

# Importation de traces existantes dans un système de gestion de bases de traces

Mohamed Besnaci<sup>1</sup>, Nathalie Guin<sup>2</sup> & Pierre-Antoine Champin<sup>2</sup>

<sup>1</sup> Université BADJI MOKHTAR - ANNABA

<sup>2</sup> Université de Lyon, CNRS Université Lyon1, LIRIS, UMR5205, F-69622, France

**Résumé** Les systèmes de gestion de bases de traces (SGBT) proposent des fonctionnalités de transformation et de requêtage sur les traces, pouvant intéresser plusieurs utilisateurs des systèmes tracés. Notre objectif est d'assurer l'importation de traces variées au kTBS, un SGBT développé par l'équipe SILEX du laboratoire LIRIS. Pour pallier au problème de la variété des traces, l'idée est de définir un collecteur pour chaque système tracé. Pour définir un tel collecteur, le concepteur du système tracé est invité à définir son modèle de trace dans kTBS ainsi que les correspondances (Mapping) entre les éléments de ce modèle et les éléments constituant ses traces à importer. Après cette phase, la collecte va consister à générer des traces modélisées à partir des traces existantes et du Mapping déjà défini.

## 1 Introduction

Bien que la naissance des traces informatiques date de plusieurs dizaines d'années, elles ont subi une mutation importante cette dernière décennie. En effet, de nombreux systèmes informatiques ont commencé à produire et/ou exploiter les traces, mais chacun avec son propre format, son propre contenu et pour ses propres objectifs. Le concept de trace s'est bien développé, par rapport aux formats possibles adoptés, aux contenus considérés et aux exploitations visées. Cependant, des défis majeurs se posent : celui de leur compréhension et celui de leur interopérabilité. Le problème de compréhension se pose généralement durant les processus d'exploitation des traces ayant pour but la visualisation, l'analyse et la déduction de connaissances. Notre travail s'intègre dans le domaine de l'intégration de données, pour justement proposer une solution à un des problèmes lié à l'interopérabilité. Nous visons à intégrer des traces variées dans un formalisme unique pour permettre ensuite leur exploitation.

Un système tracé est un système capable de générer des traces numériques d'interaction liées aux activités qu'il propose à l'utilisateur. Les systèmes tracés actuels utilisent des traces numériques très variées, à des fins aussi variées. Après consultation et analyse des traces d'un ensemble de systèmes tracés, nous avons abouti à quelques constatations :

- Plusieurs représentations existent, essentiellement des traces en XML, fichiers textes et sous forme de BDD (Bases De Données).

- Peu de traces sont associées à des modèles explicites (un document DTD est un exemple de modèle explicite pour des traces XML).
- Les traces peuvent être stockées dans des objets physiques uniques ou encore éparpillées dans plusieurs objets qui n'ont pas forcément la même structure. Une trace peut par exemple être stockée dans un seul document XML, ou dans plusieurs tables d'une BDD ou encore constituée d'un document texte et d'un morceau multimédia.
- Abstraitement, une trace doit englober l'inscription de temps avec d'autres données liées à l'activité exercée et au système tracé. À part la donnée temporelle (bien qu'elle puisse aussi avoir plusieurs variantes), les données inscrites dans les traces existantes sont très variées vu le grand nombre de possibilités d'activités dans les systèmes tracés.
- L'exploitant et l'exploitation de la trace sont d'autres critères différenciant les traces existantes. Selon leur niveau d'abstraction, les traces peuvent être destinées à des utilisateurs finaux à des fins d'orientation, d'assistance et d'auto-formation. Elles peuvent être destinées à des concepteurs, des enseignants, des techniciens ou des cognitivistes à des fins de diagnostic, d'évaluation, d'analyse, etc. Les traces peuvent être aussi exploitées par des programmes pour les mêmes buts, mais aussi à des fins d'inférence et de visualisation.

Devant ces constatations, la question est comment permettre à un utilisateur d'un système tracé, qui possède des traces d'activités issues de son système, de les importer en tant que traces modélisées (traces associées à leurs modèles explicites) dans un système adoptant un unique formalisme de représentation des traces, afin de pouvoir profiter de ses fonctionnalités ?

