Acquisition de connaissances pour importer des traces existantes dans un système de gestion de bases de traces

Mohamed Besnaci, Nathalie Guin, Pierre-Antoine Champin

Acquisition de connaissances pour importer des traces existantes dans un système de gestion de bases de traces

Mohamed Besnaci[1], Nathalie Guin[2] & Pierre-Antoine Champin[2]

[1] Université BADJI MOKHTAR - ANNABA
besnacimohamed@yahoo.fr

[2] Université de Lyon, CNRS Université Lyon1, LIRIS, UMR5205, F-69622, France
nathalie.guin@liris.univ-lyon1.fr, pierre-antoine.champin@liris.cnrs.fr

Résumé : Les systèmes de gestion de bases de traces (SGBT) proposent des fonctionnalités de transformation et de requêtage sur les traces, pouvant intéresser de nombreux utilisateurs des systèmes tracés. Notre objectif est d'assurer l'importation de traces variées à kTBS, un SGBT développé au laboratoire LIRIS. Pour pallier le problème de l'hétérogénéité des traces, nous proposons de définir un collecteur pour chaque système tracé. Pour définir un tel collecteur, un utilisateur ayant une bonne connaissance du système tracé est invité à définir son modèle de trace dans kTBS ainsi que les correspondances entre les éléments de ce modèle et des éléments des traces à importer. Le système généralise les mises en correspondances ainsi explicitées, en interaction avec l'utilisateur, pour créer des règles de Mapping. Après cette phase, la collecte va consister à générer des traces modélisées à partir des traces existantes et du Mapping ainsi défini.

Mots-clés : Traces, Acquisition interactive de connaissances, Système de Gestion de Bases de Traces, Collecte, Mapping, xPath

1 Introduction

Les traces numériques sont des inscriptions temporalisées liées à un système informatique donné et ce pour diverses exploitations. Bien que la naissance des traces 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 au problème de 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 BDDR (Bases De Données Relationnelles).
— 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 à des endroits 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 BDDR ou encore constituée d'un document texte et d'un morceau multimédia.
— Abstraitement, une trace doit englober l'inscription du 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 cogniticiens 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 des fonctionnalités de visualisation et d'analyse d'un tel système ?

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 (May *et al.*, 2008) – ou une combinaison de types de trace dans certains cas comme celui de Ji *et al.* (2013). À la différence de ces situations où la collecte et l'exploitation des traces sont toutes deux assurées par le même système, il existe d'autres situations où ces deux traitements sont séparés. Dans un travail comme celui de Chaachoua *et al.* (2007) ou Sanchez (2006), 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 (Bouhineau *et al.*, 2013; Champin *et al.*, 2013). Notre travail s'intègre dans ce dernier processus et essaye de résoudre le problème de l'importation de traces variées dans l'une des plateformes de gestion des traces appelée kTBS (kernel for Trace Based Systems). Ainsi, cette plateforme est le système de gestion de traces dans lequel nous souhaitons importer des traces venant de systèmes divers. Par ailleurs, nous souhaitons offrir cette fonctionnalité à n'importe quel utilisateur du système tracé qui souhaiterait en exploiter les traces, y compris si il/elle n'a pas de compétences en programmation. En ce sens, nous nous démarquons de travaux similaires (Choquet *et al.*, 2007) ayant une approche plus technique. Ainsi, nous présenterons dans cet article notre processus d'importation de traces variées dans kTBS et l'outil xCollector qui le concrétise. Ce processus a l'avantage de traiter des formats variés de traces (texte, XML et BDDR). Son principe est basé sur l'acquisition de l'expérience de l'utilisateur en matière de collecte de données éparpillées dans les traces existantes. xCollector n'est pas lié à un système tracé spécifique, nous pouvons le considérer comme étant un générateur de collecteurs pour kTBS.

Dans la section suivante, 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. En section 4 nous présentons xCollector, l'outil matérialisant ce processus, nous décrivons ses fonctionnalités es-

sentielles et nous présentons un cas réel de son utilisation. La section 5 sera la conclusion de cet article.

2 Concepts autour des traces

Dans cette section, nous présentons les définitions proposées par le laboratoire LIRIS sur le concept de trace, ainsi que les concepts afférents.

Selon notre approche (Champin *et al.*, 2013), 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 est un système informatique mettant en pratique la notion de SGBT, développé au sein du laboratoire LIRIS. kTBS propose plusieurs fonctionnalités de gestion des traces, à savoir la création, l'interrogation, la transformation et la visualisation. kTBS utilise le formalisme RDF (Schreiber & Raimond, 2014) pour décrire les traces et les modèles, et expose ses données dans différents formats (XML, JSON, Turtle).

