École doctorale Informatique et Mathématiques de Lyon

Année 2011
N° d'ordre 19-2011

THÈSE DE L'UNIVERSITÉ DE LYON

délivrée par

L'Université Claude Bernard Lyon 1

présentée et soutenue publiquement le 14 Janvier 2011

pour l'obtention du

DIPLÔME DE DOCTORAT

(arrêté du 7 août 2006)

spécialité informatique

par

Lotfi Sofiane SETTOUTI

Systèmes à Base de Traces Modélisées : Modèles et Langages pour l'exploitation des traces d'Interactions

Composition du jury

Rapporteurs :	Serge Garlatti	Professeur Télécom Bretagne
	Jean-Paul Sansonnet	Directeur de recherche CNRS LIMSI
Examinateurs :	Pierre Deransart	Directeur de recherche INRIA Rocquencourt
	Jean-Marc Labat	Professeur Université Pierre et Marie Curie
Directeurs :	Jean-Charles Marty	Maître de conférence - HDR Université de Savoie
	Yannick Prié	Maître de conférence Université Claude Bernard Lyon 1
	Alain Mille	Professeur Université Claude Bernard Lyon 1

Laboratoire d'InfoRmatique en Images et Systèmes d'information — UMR 5205 CNRS

Mis en page avec la classe thloria.

Table des matières

Table des figures

Introduction Générale

Ce chapitre est une introduction générale à cette thèse. Il décrit brièvement le contexte et les domaines dans lesquelles s'inscrit ce travail. Il résume aussi les questions de recherche traitées et les contributions mises en œuvre.

1 Introduction

La problématique du traçage et de l'exploitation des traces inscrites lors de l'observation d'activités médiées par un environnement informatique est l'objet de nombreuses recherches. Dans la majorité des travaux, la trace est utilisée comme un outil réifiant l'observation passée ou en cours des interactions homme-machine et exploitée soit de manière *ad-hoc* au moment même de la production de ces traces ou *post-hoc* après l'activité observée.

Les objectifs visés par les exploitations *post-hoc* sont multiples et essayent tous d'une certaine manière de répondre à des « questions » sur la base des observations passées inscrites dans les traces, généralement pour comprendre le comportement d'un utilisateur (ou groupe d'utilisateurs) et construire des outils plus conformes à ses attentes. À coté de ces pratiques, les exploitations *ad-hoc* sont menées en quasi-temps réel en cours de la production des traces et s'intéressent globalement à la manière d'extraire des informations ou d'inférer des connaissances à des fins d'assistance à l'utilisateur ou de personnalisation de son environnement, au moment même de l'observation de son activité. De manière générale, même si tracer offre de bonnes perspectives, cette manière d'observer les activités est loin d'être facile à mettre en œuvre avec les outils existants notamment pour prendre en compte les usages *ad-hoc* et *post-hoc* des traces obtenues. Le travail de thèse que nous allons présenter dans ce mémoire s'inscrit dans cette optique et concerne la proposition d'un cadre théorique et formel facilitant la manipulation des traces pendant et après leur production.

2 Contexte de notre travail et domaines concernés

Ce travail de recherche se situe dans le contexte général de la « personnalisation des Environnements Informatiques pour l'Apprentissage Humain (EIAH) [1] » mais constitue une contribution

1. « Un EIAH est un environnement informatique conçu dans le but de favoriser l'apprentissage humain »[Tch02]

qui relève de « l'Ingénierie des Connaissances (IC) » dynamique.

2.1 Traces et Environnements Informatiques pour l'Apprentissage Humain (EIAH)

Le travail de thèse que nous allons présenter s'inscrit dans le cadre du cluster ISLE [2] et plus précisément du projet « personnalisation des EIAH [3] » finançant notre recherche. Ce projet a pour objectif final d'étudier et de proposer les cadres théoriques, méthodologiques et informatiques permettant la personnalisation des EIAH. L'originalité de ce projet est de fonder cette personnalisation sur la notion de *trace interactions* pour permettre à la fois de répondre à des besoins spécifiques et précis comme par exemple de comprendre les comportements d'un apprenant qui utilise un EIAH et fournir à l'enseignant des informations pertinentes sur ces comportements. Il s'agit également pour ce projet de répondre à des besoins généraux et récurrents en fournissant des outils génériques et réutilisables pouvant être exploités dans différents contextes et applications pour soutenir différentes techniques de personnalisation, ce qui relève de l'IC.

2.2 Traces et l'Ingénierie des Connaissances

Là où le domaine des EIAH en général et le projet personnalisation des EIAH en particulier fournissent d'une certaine manière un cahier des charges des propriétés, fonctions et services nécessaires pour l'exploitation des traces, l'ingénierie des connaissances permet de fournir le cadre pour construire les outils génériques répondant à ce cahier des charges. Il s'agit dans ce domaine d'étudier les apports et les limites des langages de représentation de connaissances existants pour prendre en compte les exploitations des traces telles qu'elles ont été identifiées et mises en œuvre dans le projet et la communauté EIAH. Nous nous intéressons notamment à une nouvelle classe de systèmes à base de connaissances, considérant comme source première de connaissances des traces d'interaction explicitement modélisées, baptisée *système à base de traces modélisées*.

3 Questions de recherche

La question de la conceptualisation des traces à l'interface des EIAH, de leurs représentations et de leurs traitements, a élargi le champ des travaux de recherche autour de cette notion et a conduit dans certaines situations à de nouvelles problématiques. Dans le cadre des recherches en EIAH, elle a été considérée sous différents angles : support à la modélisation de l'apprenant, conteneur d'informations ou de connaissances pour le concepteur ou objet de réflexion pour l'enseignant ou l'apprenant, la trace pose explicitement plusieurs défis allant de sa représentation de manière normalisée et intelligible à son exploitation au moment même de l'apprentissage. Dans le cadre de l'IC, la trace, considérée comme un ensemble de connaissances pouvant évoluer

2. Informatique, Signal et Logiciel Embarqué, http ://cluster-isle.grenoble-inp.fr/
3. http ://cluster-isle-eiah.liris.cnrs.fr/

dynamiquement dans le temps et disposant de modèles explicites de représentation, soulève des questions de son exploitation en mode continu et incrémental dés la phase de collecte. Notre travail s'efforce de proposer des contributions à ces besoins essentiels qui ne cessent de s'accentuer, que ce soit dans le cadre des EIAH ou en IC en permettant :

1. de traiter des traces complexes et hétérogènes. En effet, les traces comporte des éléments aux formes de plus en plus variées (texte structuré, base de données, etc.), incluant des relations riches (composition, liens hypertexte, dépendances temporelles, etc.) et offrant des possibilités de navigation poussées. Cette évolution met clairement en évidence l'importance d'un cadre général pour construire des outils génériques interopérables permettant de traiter, de représenter et de modéliser des traces issues de différents systèmes et exploitées pour de multiples objectifs.

2. fournir des langages pour l'exploitation de traces. En effet, l'interrogation et l'interprétation de traces explicitement modélisées nécessite de spécifier et de formaliser les langages capables d'exprimer des requêtes sur les traces, y compris dans le cadre d'une évaluation incrémentale et continue.

4 Contributions

La complexification des technologies et des architectures supportant les EIAH et la multiplication des pratiques et des usages à base de traces a conforté notre objectif de création d'un cadre théorique générique pour leurs exploitations. Nous avons commencé par définir la notion de *système à base de traces* : une classe de systèmes à base de connaissances facilitant l'exploitation et le raisonnement sur des traces modélisées. Le cadre théorique proposé pour mettre en place de tels systèmes permet d'adresser les besoins énoncés dans la section précédente et décrit une double contribution :

- La première contribution concerne la définition de l'architecture et des processus mobilisés par un système à base de traces. Ceci permet de disposer d'un cadre conceptuel fécond permettant d'une part de construire de nouveaux systèmes interopérables pour l'exploitation des traces et d'autre part de fournir un vocabulaire commun pour parler et étudier les systèmes et les pratiques existantes à base de traces.
- La seconde contribution concerne la définition formelle des concepts relatifs au cadre des systèmes à base de traces. Plus précisément, la proposition d'un langage pour l'interrogation et la transformation de traces modélisées, y compris dans cas du traitement continu de la trace. La sémantique est définie formellement en s'appuyant sur le formalisme des théories des modèles et la théorie du point fixe, habituellement utilisé pour décrire la sémantique formelle des langages de représentation de connaissances.

5 Plan du mémoire

Ce mémoire est composé de 4 parties, s'organisant elles-mêmes dans plusieurs chapitres.

- La partie I comporte deux chapitres. Le premier chapitre 1 détaille le contexte, la problématique et les objectifs de cette thèse. Le second chapitre 2 fait le tour des différentes notions de traces existantes et les propriétés quelles regroupent et discute les nombreux travaux sur les traces dans la communauté EIAH.
- La partie II regroupe les contributions de cette thèse en trois chapitres. Les chapitres 3 et 4 présentent respectivement le cadre conceptuel et formel des systèmes à base de traces modélisées. Le chapitre 5 fait le tour des technologies existantes pour la mise en œuvre de ces systèmes tout en résumant les avantages et inconvénients de leur usage.
- La partie III est composée de deux chapitres. Le premier (chapitre 6) décrit deux cas d'application illustrant l'usage des systèmes à base de traces dans le contexte des EIAH. Le second chapitre 7 décrit les implémentations faites de notre travail.
- La partie IV comporte le dernier chapitre 8 décrivant la conclusions et les perspectives de ce travail de thèse.

Première partie

Problèmatique et État de l'art

Chapitre 1

Problématiques et objectifs

Sommaire

Ce chapitre donne un aperçu des problématiques et objectifs traités dans ce travail de thèse. La section 1.1 introduit le chapitre en précisant les méthodes de personnalisation. La section 1.2 présente le contexte dans lequel s'inscrit cette thèse. La section 1.3 motive notre travail en décrivant les usages et les exploitations des traces dans le contexte des EIAH. La section 1.4 détaille les problématiques soulevées par nos travaux. Enfin, la section 1.5 résume nos contributions.

1.1 Introduction

La personnalisation des EIAH a fait naître beaucoup de questions de recherche dans différentes disciplines notamment dans le cadre des systèmes dits «ouverts» comme les plateformes d'apprentissage à distance [4] ou les systèmes d'apprentissage supportant des situations collaboratives [5]. À l'inverse des EIAH de type tuteurs intelligents [6] où les interactions sont modélisées en tant que telles au moment de la conception, les systèmes d'apprentissage ouverts sont moins limitatifs et laissent à l'utilisateur beaucoup de liberté dans ses interactions. Si on considère la large variété de situations pouvant être mises en œuvre avec ces EIAH (e.g. à distance, en collaboration, en présentiel mais médié par un outil informatique, etc.), leur appropriation et *personnalisation* devient alors problématique. L'émergence de nouveaux usages au moment même de l'activité et la complexité des interactions pouvant se produire nécessitent des services de *personnalisation* capables d'observer et de reconnaître les besoins des utilisateurs. Par *personnalisation*, nous désignons la capacité d'un EIAH à s'adapter par ses services, ses ressources ou ses interfaces aux besoins des utilisateurs.

Il existe de nombreuses approches pour la personnalisation : toutes nécessitent une « prise en compte » des besoins de l'utilisateur. La prise en compte de ces besoins peut être faite par des méthodes directes qui « demandent » directement à l'utilisateur son avis : il s'agit de systèmes de description de préférences, de configuration, d'édition de profil, etc. Ces méthodes ont l'avantage d'être explicites pour l'utilisateur, mais le désavantage d'exiger de celui-ci qu'il ait une bonne connaissance des effets de ses choix sur la personnalisation. Ces approches sont bien adaptées pour des effets immédiats (aspects d'interface par exemple), mais très mal adaptées pour des effets sur les processus interactionnels (le processus de réalisation d'une activité est spécialisé, certaines possibilités sont filtrées par exemple). Une autre approche, retenue comme base de personnalisation dans cette thèse, consiste à « observer » l'utilisateur dans son activité afin d'exploiter ces observations pour personnaliser l'environnement. L'observation nécessite l'inscription des interactions dans des traces informatiques, qui seront l'objet de visualisation directe, ou/et d'interprétations et calculs pour construire des indicateurs d'activité, qui eux-mêmes peuvent être produits à l'utilisateur. Les traces informatiques sont à la base de nombreuses approches de personnalisation, mais n'ont pas fait l'objet d'étude approfondie en tant qu'objets informatiques possédant des propriétés et des méthodes propres. Sortant les traces de leur rôle de matériel secondaire, ce travail de thèse est un premier pas vers l'étude des traces et des outils associés. Les sections suivantes présentent respectivement le contexte, les besoins des EIAH en termes d'observations à base de traces d'interactions et les problématiques engendrées.

4. Learning Management System (LMS).
5. Computer Supported Collaborative Learning (CSCL).
6. Intelligent Tutoring System (ITS)

1.2 Contexte : le projet « Personnalisation des Environnements Informatiques pour l'Apprentissage Humain (EIAH) »

Le travail de recherche présenté dans ce mémoire s'inscrit dans le cadre du projet de *personnalisation des environnements informatiques pour l'apprentissage humain [BM05] (EIAH)* financé par la Région Rhône Alpes dans le cadre du cluster ISLE [7] (Informatique, Signal, Logiciels embarqué). Le postulat de base considéré dans ce projet est que la résolution du problème de la personnalisation des EIAH est essentiellement dépendante de la capacité à produire des traces pertinentes et exploitables des activités des apprenants médiées par un EIAH [BM05]. Le problème de la personnalisation et de la conceptualisation des traces d'activité est fortement pluridisciplinaire, il implique des modèles issus de l'informatique (apprentissage automatique, IHM, modélisation des connaissances et du raisonnement), des sciences de l'apprentissage (psychologie, didactique, pédagogie et sciences de l'éducation), de l'ergonomie, des sciences du langage et de la communication, et des mathématiques (statistique, logique). Le projet s'attaque à cette problématique sur trois volets :

- *un volet théorique :* articulation des modèles computationnels et conceptuels de l'activité de l'apprenant ou d'un groupe d'apprenants (niveau de granularité de l'acquisition de données, représentation et interprétation des données, diagnostic et exploitation didactique ou pédagogique du diagnostic, etc.).
- *un volet méthodologique :* constitution de corpus de traces, standardisation et documentation des pratiques au sein d'une bibliothèque partagée, validation informatique, didactique, pédagogique et cognitive.
- *un volet technologique :* instrumentation de la recherche par la mise en place d'outils et de protocoles pour la gestion et la manipulation de corpus.

Les exploitations et objectifs dont il est question dans le projet sont organisés en cinq tâches :

Tâche T_1 : Production et Représentation de Traces. L'objectif de cette tâche est de définir un cadre théorique pour l'exploitation des traces d'apprentissage dans le contexte d'activités d'apprentissage médiées par un environnement informatique.

Tâche T_2 : Couplage scénario-traces pour la personnalisation des EIAH. L'objectif de cette tâche est d'étudier le scénario pédagogique et les méthodes permettant sa personnalisation à partir des traces de l'activité d'apprentissage.

Tâche T_3 : Interprétation des traces et représentation des connaissances. L'objectif de cette tâche est d'explorer l'acquisition et la représentation des traces et leur relation avec les spécificités des contenus d'apprentissage et les modalités et caractéristiques de l'activité d'un apprenant (ou d'un groupe d'apprenants).

Tâche T_4 : Interprétation des traces et régulation des interactions sociales et langagières. L'objectif de cette tâche est de mettre en œuvre les méthodes d'analyse de traces d'interactions médiatisées et des diagnostics réalisés en temps réel.

7. http ://http ://cluster-isle.grenoble-inp.fr/

Tâche T_5 : Capitalisation des Connaissances sur l'exploitation des traces et benchmarks. L'objectif de cette tâche est de mettre en place des corpus de traces à des fins de capitalisation des pratiques et de formaliser les procédures de validation (benchmarks).

Le travail présenté dans ce mémoire s'inscrit dans le cadre de la tâche T_1. L'objectif de la tâche T_1 est de définir et d'exploiter ce cadre théorique pour permettre la génération de traces selon des formats explicités par des modèles à différents niveaux d'abstraction pouvant donc plus facilement alimenter des benchmarks (tâche T_5) pour valider des capitalisation des différentes exploitations proposées dans les autres tâches (T_2 à T_4). Il vise l'élaboration des bases théoriques et informatiques pour la manipulation des traces et fait suite en partie au travaux de l'équipe Silex sur l'approche MUSETTE [CPM03, Cha03]. Dans cette approche, différents niveaux de traitements sont proposés : la collecte des informations en format natif (issus des *capteurs* instrumentant un environnement d'apprentissage) au travers de *flux de traçage*, la mise en forme des informations natives dans un langage autorisant des exploitations variées sous la forme de traces qualifiées de *brutes*, la définition de *modèles d'utilisation* permettant de fournir un premier niveau de traces *lisibles* qualifiées de *primitives*, et enfin la description de modèles d'interprétation aussi bien pour des tâches d'analyse par les enseignants que par les apprenants eux-mêmes (dans des temporalités différentes naturellement) par un mécanisme de *signature de tâche*. L'élaboration des modèles d'interprétation et d'abstraction (à partir du niveau des traces primitives) nécessite une ingénierie de la connaissance du processus d'apprentissage qui exploite aussi bien les méthodes issues de l'acquisition de la connaissance que des méthodes de découverte de ces connaissances dans les flux de plus bas niveau.

Le contexte du projet personnalisation des EIAH motive énormément ce travail de thèse. Comme nous allons le montrer dans la suite, installer les bases théoriques, méthodologique et technologiques pour contribuer à l'objectif commun du projet est un élément central de notre recherche notamment pour la création et la capitalisation de modèles et de langages pour soutenir la personnalisation.

1.3 Motivations

L'activité d'apprentissage virtualise des objets pérennes de médiation classiques (cahier, livres, paillasses de travaux pratiques, machine à calculer, instruments de mesure, crayon, papier, etc.) et crée de nouvelles concrétisations éphémères à l'interface de l'environnement, tandis que les informations autorisant ces concrétisations éphémères sont enfouis dans le système d'information. Cette virtualisation et re-concrétisation labiles provoquent bien souvent la perte de la perception de l'activité en tant que processus cohérent. Les problèmes liés à la perte de perception de l'activité ne sont pas nouveaux et ont été largement identifiés dans le monde des EIAH. Les exemples rendant compte du manque de perception dans les EIAH sont nombreux et réifient souvent de réels besoins : besoin d'avoir le contexte global pour un apprenant immergé dans le cadre de sa tâche spécifique (e.g., percevoir son avancement personnel relatif au collectif), besoin d'avoir une vue d'ensemble pour un enseignant voulant par exemple adapter et personnaliser

l'activité des apprenants selon leurs besoins (e.g., percevoir l'avancement du collectif). etc.

L'observation d'activité médiée par un environnement informatique nécessite l'inscription des interactions dans des traces informatiques par cet environnement. Les traces sont ainsi devenues le support incontournable à l'observation d'activités menées avec un EIAH, et sont à la base de nombreuses approches de personnalisation. Cependant, bien que l'utilisation des traces dans les EIAH soit une approche courante, l'élargissement de leur usage pour différents acteurs a engendré de nouveaux besoins. Certains de ces besoins sont fondamentaux, nécessitant des langages spécifiques pour mettre en œuvre des interprétations et des exploitations de traces puisque, dans beaucoup d'EIAH, différentes catégories d'acteurs peuvent être concernées par leur usage (e.g. administrateurs, enseignants, etc.).

Ces acteurs, comme nous allons le voir (cf. figure 1.1), ont des objectifs et des besoins d'observation spécifiques. Afin de bien cerner les besoins et les propriétés requises par les traces et les outils les exploitant, nous allons présenter ce que signifie l'observation pour chacun d'eux, i.e., pour l'enseignant, pour le concepteur de l'activité, pour le chercheur ou encore pour l'apprenant.

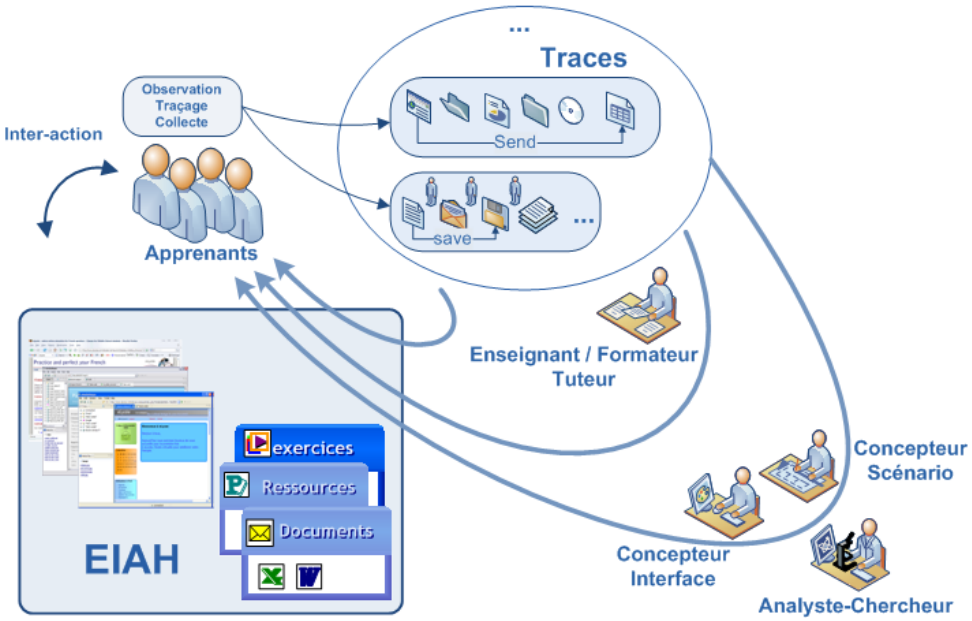

FIGURE 1.1 – Utilisation des traces dans les EIAH par différents acteurs

La Figure 1.1 montre l'utilisation d'une trace d'interaction dans les EIAH. Elle décrit la boucle générale de personnalisation des EIAH par différents acteurs en utilisant les traces. Les participants à l'activité d'apprentissage interagissent avec un EIAH, individuellement ou en groupes. Suivant leur rôle respectif, les participants n'auront pas les mêmes exploitations de traces.

Par exemple, un enseignant-tuteur peut guider l'activité individuelle ou collaborative en essayant de comprendre les dysfonctionnements éventuels par rapport au scénario qu'il avait préconisé. Il peut alors adapter la session, introduire des aides personnalisées, fournir des supports pédagogiques adaptés aux différents publics. Un enseignant-concepteur peut exploiter les

traces pour personnaliser un scénario pédagogique, permettant ainsi de réguler le déroulement d'une session d'apprentissage en tenant compte de certains aspects qui ne peuvent être mesurés qu'en cours de session, tel le temps de réponse à un exercice. L'apprenant peut visualiser sa trace et se faire une image de son évolution dans l'activité, ce qui lui permettra de comprendre son cheminement et une meilleure évolution dans ses tâches. Ces quelques exemples d'utilisation des traces par les différents acteurs donnent une vue très parcellaire de la puissance de l'exploitation des traces car les participants peuvent avoir des besoins très différents. Il serait donc difficile de décrire exhaustivement l'ensemble des fonctionnalités imaginables. La suite dresse un rapide tour d'horizon des attentes pouvant être satisfaites par l'exploitation des traces, et ceci du point de vue des différents acteurs de l'EIAH.

1.3.1 Traces comme support d'observation pour l'apprenant

L'observation à base de trace permet à l'apprenant d'analyser son propre parcours dans le processus de construction de connaissance et ainsi de prendre conscience de son activité (support à la méta-cognition [PW90, GB07]). D'un côté, cette approche réflexive permet de comprendre la manière dont la compétence a été acquise, ce qui permet à l'apprenant de repérer également ses lacunes et les éléments qui nécessitent le plus d'attention pour la suite. D'un autre côté, la visualisation de sa propre trace et parcours (avec ses performances) au sein de la classe permet à un apprenant de mieux se positionner par rapport aux autres et de connaître leurs compétences. Il est également possible d'inciter les apprenants à confronter leurs traces respectives, ce qui peut aussi être un moyen de favoriser des pratiques collaboratives. Enfin, l'usage de traces permet l'assistance de l'apprenant dans son parcours pédagogique (approche «Guiding») [JSM01] éventuellement par la construction et la comparaison du parcours des apprenants avec un modèle générique de parcours d'apprenants ayant réussis [Zai02]. La démarche permet ainsi une réutilisation de l'expérience d'apprentissage [Mil06b].

1.3.2 Traces comme support d'observation pour l'enseignant-tuteur

Les EIAH ont modifié les conditions auxquelles l'enseignant est habitué (présentiel classique). D'ailleurs, ces environnements s'efforcent de plus en plus de reproduire les conditions « classiques » notamment en permettant à l'enseignant de comprendre et de rester au courant du déroulement de l'activité pédagogique mise en place afin de pouvoir réagir de aux différentes situations qui peuvent se présenter. Certains travaux ont mis en évidence le manque de conscience de l'activité par l'enseignant par rapport à l'activité des apprenants [FH05, FHM$^+$06]. Ces travaux montrent que dans un EIAH, l'enseignant obtient généralement moins de retours sur le déroulement de la session que dans une situation d'apprentissage classique. D'une part, l'enseignant a des difficultés à percevoir l'activité de chaque apprenant ou des groupes d'apprenants. D'autre part, les éléments de perception de l'activité pouvant être utilisés par l'enseignant à partir des traces sont très variables et sont souvent très liés à l'objectif de l'activité pédagogique en cours d'observation.

Par exemple, l'enseignant peut chercher à récupérer des informations liées à la performance des apprenants (i.e. nombre d'exercices terminés, score obtenu dans un QCM) afin de détecter les difficultés rencontrées par les apprenants ou les dysfonctionnements éventuels par rapport à l'évolution prévue de la séance [SS05]. L'enseignant peut aussi faire attention aux éléments lui permettant d'évaluer le degré de collaboration au sein d'un groupe d'apprenants afin de voir si les apprenants collaborent pour s'approprier le savoir [RC03, Per03]. D'autre part, durant une séance pédagogique l'enseignant a une multitude de tâches à réaliser comme par exemple contrôler la bonne compréhension du sujet et la vitesse d'avancement des apprenants, répondre à leurs appels, adapter le scénario pédagogique proposé, ce qui l'empêche d'observer continuellement et parfaitement l'activité de chaque apprenant durant la séance. Labat [LPV06] met en évidence la nécessité de pouvoir suivre l'apprenant pendant la réalisation de son activité de manière synchrone afin de pouvoir lui fournir un soutien très réactif pendant le déroulement de l'activité. Du point de vue de l'enseignant-tuteur, cela implique une bonne perception de l'activité pédagogique mise en place.

Dans le contexte des situations d'apprentissage, le modèle SAAD (Suivi d'Activités d'Apprentissage à Distance) [Des01] décrit des fonctionnalités à intégrer dans un environnement de suivi pédagogique synchrone pour supporter l'activité du tuteur. Ce modèle met l'accent notamment sur la nécessité de fournir une perception de l'activité de l'apprenant qui regroupe des fonctionnalités lui permettant de consulter à partir des traces son parcours, le détail de ses actions, ses productions ainsi que les avertissements issus d'une analyse des traces de ses activités. Dans ce sens, le projet FORMID [GAP+04] ajoute principalement des fonctionnalités visant à faciliter la perception synthétique de l'activité de la classe (ou d'un sous-groupe d'apprenants) en complément de la perception détaillée de l'activité d'un apprenant. Cette approche est justifiée par le fait que les consultations successives des activités de chaque apprenant en vue d'une perception globale de l'activité de la classe peuvent conduire à une surcharge cognitive pour le tuteur (également mis en évidence dans [FHM+06]). Enfin, même en ne considérant que l'enseignant-tuteur, nous pouvons noter que la forme d'observation peut être très différente selon les objectifs pédagogiques visés et selon la situation des cours et que le résultat obtenu par exploitation des traces sert à l'adaptation de la session d'apprentissage et au soutien fourni aux apprenants.

1.3.3 Traces comme support d'observation pour le concepteur de l'activité

Dans un EIAH, les activités proposées aux apprenants sont généralement décrites sous forme de scénarios pédagogiques prescrits, représentés en utilisant des langages de scénarisation spécialisés comme IMS-LD [Con03] ou LDL [FMV07] par exemple. Le scénario pédagogique permet au concepteur de l'activité de définir le parcours d'apprentissage des apprenants par rapport aux objectifs pédagogiques visés pendant la séance.

Du côté concepteur du scénario pédagogique, l'exploitation des traces permet d'observer et suivre l'état d'avancement du processus recommandé, de l'analyser *a posteriori* et éventuellement

de le remettre en question. On s'inscrit ainsi dans une démarche d'amélioration continue de la qualité du scénario pédagogique proposé aux apprenants [MHCF04] ou de l'environnement informatique lui-même, approche correspondant à une démarche plus orientée réingénierie des EIAH [Cho07]. L'analyse des activités prescrites par rapport à un usage réel observé à partir des traces à l'issue d'une session d'apprentissage permet donc de faire évoluer le scénario prescrit. Par exemple, le concepteur du scénario peut décider d'insérer une nouvelle activité ou d'enlever une activité du scénario afin de le rendre plus explicite/adéquat au regard de l'objectif pédagogique et aux difficultés rencontrées par les apprenants durant les sessions précédentes.

1.3.4 Traces pour le chercheur visant à interpréter l'activité collaborative

De plus, l'observation de l'activité pédagogique réalisée dans un EIAH représente aussi une source de données pour les chercheurs (en sciences de l'éducation, sciences de l'information, sciences humaines et sociales) qui permet d'étudier les situations d'apprentissage et le comportement des apprenants (voir par exemple [CCG05]). L'approche permet d'analyser le comportement des apprenants dans un contexte individuel mais aussi dans un contexte d'apprentissage collaboratif.

Par exemple, il devient possible d'étudier les caractéristiques des différents groupes d'apprenants, la dynamique des interactions et des échanges entre les apprenants, l'utilisation des ressources et des outils mis à disposition des apprenants, les stratégies et les actions de l'enseignant influençant le processus d'apprentissage mis en place. Il est possible également de mettre en exergue de nouvelles pratiques. Par exemple, dans [CCG05], l'analyse des usages montre une utilisation nomade de l'E.N.T. de l'université de Savoie «cartable électronique» plus spécifiquement pour les outils non collaboratifs alors que les outils collaboratifs sont utilisés de préférence lorsque les étudiants sont présents sur le site de l'université. C'est donc un autre type d'exploitation de trace qui est attendu ici et qui s'adresse à des non informaticiens pour l'interprétation des traces d'actions collaboratives complexes.

1.3.5 Projet « personnalisation des EIAH »

Le projet de recherche dans lequel s'inscrit ce travail conforte plus la diversité des usages, des exploitations et des acteurs concernés par la manipulation des traces. Le projet étant structuré autour de cinq tâches montre également la variété des besoins et des objectifs potentiels pouvant être mis en œuvre. Dans la suite, nous allons donner une brève description des besoins les plus essentielles dans le cadre du projet.

Une représentation homogène de traces hétérogènes

Les besoins des différentes tâches du projet sont multiples et nécessitent évidemment des représentations communes pour pouvoir comparer, rapprocher et faciliter la réutilisation des pratiques impliquant des traces. La majorité des travaux du projet repose sur des systèmes

hétérogènes pour stocker et traiter les traces. L'évolution des technologies, la distribution géographique des équipes, l'évolution des techniques d'exploitation et la multiplication des usages de traces contribuent grandement à cette diversité. Les systèmes mis en place ont été conçus indépendamment les uns des autres, avec des modèles, des langages et des objectifs différents. La plupart des EIAH ainsi que leurs usages des traces n'ont pas été créés pour être interopérables. Cependant, le désir d'industrialisation, de réutilisation des techniques et les collaborations incitent ces EIAH et les pratiques à base de traces hétérogènes à être homogènes. Un des objectifs de notre travail est de permettre cette interopérabilité.

Permettre l'interopérabilité entre systèmes utilisant des traces commence par normaliser les interfaces entre des EIAH autonomes, de sorte que les traces produites par un EIAH peuvent être considérées comme des entrées pour un autre. Ceci implique, au-delà de l'usage et l'adoption de représentations communes, la définition de modèles prenant en compte les invariants entre traces issues de différents EIAH mais aussi leurs spécificités (les temporalités, les modalités d'exploitations, etc.). Ceci est un des besoins partagés par toutes les tâches du projet.

Une architecture générale pour l'exploitation des traces

Soutenir des échanges et des collaborations entre travaux exploitant des traces au sein des différentes tâches et avec des chercheurs externes nécessite l'identification des processus et les phases concernées par leurs exploitations. Ceci peut être caractérisé par une architecture et un vocabulaire commun permettant de positionner les pratiques des uns par rapport aux autres. En effet, une architecture doit décrire la structure des composants qui en font partie, leurs inter-relations mais aussi les principes et les règles régissant leur usage. Dans le cas des systèmes utilisant des traces, l'architecture se doit, en plus de permettre des extensions pour de nouveaux besoins (non identifiés lors de la conception), de soutenir une insertion facile dans l'environnement dans lequel elle opère.

L'architecture doit aussi servir à considérer des systèmes existants utilisant explicitement ou implicitement des traces. Ceci permettra l'étude des systèmes existants en comparant les modules, les services offerts par ces modules, leurs inter-connexions. Ceci permettra d'identifier les invariants en terme de modèles et de processus, les dépendances spécifiques (aux domaines, applications, etc.) et les pratiques récurrentes (applicables dans différents contextes) pouvant être considérées comme génériques. Le besoin d'une architecture qui permette de construire de nouveaux systèmes, tout en prenant en charge les systèmes existants est aussi un besoin exprimé par toutes les tâches.

Des langages spécifiques et performants pour la manipulation des traces

Différentes technologies sont exploitées pour mettre en place des outils exploitant les traces (XML, systèmes de gestion de bases de données relationnelles, etc.). Que ce soit pour le projet et les différentes tâches ou de manière générale, l'usage des traces requiert des langages permettant de prendre en compte les aspects spécifiques de leurs exploitations (en terme d'expressivité, de

mode d'évaluation, etc.).

L'analyse de traces après une session d'apprentissage par l'enseignant-chercheur, le diagnostic à base de traces en temps d'apprentissage pour un tuteur, la réflexivité des tâches accomplies pour l'apprenant sont autant de fonctionnalités réalisées et implémentées dans les différentes tâches du projet. La réification et mise en œuvre des modalités d'exploitation de traces (synchrone/asynchrone, local/à distance, etc.) se fait généralement par un couplage entre des langages de programmation généraux (Java, php, C++, etc.) et des langages dits déclaratifs pour l'interrogation de données (e.g. XSLT/XQuery, SQL, etc.). Ce couplage est d'autant plus important lorsqu'il s'agit de prendre en compte certains aspects non couverts par les langages déclaratifs comme la prise en compte de modèles temporels différents, la détection de certains motifs spécifiques, l'évaluation continue de l'observation des informations et enfin pour la mise en œuvre d'inférences déductives, bayésiennes ou autres. L'usage de plusieurs langages va à l'encontre des besoins exprimés par toutes les tâches et nuit à la facilitation des échanges, la réutilisation des diagnostics et même à l'évolution des exploitations des traces. Il est intéressant de considérer les langages déclaratifs, comme ceux définis pour la représentation des connaissances, pour prendre en compte complètement les aspects relatifs à l'exploitation des traces. Ceci simplifiera grandement les développements en minimisant les usages des langages hôtes des EIAH et permettra de soutenir réellement une réutilisations des traitements. Là aussi, ce besoin est prédominant pour toutes les tâches du projet, et plus généralement dans tous les travaux à base de traces.

1.3.6 Synthèse concernant les usages de traces

Ce tour d'horizon rapide des besoins montre que l'exploitation des traces est au cœur de plusieurs pratiques guidées par différents acteurs ayant diverses rôles dans l'EIAH. Il nous a permis aussi de mettre la lumière sur les différentes utilisations qui peuvent être faites de l'observation dans un cadre pédagogique. On se rend bien compte que le domaine est riche, les possibilités sont multiples et les objectifs sont divers et que les outils pour exploiter les traces doivent soutenir ces utilisations. Dans ce contexte, deux aspects primordiaux motivent ce travail de thèse :

Généricité des outils exploitants les traces. Si la littérature montre des travaux ciblés sur des points précis, il est difficile de trouver une approche de l'observation suffisamment générale pour pouvoir être étendue à ces nombreux besoins. Au meilleur de notre connaissance, peu de travaux proposent des solutions pour construire des systèmes permettant des observations à base de traces pour ensuite exploiter ces observations pour soutenir plusieurs personnalisations, faites par différents acteurs, pour de multiples objectifs. En effet, l'observation à base de traces au sein des EIAH est nécessaire pour rendre compte de services de personnalisation extrêmement variés, qui comme nous l'avons vu, varient selon les rôles, les besoins des utilisateurs, les objectifs pédagogiques et peuvent même changer au cours d'une même session (on ne se focalise plus sur les mêmes informations).

Vers des exploitations de traces pendant et après l'activité. L'exploitation des traces s'oriente de plus en plus vers des exploitations en temps réel de l'activité, par exemple le diagnostic des activités des apprenants directement sur leur machine pendant l'activité pédagogique. Cependant, l'exploitation de traces se fonde aussi sur des pratiques bien établies exploitant des traces très riches et souvent de bas niveau d'abstraction *a posteriori* de l'activité. Cette dualité dans l'usage nécessite la spécification de nouveaux outils pour la manipulation des traces en temps quasi-réel et en temps différé du traçage.

La construction de nouveaux outils pour l'exploitation des traces, génériques et capables de prendre en charge les différents besoins de personnalisation pour diverses utilisateurs soulève un certains nombre de problématiques. Ceci est d'autant plus fondamental lorsqu'on considère deux modalités d'exploitation dans un même système : les exploitations précises en temps réel de l'activité et des exploitations minutieuses et réfléchies *a posteriori* de l'activité. La section suivante détaille les problématiques auxquelles nous nous intéressons particulièrement et le résumé de nos contributions.

1.4 Problématiques traitées

Si le cadre applicatif et l'inspiration principale de la thèse relève du domaine des Environnements Informatique pour l'Apprentissage Humain (EIAH) et plus particulièrement de la problématique de la personnalisation des EIAH et par là des apprentissages, ce travail s'inscrit clairement dans le domaine de l'ingénierie des connaissances et plus particulièrement de la représentation de connaissances et des langages de requêtes associés. Les traces informatiques sont en effet des inscriptions de connaissances et il particulièrement productif et utile d'en fournir la sémantique descriptive et opérationnelle.

1.4.1 Représentation et modélisation des traces

Actuellement, les modèles de traces, leurs formats et leurs traitements sont en général spécifiques à l'application qui les utilise. Chaque application produit ses propres traces avec des modèles et des méthodes de traitement spécifiques. Un de nos objectifs est de pouvoir exploiter les traces provenant de sources hétérogènes en les intégrant pour répondre à des requêtes d'applications de plus haut niveau.

Une représentation générique des traces est nécessaire afin de fournir une vue commune des sources de traces hétérogènes. Nous devons identifier les aspects communs mais aussi spécifiques aux traces pour en proposer un modèle qui tout à la fois couvre de très nombreuses possibilités de traçage et respect les spécificités liées au statut de trace. Dans le cadre des travaux sur les traces, nous avons identifié quelques caractéristiques complémentaires (ou aspects) qui doivent être pris en compte par la représentation des traces : les éléments de la traces sont typés et

structurés (i.e., pouvant avoir des attributs), pouvant être liés par des relations typées et sont situés temporellement les uns par rapports aux autres (par rapport à une représentation du temps de la trace).

Si on considère les traces comme des inscriptions de connaissances, différents langages de représentation de connaissances peuvent être utilisés. Quelle représentation doit-on utiliser pour modéliser des traces ? A quel point un langage de représentation de données ou de connaissances réifie-t-il chacun de ces aspects ? Ce travail de thèse donne des éléments de réponse et des mesures concrètes pour juger la pertinence et l'expressivité des langages pour la représentation des traces.

1.4.2 Traitements complexes et inférences continues sur des traces infinies

La nécessaire reformulation des traces issues de l'observation pour les amener dans le registre permettant calculs (indicateurs) et interprétations (humaines) impose des requêtes complexes dont les réponses sont utilisées pour proposer des feedbacks en quasi temps-réel. Ces requêtes incluent des opérateurs comme des conjonctions, des disjonctions ou la négation mais doivent prendre en compte aussi les *relations temporelles et structurelles* entre éléments de la trace. Nous nous intéressons donc aux langages d'interrogation de données prenant en compte les aspects temporels inhérents à toute trace.

Comme inscription de connaissances, les traces peuvent représenter des connaissances si des mécanismes de raisonnement spécifique lui sont associées. Raisonner consistera en l'essentiel à déduire logiquement certains éléments comme « vrais » à un certain moment quand une combinaison logique et satisfaisant certaines contraintes temporelles sera « vraie » dans la trace. Les éléments déduits enrichissent la trace en permettant des substitutions par équivalence, ce qui permet de concevoir un mécanisme permettant d'inférer une trace à partir d'une autre.

Les règles (déductives) permettent de définir de nouveaux éléments à partir des éléments de la trace existants. Exprimer des règles dans un certain langage pour traiter les traces est ainsi hautement souhaitable. Les règles peuvent servir de mécanisme de reformulation rendant les traitements sur les traces plus lisibles et les traces plus expressives. Elles permettent de transformer les traces de bas niveau (e.g. événements système, actions basiques, etc.) en traces de haut niveau (e.g. tâche spécifique, situations complexes, etc.). Différentes règles peuvent fournir différents points de vue (par exemple, pour les apprenants, les enseignants, les administrateurs, etc.) sur le même système. Les règles permettent de servir de médiateur entre les différents modèles et représentations des traces. En outre, les règles peuvent être bénéfiques lorsque le raisonnement sur des relations entre les éléments de la traces est requis (raisonnement temporel, raisonnements sur les relations hiérarchiques, etc.).

La mise en place de règles permettant de transformer une trace en une nouvelle trace requiert des propriétés spécifiques non offertes par les langages existants notamment dans le cas d'une évaluation incrémentale et continue. En effet, les traces considérées sont conceptuellement infinies, i.e., recevant continuellement des éléments issus des outils d'observations (e.g., capteurs, sondes logiciels, etc.). Certains aspects deviennent importants et doivent être traités au

moment même de leur collecte. Ceci requiert de nouveaux opérateurs comme des conjonctions et disjonctions temporelles, prises en charge des relations temporelles relatives (e.g., `extends`, `shift-forward`, etc.), et absolues (`before`, `after`, etc.), filtrages selon des conditions sur les éléments de la trace, etc. Quel langage peut-on utiliser pour l'interrogation et la transformation de traces ? A quel point un langage d'interrogation et de transformation de traces est-il adapté aux traces potentiellement infinies ? Ce sont là des questions fondamentales auxquelles ce travail de thèse s'efforcera de répondre.

1.4.3 Fondements formels pour les langages de manipulation de traces

Les systèmes et les langages traditionnels (non spécifiques aux traces) pour exprimer des requêtes et/ou des règles ont de très solides fondements formels. Bien qu'un certain nombre de langages classiques peuvent être utilisés pour les traces, la représentation et manipulation de traces n'a toujours pas de bases formelles comparables. Ce manque de bases formelles a aussi été relevé et identifié par le projet personnalisation de EIAH, à la fois pour les concepts relatifs à la notion de trace et les langages permettant son exploitation. Ceci est également une des problématiques centrale à laquelle cette thèse tente de répondre. Il est intéressant d'étudier, notamment pour les langages de représentation de connaissances, les propriétés qui font changer, utiliser ou étendre ces langages et leurs sémantiques existantes pour la manipulation de traces.

1.5 Résumé des contributions

La proposition principale de cette thèse est la définition de la notion de *Système à Base de Traces modélisées (SBT)*. Un Système à Base de Traces modélisées est défini comme une classe particulière de système à base de connaissances facilitant et supportant la manipulation et le raisonnement sur des traces modélisées en tant que telles. Pour définir cette notion, notre approche s'articule sur deux propositions.

1.5.1 Cadre conceptuel des systèmes à base de traces modélisées

Nous proposons un cadre conceptuel fournissant une terminologie et un vocabulaire communs permettant à différentes communautés de modéliser, de parler et rapprocher des exploitations de traces existantes. Ce cadre conceptuel permet également de développer de nouveaux systèmes à base de traces modélisées interopérables, offrant des langages pour modéliser et interpréter les traces. Pour traiter l'hétérogénéité des logiciels utilisant les traces, une architecture flexible et modulaire a été définie. Nous adoptons une approche orientée service. Les fonctionnalités d'interrogation et de transformation de traces sont donc fournies en termes de services. Ce cadre conceptuel définit autour de la notion de trace modélisée les services intrinsèques requis pour son exploitation.

Notre approche est flexible puisque d'autres modules, services ou EIAH peuvent être construits, et ce, en se fondant sur les services de base offerts. Elle est générique puisqu'elle permet de

construire de nouvelles fonctionnalités en temps réel ou *a posteriori* de l'activité pouvant être utilisée par plusieurs acteurs et dans différentes situations.

1.5.2 Cadre formel pour les langages d'interrogation et de transformations de traces

Nous proposons un cadre formel définissant la notion de modèle de trace, de trace ainsi que les langages pour leur interrogation et transformation. Nous définissons la sémantique de notre langage sous forme d'une théorie des modèles (dans le style de Tarski [8]) et d'une théorie de point fixe, deux formalismes habituellement utilisés pour décrire la sémantique formelle des langages de représentation de connaissances. Notre sémantique est déclarative et présente l'avantage important de prendre en compte aussi bien les requêtes et les règles de transformation pour des évaluations ponctuelles et continues, deux aspects souvent négligés dans la sémantique des langages existants. Bien que l'approche par théorie des modèles soit bien établie pour les langages traditionnels de manipulation de connaissances, son usage, application et adaptation aux langages de requête et de transformation de traces est nouveau. À notre connaissance, il s'agit de la première proposition d'un langage permettant la prise en charge complète des aspects requis par l'interrogation et la transformation continue de traces explicitement modélisées et potentiellement infinies. Dans ce sens, nous démontrons que notre sémantique est appropriée pour l'interrogation et la transformation des traces ouvertes à de nouvelles adjonctions, s'augmentant continuellement au fur et à mesure de la collecte de nouvelles observations.

8. i.e. récursivement définie sur la structure du langage

Chapitre 2

Traces numériques et leurs exploitations par les EIAH

Sommaire

Ce chapitre fait le tour d'horizon des différents usages de traces dans le cadre des EIAH. Après avoir préciser de la notion de trace (section 2.1), ce chapitre présente les travaux impliquant des traces dans trois usages correspondant à différents temps d'exploitation de traces. La discussion précise les propriétés des outils permettant de les exploiter.

2.1 Notion de Trace : définitions et propriétés

Cette section présente une discussion autour des différentes notions de trace, et de leurs significations et caractéristiques dans différents contextes d'usage. Bien que la notion de trace en général, ne soit que rarement prise pour objet de réflexion ou de recherche [Ser02], elle apparaît régulièrement dans des domaines de recherche liés au paradigme documentaire, à l'étude des usages et des comportements et à l'ingénierie des connaissances en général. C'est le cas par exemple en sciences de l'information ou en histoire où l'on articule la notion de trace avec celle de mémoire et de document : on parle de *trace-indice* et de *document-trace* [Cha04]. Dans cette section, on s'intéressera à la définition du terme *trace* en général tout en l'étendant pour définir sa signification considérée dans les EIAH.

La définition du terme « Trace »

L'utilisation commune du terme *trace* dans le langage courant peut donner l'illusion d'une simplicité de la notion qu'elle ne possède pourtant pas. Signifiant à l'origine « l'empreinte ou la suite d'empreintes laissées par le passage d'un homme ou d'un animal » [Ser02], le terme possède aujourd'hui plusieurs acceptions.

Dans sa définition la plus générale, une *trace* est définie comme *« l'influence d'un événement sur son environnement »* [Wik09]. Lorsqu'on s'intéresse à l'usage de ce terme dans différentes disciplines, cette notion se décline selon plusieurs facettes précisant d'une certaine manière la définition générale. On retrouvera ainsi dans certaines définitions des précisions des termes utilisés dans la définition générale, e.g., concernant *l'influence* pratiquée, *l'environnement* influencé et *l'événement* qui a provoqué cette influence. On peut relever donc des définitions de traces comme étant *« une suite d'empreintes ou de marques que laisse le passage d'un être ou d'un objet »* [LeP05] ; *une marque laissée par une action quelconque*[Lar09]. Le terme trace fait ainsi référence à plusieurs sens : en littérature, *trace est ce qui subsiste de quelque chose du passé sous la forme de débris, de vestiges, etc.* [Lar09] ; en psychologie, la trace désigne *ce qui subsiste dans la mémoire d'un événement passé* [Lar09]. Au vu de ces différentes définitions, le mot *trace* regroupe finalement beaucoup de sens.

Dans l'optique de préciser sa signification, Serres [Ser02] analyse le terme *trace* et l'évolution de sa définition. Il argumente que ce terme, bien d'une grande banalité, a vu sa polysémie s'étendre au fil des siècles. Le terme *trace* a d'abord signifié, vers 1120, l'empreinte ou la suite d'empreintes laissées par le passage d'un homme ou d'un animal, puis d'une chose (1690) ou encore un exemple à suivre (1530). Un deuxième sens de la trace se développera (à partir de 1250) sur l'idée de « ce qui subsiste du passé » notamment en mémoire. La définition première de la trace, dont de nombreux sens (figurés ou non) ont dérivé, est donc bien celle de l'empreinte, matérielle ou morale, celle de la « marque laissée par une action » [Ser02]. C'est sous cet angle que nous souhaitons définir la notion générale de trace.

Définition 1 (Notion générale de Trace)**.** *Une Trace est la marque laissée par une activité.*

Derrière son apparente simplicité, la notion de trace cache donc une certaine complexité. Comparable en ce point à la notion de «document» [Pé03], la notion de trace n'a quant à elle que rarement été prise pour objet de recherche. Parmi les différentes facettes de la notion, nous nous focalisons dans ce chapitre sur la notion de *trace numérique* telle qu'elle est considérée dans les systèmes d'information comme la «marque laissée» lors d'une activité médiée par une plateforme informatique.

Mais qu'est-ce qu'une trace numérique ?

Comment peut-on définir le plus sûrement une *trace numérique* ? On serait tenté de reprendre la définition générale de wikipédia [Wik09] et de répondre qu'il s'agit de *l'influence d'un événement informatique sur son environnement.* Une telle définition reste évasive et limitée, car la trace en informatique dite trace numérique est souvent liée à la notion *d'historique* et de *traçabilité* et elle n'est que rarement considérée sous l'angle de *l'empreinte* ou *la marque laissée par une action*, qui est pourtant son sens premier. La trace numérique est donc abordée par plusieurs aspects, du témoignage historique au pistage de produits industriels et peut être considérée comme un objet reflétant plus ou moins la capacité à conserver des informations concernant les états, les services, les données exploitées par un système informatique. Et c'est d'ailleurs dans ce sens que nous l'avons définie dans [SPMM07] : « *la trace numérique est trace de l'activité d'un utilisateur qui utilise un outil informatique pour mener à bien cette activité, s'inscrivant sur un support numérique* ».

Cette définition nous met sur deux pistes : la propriété des traces à garder des informations sur ce qui s'est passé et aussi le fait qu'une trace est enregistrée sur un support numérique matérialisant son existence. En considérant cette définition, la trace est avant tout un ensemble d'informations liées à l'activité qui s'est s'effectuée dans un environnement informatique. Cependant, cette définition ne nous permet pas d'échapper à l'hégémonie du terme *trace* et de sa signification première et générale (cf. définition générale 1) comme *la marque laissée.* Ceci est bien souvent source d'ambiguïté dans les travaux considérant des traces. Bien qu'il n'y ait qu'une seule notion de trace qui prédomine dans le domaine de l'informatique, celle de l'histoire d'utilisation d'un environnement informatique, il existe, dans bien d'autres travaux plusieurs considérations, toutes faisant référence implicitement ou explicitement à la signification générale.

En effet, la notion de trace numérique peut prendre le sens général et être vu comme tout objet restant et subsistant d'une activité, comme une marque laissée lors de l'utilisation d'un environnement numérique. On considérera donc comme *trace*, l'historique de l'utilisation de l'environnement mais aussi toute production, résultat ou objet informatique subsistant de l'activité médiée par la machine informatique. Par exemple, supposons qu'on trace l'activité de rédaction d'un mémoire de thèse, la trace sera la séquence des phrases, textes, mots, les cliques de souris, les copier-coller, les commandes latex, etc. qu'un doctorant a effectué pendant son activité de

rédaction. Le document de thèse est aussi une trace, résultant et produit lors de la rédaction mais au sens général de la définition 1, mais cette trace peut être fondamentalement différente de la trace composée de séquences de phrases, commandes, etc. Par exemple, le document pdf de la thèse ne décrit pas explicitement l'historique du processus de l'activité de rédaction puisque le doctorant peut écrire l'introduction après la conclusion ; il peut aussi commencer par rédiger la fin d'un paragraphe avant son début sans que cela ne se reflète dans le résultat final. En fait, ce document produit peut être un élément de la trace de l'historique et non pas toute la trace si par exemple le doctorant utilise un système de gestion de version (SVN) qui trace les différentes versions de son document de thèse. La trace sera l'historique des versions de son document contenant l'ensemble des versions de sa thèse. La temporalité de la trace est le temps réifiant la chronologie des différents versions qui ont été sauvegardées sur le SVN.

La trace numérique ? histoire ou résultat d'une activité ?

La distinction entre trace considérée comme produit restant et subsistant d'une activité au sens général (cf. définition 1) et la trace considérée comme l'histoire d'interaction et d'utilisation d'un environnement informatique est sujette à confusion dans beaucoup de travaux exploitant des traces. Bien que l'intersection entre ces deux considérations soit loin d'être nulle et au-delà des divergences entre points de vue sur ce que pourrait être une trace, les objets considérés comme étant des traces se retrouvent souvent être différents ou très hétérogènes. Actuellement, il n'existe pas une définition commune de la notion de trace numérique précisant clairement la distinction entre ces deux considérations, car *« la trace est toujours trace (au sens général) de quelque chose... »* [Ser02]. Cependant, il est quand même possible de distinguer une définition objective et une définition subjective de la notion de trace numérique.

Dans la définition objective de la *trace*, il s'agit d'un ensemble de données issues d'un processus de traçage, ordonné par la chronologie de l'utilisation ou l'inter-action tracée, propre à revêtir le sens d'une histoire ou historique particulière lors de l'observation d'une activité médiée par une machine. Tout ensemble de données, reflètant une séquence implicite ou explicite, porteur de sens d'historique produite dans un objectif d'observer un système numérique et son utilisation sera qualifié de trace. Ainsi, par exemple un fichier log issue d'un traçage d'une activité de rédaction de mémoire de thèse contenant des événements concernant la rédaction de texte, l'exécution d'une commande latex, etc.) sera considéré comme une trace numérique, ou si l'on préfère trace numérique liée à l'activité de rédaction de thèse.

Au côté de la définition objective de la trace qui consiste à considérer que tout objet informatique conteneur d'une séquences d'informations relatives à l'utilisation d'un système informatique peut être une trace, une autre approche, considère uniquement le regard que porte l'être humain sur cet objet qui le rend trace. Cette conception est éminemment *subjective*, puisque ce n'est plus l'objet en lui-même (le fichier contenant l'historique des événements concernant la rédaction de mémoire dans la définition objective) qui est trace de par ses propriétés, mais c'est le regard extérieur et subjectif ne tenant pas en comptes des caractéristiques intrinsèques devant

être véhiculées par la trace qui est son créateur. Ainsi, sous la considération de la définition subjective, le document mémoire est lui même une trace de l'activité de rédaction de mémoire, parce que et seulement parce que l'utilisateur considéra qu'il est trace subsistant de son activité (usant dans ce cas de la définition 1).

Cependant, si on considère ces deux définitions dans le cas d'une trace d'une activité médiée par un outil numérique, la définition objective est plus précise car elle considère que la trace doit avoir le plus d'informations précises sur l'usage de l'outil et la chronologie de l'activité tracée en plus des informations produites. Ainsi, tout objet informatique est une trace numérique d'une activité si et seulement s'il décrit de la manière la plus fidèle possible la chronologie des interactions effectuées durant cette activité. Ainsi, lorsqu'on parle de trace numérique d'une activité médiée par un système informatique, la définition la plus précise est celle qui identifie objectivement l'objet informatique par les propriétés qu'il possède.

La trace numérique d'interactions médiatisées et médiées

Tout d'abord, il convient de préciser la notion d'interaction. Baker et Lund [BL97] distinguent les interactions entre humains des interactions entre un humain et un artefact (e.g. un ordinateur). Ils définissent une interaction (action entre) comme étant une suite d'actions verbales ou non-verbales qui sont interdépendantes pouvant s'influencer mutuellement. Dans le cas où un humain interagit avec un ordinateur, il est préférable de parler d'une suite d'événements prenant la forme d'une action-réaction [Mil06a]. Dans ce mémoire, il s'agit d'humains interagissant entre eux au travers un ordinateur ainsi que d'un humain interagissant avec un ordinateur. Malgré la distinction qui vient d'être faite — une distinction importante puisqu'elle reflète la nature différente des informations tracées — le terme « interaction » sera utilisé plus généralement afin évoquer les deux cas.

Quant à l'interaction entre êtres humains au travers d'un ordinateur, les expressions « d'interaction médiatisée » et « d'interaction médiée » par ordinateur se trouvent toutes deux dans la littérature. Il est également utile pour notre propos de les distinguer. Pour Peraya et Meunier [Per03, MP04], les processus de médiatisation et de médiation portent sur des objets différents. La médiatisation désignerait le processus de mise en dispositif médiatique, incluant ainsi le choix de médias et d'une scénarisation, alors que la médiation serait le processus par lequel un artefact technique et symbolique peut modifier les processus de production et de communication des connaissances ou encore influer sur l'apprentissage ou sur le développement des processus cognitifs des êtres humains [CDD07]. Dans ce mémoire, il s'agit des deux : médiatisation et médiation.

Nous sommes maintenant en mesure d'étoffer la définition d'une trace numérique d'interactions dont il est question dans cette section.

Définition 2 (Notion de trace numérique d'interactions)**.** *Une trace numérique d'interactions est une suite temporellement située d'éléments, qui relève soit d'une interaction entre humains, médiatisée et médiée de diverses façons par une machine soit d'une suite d'actions et réactions*

entre un humain et une machine [MM09].

Notion de Traces pour les EIAH

Avant de s'intéresser à l'utilisation des traces, il faut d'abord lever certaines ambiguïtés associées au terme de trace dans le contexte des EIAH. Certaines définitions sont trop générales pour permettre d'identifier l'hétérogénéité des objets informatiques réellement manipulés et considérés comme des traces. Par exemple, pour Pernin [Per05], la trace représente un indice de l'activité des acteurs d'une situation d'apprentissage qu'elle soit ou non instrumentée. À son sens, la trace peut représenter un résultat obtenu au cours ou au terme d'une activité ou alors un événement ou un ensemble d'événements relatifs au déroulement de l'activité. Ce point de vue est partagée par Choquet [Cho07] qui considère la trace comme un objet pédagogique au même titre que les ressources ou les scénarios pédagogiques, en la définissant comme pouvant relever du résultat (au sens production laissée par l'apprenant) ou bien de l'ensemble des actions de l'utilisateur.

Dans le contexte des EIAH, la notion de trace peut avoir deux significations : (1) l'histoire interactionnelle d'un apprenant utilisant un système informatique ou (2) les productions qu'il a laissées lors de son apprentissage. Pour la première considération, la trace est alors l'empreinte restante de l'activité (définition 1) relatant l'historique et la chronologie de l'interaction de l'apprenant (définition objective). Dans le second cas, la trace n'est plus trace à cause de sa capacité à «retracer» l'interaction, mais uniquement parce qu'elle est un élément passé restant et résultant de cette activité (correspondant à la définition subjective).

En fait, la personnalisation considère la notion de trace sous les deux angles simultanément : la trace comme processus du déroulement de l'activité et les productions laissées lors de l'apprentissage. Toutefois, Akhras et Self [AS96, AS99] argumentent que le plus important pour un environnement d'apprentissage, pour son adaptation et sa personnalisation, est la prise en compte du processus d'apprentissage. Sous cet angle, la notion de trace considérée seulement sous l'angle du résultat d'une activité va donc omettre un des points fondamentaux de la personnalisation puisque les productions résultant des interactions ne réifient pas toujours la chronologie du processus d'apprentissage soutenant leurs création. Cette notion de trace met en avant l'image finale du processus d'apprentissage, tout en reléguant en arrière plan les caractéristiques et la chronologie du processus produisant cette image. En plus, cette notion tend à s'approcher plus de la signification générale du terme *trace* que des concepts informatiques réellement manipulés par les systèmes existants. En pratique, il est difficile de considérer la définition subjective pour construire des outils facilitant l'exploitation de traces numériques puisqu'elle regroupe des objets informatiques très hétérogènes fondamentalement différents.

C'est sous un regard objectif qu'on souhaite étudier la notion de trace numérique telle qu'elle est considérée par la majorité travaux en EIAH. Ainsi, dans ce mémoire de thèse, nous nous intéressons plus particulièrement à la notion de trace numérique d'interaction que nous considérons dans sa définition objective, comme étant une succession d'informations temporellement situées inscrites sur un support numérique lors d'une observation par le système informatique

de l'interaction d'un utilisateur avec ce même système ou un autre.

Après cette discussion autour de la notion de trace, nous allons aborder dans la suite les différentes manières d'obtenir et de récupérer des traces. Produire des traces est une opération parfois complexe et fastidieuse nécessitant dans certaines situations des efforts considérables pour instrumenter et récupérer les données tracées. Nous discuterons dans cette partie les différentes méthodes permettant d'obtenir des traces. Ces méthodes ont d'ailleurs une grande influence sur la nature et le contenu des traces ainsi que sur leurs exploitations.

2.2 Comment obtenir des traces ?

On peut dire de façon schématique qu'il existe deux types d'approches pour obtenir des traces numériques. La première consiste à utiliser des informations enregistrées automatiquement dont l'exemple typique est le log. Si on peut effectivement récupérer des logs facilement, il est plus difficile d'affirmer que l'on peut en faire des traces pertinentes pour une activité donnée [RTL06]. Une des raisons est que les logs récupérables sont intrinsèquement liés au modèle de conception des outils qui les génèrent. Ce modèle de conception ne correspondant pas toujours au modèle d'utilisation [MP06], les traces de type log restent souvent éloignées de la sémantique de l'activité tracée (e.g. les historiques Web classiques).

La seconde approche, au contraire, vise à implémenter de façon *ad hoc* un ensemble fini de traces possibles, à un niveau de reformulation correspondant au registre de l'activité. C'est le cas par exemple de certains EIAH qui proposent à l'apprenant de se situer, au fur et à mesure de son travail, dans un espace de concepts qu'il doit acquérir, en lui indiquant « où il en est » [Her02]. Nous détaillons dans la suite ces deux approches.

Les traces dans les systèmes traçant nativement leurs activités

Pour obtenir des traces, il suffit le plus souvent de récupérer directement des fichiers, nommés fichiers journaux (log file) inscrivant les plus importants événements ayant marqué ou changé l'état d'une application informatique. Sans être amené à modifier le code source, il est en effet possible dans beaucoup de systèmes informatiques [9] de récupérer une trace relatant des informations sur les états de l'application utilisée. Même si ces traces peuvent être utilisées pour observer l'interaction de l'utilisateur avec le système, elles ont été initialement pensées plutôt pour observer le fonctionnement du système et son administration. L'histoire de l'informatique explique en partie cette pratique. Les premières pratiques de traçage avaient comme objectif d'observer l'exécution d'un programme informatique et de vérifier s'il est conforme aux spécifications écrites [Bat95, And98]. Les traces issues d'exécution de programmes permettaient aussi de disposer des états des différentes variables et registres afin de déboguer les erreurs et de corriger le code mis en œuvre.

9. e.g. serveurs de fichiers.

La tâche de déboguage et de vérification de programmes informatiques s'étant généralisée, une pratique récurrente s'est installée dans la conception de logiciels (surtout les serveurs), consistant à journaliser (ou à *logger*) leurs activités. Ceci a permis d'obtenir la forme la plus disponible des traces : les fichiers log. À la naissance des premiers réseaux informatiques, la plupart des serveurs [10] permettaient ainsi de journaliser ou de *logger* des informations vitales à leurs gestion. On assista ainsi à la naissance d'une nouvelle forme d'exploitation : des traces pour l'administration. L'exploitation de ces traces se faisait alors par des administrateurs qui veiller à minimiser l'espace disque qui pouvait être alloué à ces fichiers traces. D'ailleurs, ceci a même amené les administrateurs à coder des traces sous forme compressée et parfois cryptée visant avant tout à réduire le plus possible les couts élevés de stockage. De nos jours, le traçage d'information est devenu une tâche courante, banale et même systématique dans certains systèmes informatiques comme les serveurs Web. Les traces journalières obtenues servent principalement :

- à l'aide à l'administration avancée de systèmes, e.g., reprise après une panne ou un blocage, détection d'erreurs système, etc. ;
- à la sécurité, e.g., détection des intrusions ;
- à la maintenance, e.g., pour obtenir les fichiers détruits par la fausse manipulation de la semaine dernière.

Dans ce contexte, beaucoup de systèmes existants [11] soutenant des études à base de telles traces ont été menées notamment dans le contexte du «Web». Toutefois, ces outils ont rapidement trouvé leurs limites notamment lorsqu'on s'intéresse à l'observation de l'activité de l'utilisateur et non seulement à l'activité du système informatique. En effet, même si les serveurs ont été conçus pour journaliser leur activité, ces traces ne sont ni tout à fait pertinentes ni tout à fait complètes [12]. De toute évidence, ces traces produites décrivent plus le fonctionnement interne et le comportement des serveurs plutôt que les interactions ou les comportement des utilisateurs. Certains travaux ont clairement mis en évidence les lacunes des fichiers log [Dav99, ZZ01, RV07].

Le Web a été un des premiers domaines à sortir les fichiers log de leurs rôle de matériaux d'administration (surement appâté par le potentiel commercial d'en savoir plus sur les préférences des utilisateurs). Ainsi, beaucoup de travaux se sont intéressé à l'exploitation de telles traces, notamment en ce qui concerne :

- l'évaluation de l'utilité et l'utilisabilité des sites web [Hoo97, BP97],
- la fouille des usages (Web Usage Mining) pour améliorer dynamiquement la navigation et la structure des sites Web [YJGMD97, CPY98, MPT99, PHMAZ00] ;
- l'adaptation et la personnalisation de pages web [MCS99, MDL+00, PPPS03, KAD04, ZZ01, RVZB09] ;
- le profilage des utilisateurs [WPB01, FMCL06].

