
Etude des objectifs d'une plate-forme de coopération pour les EIAH

Marilyne Rosselle

*ESIEE-Amiens
14 quai de Somme
80083 AMIENS CEDEX 03
Marilyne.Rosselle@esiee-amiens.fr*

RÉSUMÉ. La recherche sur les EIAH a produit de nombreux prototypes implantant de nombreuses fonctionnalités. Cependant il demeure difficile de mener une activité complète impliquant plusieurs fonctionnalités complémentaires avec un seul prototype. C'est pourquoi nous avons étudié la conception d'un atelier d'expérimentation facilitant l'utilisation de prototypes aux fonctionnalités complémentaires lors d'une activité (expérimentation ou séquence pédagogique). Nous avons utilisé une approche par composants. L'analyse des objectifs de l'atelier montre le besoin d'indexer les prototypes, de les utiliser conjointement et de gérer l'activité. Elle débouche sur la formulation de recommandations pour favoriser la communication et la coopération inter-logiciels et cela, dès la conception des produits.

MOTS-CLÉS : recommandations, approche par composants, plate-forme, coopération, interopérabilité, géométrie.

1. Introduction

La recherche en EIAH a produit des prototypes de logiciels éducatifs intelligents et interactifs pour l'enseignant et pour l'apprenant. Ils implantent des fonctionnalités diverses. Par exemple, *l'édition d'une figure géométrique*, *l'aide à l'élaboration d'une démonstration* (ou preuve) et *l'aide à la rédaction de cette démonstration* sont des fonctionnalités actuellement implantées dans le domaine de la géométrie. Cette recherche s'appuie sur le constat suivant : s'il est vrai que de nombreux prototypes sont développés dans les laboratoires, peu d'entre eux les quittent pour une utilisation effective dans les classes. Outre les raisons liées à la nature expérimentale des approches ou des moyens matériels mis en œuvre dans certains de ces prototypes de recherche, la raison la plus importante pour notre propos est que chaque prototype n'offre souvent qu'une partie des fonctionnalités existantes (explication ou simulation...). Par conséquent, il n'est aisé ni à l'enseignant de proposer une activité pédagogique impliquant plusieurs fonctionnalités complémentaires, ni à l'apprenant de repérer quelle fonctionnalité de quel prototype utiliser dans les tâches qu'il doit accomplir, ni aux chercheurs d'évaluer les prototypes ou d'expérimenter une nouvelle fonctionnalité.

Pour remédier à ces difficultés, nous aimerions qu'un utilisateur puisse employer l'ensemble des fonctionnalités déjà implantées dans différents prototypes. Pour cela deux approches sont possibles. La première approche consiste à refaire un produit intégrant plusieurs fonctionnalités en développant à nouveau entièrement toutes les fonctionnalités désirées. La seconde approche consiste à faire coopérer les prototypes au sein d'un environnement unique afin de tirer partie de leurs fonctionnalités complémentaires. Cette approche a l'inconvénient de ne pas pouvoir être directement mise en œuvre car ces prototypes n'ont été conçus ni pour être intégrés ni pour coopérer. Cependant de nombreux auteurs [BRUSILOVSKI 2002, KOEDINGER 1998] la privilégient actuellement en proposant des architectures de composants [SZYPERSKI 1998].

Nous pensons qu'il est nécessaire de favoriser l'utilisation conjointe de plusieurs prototypes possédant des fonctionnalités complémentaires. Ainsi, nous étudions les conditions requises pour amener les prototypes existants à coopérer. Pour cela, nous définissons un environnement permettant l'utilisation conjointe de plusieurs prototypes pouvant si possible coopérer voire interagir les uns sur les autres (interopérer) et ainsi partager des ressources et échanger des données. Nous appelons cet environnement un *atelier* [WASSERMAN 1989].

En « intégrant » des prototypes existants, il s'agit de faciliter la réutilisation de ceux-ci. En effet, cette réutilisation permet de capitaliser l'expertise nécessaire et d'optimiser les efforts. Par conséquent, elle facilite la conception et la réalisation d'expérimentations sans avoir à tout reconstruire. Il s'agit aussi de permettre à l'utilisateur de choisir les fonctionnalités dont il a besoin. Pour cela, nous présentons un éventail de fonctionnalités proposées par défaut. Elles servent de briques de base à la définition de l'activité. Il faut donc identifier les fonctionnalités souhaitées par

l'utilisateur, permettre un accès aisé à celles-ci (les indexer) et finalement faciliter la sélection et l'enchaînement de ces briques par l'utilisateur.