3 Importation des traces

A l'instar du LIRIS, l'exploitation des traces intéresse de plus en plus d'utilisateurs dans des disciplines variées. En effet, les traces sont largement considérées comme des sources de connaissances et de raisonnement. Ainsi, dans une perspective de capitaliser les traces, d'en faire des déductions et des interprétations, les SGBT sont proposés. Si les potentialités des SGBT sont suffisantes pour motiver leur utilisation, leur accessibilité et leur utilisabilité pourraient rebuter certains utilisateurs. Ce travail est donc une manière de faciliter la tâche aux utilisateurs pour importer leurs traces dans le SGBT kTBS. Comme mentionné dans l'introduction, nous entendons par utilisateur toute personne ne voulant et/ou ne pouvant pas programmer directement son collecteur. Il n'est donc pas demandé à cet utilisateur d'avoir des compétences en programmation pour profiter de ce que nous proposons. Il doit cependant avoir certaines connaissances de base notamment sur : la structure XML, les caractères séparateurs dans les fichiers texte et la manière d'accéder aux BDDRs, si besoin est.

Pour chaque système tracé, le processus d'importation de traces que nous proposons est composé de deux étapes : une étape de conception et de création (génération) de collecteur et une étape d'utilisation de ce dernier (l'importation en elle-même). Trois types de traces sont gérés par ce processus : texte, XML et BDDR.

3.1 Génération du collecteur

Le rôle essentiel des collecteurs que le générateur doit produire est de construire des traces importables à kTBS à partir des traces existantes. 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 d'exemples de traces externes ? 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. L'utilisateur intervient dans ce processus pour montrer, à travers des exemples de traces, où sont les données à transformer et en quoi on devrait les transformer. 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 ce processus. Le Mapping sert ainsi à définir les correspondances entre tous les éléments du modèle kTBS de trace et leurs 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 (Figure 1).

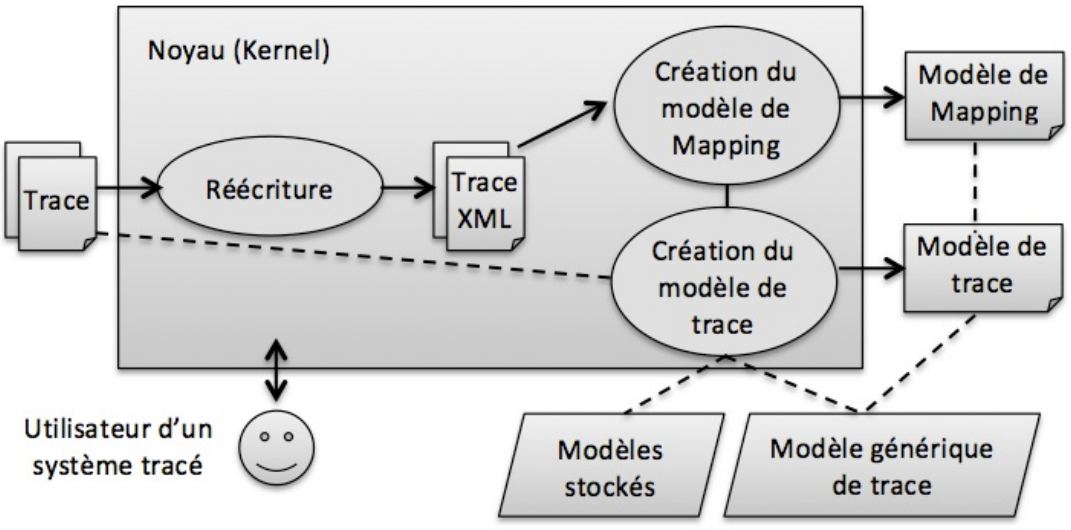

FIGURE 1 – Processus de génération des collecteurs

3.1.1 Réécriture

Compte-tenu des constatations effectuées en introduction, trois formats de traces sont essentiellement distingués. Pour simplifier le processus d'importation des traces à kTBS et éviter un traitement spécialisé pour chaque format, nous avons choisi de considérer XML comme format pivot. Autrement dit, les traces dans d'autres formats seront réécrites en XML, ce qui va ainsi unifier le processus d'importation. La réécriture est un processus automatique dépendant du format de la trace. Elle consiste à transformer les traces textuelles en XML en respectant des contraintes de séparation du texte définies par l'utilisateur. Pour les traces en BDDR, elle consiste à les transformer en XML en prenant en compte leur structure dans la BDDR. XML (W3C, 2008) est un standard largement utilisé pour sérialiser des contenus structurés (processus de transformation des objets en flux de données). 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 structure arborescente d'XML rend (à notre avis) son contenu facilement accessible et même plus lisible dans certains cas de traces.