10. e.g. les premiers serveurs de fichier dès les années 1980

11. e.g. outils d'administration, Log file analyser, etc. La majorité d'entre eux sont des produits commercialisés permettant notamment de faire des statistiques. Des outils comme http ://www.squid-cache.org/Scripts/, http ://www.xplg.com/home/products/, etc.

12. Pire, elles sont parfois trompeuses et peuvent conduire à des déductions incorrectes [Dav99].

La majorité de ces travaux mettent en évidence le manque de précision des traces log lorsqu'on s'intéressent aux interactions des utilisateurs avec les interfaces graphiques. En effet, les activités des utilisateurs ne sont pas observables directement sur les serveurs, mais seulement à travers l'interface graphique qu'il faudra instrumenter. Ainsi, les traces d'utilisation les plus significatives ont commencé à voir le jour grâce à une instrumentation volontaire des systèmes informatiques pour capturer enfin les informations utiles relatives aux interactions.

Des traces volontaires par instrumentation de systèmes informatiques

Les fichiers log et le traçage natif ayant montré leurs limites, l'instrumentation de logiciels a été de plus en plus considérée comme technique de traçage. L'appareillage et instrumentation de logiciels comme les navigateurs client comme Mozilla afin qu'ils fournissent les informations sur l'utilisation du système au delà de son fonctionnement interne a permis d'obtenir toutes les observations utiles et même, essentiellement dans le cadre du Web, des données habituellement inaccessibles comme l'interaction de l'utilisateur avec l'interface graphique. Néanmoins, l'instrumentation des logiciels elle-même est un obstacle, du fait du travail nécessaire et des précautions à prendre pour ne pas ajouter des erreurs au code original. De plus, ce travail d'instrumentation est à recommencer à chaque évolution du logiciel ou lorsque les préférences des utilisateurs se tournent vers un autre outil.

Les approches de traçage hybrides

Certains logiciels fournissent des traces sans besoin de reprendre leur code. Certains programmes donnent le moyen de configurer la quantité et la qualité des données tracées (e.g. serveur web apache). Si l'outil offre le moyen de paramétrer ce qui peut être tracé et si l'utilisateur configure correctement le programme qui génère ces traces, il est possible dans certains cas de maitriser les fichiers log que les logiciels produisent.

D'autres techniques de traçage permettent d'observer les logiciels dans leur environnement d'exécution, par exemple, en interceptant les cliques souris et les frappes clavier ou en espionnant les échanges avec leur environnement (communication réseau, appel de procédures, etc.). Ce type de traçage n'autorise pas toujours un niveau d'abstraction suffisant pour produire une trace pertinente, e.g. TaskTracker [SBD+05], GRUMPS [GMD+04].Ces techniques permettent d'avoir des traces collectées pendant le temps de l'activité tout en offrant la possibilité de paramétrer et réguler le traçage. Les informations contenues dans ce type de trace sont trop générales pour décrire pertinemment différentes activités. Un modèle commun standardisé de traçage d'application n'existant pas encore, ces systèmes sont génériques pour tracer et moins pour comprendre et exploiter ce qui a été tracé dans différents contextes d'utilisation.

Les traces issues d'une instrumentation volontaire d'un artefact informatique peuvent être exploitées et combinées avec les traces issues d'un traçage natif et/ou les traces issues d'un traçage de l'environnement. Ceci permet de combler les écarts en terme d'informations et de niveau d'abstraction. Par exemple, l'expérimentation dans [MHFC07] ou les travaux de Madeth May

[MGP07, MGP08] combinent ces trois manières d'obtenir des traces pour avoir les informations utiles à la compréhension de l'activité de l'utilisateur. Cependant, combiner des traces issues de plusieurs sources d'observation, utilisant différentes techniques de traçage nécessite des mécanismes évolués de synchronisation de temps et des interprétations pour pouvoir les exploiter. Ce type de traçage est souvent employé dans les expérimentations de certains EIAH.

2.3 Exploitations des traces dans les EIAH

Dans les sections qui suivent, nous présentons une sélection d'exemples de travaux de recherche récents impliquant les traces numériques d'interaction dans trois grandes familles d'usage de traces : 1) les études empiriques et l'analyse *post-hoc* des interactions, 2) la ré-ingénierie des EIAH, 3) et l'adaptation des interactions en temps réel. Dans chaque type d'usage, la trace est manipulée, par des acteurs pouvant être différents et pour des objectifs différents. Le temps de la manipulation en question sur les EIAH est également différent selon l'usage.

Pour chaque famille, nous allons mettre en évidence les caractéristiques qui différencient les traces entre elles. Les deux premières sections présentent les travaux faisant référence aux traces sur trois points : les modèles et représentations considérées pour les traces, les traitements et les processus mis en œuvre pour leurs exploitations et enfin une discussion des travaux présentés. Chaque section est déclinée suivant un ensemble de descripteurs : l'acteur qui produit, qui manipule et qui est le destinataire de la trace, sachant qu'il ne s'agit pas forcément du même acteur et enfin l'objectif précis de la manipulation ainsi que l'adaptation proposée. Puis nous proposons une synthèse des propriétés attendus pour ces différentes manipulations.

2.3.1 Études empiriques et analyse *post-hoc* de traces d'interactions

Les études empiriques, qu'elles soient menées dans un paradigme de psychologie expérimentale ou qu'elles aient pour but d'observer des situations écologiques, se focalisent sur une description des caractéristiques de l'interaction humaine, médiatisée et médiée par un outil informatique. Une manière d'observer et de décrire ces caractéristiques est de garder la trace des échanges, communications et des interactions des utilisateurs exploitant une machine informatique (ordinateur, téléphone, etc.). Beaucoup de travaux portant sur l'analyse de traces d'interactions se basent sur les possibilités natives de traçage offertes par certaines applications comme les navigateurs Web, et utilisent le plus souvent les traces comme des « objets d'investigation ». La complexité et la spécificité des traces dans le contexte d'apprentissage font souvent que ces études sont effectuées après le temps de l'observation i.e. après le déroulement des interactions observées.

L'analyse de traces d'apprentissage a vu son intérêt grandir avec l'évolution rapide des EIAH. La diversification des usages et des situations (apprentissage individuel et en complète autonomie, apprentissage en classe médié par un artefact informatique, apprentissage à distance et/ou en collaboration avec d'autres apprenants, etc.) a amené les chercheurs à s'interroger sur la na-

ture même de l'apprentissage [Ren00]. Même si les traces d'utilisation d'un EIAH permettent de rendre compte de sa qualité d'un point de vue « génie logiciel et IHM », rendre compte de la qualité d'apprentissage est néanmoins fondamentalement différent. À cette fin, des outils d'analyse de traces spécifiques ont vu le jour notamment pour supporter l'étude des caractéristiques du processus d'apprentissage, et ce, à partir de traces issues de sources multiples [MHFC07, AFK+07].

L'utilisation d'outils d'aide à l'analyse de traces est d'autant plus importante dans le contexte de l'apprentissage collaboratif. En effet, beaucoup de travaux de recherche se fondant sur les traces y ont été menés, généralement pour déduire les caractéristiques et la dynamique d'un groupe. L'objectif premier de ces travaux consiste à extraire les indicateurs pertinents, les stratégies et les modèles computationnels influençant l'apprentissage (voir notamment l'état de l'art de Dimitracopoulou [Dim04]).

Partant du postulat qu'un apprentissage collaboratif satisfaisant est un apprentissage permettant à un apprenant de collaborer et d'apprendre à collaborer [Jer04], ces environnements s'efforcent d'intégrer des outils d'analyse et de calculs d'indicateurs afin de guider et de conseiller l'apprenant et/ou le tuteur aussi bien dans les tâches d'apprentissage que dans les tâches de collaboration (certains exemples sont donnés dans Soller, Jermann et Muhlenbrock [SMJM05]).

On peut donner également l'exemple du système ColAT [AFK+07] développé comme outil externe permettant d'analyser des traces associées à des vidéos et des transcriptions issues de différents EIAH. L'acteur effectuant cette analyse peut être le chercheur étudiant une situation d'apprentissage spécifique ou l'enseignant qui a conçu les supports pédagogiques et les scénarios de collaboration. D'autres exemples de la littérature permettent d'illustrer des outils analysant les traces accompagnant des systèmes assistant la collaboration, nous allons en présenter certains dans les sections suivantes.

Modèles et représentations considérées pour les traces

Le premier exemple que nous présentons est l'outil Tatiana (Trace Analysis Tool for Interaction ANAlysts)[DGLC07, Dyk09] qui permet à un chercheur l'analyse de corpus de traces numériques d'interactions humaines.

Le système Tatiana est orienté vers des études empiriques sur les caractéristiques de l'interaction humaine. Tatiana permet d'effectuer des analyses qui ne sont pas de nature à être menées en temps réel. En effet, il s'agit de rejouer un ensemble d'éléments tracés qui ont été récoltés et synchronisés afin de comprendre, d'annoter et de coder toute une séance ou des séquences choisies en fonction de critères définis par le chercheur. Le rejouage et l'analyse de l'activité sont évidemment effectués non pas au moment de la collecte des données mais *a posteriori*. Conceptuellement, Tatiana se fonde sur la notion d'objet rejouable (*replayable*) qui est considéré comme une trace particulière permettant des visualisations et des synchronisations. Le modèle de Tatiana est illustré dans la figure 2.1 représente la classe des traces rejouables. Une trace enregistre des observables (*observable*), qui peuvent être enrichies par des relations

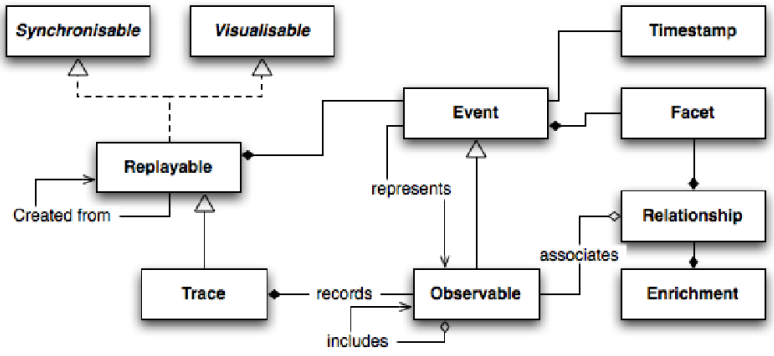

FIGURE 2.1 – Diagramme de classe UML pour représenter des objets rejouables [Dyk09]

(*relationship*). Les relations associent des facettes (*facet*) à des événements qu'elles enrichissent. Un rejouable est constitué d'événements. Tous les événements peuvent être composés de facettes et disposent d'un *timestamp*. Le modèle se focalise sur les traces qui peuvent être rejouées, i.e., synchronisables et visualisables dans l'outil les produisant.

Le deuxième exemple de modèles de représentation de traces est le fruit des travaux effectués dans les projets européens ICALTS (Interaction & Collaboration Analysis' supporting Teachers & Students' Self-regulation) [ICA04], IA (Interaction Analysis' supporting teachers & students' self-regulation) [IA05] et CAViCoLA (Computer-based Analysis and Visualisation of Collaborative Learning Activities) [CAV06] Ces projets se sont intéressés principalement à l'analyse des traces d'interactions notamment dans le contexte collaboratif avec pour objectif la définition et le développement d'outils d'analyse génériques, interopérables et réutilisables. Chronologiquement, il y a eu ICALTS en 2004, puis IA en 2005, et CAViCoLA sur les années 2006 et 2007 : tous se sont centrés sur l'analyse de l'interaction entre les acteurs d'un dispositif d'apprentissage collaboratif avec pour objectif :

1. de mieux définir le champ de l'analyse de l'interaction afin de proposer un cadre terminologique et méthodologique à ce champ et de permettre ainsi la comparaison des approches d'analyse de l'interaction (question abordée notamment dans le cadre du projet ICALTS) ;
2. de proposer des solutions pour permettre l'interopérabilité des outils et des méthodes d'analyse de l'interaction (question abordée notamment dans le cadre des projets ICALTS et AI) ;
3. d'étudier la problématique de l'adaptation dynamique des représentations des données analysées et de leur visualisation en fonction des centres d'intérêt de l'usager de ces données. (question abordée notamment dans le cadre du projet CAViCoLA).

Une des questions fondamentales traitées dans ces projets (plus en détail dans ICALTS [Mar05]) est illustrée dans la figure 2.2. Le constat relevé est que différents outils d'analyse très hétérogènes ont été réalisés dans le cadre des environnements supportant l'apprentissage collaboratif. En effet, chacun de ces outils a été créé par une équipe de recherche pour répondre à des questions spécifiques au sujet de leurs propres environnements d'apprentissage.

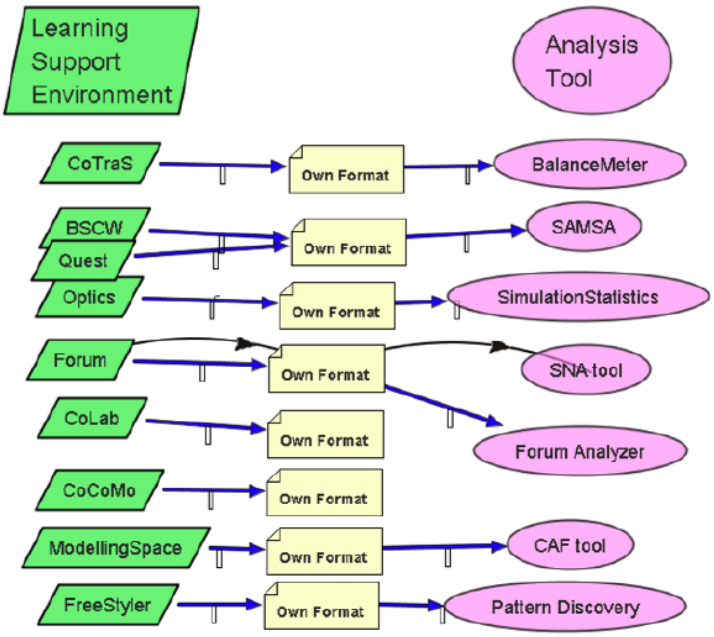

FIGURE 2.2 – La situation initiale : des outils d'analyse intégrés dans les EIAH (extrait de [Mar05])

L'inconvénient majeur de cette situation est que chaque environnement produisant des traces et les outils d'analyse correspondants sont liés à un format de fichier particulier. Ceci rend difficile la réutilisation de différents outils d'analyse sur des données provenant d'autres environnements.

L'un des objectifs de ces projets étant de créer une bibliothèque d'outils d'analyse d'interaction, un format commun a été défini pour permettre à ces outils de coopérer. Une partie de la DTD XML pour ce format commun est présenté dans la Figure 2.3. Ce format de données décrit l'interaction (une trace numérique) comme étant composée d'un préambule décrivant les entités qui peuvent être identifiées et trouvées dans la trace et un ensemble d'actions. Chacune de ces actions est caractérisée par certaines propriétés nécessaires et d'autres facultatives telles que le temps d'occurrence, le type d'action, l'utilisateur impliqué et son rôle, le contenu décrivant l'action et les objets sur lesquels l'action s'applique. Le but de ce format est de servir d'intermédiaire entre les outils produisant des traces et outils qui les analysent. Deux cas de figures se posent alors, soit les environnements tracés produisent directement un format qui peut être lu directement par les outils d'analyse, soit ils conservent leur propre format en tentant de se rapprocher au mieux du format commun moyennant des transformations XSLT.

Bien que l'outil ait été pensé pour des analyses *ad-hoc* et *post-hoc*, les usages soutenus et les exemples proposés sont dans des temporalités loin d'être temps réel (e.g. en de fin session, ou en fin de tâche, etc.). Du point de vue de l'analyse *post-hoc*, ce format a été proposé comme un moyen pour l'interopérabilité entre les outils d'analyse.

Le troisième exemple de représentation de trace pour assister des études empiriques nous

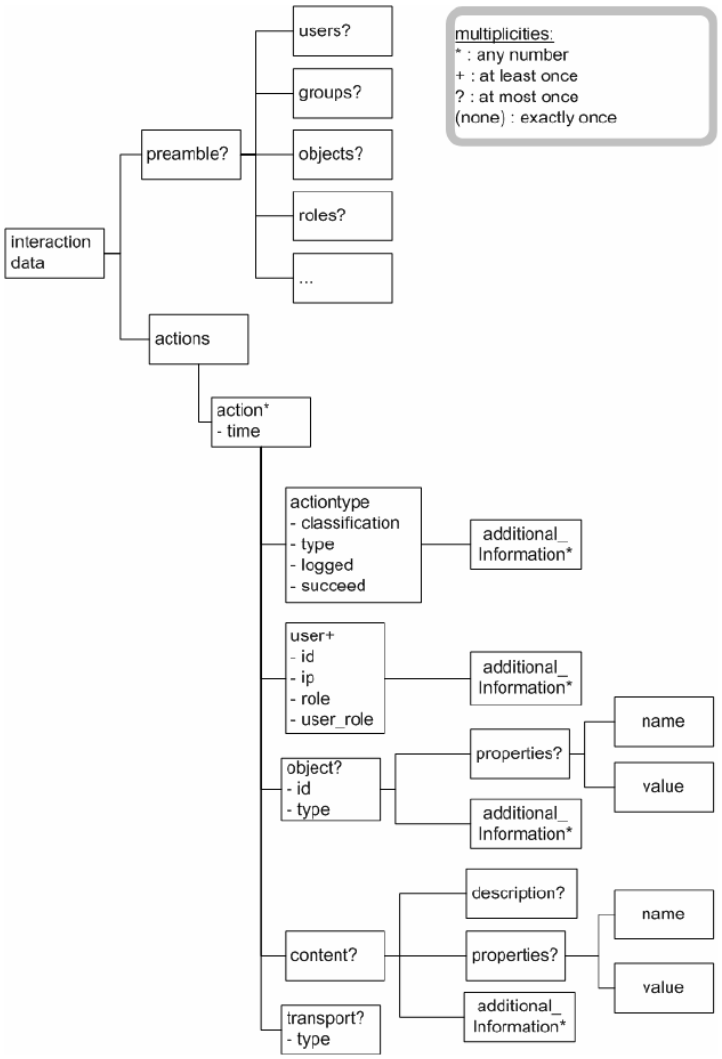

FIGURE 2.3 – Une partie de la DTD du format commun pour les traces défini dans CAViCoLA (extrait de [Mar05])

vient du projet MULCE (MUltimodal contextualized Learner Corpus Exchange). Le projet MULCE est justifié par la nécessité au sein de la communauté TEL de partager des corpus et des analyses, afin de permettre la vérification, la validation et le raffinement de résultats de recherche [RCNB08]. Les auteurs précisent les exigences relatives à la construction d'un *corpus d'apprentissage*, en expliquant ce qu'il doit contenir, la façon dont il peut être structuré et dans quelles conditions il peut être partagé. Les auteurs ont commencé d'abord par définir la notion de corpus d'apprentissage, en s'inspirant des corpus linguistiques, comme étant constitué des données d'interaction (traces numériques, vidéo, pré-et post-tests), mais aussi du contexte d'apprentissage (connaissances des apprenants actuels, les buts, les scénarios pédagogiques, les outils utilisés, etc.), le contexte de la recherche (questions de recherche, protocole expérimental, etc.), la licence sous laquelle est distribué le corpus (décrivant les conditions dans lesquelles il

peut être utilisé), et tous les travaux d'analyse disponibles (tels que la transcription de vidéos).

Reffay et ses collègues ont proposé une structure XML permettant de définir un corpus, où chaque composant est divisé en trois niveaux : la description du composant, la référence au fichier contenant les données relatives à ce composant et le fichier de données lui-même (pas inclus dans le fichier XML, mais disponible à l'URL référencée). En ce qui concerne les données contenues dans le corpus, ils recommandent le respect des normes existantes (lorsqu'elles existent), comme le Text Encoding Initiative [13] pour le texte, IMS-LD [Con03] pour les scénarios pédagogiques, etc.

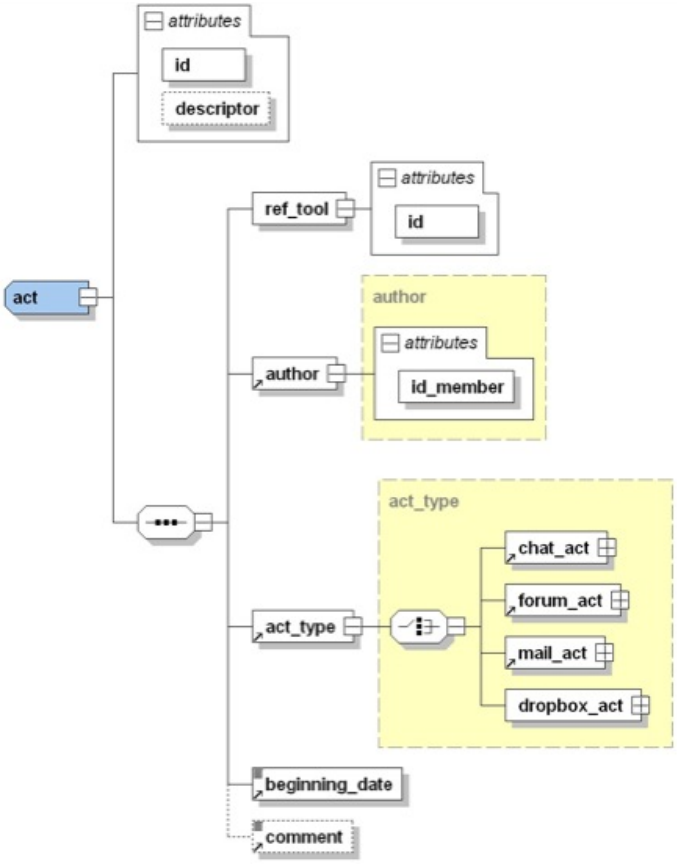

FIGURE 2.4 – Extrait du schéma XSD représentant les actes pour les traces dans MULCE [RCNB08])

En raison de l'absence de format commun pour les traces numériques, [RCNB08] proposent un modèle pour leurs propres types de données, où une session est composée d'actes et un acte est défini de façon générique, par un autre type de données représentant les informations spécifiques pour chaque type d'acte (email, forum, etc.). Ce format permet à d'autres utilisateurs de l'étendre aux actes qui peuvent être effectués dans leurs propres outils (cf. Figure 2.4). Ce travail a conduit à la création d'une plate-forme permettant la mise en ligne de corpus ainsi que le partage des analyses effectuées sur ces corpus.

13. http ://www.tei-c.org/index.xml

Ce projet adopte une approche très pragmatique pour la modélisation de corpus regroupant des traces, des informations contextuelles et des analyses. Dans chaque document, il devrait y avoir une description qui explique ce qu'il contient, dans quel format il est formalisé et le logiciel pouvant être utilisé pour le lire. Dans la mesure du possible, les auteurs conseillent l'utilisation de formats standards ouverts. Une telle approche pose de bonnes bases en termes de simplification de l'effort et du coût du partage d'un corpus (e.g. il n'est pas nécessaire de le convertir en un format commun, ou d'utiliser un ensemble restreint d'outils). Cependant, elle fournit peu d'informations sur la façon dont on peut manipuler ces corpus en terme d'opérations, pour mettre en œuvre des analyses. Ceci dépend bien sûr des formats standards et des langages associés.

Traitements et processus pour l'exploitation des traces

Après avoir présenté les modèles et les représentations de traces pour les trois exemples (Tatiana, les projets européens (ICALTZ, IA et CAViCoLA) et MULCE), nous nous intéressons dans cette partie aux traitements et manipulations mises en place pour l'exploitation des traces.

En termes d'élaboration et de définitions, Tatiana est sans doute le système le plus abouti pour l'analyse des traces et surtout pour leur rejouage et visualisation. Les traces, considérées comme des objets rejouables, disposent de plusieurs opérations pour leurs manipulations. Ces opérations sont présentées comme des spécialisations d'un opérateur de transformation permettant à partir d'autres objets rejouables existants de construire de nouveaux objets rejouables.

[Dyk09] définit une transformation comme une fonction dont le domaine est l'ensemble des rejouables qui peuvent être transformés et dont le co-domaine est l'ensemble des rejouables qu'elle produit. Cette représentation en fonction sert surtout à illustrer certaines propriétés que des transformations confèrent à leurs rejouables sources et résultats. Avec cette représentation, [Dyk09] définit des opérations de transformations génériques comme la transcription, segmentation, identification, catégorisation, filtrage, verbalisation, recherche, pliage, fusion et groupage. Dans Tatiana, les transformations (la création de nouveaux rejouables) peuvent être effectuées automatiquement ou manuellement (mais ces deux mécanismes ne peuvent être combinés pour un seul rejouable). Elles peuvent être appliquées sur la trace ou un objet rejouable (i.e., la trace plus les représentations des informations permettant le rejouage). Le mécanisme de transformation de Tatiana automatisé peut être appliqué indifféremment à des traces XML et aux rejouables dans le format Tatiana (c'est à l'analyste de veiller à ce que la transformation donnée soit jouable).

Le second exemple concerne les traitements proposés dans les projets ICALTS, IA et CAViCola. Pour le projet ICALTS, les analyses de traces d'interactions sont guidées par deux aspects importants : d'une part l'énoncé d'hypothèses sur l'interaction et d'autre part la volonté de prouver ou de rejeter ces hypothèses par l'observation. Un des premiers résultats de ce projet concerne l'étude du concept d'*indicateur d'analyse d'interactions* comme étant la réponse aux questions d'ordre pédagogique et/ou psychologique, e.g.,« quelles sont les questions importantes

à poser au dispositif d'analyse ? » et « que veut-on analyser ? » [HMMD08].

D'un point de vue computationnel, un indicateur est défini comme une variable généralement calculée ou établie à l'aide de traces, témoignant du mode, du processus ou de la qualité de l'interaction [Dim04]. Le questionnement et la formulation d'hypothèses sur l'interaction amènent à la définition d'indicateurs pédagogiques, qui eux-mêmes conduisent à la définition d'un ensemble d'indicateurs computationnels. Chacun de ces indicateurs détermine ce qui doit être capturé comme données dans le dispositif d'apprentissage pour les établir. À ce stade, l'EIAH considéré produit un ensemble de traces brutes dans un format spécial conditionnant de fait la possibilité d'établir ou non l'indicateur. Les traces brutes sont traitées et exploitées par une méthode d'analyse chargée d'établir l'indicateur qui pourra être transmis à un outil (l'EIAH lui-même ou un outil spécifique) chargé par exemple de sa visualisation.

L'utilisation concrète de l'indicateur peut aussi répondre à une stratégie (par exemple le guidage de l'apprenant). Si cette stratégie existe, elle est influencée par la méthode d'analyse – cette dernière conditionne les formats de sortie des indicateurs –, mais elle influence également le choix de l'outil exploitant l'indicateur. Le projet CAViCoLA s'est basé sur cette considération pour proposer une modélisation du processus d'analyse de l'interaction dans les EIAH collaboratifs. CAViCoLA détaille le flot des données traités et analysés depuis l'EIAH source jusqu'à l'outil d'exploitation. La figure 2.5 présente une représentation graphique de ce processus, extraite de [HMMD08]. La partie gauche de la figure 2.5 présente la séquence générique de traitement des données. La partie droite de la figure 2.5 présente le processus d'analyse proposé par CAViCoLA qui reprend à son compte l'idée d'une combinaison des résultats dans une approche multi-méthodes : des méthodes quantitatives, des méthodes qualitatives, et un schéma de classification proposé par Meier & al. [MSR07] pour évaluer la qualité du processus de collaboration.

Toutes ces méthodes d'analyse suivent la séquence générique de traitement des données présentée en partie gauche de la figure 2.5 :

- collecter les données brutes ;
- segmenter ces données, i.e., les transformer de manière à isoler les variables signifiantes ;
- pré-traiter les données segmentées par annotations qualitatives ou mesures quantitatives ;
- analyser les données suivant la méthode considérée ;
- visualiser ces données analysées ;
- interpréter les données.

Pour le troisième exemple, le projet MULCE, bien que pragmatique sur les aspects modélisation et représentation des traces, ne précise pas les traitements et les analyses à faire sur les traces. Les modèles représentés dans MULCE semblent s'orienter vers des analyses faites avec des outils externes (si possible standards). Ces analyses peuvent être décrites à différents niveaux où chaque niveau réifie l'état des données à une certaine étape. Ceci permet d'identifier les liens de dépendance entre les niveaux et de gérer une série d'analyses successives pouvant être appliquées par différents chercheurs sur un même corpus. Cependant, à long terme, la plateforme MULCE vise à intégrer des outils standard de traitement ou de fouille des données pour

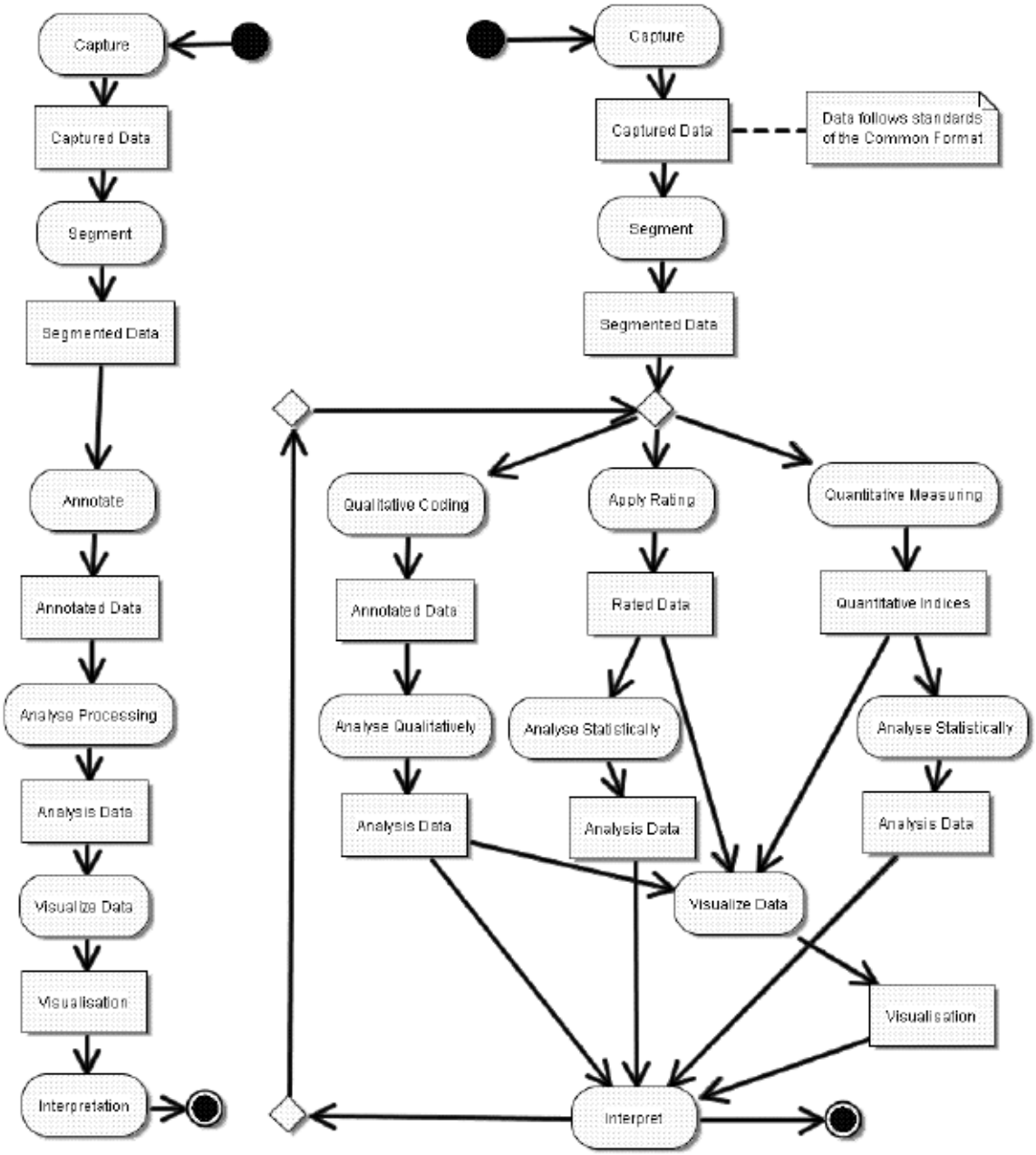

FIGURE 2.5 – Le processus d'analyse des interactions proposé dans le projet CAViCoLA.

permettre des analyses intra- et inter-corpus.

Discussion

Concernant le système Tatiana, le modèle proposé permet de représenter les traces pour un analyste disposant de plusieurs sources. Ce modèle permet une certaine extensibilité moyennant les relations et les facettes. Les usages de Tatiana et leurs applications sur plusieurs études de cas (voir [Dyk09]) démontrent le concept et font de Tatiana un candidat idéal pour mettre en place des analyses génériques. Cependant, les objets rejouables considérés semblent être plus des entités destinées à être visualisées et synchronisées avec d'autres sources qu'a être interrogées et transformées. Ceci est d'autant plus manifeste lorsqu'on considère des analyses menées par un agent artificiel, i.e., un système informatique disposant de certaines connaissances et contraintes qu'il doit satisfaire et interpréter pendant l'observation. Ceci est dû au fait que Tatiana est orienté vers des analyses hors ligne faites par des agents humains. Même si certaines analyses peuvent être appliquées (par un agent informatique) pendant l'observation, XQuery, le langage utilisé dans Tatiana, ne permet pas la prise en charge native de traitements continus en temps quasi réel et doit être couplé avec un langage de programmation général (i.e, le langage de programmation hôte de l'environnement d'analyse). Dyke [Dyk09] pointe d'ailleurs certaines de ces limites en projetant de s'inspirer des langages de représentation de connaissances (comme les frames [Min74]) pour permettre un meilleur typage et des mécanismes d'inférence sophistiqués.

Concernant les projets européens ICALTS, IA et CAViCoLA, l'hypothèse de base de cette approche est qu'une certaine sémantique unifiée des traces des actions de collaboration peut être atteinte. Ceci permettra la construction d'un outil effectuant plusieurs analyses, moyennant des traitements et des calculs automatisés. Cependant cette sémantique unifiée atteint rapidement ses limites lorsqu'on s'intéresse à des traces regroupant les activités individuelles et collaboratives des utilisateurs.

Afin d'offrir une flexibilité supplémentaire, Martinez [Mar05] fait remarquer que la description des objets de la trace peut être enrichie au besoin par des propriétés, i.e., des couples attribut-valeur associés aux objets. Ceci fournit un mécanisme très souple pour la modélisation. Toutefois, il introduit aussi un manque de rigueur et de précision dans la représentation, notamment lorsqu'il est nécessaire de vérifier d'une manière générale, si tous les éléments nécessaires pour des analyses spécifiques sont disponibles. En effet, la solution pour les formats qui ne correspondent pas complètement au format commun est que les éléments spécifiques soient considérés comme optionnels. Cependant, les outils qui doivent utiliser cette partie optionnelle ont alors plus de difficultés à expliquer leurs besoins et à exiger les informations devant être fournies.

Pour opérationnaliser cette approche multi-méthodes d'analyse, il est nécessaire que les EIAH sources et les outils d'analyse cibles communiquent entre eux moyennant le format commun. Cette communication est assurée par l'application de transformations XSL sur le format de sortie de l'EIAH et le format d'entrée de l'outil d'analyse pour passer de l'EIAH vers le format commun (i.e., EIAH $\rightarrow$ XSL-T $\rightarrow$ DTD du format unifié) et du format commun vers l'outil

d'analyse (DTD du format unifié → XSL-T → outil d'analyse) (cf figure 2.6).

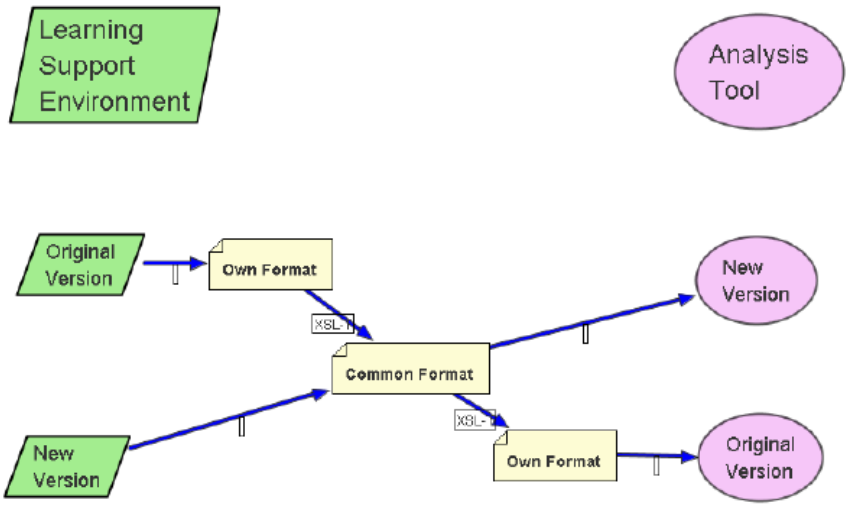

FIGURE 2.6 – Les différents modes d'usage du format commun avec un EIAH et un outil d'analyse

Bien que le projet CAViCoLA ait signalé une utilisation réussie de ce format commun sur plusieurs outils, le format n'est pas actuellement d'un usage répandu. La raison principale à notre sens est que le problème se situe au niveau sémantique (i.e, ontologie des éléments à représenter) et non au niveau des formats. Bien que cette approche ait été conçue pour être utilisée dans des situations de collaboration assistée par ordinateur, à la fois en temps réel (c'est à dire fournir une rétroaction immédiate aux apprenants et enseignants) et dans les analyses *post-hoc*, les technologies choisies (XML, XSLT) ne se prêtent pas facilement aux exigences de traitements quasi-temps réel de certains EIAH et de certaines personnalisations. Ces aspects de traitements doivent être pris en charge par un langage de programmation tiers ce qui nuit grandement à l'extensibilité et la réutilisabilité des traitements sur les traces.

Nous avons fait le tour de certains outils permettant des analyses empiriques *post-hos*. Dans la section suivante, nous allons présenter les usage de traces à des fins de réingénierie de l'EIAH tracé. Certaines analyses *post-hoc* peuvent être menées dans un objectif de réingénierie. Cependant, les analyses considérées dans la section suivante sont dans un cycle de réingénierie plus court en temps que les exemples présentés puisqu'elles peuvent être menées peu de temps après l'observation, e.g., analyser une séance d'apprentissage en fin de session pour améliorer le scénario pédagogique pour la prochaine session d'apprentissage.

2.3.2 Réingénierie des EIAH

Une des visées de l'analyse de traces d'usage d'un EIAH est sa ré-ingénierie [Cho07]. En analysant les traces, un concepteur peut par exemple proposer la conception d'une nouvelle version à partir de la version déjà existante ou bien améliorer le contexte dans lequel l'EIAH sera utilisé. Comme l'a argumenté Choquet [Cho07], la ré-ingénierie d'un EIAH à des fins d'amélioration de la qualité pédagogique se pratique en se basant sur l'observation des caractéristiques

des interactions humaines, les caractéristiques du scénario pédagogique et les objectifs à atteindre. L'observation des indicateurs est soit préalablement défini — souvent en collaboration avec l'enseignant — soit realisée à la volée en cas de nouveaux besoins, lors du déroulement d'un scénario pédagogique. Il s'agit de tenir compte des écarts observés entre le scénario tel qu'il a été conçu pour fonctionner (scénario prédictif) et tel qu'il s'est réellement déroulé (scénario descriptif), l'idée étant d'identifier comment la ré-ingénierie de tel ou tel aspect pourrait faire mieux atteindre les objectifs pédagogiques. Choquet [Cho07] fait remarquer aussi que *«si la majorité des EIAH ont adopté une approche de développement par prototypage, permettant d'évaluer les usages du dispositif pédagogique avant d'envisager la construction d'un nouveau prototype, l'adoption d'un processus de développement intégrant de manière systématique cette analyse et donc l'explicitation et la justification des observations à mener, reste très marginale».*

Marty et ses collègues font le même constat et mettent l'observation à base de traces au centre de la ré-ingénierie du scénario pédagogique. Ils affirment aussi que l'observation à base de traces est un important facteur pour la qualité du scénario d'apprentissage [MHCF04]. En effet, les traces fournissent de nombreuses informations utiles à l'analyse d'une séance d'apprentissage, comme par exemple la compréhension de la réussite ou de l'échec des apprenants, mais aussi la reconception du scénario pédagogique. L'approche proposée dans [MHFC07] se base sur plusieurs traces, collectées à différents niveaux d'abstraction, pour permettre de donner un vue globale et détaillée de l'activité de l'apprenant.

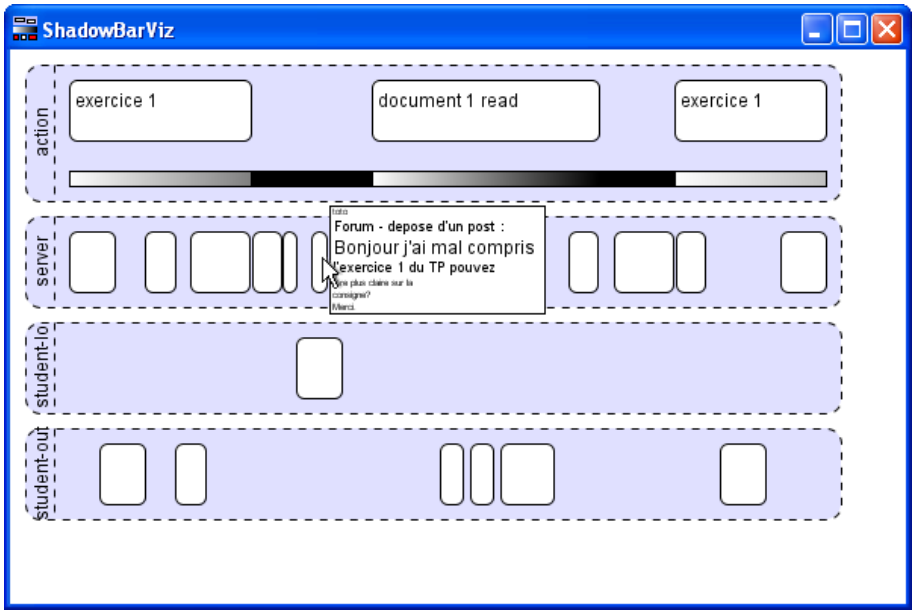

FIGURE 2.7 – Outil de visualisation de la barre d'ombre et d'aide à la composition de traces [MHFC07].

Marty et ses collègues proposent ainsi une visualisation de plusieurs traces baptisée « la barre d'ombre » (figure 2.7) qui permet à un enseignant d'être conscient de l'efficacité de son scénario pédagogique en fonction du temps pris par les apprenants pour faire les différentes

tâches. La barre d'ombre permet de représenter graphiquement l'avancement de l'apprentissage. La couleur de chaque zone est déterminée par l'estimation de la clarté : claire pour les zones dans lesquelles l'activité de l'apprenant a été identifiée, foncée dans le cas où il s'est éloigné de son activité et donc du scénario pédagogique. Le processus d'interprétation est défini de manière détaillée dans [MHFC07], il combine plusieurs sources de données : des traces issues de l'exécution du scénario pédagogique, des traces du serveur web de la plateforme d'apprentissage, des traces d'un keylogger installé sur les machines des apprenants et des traces des outils de communication (Webmail, forums).

Cette visualisation permet à l'enseignant d'améliorer le scénario mis en place mais aussi de contrôler et de réguler l'activité d'apprentissage [HMFC05, MHFC07]. L'analyse de traces d'interactions permettra ainsi à l'enseignant de proposer des façons d'agir en temps réel lors des prochains scénarios – autrement dit, de proposer des actions à mener par le système informatique qui pourraient être déclenchées à la découverte d'un élément observable décrit et spécifié par l'enseignant mais calculé de manière automatique par un outil d'analyse.

Modèles et représentations considérées pour les traces

Un des projets mettant explicitement au centre de la réingénierie des EIAH la problématique de l'exploitation des traces est le projet ReDim (Reengineering Driven By Models). Dans ce cadre, Choquet et ses collègues [CI07] estiment que les traces peuvent être considérées comme des objets pédagogiques, au même titre que les scénarios ou les ressources pédagogiques.

Le projet de recherche REDiM considère l'enseignant comme le principal acteur de la conception et de la réingénierie d'un EIAH mais confère le traitement des traces d'utilisation à trois rôles différents : le concepteur (l'enseignant), l'analyste et le développeur. L'enseignant, qui est le mieux qualifié d'un point de vue pédagogique pour spécifier ce qu'il faut observer dans une session d'apprentissage, doit disposer des outils pour appréhender, dans son univers métier, les résultats de cette observation. Pour assister l'enseignant dans sa démarche de ré-ingénierie, Choquet et ses collègues ont défini un méta-langage de modélisation et de construction de traces baptisé UTL (Usage Tracking Langage)[CI07].

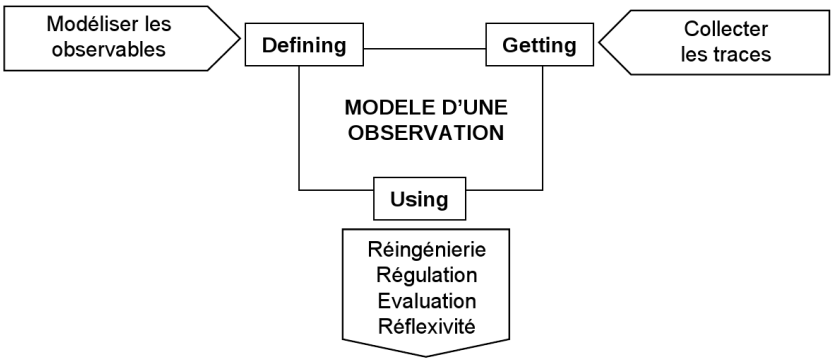

FIGURE 2.8 – Le Modèle du langage UTL [Cho07]

L'idée générale derrière UTL est que chaque trace d'utilisation prise en considération lors de la modélisation de l'observation doit concourir à témoigner d'un usage observé, et que ces usages observés sont à mettre en relation avec le scénario pédagogique prédictif afin de pouvoir les analyser [CI07]. UTL décrit sous trois facettes les besoins (cf. figure 2.8), la manière de collecter et d'utiliser les traces : la première facette «Defining» permet d'assister notamment le concepteur/pédagogue dans la phase de définition des observables dont il aura besoin, la deuxième facette «Getting» permet la description des moyens d'acquisition et de collecte des traces et enfin la troisième facette «Using» permet de définir l'utilisation des traces (ré-ingénierie, régulation de l'activité). De ce point de vue, la démarche présente certains avantages. Elle permet tout d'abord de décrire des observables à partir de la structure du scénario construit par l'équipe des concepteurs. Elle propose ensuite une façon de retranscrire ces observables dans un langage interprétable en conception, dans le but de pouvoir comparer le scénario conçu *a priori* à celui résultant des usages observés *a posteriori*.

Le méta-language UTL, dans sa première version, ne considère que des traces collectées de manière automatique. Il est centré sur l'importation des traces dans un format indépendant du dispositif d'apprentissage. Pour instrumenter la modélisation et l'analyse de l'observation dans une perspective de capitalisation, Choquet et ses collègues ont défini une nouvelle version du méta-langage UTL [CI06, CI07] permettant la description des méthodes d'analyse de l'observation en vue de leur capitalisation en étendant UTL avec un ensemble d'éléments dédiés à cette description. Cette nouvelle version permet par contre une description structurée des indicateurs dans une forme indépendante des formats de traces générées par un dispositif d'apprentissage et du langage de modélisation pédagogique employé pour décrire le scénario pédagogique.

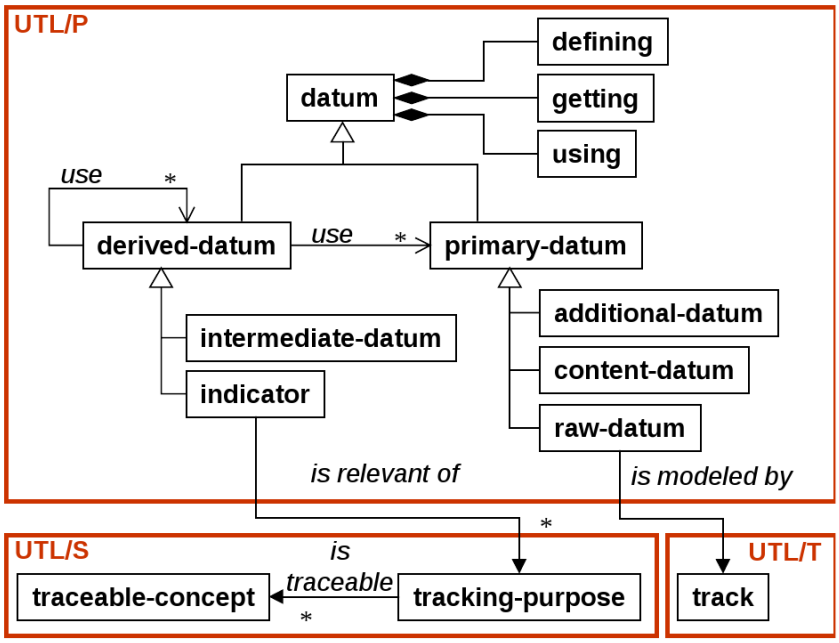

FIGURE 2.9 – Le Modèle du langage UTL2 [Cho07]

Dans UTL-2 (figure 2.9), différents types de données peuvent être modélisés dans la trace. [CI07] identifie deux types principaux de données impliqués dans l'analyse de l'observation : la donnée dérivée (derived datum) et la donnée primaire (primary datum). Les données primaires ne sont pas calculées ou établies avec l'aide d'autres données et peuvent être issues d'un fichier log ou d'une base de données. Une donnée dérivée est une sorte de production de l'apprenant. Les données additionnelles (additional-datum) sont des données qui ne sont pas collectées à partir de l'activité de l'apprenant mais qui pourraient être utiles pour l'analyse. Une donnée dérivée est calculée à partir des autres données et peut être soit une donnée intermédiaire (utilisée temporairement pour calculer quelque chose d'autre) ou un indicateur : une information définie comme ayant une signification spéciale dans un scénario pédagogique spécifique ou situation d'apprentissage.

Un autre exemple de projet considérant les traces pour la ré-ingenierie des EIAH est le projet TRAILS (Personalized and Collaborative Trails of Digital and Non-Digital Learning Objects) [TRA04]. Ce projet s'est centré sur la caractérisation des traces d'utilisation dans un environnement de formation à distance sur Internet. Les membres de ce projet utilisent le terme de « parcours » (« trail ») plutôt que celui de « trace » puisque leurs travaux ne considèrent que les traces des parcours. Deux types de parcours sont identifiés : les parcours qui sont générés lors de la planification d'une activité d'apprentissage (par exemple, un scénario pédagogique) et les parcours qui émergent pendant une session (i.e., la trace d'utilisation). Le projet fait le constat que les traces de type parcours dépendant de plusieurs paramètres :

- la nature de la situation pédagogique ;
- les possibilités d'interaction et du contexte d'apprentissage ;
- les objectifs d'utilisation des parcours (évaluation des apprentissages, régulation de la session d'apprentissage, réflexivité sur l'activité, ré-ingénierie).

Partant de ce constat, le projet TRAILS a classé les types de traces dans une taxonomie décrites par la figure 2.10. La classification des traces de type parcours d'apprentissage présentée par la figure 2.10 est décrite plus en détail dans [SLH+07].

Les types en blanc caractérisent des parcours existants avant l'activité d'apprentissage (typiquement les traces générées automatiquement à partir des scénarios pédagogiques préconisés). Les types en noir ne s'appliquent qu'aux parcours construits pendant l'activité d'apprentissage (typiquement les traces observées de l'exécution des scénarios définis), et les concepts en gris caractérisent ces deux types de parcours. Ceci montre que le projet TRAILS considérait sur une même échelle les traces des parcours élaborés dans l'optique d'organiser la session d'apprentissage et les traces des parcours construites par l'utilisation et l'interaction avec le dispositif d'apprentissage.

Dans le contexte de la scénarisation pédagogique, cette classification des types de traces permet de reconstruire les scénarios descriptifs de l'activité menée pendant une session d'apprentissage. En plus de la ré-ingénierie du scénario pédagogique, le projet TRAILS revendique aussi que l'apprentissage à base de traces de type parcours peut se décliner sur plusieurs points de vue (régulation par le tutorat, adaptation automatique, apprentissage réflexif, etc.).

FIGURE 2.10 – Classification des types de traces décrivant des parcours d'apprentissage proposée par le projet TRAILS [SLH+07].

Un autre projet LISTEN (Literacy Innovation that Speech Technology ENables) [LIS07, MA01] donne de bonnes indications pour la ré-ingénierie des EIAH par les traces d'apprentissage. Ce projet, au delà de la qualité de ses résultats, est un excellent exemple de par son déroulement et l'évolution de la collecte et l'analyse des traces. Dès ses débuts, ce projet s'est structuré autour du développement et la ré-ingénierie d'un Tutoriel Intelligent nommé « Reading Tutor » destiné à l'apprentissage de la lecture avec le principe suivant : le système écoute l'apprenant lire à haute voix et permet d'interagir avec lui afin d'améliorer sa lecture, sa compréhension moyennant des algorithmes de reconnaissance vocale [BJM04].

L'usage des traces dans ce projet ne se limitait pas seulement à l'évaluation des connaissances de l'apprenant, exploité de manière indirecte, pour la réingénierie mais servait également à analyser statistiquement de vastes ensembles de données collectées pendant plusieurs sessions d'apprentissage [MA01]. Pour la représentation des traces, les chercheurs du projet LISTEN ont créé des bases de données relationnelles spécifiques pour stocker les données collectées pendant les sessions. Ils ont également stocké les connaissances et ressources manipulées par le tuteur

dans d'autres bases de données, liées avec les bases de collectes, de manière à faciliter l'analyse et la compréhension des traces. En ce qui concerne la modélisation des traces, les chercheurs de LISTEN ont identifié plusieurs dimensions pour définir le schéma des données. Puisque le dispositif d'apprentissage et les objectifs évoluent avec le temps (e.g. application d'une nouvelle technique de fouille), les chercheurs ont donc défini différents schémas de base de données en fonction de la version du système d'apprentissage (Reading Tutor) et des objectifs d'analyses. En outre, certaines des tables de la base de données contiennent des millions d'enregistrements. Les chercheurs ont alors privilégié la redondance des données à la normalisation du modèle conceptuel de la base, de manière à réduire le temps d'exécution des requêtes qui augmente considérablement si la requête porte sur plusieurs tables.

Traitements et processus pour l'exploitation des traces

Cette section montre les approches mises en place pour l'exploitation des traces pour les exemples que nous venons de détailler (UTL, TRAILS et LISTEN).

Les traitements effectués sur les traces décrites en UTL sont essentiellement des calculs d'indicateurs qui ne sont pas décrits dans le langage de base. Pour y remédier, une extension pour UTL appelée DCL4UTL (Data Combination Language For UTL) a été définie [NICK09]. Ce langage, inspiré des langages de requêtes classiques (SQL, Xquery), a comme objectif de combiner les données pour établir une nouvelle donnée ainsi que de capitaliser les méthodes de combinaison pour les réutiliser. DCL4UTL comporte deux composantes principales : l'opérateur et l'opérande. L'opérande correspond aux données dans la signification d'UTL. Il est l'un des éléments de la facette « Using » d'UTL. L'opérateur peut être un symbole ou une fonction. Un symbole est un opérateur arithmétique $(+, -, *, /)$, un opérateur relationnel $(>, <, >=, <=, ==, ! =)$ ou un opérateur logique (et, ou). Le langage propose également un ensemble de fonctions : les fonctions statistiques (`sum`, `min`, etc.), la fonction de chaîne de caractère (`concat`), la fonction conditionnelle (`compare`), la fonction de filtrage et de tri (`filter`, `sort`), la fonction de position (`first`, `end`, `index`, etc.), etc. En outre, DCL4UTL dispose aussi également d'une boucle pour parcourir et calculer les données dérivées qui ont plusieurs instances (`for-each`).

Bien que la grammaire du langage semble proposer des opérateurs intéressants, elle n'est pas encore publiée et donc pas disponible. Une simple version de la syntaxe du langage est définie sous la forme : `cal` <Elements [`as` <Conditions>] > `where` <Data> où :

- <Elements> représente les éléments de la facette «Using» à calculer.
- <Conditions> représente les conditions à satisfaire avant le calcul.
- <Data> représente les données nécessaires pour le calcul.

Bien qu'une implémentation de ce langage ait été présentée dans [Ngo10] pour l'EIAH Hop3x, il n'est pas possible pour le moment de distinguer les propriétés et les opérateurs qui ont été opérationnalisées. Il est aussi difficile d'utiliser et d'étudier ce langage puisqu'on dispose pas de sa grammaire complète. À notre connaissance, aucune sémantique formelle n'a été définie, ce qui rend sa comparaison avec les langages existants (comme SQL et XQuery) difficile. Enfin,

DCL4UTL dispose d'un mécanisme d'appel de fonctions externes (définies dans avec le langage Java) similaire aux fonctions définies de SQL. Les opérateurs et les fonctions définies pouvant être non monotones, leurs évaluations en temps réel de l'activité se fait toujours sur toutes les données disponibles dans la trace. Ceci peut être très pénalisant en terme de performance lors du calcul de plusieurs indicateurs sur plusieurs traces volumineuses et denses.

Au delà de simples calculs d'indicateurs, et en matière d'analyse des traces d'usage pour améliorer et re-conceptualiser les logiciels EIAH, ce sont les techniques de fouille de données appliquées aux EIAH qui ont fait l'objet d'études les plus élaborées. La communauté EDM (« Educational Data Mining[14] ») s'est bien structurée ces dernières années notamment en termes de développement de méthodes de fouille de données et leurs applications aux EIAH afin d'en examiner les utilisations possibles.

Dans le cadre du projet TRAILS, des techniques de data mining ont été utilisées pour décrire les parcours des apprenants sur l'EIAH basé sur le web. L'exploitation des traces des apprenants du point de vue de l'environnement d'apprentissage est faite par la construction d'un modèle de Markov [HLK+04]. La topologie du modèle de Markov est déterminée par des relations entre les objets d'apprentissage de l'environnement d'apprentissage. Les probabilités de transition entre les relations sont déterminées par les statistiques d'utilisation déduites à partir des fichiers log des sessions de navigation des utilisateurs. Le calcul de statistique se fait par des techniques de fouille de données issues du Web (Web Usage Mining) appliquées sur les traces log du serveur Web [HLK+04].

Cependant, le projet LISTEN a été un des premiers à utiliser les techniques de data mining sur les traces collectées du tutoriel intelligent Reading Tutor. Du point de vue des processus mis en place pour l'exploitation des traces à des fins de data mining, les chercheurs du projet LISTEN ont identifié un ensemble de bonnes pratiques pour s'assurer de la validité des traces collectées. Ceci a amené Mostow et ses collègues à abstraire leur savoir-faire en matière d'analyse de traces en définissant le processus «*Modify, Map, Mine*» [MBC+02, HBM04, Mos04]. Ce processus est en fait un ensemble de recommandations méthodologiques pour l'utilisation, par un système d'analyse de traces, des données collectées par un système tutoriel intelligent.

La premier aspect («Modify») porte sur l'instrumentation du tuteur intelligent et la manière de concevoir ou ré-instrumenter un EIAH de manière à le rendre capable de collecter le maximum d'informations signifiantes. Mostow et collègues insistent sur trois points essentiels :

– Mettre dans la trace les événements qui relèvent des productions de l'apprenant («inputs») et des productions du tuteur ainsi que les intentions et les objectifs ayant amené le tuteur à faire ses productions. Les auteurs recommandent également d'envisager différents types d'analyses pour caractériser les éléments de la trace, en donnant l'exemple d'un clic souris qui, s'il est représenté par ses coordonnées spatiales, peut donner des indications sur la motricité de l'apprenant, mais qui, s'il est représenté par référence à l'élément de l'IHM, donne des informations sur la nature de l'interaction.

14. http ://www.educationaldatamining.org/

- Situer temporellement les événements dans la trace de manière à pouvoir mener des analyses sur les précédences, les récurrences, les fréquences, etc. Réifier les opérations dans le tuteur intelligent de manière à les rendre analysables. En particulier, les auteurs recommandent d'instrumenter aussi l'environnement externe (au tutoriel) pour que toute l'activité de l'apprenant puisse être tracée (i.e. développer un éditeur de texte ou un tableau blanc plutôt que de prévoir l'usage d'un brouillon papier par exemple).
- Inclure une étape de traitement manuel des données collectées pour préciser et compléter ces données.

Le second aspect («Map») concerne la transformation des événements collectés en variables signifiantes de façon à identifier les moyens d'abstraire et de reconstruire des informations signifiantes à partir du flux d'événements collectés. Mostow et ses collègues insistent à nouveau sur trois points essentiels :

- Identifier des points temporels significatifs dans la trace, permettant de segmenter le flux d'événements en séquences temporelles, identifiées par ces instants remarquables, comme par exemple le moment où l'apprenant donne une réponse erronée. Selon les auteurs, cette technique est particulièrement utile pour définir et modéliser un modèle comportemental de l'apprenant.
- Identifier dans le flux d'événements les séquences et enchaînements relatifs à une connaissance ou compétence particulière de l'apprenant ou du tuteur intelligent.
- Reformuler le flux d'événements en un ensemble de données plus descriptives des tâches et du contexte de l'apprenant, de manière à mesurer la qualité intrinsèque de l'EIAH en terme de processus d'apprentissage (i.e., mesurer son impact relatif à un contexte et un apprenant donné).

Le troisième aspect («Mine») est concerné par l'exploration de l'ensemble des données collectées dans la trace. Cette phase porte essentiellement sur les techniques de définition ou d'exploitation d'un modèle qualifiant l'apprenant ou le tuteur intelligent. Les points identifiés par les chercheurs du projet sont principalement de :

- Inspecter des échantillons d'interactions de manière à pouvoir assumer certaines hypothèses de recherche ;
- Grouper, agréger et composer un ensemble de données pour vérifier une hypothèse, un objectif de recherche spécifique ;
- Adapter un modèle d'analyse à un ensemble de données en déterminant les paramètres qui minimisent ses erreurs de prédiction ;
- Tester et entraîner un modèle d'analyse sur un sous-ensemble de données avant de l'exploiter sur l'ensemble des données collectées.

Discussion

La fouille et l'analyse de traces ont montré leurs limites dans le contexte de ré-ingénierie des EIAH. En effet, les projets LISTEN et TRAILS montrent bien que l'essentiel des résul-

tats existants sont difficilement partageables car développés de manière *ad-hoc*, pour un EIAH donné. Les évolutions récentes du positionnement des acteurs de ces projets, et plus largement de la communauté EDM, laissent espérer une capitalisation des pratiques. Le langage UTL et ses extensions se voulaient une avancée dans ce sens permettant la réutilisation de calculs d'indicateurs. Bien que la capitalisation des usages de traces ait été au centre des réflexions et des propositions autour du langage UTL, sa généralisation aux techniques de data mining est plus complexe.

L'expérience des chercheurs du projet LISTEN donne de très bonnes indications sur les besoins concernant les outils d'analyse de traces notamment le besoin de langages et de technologies qui simplifient cette tâche. En effet, dès sa première version, Reading Tutor a implanté des techniques de collecte de traces permettant d'une part au système de réguler les actions à entreprendre pour soutenir l'apprentissage et d'autre part aux concepteurs du système d'analyser la qualité des apprentissages et de corriger les erreurs de programmation du logiciel. Les traces collectées sous forme de fichiers logs (un fichier généré pour chaque session) n'étant pas une source de données facilement analysable manuellement, des techniques de fouille de traces automatiques ont été employées. L'application de ces techniques se faisait moyennant des scripts Perl, qui se sont très vite complexifiés pour différentes raisons : les expérimentations impliquaient des apprenants en plus grand nombre et des motivations d'analyse diverses, au point que les résultats établis dans le projet par ces scripts pouvaient être mis en doute [HBM04]. En plus, la réutilisation de ces scripts s'est avérée très complexe et parfois impossible. Les membres du projet LISTEN ont été alors amenés à repenser et revoir la collecte des traces dans Reading Tutor, mais aussi leurs stratégies d'analyse, ce qui les a conduit à modifier l'architecture logicielle de l'EIAH mais surtout à considérer des langages plus sophistiqués pour la manipulation des traces.

Si l'on considère en plus les traces obtenues à différents niveaux d'abstraction, le choix des langages et des modèles de représentation devient primordial. Les travaux sur la barre d'ombre [MHFC07] montrent également la complexité des traitements lorsqu'il s'agit de manipulations de plusieurs traces.

2.3.3 Adaptation des interactions en temps réel

Les deux types d'usage présentés dans les sections précédentes peuvent avoir tous deux comme objectif de personnaliser une interaction en l'adaptant en temps réel. Pour les exemples de travaux de recherche que nous présentons dans cette section, l'adaptation est quasi immédiate en temps réel et elle est l'objectif premier.

Modèles et représentations considérées pour les traces

Un premier exemple montrant l'utilisation des traces à des fins de réingénierie mais aussi pour de l'adaptation en temps réel de l'interaction sont les travaux de Luengo et collègues [LMAVMV07, MALV07] dans le cadre de l'EIAH TELEOS (Technology Enhanced learning Environment for Orthopaedic Surgery). Il s'agit d'un travail didactique portant sur l'analyse des

connaissances des apprenants en milieu hospitalier (chirurgie orthopédique) et sur un système d'enseignement et d'apprentissage correspondant. L'acteur produisant la trace est un apprenant agissant seul, qui s'entraîne à poser une vis dans le bassin d'un patient grâce à un simulateur. L'acteur manipulant la trace est l'EIAH et cette manipulation se fait en temps réel. Le type de manipulation effectuée porte sur la modélisation de connaissances de l'utilisateur en fonction des actions qu'il effectue à l'interface informatique du simulateur pendant la pose de la vis ainsi que sur la prise de décision didactique qui suit.

L'objectif précis est l'automatisation de la prise de décision didactique en utilisant les connaissances chirurgicales représentées par les réseaux bayésiens. En termes d'adaptation proposée, les auteurs préconisent trois types : de pointer vers les cours en ligne qui sont pertinents vis-à-vis du diagnostic du travail de l'apprenant ; de choisir un problème similaire et mettre le simulateur dans un état précis afin que l'apprenant puisse le prendre en main ; de choisir un cas clinique et le présenter à l'utilisateur. Ces adaptations sont proposées suite à la décision diagnostic et sont implémentées plus tard. Elles sont donc considérées comme relevant de la réingénierie et non pas de l'adaptation en temps réel.