Plus précisément, l'objectif est : de permettre l'importation de traces variées au système kTBS (système de gestion de traces que nous présenterons dans la section suivante) et ce à des fins de transformation, de requêtage et de visualisation. Pour pallier au problème de la variété des traces existantes, l'idée est de définir un collecteur pour chaque système tracé. Pour définir un tel collecteur, l'utilisateur du système tracé est invité à définir son modèle de trace dans kTBS. Il doit aussi définir les correspondances entre les éléments de ce modèle et les éléments présents dans ses traces. Ainsi, le système qui s'occupe de la définition des collecteurs doit garantir deux tâches essentielles : la création du modèle de trace et la définition des correspondances (Mapping) servant à la création du collecteur. Une fois créé, ce collecteur permettra de prendre en entrée des traces du système dont on souhaite collecter les traces et de fournir en sortie des traces conformes au modèle défini. Les correspondances faites suite au Mapping sont à la base du noyau du collecteur défini.

Le problème de la collecte des traces a été largement abordé dans les travaux sur les traces d'interaction. En effet, la collecte est une étape critique dans le processus logique de traitement des traces. Son cas le plus usuel consiste à instrumenter les systèmes tracés afin de produire des traces de format et structure spécifiques [13](ou une combinaison de types de trace dans certains cas comme celui de [10]). À la différence de ces situations où la collecte et l'exploitation des

traces sont à la fois assurées par le même système, il existe d'autres situations où ces deux traitements sont séparés. Dans un travail comme celui de [4] ou [15], le rôle du système tracé se résume à la collecte. Les traces collectées sont destinées à des plateformes dédiées au stockage et à l'analyse [3][5]. Notre travail s'intègre dans ce dernier processus et essaye de résoudre le problème d'importation de traces variées dans une des plateformes de gestion des traces. La solution que nous proposons est destinée à des utilisateurs non informaticiens, ce qui implique qu'elle doit être loin des manipulations techniques comme celles proposées avec le langage UTL [6].

Dans la section suivante (section 2), nous présenterons la définition de quelques concepts liés aux traces. La section 3 sera réservée au détail de notre processus d'importation de traces en expliquant chacune de ses étapes : réécriture, création du modèle de trace et Mapping. xCollector, l'outil matérialisant ce processus sera présenté en section 4 en présentant ses fonctionnalités essentielles et à travers la présentation d'un cas réel d'utilisation. La section 5 sera la conclusion de ce travail.

## 2 Concepts autour des traces

Les traces numériques sont des inscriptions temporalisées liées à un système informatique donné et ce pour diverses exploitations. Dans cette section, nous présentons les définitions proposées par l'équipe SILEX du laboratoire LIRIS sur le concept de trace, ainsi que les concepts inhérents.

Selon l'équipe SILEX du LIRIS [5], une *trace* est composée d'éléments temporellement situés appelés *obsels* (pour *observed element* / élément observé). Ces derniers sont des éléments considérés comme potentiellement porteurs de sens dans l'activité tracée. Un obsel est constitué essentiellement : d'un type rattachant cet obsel à une catégorie explicite d'éléments observés, et d'un ensemble d'*attributs* de la forme <attribut : valeur>.

Une *m-trace* (pour *modeled trace* / trace modélisée) est une trace associée à un modèle qui en fournit un guide de construction et de manipulation. Un modèle de trace doit définir :

- La manière de représenter le temps.
- Les types d'obsels permettant de décrire l'activité.
- Pour chaque type d'obsel, les types d'attributs possibles.
- Les types de *relations* binaires que peuvent entretenir les obsels entre eux.

Un Système de Gestion de Bases de Traces (*SGBT*), est un outil informatique permettant manipulations et transformations de traces modélisées. En effet, un SGBT joue le même rôle qu'un SGBD (Système de Gestion de Bases de données) dans les applications standard, mais gère plutôt des traces modélisées (m-traces). Le SGBT est alimenté par un ensemble de collecteurs, dont le rôle est de récolter les informations nécessaires à la constitution des traces.