Pour ce faire, nous avons mis l'accent sur les trois axes de travail suivants : la conception de l'atelier; la définition des conditions de l'utilisation conjointe de plusieurs prototypes; et la formulation de recommandations pour favoriser la communication et la coopération inter-logiciels et cela, dès la conception des produits. Cet article présente donc une expérience de travail à partir de « composants » : la conception d'un atelier d'expérimentation de logiciels issus de la recherche sur les EIAH. Les choix techniques que nous avons fait au moment de l'implantation de l'atelier ne sont plus d'actualité. Cependant, les problèmes que nous sommes posés demeurent. Nous les formulons ici en restant aussi indépendant possible des contingences techniques, évolutives par définition.

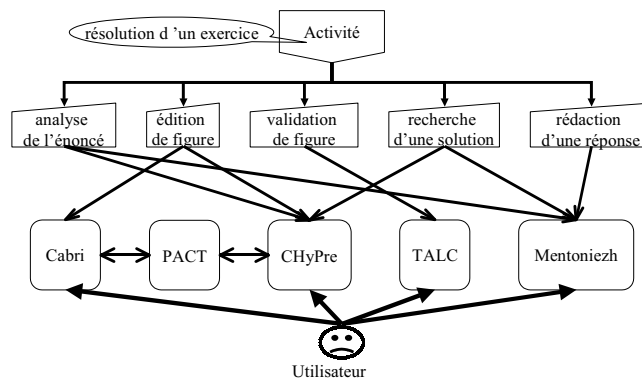
Nous proposons d'aborder cette problématique à partir de la réalisation d'un atelier d'expérimentation des EIAH de géométrie. Le choix du domaine d'application de ce travail se justifie premièrement, parce que les travaux antérieurs nationaux et internationaux (voir [Rosselle 2001] pour une revue de domaine) de la recherche en EIAH fournissent de nombreux prototypes ou produits aux fonctionnalités variées. Cette variété de fonctionnalités proposées permet en outre de disposer d'une combinatoire de possibilités de coopération entre prototypes. Deuxièmement, la géométrie est un domaine bien formalisé qui a donné naissance à de nombreuses représentations des concepts géométriques; ainsi c'est un bon champ d'expérimentation pour permettre l'échange de connaissances du domaine.

Le plan que nous suivons dans la suite est le suivant : nous commençons par poser le problème, puis nous décrivons l'atelier proposé (l'atelier) à la section 2. Nous étudions ensuite très précisément les objectifs de l'atelier proposé à la section 3. Cette étude est illustrée à la section 4. Elle débouche sur des recommandations qui sont synthétisées dans la conclusion (section 5).

2. Problématique

2.0. Position du problème

Nous illustrons à la Figure 1 la situation d'un utilisateur qui veut réaliser une activité : la résolution d'un exercice en géométrie. Il dispose pour cela de fonctionnalités (par exemple, *l'analyse de l'énoncé*). Ces fonctionnalités sont implantées dans des prototypes (par exemple, *l'édition de figure* est implantée dans CABRI-Géomètre et dans CHyPre). Sans atelier, l'utilisateur doit alors « jongler » entre les différents prototypes. De plus, certains prototypes peuvent tourner en parallèle. Par exemple le prototype PACT analyse les interactions de l'utilisateur avec Cabri ou CHyPre. Il faut donc penser à lancer les deux prototypes lors d'une édition de figure afin de permettre l'analyse des interactions.

