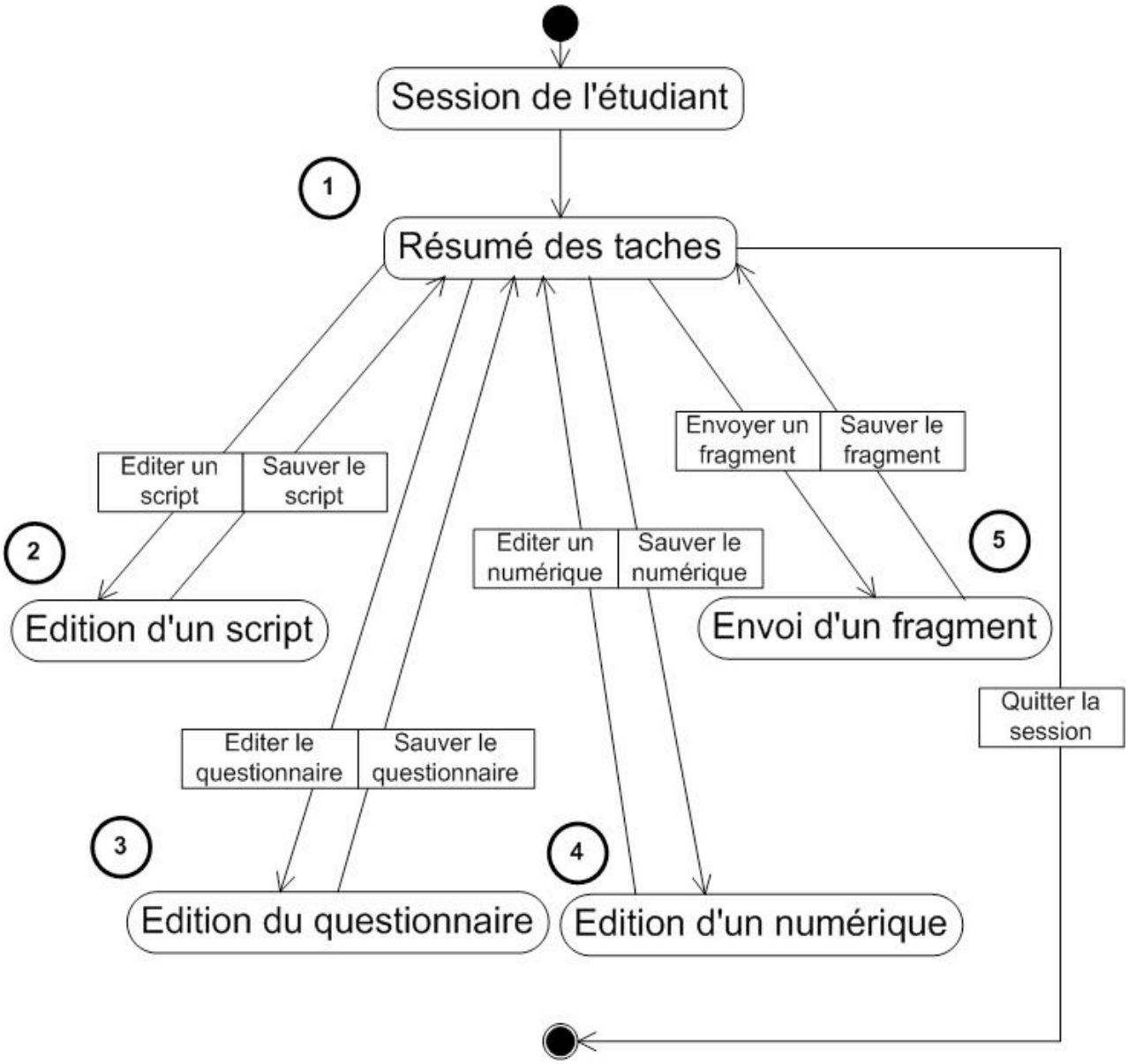


fig. 5. Diagramme d’activité du module du professeur.

vi. Conception de la base de donnée

Pour la base de données, nous avons utilisé la même structure que pour le professeur, tout en changeant les valeurs stockées en fonction des besoins (cf. fig. 6). En effet, pour chaque groupe, il est important de connaître le résultat de chaque déliverable et de savoir le nombre d’essais avant la réussite. De plus, il faut aussi avoir l’id de chaque déliverable correspondant du professeur pour pouvoir effectuer la correction. Pour ce qui est du fragment, il n’est pas copié une deuxième fois (déjà enregistré dans le eJournal), seul l’id suffit. Cette base de données sera remplie en partie par le professeur lors de l’assignation des taches aux groupes.