*kTBS* (kernel for Trace Based Systems), est un système informatique mettant en pratique la notion de SGBT, développé au sein de l'équipe SILEX du laboratoire LIRIS. kTBS propose plusieurs fonctionnalités de gestion des traces,

à savoir la création, l'interrogation, la transformation et la visualisation. Le seul formalisme utilisé pour représenter les traces ainsi que leurs modèles dans kTBS est le standard RDF, vu ses possibilités de description structurelles et sémantiques.

### 3 Importation des traces

Dans un objectif d'importation des traces dans kTBS, notre processus de génération de collecteurs peut traiter trois types de traces : texte, XML et BDD. Deux phases importantes doivent être effectuées pour qu'un utilisateur puisse transcrire ses traces brutes en traces modélisées importables : la création ou la génération du collecteur et son utilisation.

#### 3.1 Génération du collecteur

Le rôle essentiel des collecteurs que le générateur doit produire est de construire des traces importables au kTBS en se basant sur les traces existantes en entrée. Les traces importables sont des traces conformes et associées à des modèles kTBS de traces. Le problème revient donc à créer un modèle de trace pour chaque collecteur et à construire des traces qui lui sont conformes à partir de celles fournies en entrée. La question qui se pose est : comment former des traces kTBS à partir des exemples de traces externes en entrée ? Pour répondre à cette question, on propose de définir des règles de transformation qui vont permettre de chercher et calculer les éléments de la trace kTBS à partir de la trace fournie en entrée. La définition de règles de Mapping entre les éléments de la trace externe et les éléments du modèle de trace est à la base de cette phase. Le Mapping sert ainsi à définir les correspondances entre tous les éléments du modèle kTBS de trace et leurs éléments instances dans les traces en entrée. Il doit être effectué de la façon la plus générale possible pour qu'il reste pertinent quelles que soient les traces introduites. Afin de pouvoir créer des modèles kTBS de traces accompagnés des règles de Mapping capables de convertir des traces externes en traces importables respectant ce modèle, un processus en trois étapes est proposé : la réécriture, la création du modèle et le Mapping (Figure1).

**Réécriture** Compte tenu des constatations annoncées en introduction, trois formats de traces sont essentiellement distingués. Pour simplifier le processus d'importation des traces au kTBS et éviter le traitement spécialisé pour chaque format, on a choisi de considérer XML comme format pivot. Autrement dit, les traces de format non XML seront réécrites en XML, ce qui va ainsi unifier le processus d'importation. XML [16] est un standard fortement utilisé pour structurer des contenus et reconnu par ses possibilités de sérialisation. Ce qui motive aussi le choix de XML, ce sont les possibilités qu'il offre en termes d'outils d'interrogation, de transformation et de repérage.

La réécriture est une étape préparatoire par rapport au Mapping. Son objectif peut être, selon le type de l'entrée, la structuration ou la restructuration des traces avant leur chargement. Ainsi, la réécriture va consister à :

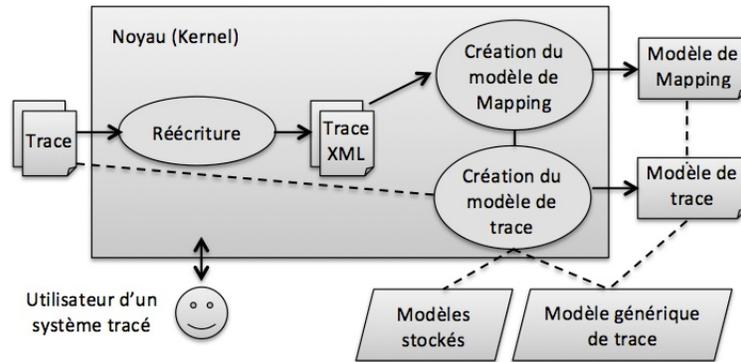


FIGURE 1: Processus de génération des collecteurs

- Structurer la trace dans un format XML s'il s'agit de texte : les lignes du texte deviennent des balises `<ligne>` XML dont les fils sont des balises `<cellule>` contenant chacune un des items de la ligne texte en question.
- Restructurer la trace dans un format XML s'il s'agit d'une source de BDD (table ou requête) : les lignes de la source de BDD deviennent des balises `<ligne>` XML dont les attributs et leurs valeurs sont les noms des champs avec les valeurs de la ligne de source BDD en question.