Figure 1. *Exemple d'activité*

2.1. Description de l'atelier

Dans cette section, nous reprenons les descriptions du système apparues dans l'introduction. Description 1 : *l'atelier est une plate-forme logicielle*. L'atelier est un environnement informatique dans lequel différents prototypes de logiciels sont mis à la disposition d'un utilisateur. Description 2 : *l'atelier permet lors d'une activité d'utiliser dans un environnement unique l'ensemble des fonctionnalités déjà implantées dans différents prototypes*. Nous considérons un utilisateur qui réalise une activité nécessitant plusieurs prototypes. Chaque prototype peut tourner sur une machine distincte de celle sur laquelle l'utilisateur réalise son activité. Par conséquent, l'architecture de l'atelier est distribuée. De plus, les logiciels de l'atelier sont des prototypes dont nous voulons utiliser les fonctionnalités. Description 3 : *l'atelier permet à l'utilisateur de choisir les fonctionnalités dont il a besoin*. Description 4 : *l'atelier permet de réduire la charge cognitive de l'utilisateur, en lui présentant uniquement les informations pertinentes à un moment donné*. Les descriptions 3 et 4 débouchent sur l'idée que pour l'utilisateur, tout se passe comme s'il utilisait un « prototype virtuel » dont les fonctionnalités sont choisies.

Sur la Figure 2, nous avons, par exemple, trois prototypes P1, P2 et P3, fournissant respectivement les fonctionnalités A, B et C. L'objectif de l'atelier est que pour l'utilisateur tout se passe comme s'il avait sur sa machine toutes les fonctionnalités choisies dans les différents prototypes : ici la fonctionnalité 'A' de P1, la fonctionnalité 'B' de P2 et la fonctionnalité 'C' de P3. L'atelier décrit comprend un équipement (« médiateur ») qui sert d'intermédiaire (partie hachurée) entre les différents prototypes impliqués et le « prototype virtuel » (dans l'hexagone). Les flèches sur cette figure correspondent à une mise en relation entre les divers éléments. Nous avons implanté le médiateur et le prototype virtuel avec les technologies CORBA et java, où chaque prototype était inclus dans un composant.

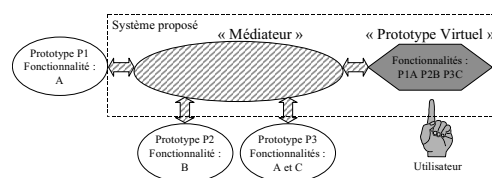


Figure 2. Le système proposé

Description 5 : l'atelier offre à des prototypes qui en ont la capacité¹ la possibilité d'interagir avec d'autres prototypes. Description 6 : l'atelier permet à ces prototypes de partager des ressources et d'échanger des données. Les descriptions 5 et 6 précisent certaines fonctions offertes par l'atelier : supporter l'interaction entre prototypes, le partage de ressources et l'échange de données. De plus, ils introduisent des propriétés qui permettent aux prototypes d'être utilisés par l'atelier : une capacité à interagir et à échanger des données.

3. Etude des objectifs

Le but de cette section est de permettre de cerner ce dont nous avons besoin. Nous voulons utiliser plusieurs logiciels. Sans l'existence d'un atelier pour permettre cette coopération de logiciels, l'utilisateur est confronté à plusieurs difficultés. Nous allons les passer en revue dans cette section. Résoudre chaque difficulté correspond à un ou plusieurs objectifs de recherche et à une ou plusieurs fonctions de l'atelier. Nous regroupons ces objectifs en trois grands « axes » suivant qu'il s'agit d'indexer les ressources, de les utiliser conjointement ou de les gérer.

3.0. Objectif : indexer

Il est nécessaire de connaître les fonctionnalités intéressantes de chaque prototype, c'est-à-dire celles qui peuvent être utilisées dans une activité. Dans l'exemple de la Figure 1, cinq prototypes sont à indexer : Cabri, CHyPre, Mentoniez, PACT et TALC. Il est nécessaire de décrire ces prototypes vis-à-vis des fonctionnalités qu'ils implantent. Nous regroupons les fonctionnalités suivant (a) qu'elles relèvent du domaine d'apprentissage, (b) qu'elles constituent une aide ou une fonctionnalité pédagogique, (c) qu'elles fournissent des outils ou des micromondes, (d) qu'elles relèvent de la dynamique de l'interaction de l'utilisateur avec le prototype ou (e) de l'interface utilisateur.