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

— Structurer la trace dans un format XML s'il s'agit d'un 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 BDDR (table ou requête) : les lignes de la source de BDDR deviennent des balises <ligne> XML dont les attributs et leurs valeurs sont les noms des colonnes avec les valeurs de la ligne de la source BDDR en question.

3.1.2 Création du modèle kTBS de trace

D'après la définition donnée dans la section 2, 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 exister entre les obsels. Pour faciliter la rédaction 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.

3.1.3 Mapping

Dès que la trace chargée par l'utilisateur ou sa copie réécrite est visualisée, et dès que l'utilisateur commence à construire son modèle de trace (éventuellement guidé par le 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 de la forme "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 est utile de rappeler le but recherché, i.e. 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 menés dans le contexte de l'annotation dans le web, dont l'objectif était de générer des repères (expressions xPath) dans des pages web correspondant aux clics de l'utilisateur (Abe & Hori, 2003; Kitano *et al.*, 2010; Kowalkiewicz *et al.*, 2006; Paz & Díaz, 2010), qui soient robustes aux changements les plus fréquents dans la structure de ces pages web. Effectivement, xPath (W3C, 2010) - 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 représentés 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 Abe & Hori (2001, 2003) et Hori *et al.* (2003b,a, 2004), plusieurs modèles d'expressions xPath sont discutés :

— Les expressions "absolues" qui effectuent 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 introduisons 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 égale 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

Input : $SelectedElt$, trace element selected by user
Output : $Expressions$, the generated xPath expressions
$ExpName \Leftarrow \phi$
$Conditions \Leftarrow \phi$
$ExpName.Create(SelectedElt.name)$
if ($SelectedElt.attributes \neq \phi$) **then**
  for (each attribute in $SelectedElt.Attributes$) **do**
    $Conditions.addValue(attribute.value)$
  end for
else
  $Conditions.addValue(SelectedElt.value)$
end if
for (each child in $SelectedElt.Childs$) **do**
  if ($child.Attributes \neq \phi$) **then**
    for (each attribute in $child.Attributes$) **do**
      $Conditions.addValue(attribute.value)$
    end for
  else
    $Conditions.addValue(child.value)$
  end if
end for
$Conditions.addType(SelectedElt.valueType)$
$Conditions.addPosition(SelectedElt.position)$
$Expressions.add(ExpName)$
for (each condition in $Conditions$) **do**
  $Expressions.add(ExpName + condition)$
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).

En effet, les expressions xPath sont proposées à l'utilisateur après une transcription en langage naturel. Pour choisir une proposition ou un contexte donné, on propose aussi à l'utilisateur de se fonder sur l'effet de son choix d'expression. Autrement dit, quand l'utilisateur choisit une (ou plusieurs) propositions, le système lui indique les éléments repérés par cette (ou ces) proposition(s), ce qui permet à l'utilisateur de juger de la pertinence de son choix. Ce choix est donc fondé d'une part sur la formulation en langue naturelle des expressions xPath et d'autre part sur l'effet de ces expressions pour repérer les éléments de la trace que l'utilisateur souhaite récupérer.

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 partie importante du collecteur cherché.

3.2 Utilisation du collecteur

Les modèles de Mapping et de traces sont des unités statiques incapables à elles seules 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> (Figure 2).

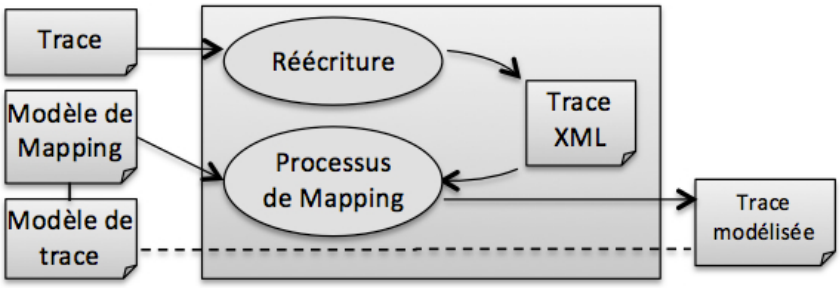

FIGURE 2 – Utilisation du collecteur

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 (le type de la trace, les séparateurs, le type de codage du texte, le chemin de la BDDR, 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 être importées au kTBS.