**Création du modèle kTBS de trace** D'après la définition donnée ci-dessus, un modèle de trace dans kTBS est une description de trace exprimée dans un langage fondé sur RDF, définissant des types d'obsels, des types d'attributs d'obsels et des types de relations pouvant être existants entre les obsels. Pour faciliter la tâche de rédiger du code RDF décrivant ces éléments du modèle par l'utilisateur, nous proposons un moyen graphique pour le faire. Dans cette même perspective d'aider l'utilisateur durant le processus de création du modèle, un exemple de trace externe peut être visualisé. La réutilisation de modèles kTBS existants ainsi que leur modification pour en créer d'autres est un moyen proposé pour construire facilement des modèles kTBS de traces.

**Mapping** Dès que la trace chargée par l'utilisateur ou sa copie réécrite est visualisée, et dès que ce dernier commence à construire son modèle de trace correspondant (éventuellement par le guidage du contenu de la trace) en définissant ses éléments, il peut parallèlement commencer à dresser des liens sémantiques entre les éléments de sa trace et des éléments de son modèle en construction. La sémantique de ces liens est du genre "est de type". Si on suppose qu'un lien `<eltXML - eltModel>` est dressé, ceci s'interprète par `eltXML est de type eltModel`.

Pour mieux justifier l'utilité de tels liens, il sera utile de se rappeler de but recherché, soit un ensemble de règles de conversion des traces externes en traces

importables au kTBS. Ainsi, ces règles doivent être capables de repérer des instances des éléments du modèle dans la trace externe et aussi de les convertir en instances dans la trace résultat. Les liens que l'utilisateur est invité à faire ont donc pour objectif la définition des éléments pertinents dans la trace externe. Il reste au générateur de collecteurs à généraliser ces définitions pour prendre en compte les variations éventuelles entre les exemples de traces externes.

À ce propos, plusieurs travaux ont été faits dans le contexte de l'annotation dans le web, dont l'objectif était de générer des repères (expressions XPath) robustes dans des pages web correspondant aux clics de l'utilisateur [2][11][12][14]. Le mot robuste signifie ici la souplesse qui contourne les changements fréquents dans les pages web. Effectivement, XPath [17] - le premier outil XML de pointage - n'a pas seulement prouvé ses capacités de repérage mais aussi il en donne des choix multiples.

Dans notre contexte le problème devient : comment générer des expressions XPath correspondant au choix de l'utilisateur (éléments XML sélectionnés) qui restent valides pour les futurs exemples de traces XML ? Une expression `<Exp>` créée à partir d'un élément sélectionné `<Elt>` qui correspond au type `<Type>` dans le modèle de trace est valide si et seulement si elle repère tous les éléments de type `<Type>`. Les liens que l'utilisateur doit dresser et que nous avons représenté par des couples `<eltXML - eltModel>` auront la forme `<ExpressionXPath - Type>` où Type est un type d'obsel ou d'attribut dans le modèle.

Dans les travaux de M. Abe [1][2][7][8][9], plusieurs modèles d'expressions XPath sont discutés :

- Les expressions "absolues" qui font le repérage en suivant la hiérarchie du document XML de la racine jusqu'au nœud cible, avec la forme : */Racine/fils/ ... /eltSélectionné*.
- Les expressions "relatives" qui repèrent un nœud par rapport à des positions d'ancres stables, avec la forme : *ExpressionAncre//eltSélectionné [Distance]*.
- Les expressions "conditionnelles" repérant les nœuds par le biais d'une condition à satisfaire, avec la forme : *//eltSélectionné [Condition]*.

Quant à nous, nous définissons la notion de "contexte" qui est à la base de nos modèles d'expressions XPath. Un contexte est une expression XPath repérant un endroit voisin d'un élément instance (obsel ou attribut) du modèle de trace ou d'un autre contexte. Pour donner plus de robustesse à nos expressions, nous avons aussi proposé la combinaison d'expressions relatives et conditionnelles. Globalement, nos expressions ont la forme suivante : *ExpressionContexte //eltSélectionné [Condition]*. Nous avons utilisé ce modèle d'expression pour exprimer des repères, d'instances de types d'obsel, d'instances de types d'attributs et aussi pour définir des contextes.