La *conséquence pour l'atelier* est qu'il doit disposer d'une base d'informations pour décrire et indexer les prototypes et les fonctionnalités disponibles. La *conséquence pour chaque prototype* est qu'il devrait permettre d'accéder indépendamment à ses différentes fonctionnalités. La *recherche nécessaire sur cet objectif* concerne la

¹ Nous précisons cette notion de capacité à la section 3.

normalisation de la description des prototypes. Elle s'appuie sur des modèles pour décrire (et indexer) une ressource pédagogique, plus particulièrement un prototype, afin de la retrouver et de faire appel à ses fonctionnalités. En effet, comment décrire une ressource pédagogique [GRANBASTIEN 2002], en particulier en ce qui concerne les prototypes ? Nous avons abordé cette problématique dans [ROSSELLE 2001]. Nous avons recensé les fonctionnalités liées à l'activité en résolution de problème de géométrie en classe de 4^{ième}. La liste que nous avons obtenue sert à décrire les prototypes de géométrie pour les indexer dans l'atelier. Elle est fonction des besoins et de l'évolution des travaux dans le domaine. Elle reste actuellement difficile à prendre en compte dans les normes existantes ou en cours d'élaboration.

3.1. Objectif : utiliser conjointement

3.1.0. Communication des données

Actuellement, si les utilisateurs réalisaient un exercice de géométrie de la lecture de l'énoncé à la rédaction d'une réponse, avec les différents prototypes existants, ils se trouveraient dans la nécessité de saisir des données plusieurs fois, sous des formats parfois différents. Pour l'exemple de la Figure 1, nous disposons initialement d'un énoncé du problème en langage naturel. Chaque utilisateur interprète cet énoncé afin de pouvoir fournir les informations adéquates aux prototypes dans une forme appropriée. L'apprenant extrait de l'énoncé les faits à partir desquels il peut raisonner et la conjecture qu'il doit prouver. Il utilise ensuite les faits identifiés pour construire la figure avec Cabri. Pour cela, il applique les outils d'édition de figure proposés. Il exploite la conjecture et une nouvelle fois les faits, lorsque Mentoniezh demande d'identifier les faits et la conjecture présents dans l'énoncé. Il utilise pour cela les menus proposés. Il effectue la même tâche (entrée des faits et de la conjecture) avec CHyPre grâce à d'autres menus. De son côté, l'enseignant identifie les hypothèses et la conclusion de l'énoncé. Il traduit ces informations dans le langage de représentation des connaissances de géométrie de Mentoniezh. Il traduit une nouvelle fois ces informations dans le langage de représentation des connaissances de géométrie de TALC. Il entre une fois encore ces informations dans PACT (par le biais de menus). De plus, différents logiciels communiquent les informations (faits, conjectures) directement à d'autres logiciels. En résumé, l'utilisation conjointe des cinq prototypes de notre exemple requiert de chaque utilisateur de multiples saisies sous plusieurs formes des mêmes informations, ainsi qu'une communication inter logiciels de certaines des informations. Le premier intérêt de l'atelier est de mettre au point un système de communication de données et de gestion des formats afin de supprimer ces saisies multiples.

Dès lors, la *conséquence pour l'atelier* est que celui-ci doit permettre le transfert de données à des logiciels divers sans multiplier les saisies. Pour cela, l'atelier doit assurer la communication entre les prototypes, d'où la prise en compte d'un média de communication et d'un protocole de transfert des données. Ainsi, la *conséquence pour chaque prototype* est qu'il faut assurer la communication des données au format adéquat c'est-à-dire que les données soient représentées dans un format

compris par le prototype. Deux possibilités sont envisageables. Premièrement, il existe un format standard de représentation des données. Deuxièmement, il existe un moyen de traduire les connaissances à échanger dans le langage de représentation de chaque prototype cible de ces données [DELEVENAY 1959]. Ces deux possibilités peuvent être utilisées conjointement. Elles ne sont pas bien sûr exclusives. La *recherche nécessaire sur cet objectif* concerne la gestion des formats de données entre différents prototypes. Elle comporte des concepts et des outils pour la communication de données entre applications. Elle consiste en la définition d'un interprète généraliste, permettant de traduire les connaissances d'une représentation vers une autre à l'aide de macro-définitions [MACRELLE 1998].