Un autre exemple concerne l'outil Travis et les travaux May [MGP07, MGP08]. Ils proposent une architecture pour le suivi, l'analyse et la visualisation de la communication en ligne, et plus particulièrement dans les forums.

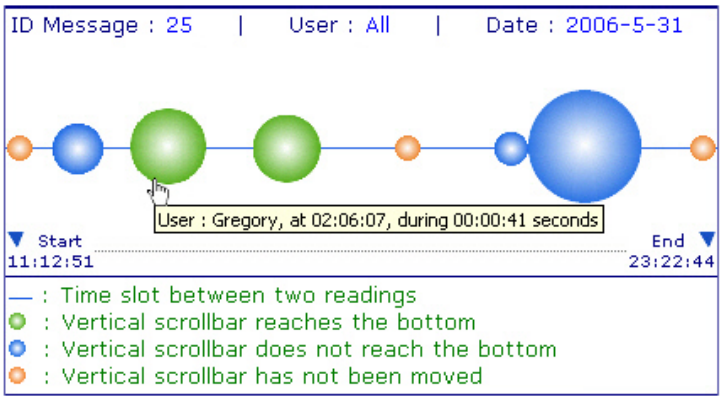

FIGURE 2.11 – Visualisation du temps possible de lecture passé par plusieurs lecteurs, sur un message donné. Chaque boule correspond à un lecteur ; la couleur représente l'activité de l'ascenseur de la fenêtre qui contient le message.

May, George et Prévôt [MGP07] expliquent que, dans le but d'analyser les usages d'un forum, il est important d'avoir des informations sur la façon dont un message est lu et écrit (le temps passé, si le message a été lu ou non, etc.). Ceci permet de générer des visualisations qui donnent des informations détaillées sur l'activité de lecture d'un utilisateur (cf. figure 2.11), permettant à celui-ci ou à un autre utilisateur (un chercheur ou un tuteur) de mieux comprendre si chaque message a été lu en détail ou si certains d'entre eux n'ont pas été lus complètement.

Bien que cette architecture soit applicable à tout outil de communications médiées par ordinateur, elle n'a été mise en œuvre que pour le suivi des forums et de leur analyse par les

apprenants et les tuteurs. L'outil développé Travis permet ainsi d'analyser les traces issues des outils de collaboration de type forum. L'acteur produisant la trace est l'apprenant lecteur. L'acteur manipulant la trace est l'EIAH. Le destinataire de la trace est multiple pour ces auteurs – l'apprenant, l'enseignant ou le chercheur. La manipulation de la trace est faite en temps réel et il s'agit entre autre de calculer de manière automatique le nombre de messages lus par personne ainsi que le temps qui aurait pu être passé pour la lecture d'un message.

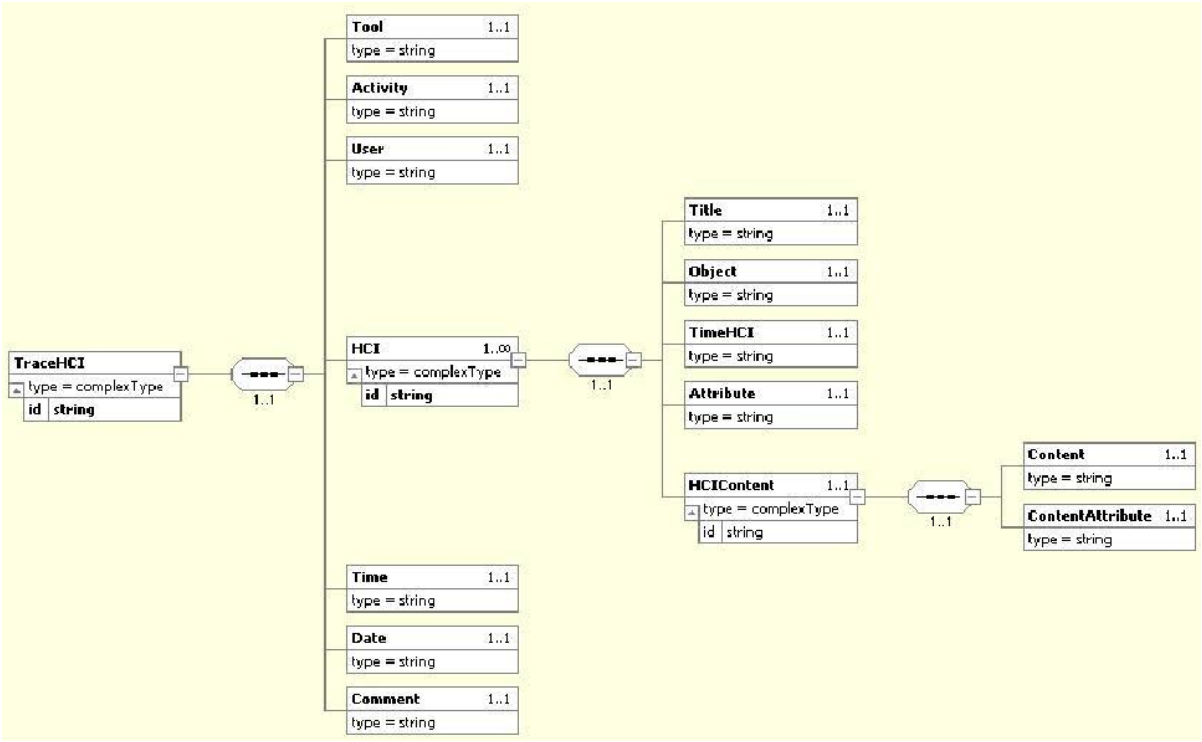

FIGURE 2.12 – Vue générale du modèle général de traces de communications médiées par ordinateur ([May09] page 110.)

Une des propositions des travaux de May concerne la définition d'un modèle général pour représenter les traces issues d'outils de communications médiatisées par ordinateur [May09]. Le modèle représenté sous forme d'un schéma XML (figure 2.12) décrit une trace par les actions et les interactions (<Activity> <HCI> <Title>) qu'un utilisateur (<User>) effectue sur des objets (<Tool> <Object>). Ces actions et interactions peuvent évidemment avoir des attributs (<Attribute>) et un contenu (<HCIContent> <Content> <ContentAttribute>) décrivant le résultat et les productions de l'activité. Enfin, la trace et ses éléments sont situés temporellement, avec les éléments <Date> <Time> pour l'activité et <TimeHCI> pour les actions et les interactions. À l'instar du projet CAViCOLa, May identifie deux cas de figures pour l'usage de ce modèle, soit l'outil de communication a été modifié ou conçu de manière à fournir des traces conformes au modèle général, soit les traces produites sont converties vers le modèle de trace moyennant des transformations XSLT.

Un autre exemple d'adaptation en temps réel des interactions de l'apprenant consiste à

l'assister dans son activité d'apprentissage. Cette approche a été mise en œuvre dans le cadre du système PIXED [Her02, HM02, HM03, HFM04]. Ce système propose aux apprenants suivant un cours en ligne de réutiliser le parcours d'apprentissage d'autres apprenants en se fondant sur les principes du Raisonnement à Partir de Cas [HM03]. PIXED exploite les traces d'apprentissages comme source de connaissances pour assister l'orientation des apprenants dans la progression de leurs apprentissage dans un cours. Les traces sont considérées comme base de cas qu'on peut réutiliser [HFM04] pour proposer à d'autres d'apprenants des parcours similaires. Les expériences précédentes sont des cas qu'un mécanisme de remémoration-adaptation permet de réutiliser dans un contexte similaire.

Un autre exemple concerne les travaux de France, Heraud, Marty & Carron [FHM+06] montrant comment une visualisation de trace en temps réel permet au tuteur d'observer l'activité d'un groupe d'apprenants et d'adapter l'activité en interagissant directement avec l'interface de visualisation. La figure 2.13 utilise des bulles pour représenter des activités et des liens pour représenter les parcours entre les activités. Les apprenants sont représentés par des émoticônes.

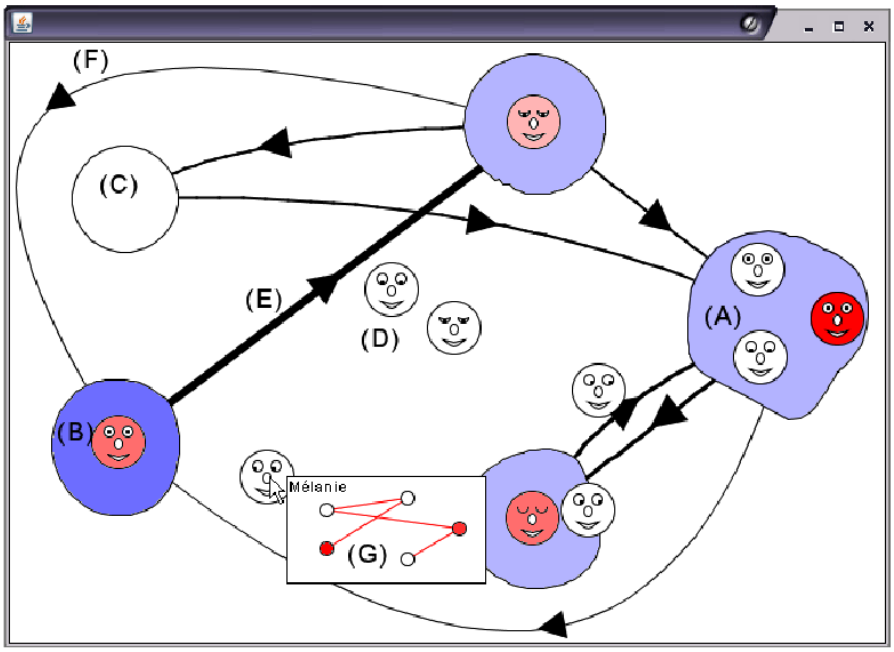

FIGURE 2.13 – Une visualisation des états des apprenants, les parcours effectuées et les activités réalisées [FHM+06]

Ces différentes visualisations évoluent dynamiquement dans le temps, en fonction des traces laissées par l'apprenant pendant la session. Par exemple, la forme des yeux du «visage» d'un apprenant donné (e.g. fermés ou ouverts) correspond à la quantité d'événements produits par l'apprenant pour une certaine activité. Si un apprenant «dort» dans une activité particulière, le tuteur peut intervenir en lui demandant s'il rencontre un problème particulier. L'acteur produisant la trace est ici l'apprenant au sein de son groupe. C'est l'EIAH qui manipule la trace

en temps réel et c'est l'enseignant qui est destinataire de la visualisation de la trace. L'objectif précis est de permettre des rétroactions du tuteur par rapport à la visualisation de l'activité des élèves exprimée en termes de sa participation et de son parcours au travers les activités. Les adaptations peuvent être de plusieurs natures : aider un apprenant qui a des difficultés (cela se voit par son retard pour une activité donnée), proposer une activité supplémentaire, échanger avec l'ensemble des étudiants travaillant sur une activité donnée, réguler les outils disponibles pour une activité donnée ou encore ordonner les activités.

Traitements et processus pour l'exploitation des traces

Les traitements considérées pour les adaptations des interactions en temps réel sont essentiellement des calculs d'indicateurs, des statistiques ou des agrégations de données .

Certains EIAH permettent la visualisation en temps réel de statistiques calculées à partir des traces. Par exemple, [Des01] a développé un environnement ESSAIM permettant au tuteur de voir à partir de traces d'interactions le degré d'interaction de l'apprenant ainsi que le temps qu'il a passé dans chaque étape. L'application Combien ? [CGT08] présente des statistiques pour chaque exercice fait (durée pour trouver la solution, nombre d'erreurs ou d'exercices achevées). Synergo [AMK04] dans un contexte d'usage collaboratif permet lui aussi de visualiser entre autres le nombre de messages émis et le nombre d'objets manipulés. Le système DIAS (Discussion Interaction Analysis System) [BD06] permet la visualisation d'indicateurs (degré d'interaction, taux de contribution, etc.) pour mieux assister la collaboration.

La plateforme COTRAS [Jer04] permet de calculer l'indicateur de proportion entre deux types d'actions en comparant deux types d'actions différents comme par exemple la proportion entre les actions de type chat, messages privés, etc. COTRAS permet de calculer l'équilibre entre les actions qui servent à la production dans la résolution d'un problème donné, et les actions classées comme actions de dialogue entre les utilisateurs de la plateforme. La plateforme FrEe-Styler [Gas03] permet de visualiser le taux de participation en calculant le taux de participation des acteurs par rapport à un type d'action donné. Cette information obtenue en temps réel à partir des traces permettra (1) aux enseignants de repérer les apprenants actifs par rapport au nombre d'actions faites par chaque apprenant et (2) aux apprenants de régler leurs efforts par rapport à la valeur affichée par l'indicateur. L'outil DEGREE [BV00] permet de calculer le nombre d'actions faites par tous les acteurs de la plateforme en prenant en compte la contribution des différents acteurs dans une activité par rapport à une action. DEGREE affiche en temps réel pour chaque type d'action le nombre des actions faites par tous les acteurs. Ceci permettra d'aider l'enseignant à voir le type d'action le plus utilisé par les apprenants et ainsi choisir par exemple de regarder ce qui se passe sur le « Chat » et non pas sur le « Forum » dans le cas où le nombre des actions de type «Chat» est plus significatif.

D'autres exemples d'outils peuvent être combinés pour l'analyse de traces. Par exemple, les trois outils d'analyse QUEST, SAMSA (System for Adjacency Matrix and Sociogram-based Analysis), et Nud*IST [MGRD03] sont utilisés pour calculer le degré de centralité en utilisant

des techniques d'analyse des réseaux sociaux.

Les exemples précédents montrent bien que même si la majorité des EIAH tracent les interactions des apprenants (en inscrivant dans une trace généralement les actions et les objets manipulés), les adaptations sont faites sur les informations compilées à partir de cette trace. Certains EIAH permettent une visualisation cohérente de ce parcours, synthétisée au bon niveau d'abstraction. Par exemple, le projet FORMID [GAP+04] offre à l'enseignant une visualisation de l'état d'avancement des apprenants en utilisant un modèle de tâche permettant le traitement et la structuration de la trace au bon niveau de besoin du tuteur. CourseVis [MD03, MD07] et VLE [HAB04] permettent de visualiser le parcours d'un apprenant pour cerner son comportement en utilisant des traces issues de la plateforme WebCT [15]. CollabLogger [MS00] permet de voir les parcours d'un groupe d'apprenants collaborant. GISMO (Graphical Interactive Student Monitoring System for Moodle) [MD05, BM07] trace les activités des apprenants sur la plateforme Moodle. Cet outil permet de générer des visualisations à partir des traces d'interactions capturées dans la plateforme en représentant les parcours des apprenants sous forme de graphe relatant le nombre d'accès aux différentes ressources Moodle.

En fait, les traitements mis en place pour supporter des adaptations en temps réel essayent tous d'augmenter le niveau d'abstraction des éléments de la trace pour qu'il soit le plus proche possible de l'activité observée et faciliter ainsi la prise de décision. Dans cette classe de traitements effectués sur les traces, on pourra inclure également les travaux sur la réflexivité et comment palier le phénomène de désorientation provoqué par les IHM des EIAH. Plus généralement, ces travaux ont pour objectif affiché de répondre à un manque de réflexivité de l'activité induit par l'utilisation d'un environnement numérique. Cette question est particulièrement importante pour les EIAH, systèmes particulièrement sensibles à la question des processus métacognitifs [OB06]. Le traitement des traces pour construire une image réflexive de l'activité est un processus temps réel nécessitant la conversion d'informations basiques de bas niveau en informations pertinentes et significatives par rapport à l'activité observée. Des traces visualisées au bon niveau d'abstraction jouent un rôle majeur dans l'apprentissage (comme dans d'autres activités cognitives instrumentées) ce qui a été largement démontré [BL97, CBFA96, HB97].

Discussion

Les travaux présentés dans cette section ne sont pas évidemment représentatifs de toutes les techniques d'adaptation en temps réel (voir [MM09] pour un état de l'art exhaustif). Cependant, ils montrent une pratique récurrente et une certaine tendance : l'usage de traces pour combler les lacunes des EIAH à observer convenablement, par des enseignants, les activités des apprenants. En effet, les enseignants-tuteurs animant des activités pédagogiques au sein de certains EIAH sont parfois désorientés par le manque d'éléments perceptibles permettant le suivi des activités des apprenants. Certains travaux ont mis en évidence la problématique du « manque de conscience » de l'enseignant des activités de ces apprenants pendant une situation d'appren-

15. http ://www.webct.com/

tissage médiée [MHFC07]. Dans une situation présentielle classique, l'enseignant a toujours des « feedbacks » visuels ou exprimés par les apprenants (questions, commentaires, etc.) qui lui permettent de réagir de différentes manières. Pour pallier cette différence de « repères », les traces sont utilisées pour reconstituer des éléments de perception de l'activité.

Ces éléments sont obtenus en effectuant un certain nombre de traitements sur les traces (extraction, filtrage, calculs, etc.). Il est important de noter que d'un tuteur à un autre les éléments de perception nécessaires peuvent être très différents, certains étant par exemple préoccupés par les éléments liés à la performance des apprenants, d'autres à la communication qui s'instaure dans l'activité d'apprentissage collaborative. Sous cet angle, la personnalisation de la perception au sein de l'EIAH est fortement liée à la possibilité d'exploitation des traces d'usage. Et comme le montrent les exemples présentés, les indicateurs calculés à base de traces sont les éléments de perception les plus courants dans les EIAH.

Certes, les indicateurs calculés permettent de présenter la trace de manière très synthétique sous forme de bilan, ce qui permet de fournir à l'enseignant une analyse « objective » du travail d'un apprenant [Ren00]. Utilisés pendant ou après l'activité d'apprentissage, des indicateurs simples peuvent décrire des informations précises et ponctuelles, comme par exemple le nombre d'accès à une ressource en ligne, le nombre d'essais pour chaque réponse, le temps passé dans chaque activité, le taux de réussite, etc. Cependant, les indicateurs sont pertinents lorsque l'EIAH adapte son IHM pour donner une vue globale de l'activité pour l'apprenant ou l'enseignant. Par exemple, les renseignements et indicateurs tirés de la trace sont d'importance pour l'enseignant qui veut s'assurer que ses étudiants ont parcouru le matériel pédagogique, mais ils ne révèlent rien sur la nature de ce parcours. En fait, l'utilisation des traces comme une source d'information quantitative révèle par exemple que l'apprenant s'est engagé dans une tâche ou activité, mais ne montre pas l'expérience et le détail réel sur son processus d'interaction. Dans la plupart des cas, il est en effet nécessaire pour un enseignant-tuteur voulant intervenir sur une situation de connaître le cheminement effectué par l'apprenant pour accomplir sa tâche. Ceci permettra par exemple de détecter, pendant l'activité, que l'apprenant s'est égaré et éloigné de sa tâche ou bien *a posteriori* d'expliquer à l'apprenant la cause de son succès ou de son échec.

Le niveau de détail des éléments disponibles dans la trace permettant de suivre la progression d'un apprenant peut être discuté et est en soi une question de recherche complexe. Une trace brute montrant la succession des actions élémentaires opérées sur l'interface (clique souris, frappe clavier, etc.) n'est pas souvent pertinente. Celle-ci serait en effet inexploitable par exemple par un tuteur en situation de suivi car les éléments produits par les apprenants au niveau de l'interface sont trop éloignés de la sémantique de l'activité qu'ils effectue [GAP+04]. Dans ce sens, Labat [Lab02] argumente que les EIAH doivent fournir des informations d'un plus haut niveau d'abstraction aidant le tuteur à apprécier le travail de manière quantitative et surtout qualitative.

En plus de calculs d'indicateurs, il est essentiel de traiter la trace de manière à garder le détail et la chronologie des observations mais à un niveau d'abstraction significatif par rapport aux tâches et activités tracées. Ces traitements permettront entre autres de refléter l'activité de

l'apprenant. Refléter ce que fait l'apprenant avec un EIAH lui permet de prendre « conscience » de la tâche qu'il effectue [CDP06]. Par exemple, il peut revenir sur son activité et peut (re)voir les questions et les hypothèses qu'il a formulées tout au long de sa session d'apprentissage [NB91].

Cependant, bon nombre d'EIAH tracent les interactions entre apprenant-environnement sous la forme de fichiers souvent non exploitables directement par cet apprenant. Fournir une visualisation à l'apprenant de sa propre trace, en temps réel dans une forme qui « fait sens » par rapport à son activité d'apprentissage peut s'avérer complexe. Bien, que les traces d'interaction d'un apprenant permettent de donner une « image fidèle » de son activité [OB06], elle nécessite souvent des traitements d'abstraction, de filtrage, etc.

Même si la majorité des EIAH tracent l'apprenant et permettent la visualisation temps réel de cette trace, ils ne lui permettent pas souvent d'interagir avec elle. Quelques systèmes permettent cependant cette interactivité. Par exemple, SimPLE (Simulated Process in a Learning Environment) [PRR+99] propose aux apprenants une visualisation interactive de leurs actions permettant d'annoter certaines parties et d'en éditer des éléments. APLUSIX [Nic94] permet aussi à l'apprenant d'interagir avec sa trace en parcourant les calculs qu'il a effectués. L'apprenant peut ainsi corriger lui-même les exercices qu'il n'a pas résolus correctement en reprenant la résolution et agir donc à partir de sa trace. Carroll [CBFA96] argumente que les traces peuvent être considérées comme des expériences passées pouvant aider les apprenants à comprendre ce qu'ils ont fait, à revoir leurs actions, à sauvegarder ces traces de manière à les rejouer plus tard, identifier la partie de l'histoire qu'ils ont tracé qui les intéressent. Gay & Mazur [GM93] considèrent que les apprenants ont une perception de leur travail qui ne reflète pas toujours la réalité. La trace favorise la « prise de conscience » de leur activité à un niveau « métacognitif » [OB06] en les aidant notamment à approfondir leur apprentissage [WM06].

Il est évident au vu de ces exemples que les traces requièrent de nécessaires reformulations et abstractions pour les exploiter. Comme on l'a montré, ces traitements sont effectués moyennant des calculs (indicateurs) et des interprétations (humaines ou automatiques). L'objectif de ces reformulations est d'amener le niveau d'abstraction des traces dans le registre de l'activité observée. La reformulation et le traitement des traces imposent l'expression de requêtes et de règles complexes dont les réponses sont utilisées pour offrir des visualisations interactives ou proposer des feedbacks en quasi temps-réel. L'interprétation de traces en temps réel ou différé nécessite ainsi le développement d'opérateurs permettant ces traitements et ces opérateurs doivent permettre d'obtenir des traces transformées au bon niveau d'abstraction.

2.4 Synthèse de pratiques autour des traces

Les trois usages de traces numériques d'interaction issus de la littérature et présentés ici peuvent s'organiser comme l'illustre le tableau 2.1. Pour chaque exemple d'usage, l'entité produisant la trace est l'individu ou le groupe. L'acteur manipulant la trace est l'apprenant, l'enseignant, le chercheur ou l'EIAH ; le destinataire étant parmi les trois premiers de cette liste. La manipulation de trace se passe en temps réel ou *a posteriori*. Le type de manipulation est

automatique ou manuel. L'objectif précis porte sur une étape allant vers un but d'apprentissage ou sur une façon de communiquer qui, par hypothèse, favorise l'apprentissage. Bien sûr, l'adaptation proposée varie en fonction de l'objectif considéré. Les exemples que nous avons précédemment présentés sont synthétisés dans le tableau 2.1 de façon à pouvoir comparer les trois usages de traces numériques d'interactions.

	Études empiriques et analyse des interactions	**La réingénierie d'un EIAH**	**Adaptation des interactions en temps réel**
Acteur produisant la trace	Apprenants – individus et groupes	Apprenants – individus	Individus , Apprenants en groupe
Acteurs manipulant la trace	Apprenants et chercheurs	EIAH	EIAH et Apprenants
Destinataire des manipulations de la trace	Apprenants et chercheurs	Apprenant, concepteur d'EIAH, enseignant	Apprenant ou enseignant
Quand se passe les manipulations	Lors d'une phase où l'historique des interaction est ré-utilisée (apprenant) ou *a posteriori* du scénario (chercheur)	En temps réel ou *a posteriori*	Temps réel ou au bout d'une certaine quantité de temps
Type de manipulation	Choisir les éléments à proposer (apprenants) ; Rejouer et /ou analyser la trace numérique d'interactions à la main ou assisté d'un logiciel (chercheur)	Modéliser les connaissances et/ou l'activité de l'apprenant ; enregistrer des données quantitatives par rapport à son activité, la diagnostiquer	Calculer automatiquement des informations de natures quantitatives et qualitatives par rapport à l'activité menée
Objectif précis des manipulations	Évaluer les différences entre les activités menées par les apprenants selon le scénario pédagogique diagnostiqué	Définir une personnalisation basée sur un diagnostic de l'activité de l'apprenant	Définir une personnalisation basée sur un diagnostic de l'activité de l'apprenant et/ou Refléter l'activité de l'apprenant et d'un groupe d'apprenants
Adaptation proposée	Faire un appariement entre l'interaction voulue et les fonctionnalités qu'elle provoque	Proposer du contenu sous diverses formes, adapté au profil de l'élève lors d'un prochain scénario ou lors d'une phase successive	Suggérer une action pour combler un manque ou pour rendre plus équilibrée une activité ou une tâche donnée ou visualiser l'historique des interactions avec parfois des possibilités de zoomer, filtrer, annoter ou regrouper certaines parties de la trace

TABLE 2.1 – Synthèse des usages des traces en EIAH

Le tableau ne reflète donc que les exemples cités et ne prétend pas être exhaustif par rapport à la totalité des recherches menées sur les trois usages présentés. Nous pouvons remarquer un certain nombre de choses. Premièrement, il manque dans les recherches décrites des exemples d'enseignants ou de chercheurs produisant des traces pour un objectif spécifique. Deuxièmement, parfois la manipulation sur une trace amène directement à une adaptation et parfois la

manipulation et l'adaptation qui en résulte sont décalées dans le temps. Une adaptation peut être proposée dès qu'il y a suffisamment de données recueillies (dans un cas de surveillance d'un groupe d'apprenants par un tuteur). Une adaptation peut aussi cibler une phase particulière d'un même scénario pédagogique (proposer un nouveau contenu adapté à un diagnostic) ou d'un futur scénario (proposer un outil qui donne lieu à un type d'interaction supposé favorable pour l'apprentissage). Troisièmement, certains types d'usage de traces ont pour objectif d'analyser des différences entre les activités menées par les apprenants et les caractéristiques du scénario pédagogique. La personnalisation d'une interaction EIAH pour ce type d'usage de trace, peut venir dans un deuxième temps. Cependant, l'usage de trace pour la réingénierie peut avoir pour objectif premier une adaptation personnalisée et en temps réel.

2.5 Discussion

La majorité des travaux présentés utilisent la trace comme « objet de soutien » à l'activité de l'apprenant, de l'enseignant, du concepteur et de l'analyste-chercheur et peu d'entre eux proposent d'identifier une raison fondamentale à la création d'outils pour l'exploitation des traces. Le plus souvent, les techniques de manipulation et d'exploitation des traces d'interaction sont développées avant tout comme une fonctionnalité pratique du système tracé.

Afin de construire des outils génériques pour la manipulation de traces, il nous faut d'abord analyser les pratiques existantes et dégager les propriétés les plus saillantes des traces considérées pour la personnalisation. Nous identifions dans la suite ces propriétés par rapport à l'existant. De nombreux travaux de la communauté EIAH sur l'analyse des traces, issues d'environnements différents et pour des objectifs divers et parfois multiples ont été menés. L'étude de ces travaux permet de dégager un certain nombre de propriétés requises par les traces et les outils permettant leur exploitation.

2.5.1 Unifier les traces : formats ou modèles ?

Il existe beaucoup d'EIAH dits généraux (e.g. Moodle, Synergo [AMK04]) sur lesquels les chercheurs informaticiens peuvent intervenir pour définir les traces à générer et leur format. Cependant, de très nombreuses expérimentations utilisent ces plateformes généralistes (e.g. WebCT, Dokeos, Moodle, Lyceum) sans avoir la main sur le processus de génération des traces. Il est difficile d'instrumenter un EIAH pour obtenir des traces voulues (pour répondre à un objectif précis) mais c'est au moins aussi difficile d'utiliser les traces générées automatiquement par cet EIAH notamment dans le cas où elles ont été produites pour répondre à un autre objectif. L'approche par la mise en place de format commun peut être une solution puisqu'elle permettra d'homogénéiser les traitements et les traces.

D'ailleurs, c'est la solution la plus courante mise en œuvre en premier par les projets CAViCoLA et IA pour résoudre le problème de l'interopérabilité des EIAH et des outils d'analyse. Un format unifié de traces d'interaction, qui a vocation de standard pour le développement d'outils

d'analyse et d'EIAH [HMMD08], peut jouer le rôle de représentation pivot entre les formats de sortie et d'entrée des EIAH et outils existants. Dans ce sens, [May09] a tenté aussi de proposer un format pour les outils de communication. Bien que tous ces résultats soient intéressants, ils ne répondent que de manière partielle au problème de l'interopérabilité. En effet, une approche par (méta) modélisation donne de meilleurs résultats (notamment en termes de représentation bien définie et structurée) plutôt qu'un format de représentation des données. Le format ne fait que limiter les modèles de traces au lieu de les représenter et un modèle commun pour représenter toutes les traces nécessite un consensus au niveau ontologique (i.e., que doit-on représenter ?), et non seulement au niveau des formats. Un tel consensus étant souvent impossible pour des usages spécifiques, un méta-modèle permettant de représenter les modèles des traces produites par les EIAH et les modèles de traces requis par les outils d'analyse est plus adéquat.

2.5.2 Trace support d'analyses *post-hoc* et d'adaptations *ad-hoc*

Nous avons vu dans les différentes catégories d'usage que la trace obtenue peut être visualisée *a posteriori* par des chercheurs ou *à la volée* par les apprenants (mirroring). Elle peut aussi subir des traitements ou des analyses automatiques pour construire des indicateurs de suivi (monitoring). Elle peut également être comparée à un modèle attendu, caractérisant un succès ou une erreur documentée pour construire une rétroaction permettant le guidage (guiding) de l'acteur au cours de l'activité [JSM01]. Les traces d'interactions recueillies concernent, pour une part importante, une communication qui s'inscrit dans un contexte initiée par une action sur l'IHM. Les interactions humaines médiées par les plateformes y jouent un rôle essentiel en particulier dans le cas des scénarios d'apprentissage collaboratifs. Les analyses quantitatives [RC03] apportent quelques indicateurs utiles pour le suivi de l'activité (durée, volume des interactions, fréquence, etc.), mais le besoin d'instrumenter les analyses qualitatives ne cesse de s'accentuer, notamment à travers la recherche de patterns, ou sur le détail des interactions homme-machine (e.g. [MGP07] sur le poste client) pour donner plus de précision sur la nature des activités des apprenants.

Construire des outils pour l'exploitation des traces consiste à concilier toutes ces pratiques à base de traces. D'un point de vue applicatif, il s'agit de combler l'écart et rapprocher les analyses minutieuses faites *a posteriori* de l'observation et les exploitations spécifiques faites sur des traces durant l'activité pour mettre en place la fonctionnalité voulue (e.g. monitoring, guiding, etc.)

Pour ce qui est des analyses *post-hoc*, les pratiques sont guidées par les méthodes et les outils issus des sciences humaines utilisant souvent des outils généraux de traitement automatique des langues, des outils de transcription et d'annotation vidéo, etc. Ceux-ci fonctionnent assez bien pour les sciences humaines, mais pour s'adapter aux multiples formes présentes dans les situations écologiques d'apprentissage, ils doivent être enrichis et intégrer des fonctions plus poussées en termes de recherche de motif (pattern), de transformation et d'abstraction de données. Certains outils ont progressé dans cette voie et peuvent être utilisés par des enseignants chercheurs (e.g.

l'évaluation des usages notamment en terme d'IHM). Dans ce contexte, il existe, en réduisant, deux types d'approches opposées, se combinant parfois dans certains systèmes d'aide à l'analyse de traces. Les premières, quantitatives, consistent à utiliser une observation passive des interactions en récupérant des fichiers log. Sur la base de calculs statistiques, ces traces permettent de donner des indicateurs pour décrire des critères d'évaluation de systèmes informatiques comme Utilité et Utilisabilité [Hoo97, HR00] ou caractériser le comportement de l'utilisateur [Hul04]. On peut citer dans ces catégories les systèmes : AUS [CD97], EDEM [HRR97], USINE [LP98], DRUM [MR93], MacSHAPA [SF94]. Les secondes approches, qualitatives, consiste à travailler sur les éléments de la trace en s'appuyant sur une observation minutieuse *in situ*, appuyée par des enregistrements divers notamment audio et vidéo ainsi que d'autres sources complémentaires comme les interviews. Les travaux sur ce type d'analyse sont utilisés dans différents domaines : Interactions humain-machine (HCI)[Bel90, Fis01], ergonomie et IHM [HR00], ethnométhodologie [Mag96], etc. L'analyse qualitative consiste à travailler sur les éléments de la trace en identifiant de manière exploratoire les séquences et parties pertinentes de la trace. Beaucoup de systèmes permettent ce type d'analyse, par exemple :

- La détection et reconnaissance de séquence pertinente dans les traces, e.g., les systèmes I-Observe[BGHS95], EBBA[Bat95] et GEM [MsSS97].
- La comparaison entre séquences de traces, e.g., les systèmes ADAM [FH90], EMA [Bal96], USINE[LP98], THEME [Mag96].
- La caractérisation et extraction de modèles de prédiction à partir des traces en se basant sur des approches de reconnaissance à base de grammaire [OHR94] ou des approches probabilistes basé sur le modèle de Markov [Guz93].
- La sélection, abstraction, recodage et transformation d'éléments de la trace comme dans EDEM [HR00], Incident Monitoring [Che90] ;

Ces outils ont tous en commun la rigidité du modèle de trace qu'ils utilisent. En effet, ces systèmes se limitent à un modèle défini par la concepteur de l'outil, pouvant être traité de manière *ad-hoc* par l'analyste, un modèle trop général s'adaptant à différentes situations sans pour autant pouvoir fournir la possibilité de soutenir convenablement les différents besoins. Peu de ces systèmes sont vraiment génériques, ils contraignent leurs utilisations et utilisateurs au passage à un modèle propriétaire souvent statique délimitant les possibilités de traiter la trace comme une inscription de connaissances qu'un EIAH peut utiliser.

À côté des pratiques d'analyses de traces *a posteriori* de l'activité d'observation, les exploitations en temps réel de l'activité répondent à des besoins pratiques des EIAH. Que ce soit pour rendre l'activité d'apprentissage réflexive, guider l'apprenant dans sa tâche ou surveiller son activité pour mieux l'aider, des calculs d'indicateurs et des traitements sont appliqués pour qualifier l'activité de l'utilisateur en s'appuyant sur un traçage minutieux *in situ*, appuyé par une visualisation adéquate. Les traces destinées à l'apprenant peuvent également être plus ou moins interprétées par l'environnement en fonction de l'approche d'assistance choisie. Si un profil d'utilisateur ou un modèle de tâche existe, l'EIAH pourra prédire ce que l'utilisateur fera en fonction de ce qu'il a fait auparavant, ou en réutilisant des parties de traces qu'on a jugé « similaires »

pour lui faire des suggestions ou proposer des conseils. De tels techniques se fondent sur des profils d'utilisateur considérant la trace comme une inscription de connaissance que le système peut traiter. La notion de profil d'utilisateur ou de modèle d'utilisateur est un concept abondamment exploité par les systèmes dit adaptatifs (pour une synthèse, voir l'état de l'art de Brusilovsky [Bru01] et celui de Kobsa [Kob01]). Le modèle de l'utilisateur décrit généralement d'une manière abstraite des informations caractérisant l'utilisateur [Kob01]. Ces informations contenues dans le modèle de l'utilisateur caractérisent aussi en partie son usage et son comportement lors d'une activité médiée par un système informatique. Les traces d'utilisation semblent être au moins aussi légitimes que les modèles d'utilisateurs pour piloter la personnalisation [CP02]. En effet, personnaliser un système informatique revient à influer sur ses modes d'utilisation pour répondre aux besoins de l'utilisateur, et donc nécessite d'observer et de tracer son utilisation.

Dans le contexte des EIAH, la question des modèles utilisateur et de leurs relations avec les traces d'utilisation a été largement identifiée notamment pour le champ des tutoriels intelligents. Le modèle de l'apprenant, qui peut être considéré comme un modèle utilisateur particulier, est défini comme étant une représentation de tous les aspects liés au comportement et aux connaissances de l'apprenant [Wen87]. Dans ce domaine, la question des traces et de leurs liens avec le modèle de l'apprenant a été pointée par les pionniers du domaine dès l'apparition des premiers tutoriels intelligents. Par exemple, Self souligne que pour construire le modèle de l'apprenant, il est nécessaire de posséder un historique des interactions de l'apprenant avec le système [Sel87]. Greer [HDJ+94] considère le modèle de l'apprenant comme une représentation abstraite des croyances, des connaissances et des compétences de l'apprenant dans le système, incluant aussi les traces de l'historique des actions de l'apprenant qui peuvent être analysées et interprétées.

Le domaine de modélisation de l'apprenant est un domaine très actif relatant de nombreuses approches issues en grande majorité de l'Intelligence Artificielle (IA) (voir le tour d'horizon des techniques utilisées dans [Wen87, Sel87, Van88]). La modélisation de l'apprenant désigne le procédé, ou la méthode de calcul, qui permet d'instancier le modèle de l'apprenant à partir des traces d'interactions de l'apprenant avec le système. On parle aussi de diagnostic [16] comme étant l'inférence de l'état cognitif caché de l'apprenant. Balacheff [Bal94] distingue deux manières d'appliquer le diagnostic : *le diagnostic orienté par le modèle* (*top-down*), qui consiste, à partir d'un modèle très général englobant tous les comportements possibles, à réduire les cas possibles à partir des traces du comportement de l'apprenant ; *un diagnostic orienté par les interactions* (*bottom-up*) qui consiste à construire le modèle à partir des traces du comportement, sans forcement se baser sur un modèle général prédéfini.

Les environnements spécifiques comme les tutoriels intelligents (e.g. Aplusix [BI00], Logic-ITA [Yac05]) considèrent aussi les traces comme des connaissances sur lesquelles l'EIAH peut raisonner, faire des interprétations et des inférences. Les traces sont contraintes par des règles calculables, qui permettent une analyse sémantique. Ces environnements proposent des activités d'apprentissage personnalisées, puisque le contenu de trace (considérée comme connaissance) est

16. faisant référence au diagnostic médical : à partir d'observations, le médecin diagnostique l'état physiologique caché du patient.

interprétable, pouvant être traité pour construire automatiquement une rétroaction (feedback) appropriée.

Créer un outil permettant la mise en œuvre d'analyses de traces pendant et après l'activité nécessite la définition de langages permettant non seulement l'interrogation et la recherche de motifs intéressants, le calcul et l'agrégation d'indicateurs pertinents pour l'activité mais aussi des inférences automatiques de nouveaux éléments de la trace.

2.5.3 Traces ouvertes et fermées : exploitation ponctuelle ou continue ?

D'un point de vue technique, la mise en œuvre de langages de manipulation de traces nécessite deux considérations très importantes qui n'ont pas été identifiées et explicitées dans les travaux cités auparavant.

Le première considération concerne l'état des traces manipulées. Les exemples montrés dans ce chapitre et notre expérience dans différents projets montrent bien que les traces peuvent être considérées de deux manières différentes : (1) *fermées* dans le sens où elles n'attendent plus des ajouts de nouveaux d'éléments et (2) *ouvertes* lorsqu'elles sont toujours en cours de collecte et ouvertes à l'ajout de nouveaux observés. Il est évident que le premier correspond aux exploitations post-observation où les traces sont fermées pour toutes les adjonctions d'éléments alors que la seconde correspond à une analyse en temps réel de l'activité où de nouvelles observations sont capturées continuellement au fur et à mesure du déroulement de l'activité. Une trace ouverte est fermée si on la considère dans un intervalle de temps figé.

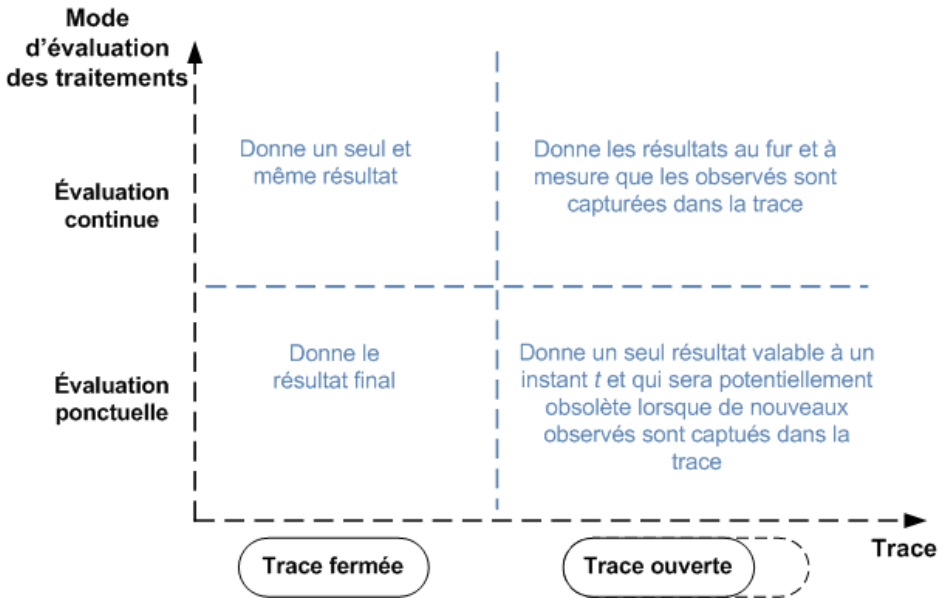

FIGURE 2.14 – Mode d'évaluation des traces ouvertes et fermées

La seconde considération concerne les modes d'évaluation des traitements effectués sur la traces (pouvant être ouvertes ou fermées selon les situations). En effet, les deux cas de figures contraignent énormément la nature des traitements qui peuvent être effectués sur les traces.

Dans la littérature, deux modes de traitements ou d'évaluations sont possibles : les traitements ponctuels, i.e., exécutés une seule fois à un instant t et qui donnent un seul résultat ,e.g., une requête sur une base de données [AHV95] ; les traitements continus qui sont exécutés plusieurs fois selon une périodicité ou à chaque fois qu'un nouvel élément de la trace arrive, e.g., les requêtes continues sur un flux de données [Ara06]. L'originalité de la problématique des traces et de leurs traitements découle de cette combinaison de ces deux considérations et modes de traitements dans un même cadre théorique.

2.6 Notre approche pour l'exploitation des traces : Système à Base de Traces Modélisées

Pour construire des outils interopérables, facilitant l'usage des traces dans différentes configurations, nous présentons dans la suite un cadre théorique définissant les concepts et les services sous-jacents à l'exploitation des traces. Le cadre théorique que nous allons définir trouve son origine dans les travaux de l'équipe Silex sur l'approche MUSETTE [CPM03, Cha03]. L'objectif de notre cadre est de définir une nouvelle classe de systèmes à base de connaissances baptisée *Système à Base de Traces modélisées (SBT)*. L'idée principale soutenant notre proposition est de permettre de construire des systèmes informatiques fonctionnant comme des SBC en permettant de modéliser des traces (pouvant être dans différents formats), les interroger pendant et/ou après leurs obtentions et de faire des interprétations et des raisonnements de manière ponctuelle ou continue.

Contrairement aux approches existantes (notamment les projets ICALTS [ICA04], IA [IA05], Cavicola [CAV06]), notre cadre ne fixe aucun format de trace. En outre, il définit les services pouvant être effectués sur les traces dans des modalités d'exploitations *in situ* et *a posteriori* de l'activité tracée (i.e., des traitements ponctuels et continus). Les systèmes et les utilisateurs voulant exploiter les traces disposent de langages pour modéliser, interroger et transformer des traces moyennant des règles de transformation.

Le cadre des SBT permet aussi d'identifier les propriétés des langages devant être considérées lors de la mise en place de traitements de traces. En effet, la discussion des différents travaux utilisant les traces et notre expérience dans plusieurs projets nous ont amenés à revoir et à analyser les caractéristiques requises pour les langages soutenant l'exploitation des traces. Les langages proposés doivent prendre en compte plusieurs dimensions complémentaires et nécessaires pour permettre l'interrogation et la transformation efficace des traces :

– Ces langages doivent permettre d'**interroger des données complexes et structurés** disposant explicitement de **modèles riches** (hiérarchies de types et de relations typées etc.). En effet, les éléments de la trace peuvent avoir des valeurs d'attribut, être en relation avec d'autres éléments. Les langages doivent offrir ainsi la possibilité d'exprimer des requêtes prenant en compte ces caractéristiques pour permettre l'identification de motifs pertinents pour différentes exploitations.

- Ces langages doivent permettre la représentation, l'interrogation et la construction d'éléments de la traces qui sont situés dans un **domaine temporel**.
- Ces langages doivent permettre de **mettre en œuvre des transformations de traces** permettant de construire de **nouvelles traces à partir des traces existantes**. Moyennant des règles de transformations, ils doivent permettre de déduire et de construire de nouveaux éléments à partir des éléments existants.
- Ces langages doivent permettre des **évaluations de requêtes et de transformations de manière continue** sur des traces pouvant être non bornées dans le temps, i.e., des évaluations continues appliquées sur **des traces évoluant continuellement dans le temps au fur et à mesure de la capture de nouveaux éléments de la trace**.

Ces propriétés sont souvent implicites dans les langages existants, mais nous avons pas identifié jusqu'à maintenant des travaux les étudiant explicitement en tant que telles. À notre connaissances, notre approche est le premier travail à traiter des éléments contraints par un modèle ayant des relations complexes enregistrées dans des traces pouvant être ouvertes ; potentiellement infinies et utilisant les règles de transformation comme mécanisme de raisonnement. Notre cadre utilise des règles de transformation comme un mécanisme d'abstraction et de raisonnement, avec la même motivation et les mêmes avantages de l'usage des règles dans les systèmes de base de connaissances traditionnels. Il intègre des langages qui permettent de spécifier des modèles, des requêtes pour l'extraction des éléments de la trace (sous forme de substitutions de variables), des transformations pour construire de nouveaux éléments dans des traces transformées, et ce, de façon ponctuelle ou continue.

Les chapitres suivants présentent le cadre conceptuel et formel des système à base de traces modélisées et discute les langages permettant leurs mises en œuvre.

Deuxième partie

Contributions

Chapitre 3

Notion de Système à Base de Traces modélisées

Sommaire

Le chapitre précédent a permis de présenter les différentes facettes de l'usage des traces numériques d'interactions dans le cadre des EIAH. Il a été également l'occasion de démontrer que les traces numériques d'interaction sont centrales pour permettre des personnalisations diverses de l'activité d'apprentissage, justifiant ainsi une théorisation de ces traces comme des « objets informatiques » à part entière manipulables en tant que telles. Dans le prolongement des chapitres précédents, ce chapitre sera consacré à la présentation de l'approche des Systèmes à Base de Traces modélisées positionnée dans le cadre des Systèmes à Base de Connaissances (SBC). Après la mise en lumière des différences existantes entre les SBC et les SBT, nous présenterons les différents concepts et services impliqués par le cadre des SBT. Enfin, nous discuterons les propriétés et les langages requis par l'exploitation des traces.

3.1 Introduction

La conception et la réalisation de systèmes génériques et interopérables pour l'exploitation de traces pour rendre des services divers et variés nécessite avant tout la spécification d'un cadre définissant les propriétés qu'elles requièrent. Les chapitres précédents ont montré que cette spécification doit prendre en compte quelques aspects distincts et complémentaires : (1) modélisation des traces, (2) extraction d'éléments pertinents de la trace et (3) construction de nouveaux éléments à partir des traces existantes.

Notre proposition pour prendre en compte ces aspects est la définition d'une classe particulière de Systèmes à Base de Connaissances (SBC) appelée *système à base de traces modélisées* pouvant être exploités dans différentes situations. Bien que les SBC ne soient pas bien adaptés pour l'interrogation et l'extraction efficace de motifs à partir des données (aspect 2), ils excellent pour des tâches de représentation de connaissances et de raisonnement (aspects 1 et 3). La suite de chapitre se propose d'illustrer cette proposition et d'en montrer l'intérêt théorique et pratique (avec plus de détails dans les chapitres 4 et 5).

3.2 Pourquoi un cadre conceptuel pour la manipulation de traces

Un cadre conceptuel définit l'ensemble des connaissances, des théories, qui ont un rapport quelconque avec un sujet de recherche. Dans le contexte de la manipulation des traces, il s'agit des concepts et des processus mobilisés pour et lors de l'exploitation des éléments de la trace pour mener à terme une tâche ou fournir une fonctionnalité donnée. Ces concepts vont servir de points de repère et devenir par la suite un cadre formel pour la manipulation des traces (le chapitre 4).

L'objectif de ce cadre conceptuel est double : d'une part, il offre une fonction d'organisation dans le sens où il oriente la démarche et la logique des étapes nécessaires pour manipuler les traces, et d'un autre coté, le cadre conceptuel est indispensable pour évaluer les usages mis en œuvre et les résultats. Plus concrètement, le cadre conceptuel permettra d'une part d'étudier les outils utilisant des traces en repérant les différentes exploitations possibles, et d'autre part, ces exploitations identifiées permettront de construire une nouvelle classe d'outils pour la manipulation de traces homogènes et interopérables. D'un point de vue technique, le cadre conceptuel permet ainsi de (Figure 3.1) :

- modéliser les traces et spécifier les traitements dans les systèmes existants et ainsi à long terme de pouvoir s'échanger/partager/réutiliser des traces à des fins de capitalisation de pratiques ;
- construire un cadre logiciel (Framework), qui pourra être spécialisé, intégré ou utilisé pour construire divers outils ou services à base de traces.

La figure 3.1 montre les différents usages du cadre conceptuel que nous allons définir. En plus de soutenir l'étude des pratiques et des systèmes existants en terme de représentation et de traitements des traces, le cadre a pour vocation de permettre de construire un cadre logiciel pour

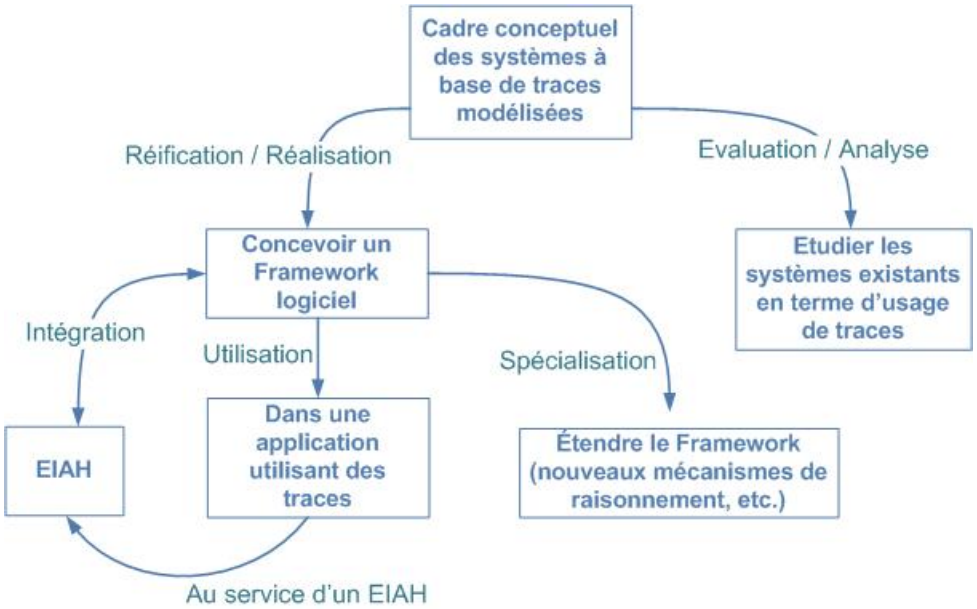

FIGURE 3.1 – Cadre conceptuel pour la manipulation des traces

l'exploitation des traces. Ce cadre logiciel pourra être intégré dans un EIAH pour manipuler des traces (déduire de nouveaux éléments, calculer des indicateurs, etc.). Il peut aussi être utilisé par une application tierce pour exploiter les traces d'un ou plusieurs EIAH afin de rendre différents services (e.g. assistance, évaluation, awareness, etc.).

Avant de présenter notre cadre conceptuel, nous présentons d'abord la notion de système à base de connaissances tout en montrant les limites des langages associées.

3.3 Systèmes à Base de Connaissances

L'ingénierie des connaissances étudie les concepts et les méthodes permettant de modéliser et d'acquérir des connaissances. Son objectif est de faciliter la conception et la réalisation de systèmes informatiques qualifiés parfois d'*intelligents* que l'on appelle les Systèmes à Base de Connaissances (SBC). Ce domaine de recherche est pluridisciplinaire. Il fait appel à la logique pour la construction de modèles formels, à la linguistique pour la formulation des connaissances et évidement à l'informatique pour la mise en application des modèles définis.

En termes informatique, un système à base de connaissances est un programme capable de raisonner pour résoudre un certain problème en s'aidant de connaissances relatives au domaine d'application considéré. Les systèmes à base de connaissances comportent une fonction de représentation des connaissances associée à une fonction de manipulation des connaissances représentées appelée raisonnement. Parmi les langages de représentation et raisonnement sur des connaissances et permettant la mise en œuvre de SBC, nous présentons dans la suite deux familles de représentations : représentation de connaissances avec des règles déductives et représentation de connaissances avec des ontologies.

3.3.1 Représentation de connaissances avec des règles déductives

L'usage de règles pour des tâches de représentation de connaissances et de raisonnements telles que des inférences déductives est relativement commun dans les SBC et bien adaptée pour leur mise en œuvre. En effet, pour permettre à des machines de déduire de nouvelles informations et de «comprendre» les données, de telles inférences sont nécessaires.

Les règles déductives sont une forme de raisonnement qui a suscité beaucoup d'attention notamment dans le domaine de IA et du Web sémantique. Les règles fournissent un mécanisme de représentation de connaissance de haut niveau, particulièrement expressif et naturel pour les ingénieurs de connaissance.

Bien qu'il y ait beaucoup de variations des langages à base de règles déductives, (pas simplement en syntaxe mais surtout dans leur sémantique), l'idée générale est toujours de spécifier la connaissance en terme de règles. Une règle reflète une situation *si-alors* et statue qu'une conclusion (la partie *alors*) peut être dérivée si la condition (la partie *si*) est vérifiée. Les conditions et les conclusions sont habituellement écrites en tant que formules logiques. La conclusion d'une règle s'appelle également *entête* et la condition son *corps*, et elle est écrite généralement sous la forme : entête ← corps. En raison de sa relation proche avec les requêtes et les formules logiques, le corps d'une règle peut également être compris comme une requête. Elles sont appelées des règles déductives parce qu'elles déduisent seulement des conclusions et ne modifient pas les données existantes et ne causent aucun autre effet secondaire contrairement aux règles de réécriture (qui spécifient des transformations de données) et les règles réactives (qui spécifient des actions à effectuer).

Dérivant de la logique du premier ordre (First Order Logic), la plupart des formalismes courants à base de règles déductives représentent des faits en utilisant des prédicats (ou des relations) et des termes. Ils emploient donc essentiellement le modèle de données relationnel avec une extension importante que beaucoup de langages de règles permettent qui est l'usage de symboles de fonction dans les termes (alors que les bases de données permettent habituellement l'usage seulement de constantes).

Le fait qu'il y ait une action avec l'identifiant "111" de nom "copy" sur le document "D11" serait par exemple représenté comme `action`("111","copy","D11"). Les règles permettent de représenter la connaissance qui dérive de nouveaux faits des faits existants. Par exemple, nous pourrions savoir que les utilisateurs ne peuvent effectuer des actions que sur les documents dont ils sont les propriétaires (`owner_document`). Avec le langage de règles Datalog [AHV95], cette connaissance pourrait être formalisée comme suit :

$$\texttt{owner\_document}(Z, U) \quad \leftarrow \quad \texttt{action}(X, Y, Z), \texttt{operated\_by}(U, X)$$

Dans cette règle X, Y, Z, et U sont des variables libres. La virgule («,») dans le corps de la règle est lu comme une conjonction. Le plus important est que la plupart des langages à base de règles déductives prennent en compte les inférences récursives. Les fermetures transitives comme

« s'il y a un lien de X à Y et de Y à Z, alors il y a un lien de X à Z» sont une application typique de telles inférences récursives. Une telle règle pourra être exprimée comme suit :

$$\begin{aligned} \texttt{link}(X,U) &\leftarrow \texttt{action}(X,Y,Z), \texttt{action}(U,Y,Z), \texttt{click}(Y) \\ \texttt{link}(X,Z) &\leftarrow \texttt{link}(X,Y), \texttt{link}(Y,Z) \end{aligned}$$

3.3.2 Représentation de connaissances avec des ontologies

Les règles déductives ne sont pas la seule forme de représentation de connaissances, les ontologies sont aussi activement utilisées pour représenter les connaissances dans les SBC. La norme W3C pour représenter les ontologies est le langage OWL (Ontology Web Language) [SWM04].

OWL, comme d'autres langages ontologiques, est basé sur des logiques de description. Les logiques de description peuvent être définies comme un fragment de la logique du premier ordre (LPO), habituellement limitées de telle manière que les tâches typiques de raisonnement, contrairement à la LPO, restent toujours décidables [17]. Les ontologies modélisent les connaissances d'un domaine en spécifiant des axiomes au sujet des concepts et de leurs relations [18]. Les concepts et les relations correspondent respectivement aux relations unaires et binaires de LPO. Les axiomes correspondent aux formules de LPO, mais sont habituellement écrits en syntaxe qui laisse les variables implicites. OWL et la plupart des logiques de description ne soutiennent pas les relations ternaires et plus, ce qui peut être une limitation sur les connaissances qui peuvent être exprimées en OWL.

Considérons comme exemple une ontologie qui modélise la connaissance au sujet des actions possibles sur des documents. Il y a des concepts tels que `DELETE_ACTION`, `ACTION`, `PRIVILEGE` et `DOCUMENT` et les relations telles que `concerns` et `grant`. Les axiomes peuvent définir de nouveaux concepts, par exemple `DELETE_ACTION` pourrait être défini comme une `ACTION` qui concerne (`concerns`) un document qui n'a pas de propriétaire (`OWNER_DOCUMENT`). Les axiomes peuvent également spécifier des relations de typage telles que le concept `OWNER_DOCUMENT` est une sorte de privilège `PRIVILEGE` ou des conditions plus générales telles chaque `DOCUMENT` doit accorder (`grant`) au moins un privilège `PRIVILEGE`. En syntaxe typique de logique de description, ces axiomes seraient écrits comme suit :

$$\begin{aligned} &\texttt{DELETE\_ACTION} \sqsubseteq \texttt{ACTION} \sqcap \forall \texttt{concerns}.\neg\, \texttt{OWNER\_DOCUMENT} \\ &\texttt{OWNER\_DOCUMENT} \sqsubseteq \texttt{PRIVILEGE} \\ &\texttt{DOCUMENT} \sqsubseteq \exists \texttt{grant}.\texttt{PRIVILEGE} \end{aligned}$$

Ces axiomes correspondent aux formules suivantes écrites en syntaxe traditionnelle de la logique du premier ordre :

17. Les lettres «DL» dans OWL-DL reflètent sa correspondance aux logiques de description. OWL-DL et le OWL-Lite sont décidables tandis que OWL-full ne l'est pas.

18. Les relations sont appelées parfois aussi des rôles.

$\forall x(\texttt{DELETE\_ACTION}(x) \Leftrightarrow (\texttt{ACTION}(x) \wedge \forall y(\texttt{concerns}(x,y) \Rightarrow \neg\texttt{OWNER\_DOCUMENT}(y))))$
$\forall x(\texttt{OWNER\_DOCUMENT}(x) \Rightarrow \texttt{PRIVILEGE}(x))$
$\forall x(\texttt{DOCUMENT}(x) \Rightarrow \exists y(\texttt{grant}(x,y) \wedge \texttt{PRIVILEGE}(y)))$

Contrairement aux règles déductives, les ontologies sont considérées souvent comme étant plus intelligibles pour les tâches de modélisation des connaissances et moins intuitives pour les tâches d'interrogation. C'est d'ailleurs une des raisons pour laquelle certains travaux proposent une combinaison des deux formalismes (e.g. [Ros06] ou encore [GHVD03]) pour prendre en compte au mieux les aspects modélisation et interrogation).

3.4 Notion de Systèmes à Base de Traces modélisées

Selon Charlet [Cha02], *« les Systèmes à Base de Connaissances (SBC) manipulent des représentations symboliques selon des prescriptions formalisées lors de la modélisation des connaissances. Ces représentations s'expriment à l'aide de primitives qui renvoient à des notions du domaine, en leur empruntant leur libellé ou terminologie : les primitives sont les termes du domaine ...»*. Pour concevoir un SBC, il faut donc tenir compte de la manière dont des utilisateurs s'approprient et attribuent du sens en modélisant les représentations comme des expressions de connaissances. Cette modélisation selon [Cha02] repose sur le contexte d'usage du SBC et son intégration dans un système de pratiques où il prend sens et justification.

Le contexte d'usage des SBC considéré dans cette thèse concerne les pratiques relatives à l'exploitation des traces à des fins de personnalisation immédiate, d'analyse *post-hoc*, etc. Dans ce contexte, les SBC visés sont très particuliers puisqu'ils considèrent comme représentation principale de connaissances les traces d'interactions. Leur principale fonction est de permettre de décrire différentes traces comme des représentations de connaissances sur lesquelles il est possible d'appliquer des requêtes et des règles de transformations pour déduire de nouvelles traces.

Comme nous l'avons décrit auparavant, les représentations courantes des connaissances dans les SBC, que ce soit par des règles ou des ontologies ou une combinaison des deux, donnent une impression assez « passive » des exploitations qui peuvent être mises en œuvre avec ces SBC. Certaines informations peuvent être inférées et obtenues par un simple accès aux connaissances représentées dans le SBC. Cependant, beaucoup de sources de connaissance comme les traces évoluent, dans le sens où elles changent avec le temps (au fur et à mesure de leurs productions) et requièrent des manières d'inférer qui prennent en compte cette part essentielle de n'importe quel système permettant un traçage. Traiter des traces qui changent au fil du temps et permettre la réalisation d'opérations communes de ces systèmes exploitant des traces nécessite une nouvelle classe de SBC.

La notion de « Systèmes à Base de Traces modélisées (SBT) » désigne une classe particulière de SBC. Les SBT partagent avec ces systèmes d'une part la nécessité de modéliser les connaissances dont est constituée la trace moyennant des représentations de l'interaction de

l'utilisateur exprimées à l'aides de primitives relatives à l'environnement informatique tracé. Et d'autres part, la nécessité de manipuler ces traces pour inférer de nouvelles connaissances, i.e., de nouvelles traces. Si les ambitions inférentielles d'un SBC consistent à raisonner automatiquement sur une représentation formelle, volontaire et suffisante du monde, les SBT s'inscrivent plutôt dans le cadre d'une inférence basée sur des traces d'interactions, inscrites automatiquement par ce système, mais évoluant dynamiquement durant l'activité. En effet, dans les SBT, les traces représentent l'ensemble de connaissances porteuses de sens, i.e., ayant une sémantique interprétable par la machine avec la particularité que ces connaissances s'inscrivent dans des traces pouvant être infinies, puisqu'elles sont induites par une utilisation et interaction continue avec le système considéré. Les principales différences entre les SBC et les SBT résident dans les points suivants :

- la particularité des traces considérées dans les systèmes à base de traces est que les observés sont reçus au fil du temps d'une manière semblable aux flux de données, tandis que dans un système à base de connaissances tous les faits sont disponibles à la fois et généralement stockés dans une base de faits.
- le flux des observés dans les traces n'est pas borné dans le futur, et est potentiellement infini, alors que les bases de faits sont limitées. Ceci a des conséquences particulières pour les dispositifs de recherche non-monotone comme la négation ou l'agrégation.
- les relations entre les observés tels que l'ordre temporel ou la causalité jouent un rôle important pour l'interrogation et/ou transformation des traces. Dans les SBC, les relations entre les faits font généralement partie des faits (e.g., les références à des clés étrangères dans une base de données déductive).
- le timing des réponses doit être considéré lors de l'interrogation des traces : les requêtes sont évaluées en permanence et de manière continue sur le flux d'observés dans la trace pour générer des réponses à des moments différents. Ces réponses peuvent être utilisées pour construire de nouvelles traces. En général, ces constructions de nouvelles traces sont sensibles à l'ordre des observés, ce qui rend la transformation d'une trace vers une autre sensible au moment de l'élaboration des réponses.

Ces divergences montrent bien les limites actuelles des SBC pour la manipulation de traces et rendent les langages d'interrogation et les moteurs d'inférence traditionnels non adaptés à la tâche d'interrogation et de raisonnement sur les traces. Elles engendrent ainsi le besoin de disposer d'un langage et moteur d'inférence spécifique aux traces. Avant de définir formellement ces langages (dans le chapitre 4), nous définissons le fonctionnement, l'architecture, les notions et les services de base des SBT.