Concrètement, après que l'utilisateur ait défini son Mapping entre un élément XML et un élément du modèle, notre système propose un nombre de suggestions XPath avec des conditions variées selon le contenu de la trace, pour désigner l'élément XML sélectionné. Ces conditions peuvent porter sur :

- La valeur de l'élément sélectionné, par exemple : `//N [text() = "Exercice"]` pour choisir tous les nœuds N dont le texte est égal à "Exercice".
- Les valeurs des attributs de l'élément sélectionné, par exemple : `//N [@action = "Envoyer Msg"]` pour choisir tous les nœuds N dont l'attribut "action" égale "Envoyer Msg".
- La valeur du texte ou des attributs des fils de l'élément sélectionné, par exemple : `//N [./time/@begin = "10 :00"]` pour choisir tous les nœuds N dont l'attribut "begin" du fils "time" est égal à "10 :00".
- Le type de valeur de l'élément sélectionné, par exemple : `//N [matches(@option, Numérique)]` pour choisir tous les nœuds N dont le type de l'attribut "option" est numérique.
- La position de l'élément sélectionné, par exemple : `//P/N [2]` pour choisir tous les nœuds N dont la position est égal à 2 par rapport au nœud parent P.

Les propositions XPath repérant un élément sélectionné par l'utilisateur sont générées par l'algorithme présenté ci-dessous. Son principe est de générer des expressions XPath prenant en compte le nom de l'élément sélectionné, sa valeur, le type de sa valeur, sa position, les valeurs de ses attributs et le contenu de ses fils.

---

**Algorithm 1** Génération des expressions XPath
 

---

```

ExpName ← ExpName(SelectedElt.name)
if SelectedElt.attributes ≠  $\phi$  then
    for all attributes do
        Condition.addValue(SelectedElt.attribute.value)
    end for
else
    Condition.addValue(SelectedElt.value)
end if
if SelectedElt.chilids ≠  $\phi$  then
    for all Childs do
        if Child.attributes ≠  $\phi$  then
            for all child attributes do
                Condition.addValue(SelectedElt.child.attribute.value)
            end for
        else
            Condition.addValue(SelectedElt.child.value)
        end if
    end for
end if
Condition.addType(SelectedElt.valueType)
Condition.addPosition(SelectedElt.position)
Propositions.add(ExpName)
for all Condition do
    Propositions.add(ExpName + Condition)
end for
    
```

---

Cet algorithme produit des expressions de la forme : *//eltSélectionné [Condition]*. L'utilisateur aura le choix d'utiliser un contexte pour repérer sa sélection ou pas. Si un contexte est utilisé, le repérage de l'élément sélectionné devient relatif à ce contexte et l'expression devient : *ExpressionContexte//eltSélectionné [Condition]*. À son tour, *ExpressionContexte* a la forme : *[ExpressionContexte'] //eltSélectionné*, etc. Dans certains cas de traces, la combinaison de plusieurs types de conditions peut avoir de la valeur afin de concevoir des expressions pertinentes. Selon le contenu des traces traitées, on peut imaginer des expressions XPath : (1) avec une seule condition (avec/sans contexte) ou (2) avec plusieurs conditions combinées (avec/sans contexte). Pour choisir une proposition ou un contexte donnés, on propose de se focaliser sur l'effet du choix de l'utilisateur. Autrement dit, le choix se fait en visualisant les éléments repérés par chaque expression choisie. Il n'est donc pas question de comprendre la sémantique des expressions XPath mais d'en chercher la plus pertinente d'après les éléments qu'elles repèrent. À travers un ou plusieurs exemples, l'utilisateur va pouvoir créer son modèle kTBS de trace, définir tous les Mapping nécessaires et donc choisir toutes les expressions efficaces qui vont servir à la conversion des traces externes en traces importables. Les couples *<ExpressionXPath - Type>* (le modèle de Mapping) ainsi produits et le modèle kTBS de trace forment une grosse partie du collecteur cherché.