3.1.1. Monitoring

Le transfert de données est le premier maillon de la solution. La notion même de communication implique sur les prototypes des actions, des inter-actions. En effet, il est parfois nécessaire de prendre le contrôle de certains prototypes afin de mettre en évidence un objet ou bien de lancer une action. Par exemple, dans la Figure 1, PACT a pour but d'aider l'apprenant à partir de l'analyse de ses interactions avec Cabri. Dans Cabri, l'apprenant est libre d'agir. Il peut ainsi ne jamais atteindre le but fixé. Lorsque PACT détecte que l'apprenant est en difficulté, il peut afficher un message à celui-ci. Toutefois afficher un simple message est parfois insuffisant : par exemple, il peut être souhaitable de rendre saillants certains objets pour mettre en évidence une propriété de la figure en épaississant les segments d'une sous figure. Ainsi PACT doit pouvoir pour provoquer l'épaississement d'objets géométriques de Cabri.

La *conséquence pour l'atelier* est que l'atelier doit disposer d'un langage de commandes pour donner des ordres aux prototypes (les activer, les désactiver...). La *conséquence pour chaque prototype* est qu'il devrait permettre une certaine forme de prise de contrôle par un autre prototype. Le prototype est « scriptable » [RITTER 1996]. La *recherche nécessaire sur cet objectif* concerne la définition d'un ensemble de commandes pour agir sur tous les prototypes. Elle comporte des concepts, des langages et des normes [LTSC HTP 2003] pour commander un logiciel de l'extérieur. Le but est scripter tous les prototypes avec le même langage de scripts [RITTER 1996, ROSSELLE 2001].

3.1.2. Protocole de coopération des prototypes

Finalement, lorsque l'on rend possible pour les prototypes des inter-actions, il apparaît indispensable d'ajouter encore une notion de gestion des inter-actions : ordre d'apparition des prototypes, condition de leur clôture... Par exemple, dans la Figure 1, CHyPre et MentoniezH sont actifs en même temps. De même, Cabri et TALC sont actifs en même temps, pour que TALC puisse récupérer les objets créés avec Cabri et ainsi vérifier la conformité de la figure relativement à l'énoncé. Cependant le diagnostic de TALC ne doit être lancé que lorsque l'utilisateur a construit une figure avec Cabri, avec l'aide de PACT, et qu'il demande une validation.

La *conséquence pour l'atelier* est qu'il doit posséder un moyen de définir un protocole de coopération entre les prototypes. Ce protocole consiste à définir au cours du temps le statut (actifs, inactifs) des participants et les règles de participation (l'un après l'autre, en parallèle, *etc.*). L'atelier doit aussi scruter des états ou des variables d'un prototype afin de les utiliser dans le protocole de coopération. La *conséquence pour chaque prototype* est qu'il devrait pouvoir être rendu actif ou inactif, totalement ou partiellement. Nous retrouvons ici la propriété de scriptabilité. Il doit aussi permettre l'observation de certains états ou variables. Nous parlons, dans ce cas, d'états ou de variables « inspectables » (scrutable) [RITTER 1996]. La *recherche nécessaire sur cet objectif* concerne deux points. Le premier est la définition d'inspectables commun à tous les prototypes. Notre contribution ici consiste à expliquer comment rendre scriptable un prototype et inspectable des variables ou objets. Le second est la définition d'un protocole de coopération entre les prototypes formalisé avec la notion de scénario [ROSSELLE 2001].

3.2. Objectif : gérer l'activité

3.2.0. Interface

Tous les menus et toutes les fenêtres des prototypes ne sont pas pertinentes à chaque instant. De plus, dans le cas d'un affichage exhaustif de toutes les fenêtres et tous les menus, la charge cognitive dédiée à la gestion de l'interface serait trop lourde. Nous avons donc besoin de savoir comment gérer les informations présentées à l'utilisateur à travers différentes interfaces graphiques (barre de menus, fenêtres, *etc.*) produites par les différents logiciels. Ainsi, pour diminuer la charge cognitive de l'utilisateur, nous proposons de sélectionner et de faire cohabiter diverses fenêtres de présentation. Dans l'exemple de la Figure 1, les interfaces graphiques de Cabri, CHyPre, Mentoniez, PACT et TALC doivent cohabiter.