4 L'outil xCollector

4.1 Présentation

xCollector est un outil java destiné aux utilisateurs voulant importer à kTBS des traces texte, XML ou BDDR. Via son interface graphique, 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 BDDR sous forme de lignes de champs identiques. Lors du chargement, des données doivent éventuellement être renseignées : le format de la trace, les séparateurs pour les traces texte, la requête/la table et la BDDR pour les traces stockées dans des BDDR, etc. En effet, ces données serviront à choisir et à lancer le processus automatique de réécriture lors de l'utilisation des collecteurs.

— 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 au 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 coté 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 connaitre 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 toutes 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 pre-

nant 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 (Sanchez, 2006) produit des traces textuelles composées de lignes contenant chacune une liste d'informations séparées par des virgules. Ces informations correspondent aux traitement effectués 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 une apparence proche du format texte (cf. Figure 3). 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 (Figure 3). Il y a alors deux éléments (zoom et chemin) pour lesquels on peut définir des Mapping. Afin de repérer le type "zoom", il suffit qu'on choisisse un des éléments XML de la trace 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 Figure 4). L'utilisateur confirme l'enregistrement de la proposition (expression xPath) choisie en tant que règle de Mapping pour le type "zoom" (apparaissant dans Figure 3). Pour utiliser le type "zoom" comme contexte pour le type d'attribut "chemin", il doit être chargé afin de former une expression de 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 Figure 4). 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.

5 Conclusion

Les SGBT sont des plateformes susceptibles d'intéresser beaucoup d'acteurs intervenant sur des systèmes tracés, étant données 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 de construire des collecteurs propres aux systèmes tracés, sans recours à la programmation. xCollector a été conçu et réalisé suite à une étude d'un corpus de traces de systèmes variés. Nous avons ensuite exploité ce