### 3.2 Utilisation du collecteur

Les modèles de Mapping et de traces sont des unités statiques incapables seuls de convertir ou de collecter des traces. Le Module Kernel intégré dans notre modèle sert à compléter ces deux modèles pour former de véritables collecteurs. Le rôle du Kernel est d'assurer deux fonctionnalités : (1) la réécriture des traces externes en traces XML et (2) l'utilisation du modèle de Mapping pour créer des traces kTBS puis les associer à leur modèle pour qu'elles soient importables. Les collecteurs résultants sont ainsi définis par le triplet : *<ModelMapping - ModelTrace - Kernel>* (Figure2).

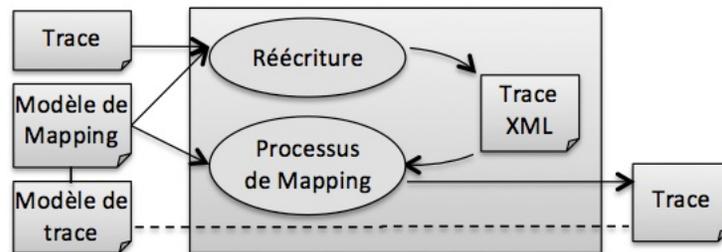


FIGURE 2: Utilisation du collecteur

À la présentation d'une nouvelle trace externe, le kernel identifie le type de la trace puis procède éventuellement à sa réécriture en exploitant les informations de réécriture (les séparateurs, le type de codage du texte, le chemin de la BDD, etc.) déjà définis lors de la création du collecteur. Le kernel commence ensuite à utiliser le modèle de Mapping et à chercher des occurrences de chaque type appartenant au modèle de Mapping. Il parcourt les types du modèle et en cherche toutes les occurrences présentes dans la trace. La recherche est simplement faite en lançant les expressions XPath correspondantes. Chaque occurrence trouvée est affectée à la valeur de l'obsel ou de l'attribut correspondant au type en question. La trace kTBS se construit ainsi progressivement en détectant tous les obsels et attributs existants dans la trace et en en déduisant les valeurs correspondantes. De cette façon le collecteur construit des traces kTBS conformes à leurs modèles et prêtes à les importer au kTBS.

## 4 L'outil xCollector

### 4.1 Présentation

xCollector est un outil java destiné aux utilisateurs voulant importer des traces texte, XML ou BDD, au kTBS. Via son interface graphique intuitive, xCollector permet de créer des modèles kTBS de trace et de générer des repères (expressions) XPath pour identifier, dans les traces externes, les éléments nécessaires pour créer de nouvelles traces importables conformes à leurs modèles. xCollector propose diverses fonctionnalités :

- Le chargement des traces externes : les traces de l'utilisateur sont chargées et visualisées sous forme d'arbre XML avec un habillage proche de l'original. xCollector affiche les traces texte sous forme de lignes de texte et les traces BDD sous forme de lignes de champs identiques. Lors du chargement, des données doivent éventuellement être renseignées : les séparateurs pour les traces texte, la requête/la table et la BDD pour les traces stockées dans des BDD.
- La création du modèle kTBS de trace : sans se soucier de coder en RDF ni de connaître la syntaxe kTBS pour définir des modèles de trace, l'utilisateur peut facilement créer son modèle de trace en créant des types d'obsels, d'attributs et de relations. xCollector affiche sous forme arborescente le modèle de trace et permet la sélection de ses éléments. A tout moment, il est possible de générer et d'accéder au modèle créé en format RDF-turtle.
- La création du modèle de Mapping : ce processus doit être initié par l'utilisateur en établissant des liens entre des éléments de la trace et des éléments du modèle. Un lien est défini si l'utilisateur sélectionne, d'un côté un élément XML de la trace et de l'autre côté un élément de l'arbre du modèle. Selon le type, le contenu et le voisinage de l'élément de trace sélectionné, le système propose un ensemble de références pour cet élément. Afin de connaître les éléments XML repérés par une référence donnée et pour faciliter ainsi le choix de celle-ci, xCollector utilise