La *conséquence pour l'atelier* est qu'il doit sélectionner ce qui doit être présenté et organiser l'écran. La *conséquence pour chaque prototype* est qu'il devrait pouvoir exporter son interface graphique c'est-à-dire que son interface graphique doit être paramétrable afin de permettre à l'atelier de définir l'endroit où elle sera placée, son statut (actif, inactif), son état (*e.g.* visible), *etc.* L'idéal serait quelle soit tellement bien spécifiée et indépendante du prototype, que cela faciliterait sa re-programmation. Cela permettrait de l'adapter au goût du jour et aux spécificités de la nouvelle activité pour laquelle le prototype a été choisi. La *recherche nécessaire sur cet objectif* concerne la gestion des interfaces graphiques. Elle nécessite des concepts, des langages pour exporter ou re-programmer les interfaces d'un logiciel. La recherche sur les interfaces est un domaine en soi, nous ne l'avons pas approfondi. Nous avons adopté une solution *ad hoc* afin de permettre le test des diverses propositions que nous avons faites. Notre contribution ici consiste à définir précisément la spécification du gestionnaire d'interface graphique.

3.2.1. Recueillir des données

Pour l'analyse de l'activité après celle-ci, l'atelier doit pouvoir reprendre ou rejouer une séquence d'interaction de l'utilisateur avec l'atelier ou l'un des prototypes qui participe à l'activité. L'atelier doit donc disposer de fonctions propres. Par exemple, nous imaginons que le prescripteur (enseignant ou chercheur) qui a choisi d'utiliser conjointement les cinq EIAO, a pour but d'étudier en quoi l'extraction de sous-figures permet à l'apprenant de mieux résoudre l'exercice. Pour cela, il recueille des données dans diverses conditions expérimentales. Les données dont il a besoin sont principalement liées à l'interaction de l'apprenant avec la fonctionnalité d'extraction de sous-figures offerte par CHyPre. Cependant, la sauvegarde de toutes les traces de l'interaction CHyPre-apprenant n'est pas pertinente (certaines actions de l'apprenant sur CHyPre n'étant pas liées à la fonction de CHyPre qui intéresse le prescripteur). De plus, la granularité des informations sauvegardées doit être suffisante, pour que le prescripteur puisse saisir la sémantique de l'action qu'entreprend l'apprenant (sans toutefois être trop fine, car dans ce cas, les informations recueillies risquent d'être inexploitable). Par exemple, il est inutile de mémoriser que l'apprenant clique à tel endroit de l'écran, mais il est utile de savoir que l'apprenant clique sur le bouton qui permet la création d'un nouveau calque (extraction d'une sous figure, visualisée dans une nouvelle fenêtre graphique).

La *conséquence pour l'atelier* est qu'il doit permettre de sauvegarder les traces (ou l'historique) de l'interaction de l'utilisateur avec l'atelier et les divers prototypes et de choisir les traces à sauvegarder parmi celles que proposent les différents prototypes. La *conséquence pour chaque prototype* est qu'il devrait permettre de sélectionner les interactions utiles à l'atelier. Pour cela il doit définir la sémantique des interactions de l'utilisateur avec le prototype. Cela permet à la fois de diminuer le nombre de traces à sauvegarder et d'obtenir des traces plus facilement exploitables. La *recherche nécessaire sur cet objectif* concerne la définition des traces pertinentes à sauvegarder dans le but de collecter des données exploitables sur l'activité. Elle nécessite des concepts, des langages et des outils pour la collecte de traces d'interactions didactiques. Nous donnons un exemple d'exploitation des données sur l'interaction du sujet avec un prototype à la section suivante.

4. Exemples de mise en œuvre : observables

L'analyse de l'expérimentation menée dans l'atelier passe par le recueil des informations concernant les interactions du sujet avec chaque outil. Nous avons montré en 3.2.1 que dans son interaction avec l'outil, le sujet produit des événements sémantiques que nous structurons en « observables » intégrant une sémantique. Pour associer aux événements sémantiques les observables adéquats nous avons défini une grammaire. S'appuyant sur cette grammaire, un interprète fabrique les observables par composition à partir des événements sémantiques. Le principe de fonctionnement ici est analogue à celui d'une calculatrice à pile. Les opérandes et opérateurs sont entrés dans un certain ordre (qui dépend de l'arithmétique choisie : infixée, postfixée ou préfixée). Lorsqu'une opération est possible, les opérateurs et opérandes associés sont dépilés et le résultat empilé. Dans notre cas, les événements

sémantiques sont empilés. Ils sont dépilés pour composer un observable. L'observable ainsi composé est envoyé dans un flux de données afin d'être stocké ou utilisé par un outil. En pratique, ce flux arrive dans une file d'observables. Ainsi à la fin de l'activité, où à la demande, la file d'observables peut être vidée pour stockage ou exploitation par un outil adapté. C'est dans ce dernier cas, qu'une file s'avère plus appropriée qu'une pile.