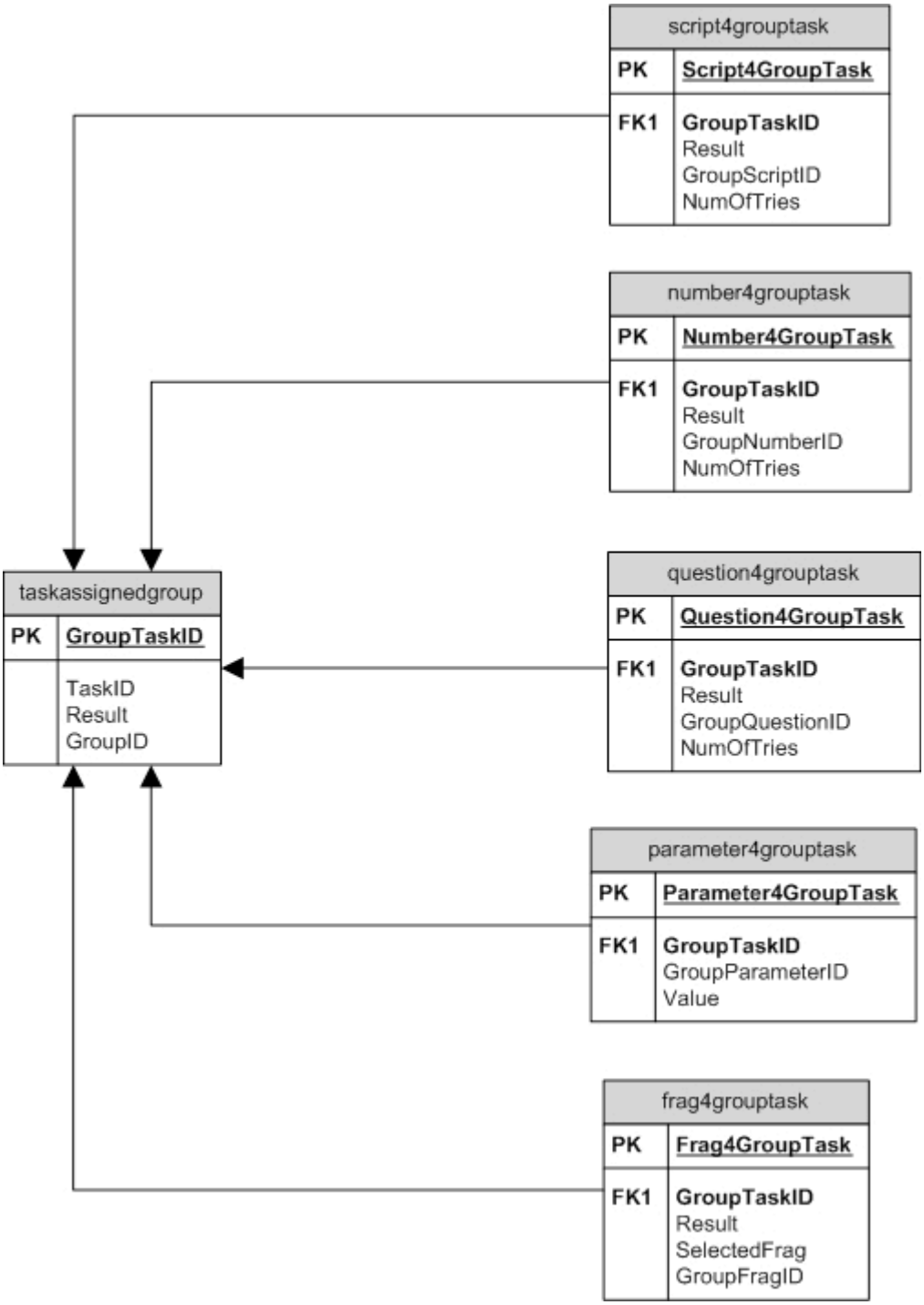


fig. 6. Diagramme de la base de donnée du module de l’étudiant.