3.5 Fonctionnement d'un Système à Base de Traces modélisées

Dans son architecture classique, un SBC comprend : (i) une base de connaissances relatives à un domaine d'application, (ii) une base de faits contenant des faits ou données caractérisant le problème courant, (iii) un programme souvent appelé *moteur d'inférence*, qui manipule les

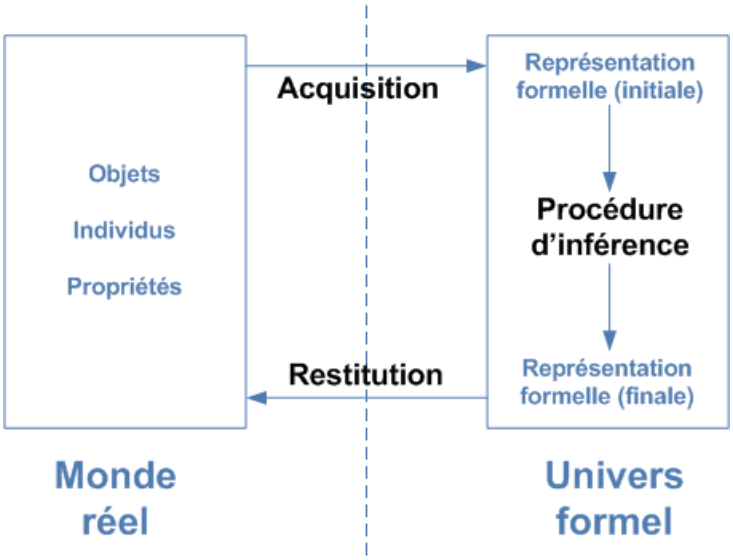

FIGURE 3.2 – La représentation et manipulation des connaissances dans un système à base de connaissances. [Kay97]

bases, recherche les connaissances adéquates et mène un raisonnement (suite d'inférences) pour résoudre le problème courant. Un SBC met en œuvre un processus décrit par la figure 3.2 :

- l'acquisition permet d'associer des structures aux éléments du domaine et du problème considéré ;
- les structures sont ensuite représentées sous forme d'expressions symboliques (formules bien formées) à l'aide d'un langage de représentation des connaissances. Parmi ces expressions, certaines sont permanentes, d'autres sont temporaires. Les expressions permanentes représentent la connaissance dont dispose le système sur le domaine étudié et peuvent être considérées comme un modèle opérationnel et utilisable du domaine, par exemple une ontologie. Les expressions temporaires contiennent les informations sur le problème particulier à résoudre ;
- les expressions symboliques sont combinées entre elles pour produire, par application de règles d'inférence, de nouvelles expressions symboliques ;
- les nouvelles expressions symboliques sont interprétées et restituées dans le cadre de l'univers du problème considéré. Elles apportent des éléments de solution au problème où elles sont évaluées en terme de validité.

Concevoir un SBC consiste ainsi à modéliser des connaissances propres à un domaine à l'aide d'un formalisme de représentation puis à manipuler les connaissances par l'intermédiaire de règles d'inférence pour résoudre les problèmes posés [Kay97].

Comme étant une classe particulière des SBC, les SBT fonctionnent selon le schéma décrit par la figure 3.3 :

- l'acquisition des traces consiste à collecter une trace qualifiée de *première* décrivant les interactions de l'utilisateur et le système observé. Une ou plusieurs sources de collecte

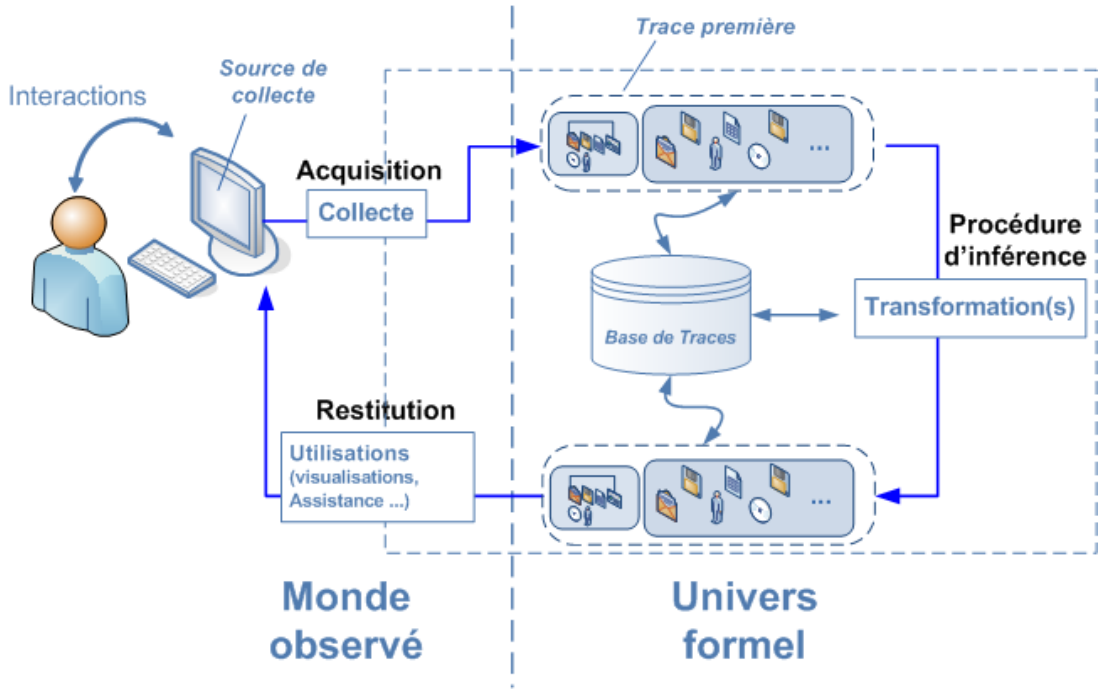

FIGURE 3.3 – Fonctionnement générale d'un Système à Base de Traces modélisées

(e.g., capteur logiciel, keylogger, etc.) sont utilisées pour obtenir cette trace.
- la trace première peut être transformée pour produire, par application de règles de transformation, de nouvelles traces.
- les traces transformées peuvent faire l'objet de restitutions dans l'univers observé en permettant des visualisations pouvant être interprétées par les utilisateurs concernés (l'utilisateur tracé, l'utilisateur observant l'activité, etc.)

Concevoir un SBT consiste ainsi à modéliser puis représenter des traces issues de l'observation d'un activité à l'aide d'un formalisme de représentation puis à transformer les traces par l'intermédiaire de règles de transformations pour répondre à un besoin spécifique. Le SBT permettra ainsi de fournir des services d'inférence à l'aide de règles de transformations raisonnant sur un ensemble de traces. Ces services d'inférence se baseront sur des représentations de modèles de traces, et des langages pour spécifier les requêtes et les transformations sur les traces.

3.6 Concepts relatifs aux Systèmes à Base de Traces modélisées

Si on considère les traces comme des inscriptions de connaissances (manipulées sous une représentation donnée), il convient à notre sens de mettre l'accent sur le type de SBC soutenant leurs exploitations, c'est-à-dire les systèmes à base de traces. Notre objectif est alors de définir un cadre conceptuel adéquat pour cette classe de système. La définition d'un tel cadre conceptuel dépend essentiellement de la définition précise des différents concepts requis pour considérer les SBT et les services permettant de représenter les traces ainsi que les traitements pouvant s'y

appliquer. Il s'agit dès lors de définir le plus précisément possible la notion de *trace « modélisée »* en s'appuyant sur les invariants repérés dans les usages de traces. Cette notion de *trace modélisée* nous permettra de considérer de manière unifiée les différentes traces étudiées, permettant aussi par la suite de préciser les différents services requis pour son exploitation. Ceci permettra de préciser l'architecture des systèmes à base de traces modélisées.

De manière informelle et générale, nous définissons le Système à Base de Traces comme *un système informatique permettant et facilitant l'exploitation des traces.* Il est composé de différents composants comme le montre la figure 3.6.

Avant de détailler les différents modules d'un tel système, il est important dans un premier temps de définir les notions de trace et de modèle de trace, deux notions essentielles pour tout système utilisant les traces. Nous présentons ensuite la notion de SBT, et précisons les étapes de collecte et de transformation. Une vue d'ensemble des utilisations générales des SBT dans le contexte des EIAH permettra d'en définir les fonctionnalités.

3.6.1 Trace et Observés

Comme nous l'avons évoqué, dans sa définition la plus générale, « une trace est une chose ou une suite de choses laissées par une action quelconque et relatives à un être ou un objet ; ce qui subsiste d'une chose passée ». Dans notre contexte, une trace numérique est issue de l'observation d'une activité, elle représente une signature du processus interactionnel système-utilisateur.

Définition 1 (Trace)**.** *On appelle trace une collection d'observés temporellement situés. On dénote par observé toute information structurée issue d'une observation.*

Une trace numérique est composée d'objets qui sont situées les uns par rapport aux autres parce qu'on les observe et qu'on les inscrit sur un support. Cela signifie qu'une trace est explicitement composée d'objets arrangés et inscrits par rapport à une représentation du temps de l'activité tracée (Figure 3.4). L'arrangement peut être séquentiel explicite (chaque observé est suivi et/ou précédé par une autre) ou découler du caractère temporel des objets de la trace.

Un observé est temporellement situé dès le moment où il est associé à un élément de la représentation du temps de la trace à laquelle il appartient. Dans le cas où cette représentation temporelle est précise et relative à un calendrier donné, l'observé pourra être associé à un instant ou à un intervalle de temps. Dans ce cas, on pourra s'intéresser aux relations chronologiques entre observés (Figure 3.4b et 3.4c) ; Dans le cas où la représentation du temps est une séquence, l'accent sera mis sur la succession ou la précédence des observés sans temps chronologique (Figure 3.4a). La structure d'un observé peut renvoyer à d'autres observés via des relations (exemple de R1, R2 et R3 dans la Figure 3.4).

Un observé est un élément constitutif d'une trace. Tout observé est temporellement situé dans la trace qui le contient, il possède un type défini par le *modèle de trace* et peut être *en relation* avec d'autres observés de la même trace.

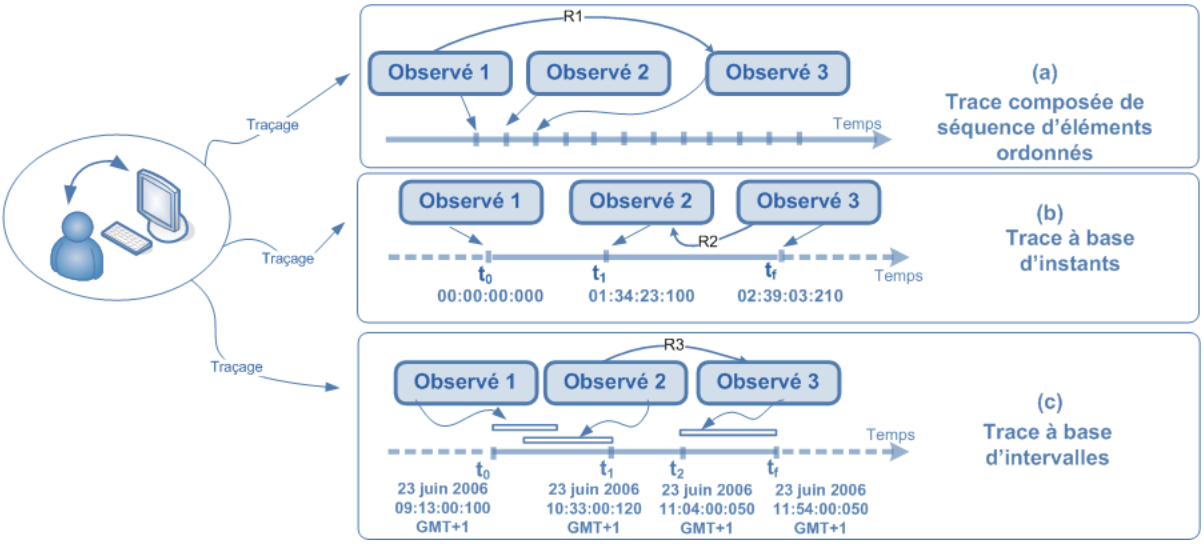

FIGURE 3.4 – Trace, observés et extension temporelle de l'activité

3.6.2 Modèle de Trace et Trace Modélisée

De manière informelle, nous définissons un modèle de trace comme le vocabulaire de la trace qui décrit de manière symbolique les objets qui en font partie. Quel que soit le niveau d'abstraction des éléments de la trace (i.e. leur proximité des événements informatiques), le modèle de trace vient préciser comment il est possible de les comprendre et de les utiliser. Un modèle de trace peut se limiter à une description des classes d'objets, mais peut aussi prendre en compte les types de relations.

Définition 2 (Modèle de trace)**.** *Un modèle de trace est une spécification formelle d'une structure d'observés décrivant comment sera représenté le résultat de l'observation d'une situation, sous la forme d'observés et de relations.*

Une trace modélisée (M-Trace dans la suite) donc est toujours associée à un modèle de trace définissant les éléments qui la composent : observés et relations. Un modèle de trace peut être implicite – c'est-à-dire uniquement présent implicitement dans le code de l'outil l'utilisant – par exemple le Common Log Format est le modèle des traces du serveur Apache. Il peut au contraire être explicite – c'est-à-dire formalisé suffisamment pour permettre l'échange et la réutilisation de traces (e.g. un schéma XML décrivant et contraignant le contenu d'une trace XML).

Définition 3 (Trace Modélisée)**.** *On appelle trace modélisée toute trace issue d'un processus de collecte, composée d'observés temporellement situés, et conforme à un modèle de trace.*

La figure 3.5 montre un exemple de modèle simple représentant les traces issues de Géonote[19]. Géonote est un EIAH destiné à l'enseignement des sciences de la Terre dans l'enseignement secondaire. Il permet d'accéder à des données géo-référencées sur une carte. L'élève

19. http ://praxis.inrp.fr/praxis/projets/geomatique/geonote/

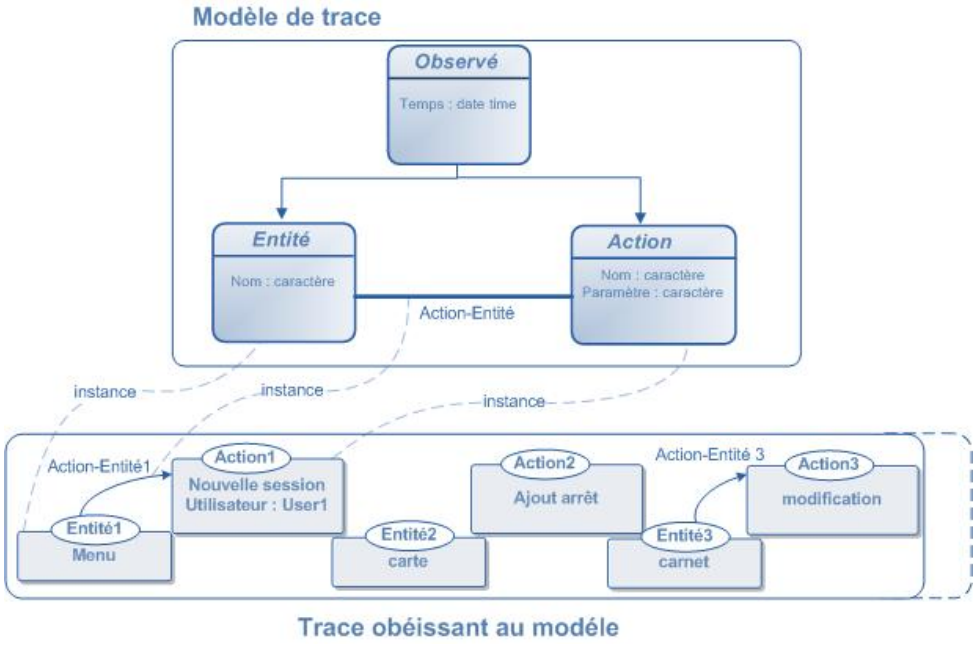

FIGURE 3.5 – Une partie de la M-Trace de l'EIAH Géonote

peut superposer différents types de cartes, réaliser des mesures, zoomer sur un détail, rechercher des données indexées, annoter ces données, prendre des notes, accéder à de la documentation. Le modèle de traces décrit simplement les observés de la trace qui peuvent être des `action` ou `entité`. Les actions sont par exemple : *ajout, arrêt, modification, nouvelle session* et les entités : *menu, carnet, carte, etc.* La trace montre que l'utilisateur a utilisé le menu en cliquant sur l'action *nouvelle session*, puis l'entité *carte* et a effectué l'action *ajout arrêt*. Le modèle de la trace de l'exemple présenté est simple mais des modèles plus complexes et plus spécifiques (proche de l'activité) peuvent être élaborés pour décrire d'autres traces.

3.7 Architecture des Systèmes à Base de Traces Modélisées

Les concepts de trace et de modèle de traces définis dans la section précédente constituent le noyau du cadre conceptuel que nous proposons. Un système informatique implémentant ces deux notions est un système à base de traces modélisées qui offrira divers services de manipulation de traces que nous décrivons dans la suite.

Définition 4 (Système à Base de Traces modélisées)**.** *On appelle système à base de traces modélisées tout système informatique dont le fonctionnement implique à des degrés divers la gestion et la transformation de traces modélisées explicitement en tant que telles.*

La figure 3.6 montre l'architecture d'un SBT. Un SBT offre abstraitement trois services de base : un service de collecte permettant de produire une trace qualifiée de *première* à partir de sources de collecte actives (e.g., un fichier log, etc.) et/ou passives (e.g. détecteur de frappes

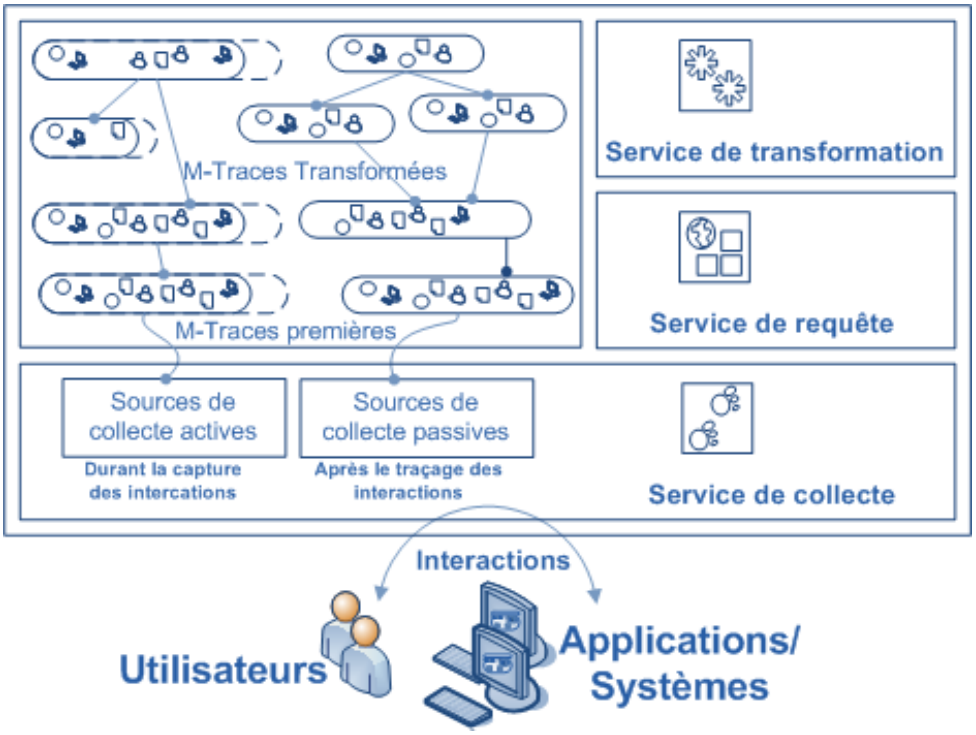

FIGURE 3.6 – Architecture générale d'un Système à Base de Traces

clavier, etc.) ; un service de requête permettant l'interrogation ponctuelle et continue des traces présentes dans le système ; et enfin un service de transformation permettant la création de trace qualifiée de *transformée* à partir d'une ou plusieurs traces.

Le cadre conceptuel des SBT définit la notion de système à base de M-Traces indépendamment de toute implémentation. L'architecture définit les composants et les services principaux pour construire un ou plusieurs systèmes informatiques à base de traces qui permettent de gérer, transformer des M-Traces ayant des modèles explicites. Nous allons à présent décrire les services de collecte et de transformation et les définitions associées.

3.8 Services des Systèmes à Base de Traces Modélisées

Collecte de traces modélisées

La collecte permet de mettre en place l'observation de l'utilisation d'un système à partir de sources de collecte. Elle consiste à transformer des informations générées par l'interaction utilisateur-système dans des sources de collecte actives pendant l'activité ou dans des sources de collecte passives récupérées *a posteriori* de l'observation en une M-Trace première du SBT (figure 3.7).

Définition 5 (Source de Collecte)**.** *On appelle source de collecte tout flux d'information à partir duquel il est possible de mettre en place un processus de collecte de traces pour un système à base*

de M-Traces.

Définition 6 (Collecte de M-Traces)**.** *On appelle collecte le processus qui consiste à créer une trace première dans un système à base de M-Traces et à y ajouter des observés à partir d'une ou plusieurs sources de collecte.*

La collecte de M-Traces est le processus qui consiste à exploiter de façon automatique, semi-automatique ou manuelle un ensemble de sources de collecte pour construire une M-Trace dans le système à base de M-Traces. La construction d'une telle M-Trace nécessite la construction d'une collection d'observés temporellement situés associée à un modèle de M-Traces. Une M-Trace issue de la collecte est appelée M-Trace première du SBT, car c'est la première à être manipulable dans ce système à l'issue du processus de collecte (Figure 3.7).

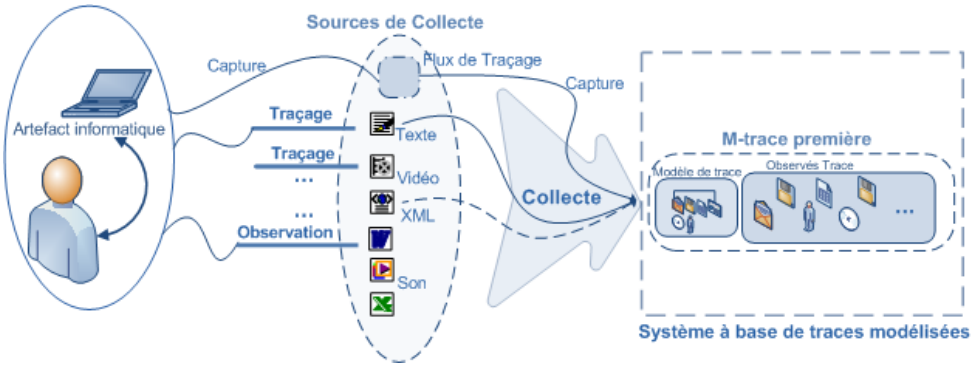

FIGURE 3.7 – Collecte de M-Trace

Il existe deux techniques de traçage dans les EIAH. La première concerne les traces générées en interne par l'EIAH lui même, qui embarque un module de traçage [20], qui peut être paramétrable. Ce module pensé par les concepteurs de l'EIAH donne des traces plus ou moins pertinentes au regard des fonctionnalités de l'EIAH (e.g. des éléments proches du scénario). La deuxième technique consiste à instrumenter l'EIAH par des logiciels de traçage externes scrutant et espionnant « tout » ce qui se passe (espions informatiques, Keylogger [21], etc.). Les traces générées par cette technique sont, la plupart du temps, inexploitables dans leur état brut initial. Une des raisons vient du fait qu'elles retiennent les informations [22] sans intention explicitée, et sont trop/pas assez riches en détails, trop/pas assez proches des événements informatiques et ne permettent presque jamais et sans ambiguïté [23] de comprendre l'activité observée.

20. Si l'EIAH est une application Web, on pourra récupérer, en plus de ce que le module de traçage, les traces laissées traditionnellement par le navigateur client et le serveur web.

21. Applications installées sur des ordinateurs enregistrant toute frappe clavier et clics souris dans un fichier trace.

22. Conséquence du syndrome des informations manquantes, on trace et on garde toutes les informations possibles.

23. Déterminer la sémantique d'une trace brute est d'une part difficile à faire manuellement par un humain et d'autre part impossible à faire de manière directe et automatique puisque nécessitant des connaissances spécifiques.

Le processus de collecte réduit en partie cette ambiguïté en prenant en compte différents sources de collecte issues de différentes techniques de traçage afin d'obtenir une trace première conforme à un modèle.

Base de M-Traces et transformation de M-Traces

Définition 7 (Base de M-Traces)**.** *On appelle base de M-Traces d'un SBT l'ensemble des M-Traces qui sont manipulées par le système à base de M-Traces.*

La trace obtenue à l'issue de la collecte est qualifiée de M-Trace première. Celle-ci n'est cependant pas toujours exploitable directement, et il faut parfois passer par une ou plusieurs transformation(s) pour atteindre une trace d'un niveau d'abstraction cohérent avec l'activité, i.e., significatif pour l'utilisateur.

Les transformations sont mises en œuvre dans SBT moyennant des règles de transformation. Ces règles permettent, à partir d'une ou plusieurs M-Traces, de déduire et construire de nouveaux observés. Le SBT permet de garder la cohérence entre les traces et leurs modèles lors d'une exécution de règles de transformation.

Il existe deux modes d'application des transformations (i.e. des règles de transformation) sur les M-Traces :

- *Allo-transformation* permet, à partir d'une M-Trace T_1 (ou plusieurs M-Traces), la création de nouveaux observés dans une nouvelle M-Trace T_2. Les M-Traces sources et la M-Trace produite ont des modèles de traces séparés et le plus souvent différents.
- *Auto-transformation* permet, à partir d'une M-Trace T_1 la création de nouveaux observés dans la même M-Trace T_1. Un seul modèle de trace est considéré dans cette transformation.

De manière informelle, une allo-transformation d'une M-Trace T_1 désigne la création d'une nouvelle M-Trace T_2 dont les différents éléments ont été inférés et calculés à partir de la M-Trace T_1 moyennant des règles de transformation. Ces règles permettent, à partir d'une ou plusieurs M-Traces, de déduire et construire une trace dite *transformée*. Le SBT garantit la cohérence entre les traces et leurs modèles lors d'une exécution de règles de transformation.

Définition 8 (Allo-transformation de M-Traces)**.** *On appelle allo-transformation de M-Traces tout processus qui transforme une ou plusieurs M-Traces gérée par un système à base de M-Traces en une nouvelle M-Trace gérée par le même système. Les M-Traces premières d'une base de M-Traces d'un SBT sont les seules M-Traces non transformées de ce SBT.*

Les auto-transformation est appliquée sur la même M-Trace. i.e., les observés sources et cible appartiennent à la même trace. Par définition (définition 8), les M-Traces premières ne peuvent subir des auto-transformations.

Définition 9 (Auto-Transformation)**.** *On appelle auto-transformation d'une M-Traces tout processus qui transforme les observés de la M-Trace en nouveaux observés appartenant à la même M-Trace.*

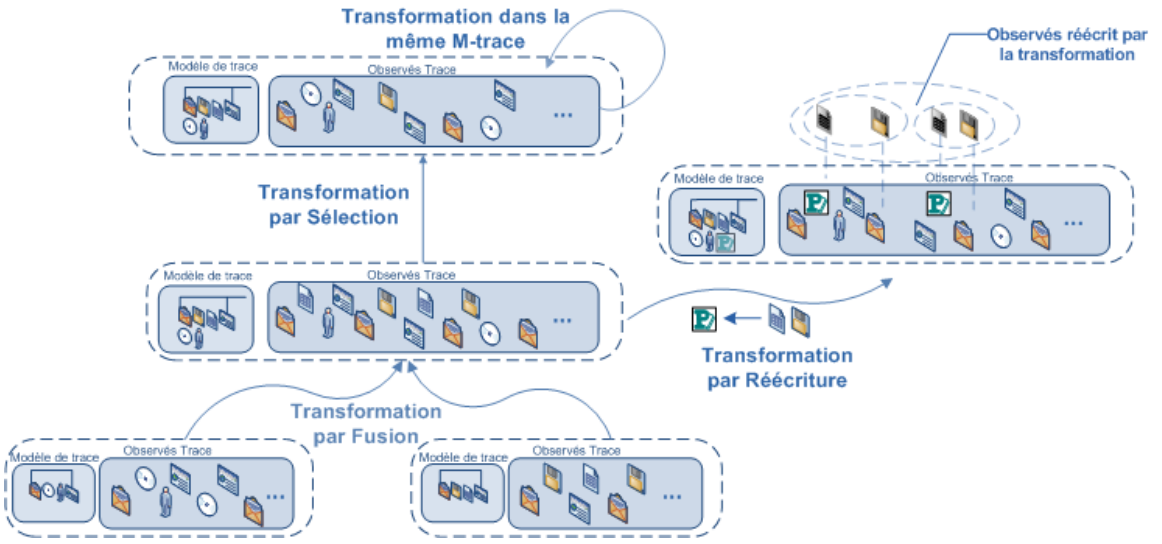

FIGURE 3.8 – Exemple de transformations dans les SBT

L'application de règles de transformation permet de composer la trace résultat *transformée* en manipulant les observés de la trace source. Les transformations peuvent consister à faire des copies, des modifications des observés des M-Traces source pour produire la trace transformée (Figure 3.8). Pour les allo-transformations, contrairement aux auto-transformations, les M-Traces sources ne sont pas modifiées mais elles servent de *données* de départ pour la transformation. Par exemple, une allo-transformation peut être une simple sélection permettant la création d'une nouvelle trace contenant tous les observés respectant un filtre exprimée dans les règles de transformation. Elle permettra entre autres de séparer les observés pertinents qu'elle sauvegarde dans la trace transformée des éléments non pertinents contenues dans la trace source. De telles allo-transformations peuvent permettre d'extraire un sous-ensemble d'observés de la trace source en exprimant les conditions de sélection ou des contraintes sur les composants de la trace. Ces contraintes peuvent concerner les observés de la trace, leurs attributs, le temps, et/ou les relations structurelles.

Les allo-transformations peuvent être plus complexes en permettant la réécriture de motifs ou patterns (reconnus dans la trace source) dans une trace transformée. De telles transformations permettront par exemple de remplacer ou réécrire un ou plusieurs observés en un nouvel observé contenu dans la trace résultat. Ce type de transformation peut toucher une séquence d'observés ou des observés qui ne se suivent pas directement. L'expression d'une succession d'observés représente un motif ou pattern exprimant la séquence à réécrire. Les allo-transformations peuvent être appliquées sur plusieurs M-Traces pour produire une M-Trace transformée. Elles peuvent permettre de fusionner temporellement plusieurs M-Traces. De telles transformations pourront ainsi assembler et grouper les observés issues de plusieurs traces en tenant compte de la temporalité des éléments afin d'obtenir une nouvelle trace. Il est évident que ce type de transformation

sera possible si les traces fusionnées disposent de domaines temporels homogènes [24].

Les auto-transformations s'effectuent sur une seule M-Trace et permettent de construire de nouveaux observés dans la même M-Trace. Cette classe de transformation permet l'expression de transformations complexes impliquant des récursions. Les auto-transformations peuvent être plus intuitives et naturelles dans certaines exploitations et cas d'application, par exemple pour rajouter des observés de plus haut niveau liés aux observés utilisés pour leur construction [25].

3.9 Discussion

Les langages pour les SBC ne sont pas toujours exploitables directement pour la modélisation, l'interrogation et transformation de M-Traces. Ils nécessitent souvent des adaptations ou des extensions pour prendre en compte les caractéristiques des M-Traces, que ce soit en termes de modélisation ou de transformation. Nous résumons dans la suite ces caractéristiques et les problématiques engendrées par l'exploitation des M-Traces.

3.9.1 Caractéristiques des M-Traces

La mise en place des SBT nécessite des représentations de connaissances prenant en compte complètement les caractéristiques des M-Traces, en particulier :

1. le modèle de trace décrit et type les observés qui composent la trace, les relations types entres eux, ainsi que les attributs les caractérisant. Une représentation sous forme d'ontologie permet de mettre en œuvre des modèles spécifiant le typage des observés et leurs relations ;
2. une M-Trace contient une séquence potentiellement infinie d'observés tracée par une application. Cette séquence est induite par un domaine temporel dans lequel les observés sont toujours situés ;

Pour la première caractéristique, les ontologies sont sans doute le formalisme le plus adapté pour prendre en compte les aspects relatifs à la modélisation avancée des M-Traces. Pour la seconde caractéristique, elle n'est pas prise en charge par les représentations habituelles des ontologies et nécessite donc des extensions et des adaptations.

Bien que les ontologies soient considérées comme étant plus populaires que les règles pour la représentation des connaissances, elles sont moins importantes dans cette thèse. Les ontologies offrent des services de raisonnement qui sont peu adaptés aux exploitations des traces. En effet, les tâches de raisonnement telles que la subsomption (vérifiant si un concept A est un sous-ensemble de concept B) ou la satisfaction (testant si un ensemble d'axiomes peut être satisfait, i.e, s'il y a un modèle) ne sont pas bien adaptés pour l'interrogation de données [26]. En effet, les

24. Des domaines temporels permettant des translations et des conversions de temps
25. En évitant les recopies inutiles et les redondances.
26. En particulier parce qu'elles sont limitées aux relations unaires et binaires et peuvent seulement spécifier des combinaisons sous forme d'arbre de relations entre les concepts.

difficultés d'interroger des ontologies se reflètent également dans les travaux de recherche sur la combinaison des règles et des ontologies, un sujet qui a suscité beaucoup d'attention récemment (voir, par exemple, [Ros06] pour une introduction, [GHVD03] pour les premiers résultats ou encore les travaux sur SWRL [HPSB$^+$]. Les règles déductives, en raison de leur relation proche avec l'interrogation de données, sont considérées dans cette thèse comme étant plus intéressantes pour l'interrogation et la transformation des M-Traces.

3.9.2 Caractéristiques des langages pour la transformation de M-Traces

L'exigence principale requise par les usages des SBT reste l'exploitation des traces pendant et après le temps de leur collecte. Ceci est un changement particulièrement important puisqu'il nécessite des langages de manipulation de connaissances permettant des évaluations continues pour exploiter en temps réel des traces ouvertes (i.e. des séquences potentiellement infinies et non bornées dans le temps). Dans ce contexte, les langages d'interrogation et de transformation de M-Traces doivent prendre en compte un certains nombre de caractéristiques :

- Les requêtes et les transformations sur des M-Traces sont censées fonctionner en permanence et de manière continue et elles peuvent produire de nouveaux résultats à chaque nouvelle arrivée.
- Un service de collecte transmet chaque élément observé une seule fois. Comme les éléments observés sont accessibles de façon séquentielle, un observé qui est arrivé dans le passé peut être accédé directement dans la M-Trace (contrairement au flux de données traditionnels qui s'oppose à une matérialisation complète).
- Les observés arrivent en permanence dans le SBT, poussés par exemple par une source de collecte active. Le service de collecte n'a la maîtrise ni sur l'ordre dans lequel les observés vont arriver, ni sur leur fréquence d'arrivée. La fréquence d'arrivée des observés dans la M-Traces est imprévisible et varie au fil du temps.

Bien qu'il soit possible d'exprimer des requêtes ou des transformations avec des règles déductives, les évaluer de manière continue pose un certain nombre de problèmes. Nous allons illustrer un exemple simple pour montrer certaines de ces limites. Prenant par exemple une transformation notée τ_1 contenant la règle suivante R_1. La requête suivante `?-unreplied_message`$(?x, ?y)$ et la règle permettent d'obtenir les messages qui n'ont pas encore été traitées, i.e., les messages auxquels l'utilisateur n'a pas encore répondu [27]. La requête `?-unreplied_message(?x,?y)` permet d'obtenir les résultats de la transformation.

$$R_1 = \texttt{unreplied\_message}(M, U) \leftarrow \texttt{message}(M, U),\ \texttt{user}(U),\ \neg\ \texttt{response}(M, U)$$

Mettre en place des requêtes ou des transformations continues sur des M-Traces qui évoluent dans le temps à l'aide de règles déductives peut être fait par l'algorithme simpliste décrit dans

27. Évidemment, l'exemple montré décrit une réification simpliste des transformations sous forme de règles. Une translation plus sophistiquée vers Datalog a été étudiée dans notre premier travail de formalisation dans [SPC$^+$09]

la table 3.1.

Bien que simple à mettre en œuvre, cette évaluation continue présente au moins trois principales carences :

- *Des résultats non déterministes.* Les observés créés (ou retournés) par une transformation (ou une requête) dépendent du moment où cette transformation (ou requête) est exécutée. Par exemple, une transformation qui est exécutée toutes les heures peut produire des résultats différents que la même transformation exécutée une fois par jour. Cela signifie que deux utilisateurs avec la même transformation continue pourrait obtenir un ensemble de résultats différent. Ceci est dû à l'usage de la négation qui est bien connue pour être non monotone et donc non souhaitable pour des évaluations continues [GO03, TGNO92, Ara06].
- *Doublons.* Chaque fois qu'une transformation est exécutée, l'utilisateur verra tous les observés créés par la transformation, anciens et nouveaux. Les faits retournés par une transformation ne cesseront d'augmenter au fil du temps. Dans la pratique, les utilisateurs ne sont intéressés que par les observés nouvellement créés qui n'ont pas encore été retournés.
- *Inefficacité.* L'exécution de la même transformation à maintes reprises est trop coûteuse. De la même manière que la taille du résultat de la transformation augmente au fil du temps, le coût d'exécution augmente aussi. Idéalement, le coût d'exécution d'une transformation en continu doit être en fonction de la quantité des nouveaux observés, et ne dépend pas de la taille de la M-Trace.

`i := +`∞
`WHILE` $i > 0$ `DO` |
Exécute la transformation τ_1
Retourne les résultats à l'utilisateur
Attendre pendant une certaine période d
`ENDWHILE`

TABLE 3.1 – Évaluation périodique et continue d'une transformation à l'aide de règles déductives

Le problème des doublons peut être résolu en ayant un système qui se rappelle de l'ensemble des résultats qui ont été retournés. Le système pourrait alors veiller à ne retourner que les résultats qui ne sont pas dans cet ensemble. Bien que cette approche évite les doublons, il y a encore les problèmes d'efficacité. Une grande partie du coût de calcul de la transformation passe dans la création d'observés qui vont être rejetés par la suite (puisqu'ils ont déjà été construits auparavant).

Pour ce qui est du problème des résultats non déterministes, les règles déductives ne permettent pas réellement des évaluations continues. L'usage de la négation est bien connue pour être non monotone et donc non souhaitable pour une évaluation continue. En outre, considérer une seule M-Trace implique une possible récursion dans les règles. La négation combinée avec la récursivité des règles se révèle problématique. Bien que ce problème soit bien identifié et résolu

pour le cas classique des règles évaluées de manière ponctuelle (en utilisant par exemple des techniques de stratification [ABW88]), il n'est pas étudié dans le cas des évaluations continues qui traitent au fur et à mesure des M-Traces qui évoluent. En résumé, les langages pour l'interrogation et la transformation de M-Traces doivent permettre des évaluations continues et efficaces même en présence de la négation.

Le chapitre suivant présente une proposition prenant en compte ces caractéristiques. Notamment, le langage dont nous allons spécifier la syntaxe et la sémantique permet des évaluations continues monotones même en présence de la négation. Il adapte également les techniques de stratification habituelles pour le cas des évaluations continues de règles de transformation.

Chapitre 4

Cadre formel pour les langages de manipulation des traces modélisées

Sommaire

Ce chapitre a pour objectif de présenter les langages sur lesquels se fondent les systèmes à base de traces modélisées. Nous proposons d'abord une formalisation des concepts de base relatifs à la notion de M-Trace. Ensuite, partant de la définition de tels concepts, nous proposons des langages pour l'interrogation et la transformation de M-Traces de manière continue. Nous avons choisi le cadre de la théorie des modèles et de la théorie du point fixe pour spécifier la sémantique formelle des langages proposés.

La suite de ce chapitre est structurée de la manière suivante : les sections 4.1 et 4.2 décrivent respectivement les motivations pour une sémantique formelle déclarative et ses caractéristiques. La section 4.3 présente une vue générale de notre formalisation et les divers cas considérés dans notre approche. La section 4.4 décrit quelques notions de base et les assomptions assumées dans ce chapitre. La section 4.5 introduit les représentations du temps définies pour les M-Traces. La section 4.6 présente formellement respectivement la notion de modèle de trace (section 4.6.1) et de trace modélisée (section 4.6.2). La section 4.7 définit la syntaxe de notre langage. La section 4.8 présente les notions de base de notre sémantique présentée sous forme d'une théorie des modèles. Ces notions permettront de définir la sémantique des requêtes (section 4.9) et des transformations simples (section 4.10) pour des évaluations continues et ponctuelles. Enfin, la section 4.11 présente les transformations complexes impliquant la négation en définissant une théorie de point fixe associée à la théorie des modèles présentée précédemment. La section 4.12 décrit les preuves des différents théorèmes proposés.

4.1 Motivations pour une sémantique formelle déclarative pour les langages de manipulation des M-Traces

Une sémantique formelle et déclarative est une exigence pour notre langage comme elle l'est pour les langages définis pour la programmation logique, requêter le web, interroger des bases de données, etc. Une sémantique déclarative relie la syntaxe d'un langage à des objets mathématiques et des expressions qui en capturent le sens recherché et attendu. Adoptant la rigueur des mathématiques, la sémantique formelle permet d'éviter les ambiguïtés pouvant exister dans la description informelle de la sémantique d'un langage [McD86]. Par ailleurs, une sémantique déclarative constitue une référence pour les développeurs implémentant le langage et aide grandement aux efforts de standardisation.

Une sémantique déclarative met l'accent sur ce que les expressions du langage signifient sans conclure sur comment doit être évaluée une expression. La définition d'une sémantique déclarative fournit ainsi une base pratique et un support pour prouver la conformité et l'exactitude de différentes sémantiques opérationnelles puisqu'il existe une myriade de moyens équivalents pour évaluer une requête donnée, et donc, un grand nombre de sémantiques opérationnelles. Comparer et valider différentes sémantiques opérationnelles revient donc à réfléchir en terme d'équivalence entre différents calculs. Il est plus simple de considérer la justesse d'une sémantique opérationnelle en s'appuyant sur une sémantique déclarative, puisque nous pouvons ainsi nous concentrer et réfléchir sur les propriétés du résultat d'un seul calcul. En outre, utiliser une sémantique déclarative pour prouver la justesse d'une méthode d'évaluation est particulièrement utile dans la recherche de techniques d'optimisation.

Une sémantique formelle déclarative donne aussi le moyen d'étudier les propriétés d'un langage de manière générale ou plus spécifique, par exemple dans notre cas pour une certaine classe de requêtes ou pour une transformation particulière. D'ailleurs, il est assez commun dans les langages de requêtes d'identifier certaines classes de requêtes à des classes de complexité associées. Dans le cas particulier d'un langage d'interrogation et de transformation de M-Traces, on peut se poser la question de savoir si un langage permet effectivement de prendre en compte une M-Trace ouverte à de nouveaux ajouts d'observés et non bornée dans le temps, pouvant être potentiellement infinie. Autrement dit, si toutes les requêtes et les transformations que nous pouvons exprimer dans ce langage peuvent être évaluées continuellement sans attendre indéfiniment l'arrivée d'observés et sans obligation de fermeture de cette M-Trace.

Enfin, facile à comprendre et mathématiquement plus « esthétique », une sémantique déclarative constitue une indication de bonne conception et aider grandement à identifier les incongruités et les défauts du langage spécifié.

4.2 La sémantique requise et désirée pour la manipulation de M-Traces

Notre objectif dans cette thèse et plus particulièrement dans ce chapitre est de définir une sémantique qui soit intuitive pour la manipulation de traces ouvertes et explicitement modélisées, pouvant évoluer et s'incrémenter dans le temps. Une telle sémantique devra prendre en compte des descriptions des hiérarchies de types (type d'observés et type de relation) (i.e. des ontologies), ainsi que des observés ayant des attributs, des relations les uns avec les autres, en plus des valeurs de temps explicitement associées.

La sémantique doit prendre en compte des requêtes sur les traces permettant de retrouver des données de la trace moyennant des *substitutions* associant à des variables les observés de la trace. Elle doit prendre en compte les règles de transformation pour déduire et construire de nouveaux observés de manière continue.

Notre posture dans cette thèse est de considérer les M-Traces dans le contexte des SBC. Il est donc naturel d'appliquer et d'adapter les formalismes traditionnels issus des langages de représentation de connaissances aux langages de manipulation de M-Traces. Cela permet de tirer partie, parfois avec des changements importants, des nombreux résultats issus du champ théorique des bases de données déductives et de la programmation logique, mais permet aussi de mettre la lumière sur les nouveaux concepts qui sont nécessaires et spécifiques aux M-Traces.

Les langages exprimant des requêtes et des règles de déduction, comme Datalog [AHV95] fondent leur sémantique sur une théorie des modèles. Une théories des modèles est considérée comme très déclarative, en particulier depuis qu'elle est définie récursivement sur la structure d'une formule, permettant ainsi d'envisager les sous-formules de manière isolée. Les théories des modèles sont également théoriquement bien fondées et relativement faciles à comprendre. En raison de leur lien avec la logique du premier ordre, les théories des modèles sont également intéressantes pour l'extension d'un langage à base de règles vers des aspects plus avancés de représentation de connaissances pour prendre en compte les spécificités des M-Traces.

Pour ces raisons, nous proposons d'utiliser une approche théorique basée sur les modèles pour spécifier la sémantique déclarative de notre langage d'interrogation et de transformation. Cependant, il y a quelques différences importantes entre les systèmes traditionnels d'interrogation de connaissances et les langages de manipulation de M-Traces. Ces différences rendent certaines adaptations des théories traditionnelles existantes nécessaires :

- les résultats de l'évaluation d'une requête ou d'une transformation sont situés dans le temps ;
- les conditions sur les attributs des observés, les relations temporelles absolues entre observés ont une interprétation fixe ;
- plus important encore, la sémantique doit prendre en compte des M-Traces infinies, et ouvertes à de nouvelles adjonctions d'observés.

Ce dernier point décrit la plus importante différence entre notre langage pour les M-Traces et les langages classiques pour déduire et interroger des connaissances. Cependant, cette différence

n'est pas inhérente à l'approche utilisée pour donner une sémantique déclarative. En effet, nous aurons donc à définir (et à prouver) un théorème explicitant que la sémantique déclarative de notre langage satisfait réellement cette exigence.

4.3 Vue globale de notre cadre formel

Le cadre formel présenté formalise la notion de M-Trace et les langages de requêtes et de transformations associés à cette notion. Après la définition formelle de modèle de trace et de M-Trace, notre cadre formel présente la syntaxe des *patterns* et des *templates* pour exprimer des requêtes et des transformations de M-Traces. De manière informelle, la notion de *patterns* et *templates* représentent respectivement les moyens d'obtenir et de construire les observés de la M-Trace. Par la suite, une sémantique déclarative des langages présentés sera abordée selon différentes dimensions.

4.3.1 Dimensions considérées dans notre étude

Nous avons considéré trois dimensions lors de la formalisation de notre langage de requêtes et de transformations de M-Traces noté $TQML$ (Transforming and Querying Modelled traces Langage) [28].

La première dimension concerne l'*expressivité du langage étudié*. Dans un premier temps, nous allons considérer une classe de notre langage n'utilisant pas la négation notée $TQML^{-}$. Les requêtes et les transformations exprimées dans ce langage seront qualifiées de simples. Par la suite, nous considérerons le cas plus expressif et général du langage, qu'on nommera $TQML^{w}$ utilisant la négation (w pour le mot clé `without`). Les requêtes et les transformations exprimées dans le langage $TQML^{w}$ seront qualifiées de complexes.

La seconde dimension concerne le *type de la transformation étudiée* : le cas des auto-transformations dans une même M-Trace et les allo-transformations d'une M-Trace pour construire une nouvelle M-Trace.

La troisième dimension concerne le *mode d'évaluation de la transformation* : l'évaluation ponctuelle et l'évaluation continue.

Ces dimensions ont pour objectif d'étudier de manière séparée les diverses facettes de notre langage et permettent d'identifier les situations complexes pour l'évaluation des transformations. Elles permettent également de traiter de manière graduée les différents problèmes qui peuvent se poser lors de la spécification des langages pour les M-Traces. Mettre en œuvre une étude en prenant en compte ces dimensions offre certains avantages :

– Ces dimensions permettent de traiter de manière ciblée le problème de la monotonie des transformations. La complexité des transformations vient du fait que la négation est non monotone et donc problématique pour les techniques d'évaluation continues [Ara06, GO05,

28. On notera QML le sous langage de $TQML$ permettant de décrire les requêtes.

GO03]. L'étude des langages avec et sans la négation permet de bien identifier les aspects problématiques liés à la monotonie des transformations.

– Ces dimensions permettent d'étudier chaque cas indépendamment des autres, ce qui facilite l'extension et le positionnement aux travaux existants. Par exemple, la complexité de certaines transformations est plus importante dans le cas d'une auto-transformation d'une M-Trace puisque des récursions peuvent se produire. Ce cas peut être rattaché aux travaux sur les règles combinant la récursion à la négation. En effet, ce thème ayant été abondamment étudié par la communauté des bases de données déductives notamment pour le langage Datalog [AHV95, Chap. 14] permet d'offrir un certain nombre de résultats théoriques intéressants que nous pouvons réutiliser ou étendre comme par exemple la stratification des règles.
– Enfin, les définitions et les propositions sur certains cas peuvent être considérées comme une base qui pourra être étendue vers d'autres cas. Par exemple, la sémantique des allo-transformations complexes peut être considéré comme une simplification de la sémantique des auto-transformations complexes. En outre, ceci permettra aussi une meilleure compréhension de la sémantique et facilitera les définitions, les comparaisons et les explications.

L'ensemble des cas considérés dans cette thèse est résumé dans le tableau 4.1. Nous donnons dans la suite un aperçu plus détaillé des cas étudiés.

Langage de transformation	**type de transformation**	**mode d'évaluation**	**étude réalisée**
$TQML^{-}$	Auto-transformation	Continue	Cas étudié faisant partie des auto-transformations complexes définies dans la section 4.11
		Ponctuelle	Cas trivial en utilisant les définitions de la section 4.11 et la relation de satisfaction définie dans la section 4.10.
	Allo-transformation	Continue	Cas étudié dans la section 4.10
		Ponctuelle	Cas étudié dans la section 4.10
$TQML^{w}$	Auto-transformation	Continue	Cas étudié dans la section 4.11
		Ponctuelle	Cas non étudié puisque trivial en se fondant sur les sections 4.10 et 4.11
	Allo-transformation	Continue	Cas non étudié mais qui peut être considéré comme trivial en comparaison aux auto-transformations.
		Ponctuelle	Cas trivial en utilisant les définitions de la section 4.11 et la relation de satisfaction définie dans la section 4.10.

TABLE 4.1 – Résumé des cas étudiés dans ce chapitre.

4.3.2 Aperçu de notre approche de formalisation

Concernant la sémantique déclarative des langages considérées pour l'interrogation et la transformation de M-Traces, le problème étudié dans cette thèse peut être décrit à un niveau très abstrait comme suit : étant donné τ un ensemble de règles de transformation (τ est appelé aussi transformation) et un flux d'observés issus d'une M-Trace source, nous voulons produire tous les observés nouveaux tels que dérivés par cet ensemble de règles τ. Une règle de transformation produit un ou plusieurs observés si un pattern d'observés est satisfait dans le flux entrant. L'étude des règles de transformation couvre également l'étude des requêtes et la satisfaction de patterns d'observés.

Adopter une démarche mettant en œuvre une théorie des modèles permet d'associer « vérité et conséquence logique » et « signification du langage » décrivant un monde dans ce contexte logique. Tarski [Tar33, Pop72] l'a démontré en définissant par cette approche la sémantique du calcul des prédicats. Nous utilisons cette approche pour définir la sémantique du calcul de transformations, i.e., les expressions de règles et de requêtes. La satisfaction de ces expressions est liée à une interprétation par la définition d'une relation de satisfaction notée $\mathcal{I}$ *satisfait* P (ou $\mathcal{I} \models P$)[29]. Informellement, une interprétation $\mathcal{I}$ est l'ensemble des observés (et leurs relations et valeurs d'attribut) constitué par les observés satisfaisant un pattern P (ou une règle de transformation) et donc satisfaisant les contraintes énoncées par le pattern P.

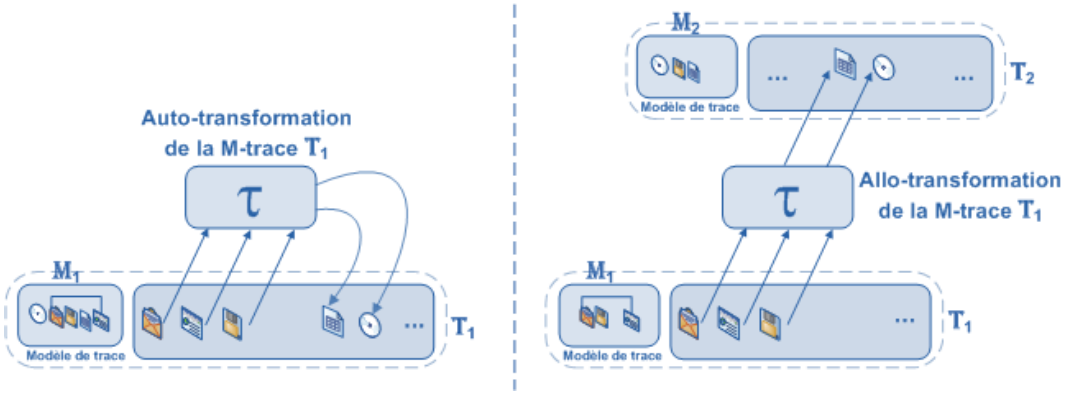

FIGURE 4.1 – Auto-transformation et Allo-transformation de M-Trace

Si on considère que chaque M-Trace est équipée de son propre modèle, il y a deux situations différentes à prendre en compte pour la satisfaction d'une transformation (figure 4.1) :

- *Cas 1 : Auto-transformation.* Le modèle de trace est unique et la transformation τ doit pouvoir être satisfaite par l'interprétation $\mathcal{I}$ (notée $\mathcal{I} \models \tau$) constituée tous les observés qui se produisent, i.e. les observés du flux entrants de la M-Trace et les observés dérivés et transformés par τ.
- *Cas 2 : Allo-transformation.* La transformation considère un flux d'observés issus de $\mathbb{T}_1$

29. C'est une simplification des définitions données dans la suite. En fait, une interprétation est l'ensemble des observés et des substitutions des variables du pattern.

et produit un flux d'observés dans $\mathbb{T}_2$. Il y a donc deux interprétations qui doivent en conjonction satisfaire τ (notée $\mathcal{I}_1, \mathcal{I}_2 \models \tau$).

Intuitivement, les interprétations intéressantes sont celles qui (1) satisfont toutes les règles d'une transformation τ à l'égard de la relation de satisfaction et (2) contiennent le flux des observés entrant et sortant. Ces interprétations sont alors appelés *modèles*.

La sémantique déclarative de notre langage est donnée sous forme d'une théorie des modèles définie récursivement sur la structure des expressions du langage que sont les patterns et les règles de transformation et adaptée aux exigences spécifiques des transformations de M-Traces, i.e., les divers aspects temporels des patterns de requête, les expressions dans les templates de règle de transformation, etc. Cette adaptation concerne notamment la relation de *satisfaction* qui prend en compte le temps (i.e. $\mathcal{I}$ *satisfait* P dans un intervalle de temps $[b, e]$ notée $\mathcal{I} \models P^{[b,e]}$) contrairement à la relation traditionnelle de *satisfaction (entailement)* considérée dans les théories plus classiques (e.g. Datalog [AHV95]).

Pour organiser notre présentation, nous utilisons donc le vocabulaire suivant :

Une *transformation* τ est constituée d'un ensemble de *règles de transformation* $\{R_i\}$. Une règle de transformation R_i est constituée d'une partie *pattern* P_i à satisfaire sur une trace pour produire les observés déclarés dans un template G_i [30] (on notera une règle $R_i = (P_i, G_i)$ comme $G_i \leftarrow P_i$). Une requête $\mathcal{Q}$ est constituée par l'expression d'un pattern P à satisfaire et de l'ensemble des variable $\mathbf{var}_\mathcal{Q}$ à retourner qui est un sous-ensemble des variables $\mathbf{var}_P$ à substituer pour évaluer cette satisfaction (une requête $\mathcal{Q} = (\mathbf{var}_\mathcal{Q}, P)$) [31]

Après avoir présenté formellement les définitions de M-Trace et la syntaxe de notre langage (respectivement dans les sections 4.6 4.7), nous en étudierons la sémantique (section 4.8). Pour cette partie nécessitant une démarche démonstrative, nous adoptons l'organisation suivante : nous donnons d'abord la définition des notions [32] de *substitution*, *interprétation* et *satisfaction* dans le cadre des M-Traces (sections 4.8.1 4.8.2 4.8.3).

Les transformations simples ($TQML^-$) et les transformation complexes ($TQML^w$) pouvant utiliser toutes les deux des patterns simples (QML^-), nous commençons par étudier la sémantique des patterns simples avec les requêtes simples (QML^-).

Par la suite, la sémantique des transformations simples (n'utilisant que des patterns simples) sera établie par un ensemble de définitions s'appuyant sur la sémantique de QML^-.

Par contre, l'étude de la sémantique des transformations complexes exige une approche plus globale puisque l'introduction de la négation pose un certains nombres de problèmes. Notamment une ambiguïté à lever dans le processus de transformation comme l'illustre l'exemple de la figure 4.2 où $\tau = \{R_1, R_2\}$. Les deux règles peuvent être lues de la manière suivante : construire un observé de type `p` (de type `q` pour R_2) quand on observe un `x` de type $\mathtt{c}_1$ et que pendant le temps de `x`, il n'y a pas `z` de type `q` (de type `p` dans R_2).

30. G également comme générateur

31. Un pattern peut également être vu comme une requêtes retournant toutes les variables substitutées.

32. Ces notions ont été volontairement définies de manière très générale pour pouvoir prendre en compte les différents cas étudiés.

Construct (w : p) On (x : c_1) and while x: without (z : q)	Construct (u : q) On (x : c_1) and while x: without (z : p)

TABLE 4.2 – Exemple d'une transformation complexe contenant deux règles

Si l'on est dans la situation d'une auto-transformation, il y a deux résultats satisfaisants mais différents sans possibilité de les arbitrer :

– résultat 1 en appliquant R_1 puis R_2 pour faire la transformation (en bas de la figure 4.2).
– résultat 2 en appliquant R_1 puis R_2 (en haut de la figure 4.2).

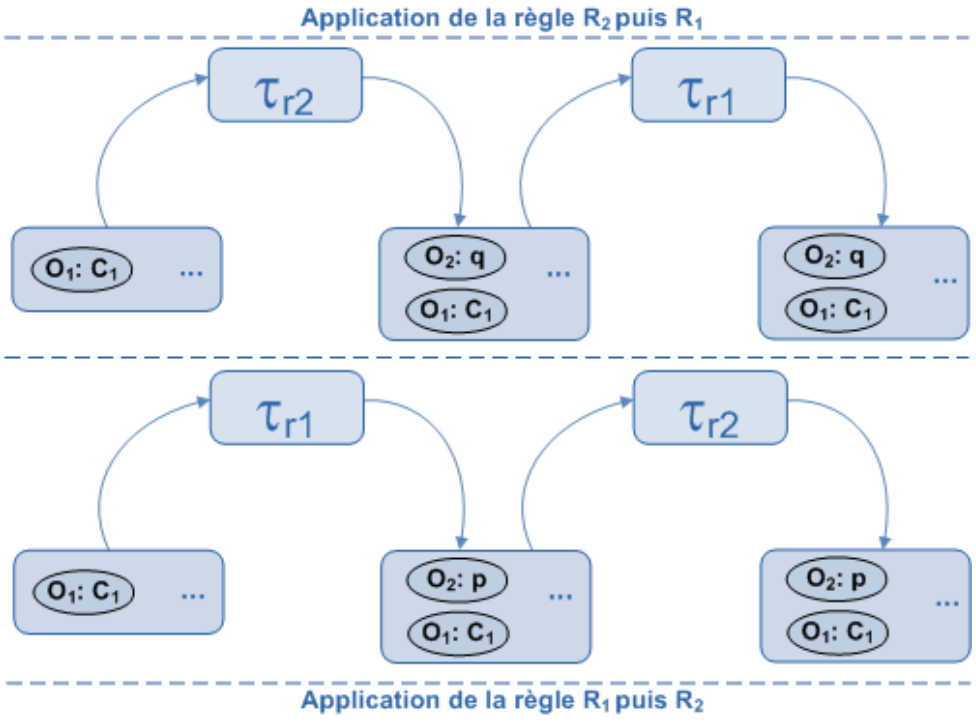

FIGURE 4.2 – Cas d'application des règles de transformation sur une M-Trace

Pour une M-Trace contenant un observé de type c_1, il y a bien deux résultats corrects impossibles à arbitrer, ce qui est précisément le cas étudié dans la théorie des modèles et résolu par la théorie du point fixe et la stratification des règles permettant de donner un ordre de préférence à l'application des règles et donc d'arbitrer.

Nous utiliserons ce principe pour proposer une sémantique du cas des transformations complexes et nous en ferons la démonstration dans le cas des transformations continues (le cas le plus difficile).

Il serait sans doute possible de considérer que les autres cas que la transformation complexe en auto-transformation et en continue en sont des cas particuliers plus simples et que si le langage

$TQML^w$ a une sémantique démontrée pour une auto-transformation en continue, alors il peut s'appliquer à tous les autres cas. Malheureusement, dans le cadre de cette thèse, nous n'avons pas eu le temps d'en faire la démonstration.

4.4 Notions de base et Notations

Nous allons maintenant commencer par définir quelques notions de base et notations utilisées dans ce chapitre.

Fonctions et relations

Une fonction $f : S \to T$ est aussi (et en premier) une relation $f \subseteq S \times T$ où les éléments de S doivent apparaitre au plus une fois. Pour des raisons de simplicité et de commodité dans les définitions, nous utiliserons les deux notations de manière équivalente, par exemple :

- $f : S \to T \Rightarrow f \subseteq S \times T$
- $f(x) = y \Leftrightarrow (x, y) \in f$

De la même manière, la fonction inverse f^{-1} est toujours définie comme suit : $f^{-1} = \{(y, x) \mid (x, y) \in f\}$ même si elle n'est pas nécessairement une fonction. Si f^{-1} est une fonction, f est bijective (et f^{-1} l'est aussi évidement).

Le domaine de f, noté $\mathbf{domain}(f)$ est le sous-ensemble de S dans lequel f est définie. Le co-domaine de la fonction f, noté $\mathbf{range}(f)$ est l'image de S par rapport à f, i.e. un sous-ensemble de T dans lequel f associe un élément de S. Plus formellement :

- $\mathbf{domain}(f) = \{x \in S \mid \exists y, (x, y) \in f\}$
- $\mathbf{range}(f) = \{y \in S \mid \exists x, (x, y) \in f\}$

f est dite totale si le $\mathbf{domain}(f) = S$, sinon elle est partielle.

Valeurs et types de données

Nous considérons un ensemble $\mathbf{V}$ de toutes les valeurs littérales (parfois appelée valeurs concrètes) d'intérêt d'un domaine d'applications. On suppose par ailleurs l'existence d'un ensemble $\mathbf{D}$ de type de données. Chaque type de données $d \in \mathbf{D}$ a un espace de valeur $\mathbf{V}_d \subseteq \mathbf{V}$. Un exemple de ces types de données sont ceux définis par XML-schemas [PMSMS09]. Pour plus de commodité, nous ferons pas de distinction dans ce chapitre entre le type de données et son espace de valeurs, et donc $v \in d$ signifiera que $v \in \mathbf{V}$ inclut dans l'espace des valeurs de $d \in \mathbf{D}$.

4.5 Représentation du temps dans les traces

De manière informelle, nous définissons une *trace* comme *une collection d'observés temporellement situés*. Un *observé* dénote toute information structurée issue de l'observation d'une interaction. Un observé est *temporellement situé* dès le moment où il est associé à un instant ou un intervalle de temps faisant partie du domaine temporel de la trace à laquelle il appartient.

Une trace numérique est composée d'objets qui sont situées les uns par rapport aux autres parce qu'on les observe et qu'on les inscrit sur un même support. Cela signifie qu'une trace est explicitement composée d'objets arrangés et inscrits par rapport à une représentation du temps de l'activité tracée. Cette arrangement peut être séquentiel explicite (chaque observé est suivi et/ou précédé par une autre) ou découler d'un estampillage temporel explicite des observés. Même si différentes traces peuvent utiliser différentes représentations de temps, du point de vue formel, les traces partagent toutes un fait commun : le temps est une collection ordonnée d'instants. Nous nommons cette représentation du temps un *domaine temporel*.

Définition 10 (Domaine Temporel)**.** *Un domaine Temporel $(\mathcal{T}, \leq_\mathcal{T})$ est ensemble d'instants totalement ordonné par $\leq_\mathcal{T}$.*

Le temps est représenté par un ensemble totalement ordonné $(\mathcal{T}, \leq_\mathcal{T})$ de points temporels ou d'instants. Noter bien que ce domaine peut être dense [33], contrairement au domaine temporel discret fréquemment considéré dans les travaux modélisant des données temporelles (comme CQL [ABW03, Ara06]). Un exemple d'un domaine temporel peut être l'ensemble des nombres réels $\mathbb{R}$ associé à son ordre usuel. Ce type de domaine pourrait représenter le nombre de secondes et leurs fractions écoulées depuis une époque comme minuit du 1 Janvier 1970.

Afin de répondre aux exigences de plusieurs applications exploitant les traces, le cadre des SBT doit être en mesure d'identifier les différents domaines temporels pouvant être requis par ces applications. Plus précisément, nous faisons la distinction entre les applications nécessitant uniquement un ordonnancement entre les instants et les applications exigeant la notion de *durée* entre les instants. Pour ces dernières, nous définissons le domaine temporel chronométrique.

Définition 11 (Domaine temporel chronométrique)**.** *Un domaine Temporel chronométrique $\mathcal{T}$ est un domaine temporelle $\mathcal{T}$ muni d'un opérateur de distance $d_\mathcal{T}$.*

Ce qui distingue un domaine temporel chronométrique c'est la mesure $d_\mathcal{T}$ qui permet de mesure la *durée* entre deux instants. Elle permet d'autre part de définir l'opérateur $- : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ permettant de mesurer la durée $a - b$ entre deux instants a, b :

$\forall a, b \in \mathcal{T},\ d \in \mathbb{R}$,

$$a - b \doteq \begin{cases} d_\mathcal{T}(a,b) & \text{si } a \leq_\mathcal{T} b \\ -d_\mathcal{T}(a,b) & \text{sinon} \end{cases}$$

$$a \oplus d \doteq b \text{ tel que } b - a = d$$

$$a \ominus d \doteq b \text{ tel que } a - b = d$$

Pour la suite, nous noterons pour simplifier dans certains cas l'ensemble $\mathbb{R}$ comme étant l'ensemble des durées $\mathbb{D}$ (e.g., `1 week 2 days`). L'addition $\oplus : \mathcal{T} \times \mathbb{D} \to \mathcal{T}$ et la soustraction

33. Un ensemble ordonné $(E, \leq)$ est dit dense en lui-même, ou plus simplement dense, si, pour tout couple (x, y) d'éléments de E tel que $x \leq y$, il existe un élément z de E tel que $x \leq z \leq y$.