Pour illustrer le principe de la « consommation des événements sémantiques », nous donnons un l'algorithme simplifié (*cf.* Figure 3) dans le cas où toutes les macro-définitions ne sont composées que d'un seul événement sémantique.

```

\DEBUT_ALGORITHM
\PREREQUIS une pile d'événements sémantiques non vide (nommée eve)
\POSTREQUIS la pile eve vide
\POSTREQUIS alimente une pile d'observables (nommée obs)
modif=vrai,
\TANTQUE modif and pileNonVide(eve)
  \COMMENTAIRE lire un événement sémantique dans la pile
  evesem := depiler(eve);
  modif := faux;
  \POUR chaque macrodefinition
    \COMMENTAIRE lire le corps de la macrodef : l'événement sémantique
    evesembut := macrodef. corps();
    \SI unification(evesem;evesembut)
      modif :=vrai;
      \COMMENTAIRE ajouter la tete de la macrodef unifiée dans la pile obs
      empiler(macrodef. tete(), obs);
      \COMMENTAIRE gérer les macroobservables
      reduireobservables (obs);
      viderpile (eve);
      break;
    \FINSI
  \FINPOUR
  \SI non(modif)
    \COMMENTAIRE remettre la pile en état avant de sortir de la boucle
    empiler(evesem);
  \FINSI
\FINTANTQUE
\FIN_ALGORITHM

```

Figure 3. Réduction de la pile d'observables (algorithme simplifié).

Cet algorithme est lancé chaque fois qu'un événement sémantique est produit. Il nécessite une pile tampon, pour qu'il n'y ait pas de modification de la pile d'événements sémantiques pendant l'exécution de l'algorithme. À chaque fois qu'un observable est produit, on essaie de réduire la pile des observables, c'est-à-dire de diminuer le nombre de ses éléments. De plus, on vide la pile d'événements sémantiques car il n'y a plus continuité dans la succession des événements. L'exemple suivant illustre le problème. Dans cet exemple, les éléments de la pile sont des chiffres car il n'est pas important de savoir quels événements sémantiques ils représentent. A l'instant t , La pile contient : 1 2. L'élément 3 arrive. L'algorithme de réduction est lancé. Or il existe justement une macro-définition : $\langle O \text{ Cons } 2 \ 3 \rangle$. Autrement dit, lorsqu'on rencontre les événements sémantiques 2 suivis de 3, on peut construire l'observable O . L'observable O est donc créé. 2 et 3 sont dépilés. La pile contient alors : 1. L'élément 4 arrive. La pile contient alors : 1 4. Après la construction de l'observable et l'arrivée de l'élément '4', il serait erroné de pouvoir déduire un observable à partir de la succession des événements '1' et '4'. Il faut considérer les éléments à partir de l'arrivée de 4. D'où l'action de vider la pile après la génération d'un observable. Nous implantons la génération des observables dans un interprète de macro-définitions, qui est le même que celui utilisé pour la gestion

des formats. Un interprète à pile convient parfaitement pour composer les observables au fur et à mesure.

5. Conclusion et perspectives

Nous synthétisons les besoins mis en évidence lors de l'étude des objectifs de l'atelier en deux groupes. Nous avons d'une part les besoins fonctionnels de l'atelier. Ils permettent d'ébaucher un cahier des charges macroscopique pour ce dernier, détaillé dans [ROSSELLE 2001]. Nous avons d'autre part les propriétés des prototypes pour qu'ils soient utilisés facilement avec d'autres prototypes dans l'atelier. Ils permettent d'ébaucher des recommandations. Nous avons recensé sept propriétés. Trois de ces fonctionnalités ont été proposées par [RITTER 1996]. Nous avons montré leur intérêt pour l'atelier. Nous les rappelons ci-dessous. Le prototype communicant et coopérant est :

1. inspectable (c'est-à-dire permettre qu'on observe certains de ses états ou de ses variables/objets);
2. traçable (c'est-à-dire fournir les traces intelligibles de l'interaction de l'utilisateur avec lui);
3. et scriptable (c'est-à-dire permettre une certaine forme de prise de contrôle, dont d'une part être rendu actif ou inactif, totalement ou partiellement et d'autre part permettre de collecter uniquement les interactions jugées utiles).