table	Primary key (Int)	Foreign key (Int)	Champs 1 (int)
taskassignedgroup	GroupTaskID	/	Result
parameter4grouptask	Parameter4GroupTask	GroupTaskID	/
script4grouptask	Script4GroupTask	GroupTaskID	Result
number4grouptask	Number4GroupTask	GroupTaskID	Result
frag4grouptask	Frag4GroupTask	GroupTaskID	Result
question4grouptask	Question4GroupTask	GroupTaskID	Result

Suite :

table	Champs 2 (int)	Champs 3 (int)
taskassignedgroup	GroupID	TaskID
parameter4grouptask	GroupParameterID	Value
script4grouptask	GroupScriptID	NumOfTries
number4grouptask	GroupNumberID	NumOfTries
frag4grouptask	GroupFragID	SelectedFrag
question4grouptask	GroupQuestionID	NumOfTries

Result : Le résultat de l’étudiant pour la tache demandée

TaskID, GroupParameterID, GroupScriptID, GroupNumberID, GroupFragID, GroupQuestionID : L’id de l’objet dans la base de donnée du professeur.

GroupID : l’id du groupe (permet de retrouver les taches assignées au groupe)

Value : La valeur numérique du paramètre (différente pour chaque groupe)

NumOfTries : Le nombre de fois que l’étudiant a essayé de faire la tache demandée.

SelectedFrag : L’id du fragment que l’étudiant a décidé d’envoyer au professeur.

5. Conclusion

i. Remarques

- La réalisation de ce projet a été très instructive quand à la complexité de la gestion et de la réalisation. En effet, le cahier des charges a passablement évolué au cours de la réalisation, ce qui a demandé une adaptation du code et de la structure non négligeable.
- De plus, ce projet nécessite plusieurs technologies différentes et nouvelles. Ceci demande une bonne compréhension de chacune et leur utilisation correcte. En effet, il n’est pas évident de corriger des bogues lorsqu’ils peuvent se trouver à plusieurs niveaux du code.
- Finalement, ce projet sera intégré au eJournal dans la prochaine version. Il faut donc une compatibilité totale, de la base de données, mais aussi du design. Ceci est possible grâce à la modification simple de ce dernier, via les CSS.

ii. Perspectives

Les améliorations possibles du projet sont les suivantes :

- La possibilité de générer un fichier XML du protocole. Ceci afin de permettre une modification
- Etendre le type de déliverables possibles. D’une part, dans le type de fichiers que l’étudiant peut envoyer en tant que fragment, par exemple : fichier vidéo, image, différente sortes de script, etc. D’autre part, en étendant les déliverables corrigés automatiquement comme des fichiers XML, ou des images.
- Changer l’algorithme pour calculer l’avancement de l’étudiant. En effet, il serait plus judicieux de compter d’avantage les parties qui demande une réflexion plus grande et un temps de développement plus long. Ceci afin de mieux évaluer la progression du groupe.
- Permettre une gestion de la séquence dans laquelle l’étudiant doit résoudre les taches, afin de savoir quelle est la tache courante.

6. Bibliographie

i. Sites Internet

[eMersion] http://emersion.epfl.ch (dernière visite 1 Février)

[Tomcat] http://jakarta.apache.org/tomcat/ (dernière visite 1 Février)

[Java & JSP] http://java.sun.com/ (dernière visite 1 Février)

[CSS] http://www.w3schools.com/css/ (dernière visite 1 Février)

[Struts] http://www.objectiva.fr/ (dernière visite 1 Février)

ii. Livres

[Goodwill 2003] James Goodwill, Richard Hightower. Professional Jakarta Struts. IDG Books, 2003

[Hunter 1999] Jason Hunter, William Crawford (Traduction de Hervé Soulard). Java Servlets. O’reilly, 1999

[Taylor 2002] Art Taylor. JSP and Java. Prentice Hall PTR., 2002

[Bergsten 2001] Hans Bergsten. Java Server Pages. O’reilly, 2001