une métaphore de drapeau levé sur les éléments correspondant à cette référence. L'utilisateur choisit alors la proposition si et seulement si un drapeau est levé sur tous les instances du type (d'obsel ou d'attribut) en question. En effet, chaque proposition est une expression XPath avec une condition différente. Le nombre et le type de ces conditions dépend alors du type, du contenu et du voisinage de l'élément sélectionné. Si, par exemple, le nœud sélectionné contient un attribut, une expression formée d'une condition sur la valeur de cet attribut est générée. Si cette valeur est numérique, une autre expression considérant ce type de valeur est générée. S'il y a plusieurs nœuds frères du nœud sélectionné ayant le même nom, une expression prenant en compte l'ordre de l'élément sélectionné est générée, etc. L'utilisateur a aussi le choix de repérer un élément sélectionné par rapport à un contexte précédemment défini. Une expression correspondant à un type d'obsel ou d'attribut donné peut être utilisée en tant que contexte. En dehors des types du modèle, l'utilisateur peut définir des contextes propres pour améliorer le repérage. Leur création suit les mêmes étapes : sélection, utilisation d'un contexte, choix de la proposition. Après sa génération, l'expression XPath, couplée avec le type sélectionné du modèle, est stockée dans un fichier XML représentant le modèle de Mapping.

## 4.2 Exemple d'utilisation

Le système GeoNotes [15], produit des traces textuelles composées de lignes contenant chacune une liste d'informations séparées par des virgules. Ces informations correspondent aux traitements faits sur des images géographiques et ce à des fins pédagogiques. Le nombre et le type des informations est variable dans chaque ligne. Après le chargement d'une trace GeoNotes, xCollector l'affiche dans son interface sous forme d'arbre XML mais avec un regard proche du format texte (Figure3 à gauche). Pour pouvoir faire des Mapping, au moins un élément doit être créé dans le modèle de trace. Supposons la création d'un type d'obsel appelé "zoom" : pour le faire il suffit d'introduire l'identificateur "zoom"; et éventuellement un super type (s'il y a héritage). Ce type d'obsel nécessite un attribut "chemin" pour stocker le chemin de l'image à zoomer. En sélectionnant l'obsel "zoom", l'insertion d'un attribut "chemin" se fait en introduisant son identifiant, label et type (Figure3 à droite). Il y a alors deux éléments (zoom et chemin) pour les quels on peut définir des Mapping. Afin de repérer le type "zoom", il suffit qu'on choisisse un des éléments XML correspondant à ce type, et le système génère les propositions d'expressions XPath. On choisit dans ce cas la proposition qui prend tous les éléments dont l'attribut @txt contient l'information "zoom" (drapeau jaune dans Figure4 (a)). L'utilisateur confirme l'enregistrement de la proposition (expression XPath) choisie en tant que règle de Mapping pour le type "zoom" (apparaissant dans Figure3 à droite). Pour utiliser le type "zoom" comme contexte pour le type d'attribut "chemin", il doit être chargé afin de former une expression à contexte pour le type "chemin". Après sélection d'un élément XML exemple pour le type "chemin", des propositions

sont encore générées, parmi lesquelles l'utilisateur choisit celle qui prend tous les nœuds fils du contexte "zoom" nommés Cell et se trouvant en quatrième position (drapeaux rouges dans Figure4 (b)). Ainsi, les drapeaux jaunes indiquent les contextes (obsels "zoom"), les drapeaux rouges indiquent les attributs "chemins". Les règles de Mapping et le modèle kTBS de trace sont visualisés dans la partie droite de l'interface.