$\ominus : \mathcal{T} \times \mathbb{D} \to \mathcal{T}$ des durées de ou à partir de points est définie en se basant sur l'opérateur $-$. Nous aurons en plus besoin de la possibilité de comparer deux durées ($t_1 \leq t_2$). Ceci peut être fait par la relation d'ordre $\leq \subseteq \mathbb{D} \times \mathbb{D}$. Enfin, un intervalle de temps est dans notre contexte toujours fermé représentant un sous ensemble convexe de $\mathcal{T}$ .i.e. $[d, f] = \{p | d \leq p \leq f\}$. Il peut être représenté par ses extrémités $d \in \mathcal{T}$ et $f \in \mathcal{T} (d \leq f)$. Noter que cette définition permet des intervalles de temps dégénérés comme $[p, p]$ qui se composent uniquement d'un point de temps ou instant unique $p \in \mathcal{T}$. Pour plus de commodité, nous définissons les fonctions suivantes et les relations sur les intervalles de temps :

- $[b_1, e_1] \sqcup [b_2, e_2] = [\min\{b_1, b_2\}, \max\{e_1, e_2\}]$,
- $[b_1, e_1] \sqsubseteq [b_2, e_2]$ ssi $b_2 \leq b_1$ et $e_1 \leq e_2$.

4.6 Représentation formelle des M-Traces

4.6.1 Modèle de Trace

Une trace modélisée obéit toujours à un modèle de trace, qui décrit les objets qui en font partie. Quel que soit le niveau d'abstraction des éléments de la trace, le modèle de trace vient préciser comment il est possible de les comprendre et de les utiliser. Un modèle de trace peut se limiter à une description des classes d'objets, mais peut aussi prendre en compte les types de relations. Enfin, un modèle de trace décrit toujours le domaine temporel considéré pour situer les observés de la trace.

Nous définissons un *modèle de trace* de manière informelle comme étant le vocabulaire permettant la compréhension de la trace, décrivant abstraitement ces observés et les relations pouvant exister entre eux. Plus formellement, un modèle de trace est une sorte d'ontologie décrivant en plus des types, relations et attributs des observés constituant la trace, le domaine temporel associé à ces éléments.

Définition 12 (Modèle de Trace)**.** *Un modèle de trace est définit comme un tuple*

$$\mathbb{M} = (\mathcal{T}, C, R, A, \leq_C, \leq_R, \mathrm{dom}_R, \mathrm{range}_R, \mathrm{dom}_A, \mathrm{range}_A)$$

constitué de :

- *un domaine temporel* $\mathcal{T}$*,*
- *un ensemble fini* C *de types d'observé (ou classes) et un ordre partiel* $\leq_C$ *défini sur* C*,*
- *un ensemble fini* R *de types de relation disjoint de* C *et un ordre partiel* $\leq_R$ *défini sur* R*,*
- *un ensemble fini* A *d'attributs disjoint de* C *et* R*,*
- *deux fonctions* $\mathrm{dom}_R : R \to C$ *et* $\mathrm{range}_R : R \to C$ *définissant le domaine et le codomaine des types de relation,*
- *deux fonctions* $\mathrm{dom}_A : A \to C$ *and* $\mathrm{range}_A : A \to \mathbf{D}$ *définissant le domaine et le codomaine des attributs,*

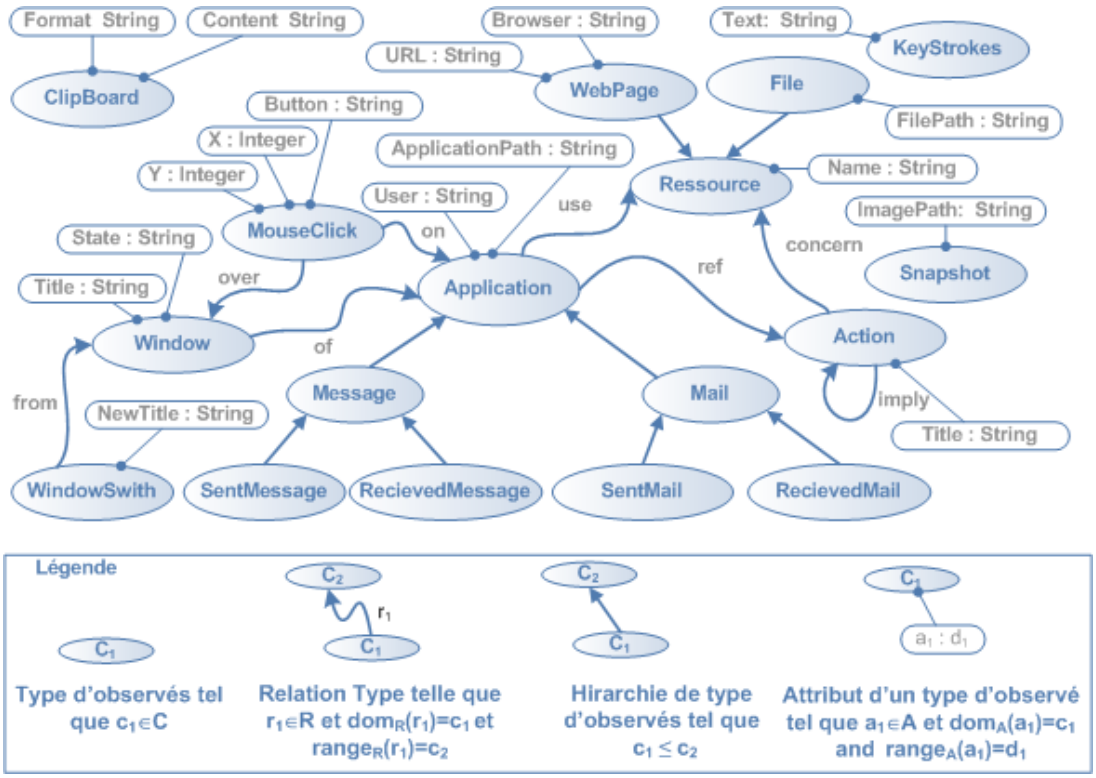

FIGURE 4.3 – Un exemple d'un modèle de Trace d'un keylogger

Un modèle de trace doit respecter la contrainte suivante pour garantir la consistance des hiérarchies de types :

$$\forall r_1, r_2 \in R, r_1 \leq_R r_2 \Rightarrow \text{dom}_R(r_1) \leq_C \text{dom}_R(r_2) \wedge \text{range}_R(r_1) \leq_C \text{range}_R(r_2)$$

Intuitivement, un modèle de trace définit un vocabulaire pour décrire des traces en définissant : comment le temps est représenté ($\mathcal{T}$), comment les observés sont catégorisés et classés (C), quelles relations peuvent exister entre des observés (R), quels attributs décrivent chaque éléments observés (A).

Les domaines de début et d'arrivée contraignent les types de relations et les attributs qu'un observé d'un type donné peut avoir. Les ordre partiaux $\leq_C$ and $\leq_R$ définissent des hiérarchies de types pour les observés et les relations. La dernière contrainte garantit la cohérence et la consistance du domaine et du codomaine pour une relation et ses parents dans la hiérarchie.

La figure 4.3 montre un exemple de modèle de trace décrivant l'observation faite d'un usage de système d'exploitation, partant des éléments d'une source de traçage d'un keylogger espionnant les touches clavier, les cliques souris, entrée/sortie réseau, lancements d'application, etc. La figure 4.3 montre à titre d'exemple le modèle de trace utilisé pour implémenter une collecte à base de sources de traçage issues du keylogger. Dans ce modèle nommé $\mathbb{M}_{key}$, les différents éléments sont :

– $\mathcal{T} = \mathbf{N}$ l'ensemble des entiers naturels représentant le domaine temporel de la trace.

- $C = \{$`Application`, `MouseClick`, `Ressource`, `Action`, `WebPage`, etc.$\}$
- $R = \{$`use`, `on`, `concerns`, `ref`, `imply`, `from`, etc. $\}$ avec dom_R(`use`)= `Application`, range_R(`use`)=`Ressource`, etc.
- $A = \{$`User`, `ApplicationPath`, `Button`, `Name`, etc. $\}$ avec dom_A(`User`)=`Application`, range_A(`User`)=`xml:String`, etc.
- $\leq_C = \{$ (`Mail`,`Application`), (`Message`,`Application`), etc. $\}$ et $\leq_R = \emptyset$ puisque ce modèle ne dispose pas d'une hiérarchie de relations type.

4.6.2 Trace Modélisée

Nous appelons *trace modélisée* (M-Trace dans la suite) l'association d'une collection d'observés temporellement situés et d'un modèle explicite de cette collection d'observés. Une M-Trace donc est toujours associée à un modèle de trace définissant les éléments qui la composent : observés, attributs et relations.

Définition 13 (M-Trace)**.** *Une M-Trace est un tuple*

$$\mathbb{T} = (\mathbb{M}, id, obs, attr, rel)$$

constituée de

- *un modèle de trace* $\mathbb{M} = (\mathcal{T}, C, R, A, \leq_C, \leq_R, \text{dom}_R, \text{range}_R, \text{dom}_A, \text{range}_A)$,
- *un ensemble d'identifiants d'observés* $id \subseteq \mathbf{V}$ *potentiellement infini,*
- *une fonction totale*[34] $obs : id \to C \times \mathcal{T} \times \mathcal{T}$ *définissant l'ensemble, potentiellement infini, des observés de* $\mathbb{T}$*. Chaque tuple* $(i, c, s, e) \in obs$ *est un observé tel que :*
  - *i est l'identifiant de l'observé. La définition de obs comme étant une fonction totale (au lieu d'une relation) assure que i identifie sans ambiguïté l'observé de la trace* $\mathbb{T}$ *;*
  - *c est le type d'observé ;*
  - *s et e sont respectivement le temps de début et de fin de l'observé.*
- $rel \subseteq R \times id \times id$ *est une relation définissant l'ensemble, potentiellement infini, des relations existant entre deux observés.*
- *une fonction partielle*[35] $attr : id \times A \to \mathbf{V}$ *définissant l'ensemble, potentiellement infini, des valeurs associés aux attributs des observés.*

En plus, la trace doit vérifier les deux contraintes suivantes :

- une trace doit être *consistente* avec son modèle, *i.e.*
  1. $\forall (i, a, v) \in attr,\ obs(i) = (c, s, e)\ c \leq_C \text{dom}_A(a) \wedge v \in \text{range}_A(a)$. Ceci permet de garantir la conformité entre les types des domaines et co-domaines des attributs.
  2. $\forall (r, i, i') \in rel,\ obs(i) = (c, s, e),\ obs(i) = (c', s', e')\ c \leq_C \text{dom}_R(r) \wedge c' \leq_C \text{range}_R(r)$. Ceci permet de garantir la conformité entre les types des domaines et co-domaines des relations.

34. *obs* est définie sur chaque $i \in id$.
35. *attr* peut être indéfini pour un certain $(i, a) \in id \times A$.

– $\forall(i, c, s, e) \in obs, s \leq_{\mathcal{T}} e$ pour garantir que le temps de début de l'observés est au moins avant son temps de fin.

Intuitivement, une M-Trace représente, en respectant un modèle de trace ($\mathbb{M}$), un ensemble d'identifiants caractérisant les observés sans ambiguïté, un ensemble d'observés identifiés, typés et situé dans le temps (obs), en relation les uns avec les autres (rel) et ayant potentiellement des attributs valués ($attr$).

Chaque observé (i, c, s, e) a exactement un type *direct* (obs est une fonction totale) ; noter aussi que le $\leq_C$ induit une représentation de hiérarchie de type, c'est à dire que chaque type $c \leq_C c'$ peut être considéré comme un type *indirect* de l'observé (i, c, s, e). Il peut y avoir zéro, une ou plusieurs relation(s) entre deux observés donnés (rel peut être n'importe quelle relation). Finalement, les valeurs d'attribut ne sont jamais obligatoires.

Noter bien qu'en plus de la définition de M-Trace, deux contraintes sont énoncées. La première contrainte permet de s'assurer qu'une M-Trace est toujours compatible avec son modèle. Ceci suppose donc les relations hiérarchiques entre types sont toujours cohérente par rapport au domaine et co-domaine des relations et les attributs des observés. La second contrainte suppose que la durée d'un observé n'est jamais négative.

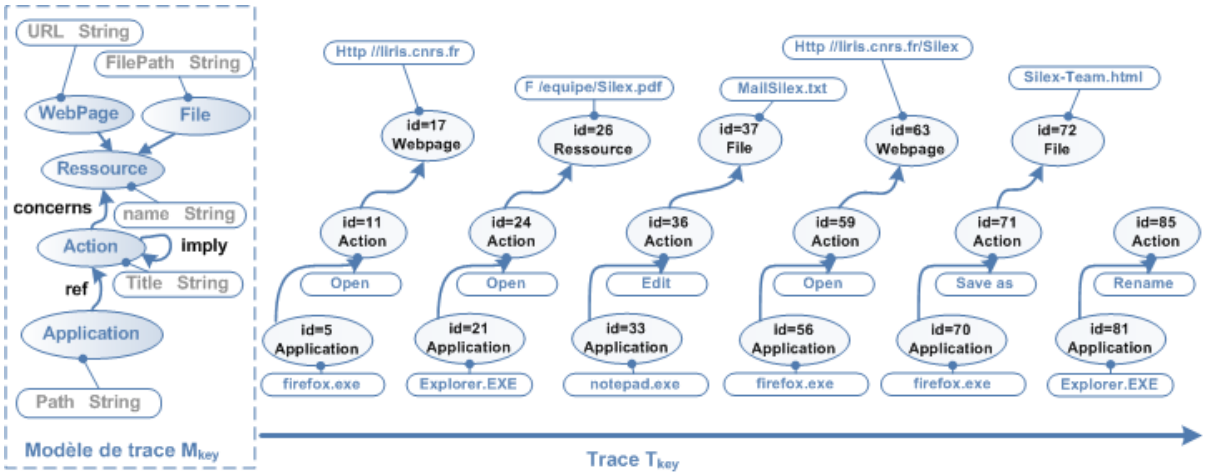

FIGURE 4.4 – Un exemple d'une portion de M-Trace

La figure 4.4 représente un exemple d'une partie d'une M-Trace nommée $\mathbb{T}_{key}$ conforme au modèle $\mathbb{M}_{key}$ définit précédement [36]. Les élément de la trace sont :

– $\mathbb{M} = \mathbb{M}_{key}$ tel qu'il est définit dans la figure 4.3
– $id = \{5, 11, 17, 21, 24, 26, \text{ etc.}\}$
– $obs = \{(5, \texttt{Application}, 1, 2), (11, \texttt{Action}, 2, 2), (11, \texttt{Webpage}, 2, 3), \text{ etc. }\}$
– $attr = \{(5, \texttt{Path}, \text{“}\texttt{firefox.exe}\text{”}), (11, \texttt{Path}, \text{“}\texttt{Open}\text{”}), \text{ etc. }\}$
– $rel = \{(\texttt{ref},5,11), (\texttt{concern},11, 17), \text{ etc. }\}$

Nous définissons les fonctions suivantes sur des traces pour simplifier les définitions :

– **model** : $(\mathbb{M}, id, obs, attr, rel) \mapsto \mathbb{M}$

36. Nous avons omis de mettre tout les éléments du modèle de trace $\mathbb{M}_{key}$ pour rendre la figure plus lisible.

- **id** : $(\mathbb{M}, id, obs, attr, rel) \mapsto id$
- **type** : $(\mathbb{M}, id, obs, attr, rel) \mapsto (i \rightarrow c \mid obs(i) = (c, s, e))$
- **start** : $(\mathbb{M}, id, obs, attr, rel) \mapsto (i \rightarrow s \mid obs(i) = (c, s, e))$
- **end** : $(\mathbb{M}, id, obs, attr, rel) \mapsto (i \rightarrow e \mid obs(i) = (c, s, e))$
- **obs** : $(\mathbb{M}, id, obs, attr, rel) \mapsto obs$
- **attr** : $(\mathbb{M}, id, obs, attr, rel) \mapsto attr$
- **rel** : $(\mathbb{M}, id, obs, attr, rel) \mapsto rel$

Par souci de lisibilité, ces fonctions seront généralement utilisées en une notation en indice, à savoir $\mathbf{model}_{\mathbb{T}}$ au lieu du $\mathbf{model}(\mathbb{T})$, $\mathbf{id}_{\mathbb{T}}$ au lieu de $\mathbf{id}(\mathbb{T})$, etc.

4.6.3 Hypothèses considérées pour les M-Traces

L'ordre considéré dans les M-Traces est l'ordre induit par le domaine temporel $\leq_{\mathcal{T}}$. Cette ordre concrétise une hypothèse fondamentale considérée dans cette thèse : deux observés qui se suivent par leur temps de fin, se suivent forcement par leur temps d'appartenance à la M-Trace, et vice versa. De manière plus précise, le moment de considération qu'un observé est inclus dans une M-Trace peut correspondre à différents temps réifiés par différents ordres : $\leq_{ajout}$ organisant les observés par leurs temps d'ajout à la M-Trace ; $\leq_{production}$ ordonnançant les observés par leurs temps de production et de génération par l'application tracée ; $\leq_{collecte}$ ordonnançant les observés par leurs temps de capture fourni par le collecteur ; $\leq_{reception}$ ordonnançant l'ordre d'arrivée des observés au système à base de traces. Dans notre formalisation, ces différents ordres sont réifiés par $\leq_{\mathcal{T}}$ [37].

Pour pouvoir traiter des M-Traces de manière continue et homogène, nous supposons que le temps du domaine temporel permet toujours de reconstituer leurs évolutions dans temps. Cette contrainte est fondamentale, elle suppose que les différents ordres identifiés et l'ordre induit par le domaine temporel sont isomorphes [38]. Si ces ordres ne sont pas coordonnés entre eux, e.g., en raison de latences dues au réseau, des techniques bien identifiées par les systèmes de gestion de flux de données, comme celles présentées dans [SW04], peuvent être appliquées.

Intuitivement, une M-Trace peut être vue comme une séquence permettant uniquement des ajouts d'observés (ayant au moins un identifiant et un estampillage temporel). Ces observés arrivent dans un ordre quelconque. Noter bien que les observés de la M-Trace peuvent se suivre ou arriver au même temps. Nous considérons aussi que chaque observé ajouté à la M-Trace doit avoir toutes ses informations associées (les valeurs d'attribut, les relations), qu'on considère ayant la même unité de temps. Comme conséquence importante, l'observé considéré comme appartenant à une M-Trace ayant un temps de fin à l'instant t ne peut pas avoir de valeurs d'attribut dans le temps $t' > t$. La seule exception est quand un élément observée doit être mis en relation (une ou plus) avec d'autres observés (un ou plus) appartenant déjà à la M-Trace à l'instant $t" < t$. En outre, nous assumons également les deux remarques suivantes :

37. Noter que cette hypothèse est forte puisque ces ordres peuvent ne pas correspondre à $\leq_{\mathcal{T}}$ dans la pratique.

38. Deux ensembles ordonnés $(A, \leq_A)$ et $(B, \leq_B)$ sont isomorphes si et seulement si il existe une bijection $f : A \rightarrow B$ tel que pour tout $a_1, a_2 \in A$, $a_1 \leq_A a_2$ si et seulement si $f(a_1) \leq_B f(a_2)$.

– les observés arrivent en permanence dans le SBT, poussés par une source de collecte active. Le service de collecte n'a la maîtrise ni sur l'ordre dans lequel les observés vont arriver, ni sur leur fréquence d'arrivée. La fréquence d'arrivée des observés dans la M-Traces est imprévisible et varie au fil du temps.
– le service de collecte transmet chaque élément observé une seule fois. Comme les éléments observés sont accessibles de façon séquentielle, un observé qui est arrivé dans le passé peut être accéder directement dans la M-Trace.

Après avoir défini la notion de M-Trace et présenté les hypothèses considérées lors de la collecte, nous définissons dans la suite les langages permettant de les interroger et transformer.

4.7 Syntaxe des langages pour l'interrogation et la transformation de M-Traces

La section suivante présente la syntaxe du langage pour traiter et manipuler des traces modélisées. Tout d'abord, nous avons besoin de décrire les notions nécessaires pour définir le langage d'interrogation de M-Traces, ensuite nous aborderons la définition des concepts permettant de définir le langage de transformation.

Nous commençons d'abord par définir la notion de *pattern*, qui nous sera utile pour définir les requêtes, mais aussi les transformations.

4.7.1 Pattern et Requête

Nous définissons les patterns en utilisant un ensemble de variables ϑ, disjoint de l'ensemble $\mathbf{V}$ des valeurs concrètes. Un pattern est toujours associé à un modèle de trace $\mathbb{M}$.

Définition 14 (Syntaxe d'un Pattern de requête)**.** *Un pattern P est construit à partir de l'élément $<$ Pattern $>$ de la grammaire de la figure 4.3.*

Étant donné une expression P construite à partir de l'élément $<$ *Pattern* $>$, l'ensemble de toutes les variables utilisées dans cette formule est noté $\mathbf{var}_P$. Noter bien qu'un pattern est toujours lié à un modèle de trace. Ceci est une propriété importante du pattern qui est toujours valable, même si elle n'est pas explicitement mentionnée ou précisée dans certaines parties de ce chapitre.

Le pattern `(x : Action)` est un exemple d'un pattern simple décrivant abstraitement l'ensemble des observés de type `Action`. Plus précisément, il décrit les observés de type égale ou inférieur au type `Action`. Un pattern simple peut aussi inclure des conjonctions (*and*), des disjonctions (*or*), des contraintes sur des attributs et des relations temporelles et structurelles entre observés, etc. Par exemple : `(x : Action) or (y.Title = 'Copy')`.

Un pattern complexe est un pattern pouvant utiliser aussi la négation (*without*). Il utilise aussi les relations temporelles relatives (`extend, shorten`, etc.) permettant de spécifier des sortes de *fenêtres temporelles* utiles à l'évaluation continue de pattern usant la négation. Cette

$< Pattern >$	::=	$< Pattern - Simple > \mid < Pattern - Complexe >$
$< Pattern - Simple >$	::=	o " : " $c \mid o\ r\ o$
		$\mid < Pattern >$ "`and`" $< Pattern >$
		$\mid < Pattern >$ "`or`" $< Pattern >$
		$\mid < Pattern >$ "`where`" $< PredicateExp >$
		$\mid$ "(" $< Pattern >$ ")"
$< Pattern - Complexe >$	::=	$< Pattern - Simple >$
		$\mid$ `while` o " : " `without` $< Pattern >$
		$\mid o < RelativeTempSpec >$ "[" o, $< Duration >$ "]"
$< RelativeTempSpec >$	::=	"`extend`" $\mid$ "`shorten`" $\mid$ "`extend-start`" $\mid$ "`shorten-start`"
		$\mid$ "`shift-forward`" $\mid$ "`shift-backward`"
		$\mid$ "`from-end`" $\mid$ "`from-end-backward`"
		$\mid$ "`from-start`" $\mid$ "`from-start-backward`"
$< Expression >$	::=	$v \mid o$"."$a \mid o$"*.id*" $\mid o$"*.start*" $\mid o$"*.end*" $\mid o$"*.origin*"
		$\mid < Expression > < ArithOp > < Expression >$
		$\mid$ "(" $< Expression >$ ")" $\mid f(< Expression >^{+})$
$< ArithOp >$	::=	" $*$ " $\mid$ "/" $\mid$ " $+$ " $\mid$ " $-$ " $\mid$ "*mod*"
$< PredicateExp >$	::=	$< Expression > < Predicate > < Expression >$
		$\mid o < AllenPredicate > o$
		$\mid$ "{" $(o$ ","?$) +$ "}" "`within`" $< Duration >$
		$\mid$ "{"o "," o "}" "`apart-by`" $< Duration >$
$< AllenPredicate >$	::=	"`before`" $\mid$ "`contains`" $\mid$ "`overlaps`" $\mid$ "`after`" $\mid$ "`during`"$''$
		$\mid$ "`overlapped-by`" $\mid$ "`starts`" $\mid$ "`finishes`" $\mid$ "`meets`"
		$\mid$ "`started-by`"$''$ $\mid$ "`finished-by`" $\mid$ "`met-by`" $\mid$ "`equals`
$< Predicate >$	::=	" $=$ " $\mid$ " $\neq$ " $\mid$ " $<$ " $\mid$ " $\leq$ " $\mid$ " $\geq$ " $\mid$ " $>$ "
$< Duration >$	::=	(Number ("`week`" $\mid$ "`weeks`"))?
		(Number ("`day`" $\mid$ "`days`"))?
		(Number ("`hour`" $\mid$ "`hours`"))?
		(Number "`min`")?
		(Number "`sec`")?
		(Number "`ms`")?

avec $o \in \vartheta,\ c \in C,\ r \in R,\ v \in V,\ Number \in N,\ a \in A, f : V^n \rightarrow V, n \in N.$

TABLE 4.3 – La syntaxe d'un pattern exprimé sur un modèle de trace

décomposition nous permettra d'étudier en premier les patterns simples et puis étendre notre étude aux patterns complexes. En effet, la négation est bien connue pour être délicate à définir, nécessitant un traitement spécifique notamment dans le cas d'une évaluation continue [GO03, Ara06, Kra07].

Une requête exprimée sur une M-Trace est une expression $\mathcal{Q}$ utilisant un pattern et l'ensemble des variables visibles après évaluation. Plus formellement, une requête est définie comme suit.

Définition 15 (Requête sur une M-Trace)**.** *Une requête sur une M-Trace* $\mathbb{T}$ *est un couple* $\mathcal{Q} = (\mathbf{var}_\mathcal{Q}, P)$ *tel que*

- P *est pattern sur le modèle* $\mathbb{M}$*,*
- $\mathbf{var}_\mathcal{Q} \subseteq \mathbf{var}_P$ *est l'ensemble ordonné des variables résultats de* $\mathcal{Q}$*.*

Nous utiliserons la syntaxe `Match <Variables> To <Pattern>` pour décrire une requête. Cependant, dans certains exemples et définitions nous omettront les mots clés `Match` et `To` pour une syntaxe plus concise. Soit la requête $\mathcal{Q}_1$ = `Match x To ((x : Action) and (x refs y))`. L'évaluation de cette requête fait correspondre la variable `x` aux observés de la M-Trace. Bien que la pattern utilisé fait correspondre deux variables `x` et `y`, la requête restreint le résultat juste à la variable `x`.

Comme pour les patterns, nous distinguons les requêtes simples et les requêtes complexes. Les requêtes complexes sont des requêtes utilisant des patterns complexes. Les requêtes simples sont des requêtes qui ne sont pas complexes.

Nous allons maintenant présenter la notion duale aux patterns que sont les templates. Les templates décrivent la partie construction d'une règle de transformation. Dans le monde des langages à base de règles, un pattern et un template correspondent respectivement au *corps* et *entête* d'une règle.

4.7.2 Template, Règle de Transformation et Transformation

Définition 16 (Syntaxe d'un Template de Transformation)**.** *Un* template de transformation *ou* template *est contruit à partir de l'élément* $< Template >$ *de la grammaire de la figure 4.4.*

Une transformation est un ensemble de règles permettant de construire une nouvelle trace transformée. Chaque règle dispose d'une partie *pattern* représentant les observés sources à transformer et d'une partie *template* décrivant l'ensemble des observés résultat.

Un template peut spécifier le début et/ou la fin des observés à construire par exemple le template G_1 = `(z : ActionCopy)`, `(z.start=x.start)`, `(z.end=y.end)` spécifie les bornes temporelles des observés créés. Un template peut aussi omettre la spécification d'une ou des deux bornes temporelles des observés à créer. Par exemple G_2 = `ActionPaste` est un exemple d'un template permettant de créer des observés de type `ActionPaste`. Les bornes temporelles des observés construits seront calculées directement et correspondent respectivement au temps de début minimal et au temps de fin maximal des observés, correspondant aux variables du

$$
\begin{array}{rcl}
< Template > & ::= & < Template - c > \\
 & & |< Template - c > \text{“;”} < Template - id > \\
 & & |< Template - c > \text{“;”} < Template - time > \\
 & & |< Template - c > \text{“;”} < Template - id > \text{“;”} < Template - time > \\
 & & | < Template > \text{“;”} < Template - attr > \\
 & & | < Template > \text{“;”} < Template > \text{“;”} < Template - rel > \\
< Template - c > & ::= & o \text{“ : ”} c \\
< Template - id > & ::= & w\text{“.id = ”} < Expression > \\
< Template - time > & ::= & < Template - start > \text{“;”} < Template - end > \\
 & & | < Template - start > | < Template - end > \\
< Template - start > & ::= & w\text{“.start = ”} o \text{ “.start”} \\
< Template - end > & ::= & w\text{“.end = ”} o \text{ “.end”} \\
< Template - attr > & ::= & w\text{“.a = ”} < Expression > \\
 & ::= & | < Template - attr > \text{ “;” } w\text{“.a = ”} < Expression > \\
< Template - rel > & ::= & w \ r \ w \\
< Expression > & ::= & v \mid o\text{“.”}a \mid \ o\text{“.id”} \mid \ o\text{“.start”} \mid \ o\text{“.end”} \mid \ o\text{“.origin”} \\
 & & | < Expression > < ArithOp > < Expression > \\
 & & | \ \text{“(”} \ < Expression > \ \text{“)”} \mid f(< Expression >^{+}) \\
< ArithOp > & ::= & \text{“}*\text{”} \ \mid \ \text{“/”} \ \mid \ \text{“}+\text{”} \ \mid \ \text{“}-\text{”} \ \mid \ \text{“}mod\text{”} \\
 & & \\
 avec & & w \in \vartheta - \mathbf{var}_P, \ o \in \mathbf{var}_P, \ c \in C, \ r \in R, \ v \in V, \ a \in A, f : V^n \rightarrow V, n \in N
\end{array}
$$

TABLE 4.4 – La syntaxe d'un template de transformation

pattern utilisé pour leur construction. Pour chaque résultat identifié lors de l'évaluation des patterns, un observé sera créé par cet template.

Une règle de transformation associe un pattern à un template. Une règle définit intuitivement la construction de nouveaux observés conformes au template à partir des observés reconnus par le pattern. Une transformation regroupe un ensemble de règles permettant de construire les nouveaux observés.

Définition 17 (Règle de Transformation et Transformation de M-Trace)**.** *Soient les modèles de trace* $\mathbb{M}, \mathbb{M}'$*. Nous définissons une règle de transformation comme un couple* (P, G) *noté* $(G \leftarrow P)$ *tel que* P *est une pattern sur* $\mathbb{M}$ *et* G *est un template sur le modèle* $\mathbb{M}'$*. Une* transformation de M-Trace *ou* transformation *notée* τ *est un ensemble de règle de transformation.*

Dans le cas d'une auto-transformation $\mathbb{M}$ et $\mathbb{M}'$ représente le même modèle, i.e, le template et le pattern sont liés au modèle $\mathbb{M}$.

De la même manière que pour les requêtes, nous distinguons les transformations complexes des transformations simples. Les transformations simples sont des transformations qui utilisent uniquement des patterns simples. Les transformations complexes peuvent utiliser des patterns complexes et introduisent donc la négation dans les patterns des règles de transformation.

Comme pour les requêtes avec la syntaxe `Match <Variables> To <Pattern>`, nous utiliserons une même syntaxe explicite pour les transformations simples et complexes : `Construct <Template> On <Pattern>`.

Enfin, nous nommons certaines variantes de notre langage de requêtes et de transformations de M-Traces. Les langages permettant d'exprimer des requêtes et des transformations simples sont notés respectivement QML^{-} et $TQML^{-}$. Pour les requêtes et transformations complexes, les variantes de langages permettant de les exprimer sont notés respectivement QML^{w} et $TQML^{w}$. Noter également que QML^{-} est inclut dans QML^{w} et que $TQML^{-}$ est un sous langage de $TQML^{w}$.

4.8 Étude de la sémantique déclarative du langage de transformation

Concernant la sémantique déclarative du langage pour l'interrogation et la transformation de M-Traces, le problème adressé par notre sémantique peut être abstrait de la manière suivante : étant donné un ensemble de règles de transformation et un flux entrant d'éléments de la M-Trace, i.e, l'ensemble des observés, leurs relations et valeurs d'attribut qui sont collectés et ne sont pas dérivés par les règles, nous voulons avoir tous les éléments qui sont dérivés par les règles de transformation. Cette approche couvre donc également la question de trouver les réponses à une requête étant donné un pattern et un flux entrant d'observés de la M-Trace. Dans la suite de ce chapitre, nous parlerons d'observés pour désigner l'ensemble des observés, des relations et des valeurs d'attribut associés à eux.

La théorie des modèles proposée dans cette thèse est définie dans le style de Tarski, i.e., définie récursivement sur la structure des expressions du langage. La description d'expressions des éléments du langage (les règles, les requêtes) est similaires aux expressions de la logique du premier ordre. Cependant, les observés de la M-Trace ne sont pas des faits logiques et sont représentés comme des données typés, pouvant avoir des attributs et des relations avec un temps de présence). Ces expressions sont liées à une interprétation par la définition d'une relation de satisfaction. La relation de satisfaction est définie récursivement sur la structure du langage, i.e., les patterns et les règles de transformation. Les interprétations intéressantes pour la sémantique sont ceux qui (1) satisfont toutes les règles d'une transformation à l'égard de la relation de satisfaction et (2) contiennent le flux des observés entrant et sortant. Les observés entrants et sortants peuvent être dans une même interprétation dans le cas d'une auto-transformation d'une M-Trace. Dans le cas d'une allo-transformation d'une M-Trace $\mathbb{T}_1$ vers une nouvelle M-Trace $\mathbb{T}_2$, deux interprétations sont nécessaires pour satisfaire cette transformation. Une interprétation $\mathcal{I}_1$ contenant tous les observés de $\mathbb{T}_1$ et une interprétation $\mathcal{I}_2$ regroupant les observés résultant des règles de transformation (correspondant évidemment à $\mathbb{T}_2$). Ces interprétations particulières sont alors appelées *modèles*.

Nous définissons dans la suite la notion d'interprétation et de satisfaction de pattern. Ces

notions seront par la suite utilisées pour la définition de modèles pour les requêtes et les transformations. Nous commençons d'abord par définir la notion de substitution.

4.8.1 Substitutions, Ensemble de Substitutions

Les patterns pour des requêtes contiennent des variables libres qui seront liées à des valeurs de la M-Trace lors de l'évaluation de la requête. Nous avons besoin donc de la notion de *substitution* pour décrire ce lien entre variables et observés de la M-Trace. Cette notion nous permettra de définir formellement la sémantique des patterns et des requêtes.

Soit ϑ l'ensemble de tous les noms de variables. Nous appelons fonction de substitution, ou tout simplement substitution, toute fonction (partielle ou totale) $\psi : \vartheta \rightarrow \mathbf{id}_{\mathbb{T}}$, où $\mathbf{id}_{\mathbb{T}}$ est l'ensemble des identifiants des éléments observés d'une M-Trace.

Les substitutions seront également utiles pour représenter les résultats d'une requête. En effet, une substitution définit une *correspondance (mapping)* ou une *affectation* d'un ensemble de variables à un ensemble d'observés. Le domaine de cette fonction (*i.e.* l'ensemble des variables sur lesquelles elle est définie) est noté $\mathbf{domain}(\psi)$. Nous écrirons des substitutions comme $\psi = (x_1 \rightarrow i_1, ..., x_n \rightarrow i_n)$, ce qui signifie que $\psi(x_k) = i_k$ pour $k \in (1, ..., n)$ et $\psi(y) = \perp$ pour $y \notin (x_1, ..., x_n)$. Nous définissons aussi la notion d'ensemble de substitutions noté Σ sur laquelle la sémantique des patterns sera fondée.

Deux substitutions ψ_1 et ψ_2 sont dites compatibles, noté $\psi_1 \sim \psi_2$, si leur union est aussi une fonction, *i.e.* si :

$$\forall\ o \in \mathbf{domain}(\psi_1) \cap \mathbf{domain}(\psi_2),\ \psi_1(o) = \psi_2(o)$$

En plus des opérations classiques sur les ensembles, nous aurons besoin des opérations suivantes. En supposant que Σ_1 et Σ_2 sont deux ensembles de substitutions pour P :

- $\Sigma_1 \bowtie \Sigma_2 = \{\psi_1 \cup \psi_2 \mid \psi_1 \in \Sigma_1, \psi_2 \in \Sigma_2, \psi_1 \sim \psi_2\}$
- $\Sigma_1 \ominus \Sigma_2 = \{\psi_1 \mid \psi_1 \in \Sigma_1, \forall\ \psi_2 \in \Sigma_2, \psi_1 \nsim \psi 2\}$
- $\Sigma_1 \rtimes \Sigma_2 = (\Sigma_1 \bowtie \Sigma_2) \cup (\Sigma_1 \ominus \Sigma_2)$

Enfin, nous avons besoin de définir la substitution d'une expression E telle qu'elle est définie par l'élément $< Expression >$ dans la syntaxe du pattern et du template. Étant donnée une expression E et une substitution ψ, l'application de ψ sur E notée $\psi(E)$ produit une nouvelle expression tel que

1. pour une expression E de la forme ($x.a$, x.id, x.start, x.end), la variable x est remplacée par $\psi(x)$ s'il est défini sinon $\psi(E) = \emptyset$
2. pour une expression E de la forme $f(E_0, ..., E_n)$, $\psi(f(E_0, ..., E_n))$=$f(\psi(E_i), ..., \psi(E_k))$ avec $\forall E_i, E_k$ avec $0 \leq i < k \leq n$ et $\psi(E_i) \neq \emptyset$ et $\psi(E_k) \neq \emptyset$, i.e, on supprime toutes les expressions qui peuvent pas être substituées.
3. pour une expression E de la forme $predicate(E_0, ..., E_n)$, $\psi(predicate(E_0, ..., E_n)) = predicate(\psi(E_i)$,...,$\psi(E_k))$ avec $\forall E_i, E_k$ avec $0 \leq i < k \leq n$ et $\psi(E_i) \neq \emptyset$ et $\psi(E_k) \neq \emptyset$, i.e, on supprime toutes les expressions qui peuvent pas être substituées.

Les deux premières substitutions d'expression sont valables pour les expressions de patterns et de templates. La dernière substitution n'est valable que pour les expressions d'un pattern.

Évaluation d'une expression de M-Trace

Nous avons besoin de définir la sémantique des expressions utilisées dans les patterns et les templates. Pour une M-Trace $\mathbb{T}$, nous avons donc besoin de définir la sémantique pour évaluer une expression E, notée $\mathbf{eval}_{\mathbb{T}}(E)$, telle qu'elle est définie dans la table 4.3.

Noter que la principale différence des expressions sur des patterns par rapport à des expressions sur des templates est que les patterns permettent de l'expression de prédicats ($=, \neq, <, \leq, >, \geq$).

Définition 18 (Évaluation d'une Expression)**.** *Soient une expression E dans un pattern ou un template et une substitution ψ. La fonction $\mathbf{eval}_{\mathbb{T}}(E)$ est définie comme suit :*

- $\mathbf{eval}_{\mathbb{T}}(\emptyset) = \emptyset$
- $\mathbf{eval}_{\mathbb{T}}(v) = v$
- $\mathbf{eval}_{\mathbb{T}}(i.\mathrm{id}) = \mathbf{id}_{\mathbb{T}}(i)$
- $\mathbf{eval}_{\mathbb{T}}(i.a) = \mathbf{attr}_{\mathbb{T}}(i, a)$
- $\mathbf{eval}_{\mathbb{T}}(i.\mathrm{start}) = b$ tel que $\mathbf{obs}_{\mathbb{T}}(i) = (c, b, e)$
- $\mathbf{eval}_{\mathbb{T}}(i.\mathrm{end}) = e$ tel que et $\mathbf{obs}_{\mathbb{T}}(i) = (c, b, e)$
- $\mathbf{eval}_{\mathbb{T}}(f((E_1), ..., (E_n)) = f(\mathbf{eval}_{\mathbb{T}}(E_1), ..., \mathbf{eval}_{\mathbb{T}}(E_n))$ *en supposant que* $f : \mathbf{V}^n \rightarrow \mathbf{V}$ *(pour les opérateurs arithmétiques classiques, i.e.«*»,«+», etc., nous appliquons la sémantique usuelle)*
- $\mathbf{eval}_{\mathbb{T}}(predicate(E_0, ..., E_n)) = true$ *si* $(E_0, ..., E_n) \in predicate$ *en supposant que :* $predicate \subseteq \mathbf{V}^n$.

avec $i, v \in \mathbf{V},\ n \in \mathbf{N}$.

4.8.2 Interprétation

Pour n'importe quel langage logique, la sémantique est définie en fixant les modèles de la logique et la façon de donner un sens aux expressions du langage à l'égard de ces modèles. Intuitivement, chaque modèle possible peut être considéré comme un monde possible dans lequel une expression peut être évaluée. L'évaluation d'une expression dans un modèle est souvent appelé *interprétation* de l'expression à l'égard du modèle. Ainsi, lors de la définition d'une sémantique d'un langage se fondant sur la logique, les points clés à indiquer sont les suivants : 1) Quels sont les interprétations et les modèles ? et 2) Comment sont satisfaites les expressions dans ces interprétations et modèles ?

De manière informelle, pour les requêtes et les transformations de M-Traces les interprétations indiquent si *un observé existe* ou *n'existe pas* dans le modèle. Ainsi, dans une première approximation, nous définissons une interprétation comme un ensemble d'observés qui satisfont le pattern, i.e., pour lesquels l'expression du pattern est satisfaite. Puisque les données de la M-Trace représentent des observations, cette définition est naturelle et donc particulièrement

bien adaptée pour le raisonnement sur les M-Traces. En outre, une interprétation fournit un ensemble de substitutions offrant des affectations à l'ensemble des variables dans les patterns envisagés. Les interprétations sont donc formellement définies comme suit :

Définition 19 (Interprétation)**.** *Une interprétation pour un pattern P ou une transformation τ est un couple*

$$\mathcal{I} = (O, \Sigma)$$

tel que

- *O est l'ensemble représentant les observés, leurs relations, et valeurs d'attribut relatives à un modèle de trace $\mathbb{M}$. Nous rajoutons les contraintes suivantes : $\forall (i, i', r) \in O$, (i, c, s, e), $(i', c', s', e') \in O$ et $\forall (i, a, v) \in O$, $(i, c, s, e) \in O$.*
- *$\Sigma \neq \emptyset$ est un ensemble de substitutions. $\forall (i, c, s, e) \in O$, $\exists \psi \in \Sigma$, $\mathbf{range}(\psi) = i$.*

On écrira par souci de lisibilité, $O \subseteq \mathbb{T}$ (ou $\mathbb{T} \subseteq O$) au lieu de $O \subseteq \mathbf{obs}_{\mathbb{T}} \cup \mathbf{rel}_{\mathbb{T}} \cup \mathbf{attr}_{\mathbb{T}}$ (ou $\mathbf{obs}_{\mathbb{T}} \cup \mathbf{rel}_{\mathbb{T}} \cup \mathbf{attr}_{\mathbb{T}} \subseteq O$).

L'ensemble des observés O correspond aux observés considérés comme existants. L'ensemble des substitutions Σ est nécessaire pour traiter correctement les patterns contenant des variables, et permet de donner une définition récursive de la relation de satisfaction définie dans la section suivante. Le résultat attendu de l'évaluation des patterns est (explicitement ou implicitement) un ensemble de substitution Σ. Cependant, dans le cas de l'évaluation de transformations (ensemble de règles) où le résultat attendu est un ensemble d'observés, Σ peut être considéré comme un simple détail technique de la définition et n'est pas pertinent pour la notion générale de satisfaction de transformation. Pour cette raison, la suite de ce chapitre assimile parfois par abus de notation les interprétations à l'ensemble des observés O notamment pour la sémantique des transformations.

4.8.3 Satisfaction de pattern pour une M-Trace

La notion d'*interprétation* a donné un sens intuitif à la notion de satisfaction de patterns. La définition de la satisfaction de patterns est similaire à l'approche adoptée en logique classique, mais en diffère par quelques aspects importants. Notre théorie de modèle considère la satisfaction des patterns par rapport à un intervalle de temps. Comme en logique classique, un pattern est satisfait s'il existe une substitution pour ces variables par des observés respectant les propriétés du pattern et inclus dans l'interprétation. Par exemple, pour un pattern de typage `(x : Action)`, la satisfaction dépend non seulement de la substitution des variables (`x` dans ce cas) mais aussi des observés (référencés par leurs identifiants) dont le type est égal ou inférieur au type considéré (i.e., $\leq_C$ `Action`) et qui doivent être inclus dans O. L'exemple d'un pattern de relation comme `(x ref y)` est plus démonstratif, puisque sa satisfaction nécessite en plus des substitutions (pour `x` et `y`) l'inclusion de deux observés dans O ainsi qu'une relation (`ref`) entre ces observés dans l'interprétation. La satisfaction d'un pattern nécessite donc en plus des substitutions, des observés respectant les propriétés et les éléments du pattern.

Un pattern est considéré comme satisfait par une interprétation si (et seulement si) il existe une substitution pour ces variables, et que l'ensemble des observés substituant ses variables est inclus dans l'interprétation O et respecte certaines conditions énoncées dans la définition 20 (e.g., le typage, les relations structurelles et temporelles, les conditions sur les attributs, etc.). La *satisfaction* d'un pattern P par une interprétation $\mathcal{I}$ est décrite par la définition 20. Enfin, un pattern est aussi lié et interprété par rapport à un modèle de trace $\mathbb{M}$.

Définition 20 (Satisfaction d'un pattern)**.** *Une interprétation* $\mathcal{I}$ satisfait un pattern P *pour un modèle de trace* $\mathbb{M}$ *dans un intervalle de temps* $t = [b, e]$*, noté* $\mathcal{I} \models_{\mathbb{M}} P^{[b,e]}$*, si et seulement si elle respecte la définition récursive du tableau 4.5 et 4.6.*

$(O,\Sigma) \models_{\mathbb{M}} (o : c)^{[b,e]}$	ssi $\exists \psi \in \Sigma$ tel que $\psi(o) = i \ \wedge \ (i, c', b', e') \in O \ \wedge \ c' \leq_C c \ \wedge \ [b', e'] = [b, e]$
$(O,\Sigma) \models_{\mathbb{M}} (o \ r \ o')^{[b,e]}$	ssi $\exists \psi \in \Sigma$ tel que $\psi(o) = i_1 \ \wedge \ \psi(o') = i_2 \ \wedge \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2), (r', i_1, i_2) \in O \ \wedge \ r' \leq_R r \ \wedge \ [b, e] = [b_1, e_1] \sqcup [b_2, e_2]$
$(O,\Sigma) \models_{\mathbb{M}} (P \texttt{ where } E)^{[b,e]}$	ssi $(O,\Sigma) \models_{\mathbb{M}} P^{[b,e]} \ \wedge \ \mathcal{S}_{O,\Sigma}(E) = true$
$(O,\Sigma) \models_{\mathbb{M}} (P \texttt{ and } P')^{[b,e]}$	ssi $\Sigma = \Sigma_1 \ \bowtie \ \Sigma_2$ tel que $(O,\Sigma_1) \models_{\mathbb{M}} P^{[b_1,e_1]} \ \wedge \ (O,\Sigma_2) \models_{\mathbb{M}} P'^{[b_2,e_2]} \wedge$ $[b, e] = [b_1, e_1] \sqcup [b_2, e_2]$
$(O,\Sigma) \models_{\mathbb{M}} (P \texttt{ or } P')^{[b,e]}$	ssi $\Sigma = \Sigma_1 \ \cup \ \Sigma_2$ tel que $(O,\Sigma_1) \models_{\mathbb{M}} P^{[b_1,e_1]} \ \wedge \ (O,\Sigma_2) \models_{\mathbb{M}} P'^{[b_2,e_2]} \wedge$ $[b, e] = [b_1, e_1] \sqcup [b_2, e_2]$
$(O,\Sigma) \models_{\mathbb{M}} (P)^{[b,e]}$	ssi $(O,\Sigma) \models_{\mathbb{M}} P^{[b,e]}$
avec	$o, o' \in \vartheta, \ c \in C, \ r \in R, \ [b, e] \in \mathcal{T} \times \mathcal{T}$

TABLE 4.5 – Satisfaction de pattern dans un intervalle de temps

Le tableau 4.5 définit la satisfaction d'un pattern dans un intervalle de temps. Cette définition a pour objectif de définir une méthode d'évaluation qui prend en compte le cas d'une M-Trace bornée dans le temps et le cas d'une M-Trace non bornée qui est en cours de collecte ou de transformation.

Un pattern est *satisfait* s'il existe, dans un intervalle de temps $t = [b, e]$, un ou plusieurs observés substituant ces variables et vérifiant les conditions de typage, relations, contraintes sur les éléments, etc. du pattern. L'ensemble de ces observés O et des substitutions sera considéré comme une interprétation qui satisfait ce pattern.

Notre relation de satisfaction utilise une interprétation fixe $\mathcal{S}_{O,\Sigma}$ pour toutes les conditions se produisant dans la clause `where` du pattern. Elle inclut aussi bien les contraintes temporelles comme `before`, etc. que les comparaisons entre calculs complexes effectués sur les données des observés. Cette interprétation fixe est une caractéristique de notre théorie des modèles qui n'est

$\mathcal{S}_{O,\Sigma}(o \text{ before } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $e_1 < b_2$
$\mathcal{S}_{O,\Sigma}(o \text{ after } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $e_2 < b_1$
$\mathcal{S}_{O,\Sigma}(o \text{ during } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_2 < b_1 \ \wedge\ e_1 < e_2$
$\mathcal{S}_{O,\Sigma}(o \text{ contains } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_1 < b_2 \ \wedge\ e_2 < e_1$
$\mathcal{S}_{O,\Sigma}(o \text{ overlaps } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_2 < b_1 \ \wedge\ e_2 < e_1$
$\mathcal{S}_{O,\Sigma}(o \text{ overlapped-by } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_1 < b_2 \ \wedge\ e_1 < e_2$
$\mathcal{S}_{O,\Sigma}(o \text{ meets } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $e_1 = b_2$
$\mathcal{S}_{O,\Sigma}(o \text{ met-by } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $e_1 = b_2$
$\mathcal{S}_{O,\Sigma}(o \text{ starts } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_1 = b_2 \ \wedge\ e_1 < e_2$
$\mathcal{S}_{O,\Sigma}(o \text{ started-by } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_2 = b_1 \ \wedge\ e_2 < e_1$
$\mathcal{S}_{O,\Sigma}(o \text{ finishes } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \in O \ \wedge$ $b_2 < b_1 \ \wedge\ e_1 = e_2$
$\mathcal{S}_{O,\Sigma}(o \text{ finished-by } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \ \in O \ \wedge$ $b_1 < b_2 \ \wedge\ e_2 = e_1$
$\mathcal{S}_{O,\Sigma}(o \text{ equals } o') = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \ \in O \ \wedge$ $b_1 = b_2 \ \wedge\ e_1 = e_2$
$\mathcal{S}_{O,\Sigma}(\{o, o'\} \text{ apart-by } d) = true$	ssi $\exists \psi \in \Sigma,\ \psi(o) = i_1 \ \wedge\ \psi(o') = i_2 \ \wedge\ \ (i_1, c_1, b_1, e_1), (i_2, c_2, b_2, e_2) \ \in O \ \wedge$ $((e_1 - b_2) \geq d \ \vee\ (e_2 - b_1) \geq d)$
$\mathcal{S}_{O,\Sigma}(\{o_1, .., o_n\} \text{ within } d) = true$	ssi $\exists \psi \in \Sigma,\ \psi(o_1) = i_1 \ \wedge\ ... \ \wedge\ \psi(o_n) = i_n \ \wedge\ (i_1, c_1, b_1, e_1), ..., (i_n, c_n, b_n, e_n) \ \in$ $O \ \wedge\ (\max\{e_1, ..., e_n)\}) - (\min\{b_1, ..., b_n\}) \leq d$
$\mathcal{S}_{O,\Sigma}(predicate(E_0, ..., E_n)) = true$	ssi $\exists \psi \in \Sigma,\ \psi(o_0) = i_0 \ \wedge\ ... \ \wedge\ \psi(o_n) = i_n \ \wedge\ (i_0, c_0, b_0, e_0), ..., (i_n, c_n, b_n, e_n) \ \in$ O tel que $\mathbf{eval}_{\mathbb{T}}(\psi(predicate(E_0, ..., E_n))) = true$
avec	$o, o' \in \vartheta,\ d \in \mathbb{D},\ [b, e] \in \mathcal{T} \times \mathcal{T},\ f : V^n \rightarrow V,\ predicate \subseteq \mathbf{V}^n,\ n \ \in \ \mathbf{N}$

TABLE 4.6 – Satisfaction des expressions sur un pattern dans la clause *where*

pas commune dans les théories traditionnelles. $\mathcal{S}_{O,\Sigma}$ est une fonction qui associe un ensemble d'observés O, un ensemble de substitutions Σ (notés en indice) et une expression E à une valeur booléenne (*true* ou *false*).

Il est à noter que les conditions temporelles sont juste des prédicats particuliers, et sont donc incluses dans le dernier cas de la sémantique de satisfaction des expressions. Cependant, bien qu'elles relèvent du simple « sucre syntaxique », elles exemplifient un usage utile des conditions les plus communes à l'interrogations de M-Trace. Et c'est dans ce sens que l'interprétation $\mathcal{S}_{O,\Sigma}$ a été délibérément définie en dehors de la définition principale. Ceci permet notamment de mieux modulariser notre théorie et montre qu'il est simple d'intégrer d'autres conditions et même d'utiliser des raisonneurs temporels externes.

Voici un exemple pour expliciter la relation de satisfaction. Soit $\mathbb{T}_0$ une M-Trace ayant le modèle de trace défini dans la figure 4.3 et les éléments suivants :

- obs = { o_1=(5, `Application`,1,2), o_2=(11, `Action`,2,2), o_3=(17, `Webpage`,2,3), o_4=(24, `Action`,3,3), o_5=(26, `Ressource`, 3,4)}.
- $attr$ = { a_1=(5, `Path`,"firefox.exe"), a_2=(11, `Title`,"Open"), a_3=(17, `URL`,"http ://liris.cnrs.fr"), a_4=(24, `Action`,"Open"), a_5=(26, `FilePath`,"F :/equipe/Silex.pdf")}.
- rel = { r_1=(`ref`,5,11), r_2=(`concerns`,11,17), r_3=(`concerns`, 24,26)}.

Prenons par exemple un pattern $P_1 = (x : \texttt{Action})$. Ce pattern est satisfait P_1 par l'interprétation $\mathcal{I}_1 = (O_1, \Sigma_1)$ dans l'intervalle $[2, 2]$, i.e., $\mathcal{I}_1 \models_{\mathbb{M}} P_1^{[2,2]}$ où $\Sigma_1 = \{(x \to 11)\}$ et $O_1 = \{o_2\}$. Nous avons aussi $(\{o_4\}, \{(x \to 24)\}) \models_{\mathbb{M}} P_1^{[3,3]}$. Il n'est pas satisfait dans les autres intervalles.

Pour le pattern $P_2 = (x \texttt{ concerns } y)$, il est satisfait dans $t_1 = [2, 3]$ par l'interprétation $\mathcal{I}_2 = (\{o_2, o_3, r_2\}, \{(x \to 11, y \to 17)\})$ et dans $t_2 = [3, 4]$ par l'interprétation $\mathcal{I}_3 = (\{o_4, o_5, r_3\}, \{(x \to 24, y \to 26)\})$.

En fait P_2 est satisfait aussi par $\mathcal{I}'_2 = (\{o_2, o_3, r_2, o_1\}, \{(x \to 11, y \to 17)\})$ dans t_1. P_2 est également satisfait par $\mathcal{I}'_3 = (\{o_4, o_5, r_3\}, \{(x \to 24, y \to 26, z \to 88)\})$ dans t_2. Dans le premier cas, on s'attend plutôt à $\mathcal{I}_2$ puisque $\mathcal{I}'_2$ contient l'observé o_1 dont l'existence dans l'interprétation est non justifiée. Pour le second, c'est l'ensemble des substitutions dans $\mathcal{I}'_3$ qui contient une affectation de plus pour la variable z. Ainsi, il peut exister une infinité d'interprétations qui peuvent satisfaire un pattern dans un intervalle donnée. Comme il peut n'y avoir aucune interprétation pour satisfaire le pattern dans certains intervalles.

Pour définir le résultat pour l'évaluation d'un pattern, il est nécessaire de préciser quelques points essentiels et inhérents à toute théorie de modèles. En effet, il peut exister zéro ou une infinité d'interprétations dans lesquelles un pattern est satisfait. Ainsi, la définition de la satisfaction d'un pattern dans le tableau 4.5 n'est pas suffisante pour identifier son résultat, il est nécessaire de spécifier de toutes les interprétations, celle qui est attendue et prévue comme résultat (*the intended model*).

Pour les langages de manipulation de connaissances pour la programmation logique ou les bases de donnés déductives comme Datalog [AHV95], le choix d'une interprétation est principalement effectué en se basant sur des hypothèses et des assomptions externes aux expressions considérées. Dans notre contexte, notre choix doit prendre en compte les deux composantes de

l'interprétation : l'ensemble des observés O et l'ensemble des substitutions Σ.

Pour l'ensemble des substitutions, il n'est pas nécessaire de considérer les substitutions arbitraires. Les ensembles de substitutions qui nous intéressent sont ceux qui sont maximaux pour la satisfaction d'un pattern dans une M-Trace en entrée $\mathbb{T}$. Intuitivement, un ensemble de substitutions Σ satisfaisant un pattern P dans $\mathbb{T}$ est *maximal* s'il n'existe pas un ensemble de substitutions Φ satisfaisant P dans $\mathbb{T}$ tel que Σ est un sous ensemble de Φ. Cependant, cette définition informelle ne prend pas en compte qu'il peut exister des ensembles de substitutions qui se différencient que par quelques substitutions et qui contiennent des affectations injustifiées et non pertinentes (puisqu'elles n'interviennent pas dans le pattern). Ainsi, nous définissons l'ensemble maximal de substitutions comme suit.

Définition 21 (Ensemble Maximal de Substitutions)**.** *Soient P un pattern, $\mathbb{T}$ une M-Trace ayant comme modèle de trace $\mathbb{M}$ et Σ un ensemble de substitutions. Un ensemble de substitutions Σ est dit* maximal *pour P dans $\mathbb{T}$ dans l'intervalle $[b, e] \in \mathcal{T} \times \mathcal{T}$ si et seulement si $(O, \Sigma) \models_{\mathbb{M}} P^{[b,e]}$ et $O \subseteq \mathbb{T}$ et s'il n'existe pas un ensemble de substitutions Φ tel que $(O, \Phi) \models_{\mathbb{M}} P$ et $O \subseteq \mathbb{T}$ et $\Sigma_{|\mathbf{var}_P}$ est un sous ensemble de $\Phi_{|\mathbf{var}_P}$ (i.e. $\Sigma_{|\mathbf{var}_P} \subsetneq \Phi_{|\mathbf{var}_P}$).*

Noter que $\Sigma_{|\mathbf{var}_Q}$ désigne la restriction de toutes les substitutions aux variables du pattern P (i.e., $\Sigma_{|\mathbf{var}_P} = \{\psi_{|\mathbf{var}_P} \mid \psi \in \Sigma\}$).

Maintenant, nous allons discuter la question du choix pour l'ensemble des observés O, parmi toutes les interprétations possibles, i.e., des interprétations qui contiennent des observés de la M-Trace $\mathbb{T}$ qui ne sont pas nécessaire pour la satisfaction du pattern. En effet, il est intéressant plutôt d'avoir l'ensemble *minimal* des observés de la M-Trace vérifiant la satisfaction d'un pattern. Le terme *minimal* sous-entend que tous les observés, leurs relations et leurs valeurs d'attribut dans l'interprétation n'existent pas dans cette interprétation sans justification. Ceci n'est pas un problème difficile pour les patterns et requêtes simples mais deviendra plus complexe lorsque nous étendons notre langage avec la négation (*without*) dans le cas des transformations.

Notre idée pour le choix de l'interprétation prévue est simplement qu'elle ne doit pas contenir plus d'observés qu'il est nécessaire pour la satisfaction du pattern pour une M-Trace donnée. Ainsi, l'interprétation attendue et prévue est l'ensemble minimal en terme d'observés par rapport à l'ensemble maximal des substitutions, i.e., égale à l'ensemble des observés pouvant substituer les variables du pattern.

Définition 22 (Ensemble Minimal d'observés satisfaisant un pattern)**.** *Soient P un pattern, $\mathbb{T}$ une M-Trace ayant comme modèle de trace $\mathbb{M}$ et une interprétation (O, Σ). O est l'ensemble minimal des observés satisfaisant P dans $\mathbb{T}$ et dans l'intervalle $[b, e] \in \mathcal{T} \times \mathcal{T}$ si et seulement si Σ est l'ensemble maximal de substitutions dans $\mathbb{T}$ et $O = \bigcup_{\psi \in \Sigma} \mathbf{range}(\psi)$*

Noter que $O \neq \emptyset$ puisqu'un pattern P n'est pas satisfait ($\not\models_{\mathbb{M}} P^{[b,e]}$) s'il n'existe pas de substitution d'observés appartenant à O.

4.9 Sémantique des requêtes simples sur une M-Trace

Maintenant, nous sommes en mesure de définir l'évaluation d'une requête $\mathcal{Q}$ sur une M-Trace $\mathbb{T}$. Le résultat attendu de l'évaluation d'une requête est un ensemble de substitutions affectant des observés à ses variables.

L'évaluation d'une requête $\mathcal{Q} = (\mathbf{var}_{\mathcal{Q}}, P)$ dans un intervalle de temps $[b, e]$ se fait par l'intermédiaire de la satisfaction du pattern P dans $[b, e]$. La définition de satisfaction de pattern prend en compte cependant les observés dont l'occurrence de temps est égale (et non pas incluse) à l'intervalle de temps de leurs évaluations. Elle se comporte de ce point de vue comme un prédicat qui teste si un ensemble d'observés et de substitutions satisferont le pattern dans un intervalle de temps.

Pour exprimer les résultats de l'évaluation d'un pattern par rapport une M-Trace, l'idée est de tester la relation de satisfaction de pattern sur tous les intervalles de temps. De cette manière, nous pouvons définir une interprétation qui satisfait l'exigence de traitement continu d'un pattern, i.e., sur tout les intervalles de temps et sur des M-Traces pouvant être infinies, i.e., ouvertes et non bornées dans le temps. Intuitivement, l'évaluation continue d'un pattern est l'ensemble des substitutions satisfaisant le pattern à tous les intervalles $[b, e]$. Son usage pour exprimer la sémantique des requêtes sur des M-Traces bornées à un seul intervalle de temps nécessite quelques précisions. Pour expliquer la subtilité de cette différence, nous expliciterons les deux cas : l'évaluation continue et l'évaluation ponctuelle sur un intervalle de temps (non continu) d'une requête sur une M-Trace.

4.9.1 Évaluation ponctuelle d'une requête sur une M-Trace

Soient une M-Trace $\mathbb{T}$, une requête $\mathcal{Q} = (\mathbf{var}_{\mathcal{Q}}, P)$. L'évaluation de la requête $\mathcal{Q}$ dans l'intervalle $[b, e] \in \mathcal{T} \times \mathcal{T}$ sur la M-Trace $\mathbb{T}$ ayant comme modèle de trace $\mathbb{M}$, notée $[\![\mathcal{Q}]\!]_{\mathbb{T}}^{[b,e]}$ peut être définie comme suit : $[\![\mathcal{Q}]\!]_{\mathbb{T}}^{[b,e]} = \{\Sigma_{|\mathbf{var}_{\mathcal{Q}}} \mid (O, \Sigma) \models_{\mathbb{M}} P^{[b,e]}\}$. En outre, il nous faudra évidement compléter et préciser quelques points : O est l'ensemble minimal des observés satisfaisant P dans $\mathbb{T}$ (i.e., $O \subseteq \mathbb{T}$ et Σ est l'ensemble maximal de substitutions dans $\mathbb{T}$). Noter aussi que $\Sigma_{|\mathbf{var}_{\mathcal{Q}}}$ désigne juste la restriction de toutes les substitutions aux variables de la requêtes $\mathcal{Q}$ (i.e., $\Sigma_{|\mathbf{var}_{\mathcal{Q}}} = \{\psi_{|\mathbf{var}_{\mathcal{Q}}} \mid \psi \in \Sigma\}$).

Cette définition énonce simplement que si le pattern de la requête est satisfait par un ensemble (minimal) d'observés inclut dans la M-Trace et un ensemble de substitutions Σ alors son résultat est Σ (restreint aux variables de la requête). Cette définition désigne clairement une évaluation ponctuelle d'une requête (ou pattern) puisqu'elle s'effectue une seule fois sur un seul intervalle de temps $[b, e]$.

Cependant, même en considérant le cas d'une M-Trace finie, cette définition échouera à donner le résultat attendu pour certains patterns. Par exemple, prenons une M-Trace contenant un seul observé $o_0 = (i_0, c_0, b_0, e_0)$ et un pattern simple $P =$`(x : c0)`. L'évaluation de ce pattern donnera $(x \to i_0)$ sur l'intervalle $[b_0, e_0]$, et qui est le résultat escompté. Cependant, elle donnera $(x \to \bot)$ dans tout les intervalles $[b_i, e_i] \neq [b_0, e_0]$ alors qu'on s'attend au même résultat notam-

ment sur un intervalle plus grand que $[b_0, e_0]$, i.e., pour des $[b_i, e_i] \supset [b_0, e_0]$. Pour traiter ce cas de figure, nous devons augmenter notre définition d'une évaluation ponctuelle dans $[b, e]$ pour traiter le cas des intervalles $[b', e']$ plus petit que $[b, e]$, i.e. $[b', e'] \sqsubseteq [b, e]$. Ainsi, nous rajouterons cette précision à cette définition : le résultat est l'ensembles des ensembles de substitutions Σ satisfaisant le pattern P dans tous les $[b', e'] \sqsubseteq [b, e]$. Nous aurons donc pour notre pattern dans $t_1 = [b_0, e_0 + 1]$ et $t_2 = [b_0, e_0]$ le même résultat puisque $t_2 \sqsubset t_1$ [39]. Nous pouvons donc définir l'évaluation d'une requête $\mathcal{Q}$ dans un intervalle de temps $[b, e]$ en terme de satisfaction de pattern.