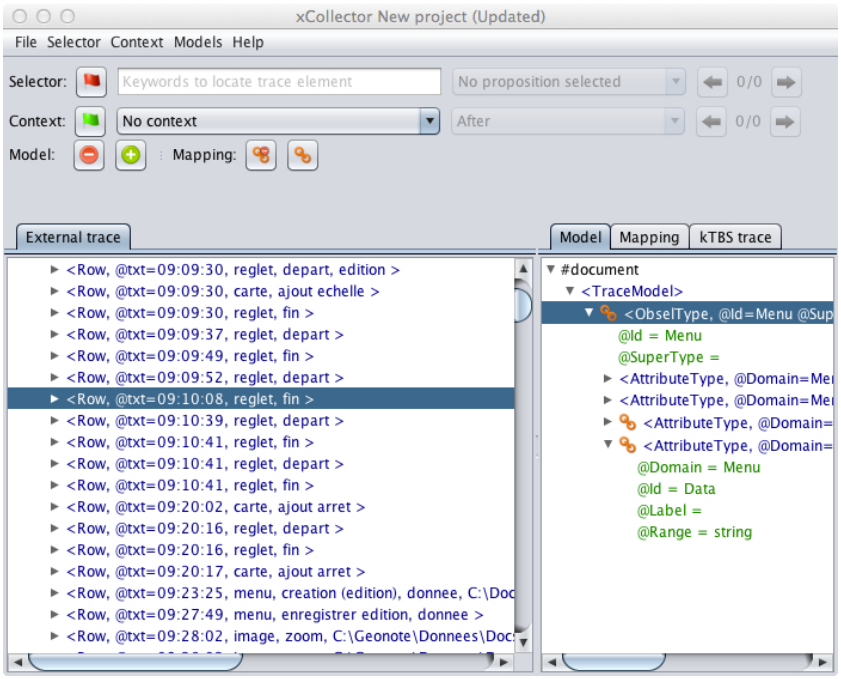

FIGURE 3 – xCollector, visualisation et création de modèle

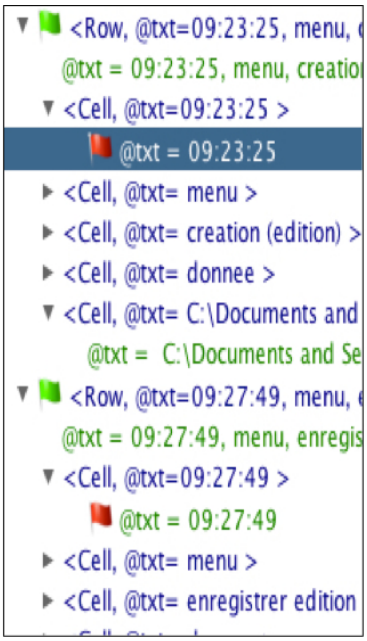

FIGURE 4 – xCollector, Mapping

corpus pour effectuer des tests montrant la validité et la généricité de l'approche choisie. Nous avons évalué la généricité de xCollector en l'utilisant pour collecter les traces de systèmes qui ne figuraient pas dans notre corpus initial, et nous avons également évalué son utilisabilité auprès d'un ensemble d'utilisateurs souhaitant importer leurs traces dans kTBS. Ces tests ont été conduits auprès de six utilisateurs ne pouvant et/ou ne voulant pas programmer des collecteurs pour leurs logiciels. Nous avons testé l'importation des traces de quatre logiciels pour lesquels les traces étaient au format texte, deux au format XML et un au format BDDR. Les utilisateurs ont pu essayer les fonctionnalités essentielles de xCollector, comme le repérage des données dans les traces chargées et leur Mapping avec les éléments du modèle de traces. xCollector a montré la validité de l'approche que nous avons choisie pour l'importation de traces, et a répondu aux principaux besoins des utilisateurs. Cependant d'autres améliorations notamment en ergonomie sont à faire, pour faciliter encore la prise en main de xCollector par les utilisateurs visés. D'autres fonctionalités restent aussi à développer pour augmenter le nombre de logiciels pour lesquels il pourra être utilisé.

Références

ABE M. & HORI M. (2001). A visual approach to authoring xpath expressions. *Markup languages theory and practice*, **3**(2), 191–212.

ABE M. & HORI M. (2003). Robust pointing by xpath language : Authoring support and empirical evaluation. In *Applications and the Internet, 2003. Proceedings. 2003 Symposium*, p. 156–165 : IEEE.

BOUHINEAU D., LUENGO V., MANDRAN N., ORTEGA M., WAJEMAN C. *et al.* (2013). Open platform to model and capture experimental data in technology enhanced learning systems. In *Workshop Data Analysis and Interpretation for Learning Environments*.

CHAACHOUA H., CROSET M.-C., BOUHINEAU D., BITTAR M., NICAUD J.-F. *et al.* (2007). Description et exploitations des traces du logiciel d'algèbre aplusix. *Revue STICEF*, **14**.

CHAMPIN P.-A., MILLE A. & PRIÉ Y. (2013). Vers des traces numériques comme objets informatiques de premier niveau : une approche par les traces modélisées. *Intellectica*, **1**, 59.

CHOQUET C., IKSAL S. *et al.* (2007). Modélisation et construction de traces d'utilisation d'une activité d'apprentissage : une approche langage pour la réingénierie d'un eiah. *Revue STICEF*, **14**.

HORI M., ABE M. & ONO K. (2003a). Extensible framework of authoring tools for web document annotation. In *Proceedings of International Workshop on Semantic Web Foundations and Application Technologies (SWFAT)*, p. 1–8 : Citeseer.

HORI M., ABE M. & ONO K. (2003b). Robustness of external annotation for web-page clipping : Empirical evaluation with evolving real-life web documents. In *Semannot 2003 : Knowledge Markup and Semantic Annotation*, Sanibel, Florida, USA.

HORI M., ONO K., ABE M. & KOYANAGI T. (2004). 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.

JI M., MICHEL C., LAVOUÉ E. & GEORGE S. (2013). 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*, p. 87–91 : IEEE.

KITANO T., IGUCHI K. & KOYAMA K. (2010). Generating robust xpaths for service customization. In *6th World Congress on Services (SERVICES-1)*, p. 166–167 : IEEE.

KOWALKIEWICZ M., KACZMAREK T. & ABRAMOWICZ W. (2006). Myportal : robust extraction and aggregation of web content. In *Proceedings of the 32nd international conference on Very large data bases*, p. 1219–1222 : VLDB Endowment.

MAY M., GEORGE S. & PRÉVÔT P. (2008). A closer look at tracking human and computer interactions in web-based communications. *Interactive Technology and Smart Education*, **5**(3), 170–188.

PAZ I. & DÍAZ O. (2010). Providing resilient xpaths for external adaptation engines. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, p. 67–76 : ACM.

SANCHEZ, OLIVIER LEFEVRE E. (2006). Geonote : un environnement informatique d'aide au travail sur le terrain pour l'enseignement des sciences de la terre. In *Biennale de l'éducation*, Lyon.

SCHREIBER G. & RAIMOND Y. (2014). Rdf 1.1 primer, w3c working group note 24 june 2014. http ://www.w3.org/TR/rdf11-primer/.

W3C (2008). Extensible markup language (xml). `http://www.w3.org/TR/xml/`.

W3C (2010). Xml path language (xpath) version 2.0. `http://www.w3.org/TR/xpath20/`.