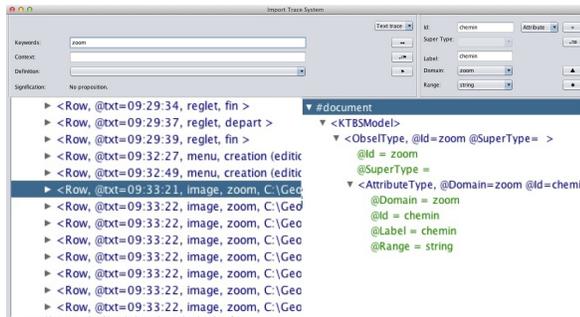


FIGURE 3: xCollector, visualisation et création de modèle

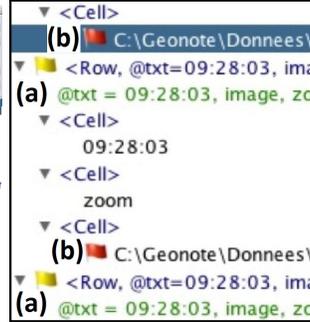


FIGURE 4: xCollector, Mapping

## 5 Conclusion

Les SGBT sont des plateformes susceptibles d'intéresser beaucoup d'acteurs intervenant sur des systèmes tracés, vu les fonctionnalités qu'ils proposent. Pour pouvoir profiter de ces fonctionnalités de traitement et d'analyse des traces, les utilisateurs doivent actuellement faire l'effort de programmer des collecteurs pour importer leurs traces dans un SGBT. xCollector, le fruit de notre travail, propose une manière interactive simple pour construire des collecteurs propres aux systèmes tracés sans aucun recours à la programmation. Nous avons constitué un corpus de traces variées sur lequel nous allons à présent expérimenter notre outil, afin d'en évaluer la généricité. Nous comptons aussi après cette étape évaluer l'utilisabilité de l'outil auprès d'un ensemble d'utilisateurs.

## Références

1. Abe, M., Hori, M. : A visual approach to authoring xpath expressions. Markup languages theory and practice 3(2), 191–212 (2001)
2. Abe, M., Hori, M. : Robust pointing by xpath language : Authoring support and empirical evaluation. In : Applications and the Internet, 2003. Proceedings. 2003 Symposium. pp. 156–165. IEEE (2003)

3. Bouhineau, D., Luengo, V., Mandran, N., Ortega, M., Wajeman, C., et al. : Open platform to model and capture experimental data in technology enhanced learning systems. In : Workshop Data Analysis and Interpretation for Learning Environments (2013)
4. Chaachoua, H., Croset, M.C., Bouhineau, D., Bittar, M., Nicaud, J.F., et al. : Description et exploitations des traces du logiciel d'algèbre aplusix. *Revue des Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation (STICEF)* 14 (2007)
5. Champin, P.A., Mille, A., Prié, Y. : Vers des traces numériques comme objets informatiques de premier niveau : une approche par les traces modélisées. *Intellectica* 1, 59 (2013)
6. Choquet, C., Iksal, S., et al. : Modélisation et construction de traces d'utilisation d'une activité d'apprentissage : une approche langage pour la réingénierie d'un eiah. *Revue des Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation (STICEF)* 14 (2007)
7. Hori, M., Abe, M., Ono, K. : Robustness of external annotation for web-page clipping : Empirical evaluation with evolving real-life web documents (2000)
8. Hori, M., Abe, M., Ono, K. : Extensible framework of authoring tools for web document annotation. In : Proceedings of International Workshop on Semantic Web Foundations and Application Technologies (SWFAT). pp. 1–8. Citeseer (2003)
9. Hori, M., Ono, K., Abe, M., Koyanagi, T. : Generating transformational annotation for web document adaptation : tool support and empirical evaluation. *Web Semantics : Science, Services and Agents on the World Wide Web* 2(1), 1–18 (2004)
10. Ji, M., Michel, C., Lavoué, E., George, S. : An architecture to combine activity traces and reporting traces to support self-regulation processes. In : Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on. pp. 87–91. IEEE (2013)
11. Kitano, T., Iguchi, K., Koyama, K. : Generating robust xpaths for service customization. In : 6th World Congress on Services (SERVICES-1). pp. 166–167. IEEE (2010)
12. Kowalkiewicz, M., Kaczmarek, T., Abramowicz, W. : Myportal : robust extraction and aggregation of web content. In : Proceedings of the 32nd international conference on Very large data bases. pp. 1219–1222. VLDB Endowment (2006)
13. May, M., George, S., Prévôt, P. : A closer look at tracking human and computer interactions in web-based communications. *Interactive Technology and Smart Education* 5(3), 170–188 (2008)
14. Paz, I., Díaz, O. : Providing resilient xpaths for external adaptation engines. In : Proceedings of the 21st ACM conference on Hypertext and hypermedia. pp. 67–76. ACM (2010)
15. SANCHEZ, O.L.E. : Geonote : un environnement informatique d'aide au travail sur le terrain pour l'enseignement des sciences de la terre
16. W3C : Extensible markup language (xml). 10. <http://www.w3.org/XML/>
17. W3C : Xml path language (xpath) version 1.0. <http://www.w3.org/TR/xpath/>