Définition 23 (Évaluation d'une requête dans un intervalle)**.** *Soient une M-Trace $\mathbb{T}$ ayant comme modèle de trace $\mathbb{M}$ et une requête $\mathcal{Q} = (\mathbf{var}_\mathcal{Q}, P)$. L'évaluation de la requête $\mathcal{Q}$ dans l'intervalle $[b, e] \in \mathcal{T} \times \mathcal{T}$ sur la M-Trace $\mathbb{T}$, notée $[\![\mathcal{Q}]\!]_\mathbb{T}^{[b,e]}$ est définie comme suit :*

$$[\![\mathcal{Q}]\!]_\mathbb{T}^{[b,e]} = \{\ \Sigma_{|\mathbf{var}_\mathcal{Q}} \mid (O', \Sigma) \models_\mathbb{M} P^{[b',e']} \text{ pour un intervalle } [b', e'] \sqsubseteq [b, e] \text{ et } O' \text{ est un ensemble minimal d'observés satisfaisant } P \text{ dans } \mathbb{T} \text{ pour l'intervalle } [b', e']\}$$

Compte tenu que la satisfaction de pattern a été définie en ayant l'intention d'une évaluation continue pour prendre en compte des M-Traces représentant un flux d'observés potentiellement infini, cette définition concrétise naturellement le fait qu'un résultat de requête évaluée ponctuellement est égal au résultat de son évaluation continue bornée à l'intervalle de temps donné. Plus précisément, la différence en terme de relation de satisfaction entre les deux manières d'évaluer est que l'évaluation continue décrit les substitutions satisfaites par rapport à tous les intervalles de temps alors que l'évaluation ponctuelle considère seulement les ensembles de substitutions satisfaits dans tous les intervalles de temps inclus dans l'intervalle d'évaluation considéré. Nous proposerons dans la fin de ce chapitre un théorème dans le cas plus complexe des transformations pour concrétiser cette intention et prouver que nos intuitions sur l'évaluation ponctuelle des patterns (et des transformations) sont valides.

En résumé, la sémantique de l'évaluation ponctuelle des requêtes appliquée sur une M-Trace revient à évaluer continuellement la requête sur tous les intervalles inclus dans $[b, e]$. Ceci montre que notre sémantique peut prendre en compte les cas des évaluations continues et ponctuelles, qui sont l'un des verrous principaux pour le traitement des M-Traces.

4.9.2 Modèle pour une requête simple sur une M-Trace

Comme nous l'avons mentionné auparavant, l'évaluation ponctuelle n'est pas suffisante et donc pas réellement de première importance pour le traitement des M-Traces qui peuvent être infinies. Cependant, nous y aurons recours pour indiquer que la définition continue donne le même résultat pour une M-Trace finie (i.e., qui n'évoluera plus ou qui est bornée à un intervalle) [40]. En

39. Il est évident que pour par exemple $[b_0, e_0 - 1]$, nous aurons le résultat attendu ($x \to \bot$) puisque o_0 n'est pas encore fini et donc n'appartient pas à la M-Trace.

40. L'évaluation ponctuelle donne aussi le même résultat qu'une évaluation continue dans le même intervalle d'évaluation avec la simple différence que la seconde évaluation donnera potentiellement plus de résultats si nous élargissons cet intervalle d'évaluation.

effet, nous avons conçu notre sémantique et la relation de satisfaction pour prendre en compte le cas d'une évaluation continue.

Pour décrire une définition pour l'évaluation continue, nous devons préciser un point essentiel par rapport à la définition précédente. Nous devons d'abord définir la satisfaction d'un pattern P pour tout les intervalles de temps. Pour cela, nous définissons, de manière analogue aux théories traditionnelles des langages formels pour la représentation des connaissances, la notion de *modèle de pattern*. Ensuite, sur la base de cette notion, nous définissons l'évaluation continue d'une requête pour une M-Trace. La définition suivante décrit formellement la notion de modèle pour un pattern. Intuitivement, un modèle pour un pattern désigne l'interprétation attendue pour son satisfaction dans un intervalle de temps.

Définition 24 (Modèle d'un pattern)**.** *Soient une M-Trace* $\mathbb{T}$ *ayant comme modèle de trace* $\mathbb{M}$ *et une requête* $\mathcal{Q} = (\mathbf{var}_\mathcal{Q}, P)$*. Une interprétation* $\mathcal{I} = (O, \Sigma)$ *est dite* modèle pour ce pattern *si et seulement si pour tous les* $[b_i, e_i] \in \mathcal{T}^2$*,* $i \geq 0$ *tels que* $(O_i, \Sigma_i) \models_\mathbb{M} P^{[b_i,e_i]}$*, nous avons* $O = \bigcup_{i=0\ldots n} O_i$ *et* $\Sigma = \bigcup_{i=0\ldots n} \Sigma_i$ *avec* O_i *est un ensemble minimal d'observés pour* P *et* Σ_i *est un ensemble maximal de substitutions dans* $[b_i, e_i]$ *pour la M-Trace* $\mathbb{T}$*.*

4.9.3 Évaluation continue d'une requête sur une M-Trace

Pour la définition d'une évaluation continue d'une requête sur une M-Trace, il faudra s'assurer que l'interprétation vérifiant le pattern P est un modèle pour ce pattern pour tous les intervalles de temps et que cette interprétation est toujours incluse dans la M-Trace en cours d'interrogation.

Définition 25 (Évaluation continue d'une requête)**.** *Soient une M-Trace* $\mathbb{T}$ *ayant comme modèle de trace* $\mathbb{M}$ *et une requête* $\mathcal{Q} = (\mathbf{var}_\mathcal{Q}, P)$*. L'évaluation continue de* $\mathcal{Q}$ *dans* $\mathbb{T}$ *est l'ensemble des substitutions* $\Sigma_{|\mathbf{var}_\mathcal{Q}}$ *tel que :*

- *L'interprétation* (O, Σ) *est un modèle pour le pattern* P *pour tous les intervalles de temps, i.e.,* $(O, \Sigma) \models_\mathbb{M} P^{[b,e]}, \forall\ [b, e] \in \mathcal{T}^2$*.*
- *L'ensemble des observés* O *est toujours inclus dans la M-Trace en cours d'interrogation, i.e.,* $O \subseteq \mathbb{T}$*.*

Nous définissons maintenant la notion de *modèle pour une allo-transformation* de M-Traces.

4.10 Sémantique des allo-transformations simples de M-Traces

Avant de définir la sémantique des allo-transformations, nous aurons besoin de préciser l'application d'un ensemble de substitution sur un template.

4.10.1 Substitutions et Template

Le *template* contient les éléments permettant de construire de nouveaux observés. L'évaluation d'un *template* consiste à appliquer des substitutions sur ces éléments afin de générer de nouveaux observés. Les substitutions utilisées résultent de l'évaluation de la partie *pattern* de

la règle. Une règle applique donc les substitutions résultant de l'évaluation du *pattern* sur le *template*.

Définition 26 (Substitution de Template)**.** *Soient un* template G*, une interprétation* (O, Σ) *et un modèle de trace* $\mathbb{M}$*. Nous assumons aussi l'existence d'une fonction* $id()$ *qui génère un identifiant unique pour les observés construits. L'application d'un ensemble de substitution* Σ *à partir d'une interprétation* O *et par rapport à un modèle de trace* $\mathbb{M}' = ((\mathcal{T}, C', R', A', \leq_C, \leq_R, \text{dom}'_R, \text{range}'_R, \text{dom}'_A, \text{range}'_A))$ *sur un template* G*, noté* $\Sigma_{\mathbb{M},O}(G)$ *est définie récursivement comme suit :*

- *si* $G = (w \ : \ c)$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = id() \ \wedge$
  $s = \min\{b_0, ..., b_n\} \ \wedge \ e = \max\{e_0, ..., e_n\}$ *avec* $(\psi(x_0), c_0, b_0, e_0), ..., (\psi(x_n), c_n, b_0, e_0) \in O\}$

- *si* $G = (w : c;\ w.id = E_{id})$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = \mathbf{eval}_{\mathbb{T}}(\psi(E_{id})) \ \wedge$
  $s = \min\{b_0, ..., b_n\} \ \wedge \ e = \max\{e_0, ..., e_n\}$ *avec* $(\psi(x_0), c_0, b_0, e_0), ..., (\psi(x_n), c_n, b_0, e_0) \in O\}$

- *si* $G = (w : c; w.start = E_{id};\ w.end = E_{start})$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = \mathbf{eval}_{\mathbb{T}}(\psi(E_{id})) \ \wedge$
  $s = \mathbf{eval}_{\mathbb{T}}(\psi(E_{start})) \ \wedge \ e = \max\{e_0, ..., e_n\}$ *avec* $(\psi(x_0), c_0, b_0, e_0), ..., (\psi(x_n), c_n, b_0, e_0) \in O\}$

- *si* $G = (w : c; w.start = E_{id};\ w.end = E_{end})$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = \mathbf{eval}_{\mathbb{T}}(\psi(E_{id})) \ \wedge$
  $s = \min\{b_0, ..., b_n\} \ \wedge \ e = \mathbf{eval}_{\mathbb{T}}(\psi(E_{end}))\}$ *avec* $(\psi(x_0), c_0, b_0, e_0), ..., (\psi(x_n), c_n, b_0, e_0) \in O\}$

- *si* $G = (w : c; w.start = E_{start})$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = id() \ \wedge$
  $s = \mathbf{eval}_{\mathbb{T}}(\psi(E_{start})) \ \wedge \ e = \max\{e_0, ..., e_n\}$ *avec* $(\psi(x_0), c_0, b_0, e_0), ..., (\psi(x_n), c_n, b_0, e_0) \in O\}$

- *si* $G = (w : c; w.start = E_{end})$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = id() \ \wedge$
  $s = \min\{b_0, ..., b_n\} \ \wedge \ e = \mathbf{eval}_{\mathbb{T}}(\psi(E_{end}))\}$ *avec* $(\psi(x_0), c_0, b_0, e_0), ..., (\psi(x_n), c_n, b_0, e_0) \in O\}$

- *si* $G = (w : c; w.start = E_{start};\ w.end = E_{end})$ *alors*
  $\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma, \forall\ x_0, ..., x_n \in \mathbf{domain}(\psi), i = id() \ \wedge$
  $s = \mathbf{eval}_{\mathbb{T}}(\psi(E_{start})) \ \wedge \ e = \mathbf{eval}_{\mathbb{T}}(\psi(E_{end}))\}$

– *si* $G = (w : c;\ w.id = E_{id};\ w.start = E_{start};\ w.end = E_{end})$ *alors*
$\Sigma_{\mathbb{M},O}(G) = \{(i, c, s, e) \mid \forall\ \psi \in \Sigma,\ i = \mathbf{eval}_{\mathbb{T}}(\psi(E_{id}))\ \wedge\ s = \mathbf{eval}_{\mathbb{T}}(\psi(E_{start}))\ \wedge\ e = \mathbf{eval}_{\mathbb{T}}(\psi(E_{end}))\}$

– *si* $G = (G'; w.a = E_a)$ *alors*
$\Sigma_{\mathbb{M},O}(G) = \Sigma_{\mathbb{M},O}(G') \cup \{(i, a, v)\ \mid\ \forall\ \psi \in \Sigma,\ (i, a) = \mathbf{eval}_{\mathbb{T}}(\psi(i.a))) \wedge v = \mathbf{eval}_{\mathbb{T}}(\Sigma(E_a))\}$

– *si* $G = (G'; G''; w\ r\ w')$ *alors*
$\Sigma_{\mathbb{M},O}(G) = \Sigma_{\mathbb{M},O}(G')\ \cup\ \Sigma_{\mathbb{M},O}(G'')\ \cup \{(r, i, i')\ \mid\ \forall\ \psi \in \Sigma,\ i = \mathbf{eval}_{\mathbb{T}}(\psi(w))\ \wedge\ i' = \mathbf{eval}_{\mathbb{T}}(\psi(w'))\}$

avec $c \in C',\ r \in R', a \in A'$

La construction de nouveaux observés se fait par application des substitutions résultantes de l'évaluation du pattern de la règle de transformation sur la partie template. La spécification du template peut être simple incluant juste le type de l'observé à construire sans préciser ses bornes temporelles ou son identifiant. Dans ce cas, la sémantique définie prévoie un calcul automatique des valeurs *start*, *end* et *id*. Pour l'identifiant, la fonction *id*() permet de s'assurer qu'un observé a un identifiant unique dans notre sémantique. Pour ce qui est des bornes temporelles de l'observé en cours de construction, la valeur de *start* (respectivement *end*) correspond à la valeur minimale (respectivement maximale) des temps de début (respectivement de fin) des observés ayant servis à sa construction, i.e., les observés identifiés par les substitutions appliquées lors de la construction.

4.10.2 Satisfaction d'une allo-transformation simple de M-Traces

Rappelons d'abord le principe général d'une allo-transformation. Soient une M-Trace $\mathbb{T}$ ayant un modèle $\mathbb{M}$ et une transformation $\tau = \{G_i \leftarrow P_i, i \geq 0\}$. L'évaluation et l'application d'une allo-transformation τ sur la M-Trace $\mathbb{T}$ consiste à produire une M-Trace $\mathbb{T}'$ ayant un modèle de trace $\mathbb{M}'$. Afin de formaliser et définir l'évaluation d'une allo-transformation, nous définissons la relation de satisfaction pour les allo-transformations. Pour cela, nous avons besoin de définir la notion de satisfaction de règle de transformation que nous généralisons par la suite à la satisfaction d'un ensemble de règles (i.e., une transformation).

Avant de commencer, il nous faudra préciser quelques différences par rapport à la satisfaction de pattern. Il est à noter que la spécification d'une allo-transformation et plus spécifiquement d'une règle de transformation dépend de deux interprétations représentants l'ensemble des observés source de la M-Trace à transformée et l'ensemble des observés résultat de la M-Trace transformée. Cette définition de la relation de satisfaction paramétrée par deux interprétations est relativement nouvelle et inhabituelle en comparaison avec les théories traditionnelles pour les langages de manipulation de connaissances.

Définition 27 (Satisfaction d'une règle de transformation M-Trace)**.** *Soit une règle de transformation* $(G \leftarrow P)$ *où* P*,* G *sont associées respectivement aux modèles de trace* $\mathbb{M}, \mathbb{M}'$*. Les interprétations* $\mathcal{I} = (O, \Sigma)$ *et* $\mathcal{I}' = (O', \Sigma')$ satisfont la règle de transformation $(G \leftarrow P)$ *dans l'intervalle* $[b, e] \in \mathcal{T}^2$*, notée* $\mathcal{I}, \mathcal{I}' \models_{\mathbb{M},\mathbb{M}'} (G \leftarrow P)^{[b,e]}$ *si et seulement si :*

- $\mathcal{I}$ *satisfait* P *dans l'intervalle* $[b, e]$*, i.e.,* $\mathcal{I} = (O, \Sigma) \models_{\mathbb{M}} P^{[b,e]}$ *tel que* O *est l'ensemble minimal des observés pour* P *dans* $\mathbb{T}$ *et nous avons* $\Sigma_{\mathbb{M}',O}(G) \subseteq O'$ *ou*
- $\mathcal{I} = (O, \Sigma) \not\models_{\mathbb{M}} P^{[b,e]}$ *pour tous les ensembles* O, Σ*.*

Vous remarquez que l'ensemble des substitutions Σ' n'est pas précisé et peut être donc quelconque. Ceci est évidement dû au fait qu'une transformation produit un ensemble d'observés et donc l'ensemble des substitutions produit n'est pas pertinent. Noter bien qu'il pourrait y avoir plusieurs interprétations O' et que l'interprétation attendue est celle qui est égale à $\Sigma_{\mathbb{M}',O}(G)$. Cependant, nous devons élargir O' pour la définition de satisfaction d'une transformation qui regroupe plusieurs règles et donc une interprétation plus large que celle qui vérifie et ne satisfait qu'une seule règle. Nous pouvons maintenant définir la satisfaction de transformation de M-Trace.

Définition 28 (Satisfaction d'une allo-transformation)**.** *Soit une transformation* τ *constituée d'un ensemble de règles de transformation* $(G_i \leftarrow P_i), i \geq 0$ *où* P_i*,* G_i *sont associées respectivement aux modèles de trace* $\mathbb{M}, \mathbb{M}'$*. Les interprétations* $\mathcal{I} = (O, \Sigma)$ *et* $\mathcal{I}' = (O', \Sigma')$ satisfont l'allo-transformation τ *dans l'intervalle* $[b, e]$*, notée* $\mathcal{I}, \mathcal{I}' \models_{\mathbb{M},\mathbb{M}'} \tau^{[b,e]}$ *si et seulement si pour toutes les règles* $(G_i \leftarrow P_i) \in \tau$*,* $(G_i \leftarrow P_i)$ *est satisfaite par* $\mathcal{I}, \mathcal{I}'$ *dans l'intervalle* $[b, e]$*, i.e.,* $\mathcal{I}, \mathcal{I}' \models_{\mathbb{M},\mathbb{M}'} (G_i \leftarrow P_i)^{[b,e]}, \forall\ (G_i \leftarrow P_i)$.

Dans cette définition, il pourrait y avoir plusieurs interprétations O' satisfaisant l'allo-transformation. La définition 29 identifie l'ensemble minimal satisfaisant une allo-transformation.

Définition 29 (Ensemble Minimal d'observés pour une allo-transformation)**.** *Pour une allo-transformation* $\tau = \{G_0 \leftarrow P_0, ..., G_n \leftarrow P_n\}$ *et* $\mathcal{I} = (O, \Sigma)$*,* $\mathcal{I}' = (O', \Sigma')$ *deux interprétations telles que* $\mathcal{I}, \mathcal{I}' \models_{\mathbb{M},\mathbb{M}'} (G_i \leftarrow P_i)^{[b,e]}$ *avec* $[b, e] \in \mathcal{T}^2$*,* O' *est l'ensemble minimal des observés satisfaisant l'allo-transformation* τ *dans l'intervalle* $[b, e]$ *si et seulement si* $O' = \bigcup_{(G_i \leftarrow P_i) \in \mathcal{T}} \Sigma_{\mathbb{M}',O}(G_i)$

Cette définition concrétise le fait que l'interprétation minimale que nous cherchons à obtenir est égale à l'ensemble des interprétations satisfaisant les règles de transformation de manière exacte sans ajout d'observés non justifiés. Donc elle est minimale au sens ensembliste correspondant, au niveau des règles, à l'ensemble des observés générés par l'application des substitution obtenues sur les templates, i.e., égale à $\Sigma_{\mathbb{M}',O}(G_i)$ et donc à l'union de ces ensembles au niveau de la transformation. Noter que la définition de l'interprétation minimale pour les auto-transformations est loin d'être triviale à définir et nécessite de ce fait l'expression d'un théorème (que nous donnerons dans la section 4.11).

4.10.3 Évaluation ponctuelle d'une allo-transformation de M-Trace

Comme nous l'avons fait pour les patterns, nous pouvons définir une évaluation ponctuelle pour les allo-transformations. Moyennant la définition de l'évaluation ponctuelle des patterns, nous pouvons construire l'ensemble des observés transformés en appliquant les substitutions résultantes sur les templates.

Définition 30 (Évaluation ponctuelle d'une allo-transformation dans un intervalle)**.** *Soient une M-Trace* $\mathbb{T}$ *ayant un modèle de trace* $\mathbb{M}$ *et une transformation* $\tau = \{G_i \leftarrow P_i, i \geq 0\}$ *où tous les templates* G_i *sont exprimés sur un modèle de trace* $\mathbb{M}'$*. L'évaluation ponctuelle de l'allo-transformation* τ*, notée* $[\![\tau]\!]^t_{\mathbb{T}}$ *est définie comme suit :*

$$[\![\tau]\!]^t_{\mathbb{T}} = \{\mathbb{T}' \mid \mathbf{model}_{\mathbb{T}'} = \mathbb{M}' \text{ et } \mathbf{obs}_{\mathbb{T}'} \cup \mathbf{rel}_{\mathbb{T}'} \cup \mathbf{attr}_{\mathbb{T}'} = \bigcup_{i=1\ldots n} \bigcup_{k=1\ldots m} \Sigma^k_{\mathbb{M}'}(G_i) \text{ pour tous les } \Sigma^k \in [\![P_i]\!]^t_{\mathbb{T}} \}$$

Rappelons que l'évaluation ponctuelle n'est pas suffisante puisque les M-Traces prises en compte peuvent être en cours de collecte et/ou de transformation et donc théoriquement infinies. L'exploitation des M-Traces nécessite une évaluation continue permettant de simuler de manière équivalente l'évaluation ponctuelle dans le cas d'une M-Trace finie, i.e., ne considérant plus d'ajouts d'observés.

4.10.4 Modèle pour une allo-transformation simple de M-Traces

Nous allons définir la notion de modèle pour les allo-transformations qui nous servira à définir l'évaluation continue d'une allo-transformation de M-Traces.

Définition 31 (Modèle pour une allo-transformation)**.** *Soit une transformation* τ *constituée d'un ensemble de règles de transformation* $(G_i \leftarrow P_i), i \geq 0$ *où* P_i*,* G_i *sont associées respectivement aux modèles de trace* $\mathbb{M}, \mathbb{M}'$*. Les interprétations* $\mathcal{I} = (O, \Sigma)$ *et* $\mathcal{I}' = (O', \Sigma')$ satisfont l'allo-transformation τ *et représentent un* modèle pour cette allo-transformation *si et seulement si pour toutes les règles* $(G_i \leftarrow P_i) \in \tau$ *et pour tous les intervalles* $[b, e] \in \mathcal{T}^2$*,* $(G_i \leftarrow P_i)$ *est satisfaite par* $(\mathcal{I}, \mathcal{I}')$*, i.e.,*

$$\mathcal{I}, \mathcal{I}' \models_{\mathbb{M},\mathbb{M}'} (G_i \leftarrow P_i)^{[b,e]},\ \forall\ [b, e] \in \mathcal{T}^2,\ \forall\ (G_i \leftarrow P_i) \in \tau$$

La notion de modèle pour une allo-transformation assure l'existence d'un résultat pour tous les intervalles. Pour un intervalle de temps $[b, e]$, s'il n'y pas d'observés dans l'interprétation d'entrée (la M-Trace source), les règles de transformation sont toutes satisfaites (selon la définition 27). Ceci permet de représenter l'évaluation continue de manière naturelle et simple : si le résultat d'un pattern dans l'interprétation d'entrée est satisfait alors le résultat de son application sur les templates est inclus dans l'interprétation de sortie $\mathcal{I}'$ sinon (si le pattern ne donne pas de résultats dans cet intervalle), rien ne change dans $\mathcal{I}'$.

Définition 32 (Évaluation d'une allo-transformation)**.** *Soient les M-Traces* $\mathbb{T}, \mathbb{T}'$ *correspondant respectivement à la M-Trace source et résultat.* $\mathbb{T}$ *et* $\mathbb{T}'$ *disposent respectivement du modèle de*

trace $\mathbb{M}$ *et* $\mathbb{M}'$*. Soit une transformation* τ *constituée d'un ensemble de règles de transformation* $(G_i \leftarrow P_i), i \geq 0$ *où* P_i*,* G_i *sont associées respectivement aux modèles de trace* $\mathbb{M}, \mathbb{M}'$*. L'évaluation continue de l'allo-transformation* τ *de* $\mathbb{T}$ *est la M-Trace* $\mathbb{T}'$ *telle que :*

- *Les interprétations* $\mathcal{I} = (O, \Sigma)$ *et* $\mathcal{I}' = (O', \Sigma')$ *sont un modèle pour l'allo-transformation* τ *;*
- *L'ensemble des observés* O *est toujours inclus dans la M-Trace source* $\mathbb{T}$ *en cours de transformation, i.e.* $O \subseteq \mathbb{T}$ *;*
- *L'ensemble des observés* O' *contient toujours la M-Trace transformée résultat, i.e.* $\mathbb{T}' \subseteq O'$*.*

Enfin, la définition précédente définit aussi de manière abstraite les étapes à suivre pour calculer la sémantique des transformations. Étant donné une transformation τ, il suffit de considérer toutes les interprétations qui sont sous-ensembles de la M-Trace transformée $\mathbb{T}'$, trouver ceux qui sont des modèles pour P et contiennent $\mathbb{T}$. Toutefois, il s'agit clairement d'une procédure difficilement réalisable dans une implémentation. La sous-section 4.11.1 fournira un algorithme général plus raisonnable sous forme d'une théorie de point fixe.

Nous avons défini une sémantique pour les patterns et transformations simples, i.e., n'utilisant pas l'opérateur de négation et les relations temporelles relatives (e.g. `extend`, etc.). Dans la suite de ce chapitre, nous allons définir un théorie des modèles, étendant la sémantique définie précédemment, pour les transformations impliquant la négation qualifiées de *transformations complexes*. La complexité des transformations vient du fait que la négation est non monotone et donc problématique pour les techniques d'évaluations continues.

Nous allons dans la suite étudier et proposer une théorie des modèles pour les transformations augmentées avec la négation dans le cas d'une auto-transformation de M-Trace. Sous cette considération, la complexité est plus importante puisque des transformations récursives peuvent se produire. Plus précisément, ceci nous permet d'étudier et de traiter le cas classique et complexe des règles combinant la récursion à la négation. En particulier, ce thème a été abondamment étudié par la communauté des bases de données déductives notamment pour le langage Datalog [AHV95, Chap. 14] offrant ainsi beaucoup de résultats théoriques intéressants que nous étendrons ou réutiliserons. Ceci permettra également de se positionner par rapport aux théories des modèles classiques et facilitera ainsi les définitions, les comparaisons et les explications.

La théorie des modèles a la particularité de permettre de nombreux modèles pour une auto-transformation d'une M-Trace. Pour donner une sémantique précise pour les requêtes et les auto-transformations complexes de M-Traces, nous devons spécifier un modèle unique. Une façon courante et pratique pour obtenir un tel modèle unique est de définir une théorie de point fixe fondée sur la théorie du modèle. L'interprétation du point fixe spécifiée par la théorie de point fixe sera le modèle d'une auto-transformation.

Lorsque les caractéristiques non-monotones comme la négation sont combinées à la récursivité des règles dans une même M-Trace, des techniques bien connues de stratification seront utilisées. En nous limitant à des transformations stratifiables, nous éviterons de tels cas et nous veillerons à ce qu'une interprétation de point fixe unique existe. Une définition formelle de la stratification

est donnée avec la définition de l'interprétation de point fixe dans la section 4.11.1.

4.11 Sémantique des auto-transformations complexes

Nous avons défini en détail la sémantique des requêtes et des transformations simples de M-Traces. Nous traitons dans cette section les transformations dites complexes car usant de l'opérateur de *négation*. En fait, cette désignation de « transformation complexe » correspond à la difficulté de l'évaluation et non à la difficulté d'expression. En effet, la négation est bien connue pour être un élément primordial du pouvoir descriptif d'un langage et son usage dans notre langage a pour objectif de faciliter la description intuitive de patterns lors de l'exploitation de traces. Ceci est particulièrement important pour des transformations qui se veulent être intelligibles et proches le plus possible des usages et des pratiques à base de traces.

Cependant, l'usage de la négation dans les auto-transformations d'une M-Trace pose au moins deux problèmes :

- *Modèle minimal.* Considérer une seule interprétation implique une possible récursion dans les règles. La négation introduit des problèmes bien connus quand elle est combinée avec la récursivité des règles notamment pour choisir le modèle minimal satisfaisant toutes ces règles.
- *Monotonie de l'évaluation.* La négation est bien connue aussi pour être non monotone et donc problèmatique pour une évaluation continue des transformations sur des M-Traces pouvant être non bornées (voir la discussion du chapitre 3).

Il nous faudra donc définir un langage et une sémantique qui prennent en compte et répondent à ces deux problèmes. Nous aurons donc à prouver (en énonçant deux théorèmes) pour ces deux points que notre langage satisfait réellement les exigences requises pour l'exploitation des M-Traces.

Nous commençons d'abord par la définition de la notion de satisfaction d'une règle de transformation complexe.

Définition 33 (Satisfaction d'une règle de transformation complexe)**.** *Soient une interprétation* $\mathcal{I} = (O, \Sigma)$ *et une règle de transformation* $(G \leftarrow P)$*.* $\mathcal{I}$ satisfait la règle $(G \leftarrow P)$*, si et seulement si elle respecte la définition des tableaux 4.7 et 4.8.*

Nous devons préciser que $\Sigma'_{\mathbb{M},O}(G)^{[b,e]}$ est une restriction de $\Sigma'_{\mathbb{M},O}(G)$ telle que $\forall\ (i, c, b', e') \in \Sigma'_{\mathbb{M},O}(G), [b', e'] \sqsubseteq [b, e]$.

Rappelant d'abord le principe général d'une auto-transformation d'une M-Trace. Étant donné une transformation τ et une M-Trace $\mathbb{T}$, nous voulons connaître tous les observés qui ont été transformés et dérivés par une règle de transformation de τ. Cela signifie que nous devons trouver une interprétation qui contient la M-Trace $\mathbb{T}$ et satisfait toutes les règles de τ. Une telle interprétation est appelée un *modèle*. Cependant, il existe de nombreux modèles (en fait, un nombre infini) pour une transformation τ donnée et une M-Trace $\mathbb{T}$. De ces modèles, nous choisirons un seul modèle en se basant sur une interprétation de point fixe qui sera le sujet

$(O, \Sigma) \models_{\mathbb{M}} (P)^{[b,e]}$	selon la définition 20 page 111
$(O, \Sigma) \models_{\mathbb{M}} (\texttt{while}\ o\ :\ \texttt{without}\ P)^{[b,e]}$	ssi $\exists \psi \in \Sigma$ tel que $\psi(o) = i \ \wedge\ (i, c', b', e') \in O \ \wedge\ [b', e'] = [b, e] \wedge$ pour tous les intervalles $[b'', e''] \sqsubseteq [b, e]$ nous avons $(O, \Sigma) \not\models_{\mathbb{M}} P^{[b'', e'']}$
$\mathcal{I} \models_{\mathbb{M}} (G \leftarrow P)^{[b,e]}$	ssi (1) $\Sigma'_{\mathbb{M},O}(G)^{[b,e]} \subseteq O$ avec Σ' est un ensemble maximal de substitutions pour P et $(O, \Sigma') \models_{\mathbb{M}} P^{[b,e]}$ ou (2) $(O, \Sigma') \not\models_{\mathbb{M}} P^{[b,e]}$ pour tous O et Σ'
avec	$o, o' \in \vartheta,\ c \in C,\ r \in R,\ [b, e] \in \mathcal{T} \times \mathcal{T},\ n \in \mathbf{N}$,

TABLE 4.7 – Satisfaction d'une règle de transformation complexe

la section suivante. Maintenant, nous définissons formellement la notion de modèle pour une auto-transformation complexe.

Définition 34 (Modèle pour une auto-transformation complexe d'une M-Trace)**.** *Soient une M-Trace* $\mathbb{T}$ *et une transformation* τ *qui est un ensemble de règles de transformation* $(G_i \leftarrow P_i), i > 0$*. Une interprétation* $\mathcal{I} = (O, \Sigma)$ satisfait une auto-transformation τ *et dite* modèle pour cette auto-transformation *dans* $\mathbb{T}$ *si et seulement si :*

- $\mathcal{I}$ *satisfait toutes les règles* $(G_i \leftarrow P_i) \in \tau$ *pour tout les intervalles* $[b, e] \in \mathcal{T} \times \mathcal{T}$*, i.e,.* $\mathcal{I} \models_{\mathbb{M}} \tau^{[b,e]}, \forall\ [b, e] \in \mathcal{T} \times \mathcal{T}, \forall\ (G_i \leftarrow P_i) \in \tau$*. Et*
- $\mathcal{I}$ *contient la M-Trace transformée, i.e.,* $\mathbb{T} \subseteq O$*.*

En analysant de près la relation de satisfaction, on peut voir que l'ensemble des substitutions Σ n'est en fait pas pertinent pour savoir si une interprétation donnée $\mathcal{I}$ est un modèle ou pas pour une auto-transformation, car ceci dépend que de O. En effet, $\mathcal{I}$ doit satisfaire toutes les règles de transformation et l'application de ces règles dans la relation de satisfaction (dernier cas dans la table 4.7) ne se réfère pas à l'ensemble Σ dans sa définition (seulement à l'existence ou la non-existence de certains Σ'). Nous pouvons donc identifier la notion de modèle pour auto-transformation avec uniquement la partie O de l'interprétation. Dans la suite de cette section, nous considérons l'interprétation $\mathcal{I} = O$ et utiliserons $\mathcal{I}$ et O comme étant interchangeables dans le cas où cette équivalence n'est pas explicitement mentionnée.

4.11.1 Théorie du point fixe pour les transformations complexes

Nous venons de définir une théorie des modèles pour les transformations de M-Traces. Toutefois, il existe de nombreux modèles pour une auto-transformation donnée. Une approche commune et pratique pour obtenir un modèle unique est de définir une théorie du point fixe. Elle a été proposée pour la première fois par Van Emden et Kowalski [VEK76] pour la sémantique du calcul des prédicats. Nous proposons son usage pour le calcul des auto-transformations notamment pour calculer le modèle minimal attendu. Le modèle attendu est le plus petit point fixe de l'opérateur de conséquence immédiate Γ, qui dérive de nouveaux observés à partir des observés connus (en se fondant sur la théorie des modèles).

$(O,\Sigma) \models_{\mathbb{M}} (o \texttt{ extend } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2\ \wedge\ e_1 = e_2 \oplus d$
$(O,\Sigma) \models_{\mathbb{M}} (o \texttt{ shorten } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (\psi,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2\ \wedge\ e_1 = e_2 \ominus d$
$(O,\Sigma) \models_{\mathbb{M}} (o \texttt{ extend-start } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2 \ominus d\ \wedge\ e_1 = e_2$
$(O,\Sigma) \models_{\mathbb{M}} (o \texttt{ shorten-start } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2 \oplus d\ \wedge\ e_1 = e_2$
$(O,\Sigma) \models_{\mathbb{M}} (o \texttt{ shift-forward } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2 \oplus d\ \wedge\ e_1 = e_2 \oplus d$
$(O,\Sigma) \models_{\mathbb{M}} (o \texttt{ shift-backward } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2 \ominus d\ \wedge\ e_1 = e_2 \ominus d$
$\mathcal{I} \models_{\mathbb{M}} (o \texttt{ from-end } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = e_2\ \wedge\ e_1 = e_2 \oplus d$
$\mathcal{I} \models_{\mathbb{M}} (o \texttt{ from-end-backward } (o'))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = e_2 \ominus d\ \wedge\ e_1 = e_2$
$\mathcal{I} \models_{\mathbb{M}} (o \texttt{ from-start } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2\ \wedge\ e_1 = b_2 \oplus d$
$\mathcal{I} \models_{\mathbb{M}} (o \texttt{ from-start-backward } (o',d))^{[b,e]}$	ssi $\exists\, \psi \in \Sigma, \psi(o) = i_1, \psi(o') = i_2,\ (i_1,c_1,b_1,e_1),(i_2,c_2,b_2,e_2)\ \in O \wedge$ $b_1 = b\ \wedge\ e_1 = e\ \wedge\ b_1 = b_2 \oplus d\ \wedge\ e_1 = b_2$
avec	$o, o' \in \vartheta,\ d \in \mathbb{D},\ [b,e] \in \mathcal{T} \times \mathcal{T}$

TABLE 4.8 – Satisfaction des relations temporelles relatives

Cependant, les fonctionnalités non monotones telles que la négation et l'agrégation introduisent des problèmes bien connus quand elles sont combinées avec la récursivité des règles. En particulier, il pourrait n'y avoir aucun point fixe pour certaines auto-transformations, et un ou plusieurs points pour d'autres. Afin de veiller à l'existence d'un seul et unique point fixe, nous limitons les transformations de telle manière quelles soient stratifiables. Il s'agit d'une approche commune et bien connue en programmation logique introduite d'abord dans [ABW88].

En plus de donner une sémantique sans ambiguïté à des transformations stratifiables, la théorie des points fixes décrit également une simple et abstraite méthode d'évaluation en chaînage avant, qui peut être facilement étendue pour le traitement incrémental et continu exigé pour la manipulation de M-Traces.

Reprenant l'exemple montrant l'intérêt de la stratification. Considérerons la M-Trace $\mathbb{T}$ tel que $O = \{(11, c_1, 2, 4)\}$ ayant un modèle de trace $\mathbb{M}_1$ où $C = \{c_1\}$. Considérons aussi l'exemple de transformation τ_1 composée de deux règles (figure 4.11.1).

```
Construct                          Construct
  (w: p)                             (u : q)
On                                 On
  (x : c1) and                       (x : c1) and
  while x: without (z : q)           while x: without (z : p)
```

TABLE 4.9 – Exemple d'une auto-transformation complexe de M-Trace contenant deux règles

Les interprétations $O_1 = \{(11, c_1, 2, 4), (12, p, 2, 4)\}$ et $O_1 = \{(11, c_1, 2, 4), (14, q, 2, 4)\}$ sont toutes les deux des *modèles* pour l'auto-transformation τ_1 pour $\mathbb{T}$. Parce que les deux modèles sont symétriques, il n'existe pas de critère claire pour choisir un modèle et pas un autre comme étant le modèle attendu. Ceci est une difficulté commune et inhérente lorsque des règles sont combinées à la négation. En fait, cette exemple est juste une adaptation de l'exemple standard (non spécifique aux M-Traces) $p \leftarrow \neg q, q \leftarrow \neg p$ décrit pour les langages pour la programmation logique et les bases de données déductives. Une solution simple et bien établie pour éviter de telles situations requière que les programmes soient stratifiables.

La stratification restreint l'usage de la récursion dans les règles d'un programme P par des ensembles appelés strates (ensemble de règle P_i avec $P = P_1 \uplus ... \uplus P_n$ [41]) de façon à ce que chaque règle d'une strate donnée ne puisse dépendre que des règles des strates inférieures (ou de la même strate dans certains cas). Dans le contexte des M-Traces deux types de stratifications sont requis :

1. Stratification de la négation : les observés négatifs (i.e., qui sont touchés par la négation) dans un pattern de règle ne peuvent être construits que par des règles de strates inférieures. Les observés qui se produisent de façon positive ne peuvent être construits que par des règles des strates inférieures ou des règles de la même strate.
2. Stratification temporelle : si le pattern d'une règle utilise les relations temporelles relatives

41. $A \uplus B$ implique toujours que $A \cap B = \emptyset$

(e.g., `extend[x,1min]`) alors l'observé de l'ancrage temporel (dans ce cas `x`) ne peut être construit que dans les strates inférieures.

Bien que la stratification de la négation soit assez standard, la stratification temporelle est une exigence propre aux transformations complexes. Nous ne savons pas si une telle notion a été considérée dans d'autres travaux de formalisation de règles.

Pour définir la stratification, il nous faut d'abord définir la notion de dépendance.

4.11.2 Dépendances entre règles de transformation complexe

Nous disons qu'une certaine règle de transformation g dépend d'une autre règle g' si la règle g interroge (potentiellement) des observés qui ont été construit par g'. Nous distinguons différents types de dépendance :

- $g = G \leftarrow P \in \tau$ *dépend temporellement* de $g' = G' \leftarrow P' \in \tau$ s'il existe une partie p du pattern P satisfaisant un observé o (ou plusieurs) qui est utilisée ailleurs dans P pour définir une relation temporelle relative et $\exists t, \exists ob \in O$ tel que $(O, \Sigma) \models_{\mathbb{M}} p^t$ et $\exists ob' \in \Sigma(G)$ tel que $id = id'$ ou $c \leq_C c'$ ou $r \leq_R r'$
- $g = G \leftarrow P \in \tau$ *dépend négativement* de $g' = G' \leftarrow P' \in \tau$ s'il existe une partie p du pattern P qui se produit négativement i.e., à l'intérieur d'un `without` et $\exists t, \exists ob \in O$ tel que $(O, \Sigma) \models_{\mathbb{M}} p^t$ et $\exists ob' \in \Sigma(G)$ tel que $id = id'$ ou $c \leq_C c'$ ou $r \leq_R r'$
- $g = G \leftarrow P \in \tau$ *dépend positivement* de $g' = G' \leftarrow P' \in \tau$ si g ne dépend pas de g' par aucune des dépendances précédentes et $\exists ob' \in \Sigma(G)$ tel que $id = id'$ ou $c \leq_C c'$ ou $r \leq_R r'$

La stratification est une technique classique d'abord proposée par [ABW88] pour définir une classe de programmes logiques où les éléments non-monotones comme la négation ne peuvent être définis de manière déclarative. L'idée principale de la stratification est d'interdire des programmes avec une récurrence sur les requêtes négatives, contribuant à décrire des programmes indésirables. Bien que cette exigence soit très stricte pour les transformations, ses avantages sont multiples : elle est facile à comprendre et peut être vérifiée par des moyens purement syntaxique sans tenir compte de connaissances externes ne faisant pas partie du programme. Plusieurs améliorations de la notion de stratification ont été proposées, par exemple, stratification locales [Prz88] qui permettent certains types de récursivité, mais en pratique, ces approches ont souvent besoin de « connaissances supplémentaires » sur le programme ou les ressources demandés.

Les transformations étudiées dans cette thèse sont toujours considérées comme stratifiables. De plus, la notion de stratification est non seulement utilisée pour le traitement approprié de la négation, mais elle est étendue également à des règles usant de spécifications temporelles relatives, i.e., une sorte de spécification de fenêtres temporelles similaires à celles utilisées par les systèmes de gestion de flux de données. Ces fenêtres temporelles étant connues pour être des régulateurs de la monotonie de l'évaluation des requêtes [GO05, GO03] définissent en général des comportements indésirables des transformations notamment lorsqu'elles sont considérées en présence de récursivité induite par des règles associées à des dépendances temporelles .

4.11.3 Stratification des Transformations Complexes

Dans la suite, $\tau = \tau_1 \uplus ... \uplus \tau_n$ décrit le découpage de la transformation τ en sous-ensembles disjoints de règles de transformation $\tau_1, ..., \tau_n$. Une stratification d'une transformation $\tau = \tau_1 \uplus ...\uplus\tau_n$ est un partitionnement des règles de τ tel que pour chaque paire de règles $g = G \leftarrow P \in \tau_i$, $g' = G \leftarrow P \in \tau_j$:

- si g dépend temporellement de g', alors $i > j$,
- si g dépend négativement de g', alors $i > j$,
- si g dépend positivement de g', alors $i \geq j$

Une transformation complexe est désignée stratifiable, s'il existe une stratification pour elle. En construisant un graphe de dépendance tel que souligné dans [ABW88], nous pouvons vérifier si une transformation donnée est en fait stratifiable et obtenir une stratification pour elle. Le graphe de dépendance est parfois aussi appelé graphe de précédence [AHV95].

La restriction à des transformations stratifiables est nécessaire pour la sémantique du point fixe définie dans la section suivante. Noter que ce n'est pas une restriction très sévère dans le domaine des transformations de M-Traces. Il serait d'ailleurs envisageable de lever cette restriction aux transformations stratifiables, cependant avec comme conséquence une sémantique et une évaluation plus complexe. Des approches dans ce sens ont été étudiés en profondeur dans les travaux de recherche liés à la programmation logique et les bases de données déductives. L'extension de la sémantique des transformations stratifiées est possible avec les approches mises en place dans la programmation logique mais en dehors du champ d'application de ce travail.

4.11.4 Opérateur de conséquence immédiate

L'idée de base pour l'obtention de l'interprétation de points fixes d'une auto-transformation stratifiée est d'appliquer les règles strate par strate : appliquer d'abord les règles de la strate la plus basse sur la M-Trace entrante, puis appliquer les règles de la strate immédiatement supérieur sur le résultat, et ainsi de suite jusqu'à la plus haute strate.

Ceci nécessite la définition d'un opérateur pour calculer la conséquence immédiate d'une auto-transformation. Nous définissons cet opérateur appelé *opérateur de conséquence immédiate* comme suit.

Définition 35 (Opérateur de conséquence immédiate Γ_τ)**.** *Soit τ une auto-transformation. Un opérateur de conséquence immédiate Γ_τ est défini comme suit :*

$$\Gamma_\tau(O) = O \cup \{ob \mid \exists\, (G \leftarrow P) \in \tau \text{ et un ensemble maximal de substitutions } \Sigma \text{ tel que } (O, \Sigma) \models_{\mathbb{M}} (G \leftarrow P)^{[b,e]} \text{ et } ob \in \Sigma(G)\}$$

L'application répétée de Γ_τ jusqu'à ce qu'un point fixe soit atteint est notée Γ_τ^w. Un point fixe signifie qu'une interprétation O telle que $\Gamma_\tau(O) = O$. Comme nous le verrons par la suite, Γ_τ^w est aussi le plus petit point fixe. Parce que nous sommes intéressés par les points fixes de

l'opérateur de conséquence immédiate, il est parfois aussi appelé opérateur de point fixe. Nous noterons par la suite par souci de lisibilité $\Gamma_{\tau}(\mathbb{T})$ au lieu de $\Gamma_{\tau}(\mathbf{obs}_{\mathbb{T}} \cup \mathbf{rel}_{\mathbb{T}} \cup \mathbf{attr}_{\mathbb{T}})$

4.11.5 Interprétation des points fixes

Soit $\overline{\tau}_i = \bigcup_{j \leq i} \tau_j$ désignant l'ensemble des règles dans le strate τ_i et les strates inférieures. L'interprétation de point fixe Γ_{τ} d'une transformation stratifiée telle que $\tau = \tau_1 \uplus ... \uplus \tau_n$ pour la M-Trace $\mathbb{T}$ est définie par le calcul de points fixes strate par strate comme suit.

Définition 36 (Interprétation de point fixe pour une auto-transformation complexe)**.** *Soit une auto-transformation* $\tau = \{\tau_1, ..., \tau_n\} (n \geq 0)$*. Une interprétation* $M_{\tau,\mathbb{T}}$ *est définie comme suit :*

$$
\begin{aligned}
M_0 &= \mathbb{T} = \Gamma^w_{\emptyset}(\mathbb{T}) \\
M_1 &= \Gamma^w_{\overline{\tau}_1}(M_0) \\
&\cdot \\
&\cdot \\
&\cdot \\
M_n &= \Gamma^w_{\overline{\tau}_n}(M_{n-1})
\end{aligned}
$$

avec $M_{\tau,\mathbb{T}} = M_n$

L'interprétation de point fixe $M_{\tau,\mathbb{T}}$ est aussi appelé *le modèle attendu* (*the intended model*) de τ dans $\mathbb{T}$ et spécifie la sémantique déclarative pour les auto-transformations complexes. Le théorème 1 dans la section suivante montrera que cette sémantique est, en effet, bien définie et non ambiguë et justifie donc notre définition. Comme nous le verrons, un théorème similaire doit généralement être prouvé également pour la sémantique du point fixe pour les langages de règles non spécifique aux M-Traces.

La sémantique du point fixe montre de manière explicite une évaluation en chaînage avant des auto-transformations, qui est la méthode d'évaluation de choix pour les programmes logiques à base de règles. La principale différence entre la sémantique du point fixe et une méthode d'évaluation est que l'évaluation proprement dite devrait fonctionner de manière continue, i.e., opère uniquement sur la partie du flux d'observés reçu jusqu'à maintenant. D'ailleurs le théorème 2 vient étayer et montrer que c'est en effet possible. Noter que ce n'est pas évident : un langage de requête et/ou de transformation non-spécifique aux M-Traces (comme Datalog [AHV95]) qui prend en charge les caractéristiques non-monotones comme la négation ne peut être utilisé dans le cas où une partie seulement de la base des faits est connue.

Le théorème que nous allons prouver par la suite énonce que la sémantique fournie par l'interprétation des points fixes définie dans la définition 36 est bien définie et non ambiguë. Il montre également que cette sémantique correspond bien à notre intuition que tous les observés

contenus dans la M-Trace transformée sont justifiés soit en étant dans la M-Trace initiale entrante ou étant dérivés par une règle de transformation.

Théorème 1. *Pour une auto-transformation stratifiable τ et une M-Trace $\mathbb{T}$, l'interprétation $M_{\tau,\mathbb{T}}$ est un modèle minimal pour τ dans $\mathbb{T}$. Ce modèle $M_{\tau,\mathbb{T}}$ est aussi indépendant de la stratification de τ.*

Le terme « minimal » dans le théorème implique donc que tous les observés, leurs relations et valeurs d'attribut dans l'interprétation sont soit dans la M-Trace entrante soit dérivées par des règles de transformation, i.e., qu'aucun observé, relation ou valeur d'attribut n'a pas été ajouté sans justification.

Noter que ce théorème n'est pas spécifique à notre langage de transformation de traces. Des théorèmes semblables doivent être spécifiés également pour les langages de représentation de connaissances non spécifique aux traces. En fait, la preuve donnée dans la suite est une adaptation de la preuve standard de [Llo87] appliquée à notre langage.

Le plus intéressant dans le contexte des auto-transformations de M-Traces est que notre notre sémantique est bien adaptée à l'évaluation continue de M-Traces. En effet, nous pouvons montrer que la théorie des modèles et la sémantique du point fixe prennent en compte des M-Traces potentiellement infinies. Le théorème 2 suivant justifie une évaluation en continue, où les résultats des auto-transformations complexes de M-Traces sont générés *« en ligne »* de telle manière à ne jamais attendre la fin du flux des observés. Ceci est particulièrement important étant donné qu'une M-Trace peut être conceptuellement infinie et donc n'ayant pas fin du tout.

En particulier, ce théorème garantit qu'un observé (i, c, b, e) peut être détecté à l'instant de fin e puisqu'aucune connaissance de tous les observés futures à l'instant e n'est requise ou nécessaire. Veiller à ce que les méthodes d'évaluation d'auto-transformations ne deviennent pas une simple évaluation statique ponctuelle figée est évidemment une condition importante et un exemple où nous pouvons utiliser notre sémantique déclarative pour prouver des déclarations intéressantes sur les auto-transformations complexes de M-Traces.

Théorème 2. *Soit $\mathbb{T} \mid [b, e]$ une restriction d'une M-Trace à l'intervalle $[b, e]$, i.e. $\mathbb{T} \mid [b, e]$ est la M-Trace $\mathbb{T}$ tel que $\forall (i, c, b_i, e_i) \in \mathbf{obs}_{\mathbb{T}}, [b_i, e_i] \sqsubseteq [b, e]$. De la même manière, soit $M \mid [b, e]$ la restriction d'une interprétation M à l'intervalle $[b, e]$. Le résultat de l'application de la procédure du point fixe pour $\mathbb{T} \mid [b, e]$ est le même que son application pour une M-Trace $\mathbb{T}$ dans l'intervalle $[b, e]$, i.e., $M_{\tau,\mathbb{T} \mid [b,e]} \mid [b, e] = M_{\tau,\mathbb{T}} \mid [b, e]$.*

En termes simples, le théorème statue que pour évaluer une auto-transformation sur un intervalle de temps $[b, e]$, nous n'avons pas à examiner tous les observés qui se produisent en dehors de $[b, e]$.

4.12 Preuves des théorèmes proposés

4.12.1 Preuve du Théorème 1

Le théorème 1. stipule que pour une auto-transformation stratifiable τ et une M-Trace $\mathbb{T}$, l'interprétation M_τ est un modèle minimal pour τ dans $\mathbb{T}$. Ce modèle M_τ est aussi indépendant de la stratification de τ. Ce théorème énonce que la sémantique fournie par l'interprétation de point fixe définie dans la définition 36 est bien définie et non ambiguë. Il montre également que cette sémantique correspondent à notre intuition que tous les observés contenue dans la M-Trace transformée sont justifiés soit en étant dans la M-Trace initiale entrante ou en étant dérivés par une règle de transformation. Nous allons dans cette partie présenter la preuve de ce théorème.

M-Trace Minimale

Nous voulons prouver que l'interprétation de point fixe $M_{\tau,\mathbb{T}}$, pour une auto-transformation stratifiée est un modèle minimal, i.e., il n'existe pas de modèle M pour τ dans $\mathbb{T}$ avec $M \subseteq M_{\tau,\mathbb{T}}$. Une déclaration analogue pour des programmes logiques stratifiés est bien établie dans [ABW88]. Pour nos preuves dans le monde des M-Traces, nous adaptons la preuve de [Llo87] [42]. Le plus important challenge dans le transfert de cette preuve concerne la stratification des transformations qui sont définies différemment : les strates d'une transformation contiennent des règles, alors que les strates d'un programme logique se composent de symboles de prédicat.

Avant de commencer notre démonstration, nous répétons le théorème bien connu de point fixe de Knaster et Tarski. Soit L un treillis complet muni d'un ordre $\leq$ et $\Gamma : L \to L$ une fonction totale monotone. Γ a un plus petit point fixe (*least fixpoint*), $lfp(\Gamma)$, qui est la plus grande borne inférieure (*greatest lower bound*) de l'ensemble de tous les points fixes de Γ ainsi que l'ensemble des points préfixés : $lfp(\Gamma) = glb\{x \mid \Gamma(x) = x\} = glb\{x \mid \Gamma(x) \leq x\}$.

Soit $\tau = \tau_1 \uplus ... \uplus \tau_n$ une transformation stratifiée et $\mathbb{T}$ une M-Trace. Nous allons montrer par induction sur le nombre n de strates que $M_{\tau,\mathbb{T}} = M_n$ est un point fixe minimal pour Γ_τ avec $\mathbb{T} \subseteq M_{\tau,\mathbb{T}}$.

Pour l'induction de base $n = 0$, ceci est évident puisque $\tau = \emptyset$ et ainsi Γ est une simple transformation *identité*.

Pour l'induction de $n - 1 \to n$, nous utilisons le théorème de Tarski énoncé précédemment. Bien sur, nous ne pouvons pas appliquer le théorème directement à Γ_τ, puisque Γ_τ est en général non-monotone. Cependant, nous allons l'appliquer à une certaine restriction de Γ_τ. Soit Λ un treillis complet défini comme suit :

42. Comme nous allons le voir, la preuve de [Llo87] utilise l'existence d'un plus petit point fixe pour des applications monotones dans un treillis complet, un résultat bien connu établi par Knaster et Tarski. [ABW88] prouve le même théorème sans utiliser ce résultat. Nous fondons notre preuve sur [Llo87] qui nous semble plus intuitive et simple à comprendre que celui de [ABW88].

$$\Lambda = \{\mathcal{I}_{n-1} \cup S \mid S \subset \bigcup_{(G \leftarrow P) \in \mathcal{T}} gr(G \leftarrow P)\}$$

Ici, $gr(G \leftarrow P)$ désigne l'ensemble des observés, leurs relations et leurs valeurs d'attributs qui peuvent être interrogées ou construits par une règle $(G \leftarrow P)$. Parlant dans le langage de la programmation logique, il s'agit de la notion de *«instance fondée» (grounded instance)* d'un terme intervenant dans une règle [43]. Formellement, avec *ob* étant un élément de *obs*, *rel* ou *attr*, nous définissons *gr* comme suit :

$$\begin{aligned} gr(G \leftarrow P) = \quad & \{ob \mid [s,e] \in \mathcal{T} \wedge \exists \Sigma, ob \in \Sigma(G)\} \cup \\ & \{ob \mid [s,e] \in \mathcal{T} \wedge \exists\ q \text{ un élément du pattern } P \wedge \exists \psi \in \Sigma, (O,\Sigma) \models_{\mathbb{M}} \\ & q^{[s,e]}, ob \in O \text{ pour toutes les substitutions } \psi \text{ de } \Sigma\}. \end{aligned}$$

Lemme 1. *La restriction $\Gamma_{\tau}|_{\Lambda}$ de Γ_{τ} au treillis complet Λ est bien définie (i.e. l'application de Γ_{τ} à un élément de Λ reste toujours un élément dans Λ) et monotone (i.e. $O_i \subseteq O_j$ implique $\Gamma_{\tau}|_{\Lambda}(O_i) \subseteq \Gamma_{\tau}|_{\Lambda}(O_j)$.*

Pour ne pas nous éloigner de notre preuve principale, nous donnerons la preuve de ce lemme un peu plus tard.

Avec le théorème de Tarski décrit plus haut, ce lemme nous donne que $\Gamma_{\tau}|_{\Lambda}$ a un plus petit point fixe M :

$$\begin{aligned} M = \ lfp(\Gamma_{\tau}|_{\Lambda}) \quad &= glb\ \{O \in \Lambda \mid \Gamma_{\tau}(O) = O\} \\ &= glb\ \{O \in \Lambda \mid \Gamma_{\tau}(O) \subseteq O\} \end{aligned}$$

Veuillez noter que $M = M_{\tau,\mathbb{T}}$ tout simplement par définition de $M_{\tau,\mathbb{T}} = \Gamma_{\tau}^{w}(O_{n-1})$. De plus, M est un modèle pour τ pour $\mathbb{T}$ en raison du lemme suivant.

Lemme 2. *Une interprétation O est un modèle pour une auto-transformation τ («$O \models_{\mathbb{M}} \tau$») si et seulement si $\Gamma_{\tau}(O) \subseteq O$.*

Encore une fois, nous reportons la preuve de ce lemme.

Nous montrons maintenant que O est minimale. i.e. si certain $O' \subseteq O$ est un modèle de τ et que $\mathbb{T} \subseteq O'$ alors $O' = O$. Par l'hypothèse d'induction, nous avons $M_{n-1} \subseteq M'$ (puisque M_{n-1} est un modèle minimal pour $\tau_1 \uplus ... \uplus \tau_{n-1}$). Par définition de Λ, cela nous donne que $O' \in \Lambda$. Lemme 2 et O étant aussi le plus petit des pré-point fixe $\Gamma_{\tau}|_{\Lambda}$ rendent $O' \subseteq O$.

Preuve du Lemme 1

Pour montrer que $\Gamma|_{\Lambda}$ est bien définie, considérons $O \in \Lambda$. Par définition de Λ, $O = O_{k-1} \uplus S_O$ avec $S_O \subseteq \bigcup_{(G \leftarrow P) \in \mathcal{T}} gr(G \leftarrow P)$. Maintenant, nous avons $\Gamma_{\tau}(O) = O \cup \{ob \mid ob$ est généré par

43. Pour éviter toute confusion, nous parlons bien de l'instance fondée d'un terme intervenant dans une règle, pas de l'instance fondée de la règle elle-même

une règle de transformation $(G \leftarrow P) \in \tau\}$ avec le côté droit de l'union est un sous-ensemble de $\bigcup_{(G \leftarrow P) \in \tau} gr(G \leftarrow P)$. Cela nous donne donc $\Gamma_\tau(O) = O_{k-1} \uplus S_\Gamma$ avec $S_\Gamma \subseteq \bigcup_{(G \leftarrow P) \in \tau} gr(G \leftarrow P)$ (S_O est l'union de $\{ob|...\}$ et de S_O) et donc $\Gamma_\tau(O) \in \Lambda$.

Pour $\Gamma_\tau|_\Lambda$ monotone, soit $O_i \in \Lambda$, $O_j \in \Lambda$, $O_i \subseteq O_j$ et $ob \in \Gamma_\tau(O_i)$. Ce que nous voulons montrer, c'est que $ob \in \Gamma_\tau(O_j)$.

Si $ob \in O_i$, nous avons immédiatement $ob \in \Gamma_\tau(O_j)$ par $O_i \subseteq O_j$ et la définition de Γ_τ. Ceci correspond à un observé qui appartenait initialement à la M-Trace. Sinon dans le cas ou l'observés a été dérivé par une règle de transformation, nous avons une règle $(G \leftarrow P) \in \tau$ et un ensemble maximale de substitutions Σ avec $(O_i, \Sigma) \models_\mathbb{M} P$ et $ob \in \Sigma(G)$. Si $(G \leftarrow P) \in \overline{\tau_{n-1}}$, nous avons $ob \in O_{n-1}$ donc aussi $ob \in \Gamma_\tau(O_j)$ car $O_{n-1} \subseteq \Gamma_\tau(O_j)$ (Nous rappelons que $\Gamma_\tau|_\Lambda$ est bien définie). Il reste à examiner le cas où $(G \leftarrow P) \in \tau_n$, où nous devons montrer que $ob \in \Sigma(G)$.

Par induction sur P, nous montrons que $(O_j, \Sigma) \models_\mathbb{M} P^{[b,e]}$. En plus de l'hypothèse d'induction (H_1) et la définition de la théorie des modèles ($\models_\mathbb{M}$) du tableau 4.7, nous disposons de $(O_i, \Sigma) \models_\mathbb{M} P^{[b,e]}$ avec Σ étant maximale (H_2) et que $O_i \subseteq O_j$ (H_3).

- **Cas 1.** $(o : c)^{[b,e]}$. Par ($\models_\mathbb{M}$) et (H_2), nous avons $ob \in O_i$ et par (H_3) on obtient $ob \in O_j$.
- **Cas 2.** $(o\ r\ o')^{[b,e]}$. Par ($\models_\mathbb{M}$) et (H_2), nous avons $ob, ob' \in O_i$ et par (H_3) on obtient le $ob, ob' \in O_j$.
- **Cas 3.** $(P \texttt{ where } C)^{[b,e]}$. Application évidente de (H_1).
- **Cas 4.** $(P_1 \texttt{ and } P_2)^t$. ($\models_\mathbb{M}$) et (H_2) nous donne $(O_i, \Sigma) \models_\mathbb{M} P_1^{t_1}$ et $(O_i, \Sigma) \models_\mathbb{M} P_2^{t_2}$ avec $t = t_1 \sqcup t_2$. Par application de (H_1), nous obtenons $(O_j, \Sigma) \models_\mathbb{M} P_1$ et $(O_j, \Sigma) \models_\mathbb{M} P_2$ et on peut appliquer donc $\models_\mathbb{M}$.
- **Cas 5.** $(P_1 \texttt{ or } P_2)^t$. Application évidente de ($\models_\mathbb{M}$), voir le Cas 4.
- **Cas 6.** $(o \texttt{ extend } [o', d])$. Trivial, puisque ($\models_\mathbb{M}$) ne fait aucune référence à l'interprétation O_i, (respectivement O_j). Nous sautons les autres relations temporelles puisqu'elles sont analogues.
- **Cas 7.** ($\texttt{while } o : \texttt{ without } P$). (H_2) nous donne $ob = (i, c, b, e)$ avec $\psi(o) = i$ et $(O_i, \Sigma) \not\models_\mathbb{M} P^{t"}$ pour tout $t" \sqsubseteq [b, e]$. Nous devons montrer aussi que $(O_j, \Sigma) \not\models_\mathbb{M} P^{t"}$ pour tout $t" \sqsubseteq [b, e]$. Maintenant, s'il y avait un $t"$ tel que $(O_j, \Sigma) \models_\mathbb{M} P^{t"}$, alors nous aurons $(M_{n-1}, \Sigma) \models_\mathbb{M} P^{t"}$ en raison de la stratification de la négation. Cependant, cela impliquerait $(O_i, \Sigma) \models_\mathbb{M} P^{t"}$, ce qui est en contradiction avec nos hypothèses.

Preuve du Lemme 2

Nous voulons montrer que $M \models_\mathbb{M} \tau$ si et seulement si $\Gamma_\tau(M) \subseteq M$.

($\Rightarrow$) On suppose que $\Gamma_\tau(M) \subseteq M$ mais avec $M \not\models_\mathbb{M} \tau$, i.e., il y a une règle $r = G \leftarrow P \in \tau$ avec $M \not\models_\mathbb{M} r$. Par conséquent, nous devons avoir un $t \in \mathcal{T}^2$ et un Σ maximal avec $(M, \Sigma) \models_\mathbb{M} P^t$ mais $\Sigma(G)^t \not\subseteq M$. i.e., il doit y avoir un $ob \in \Sigma(G)$ tel que $ob \notin M$, qui est cependant en contradiction avec $ob \in \Gamma_\tau(M) \subseteq M$.

($\Leftarrow$) Soit $M \models_{\mathbb{M}} \tau$ et $ob \in \Gamma_{\tau}(M)$ et supposons que $ob \notin M$. Ensuite, ob doit avoir été généré par une règle $r = G \leftarrow P \in \tau$. Par conséquent nous devons avoir un ensemble maximal de substitutions Σ de telle sorte que $(M, \Sigma) \models_{\mathbb{M}} P^t$ et $ob \in \Sigma(G)$. Cependant, cela signifierait que $ob \in M$ puisque $M \models_{\mathbb{M}} r^t$, ce qui nous donne une contradiction.

Indépendance de la stratification

Pour prouver que $M_{\tau,\mathbb{T}}$ est indépendant de la stratification donnée d'une transformation τ, nous montrons que deux stratifications possibles de τ sont équivalentes, i.e., la procédure de point fixe donne le même modèle. Dans le monde des programmes logiques stratifiés, c'est à nouveau un résultat bien établi. En fait, le racine de cette déclaration pour Datalog trouvée dans [AHV95, théorème 15.2.10] est transféré directement aux transformations de M-Traces. Seuls deux points doivent être adaptés pour faire face à la notion de stratification de transformation, qui est définie sur les règles et non des symboles de prédicats : la notion de graphe de précédence (parfois aussi appelé graphe de dépendance) et un lemme qui nous permet de faire valoir que si deux strates sont indépendantes les unes des autres alors elles peuvent être permutées dans la procédure de point fixe. Nous ne répétons pas la preuve de [AHV95] ici, on donnera plutôt les deux adaptations que nous venons de mentionner.

Le graphe de précédence G_P d'une transformation τ est un graphe orienté avec des arcs marqués, soit «+» (appelés arcs positifs) ou «-» (appelés arcs négatifs). Les sommets du graphe sont les règles de τ. Il y a un arc positif de r à r' si r dépend positivement de r'. Il y a un arc négatif de r à r' si r dépend de r' de toute autre manière (négativement ou temporellement) [44].

Nous devons montrer le lemme suivant pour remplacer le lemme 15.2.9 de [AHV95]. Il est à noter que c'est le seul point où la preuve de [AHV95] est spécifique à Datalog$^{\neg}$. Si une transformation τ est un programme semi-positif, i.e., qu'elle ne contient que des dépendances positives, et $\tau = \tau_1 \uplus \tau_2$ est une stratification de τ, alors la procédure de point fixe donne le même modèle pour τ et pour $\tau_1 \uplus \tau_2$: $\Gamma^w_{\tau}(\mathbb{T}) = \Gamma^w_{\tau_2}(\Gamma^w_{\tau_1}(\mathbb{T}))$ pour tout le flux des observés $\mathbb{T}$.

Preuve. Vous remarquez que $\Gamma_{\tau} = \Gamma_{\tau_2}$ est monotone (comme dans le lemme 1 ci-dessus). Avec l'inclusion $\mathbb{T} \subseteq \Gamma^w_{\tau_1}(\mathbb{T}) = \Gamma^w_{\tau_1}(\mathbb{T})$ ce qui rend $\Gamma^w_{\tau}(\mathbb{T}) \subseteq \Gamma^w_{\tau_2}(\Gamma^w_{\tau_1}(\mathbb{T}))$. D'autre part, l'inclusion $\Gamma^w_{\tau_1}(\mathbb{T}) = \Gamma^w_{\tau_1}(\mathbb{T}) \subseteq \Gamma^w_{\tau}(\mathbb{T})$ donne $\Gamma^w_{\tau_2}(\Gamma^w_{\tau_1}(\mathbb{T})) \subseteq \Gamma^w_{\tau_2}(\Gamma^w_{\tau}(\mathbb{T})) = \Gamma^w_{\tau}(\Gamma^w_{\tau}(\mathbb{T})) = (\Gamma^w_{\tau}(\mathbb{T}))$

4.12.2 Preuve du Théorème 2

Rappelons le théorème 2. Soit $\mathbb{T} \mid [b, e]$ une restriction d'une M-Trace à l'intervalle $[b, e]$, i.e. $\mathbb{T} \mid [b, e]$ est la M-Trace $\mathbb{T}$ tel que $\forall (i, c, s, e) \in \mathbf{obs}_{\mathbb{T}}, [s, e] \sqsubseteq [b, e]$. De la même manière, soit

44. Certes, ici nous sommes un peu abusif dans la notation, utilisant «-» pour l'étiquette non seulement des dépendances négatives, mais aussi des dépendances temporelles. Toutefois, il n'est pas nécessaire de faire la distinction entre dépendance négative et temporelle ici, puisque elles sont aussi traitées de la même manière dans la définition d'une stratification (ce qui nécessite que la règle dépendante soit dans une strate strictement supérieure).