À ces dernières nous ajoutons, que le prototype est :

4. indexable (c'est-à-dire pouvoir être décrit afin d'être retrouvé et d'accéder à ses fonctionnalités);
5. interface-exportable (c'est-à-dire pouvoir exporter son interface graphique, ou permettre sa reprogrammation);
6. fonctionnalités-indépendant (c'est-à-dire permettre d'accéder indépendamment à ses différentes fonctionnalités);
7. formats-normalisé (c'est-à-dire assurer la communication au format adéquat des données).

Cette liste de propriétés permet à un prototype de participer pleinement et activement à l'atelier. Elle constitue une liste de recommandations pour favoriser la communication et la coopération inter-logicielle et cela dès la conception des produits. En ce qui concerne les prototypes existants, il faut se demander ce qu'il faut leur ajouter pour les doter de ces propriétés. Des solutions techniques dépendantes des conjectures techniques existent. Lors de nos choix techniques pour concevoir l'architecture de l'atelier, nous avons opté pour les technologies et java CORBA [ORFALI 1997]. Aujourd'hui si nous devons faire des choix techniques, nous irons plus volontiers vers les webservices en architecture '.net' ou J2EE qui utilisent SOAP/XML/UDDI/HTTP et/ou les servlets/EJB. Mais que nous réserve l'avenir ? Nous pensons que la spécification des propriétés des prototypes peut aider à les adapter aux évolutions technologiques inévitables. Reste à déterminer si toutes ces propriétés sont nécessaires et suffisantes.

6. Références bibliographiques

- [BRUSILOVSKI 2002] Brusilovski P. and Su H.-D., Adaptive Visualization Component of a Distributed Web-based Adaptive Educational System. *Intelligent Tutoring Systems, ITS'2002*, Biarritz, Springer, Paris, juin 2002.
- [DELEVENAY 1959] Delevenay, E., *La machine à traduire*, Que sais-je ?, n°834, P.U.F., Paris, 1959.
- [GRANBASTIEN 2002] Grandbastien M., Bouyt B., Claës C., Bourda Y., Duval E., Les ressources numériques : Enjeux de la normalisation. *TICE'2002*, Villeurbanne, nov. 2002.
- [KOEDINGER 1998] Koedinger, K. R., Suthers, D. D. and Forbus, K. D., Component-Based Construction of a Science Learning Space. *Intelligent Tutoring Systems, ITS'98*, San Antonio, Texas, USA, pp. 166-167, LNCS 1452, Springer, 1998.
- [MACRELLE 1998] Macrelle, M. and Desmoulins, C. Macro-definitions, a basic component for Interoperability between ILEs at the Knowledge Level: Application to Geometry. *Intelligent Tutoring Systems, ITS'98*, LNCS 1452, pp. 46-55, San Antonio, Texas, USA. Springer Verlag, 1998.
- [ORFALI 1997] Orfali, R. and Harkey, D., *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, New York, 1997.
- [RITTER 1996] Ritter, S. and Koedinger, K. R., An Architecture for Plug-in Tutor Agents. *Journal of Artificial Intelligence in Education*, vol. 7(3/4), pp. 315-347, 1996.
- [ROSSELLE 2001] Rosselle, M. Conception d'un atelier d'expérimentation de logiciels éducatifs. Application en géométrie. Thèse de doctorat, Université Henri Poincaré - Nancy 1, 2001.
- [SZYPERSKI 1998] Szyperski, C., *Component Software*. ACM Press, Addison-Wesley 1998.
- [WASSERMAN 1989] Wasserman, A. I., Tool Integration in Software Engineering Environments. *Software Engineering Environments*, LNCS 467, pp. 137-149, Fred Long Ed. Springer-Verlag, 1989.

6.0. Références sur le WEB.

- [LTSC HTTP 2003] LTSC. Learning Technology Standards Committee <http://ltsc.ieee.org/>
Dernière visite le 21/02/2003.