$M \mid [b,e]$ la restriction d'une interprétation M à l'intervalle $[b,e]$. Le résultat de l'application de la procédure du point fixe pour $\mathbb{T} \mid [b,e]$ est le même que son application pour une M-Trace $\mathbb{T}$ dans l'intervalle $[b,e]$, i.e., $M_{\tau,\mathbb{T} \mid [b,e]} \mid [b,e] = M_{\tau} \mid [b,e]$.

Nous devons montrer que $M_{\tau,\mathbb{T}|u}|u = M_{\tau,\mathbb{T}}|u$ pour un intervalle de temps arbitraire u. Pour cela, nous devons d'abord faire la remarque suivante.

Lemme 3. *Soit t et u des intervalles de temps avec $t \sqsubseteq u$ et soit P un pattern. Nous avons alors :*

$$(O|u, \Sigma) \models_{\mathbb{M}} P^t \text{ ssi } (O, \Sigma) \models_{\mathbb{M}} P^t$$

La preuve de ce lemme est triviale et se fait par une induction sur P. Avec la définition de l'opérateur de conséquence immédiate Γ_{τ}, l'observation précédente nous donne

$$\Gamma_{\tau}(O\ |u)\ |u = \Gamma_{\tau}(O)\ |u$$

pour tous des intervalles de temps u et tous les transforamtions τ.

En plus, la définition de Γ_{τ} énonce que tous les observés *ob* construits par une règle de r «héritent» de leur temps d'occurrence $[b,e]$ du pattern de cette règle. Ainsi, on estime que $(\Gamma_{\tau}\ |u)^w = (\Gamma_{\tau}^w)\ |u$ et nous obtenons $M_{\tau,\mathbb{T}|u}|u = M_{\tau,\mathbb{T}}|u$.

4.13 Conclusion

Dans ce chapitre, nous avons présenté la formalisation de la notion de M-Trace et les langages permettant son interrogation et transformation. Certaines caractéristiques peuvent être dégagées de cette formalisation et serviront de cahier des charges pour comparer, réifier ou adapter les langages et modèles existants pour modéliser et traiter les M-Traces. Nous résumons dans la suite les plus importants points à prendre en compte pour l'exploitation des M-Traces.

En termes de représentation de M-Traces

Notre définition de modèle de trace et M-Trace met en œuvre un certain nombre de propriétés qui n'ont pas été considérées ensemble dans un même cadre de représentation.

- Le modèle de trace décrit et type les observés qui composent la trace. Les types d'observés sont ordonnés partiellement permettant de représenter des hiérarchies de types.
- Le modèle type aussi des relations et permet de construire des hiérarchies de type de relations. Ces relations permettre de modéliser les liens non temporelles pouvant exister entre les observés.
- Le modèle de trace structure les observés de la trace en permettant l'association d'un ou plusieurs attributs.
- Le modèle de trace spécifie le domaine temporel dans lequel les observés sont situés temporellement.

– Une M-Trace contient une séquence potentiellement infinie d'observés tracée par une application.

Certaines de ces caractéristiques peuvent être mises en œuvre avec des représentations ontologiques. Cependant, la considération des M-Traces comme un flux ouvert est nouveau dans ces approches. Le chapitre suivant fait le tour d'horizon des modèles permettant de représenter les M-Traces. À quel point certaines représentations permettent-elles de modéliser les M-Traces est une question qui sera discuter dans le chapitre suivant.

En termes d'interrogation et de transformation de M-Traces

Comme nous l'avons montré tout au long du chapitre, les requêtes et les transformations sur des M-Traces sont censées fonctionner en permanence et de manière continue et elles peuvent produire de nouveaux résultats à chaque nouvelle arrivée d'observés. Elle nécessite des langages prenant en compte les aspects relatifs au modèle de trace, i.e. des hiérarchies de type et de relations, types d'observé structurés avec des attributs. Les langages doivent également décrire des requêtes permettant d'extraire un ensemble d'observés afin de construire de nouveaux observés à l'aide de règles de transformations. L'interrogation doit être intuitive permettant l'usage de relations temporelles absolues (e.g. `before`, `after`, `during`, etc.) et relatives (`extend`, `shorten`, etc.). Les transformations doivent permettre de construire de manière intelligible des M-Traces transformées en spécifiant les bornes temporelles des nouveaux observés, l'affectation d'expression aux attributs, mise en relation entre observés mais également le calcul automatique des bornes temporelles (dans le cas ou elles n'ont pas été spécifiées). L'évaluation des requêtes et des transformations doit être possible de manière ponctuelle et continue et doit être monotone même en présence d'opérateurs non monotones comme la négation.

Le chapitre suivant présente les grandes classes des langages permettant la modélisation, l'interrogation et la transformation des M-Traces. Certains langages sont limités et nécessitent parfois des adaptations pour prendre en compte les spécificités des M-Traces. Au regard du cadre formel et des propriétés des M-Traces considérées dans cette thèse, nous discuterons les avantages et inconvénients de chacun d'eux.

Chapitre 5

Interrogation et Transformation de Traces Modélisées : représentations, modèles et langages existants

Sommaire

Dans ce chapitre, nous fournissons une description générale des travaux se rapportant d'une manière directe ou indirecte aux langages pouvant être exploités pour l'interrogation et la transformation de traces modélisées. Nous ferons le tour des formalismes pouvant être utilisés pour représenter, interroger et transformer des traces modélisées. Ces formalismes seront discutés par rapport à la formalisation donnée dans le chapitre 4 précédent.

5.1 Introduction

L'usage des traces a récemment engendré beaucoup de travaux de recherche, notamment dans le contexte des environnements informatiques pour l'apprentissage humain (EIAH). Cependant, bien que leur exploitation soit intensive dans différents domaines, les pratiques et usages des traces sont loin d'appartenir à une discipline à part entière. En effet, les traces sont considérées le plus souvent comme des objets *support* pour différentes techniques, ce qui rend leurs études et usages orientés plus vers des domaines applicatifs. Nous pouvons citer à titre d'exemples les domaines suivants :

- l'analyse de fichiers logs issues de sites web (weblog file analysis and click streams, [Hoo97, FWZ01, CFP+04]), issues d'exécution de programmes [And98]. Ce domaine est aussi lié à la fouille de données d'usage de sites web (Web Usage Mining) ([Liu06, CPY98]) ;
- la surveillance et la détection d'intrusion dans les réseaux informatiques, généralement dans le cadre de la gestion des réseaux [SR90, MsSS97]) ;
- l'acquisition de modèle utilisateur (user modelling) [PKK95, KKP01, SGT97], semantic logging [BB07] ;
- l'analyse de données séquentielles [FS96].

L'évolution de chacun de ces domaines se fait souvent en parallèle et avec peu d'échanges entre les communautés. En conséquence, il y a une terminologie très diverse, un large éventail d'approches et des attentes et des exigences différentes en termes de fonctionnalités utilisant les traces. D'ailleurs, nous pourrons rajouter bien d'autres travaux utilisant dans une certaine proportion des données relevant des traces d'interactions. En pratique, les usages les plus courants utilisant les traces exploitent le plus souvent des calculs statistiques et des algorithmes d'apprentissage automatique pour la découverte et l'extraction de connaissances pertinentes pour une fonctionnalité donnée (par exemple Web Usage Mining [Liu06, CPY98, MPT99, MDL+00, LY08] pour la personnalisation).

Même si le concept de trace semble jouer un rôle important dans ces nombreux travaux et domaines d'applications, il existe que peu d'initiatives qui en proposent une théorisation explicite comme un objet à part entière. Nous pourrons citer [Die95] qui théorise cette notion dans le cadre d'un travail définissant une algèbre de processus permettant, entre autres, d'étudier et de détecter les sections du processus qui peuvent être parallélisables, les inter-blocages, etc. Toujours pour les traces de processus, il y a aussi les travaux sur *log file analysis* qui théorisent ce qu'est un fichier log et la manière de valider l'exécution de programmes par rapport à une spécification formelle donnée [And98]. À notre connaissance, il n'existe aucune formalisation explicite des langages de manipulation de traces (qui sont naturellement différentes des traces d'exécution de programmes), puisqu'il n'existe pas une conceptualisation générique de ce qu'est une trace.

En pratique, la plupart des exploitations reposent sur des langages, des modèles et des outils qui ont des racines nombreuses et indépendantes dans divers domaines. Bien qu'il soit possible de mettre en œuvre des M-Traces, des requêtes et des transformations avec ces modèles et

langages, ces tentatives échoueront dans leur majorité à répondre aux exigences requises par notre sémantique déclarative. Nous citons quelques systèmes et langages pour la mise en œuvre des M-Traces, des requêtes et des transformations.

- Les Systèmes de gestion de Bases de Données Relationnelles (SGBDR) [AHV95, LL99] avec comme langage pour l'interrogation et transformation SQL. Certaines extensions ont traité certaines aspects requis pour l'interrogation de M-Traces. Nous pouvons citer les langages pour les bases de données temporelles [TCG+93], les bases de données séquentielles [SLR95] et les bases de données de capteurs [GS01].
- Le traitement d'événements complexes (Complex Event Processing). En particulier, la surveillance de flux d'événements « simples » pour découvrir des événements complexes, i.e., des événements ou des situations qui ne peuvent pas être détectées en considérant uniquement un seul événement [45]. Ce domaine, en pleine effervescence, fonde ses bases sur la lignée des systèmes de gestion de base de données actives notamment pour la détection d'événements composites (début des années 1990 [GJS92a, GJS93, CKAK94]) ;
- Les systèmes de Gestion de Flux de Données (SGFDs). Quelques exemples d'applications parmi les plus populaires concernent le suivi d'actions sur des pages web dynamiques [ZS02, CDTW00], ou encore la gestion des données issues de capteurs [ABB+04] ;
- Les systèmes à base de règles ne disposent pas d'un langage de requête pour les traces en tant que tel. Cependant, ils offrent un moyen assez facile et très flexible pour la mise en œuvre des requêtes et surtout des transformations. La principale raison pour laquelle les systèmes à base de règles sont plus pratiques pour la mise en œuvre des requêtes qu'un langage de programmation général est que leur algorithme d'évaluation (en chaînage avant) est très similaire aux sémantiques utilisées pour évaluer les transformations continues (voir le chapitre précédent). Ces systèmes se basent aussi sur des fondements rigoureux pour la représentation et le raisonnement sur le temps (à peu près depuis les années 1980 et les travaux de [All83, KS86], mais avec des racines beaucoup plus anciennes [MH69]) ;
- Les entrepôts XML permettent de disposer de langages capables d'exprimer des transformations et des requêtes avec XSLT et XQuery. Ces technologies étant également facile à utiliser peuvent être un bon candidat pour l'exploitation des M-Traces. Cependant, XML montre rapidement ces limites notamment pour représenter des M-Traces formalisées comme des graphes équipés de nœuds temporels. Ceci complique énormément les requêtes et les transformations pouvant être appliquées sur des arbres représentant des graphes temporels.
- Les dépôts ontologiques RDF/OWL [46] sont particulièrement adaptés pour la mise œuvre et la représentation de modèles de traces (qui peuvent être considérées comme des ontologies temporelles). Dotés du langage SPARQL [EA08, Cyg05], les dépôts RDF offrent une solution idéale pour les requêtes et les transformations de M-Traces. Cependant, ces technologies montrent leurs limites lorsque nous considérons une évaluation continue des

45. http ://complexevents.com/
46. http ://www.w3.org/RDF/

requêtes et des transformations.

Nous pourrons représenter dans chacune de ces technologies les M-Traces et utiliser les langages existants pour les interroger et les transformer. Cependant, comme nous allons le montrer, chacune des technologies nécessitera des adaptations parfois à différents niveaux d'abstraction (théorique, technique d'opérationnalisation, etc.).

La suite de ce chapitre passe en revue l'état de l'art des techniques pouvant opérationnaliser l'exploitation des traces modélisées. Toutes les technologies considérées seront discutées par rapport à notre formalisation pour dégager les apports et les limites de chacune d'elles. Nous sommes conscient que les domaines et les technologies abordées sont très vastes et sans limites claires. Par conséquent, cette section se focalisera fortement sur le sujet de cette thèse, qui est, la représentation, l'interrogation et la transformation de M-Traces. La suite se concentrera donc sur les langages et les formalismes pour représenter les modèles, les requêtes et les transformations qui sont connues et spécifiées *a priori*. En effet, d'autres aspects de l'exploitation des M-Traces, par exemple la détection de patterns complexes inconnus en utilisant des approches comme l'apprentissage automatique ou la fouille de données de traces, ne sont pas abordées dans ce chapitre, mais dans les perspectives futures des langages pour les M-Traces.

5.2 Modélisation, interrogation et transformations de M-Traces : réification dans les technologies existantes

5.2.1 M-Traces et Systèmes de Gestion de Base de Données (SGBDs)

Les Systèmes de Gestion de Base de Données (SGBDs) fournissent les moyens de spécifier des structures de données particulières, de contraindre l'ensemble des données associées à ces structures et enfin de manipuler ces données [AHV95]. La spécification de la structure et des contraintes se fait en utilisant un langage de définition de données (LDD), et la spécification de la manipulation se fait en utilisant un langage de manipulation de données (LMD). Les LMD fournissent deux services fondamentaux : l'interrogation pour l'extraction des données de la base de données actuelle et la mise à jour pour permettre la modification de l'état de la base de données.

Réification des M-Traces

Les SGBDs relationnels opérationnalisant le modèle relationnel sont très courants pour le stockage et l'interrogation des traces [Mos04] et peuvent être utilisés pour représenter les M-Traces. Dans ce modèle, les relations sont de simples structures qu'on peut manipuler et décrire avec le langage déclaratif SQL [AHV95]. La réification des traces, de leurs interrogations et transformations dans les SGBDs relationnels peut être faite de manière directe : les types d'observé, attributs et les relations types peuvent être représentés par des relations. Les domaines temporels peuvent être représentés par des attributs de type `datetime` ou `int` pour le domaine séquentiel. L'interrogation peut être faite avec une requête de type `select id as x from action` pour un

pattern (`x : action`). Quant aux transformations, les vues matérialisées permettront de créer de nouveaux observés à partir des relations existantes (`create materialized view`). L'avantage de cette réification est que l'utilisation du langage pour transformer ou requêter peut être fait de manière presque directe en utilisant directement les noms de relations. Cependant, elle devient difficilement faisable si on considère plusieurs M-Traces ou plusieurs transformations.

Une autre approche peut consister à réifier les concepts de la M-Trace à un niveau «*méta-modèle*», i.e., créer un schéma relationnel représentant les types d'observés, les relations type, les attributs, les observés, etc. Une partie du schéma concernant les types d'observés peut être définie par la relation suivante : `obselType(name, idTracemodel)` avec $<$ `action`$, 10 > \in$ `obselType` représente le type d'observé `action` définie dans le modèle de trace identifié par l'id 10}.

Avec cette approche, nous pouvons aussi exprimer des relations pour représenter les modèles de traces et les traces stockées dans la base. Cependant, l'expression des requêtes mais surtout des transformations devient complexe. On pourrait imaginer une approche hybride combinant l'approche directe pour les traces et la méta-modélisation pour les modèles. Mais ceci permettra de simplifier les requêtes sans être plus efficace pour les transformations. En outre, l'usage d'attributs représentant le temps d'un type prédéfini (e.g., `datetime`) peut aussi être problématique notamment lorsque les M-Traces interrogées ou transformées sont dans des domaines différents avec des origines différentes.

Enfin, dans toutes ces approches, la réification des hiérarchies des types d'observé et des types de relations ne peut être directement exprimée dans un schéma de base de données relationnel. Ceci nécessite des techniques pour aplatir les hiérarchies dans le modèle de traces, ce qui augmente grandement la complexité (de description) des requêtes et des transformations.

Réification des requêtes et des transformations de M-Traces

Même si on considère d'autres modèles [47] ou d'autres extensions du modèle relationnel, les SGBDs sont très limités dans le cas d'évaluations continues des requêtes [BBD+02]. Pour les aspects concernant la représentation des M-Traces, certaines extensions sont moins restrictives que d'autres. Par exemple, les SGBDs temporels [TCG+93] pourront permettre de prendre en compte nativement des représentations de temps, les SGBDs orientés objets [Kim90] pourront permettre de réifier facilement les hiérarchies dans les modèles de traces. Cependant, en ce qui concerne la manipulation des M-Traces (i.e. leur interrogation et transformation) les SGBDs en général sont peu adaptés. Comme nous l'avons mentionné dans les chapitres précédents, une des contraintes exigées et requises par l'exploitation des traces modélisées est l'évaluation continue. En fait, les limites des SGBDs relationnels et des autres modèles sont bien connues notamment dans le contexte des travaux autour des Systèmes de Gestion de Flux de Données (SGFDs) [48]. La plupart du temps, les SGBDs classiques ne sont pas conçus pour gérer des données en grande quantité qui varient très fréquemment. De plus, les requêtes destinées aux

47. Les structures les plus importantes qui ont été utilisés pour les bases de données à ce jour sont des graphes dans les modèles réseau, sémantique et orientés objets ; les arbres dans le modèle hiérarchique.

48. Data Stream Management Systems (DSMS).

SGBDs sont instantanées et portent sur l'état actuel des données, tandis que les requêtes sur les flux de données sont de nature continue et s'évaluent au fur et à mesure sur les nouvelles données arrivant dans le système.

Discussion

Dans les SGBDs traditionnels, l'évaluation des requêtes utilise des connaissances préalables sur la structure physique et la distribution des données. L'accès aux données se fait de manière directe et les réponses aux requêtes sont précises. Cependant, l'accès aux observés de la M-Trace est séquentiel, le taux d'arrivée des observés en général n'est pas maîtrisé et les données peuvent être redondantes et bruitées. Les techniques classiques d'évaluation ne sont donc pas applicables pour les requêtes sur des M-Traces. Au lieu d'avoir un plan d'exécution statique, l'évaluation doit aussi être continue et dynamique pour s'adapter, en plus des données variables des M-Traces, aux opérateurs utilisés. Par exemple, l'usage des opérateurs comme la différence peut être sacrifié ou limité afin de permettre une évaluation incrémentale.

Le travail de D. Terry [TGNO92] semble être le premier a avoir abordé ces techniques d'évaluation. [TGNO92] a définit une sémantique pour les requêtes continues sur des bases de données permettant seulement des ajouts. La sémantique des requêtes monotones a été spécifiée en terme d'évaluation incrémentale. Plus précisément, [TGNO92] a décrit une classe de requêtes continues qui peuvent être réécrites et converties en requêtes incrémentales, i.e., pouvant être évaluées périodiquement. En raison du problème des opérateurs bloquants (comme la différence), le langage utilisé a été limité à un sous-ensemble de SQL. Plus tard, Law et al. ont identifiés dans [LWZ04] que la classe des requêtes utilisant des opérateurs non bloquant correspond à la classe des requêtes monotones.

Pour les M-Traces, les requêtes et les transformations réifiées dans les SGBDs relationnels impliquant la négation ne sont pas monotones puisqu'elles peuvent produire des résultats qui cessent d'être valables. Par exemple, une requête qui extrait tous les messages qui n'ont pas encore reçu de réponse (voir la discussion du chapitre 3 section 3.9). En fait, il est bien connu que la négation est non-monotone [LWZ04]. Les SGFDs ont pu en partie traiter le cas des opérateurs non monotones. La suite présente les SGFDs et la réification des M-Traces dans cette technologie.

5.2.2 M-Traces et Systèmes de Gestion de Flux de Données (SGFDs)

Les SGFDs répondent aux besoins qu'ont des applications d'interroger des données générées continuellement en flux [GO03]. Dans ce contexte, les SGBDs ne sont pas bien adaptés bien que ce domaine ait déjà traité la question des base de données larges et volumineuses, par exemple dans les bases de données traditionnelles ou dans les entrepôts de données à l'aide de techniques de fouille de données. Cependant, le temps de traitement devient de plus en plus critique pour certaines applications qui doivent interroger et analyser les données plus vite, afin de prendre des décisions et réagir le plus rapidement possible. Quelques exemples

d'applications parmi les plus populaires sont, le contrôle du trafic réseau [CGJ+02, CJSS03], l'analyse de journaux transactionnels (transactions web, bancaires ou de télécommunication) [CFP+04], le suivi d'actions sur des pages web dynamiques [ZS02, CDTW00], ou encore la gestion des données de capteurs [ABB+04]. Ces applications ont motivé les propositions de plusieurs SGFDs dans la communauté académique, parmi lesquels, STREAM [ABB+04], Aurora [ACc+03] et TelegraphCQ [CCD+03].

Réification des M-Traces

À l'image d'une M-Trace, un flux de données est conceptuellement un ensemble infini de données qui sont générées continuellement. Les données sont en général représentées par des n-uplets et elles sont ordonnées par un attribut d'estampillage de temps. Les techniques d'évaluation des requêtes doivent prendre en compte ces faits. Par exemple, l'infinité des flux introduit des problèmes pour l'exécution de certains opérateurs ayant besoin en entrée d'un ensemble fini de données. Une des solutions est de créer, à partir d'un flux infini, un ensemble fini de données afin de permettre l'exécution de ces opérateurs. Ces ensembles finis sont appelés des «*fenêtres*».

Le modèle de flux de données est inspiré par le modèle séquentiel [SLR95] qui est en fait aussi une extension du modèle relationnel. Les n-uplets sont ordonnés par la valeur d'un attribut (en général par l'estampille du temps). Ils sont conformes à un schéma dont le format dépend de l'application.

De ce fait, la réification des M-Traces dans les SGFDs peut se faire de deux manières à l'image de leur réification dans les SGBDs. Une réification directe permet de représenter les types d'observé par des relations, etc. Par exemple, si on prend une application de surveillance environnementale avec des capteurs, la M-Trace peut être représentée par le flux de données provenant des capteurs. Par exemple, un modèle de trace contenant un type d'observé `capteur` représentant des mesures faites par les capteurs ainsi que d'autres informations comme l'identificateur ou la localisation du capteur, peut être représenté de la manière suivante : `capteur(id, location, temperature, timestamp)` avec $< i, l, t, t_s > \in$ `capteur` est un observé de type `capteur` collecté au moment t_s dont l'identifiant est i, la localisation est l et mesurant la température t. Les SGFDs permettent d'exprimer des requêtes continues sur un tel flux [49], par exemple, pour surveiller l'environnement en temps réel, pour détecter des anomalies, etc.

Bien que les schémas de flux soient liés aux applications, il est aussi possible de mettre en place un schéma générique pour représenter les M-Traces décrivant différents types de source de flux. Par exemple, une relation `obsel(id, type, start, end)` pourra représenter le flux des observés. Cependant, comme pour les SGBDs, une réification des M-Traces complique grandement les requêtes et les transformations. Bien que la représentation relationnelle soit adoptée par la plupart des propositions de SGFDs, les techniques d'évaluation des requêtes et de repré-

49. Il faut noter que, contrairement aux réseaux de capteurs, l'évaluation des requêtes se fait de manière centrale, sur les flux de données qui sont envoyés au SGFD directement par des capteurs ou par leurs stations de base.

sentation des fenêtres temporelles sont multiples. Ces techniques sont présentées dans le point suivant.

Réification des requêtes et des transformations de M-Traces

Un des avantages de l'usage de SGFD pour les M-Traces est l'évaluation continue des requêtes. En effet, les requêtes sur les flux de données doivent s'évaluer continuellement afin de prendre en compte les nouvelles valeurs s'ajoutant au flux. Donc, comme dans le cas des M-Traces, les requêtes dans les SGFDs sont continues. Cependant, il faut noter une petite nuance sur la périodicité d'exécution des requêtes. En effet, dans les SGFDs, cette périodicité peut être définie par le taux d'envoi de données des sources (e.g. capteurs), tandis que les M-Traces ne maîtrisent pas ce taux d'envoi. En conséquence, la périodicité n'est pas à définir dans les requêtes, sauf pour celles qui s'évaluent sur les fenêtres périodiquement glissantes (voir dans la suite). Les réponses aux requêtes sont créées aussi en flux. Les requêtes sont formées d'un ensemble d'opérateurs, et chaque opérateur « consomme » des données en entrée, exécute des opérations, et « produit » des résultats de manière continue. Un flux de résultat peut donc être une entrée pour un autre opérateur.

Pour la formulation des requêtes, l'approche la plus adoptée est d'utiliser le modèle relationnel et de construire des requêtes avec un langage similaire à SQL. C'est le cas pour les SGFDs comme STREAM [ABB+04] et TelegraphCQ [CCD+03]. Cependant, même si, au niveau syntaxique, les requêtes sont similaires à celles destinées aux bases de données traditionnelles, leur évaluation est bien différente. L'évaluation se fait n-uplet par n-uplet, contrairement à l'approche des bases de données traditionnelles où les requêtes sont évaluées sur une relation. Cependant ces langages ne sont pas tous formalisés. L'effort le plus important vient de STREAM qui propose un formalisme pour son langage CQL (Continuous Query Language) [ABW03]. Alternativement à l'approche relationnelle, dans Aurora [ACc+03] la construction des requêtes est procédurale, i.e. une requête est créée à partir d'une interface graphique où l'on place des boîtes qui représentent des opérateurs et des flèches qui représentent les flux de données.

Concernant les requêtes simples sur les M-Traces, traiter des flux potentiellement infinis ne pose pas réellement de problème quand la requête peut se réduire à un filtre sur les données. Par contre, si une requête contient des opérateurs dits « bloquants » [BW01] il faut réduire le flux infini à un ensemble fini. C'est le cas pour un opérateur s'appliquant sur une relation finie comme certains opérateurs d'agrégation (e.g. somme, moyen, max, min, histogramme, médian, etc.) ou la négation.

Pour effectuer de telles opérations les SGFDs utilisent deux types de fenêtres :

- fenêtres temporelles, dont la largeur est déterminée par une unité de temps. Par exemple, une fenêtre temporelle peut être utilisée pour répondre à la requête suivante : quelle est la moyenne des mesures de température faites depuis 10 minutes ?
- fenêtres basées sur le nombre d'éléments (fenêtres BNE), dont la taille est déterminée par le nombre d'éléments dans la fenêtre.

Les fenêtres sont en général glissantes, i.e les bornes des fenêtres se déplacent pour avoir dans la fenêtre les données les plus récentes. Les fenêtres peuvent aussi avoir des comportements différents pour le glissement de leurs bornes (e.g. fenêtres périodiques, tumbling [50], landmark [51], etc.). Les manières de définir ces comportements varient selon les propositions. STREAM utilise deux expressions pour définir deux types de requêtes : `[ROWS n]`, pour spécifier une fenêtre BNE de n éléments ; et `[RANGE d seconds]`, pour spécifier une fenêtre temporelle de d secondes. Par exemple, soit une requête R, formulée selon le schéma `capteur` donné en 5.2.2, qui demande les mesures > 45 faites depuis 30 secondes. Avec STREAM on peut représenter R comme suit : `select temperature from capteur [RANGE 30 seconds] where temperature > 45`. Le résultat de cette requête est aussi un flux. Il contient les température supérieure à 45 et qui datent au maximum de 30 secondes à partir du moment de l'exécution de la requête.

TelegraphCQ utilise des boucles `for` pour définir des fenêtres temporelles. Ainsi, il vise à définir des fenêtres plus génériques (fenêtres landmark, fixes, glissantes, etc.) en spécifiant la politique d'avancement des bornes de la fenêtre, ainsi que leur quantité d'avancement à chaque itération de la boucle [CCD+03]. Cependant, aucune des propositions pour définir des fenêtres n'est suffisamment générique pour exprimer «tous» les types. Par exemple, CQL est limité aux fenêtres glissantes, et TelegraphCQ ne définit que des fenêtres temporelles. De plus, aucune des deux ne permet de définir la périodicité de glissement des fenêtres. Cette période est implicite, e.g., chaque fois qu'il y a une nouvelle entrée dans la fenêtre.

Discussion

Il n'existe pas d'opérateur de transformation dans les SGFDs, néanmoins, elles peuvent être réifier dans SGFDs mais de manière moins directe que les requêtes. Pour les requêtes, la sémantique de CQL repose sur trois opérations de conversion : du flux vers la relation (via les fenêtres), relation vers relation (via SQL), et les opérateurs de conversion de relation vers le flux de données. Comprendre une requête implique toujours un aller-retour complet des flux aux relations, puis à la relation résultat, puis de retour vers le flux de données. Afin de connaître le résultat à un instant donné t, il ne suffit pas d'appliquer ce processus que pour l'instant t. En effet, il est nécessaire pour les opérateurs de livrer la différence entre l'état actuel de la relation résultante et son état antérieur. Bien comprendre une requête pourrait donc être une tâche fastidieuse qui consiste à rejouer mentalement tout le flux pour chaque instant du temps, surtout pour les transformations qui utilisent des vues sur les résultats. Pour les transformations, il faudra créer les relations résultats représentant la M-Trace, décrire les requêtes permettant de les instancier à partir d'autres flux. Ceci nécessite plusieurs conversions et peut être très complexe à faire dans le contexte d'une réification indirecte des M-Traces. En plus, si on considère la méthode choisie pour aplatir les hiérarchies dans les modèles de traces, l'expressions d'une transformation de M-Trace devient rapidement problématique notamment en présence de la

50. Les fenêtres se sautent et elles ne s'intersectent pas.
51. Seulement une des bornes de la fenêtre se déplace, l'autre borne reste fixe.

négation.

Bien que les SGFDs fournissent des moyens génériques d'interrogation de M-Traces, il existe des différences fondamentales au niveau de la manipulation du temps, des modèles de trace et des transformation. CQL et les autres langages pour interroger les flux de données n'offrent que peu de soutien pour les requêtes qui impliquent des relations temporelles entre des observés ou des tuples et ne supporte pas les observés survenus dans des intervalles de temps. Dans CQL, le temps est surtout utilisé pour les fenêtres basées sur le temps. À un niveau moins important, les requêtes peuvent utiliser le temps sous forme d'un attribut de donnée régulier faisant partie des tuples.

En résumé, CQL a une sémantique très précise pour les données, mais cette sémantique peut être considérée comme peu intuitive et moins précise pour les M-Traces. Tout d'abord, la sémantique implique toujours un aller-retour des flux aux relations, puis à la relation résultat, puis de retour vers le flux. Deuxièmement, les SGFDs offrent un support minimal pour le temps notamment pour la détection des observés sur critères temporels. Ils posent aussi bon nombre de difficultés pour la transformation de M-Traces.

Par contraste, les langages pour la détection d'événements complexes semblent très intuitifs dans leur sémantique et offrent un excellent support pour les relations temporelles. Nous étudions par la suite la réification des M-Traces dans ce contexte.

5.2.3 M-Traces et le Traitement des Événements Complexes

Le Traitement des événements complexes ou Complex Events processing (CEP) désigne une classe de systèmes de traitement d'événements. L'objectif d'un CEP est d'identifier des événements les plus significatifs à partir d'un nuage d'événements notifiés et reçus. Pour ce faire, CEP emploie différentes techniques de détection de patterns complexes à partir de multiples événements concernant la corrélation, l'abstraction, les hiérarchies entre événements, les relations entre événements (liens de causalité, l'adhésion, la chronologie, etc.). CEP vise dans sa fonction principale à découvrir, à partir d'événements simples émis dans toutes les couches d'une organisation, des « événements complexes » permettant d'analyser une situation complexe et de décider d'un plan d'action en temps réel.

Différents langages très hétérogènes ont été développés pour la reconnaissance des événements complexes. Cette hétérogénéité peut être attribuée en partie au fait que le CEP a évolué à partir de nombreuses et indépendantes racines (Base de données active, SGFD, algèbre d'événements) et n'a été que récemment reconnu comme un domaine à part entière.

Réification des M-Traces

Comme pour la notion de trace, en CEP et les domaines connexes, il y a beaucoup de discussions autour de la question de ce qui est et ce qui n'est pas un *événement*. Il n'y a pas de définition singulière généralement admise pour cette notion. Toutefois, certaines définitions,

comme celle de [LS08] [52] indiquent clairement une nécessité de représenter des événements (message, objet, appel de fonction, etc.) comme des entités ayant des données associées. Ceci a amené CEP à considérer plusieurs représentations pour les événements (sous forme de documents XML, d'objets Java, de tuples relationnels, etc.).

Pour les M-Traces, les observés peuvent être considérés et représentés comme des événements. Cependant, différents modèles et formats de données sont utilisés pour représenter des événements.

- Dans une représentation d'événements typés sans données, un événement est représenté comme un symbole unique qui marque son type ne disposant pas d'autres données qui lui sont associées. Un exemple serait `action_open` et `action_close` pour signaler (grâce à des capteurs) une ouverture ou une fermeture d'une porte. Cette représentation est assez limitée et ne fonctionne bien que pour des observés très simples pour des applications très limitées.
- Dans une représentation d'événements typés avec des attributs, un événement a un type et fournit des données sous la forme d'une paire nommée attribut-valeur e.g. `action(id="11", name="open")`. Pour ce type d'événements, la représentation des observés est semblable au modèle relationnel.
- Dans une représentation des événements comme des tuples relationnels, un événement peut être représenté par un symbole indiquant le type de l'événement et les valeurs pour les attributs. Les types respectifs sont généralement définis à l'avance dans un schéma.
- Dans une représentation des événements comme des objets, un événement est représenté comme un objet d'un langage de programmation orienté objet (Java, C++, etc.). Pour écrire ces objets sous une forme lisible par l'homme, la notation d'événements typés avec des attributs est couramment utilisée. Cependant, l'utilisation des objets pour les observés de la M-Trace permet de prendre en compte des aspects généralement pas couverts par les modèles précédents. Les systèmes orientés objet permettent de mettre en œuvre l'héritage de type.
- Dans une représentation des événements comme des messages XML, un événement peut être représenté comme un message XML. Le modèle de données de XML peut être vu comme un arbre étiqueté, ordonné. Contrairement aux représentations précédentes des événements, XML fournit non seulement un modèle de données mais aussi et surtout un format de sérialisation. Cela est particulièrement pertinent pour la représentation des M-Traces notamment les modèles.

La représentation des observés et des modèles de trace dépend de la représentation utilisée pour les événements. Cependant, bien que les différentes représentations puissent se révéler limitées, elles peuvent être comblées par la puissance des langages utilisés dans les systèmes CEP.

52. «Event : An object that represents, encodes or records an event, generally for the purpose of computer processing.» Event Processing Glossary).

Réification des requêtes et des transformations de M-Traces

Historiquement, les langages pour l'interrogation et la détection d'événements complexes ont leurs racines principalement dans les systèmes de base de données actives. Voici quelques exemples : le langage COMPOSE de base de données active Ode [GJS92a, GJS92b, GJS93], le langage de détection d'événement composite dans la base de données active SAMOS [GD92, GD94], Snoop [CM94] et son successeur SnoopIB [AC06], la formulation proposée dans [Ron98], GEM [MsSS97], SEL [ZS01], le langage dans [ME01], le langage dans [HV02], Amit [AE04], le langage dans [CL04], CEDR [BCm06], le langage dans [BKK04], ruleCore [SB05, MA], SASE [WDR06], le langage dans [SSSM05] et le langage XChange^{EQ} [BE07].

Ces langages sont aussi souvent appelés algèbres d'événements parce qu'ils peuvent être considérés comme un ensemble (d'événements simples) associé à certaines opérations (les opérateurs de composition). Toutefois, le terme « algèbre » suppose de nombreuses propriétés qui, souvent, ne se vérifient pas dans ces langages comme une restriction aux opérateurs binaires, une richesse dans les lois pour les expressions de restructuration (e.g. associativité, commutativité), les éléments neutre et inverse, ou la fermeture (i.e., le résultat d'une opération est membre de l'ensemble de base) [53]. Pour cela, nous éviterons d'employer le terme d'algèbre d'événements dans la suite et parlerons plutôt des langages d'opérateurs de composition d'événements.

Pour expliquer les opérateurs de composition dans les applications CEP, il est conceptuellement commode d'imaginer tous les événements simples comme étant reçus dans un seul flux. Évidemment, le flux peut contenir des événements de différents types. Les opérateurs de composition peuvent être considérés comme des fonctions dont l'entrée et la sortie sont des flux d'événements. Les exemples typiques des opérateurs de composition sont la conjonction binaire et la séquence binaire. Ils prennent en entrée les deux flux courants, et permettent de produire en sortie un seul flux d'événements composé. La conjonction est souvent écrites $A \wedge B$ et signifie informellement que les événements de types A et B doivent se produire (à des moments différents), indépendamment de leur ordre, pour donner une réponse pour cet opérateur. La séquence est souvent écrites $A; B$ et donne en outre la contrainte de que A doit avoir lieu avant l'événement B dans le temps. Il existe également des opérateurs ternaires, par exemple, la négation $A; \neg B; C$ permet de détecter une séquence de A et C, où l'événement B se produit pas entre eux [54].

Les langages pour les opérateurs de composition d'événements permettent d'écrire des requêtes sous une forme très compacte. À l'exception de très peu de langages, les expressions formées avec les opérateurs peuvent être arbitrairement imbriquées. Par exemple $(A \wedge B); C$ permettrait de détecter une conjonction $A \wedge B$ suivie par un événement de type C. Cette imbrication donne lieu à des requêtes plus expressives. Toutefois, elle pose aussi des problèmes, des ambiguïtés et des malentendus. Certaines équivalences auxquelles on pourrait s'attendre telles que $(A \wedge B); C \equiv (A; C) \wedge (B; C)$ ou $(A, B); C \equiv A; (B, C)$ ne se vérifient pas, ou pire, elles

53. En outre, le terme « algèbre » est souvent fortement lié à l'évaluation des requêtes, ce qui est différents (notamment en terme de sémantique) des requêtes exprimées par l'utilisateur.

54. Bien qu'elle soit souvent écrite $A; \neg B; C$, la négation doit être comprise comme un opérateur ternaire et n'ont pas comme une expression constituée de deux séquences binaires et une négation unaire.

sont fondées ou pas en fonction des diverses sémantiques des opérateurs. Nous donnons à titre d'exemple le cas de l'opérateur de séquence.

L'opérateur de séquence notée $A; B$ semble plutôt intuitive, il ne détecte que les évènements complexes composés d'un événement de type A suivie par un événement de type B. Un examen plus attentif révèle cependant qu'il existe au moins six sémantiques différentes pour cet opérateur de séquence [ZS01]. Elles sont obtenues en faisant varier l'opérateur dans deux aspects : d'abord les trois façons d'interpréter ce que signifie qu'*un événement est suivi par un autre événement*, et d'autre part les deux façons de définir les temps des événements complexes. En effet, *un événement a est suivi par un autre événement b* peut être interprété comme :

- Le temps d'occurrence de a est temporellement antérieure à celui de b, sans autres restrictions ;
- Le temps d'occurrence de a est temporellement antérieur à celui de b et il n'y a aucun événement entre a et b qui soit pertinent pour la requête, e.g., si la requête est $(A; B) \wedge C)$ alors il ne devrait pas y avoir un événement de type C entre a et b, mais il peut y avoir des événements d'un type D qui n'est pas A, B ou C) ;
- Le temps d'occurrence de a est temporellement antérieur à celui de b et il n'y aucun événement, quel que ce soit son type, entre a et b.

Les deux derniers cas peuvent aussi être pensés comme étant ce qui se produit immédiatement après b. Une question connexe est également abordée dans [WRGD07], où l'opérateur *next*, qui identifie un événement unique qui suit immédiatement un événement donné est défini et sa complexité analysée [55]. L'occurrence de temps d'un événement peut être :

- des instants dans le temps, ce qui implique que le temps d'occurrence d'une réponse à $A; B$ est le même que le temps d'occurrence d'événement b, ou
- des intervalles de temps, ce qui implique que le temps d'occurrence d'une réponse à $A; B$ est l'intervalle de temps qui commence au début de l'événement A et se termine à la fin de l'événement B.

Utiliser des instants ou des intervalles de temps peut avoir des conséquences profondes. Considérons une une requête $A; (B; C)$ sous la première interprétation de *suivi* (i.e. non immédiatement) et les événements b, a, c arrivant dans cet ordre et étant de types B, A et C, respectivement. Sous la sémantique des instants, la réponse à la requête donne : b et c qui satisfont $(B; C)$ avec le temps d'occurrence de l'événement de c (un instant), qui à son tour est postérieur à celui de a. Sous la sémantique des intervalle de temps, les événements ne donnent pas une réponse à la requête : b et c satisfont bien $(B; C)$, mais leur temps d'occurrence est l'intervalle à partir de b qui ne peut pas se produire plus tard que a.

La sémantique des opérateurs a également des conséquences sur les équivalences entre les différentes requêtes. Il semble intuitif qu'un opérateur de séquence soit associatif, i.e., $A; (B; C)$; et $(A; B); C)$ sont équivalents. L'exemple utilisé pour contredire la sémantique des instants et des intervalles montre cependant que ce n'est pas le cas dans la sémantique des instants, et que

55. Noter qu'on peut avoir plusieurs événements qui suivent immédiatement, spécialement dans les cas où plusieurs événements se produisent en même temps.

c'est le cas dans la sémantique des intervalles [GA02].

Puisque la sémantique des intervalles semble être plus naturelle pour des événements complexes, donnant des équivalences intuitives, les plus récents langages du CEP l'ont adopté. Il y a aussi une tendance à interpréter des séquences comme non-immédiatement suivies par les événements, i.e., d'autres événements pourraient se produire dans l'intervalle. Dans de nombreux langages, une séquence « immédiate » peut encore être exprimée en utilisant la négation explicite des événements qui ne doivent pas se produire dans l'intervalle de temps.

Les langages pour l'interrogation d'événements offrent une quantité considérable d'opérateurs différents en plus de la séquence. Au cœur de toutes les langages, les opérateurs suivants peuvent être identifiés : la conjonction, la disjonction et la négation. L'opérateur de négation permet de détecter l'absence d'événements. Cet opérateur est important dans de nombreuses applications, notamment pour détecter les dysfonctionnements, les indisponibilités ou les traitements différés. Pour cela, la majorité des langages offrent un opérateur de négation. Étant donné que les flux d'événements entrants peuvent être potentiellement illimités, l'absence d'événements n'a de sens que sur un nombre restreint et fini d'événements, extraits moyennant une « *fenêtre* » sur le flux. Sinon la requête devra attendre la fin du flux, qui peut jamais arriver, pour s'assurer que l'événement ne se produira pas.

En règle générale, la négation est un opérateur ternaire avec la première opérande précisant le début de la fenêtre, la deuxième l'événement qui doit être absent, et la troisième la fin de la fenêtre. Bien qu'elle soit écrite comme $A; \neg B; C$, la négation est vraiment un opérateur ternaire, et non une combinaison des deux opérateurs de séquence avec un opérateur de négation unaire. Dans les langages où les événements peuvent être des intervalles de temps, de tels événements intervalle peuvent être aussi utilisés pour spécifier la fenêtre (au lieu de deux événements pour le début et la fin). Par exemple, le langage XChange [BBEP05], où l'opérateur de négation est décrit comme `not` A `during` B (B étant une requête complexe satisfaite dans un intervalle).

Discussion

En terme de modélisation de M-Traces, les systèmes CEP montrent leurs limites puisque les représentations considérées se focalisent et correspondent au niveau des observés, ce qui peut être vu comme un inconvénient lors de la considération de M-Traces et non seulement d'observés (e.g. le domaine temporel, la représentation d'éléments non temporels comme les relations types, etc.). Cependant, ces limites peuvent être comblées et se révéler ne pas être bloquantes et décourageantes à l'usage des langages CEP pour les M-Traces. Une des raisons principales est la richesse des langages CEP pour l'interrogation et la transformation d'événements. En effet, en plus des quatre opérateurs montrés jusqu'ici, la séquence, la conjonction, la disjonction et la négation, qui sont au cœur de presque tous les langages pour le CEP, il y a une multitude d'opérateurs supplémentaires qui ont été proposés dans plusieurs langages. Cependant, quoique le fait de pouvoir ajouter facilement de nouveaux opérateurs soit considéré comme étant un point fort du CEP, il montre avant tout une certaine faiblesse : il n'existe jusqu'à présent aucun

consensus sur l'ensemble des opérateurs de composition offrant une expressivité suffisante pour la plupart des applications CEP. Ce manque de sémantique formelle standardisée est un élément rebutant pour le choix de certains langages CEP pour les M-Traces comme nous aller le discuter et montrer dans la suite.

Par exemple, certaines incongruités et ambiguïtés qui peuvent se produire lorsque les événements se produisent sur des intervalles de temps puisque la séquence n'est pas le seul ordre temporel dans lequel les événements peuvent se produire. Les temps d'occurrence des intervalles de deux événements peuvent également se chevaucher, se contenir les uns les autres, etc. Allen [All83] a identifié 13 relations possibles : `before`, `contains`, `overlaps`,`meets`, `starts`, `finishes`, `equals`, et les leurs inverses respectifs `after`, `during`, `overlapped by`, `met by`, `started by`, `finished by` (pour `equals`, l'inverse est égal à lui même). Et certains langages (e.g. [Ron98]) proposent des opérateurs de composition à base de ces relations. Dans ce cas, l'opérateur de séquence (sous l'interprétation de « non-immédiatement suivi ») est le même que l'opérateur `before`. Cependant, il existe une différence importante entre une relation temporelle comme `before` et l'opérateur de composition correspondant. Comme relation temporelle, cet opérateur est tout simplement une fonction faisant correspondre à deux intervalles de temps une valeur de vérité. Par contre, comme opérateur de composition, `before` a un fonctionnement similaire à la séquence, c'est une fonction qui prend en entrée deux flux d'événements pour produire un flux de sortie d'événements (dits complexes). Cette différence peut être problématique lors du choix d'un langage CEP puisqu'elle peut conduire à des interprétations erronées des opérateurs. Notre sémantique réifie clairement cette distinction par différentiant les filtres dans les patterns et les templates dans les règles de transformation de M-Traces.

À l'image de notre langage pour les M-Traces, les relations temporelles peuvent être considérées dans les langages CEP comme qualitatives, dans ce sens qu'elles ne concernent que l'ordre relatif des événements. Cependant, une requête peut aussi demander en plus des précisions quantitatives ou des métriques telles que deux événements se produisent dans un laps de temps spécifié. Dans le cas de flux non bornés, de telles contraintes métriques sont importantes pour restreindre les requêtes d'une manière à ce que leur évaluation ne puisse supprimer tous les événements stockés durant un laps de temps fixé et fini.

Certains langages CEP ont introduit des variations dans les opérateurs de la description de relations temporelles métriques [HV02, CL04]. Par exemple $(A;B)_{2h}$ précise que A et B divent se produire dans des séquences et dans 2 heures. Par ailleurs, certains langages, fondés sur des événements associés à des intervalles, ont défini un seul opérateur unaire C `within` $2h$ qui restreint la durée d'un événement complexe [BE07]. Noter que par le biais d'imbrication d'expressions, ce seul opérateur unaire donne essentiellement le même effet que les différentes variantes des opérateurs. Par exemple $(A;B)_{2h}$ peut être exprimé soit en laissant $C := (A;B)$, i.e., $(A;B)$ `within` $2h$. Cependant, cet opérateur n'a de sens que sur les événements associés à des intervalles.

Il est à noter que ces restrictions temporelles offertes par plusieurs langages pour les applications CEP sont assez limitées en terme d'expressivité. Une requête précisant que A se produit,

puis B se produit dans l'heure après A, puis C arrive dans l'heure après B, ne peut pas être exprimée dans la plupart des langages de CEP contrairement à notre langage pour les M-Traces. En effet, ni $((A;B)_{1h};C)_{1h}$, ni $(A;(B;C)_{1h})_{1h}$ ne permettent de résoudre la requête correctement. Les deux expressions requièrent que C doit se produire dans l'heure après A (mais pas B). Il serait possible de définir un opérateur n-aire de séquence temporelle restreinte comme $A;_{1h} B;_{1h} ; C$ qui peut résoudre cette requête. Cependant, au meilleur de notre connaissance, aucun langage CEP n'offre un tel opérateur.

Les opérateurs CEP les plus intéressants pour les M-Traces concernent les relations temporelles relatives et absolues. Les événements peuvent être définis par un lien temporel relativement à d'autres événements, par exemple, un événement `1 hour after A` serait relatif à l'événement A. Les relations temporelles relatives sont importantes en particulier dans le cadre de la négation. Par exemple une requête demandant les événements qui ne se sont pas produits `1 hour after A`, i.e. qu'elle n'a pas reçu de réponse B dans l'heure, sera désignée comme une négation de B entre l'événement A relativement à $A + 1h : A; \neg B; A + 1h$. Noter que la syntaxe $A + 1h$ (qui imite [GJS93, MsSS97] entre autres) est ambiguë dans le cas d'une requête considérant l'événement de type A à multiples reprises.

CEP regorge encore plus d'opérateurs qui n'ont pas été abordés dans ce chapitre. Un rapide tour d'horizon inclut : la répétition (dans le sens d'une opération de fermeture) qui collecte les événements du même type (e.g., *-opérateur [GJS92b, MZ97]). La sélection de certaines occurrences d'événements tels que que tous les n-ième événements (e.g., `every` n dans [Pat05]), et des versions étendues de comptage qui permettent des événements de différentes type (e.g. `ANY` dans [CKAK94, AC06, GA02], m `of` dans [BE07], `atleast`, `atmost`, n`th` dans [AE04]).

5.2.4 M-Traces et Systèmes à Base de Règles (SBRs)

D'un point de vue historique, les systèmes à base de règles sont issus de l'intelligence artificielle, et en particulier des systèmes experts. L'objectif des SBRs était alors de reproduire le raisonnement d'un expert (règles de décision, mécanismes cognitifs) essentiellement pour l'aide à la décision. On parle aujourd'hui de systèmes à base de règles «métier» (Business Rules Management System (BRMS)) ou encore de moteurs de règles «métier».

Les SBRs sont une approche classique pour mettre en œuvre des systèmes à base de connaissances avec une représentation des connaissances sous forme de règles [56]. Techniquement, un SBR est constitué d'une base de connaissance (une base de faits et une base de règles) et d'un moteur d'inférence, permettant de produire un nouveau fait à partir des règles et des faits présents dans la base.

Les SBRs ne disposent pas d'un langage pour les M-Traces en tant que tel. Cependant, ils offrent une manière assez facile et très flexible pour mettre en œuvre des requêtes ou des transformations. La principale raison pour laquelle les règles sont plus pratiques pour la mise en œuvre des requêtes qu'un langage de programmation classique est que leur algorithme d'évalua-

56. Appelées aussi des règles de production définies en premier par Post [Pos43]

tion en chaînage avant est très similaire aux algorithmes utilisés pour évaluer les requêtes et les transformations de M-Traces.

Réification des M-Traces, des requêtes et des transformations

Le premier moteur de règle a été OPS [FM77]. Depuis, beaucoup d'autres SBR ont été développés dans le monde de la recherche et de l'industrie, e.g., Drools (aussi appelée JBoss Rules) [JBo], ILOG JRules [ILO], Jess [SNL]. Les exemples présentés dans cette section sont basés sur Drools, mais les autres langages de règles sont très similaires.

Dans les SBRs, une règle, parfois aussi appelée règle *condition-action* (CA) (par opposition aux règles ECA (event-condition-action)), est une déclaration de la forme `WHEN` *condition* `THEN` *action*. Elle précise que l'action doit être exécutée chaque fois que la condition devient vrai. Il convient de souligner que l'action est exécutée seulement une fois l'état devenu vrai (i.e. à chaque fois que sa valeur change de vérité de *faux* vers *vrai*), et pas tout le temps où la condition est vérifiée (i.e. a la valeur *vrai*).

La condition est évaluée sur un ensemble de faits appelés mémoire de travail (*Working memory*). Un moteur de règles est généralement étroitement couplé à un langage de programmation comme Lisp ou Java. Par conséquent, les faits sont généralement représentés dans le modèle de données de ce langage de programmation comme des termes Lisp ou des objets Java. Les faits doivent être explicitement insérés ou supprimés de la mémoire de travail. Pour cela, le moteur de règle offre deux opérations généralement appelés *assert*[57] et *retract*.

Il n'existe aucune restriction sur la partie `action` de la règle, elle peut être une méthode arbitraire ou un appel de procédure dans le langage de programmation hôte. Particulièrement intéressantes sont les actions qui changent la mémoire de travail en faisant des inférences de nouveaux faits, des rétractations ou mise à jour des faits existants. Les règles peuvent être utilisées pour implémenter des inférences déductives simplement en faisant des assertions des faits déduits. À cette fin, de nombreux SBRs offrent également des *inférences logiques* (en plus de l'inférence standard), qui a pour effet que le fait inféré se rétracte automatiquement lorsque sa condition associée redevient fausse.

Les SBRs peuvent utiliser une évaluation de type chaînage avant dans des cycles dits *match-act*. Leur évaluation doit être explicitement invoquée avec un appel de méthode comme `fireAll()` du moteur de règles à partir du langage de programmation hôte[58]. Lorsqu'elle est invoquée, le moteur de règle vérifie toutes les conditions des règles présentes en mémoire de travail (la partie *match*). De toutes les instances de règles qui peuvent être déclenchées, il en choisit une seule selon certains stratégie de résolution des conflits. L'instance de cette seule action est alors exécutée (la partie *act*) et modifie éventuellement la mémoire de travail. Ce cycle *match-act* est ensuite répété jusqu'à ce qu'il n'y a plus d'instance de règle qui puisse déclencher. (Noter que des ensembles de règles sans terminaison sont possibles.) Des algorithmes tels que Rete [For82]

57. Depuis que de nombreux langages de programmation plus récents utilisent *assert* comme un mot clé, l'opération d'insertion est souvent renommé en *insert* pour éviter les erreurs.

58. i.e. le langage de programmation du moteur d'inférence

peuvent réaliser l'appariement (i.e. la partie *match*) de manière assez efficace par la mise à jour incrémentale des résultats de la phase *match* lorsque la mémoire de travail change. Ceci permet d'éviter ainsi de re-vérifier toutes les conditions à chaque cycle de *match-act*.

```
rule  " pdf document detection "
when
   o : file() ,
   o . ext  == "pdf"
then
   System . out . println (" the file " + o . id + " is a pdf document .");
end
```

(a) Une règle pour détecter des fichiers de type `pdf`

```
rule  " ActionPaste instance Construction "
when
   x : action() ,
   x . name == "Copy" ,
   y : action() ,
   y . name == "Paste" ,
   Minuteur . withinDSecond ( x . start , y . start , 30) ,
then
   WorkingMemory . assertObject ( new ActionPaste ( x . start , y . end ) );
end
```

(b) Une règle pour construire des observés de type `ActionPaste`

```
import java . util . Calendar ;

public static class Minuteur {
   public static boolean withinDSecond ( Calendar t1 , Calendar t2 , int d ) {
          Calendar t1PlusD = t1 . clone (). add ( Calendar . SECOND , 30);
          return t2 . before ( t1PlusD );
   }
}
```

(c) Une classe Java auxiliaire

TABLE 5.1 – Exemple d'une requête complexe de M-Trace contenant deux règles

Les SBRs permettent de mettre en œuvre des requêtes et des transformations «manuellement» (1) en inférant un fait pour chaque observé qui se produit et (2) en écrivant chaque requête comme une condition sur ces faits. Pour assurer une construction des observés en temps opportun, l'évaluation des règles doit être invoquée (avec `fireAll()`) après chaque inférence d'un fait observé. L'utilisation des règles pour les M-Traces signifie essentiellement que les conditions d'une transformation sont converties en expressions sur l'état de la mémoire de travail.

La figure 5.1 montre l'exemple de deux règles. La première figure 5.1(a) réifie comment reconnaitre des fichiers de type `pdf` avec la règle exprimée dans le langage de règles du SBR Drools basé sur Java (Drools Rule Language). Cette règle suppose que les objets Java de la classe `file` (ayant des attributs `file.id`, `file.ext`) sont insérés en mémoire de travail lorsque des observés se produisent. La règle se déclenche et effectue son action (un simple message sur la console), si un objet *o* de type `file` ayant l'attribut `ext` est égal à `pdf`. La seconde règle 5.1(b)

permet d'inférer un fait en rajoutant à la mémoire de travail un observés de type `ActionPaste`. Cette assertion est faite si (1) une combinaison des objets x, y de type `action` se trouvent dans la mémoire de travail et satisfont les conditions temporelles et (2) la règle n'a pas déjà été déclenchée avant pour cette combinaison particulière.

Discussion

Les SBRs sont conçus pour fonctionner sur des faits, pas des observés de M-Trace. Par conséquent, les langages pour les règles ne sont pas conçus pour exprimer les aspects temporels souvent nécessaire dans les requêtes et les transformations de M-Traces. Toutefois, il est possible dans une certaine mesure de mettre en œuvre ces aspects temporels comme une condition sur les attributs de l'objet représentant un *observé*.

Par exemple, chaque objet décrivant un observé pourrait porter des attributs temporels signifiant son temps début et de fin ou l'instant dans lequel il s'est produit. Mettre des instants de temps ou des intervalles est un choix de conception laissée au programmeur. Parce que les SBRs reposent sur les types et les fonctions qui sont disponibles dans leur langage de programmation, les instants du temps sont mieux pris en charge dans la plupart des langages de programmation. Souvent, l'usage d'instants de temps est pratique s'ils sont suffisants pour l'application considérée.

Considérons l'exemple de la construction des observés `ActionPaste`. Cette règle énonce la contrainte que les actions doivent se produire dans les 30 secondes. Lorsque la classe `action` a des attributs supplémentaires `start` et `end` (par exemple, du type Java Calendar, ce qui représente un instant dans le temps), les contraintes temporelles peuvent être exprimées sur ces attributs. La condition temporelle fait appel à une fonction d'assistance extérieure `withinDSecond` qui réalise le calcul temporel nécessaire sur les objets Java Calendar (figure 5.1(c)).

Bien qu'il soit possible d'exprimer les conditions temporelles comme des conditions sur les attributs des objets représentant les observés, les SBRs ne donnent aucune orientation pour les programmeurs sur la façon d'exprimer ces conditions temporelles et la façon de concevoir leurs classes d'observés de manière appropriée (par exemple, avec des attributs pour les bornes temporelles). Comme nous allons le montrer plus tard, la manière dont les règles de l'exemple donné plus haut sont écrites est influencée de façon significative par la conception de notre langage pour les M-Traces. Ni les langages pour les SBRs, ni Java n'offrent un appui suffisant pour exprimer des conditions temporelles pour traiter des M-Traces. Les fonctions d'assistance telles que celles de la figure 5.1(c) seront nécessaires dans plus d'un cas. Noter que les fonctions d'aide peuvent avoir des effets secondaires arbitraires. Dans l'exemple, il serait facile d'oublier d'appeler `clone()` et donc de commettre l'erreur de modifier le temps d'un objet dans la mémoire de travail. Enfin, l'évaluation des règles n'est pas informée de la sémantique particulière de conditions temporelles et ne peut pas les utiliser pour optimiser l'évaluation d'une requête ou d'une transformation.

En fait, les SBRs offrent peu de support pour les relations temporelles. Si les relations

temporelles relatives (par exemple, «12 heures après l'observé x») sont nécessaires dans une requête ou une transformation, un fait doit être déclaré au moment où cet observé est censé se produire. Ceci doit être fait à l'extérieur du moteur de règles dans le code Java courant, ce qui entraîne l'utilisation de fonctionnalités de bas niveau pour programmer l'exécution d'un `thread` qui infère le fait correspondant à l'exécution en temps donné (par exemple, en utilisant la classe Java `Timer`). En outre, il viole le principe que la logique de traitement des M-Traces doit être encapsulée dans les règles de transformation. La transformation des M-Traces ne peut donc pas être simplement étendue ou modifiée en changeant les règles, un code externe autre doit être aussi changé et modifié en conséquence.

Certains langages pour les SBRs permettent de retarder l'exécution d'une règle pour une durée donnée (dans Drools, ceci peut être exprimé par le mot-clé `duration`). Ceci pourrait être utilisé dans certains cas comme un substitut pour les relations temporelles relatives, mais reste une solution assez limitée.

Inférer un fait, pour chaque observé qui se produit est un problème pratique : au fil du temps, le nombre de faits et donc la taille de la mémoire de travail se développe et grandit. Partant du principe qu'un flux d'observés est potentiellement illimité dans le traitement des M-Traces, nous finirons par déborder la mémoire de travail, si nous insérons des faits sans jamais les rétracter. Pour éviter le débordement de mémoire, il faut une certaine manière de les rétracter, pour supprimer les faits représentant des observés qui n'ont plus d'importance, i.e, une sorte de *garbage collection* des faits observés.

Puisque les langages actuels pour les SBRs ne sont pas conçus pour fonctionner avec les observés et leurs relations temporelles, ils n'offrent généralement pas d'aide pour la récupération de mémoire. Noter également que le ramasse-miettes automatique du langage hôte (e.g. Java) ne sera pas d'une grande aide. En effet, la mémoire de travail conserve une liste de références aux objets représentant les faits, de sorte que n'importe quel objet dans la mémoire de travail soit toujours accessible et ne sera jamais soumis au *garbage collector* automatique. Globalement, le programmeur est charge de cette tâche manuellement. Cela peut se faire soit par l'écriture de règles qui rétractent les faits qui sont devenus sans pertinence, soit dans le langage de programmation hôte.

Le ramasse-miettes d'observés pose la question de ce que cela signifie pour un observé d'être pertinent. Une méthode simple pourrait être de définir un délai par défaut pour chaque type d'observé. Tout observé plus ancien que le délai d'attente est alors considérée comme sans importance. Cependant, cette définition influe sur la logique et la sémantique des SBRs puisque des délais d'attente différents pourraient conduire à des résultats de requête ou de transformation différents.

En général, il serait préférable d'utiliser les conditions temporelles dans les requêtes pour savoir quand un observé devient non pertinent. Cependant, savoir si un fait représentant un observé est pertinent ou non (i.e. exempté ou pas de garbage collection) peut être difficile. Par conséquent, la récupération de mémoire manuellement par programmation s'avère également difficile. Étant donné que les erreurs de récupération sont cruciales pour l'exactitude des requêtes

et des transformations et pourrait conduire à des débordements de mémoire, cette tâche peut être aussi dangereuse pour la cohérence de la base de faits. De plus, la récupération de mémoire conduit à un code qui est difficile à faire évoluer et à maintenir : l'ajout d'une règle unique pour une transformation est susceptible d'affecter la pertinence des faits en cause, et entraîne donc une adaptation du code pour le ramasse-miettes.

Il y a eu des recherches sur l'extension des langages et des moteurs pour SBRs pour prendre en compte les aspects concernant les relations temporelles et le récupération de la mémoire automatique [Ber02, WBG08]. En particulier, les règles sont étendues dans [WBG08] en y ajoutant des opérateurs de composition et un ramasse miettes automatique d'événements. Cependant, ces approches n'ont pas encore trouvé leur chemin dans les moteurs de SBRs actuels, mais on peut s'attendre à voir plus de travail à ce sujet dans un avenir proche.

La négation est prise en charge dans de nombreux langages pour les SBRs. Le constructeur `not` rend possible de tester l'absence de certains faits dans la mémoire de travail. Les constructions pour la négation sont évidemment destinées à la transformation de faits normaux plutôt que des observés se produisant au fil du temps. Leur utilisation typique dans des requêtes ou des transformations pour exprimer la négation (ou absence) d'un observé entre deux autres observés, est donc difficile à exprimer. Encore une fois cela nécessite l'utilisation de conditions sur les attributs et des mécanismes similaires.

5.2.5 M-Traces et eXtensible Markup Language (XML)

Extensible Markup Language (XML) [WX04] a été développé par le W3C en 1996 avec l'intention de fournir une plateforme pour définir des documents dont le contenu et le type est spécifié par l'utilisateur. XML permet la création de documents contenant des données semi-structurées [59] et apparaît aujourd'hui comme le standard de fait dans l'échange de données sur le Web.

Réification des M-Traces

Dans le monde des traces, XML est une des technologies les plus utilisées notamment pour représenter les journaux (logfiles) de fonctionnement de différentes applications comme les serveurs Web. Permettant de représenter facilement les données structurées, XML peut être une technologie suffisante pour représenter les M-Traces. Grâce aux concepts relatifs à XML, des M-Traces peuvent être décrites sous forme d'éléments, d'attributs, d'entités, etc.

XML permet de représenter facilement des M-Traces avec une réification direct de ces éléments. Les observés peuvent être représentés par des éléments XML. Les attributs peuvent être représentés comme attributs XML ou éléments. Par exemple, la figure 5.2 représente deux observés de type `action` avec trois attributs `id`, `begin` et `end` représentant leurs identifiants, temps

59. Les données semi-structurées peuvent se voir comme une relaxation du modèle relationnel classique, dans lequel on autorise une structure moins rigide et homogène des données. Ce modèle de données s'est révélé très utile dans la représentation de familles de documents variés : multimédia, hypertexte, données scientifiques.

de début et fin. Bien que `name` soit un élément, il représente cependant un attribut de l'observé.

```
<?xml version="1.0"?>
<action id=310" start="2" end="3">
  <name> Copy </name>
</action>
<action id="12" start="2" end="3">
  </relation idref="10">
  <name> Paste </name>
</action>
```

TABLE 5.2 – Exemple d'une M-Trace décrite en XML

Les relations peuvent être représentées par des références [60]. XML dispose d'un mécanisme de références structurelles qui prend en charge des références croisées dans un document XML, moyennant deux attributs particuliers : l'identifiant (ID) et la référence d'identification (IDREF) [61].

Le modèle peut être exprimé par une DTD ou un schéma. Une DTD (Document Type Definition) [GP00] permet de spécifier une grammaire pour un document XML. Dans une certaine mesure une DTD peut également être utilisée pour modéliser une trace XML. Cependant, cette solution reste très limitée pour les relations type, la représentation de temps et les hiérarchies de types et de relations type.

Pour gagner plus de liberté en précisant les types (le schémas) pour les documents XML, d'autres formalismes de schéma XML ont été introduits : XML Schema [WXS01] (plus riche que la DTD pour spécifier la structure et ayant une syntaxe XML), RelaxNG [CM01] (facile à comprendre et à utiliser, similaire à une grammaire pour un langage à base de règles).

Les schémas XML sont mieux adaptés pour les modèles de traces puisqu'ils permettent de spécifier les types des éléments disponibles et des contraintes sur leur contenu, sur leur occurrence et d'autres détails concernant la structure du document. Les schémas XML sont destinés à faciliter le traitement automatisé des documents XML. Un code fiable pour les applications fondées sur XML peut être obtenue si les analyseurs contrôlent la validité de la structure par rapport au schéma (i.e. s'il est valide) et effectue la vérification de format (i.e. s'il est bien formé). Enfin, les modèles de trace peuvent être associés aux traces avec les espaces de noms.

Un document XML induit un arbre contenant les données du document essentiellement par l'étiquetage des nœuds ou les arcs avec les noms de balises.

60. XML offre différents types de mécanisme de référence : (1) référencement dans un même document XML (en utilisant ID/IDREF mécanisme) (2) la liaison des documents XML sur le Web (en utilisant la recommandation du W3C XLink).

61. Le mécanisme ID/IDREF est très simple : les identifiants sont utilisés pour identifier les éléments (ce qui signifie que les valeurs des attributs d'identification doivent être uniques), et les références d'identification sont utilisés pour faire référence à un élément à l'intérieur d'un document XML ayant un identifiant unique.

Réification des requêtes et des transformations de M-Traces

Le World Wide Web Consortium (W3C) définit deux langages standard pour l'interrogation et la transformation de document XML : XSL Transformations (XSLT) [62] [Cla99, Kay07] et XQuery [Boa07]. Les deux langages XSLT et XQuery font usage de XPath [Ber07], un langage pour naviguer dans l'arborescence de documents XML. Dans sa syntaxe abrégée, les expressions XPath rappelent la manière dont les chemins de répertoire dans les systèmes de fichiers sont traités.

XSLT est le plus simple des deux langages. Comme l'indique le terme «stylesheet» dans son nom, XSLT est destiné aux tâches de transformation visant à changer le style, i.e., la manière dont l'information est présentée pour un document XML. Ainsi, XSLT est principalement destiné aux transformations des documents qui concernent de simples restructurations comme des renommage d'éléments, filtrage ou déplacement de sous-arbres vers le haut ou vers le bas. Une application typique de XSLT est de transformer un document XML qui utilise un vocabulaire spécifique dans une application dans un document (X)HTML approprié pour le visualiser dans un navigateur. Cependant, bien que XSLT soit conçu comme langage de transformation de styles de présentation, il peut être également utilisé pour transformer un document XML vers un autre.

Un programme XSLT est constitué de templates, qui précisent les patterns à reconnaitre en entrée ainsi que la sortie à produire. Le traitement d'un document XML commence à la racine et choisit une template avec un pattern qui « match » la racine. La sortie de cet template peut alors causer des correspondances récursives sur les templates. Généralement, la récursivité est sur les fils du nœud courant (cependant XSLT permet de spécifier des exceptions à ceci, quand les nœuds pour la correspondance récursive sont explicitement désignés ou des templates explicitement appelés). Cette manière de transformer prend bien en compte des transformations qui renomment, insèrent des éléments entre un nœud et ses fils (e.g. en déplaçant un sous-arbre « vers le bas »), suppriment des éléments entre un nœud et ses descendants (e.g. en déplaçant une sous-arborescence « vers le haut »), ou filtrent certaines parties en entrée.

Un exemple d'un programme XSLT pour transformer un document comme celui de la Figure 5.2 en un document XML est illustré à la figure 5.3. Ce programme permet de créer un observé de type `Copy_Action` pour chaque élément `name` dans `action` ayant un contenu égal à `Copy`. Ces observés ont deux attributs `start` et `end` dont les valeurs sont égales aux attributs `start` et `end` des éléments `action` reconnus.

Le second langage pouvant servir à l'interrogation et transformation de M-Traces est XQuery. XQuery est conçu pour effectuer des requêtes plus générale et extraire des informations à partir de grands volumes de données XML. XQuery est considéré comme plus expressif que XSLT et aussi plus riche en fonctionnalités (et donc plus difficile à apprendre). Considérant que XSLT est destiné à transformer un document XML unique, XQuery est particulièrement adapté pour les tâches nécessitant l'extraction et la combinaison de données (également à partir de plusieurs sources) comme la jointure, le regroupement et l'agrégation des données ou leur réordonnance-

62. eXtensible Stylesheet Language Transformations.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="//action/name[text()="Copy"]">
  <xsl:element name="Copy_Action">
    <xsl:attribute name="start">
      <xsl:copy-of select="../@start"/>
    </xsl:attribute>
    <xsl:attribute name="end">
      <xsl:copy-of select="../@end"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

TABLE 5.3 – Exemple d'un programme de transformation XSLT

ment.

Un programme XQuery est essentiellement constitué des déclarations FLWOR [63] pour les clauses `For-Let-Where-Order-Return`. La clause `For` sélectionne et parcourt les nœuds à partir des documents d'entrée. Noter qu'une clause `For` peut entraîner de multiples itérations. La clause `Where` limite les résultats produits par la clause `for`. Une restriction typique serait de requérir une certaine égalité entre des nœuds de sources différentes. L'ordre d'itération peut être changé en utilisant la clause `order by`. Les résultats (pour chaque itération) sont précisées dans la déclaration `return`.

Noter que les instructions FLWOR sont similaires au `SELECT − FROM − WHERE` de SQL. XQuery a de nombreuses fonctionnalités au-delà des déclarations FLWOR de base, y compris la capacité de définir des fonctions (avec une récursion libre).

```
FOR $action in document("trace.xml")//action
LET $start:=$action/@start
LET $end:=$action/@end
WHERE $action/name/text() == "Copy"
RETURN
  <Copy_Action>
     <start> { $start } </start>
     <end>   { $end   } </end>
  </Copy_Action>
```

TABLE 5.4 – Exemple d'un programme XQuery

La figure 5.4 montre un exemple d'une requête XQuery. Le fichier `trace.xml` est celui de la figure 5.3. La clause `FOR` parcourt chacun des nœuds résultant de l'expression Xpath, i.e. la variable nommée `$action` sera liée à chacun de ces nœuds. La partie `LET` affecte les attributs `start` et `end` dans les variables nommées respectivement `$start` et `$end`. La valeur de ces variables est utilisée ensuite dans le `RETURN` pour donner respectivement le contenu des balises

63. prononcez en anglais «flower»

`start` et `end` dans `Copy_Action`. La clause `RETURN` construit le résultat de la requête restreint à l'ensemble des nœuds candidats possibles ayant le contenu de `name` égal à `Copy`.

Discussion

XML est devenu un format populaire pour l'échange d'informations et son utilisation dépasse largement cette fonctionnalité. Sa popularité peut être attribuée principalement au fait que XML s'accommode très bien avec ce qu'on appelle des données semi-structurées, qui a une structure irrégulière (parfois récursive) et souvent changeante. Ceci peut être très pratique pour les M-Traces, par exemple lors de l'évolution des modèles pour mettre en place des transformations. Là où le modèle de données relationnel exige une structure rigide, XML rend facile les changements structurels. En utilisant des noms appropriés pour les balises et les attributs en XML, les données dans les documents XML deviennent aussi auto-descriptives.

Cependant, XML a également ses limites, certaines sont bien connues, spécialement lors de la manipulation de M-Traces :

- Le modèle et les relations structurelles et temporelles entre des observés dans une M-Trace permettent de la considérer comme étant un graphe et non un arbre ordonné. Mettre en place des données dans XML sous forme de graphe structuré peut être fastidieux. L'utilisation des attributs ID et IDREF ou de XLink dans un document XML permettent de modéliser un graphe structuré d'information dans le modèle XML. Toutefois, ces mécanismes sont distincts de la relation parent-fils dans le modèle de données XML, et doivent être résolus «manuellement» (par exemple, par la jointure des valeurs ou la fonction `id()` de XPath) dans les programmes et les requêtes.
- XML offre un support limité pour l'identification des nœuds spécifiques (éléments, attributs, etc.) dans un document. Les éléments porteurs d'un attribut ID sont facilement identifiables, mais pour d'autres éléments ou d'autres types de nœuds, la tâche n'est pas si facile. L'identité d'un nœud «node identity» (généralement l'adresse mémoire de l'objet qui représente le nœud) peut être utilisée pendant une requête unique, mais pas pour l'identification des nœuds sur une plus longue période de temps. En particulier, l'identité d'un nœud pourrait être complètement perdue si le document est mis à jour, même si le changement est mineur (e.g., l'ajout d'un élément). Ces questions liées à l'identité sont également discutés dans [Fur08].
- La nature auto-descriptive du langage XML vise à rendre les documents compréhensible pour les humains, mais n'est pas bien adapté pour les rendre «compréhensibles» par les machines. En voyant une balise appelé `action` et `event_action`, les humains pourront facilement savoir que se sont tout simplement différents termes pour le même concept. Pour les machines ce n'est généralement pas possible. Ainsi, XML n'offre aucune façon d'indiquer explicitement pour une machine que ces termes sont sémantiquement proches ou équivalents et doivent être traités de la même manière. Enfin, étroitement lié à cet argument, XML est purement un format pour la représentation des données. Son modèle

de données et les formalismes associés pour le typage des données, ne supportent pas des mécanismes d'inférence tels que la déduction de nouveaux faits.

Concernant l'interrogation et la transformation de M-Traces, XML dispose sans doute des langages les plus intéressants pour la manipulation de M-Traces. D'ailleurs, c'est un des langage les plus utilisés pour la manipulation de traces notamment dans le cadre de notre projet « personnalisation des EIAH » [AMM+07]. Cependant, XSLT et XQuery trouvent rapidement leurs limites notamment pour deux points essentiels :

- XSLT ou XQuery ne permettent pas explicitement d'exprimer des relations temporelles ou structurelles autres que les relations père-fils des arbres XML. L'interrogation et la transformation de graphes décrit en XML est au mieux très contraignante pour la description des requêtes et les transformations (puisque nécessitant des jointures et des techniques spécifiques) et au pire très complexe pour leurs évaluations.
- XSLT ou XQuery ne permettent pas des évaluations continues sur des flux XML.

Comme pour SQL, ceci nécessite la définition de nouvelles extensions et sémantiques pour prendre ces deux aspects. Par exemple, le travail de [BKF+07] propose une extension de XQuery en ajoutant la notion de fenêtres temporelles sur les flux XML. Une autre approche définie étend un autre langage pour XML : le projet Niagara [CDTW00], où le flux de document XML est interrogé par un langage qui est une variante de XML-QL [64]. Cependant, l'usage de ces langages pour les M-Traces avec des modèles impliquant des hiérarchies de types et de relations nécessite plus d'investigations et de spécifications.

5.2.6 M-Traces et Resource Description Framework (RDF)

RDF, acronyme pour Resource Description Framework [MM04, KC04], fournit une syntaxe et une sémantique pour représenter des connaissances sur le Web, que l'on appelle les ressources dans le cadre de RDF. Une ressource est tout ce qui peut être identifié par une URI [65]. Ceci inclut les documents Web particuliers, mais aussi des objets tels que des objets physiques pour la vente dans un catalogue qui ne sont pas «effectivement sur le Web» (i.e. ne peuvent pas être trouvés par le biais du protocole HTTP ou un autre protocole).

Les ressources sont décrites au moyen de déclarations sous la forme `sujet` − `prédicat` − `objet`, appelées des triplets RDF. Le prédicat est aussi appelé une *propriété* de la ressource décrite, et l'objet *valeur de la propriété*. Le sujet et le prédicat sont donnés avec des URI. L'objet peut être donné avec une URI ou être un littéral, i.e. une valeur constante donnée comme une chaîne de caractères tels que «toto». (Les nœuds vides ou *Blank nodes* sont une exception pour les sujets et les objets qui sont discutées dans la suite.)

Les triplets RDF forment essentiellement un graphe non-ordonné, orientée et étiquetés. Les nœuds dans ce graphe sont les sujets et les objets, et chacun triplet RDF décrit un arc de son sujet à son objet avec le prédicat en tant que label. Noter que les nœuds littéraux ne peuvent

64. http ://www.w3.org/TR/NOTE-xml-ql/
65. Uniform Resource Identfier

pas avoir des arcs sortants.

Dans certains cas, il est nécessaire ou tout simplement pratique de créer des déclarations sur des ressources, sans savoir ou avoir une URI pour eux. Par exemple, on peut vouloir exprimer qu'il y a une copie d'un document pdf, sans connaître l'URI pour ce fichier pdf. À cette fin, RDF permet l'usage de nœuds vides. Les nœuds vides peuvent être utilisés au lieu des URI pour les sujets et les objets pour exprimer qu'il y a une ressource avec des propriétés spécifiées sans pour autant identifier cette ressource.

RDF est augmenté par RDFS (RDF Schema) qui est le langage de description de vocabulaire RDF [BG04]. RDFS fournit un système de typage permettant de définir pour une application des vocabulaires spécifiques. En typant RDF, RDFS contraint le graphe RDF, par exemple, en exprimant que « concerne » est une propriété des ressources de type « action » qui a une valeur d'une ressource de type «ressource». RDFS permet des inférences simples, par exemple, si la ressource « poster.pdf » est de type « document » et « document » est une sous-classe de « fichier », alors « poster.pdf » est également de type « fichier ». La sémantique de RDF, ainsi que celle de RDF + RDF Schema, sont spécifiées formellement sous forme d'une théorie des modèles [Hay04].

En ce qui concerne la syntaxe, un graphe RDF peut être écrit, stocké et échangé en une multitude de formats appelés formats de sérialisation. Le format officiel normalisé par le W3C est la syntaxe RDF/XML [Bec04], mais il existe de nombreux formats alternatifs dont N3 [BL98, BL00].

Pour résumer, RDF va au-delà du langage XML en se fondant sur des graphes orientés comme modèles de données, en fournissant la capacité de représenter l'information (existentiellement quantifiée) moyennant des nœuds vides, permettant des déductions simples à travers RDFS et fournissant une sémantique claire sous forme d'une théorie des modèles.

Réification des M-Traces

Comme pour les SGBDs, les SGFs relationnels et les entrepôts XML, nous pouvons exprimer des modèles de traces en RDF de deux manières : modéliser le méta-modèle qui permettra de représenter les différents modèles ou représenter directement les modèles en RDF/RDFS. Contrairement aux technologies précédentes (SGBDs, SGFs, XML) qui prennent mieux en compte l'une des deux manières de représenter des M-Traces, RDF permet les deux types de modélisations. L'exemple 5.5 (rectangle de gauche) montre un vocabulaire RDF pour décrire des modèles de traces en décrivant une représentation du méta-modèle. Les observés types (`ObselType`), types de relations (`RelationType`) et attributs (`Attribute`) sont décrits comme des classes RDFS (`rdfs:Class`), les domaines (`aDomaine`, `rDomain`) et co-domaines (`aRange`,`rRange`) des attributs et des relations sont représentés comme des propriétés. La relation de hiérarchie entre types d'observé est décrite comme une sous-propriété (`rdfs:subPropertyOf`) de la relation de spécialisation des classes RDFS (`rdfs:subClassOf`). Les autres éléments (hiérarchie de types de relation, etc.) peuvent aussi être représentés avec RDFS.

L'instanciation de ce vocabulaire permettra d'exprimer des modèles de traces. L'exemple 5.5 (rectangle de droite) montre un exemple de modèle de trace représentant trois types d'observés (`Action`, `Resource`, `Document`) qui sont des instanciations de `ObselType` (la lettre `a` représente une instanciation et est équivalent à `rdf:type`), `concern` est une instance de relation type dont le domaine et co-domaine sont respectivement `Action`, `Resource` et `hasType` est une instance de `Attribute` dont le domaine et co-domaine sont respectivement `Action`, `xsd:string`.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix : <http://host/metamodel/> .

:ObselType a rdfs:Class .

:RelationType a rdfs:Class .

:Attribute a rdfs:Class .

:hasSuperObselType a rdf:Property ;
  rdfs:subPropertyOf rdfs:subClassOf ;
  rdfs:domain :ObselType ;
  rdfs:range :ObselType .

:rDomain a rdf:Property ;
  rdfs:subPropertyOf rdfs:domain ;
  rdfs:domain :RelationType ;
  rdfs:range :ObselType .

:rRange a rdf:Property ;
  rdfs:subPropertyOf rdfs:range ;
  rdfs:domain :RelationType ;
  rdfs:range :ObselType .

:aDomain a rdf:Property ;
  rdfs:subPropertyOf rdfs:domain ;
  rdfs:domain :Attribute ;
  rdfs:range :ObselType .

:aRange a rdf:Property ;
  rdfs:subPropertyOf rdfs:range ;
  rdfs:domain :Attribute ;
  rdfs:range rdfs:Datatype .
...
```

```
...
@prefix ktbs: <http://host/metamodel/> .
@prefix    : <http://host/mon_modele1> .

:Action a ktbs:ObselType .

:Resource a ktbs:ObselType .

:Document a ktbs:ObselType ;
  ktbs:hasSuperObselType :Resource .

:concern a ktbs:RelationType ;
  ktbs:rDomain :Action ;
  ktbs:rRange  :Resource .

:hasType a ktbs:Attribute ;
  ktbs:aDomain :Action ;
  ktbs:aRange xsd:string .
...
```

TABLE 5.5 – Vocabulaire et exemple d'un modèle de trace en RDF représentés avec la syntaxe N3

Réification des requêtes et des transformations de M-Traces

Pour interroger les données RDF, le W3C a récemment normalisé un langage de requêtes appelé SPARQL [PS08]. Une requête SPARQL permet soit d'obtenir un ensemble d'affectation

de variables soit de construire un nouveau graphe RDF. La première possibilité est pratique lorsque les résultats d'une requête doivent être utilisés à l'intérieur d'un logiciel, la seconde est intéressante quand le graphe RDF doit être transformé. Ces deux possibilités peuvent ainsi correspondre respectivement à l'interrogation et la transformation de M-Traces décrites en RDF. Les affectations de variables qui sont soit directement retournées à la suite ou utilisées pour la construction d'un graphe RDF résultat sont obtenues en évaluant ce que l'on appelle un pattern de graphe (graph pattern) sur les graphes RDF en entrée.

SPARQL partage beaucoup de ses mots clés avec SQL, une requête typique SPARQL est de la forme `SELECT − FROM − WHERE` ou `CONSTRUCT − FROM − WHERE` (cf. exemple 5.6). La clause `FROM` liste les URI des graphes RDF qui doivent être accessibles pour cette requête. La clause `WHERE` spécifie un pattern de graphe sous la forme d'un ensemble de patterns triplet. Comme un triplet RDF, un pattern triplet a la forme `sujet − prédicat − objet`, cependant, à l'inverse d'un triplet RDF, il peut contenir des variables libres. La même variable peut se produire dans plusieurs patterns triplet donnant lieu à un graphe. Le pattern de graphe permet l'évaluation des requêtes pour trouver les affectations de variables de sorte que le graphe obtenu à partir de substitutions de variables dans le pattern avec leurs liaisons soit un sous-graphe du graphe en entrée. La clause `SELECT` spécifie le retour (de certaines) des affectations de variables comme résultat, tandis que la clause `CONSTRUCT` construit un nouveau graphe RDF en utilisant les affectations de variables.

Au-delà de l'appariement de base de graphes (basic graph matching), SPARQL permet des disjonctions (mot-clé `UNION`), de spécifier des patterns optionnels (mot-clé `OPTIONAL`), et filtrer des résultats (mot-clé `FILTER`). En combinant les patterns optionnels et le filtrage (en utilisant la propriété `bound`), la négation et la quantification universelle peuvent être exprimées en SPARQL. Cependant, d'autres requêtes avancées (e.g., patterns impliquant la recherche de nœuds d'un graphe à des profondeurs arbitraires) ne sont généralement pas un point fort de SPARQL et vont au-delà de ses capacités expressives. En outre, SPARQL ne permet pas des inférences récursives dans les requêtes.

Comme RDF pour la représentation des M-Traces, SPARQL est particulièrement bien adapté à l'interrogation et transformation de M-Traces notamment pour un usage *a posteriori* de l'activité de traçage. L'exemple 5.6 montre une requête (la partie `SELECT`) et une transformation (la partie `CONSTRUCT`) exprimées en SPARQL. La requête permet d'extraire les actions effectuées sur des ressources de type `Document`. La transformation permet d'extraire la même chose en plus des identifiants, le temps de début des `Action` et le temps de fin des `Document` et construit un nouveau graphe représentant un nouvel observé de type `DocumentAction` ayant le temps de début et l'identifiant des actions et les temps de fin des documents concernés par ces actions.

Discussion

En terme de modélisation des M-Traces, RDF/RDFS est sans doute la meilleure solution pour la représentation des M-Traces. Étant un standard pour la représentation des connaissances

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX :    <http://host/sbt/jdoe/models/mon_modele1>

SELECT ?a, ?r
WHERE   {
   ?a rdf:type :Action ;
      :concern ?r .
   ?r rdf:type :Document .
}
```

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ktbs: <http://host/metamodel/>
PREFIX s:    <http://host/models/mon_modele1>
PREFIX d:    <http://host/models/mon_modele2>
CONSTRUCT {
    [ rdf:type d:DocumentAction
      ktbs:hasId ?id ;
      ktbs:hasBegin ?begin ;
      ktbs:hasEnd ?end ;
      d:about ?r .
    ]
}
WHERE   {
    ?a rdf:type :Action ;
       :concern ?r ;
       ktbs:hasId ?id ;
       ktbs:hasBegin ?begin .
    ?r rdf:type :Document ;
       ktbs:hasEnd ?end .
}
```

TABLE 5.6 – Exemple d'une requête et d'une transformation SPARQL

sur le Web, RDF/SPARQL dispose de liens clairs et bien identifiés avec les langages classiques de représentation de connaissances (e.g. l'équivalence entre SPARQL et Datalog [Pol07, Sch07, AG08]). Cependant, l'usage de RDF/SPARQL pour l'exploitation de M-Traces nécessite des adaptations notamment sur deux points importants :

- La prise en charge du temps que ce soit pour la représentation ([GHV07], [GHV05]), ou pour l'interrogation et le raisonnement temporel ([Aas08]) est plus orientée vers des usages non continus des graphes RDF. Les requêtes impliquant des relations temporelles relatives ou des séquences sont exprimables de manière peu intuitive ne prenant pas en compte un flux RDF en cours d'évolution.
- Certains travaux ont identifié ces limites, e.g. [BGJ08, GGKL07, BBC+09] dans le contexte du Web social. Pour y remédier, ils ont défini de nouveaux opérateurs pour SPARQL. À l'instar des SGFDs, la représentation des fenêtres temporelles se fait moyennant des mots clés comme `Window` pour [BGJ08], `Range...Step` pour [BBC+09], ou de nouveaux opérateurs comme `Stream` dans [GGKL07]. Cependant, il n'est pas clair si les requêtes (`SELECT` et `CONSTRUCT`) sont monotones ou non en présence de la différence et/ou de la

négation. En effet, une critique commune a tous ces travaux est qu'ils ne font pas une étude poussée de la monotonie des requêtes et des transformations dans ce contexte, à l'inverse des travaux dans la communauté des SGFDs (e.g. [GO05]).

Bien que l'usage de RDF et SPARQL pour les M-Traces offre la meilleure combinaison pour mettre en place une certaine classe de SBT orienté vers des usages *post-hoc* des M-Traces (voir par exemple ABSTRACT [Geo08]), elle nécessite plus d'investigations et la spécification d'une sémantique opérationnelle pour des SBT exploitant les traces de manières *ad-hoc* et *post-hoc*.

	SGBDR	**SGFD**	**CEP**	**SBRs**	**XML**	**RDF**
Modèles pour les M-Traces						
Hiérarchies de types et de relations	0	0	− (avec des exceptions)	− (conversion sous forme d'objets)	0	++
Aspects Temporels	0	+	0 (avec des exceptions)	−	−	+
Requêtes, transformations et sémantiques pour les M-Traces						
Négation	− (l'aspect temporel)	+	+	− (l'aspect temporel)	− − (l'aspect temporel)	0 (l'aspect temporel)
Occurrence et détection temporelles	−	− (dépend de l'opérateur relation-to-stream)	++	− − (à la charge du programmeur)	−	−
Transformation de M-Traces	−	− −	0	+	0	+
Sémantique formelle	+ (précise mais peu intuitive pour les M-Traces)	+ (précise mais peu intuitive pour les M-Traces)	0 (les données des observés ne sont pas considérées)	− (sous forme de programmes impératifs)	+ (peu intuitive pour les M-Traces)	+ (peu intuitive pour les M-Traces)

TABLE 5.7 – Synthèse sur la comparaison entres les différentes technologies pouvant être utilisées pour l'exploitation des M-Traces.

5.3 Synthèse et discussion des différentes technologies présentées pour l'exploitation des M-Traces

Le tableau 5.7 résume les discussions faites dans cette section à propos des différentes technologies. Les entrées dans le tableau utilisent l'échelle ++, +, 0, −, − − et permettent d'indiquer dans quelle mesure chaque approche prend en compte une certaine caractéristique des M-Traces. Comme avec n'importe quel tableau, certaines entrées seront une généralisation ; nous nous référons donc aux discussions précédentes pour une comparaison plus approfondie et des explications.

Nous pouvons remarquer un certain nombre de choses. En ce qui les modèles des M-Traces, RDF est le langage qui offre le plus de facilité pour représenter les hiérarchies de types et de relations ou les aspects temporels relatifs aux M-Traces. Cependant, pour les requêtes et les transformations continues, RDF s'avère très limité à l'instar des langages pour les SGBDR ou XML qui ne permettent pas des évaluations continues monotones notamment en présence de la négation. Les langages pour CEP offrent des opérateurs de négation et de détection temporelles relatives mais ne permettent pas d'exprimer des règles de transformation contrairement aux SBRs.

Une critique commune à tous les langages examinés jusqu'à maintenant est qu'ils ne sont pas polyvalents dans le mode d'évaluation qu'ils offrent. Certains langages ne conviennent que pour l'interrogation ponctuelle de données (SQL, règles, SPARQL, XSLT, XQuery) et d'autres sont conçus uniquement pour permettre une interrogation continue (CQL, opérateurs CEP). Toutefois, il existe de nombreux cas d'utilisations potentielles pour des usages de M-Traces qui impliquent les évaluations continues suivies d'évaluations ponctuelles. Par exemple, une transformation continue pour proposer une visualisation réflexive à l'apprenant durant l'activité d'apprentissage suivie de transformations ponctuelles pour évaluer son activité faites par le tuteur après l'apprentissage. L'association de certains langages de requêtes (par exemple, SQL et CQL) peut offrir cette polyvalence mais constitue un support peu adapté pour les transformations.

Le tableau 5.7 présenté peut suggérer également l'idée de combiner les langages pour prendre plus en compte les aspects relatifs aux M-Traces. D'ailleurs, il manque pour compléter ce tour d'horizon des langages pouvant servir à l'exploitation des M-Traces les approches hybrides qui tentent de combiner les différents styles d'interrogation. Nous pouvons citer les langages permettant la détection de patterns dans les SGFDs (e.g. [SZZA01, SZZA04], Esper [Esp]). En effet, le modèle original de CQL met en œuvre seulement des opérateurs impliquant des flux de données et des relations (i.e. les opérateurs stream-to-relation, relation-to-relation et relation-to-stream.). La détection de patterns sur des flux de données peut être ajouté à ce modèle comme un opérateur flux vers flux (i.e. stream-to-stream). Un tel opérateur permet de spécifier des patterns qui ressemblent à des expressions régulières (par filtrage des chaînes de caractères) dans les flux de données [SZZA01, SZZA04].

Une autre approche hybride permettant de combiner les capacités des opérateurs CEP avec les capacités d'un langage de SGFDs est décrite dans [GAC06, CA08]. Les flux de données entrants sont d'abord traités avec un langage de requête de flux de données, puis les événements de sortie sont traitées ultérieurement avec un langage CEP afin de déceler les événements complexes souhaités par l'application. Noter que ce travail ne propose pas un langage qui intègre les capacités des deux technologies, mais préconise plutôt un modèle architectural afin d'utiliser les deux langages dans une seule application, et donc la combinaison de leurs avantages respectifs. En comparaison avec l'approche hybride précédente concernant l'ajout de pattern à un langage SGFD, cette approche peut être considérée comme étant son inverse : elle applique d'abord la détection de patterns par le biais d'opérateurs CEP puis les requêtes de flux de données.

La dernière approche concerne le couplage des technologies CEP dans les système à base de règles. Le travail de [WBG08] vise à ajouter des opérateurs CEP à un SBR basé sur l'algorithme

Rete. Des événements se produisent sur des intervalles de temps et les opérateurs CEP proposés sont fondés sur les relations intervalle d'Allen [All83]. Cependant, ce travail ne propose pas un opérateur de négation.

Nous pourrions citer encore plus de langages. Cependant, à l'instar des langages présentés dans le tableau 5.7, aucune de ces approches hybrides ne satisfait toutes les exigences de la sémantique formelle des M-Traces. Entre la démarche de combiner deux langages et d'étendre un des langages pour atteindre ces exigences, nous penchons plus pour la seconde. En effet, l'usage de deux langages reposent sur des mises en correspondance entre les résultats de chacune des technologies utilisées, et ces correspondances sont soit impossibles soit trop coûteuses en performance. Une bonne méthode pour étendre les langages existants aux aspects relatifs aux M-Traces passe forcement par la définition d'une sémantique opérationnelle qui doit être conforme à notre sémantique déclarative. Nous pensons évidemment aux langages ayant des sémantiques formelles bien établies. Certains des langages existants peuvent servir à réifier les M-Traces mais nécessitent une spécification rigoureuse et claire de la manière de réifier les propriétés des M-Traces et les langages associés.

Troisième partie

Cas d'application et Implémentation

Chapitre 6

Exploitation des Systèmes à Base de Traces modélisées pour les EIAH : Cas d'applications

Sommaire

Ce chapitre décrit deux utilisations concrètes de la notion de systèmes à base de traces modélisées. Le premier usage décrit l'application des SBT dans le cadre d'une plateforme d'apprentissage collaborative nommée *eMédiatheque*. La seconde illustration concerne l'exploitation des SBT dans une plateforme d'apprentissage par le jeu nommée le *donjon pédagogique*.

6.1 Introduction

Pour montrer des usages de systèmes base à de M-Traces au service d'EIAH, ce chapitre présente deux cas d'utilisation. Ces cas d'application sont des concrétisations du cadre conceptuel de cette thèse et montrent la mise en place de l'architecture des SBT en action, au sein de deux EIAH et pour des objectifs différents. Les cas d'application présentés illustrent de manière informelle la logique d'exploitation des SBT dans un certain nombre de scénarios d'application possibles, sans se soucier de l'objectif initial de l'observation mise en place.

Nous allons présenter deux cas d'utilisation des SBT : le premier est orienté vers l'utilisation des SBT pour les apprenants et décrit l'usage des traces pour mettre en place une réflexivité des activités qui leur soit destinée. Le second cas est orienté vers l'usage des SBT pour l'enseignant et décrit la personnalisation des informations contextuelles sur les apprenants par l'enseignant. L'objectif de ce chapitre est d'exemplifier l'usage concret de notre cadre conceptuel pour concevoir et implémenter des systèmes permettant la manipulation de M-Traces.

6.2 L'exploitation des traces dans la plateforme eMédiathèque

6.2.1 Présentation de la plateforme

La plateforme eMédiathèque [SPMM09, CJM07] est un environnement d'apprentissage synchrone permettant la collaboration via internet développé par la société eLycée [66] pour mettre en application le concept de classe virtuelle. La platerforme eMédiathèque a été développée pour permettre à des apprenants de travailler ensemble en temps réel, comme s'ils étaient dans une vraie salle de classe, avec un accès partagé aux outils et aux ressources en ligne comme des tableaux blancs partagés, navigateur partagés, chat, vidéo-conférence, lecteurs multimédia synchrones, etc.

Comme le montre la figure 6.1, les différents outils se présentent dans des éditeurs dans la zone centrale (le rectangle bleu), ou en tant que vues périphériques. Les vues et les éditeurs peuvent être empilés, mis en cascades ou être bougés librement par l'utilisateur, ce qui donne une flexibilité maximum, et beaucoup de possibilités. Par exemple, il est possible de mener des activités avec plusieurs documents et outils en même temps, commenter une vidéo sur un tableau blanc ou comparer deux tableaux blancs. À part le chat (global pour toute la classe), chaque outil peut être instancié à différents moments : une classe peut avoir autant de tableaux blancs ou de co-lecture rapide qu'il est nécessaire. Une pratique commune est que chaque apprenant ou binômes a son propre tableau blanc, représentant son cahier de brouillon personnel, en plus du tableau blanc de l'enseignant qui représente le tableau de la salle de classe. En outre, eMédiathèque fournit un service de vidéo-conférence multi-utilisateur, en se basant sur les services de Marratech [67], intégrée et visuellement présentée à l'utilisateur dans son ensemble.

66. Elycée (http ://www.elycee.com/) est une jeune et innovatrice compagnie proposant des services en ligne d'apprentissage de la langue et la culture française aux apprenants francophones expatriés.

67. http ://www.marratech.com

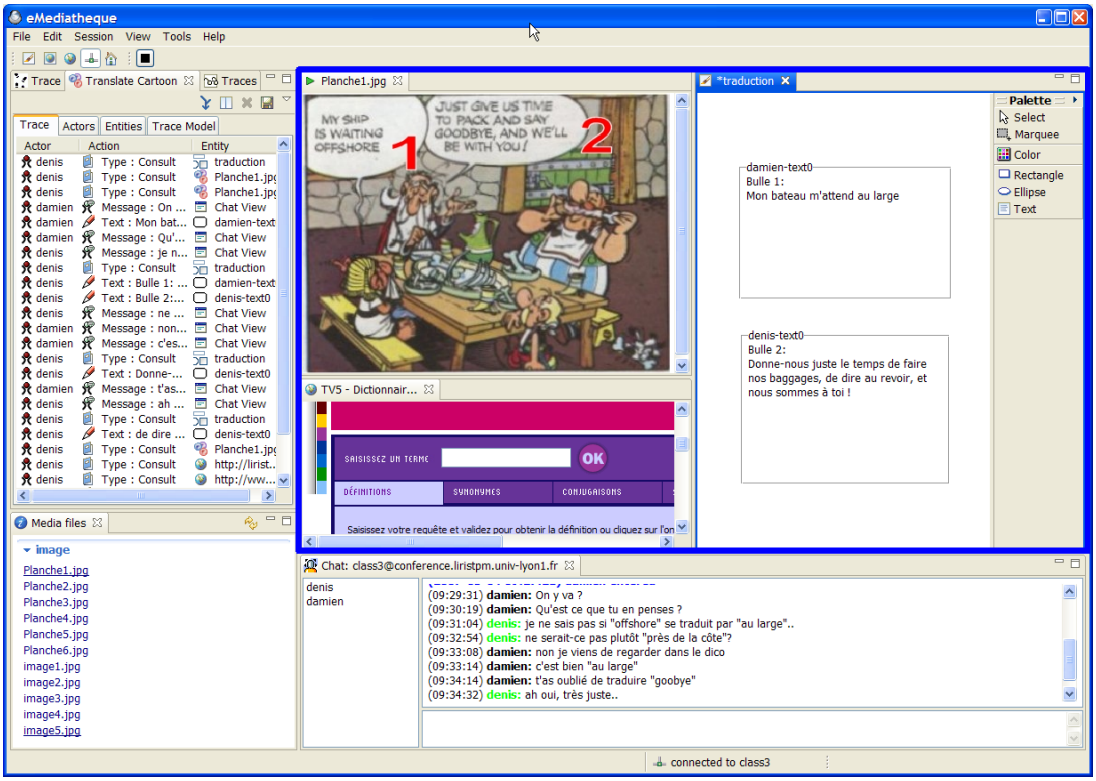

FIGURE 6.1 – Une copie d'écran du client de la plateforme eMédiatheque

6.2.2 Système à Base de Traces modélisées dans eMédiathèque

Dans n'importe quel EIAH, il est crucial que les utilisateurs se rendent compte des différents aspects liés à leurs activités individuelles et collaboratives. Plus particulièrement dans eMédiathèque où l'activité des apprenants doit être adaptée en mettant en œuvre des personnalisations (réflexivité, awareness, etc.) dans le contexte de différentes situations pédagogiques. Cependant, eMédiathèque est une plate-forme neutre, qui n'implémente pas des activités pédagogiques prédéfinies. Le contenu pédagogique est contrôlé par un système de gestion de cours (Course Management System), semblable à Moodle [68], en fournissant via des services Web des pages web et des dossiers de médias préchargés.

Cependant, la réflexivité des activités des apprenants exige la mise en place de mécanismes de personnalisation dépendant d'activités pédagogiques pas encore spécifiées (ou définies) au moment de la conception de la plateforme. Pour permettre la personnalisation de diverses activités qui peuvent être mises en place avec eMédiathèque, sa conception a été fondée sur la notion de trace. En la couplant à un SBT, eMédiathèque permet de personnaliser les sessions des apprenants grâce à des traces interprétées qui prennent sens pour ces apprenants selon le contexte de leurs activités spécifiques. Figure 6.2 montre une vue globale de l'architecture des visualisations personnalisées et interactives des traces.

68. http ://moodle.org/

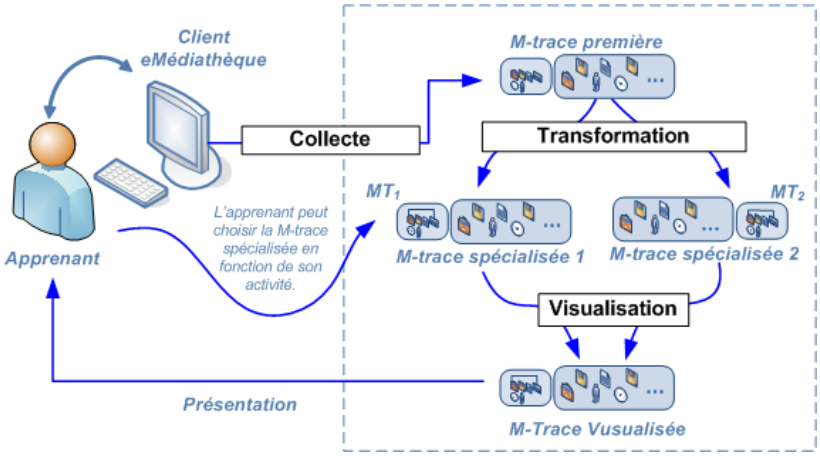

FIGURE 6.2 – L'architecture de la visualisation temps réel et interactive basée sur un SBT dans eMédiatheque.

6.2.3 Collecte des M-Traces dans eMédiathèque : la M-Trace première

L'EIAH eMédiathèque a été conçu pour tracer la plupart des actions de l'utilisateur et a été pensé dès le début autour de notion de trace modélisée. Contrairement aux traces de très de bas niveau (e.g. frappes clavier et cliques de souris) habituellement obtenues dans certains EIAH, la granularité des observés dans eMédiathèque est beaucoup plus pertinente et permet souvent d'obtenir des observés ayant une sémantique de plus haut niveau fournissant la réflexivité des activités.

La figure 6.3 montre le modèle de trace général d'eMédiathèque. Une trace est une liste d'actions observables temporellement estampillées (type `ObservableAction`). Chaque action dispose d'une simple identification et un attribut `shared` indiquant si l'action est partagée ou pas. Chaque action a un acteur (type `User`) et est censée s'effectuer sur une entité de l'environnement (de type `ObservableEntity`). Ce dernier point est une restriction de notre modélisation pour faciliter la visualisation des M-Traces. Chaque entité a également un attribut `shared` indiquant si cette entité est accessible ou pas à tous les utilisateurs. Chaque M-Trace première traitée dans eMédiathèque dispose de ce modèle de trace ou d'une version restreinte et/ou spécialisée (i.e. définissant des sous-types et des sous-relations des types existant).

Cependant, eMédiathèque est un ensemble d'outils qui peuvent être opportunément employés et adaptés pour diverses activités spécifiques. En ce qui concerne la modélisation de trace, eMédiathèque a été fondée sur deux types de modèles différents :

– le modèle de la M-Trace première, créé par le concepteur du système, représentant l'historique des actions effectuées par l'utilisateur. C'est un modèle de trace est orienté « outil » puisque sa sémantique décrit seulement l'utilisation « objective » de la plateforme, indépendamment de l'activité de l'utilisateur. La figure 6.3 à droite montre une représentation sous forme d'arbre d'un fragment de la trace première produite par eMédiathèque [69] Les

69. Évidemment, cette visualisation n'est pas censée être présentée aux apprenants. Cependant, elle donne une

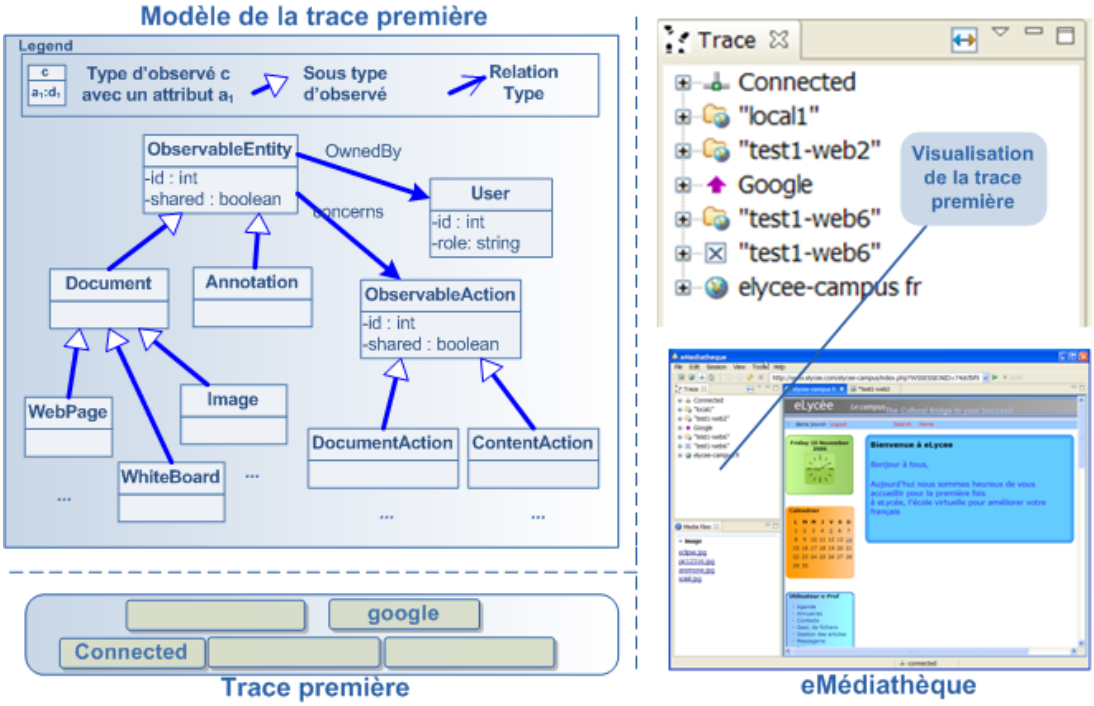

FIGURE 6.3 – Visualisation de la trace première dans eMédiathèque et son modèle

nœuds du premier niveau sont des actions où les icônes représentent les type d'observé et les labels identifient l'objet principal manipulé par l'action.

– un modèle de la M-Trace transformée est un modèle plus abstrait et inclut la sémantique de l'activité pédagogique.

La visualisation personnalisée des M-Traces dans eMédiathèque est un processus de transformation à deux étapes montrées dans la figure 6.4. La première étape (spécifiée par un concepteur d'activité) est la transformation de la M-Trace première en trace spécialisée qui fait sens pour l'apprenant dans le cadre de son activité pédagogique (transformation statique). La seconde étape est faite par l'apprenant lui-même, et consiste à transformer interactivement la trace spécialisée en représentation graphique (transformation dynamique).

6.2.4 Transformation de M-Traces dans eMédiathèque

Pour valider notre approche SBT, nous avons expérimenté le concept de transformation de M-Traces dans le cadre d'un exemple simple d'une activité pédagogique menée avec eMédiathèque [SPMM09, CJM07]. L'activité expérimentée est une traduction de bande dessinée où les apprenants sont organisés en groupe de deux. Dans chaque groupe, les apprenants collaborent pour traduire un dessin animé de l'anglais vers le français. Ils écrivent la traduction qu'ils proposent dans un document partagé (un tableau blanc) et peuvent communiquer par chat pour organiser l'activité et négocier la traduction.

bonne vue d'ensemble du genre d'informations trouvées dans la M-Trace première.

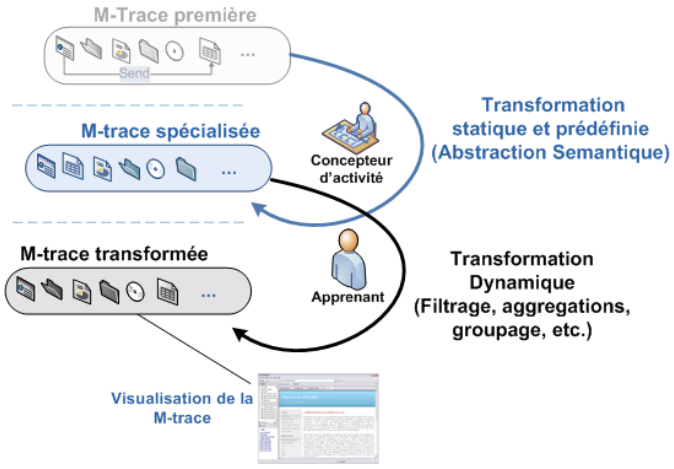

FIGURE 6.4 – Cycle des transformations des M-traces dans eMédiathèque.

La figure 6.5 présente le modèle de trace correspondant à cette activité. Ce modèle de trace décrit quatre types d'action et cinq types d'entité qui sont significatifs dans le cadre de cette activité, e.g., lire une ressource, envoyer un message dans le chat, éditer un texte dans une bulle et rechercher un mot dans un dictionnaire.

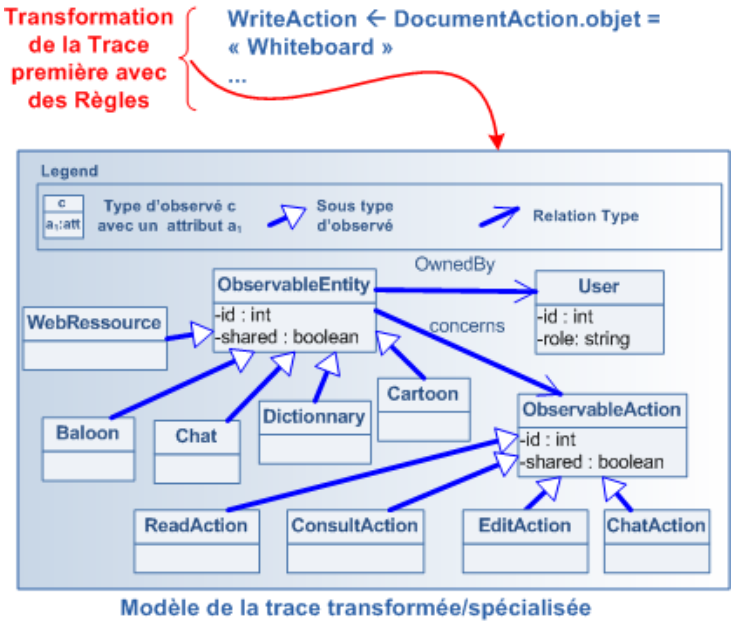

FIGURE 6.5 – Modèle de la trace transformée correspondant à l'activité de traduction de bande déssinée.

Pour obtenir une trace conforme à ce modèle, des règles de transformation sont appliquées à la trace première. Ces règles sont décrites par un « concepteur d'activité » qui a les qualifications requises pour modéliser l'activité pédagogique. Il spécifie et met en application le modèle spécialisé de la trace de l'activité pédagogique considérée (i.e. traduction de dessin animé) et des

règles de transformation dans le modules SBT d'eMédiathèque. Ces tâches sont faites en différé comme une préparation d'une session d'apprentissage. Une fois que la session est lancée, les règles de transformation sont fixées et appliquées automatiquement. À l'exécution, quand une nouvelle action apparaît dans la trace première, elle est détectée par la règle de transformation, qui régénère la trace spécialisée. Par exemple, les règles de transformation transformeront tous les objets de type `image` contenant le dessin animé original en une entité de type `dessin-animé`. La figure 6.5 décrit par exemple une règle pour que chaque action de type `DocumentAction` impliquant un tableau blanc produit une entité `WriteAction` dans la trace transformée.

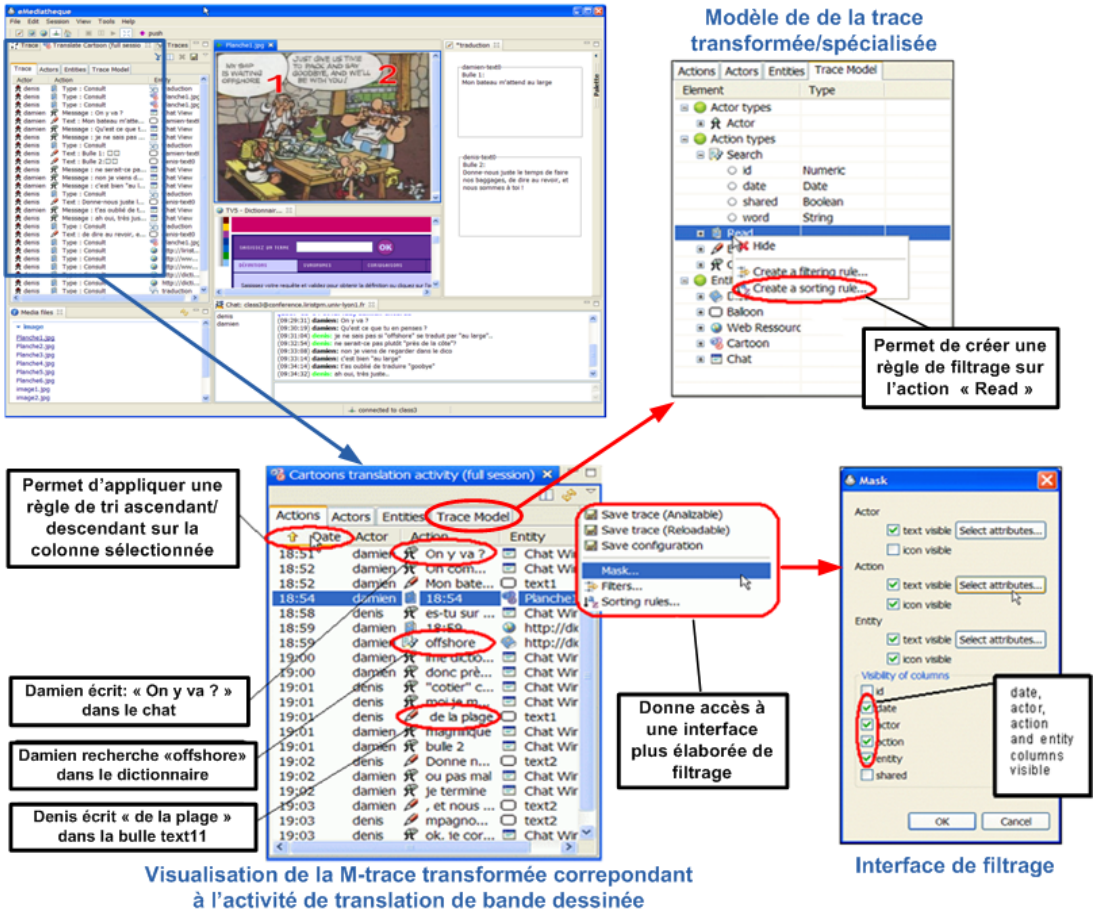

FIGURE 6.6 – Interfaces de visualisation du modèle et trace transformée correspondant à l'activité de traduction de bande dessinée.

La figure 6.6 montre une visualisation de la M-Trace transformée obtenue à partir de la M-Trace première. Pour l'apprenant, cette M-Trace est plus significative et fait sens dans le cadre de son activité pédagogique (i.e. traduction de dessin animé). L'apprenant peut appliquer des transformations secondaires la trace spécialisée (cf. figure 6.6) de manière interactive en exprimant des règles de filtrage, de réécriture, de fusion, etc. Ces transformations peuvent être exprimées directement sur les éléments du modèle de la M-Trace transformée (cf. en haut à droite de la figure 6.6). L'apprenant peut également filtrer la trace visualisée via une interface

de filtrage (cf. en bas à droite de la figure 6.6).

Chacune des transformations est décrite dans l'API [70] du module SBT. L'interface de visualisation recalcule la trace vue par l'apprenant toutes les fois qu'un changement induit par la transformation est appliqué, ou toutes les fois que la trace spécialisée est modifiée par les premières transformations. La trace présentée est très volatile et représente une transcription graphique et personnalisée de la trace spécialisée, pouvant être transformée par le système et/ou l'apprenant.

6.3 L'exploitation des traces dans la plateforme « donjon pédagogique »

L'observation dans eMédiathèque est davantage orientée vers l'apprenant que vers l'enseignant. L'exemple suivant de SBT, présenté dans la suite, est plus orienté vers le tuteur. Cette section décrit l'observation de ce qui se produit dans une plateforme d'apprentissage par le jeu nommée « donjon pédagogique »[CMH08]. Dans cette plateforme, des informations contextuelles sont fournies grâce à des transformations spécifiques sur les traces des apprenants. Comme pour eMédiathèque, nous commençons par décrire la plateforme du donjon pédagogique, puis nous détaillons son module SBT et les exploitations proposées.

6.3.1 Présentation de la plateforme

Le « donjon pédagogique » [CMH08] est une plateforme d'apprentissage par le jeu (Learning Game) permettant un certains nombre d'interactions coopératives (voir [DBBO96] pour la liste des services de coopération). Cette plateforme a été développé par l'équipe Observation [71] (laboratoire SYSCOM) à l'université de Chambéry, avec le postulat que la dimension sociale est cruciale pour les processus cognitifs impliqués dans l'activité d'apprentissage. Le donjon pédagogique a été conçu de manière à réifier la métaphore d'exploration d'un monde virtuel, un donjon, dans lequel chaque apprenant rassemble les connaissances liées à une activité d'apprentissage, i.e., l'activité d'acquérir la connaissance pendant une session d'apprentissage est semblable à l'exploration d'un donjon.

Le donjon représente l'endroit où la session d'une activité d'apprentissage a lieu, e.g., la figure 6.7 montre une salle du donjon pédagogique où deux apprenants et un enseignant se retrouvent dans le cadre d'une activité pédagogique spécifique. La figure montre également des ressources pédagogiques et un portail de téléportation permettant aux apprenants de se déplacer vers de nouvelles salles. La plateforme permet de créer un donjon particulier consacré à une activité d'apprentissage particulière, pour un sujet particulier. Chaque pièce du donjon représente l'endroit où une activité (secondaire) donnée peut être exercée. Dans le donjon pédagogique, plusieurs acteurs (des apprenants ou des enseignants) peuvent se déplacer, exécutant

70. Application Programming Interface
71. http ://www.syscom.univ-savoie.fr/equipes/presentation/ ?equipe=7

dans un certain ordre des sous activités afin d'acquérir la connaissance. Des activités peuvent être effectuées d'une manière individuelle ou collaborative. Les apprenants peuvent accéder à la connaissance par des documents, par l'intermédiaire d'une aide des enseignants ou en travaillant avec d'autres apprenants.

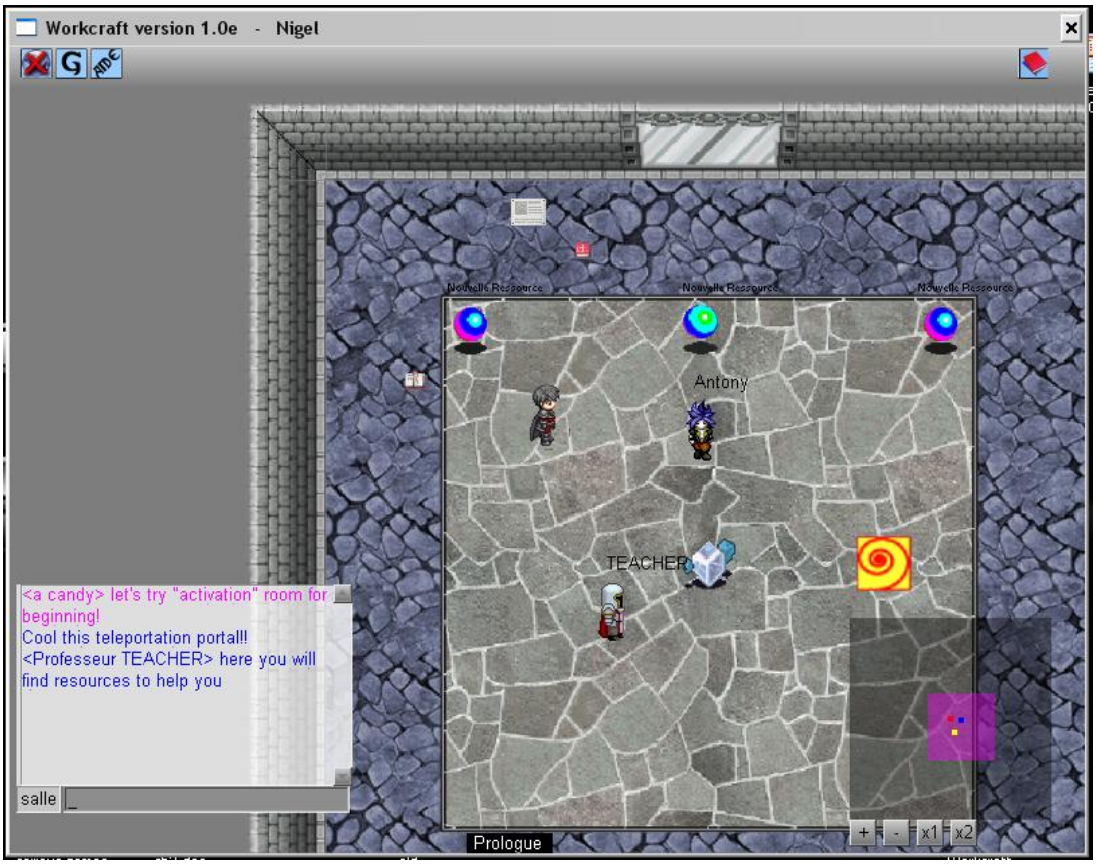

FIGURE 6.7 – Copie d'écran de l'interface pour l'apprenant du donjon pédagogique.

Le donjon peut être flexible. Par exemple, les « portails de téléportation » peuvent mener à de nouvelles salles créées dynamiquement. Chaque pièce est consacrée à une activité dans laquelle des ressources explicatives peuvent être trouvées telles que des textes, des liens, des vidéos, etc. Ces ressources fournissent à l'apprenant toutes les informations utiles. L'apprenant atteint l'objectif local d'une activité s'il répond avec succès à un questionnaire situé dans la salle. Dans la figure 6.7, nous pouvons voir un exemple d'une salle dans le donjon où l'apprenant peut se déplacer et rencontrer d'autres apprenants ou enseignants impliqués dans même session. Le fait qu'un apprenant se retrouve dans la même salle qu'un autre apprenant signifie qu'ils exercent la même activité.

La topologie du donjon représente le scénario global de la session de l'activité d'apprentissage, i.e., l'ordonnancement entre les activités. Il y a autant de salles que d'activités réelles, et ces salles sont liées par des couloirs. Un exemple d'un scénario vu comme une topologie du donjon est présenté dans la figure 6.8. La construction d'une topologie d'un scénario par l'enseignant se

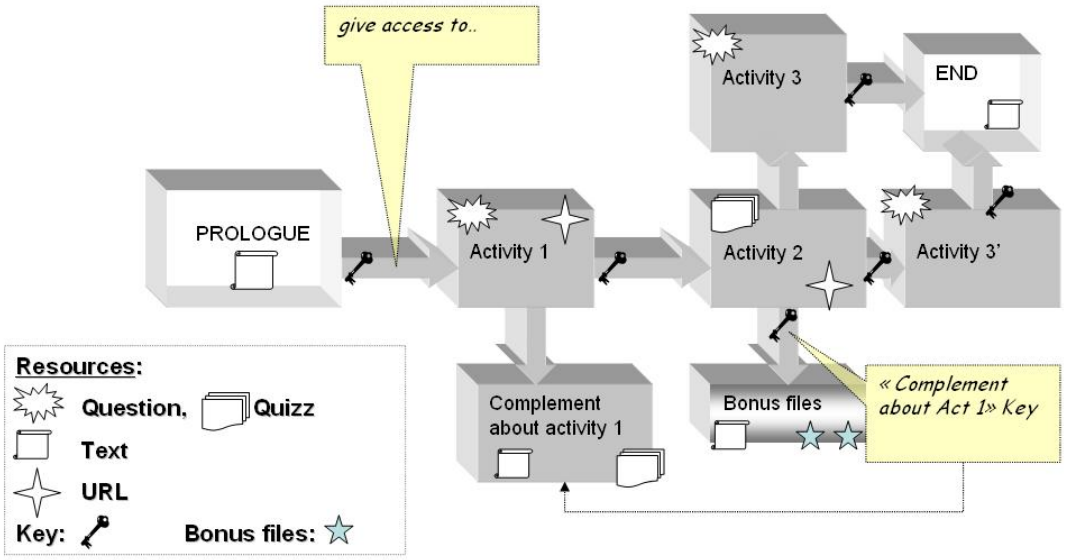

FIGURE 6.8 – Un exemple de scénario vu comme une topologie de donjons ([CMH08])

fait par un constructeur de session qui permet la définition :

1. des différents types d'activités, e.g., une activité peut être collaborative ;
2. des descriptions de ressources disponibles pour chaque activité. Les ressources (texte, graphiques ou vidéos) sont des ressources locales ou des liens vers un matériel pédagogique en ligne ;
3. des contraintes sur les activités (liens temporels et l'organisation logique) ;
4. des procédés de validation pour chaque activité, e.g., l'obtention d'une clé synonyme de la réussite dans une activité. L'enseignant peut choisir comment évaluer chacune des activités prédéfinies. La manière la plus simple d'obtenir une clé est juste de lire un texte. Mais, le plus souvent, l'apprenant doit répondre à une question ou questionnaire (choix multiple, ouvert, etc.).

Avec le donjon pédagogique, l'enseignant peut également mettre en place des activités de collaboration. Dans ce cas, les salles associées à ces activités sont les endroits de collaboration. Actuellement, un service de chat est fourni dans les salles du donjon, mais d'autres outils de collaboration peuvent disponibles (forum, espaces partagé, etc.). Si l'enseignant met en place une activité collaborative pendant une session d'apprentissage, il doit définir des équipes d'apprenants. Les apprenants appartenant à la même équipe sont censés effectuer des activités collaboratives ensemble. Dans les salles collaboratives, les questionnaires sont également collaboratifs. Les apprenants dans la même équipe doivent tous être présents dans la salle et doivent échanger par l'intermédiaire du chat avant de répondre à la question. Comme dans une salle de classe traditionnelle, un apprenant peut également collaborer avec un enseignant, par exemple quand il a besoin d'une aide ou assistance.

6.3.2 Système à Base de traces modélisées dans la plateforme « donjon pédagogique »

Les enseignants utilisant des EIAH ont besoin de moyens pour suivre de façon précise la progression de l'activité d'apprentissage des apprenants et éviter le manque de feedbacks et de personnalisation. Pour permettre cela pour des enseignants et des apprenants, le donjon pédagogique a été équipé d'un système à base de traces modélisées.

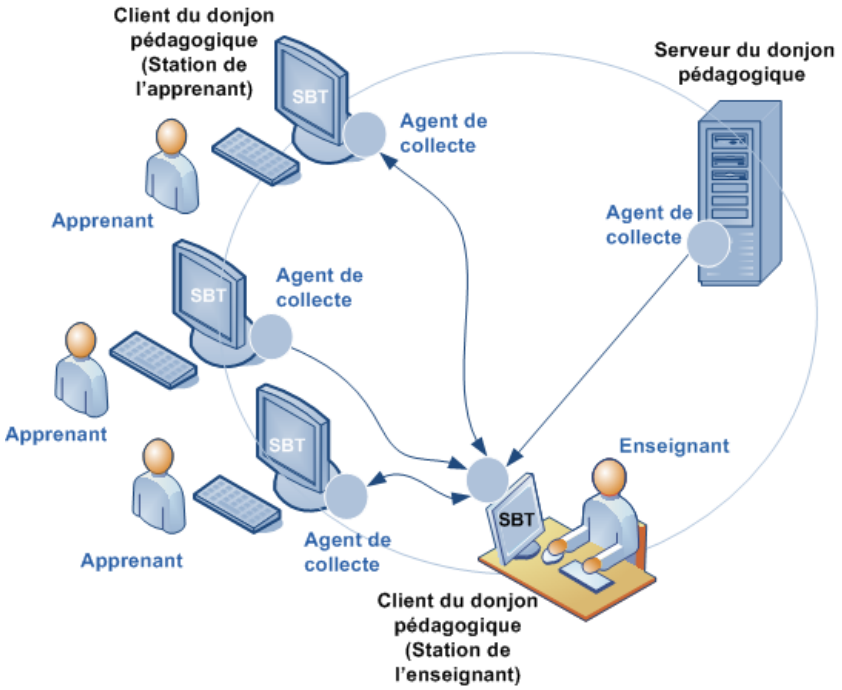

FIGURE 6.9 – L'architecture du SBT dans le donjon pédagogique [CMH08]

Le donjon pédagogique a été développé avec la possibilité de permettre des adaptations en fonction des traces des apprenants. Comme mentionné précédemment, l'enseignant est habitué à percevoir des informations informelles dans une vraie salle de classe. Il peut observer un comportement ou une situation spécifique et fournir à l'apprenant les retours prévus sur l'activité globale. Dans le cas du donjon pédagogique, ces retours sont mis en œuvre en tirant profit des traces collectées. Ces retours peuvent être des indicateurs calculés à partir des traces et montrés à l'apprenant pour lui permettre de prendre conscience de la situation dans laquelle il évolue ou bien des explications/adaptations des activités à effectuer. Pour permettre cela, le donjon pédagogique se base sur son module SBT. Ceci permet d'offrir la possibilité à l'enseignant de réagir, par exemple, en envoyant des messages à certains apprenants, en fournissant des ressources d'aide. L'enseignant peut même adapter la session d'apprentissage en ajoutant dynamiquement une partie du donjon pour fournir de nouvelles activités.

D'un point de vue technique, le module SBT du donjon pédagogique est réalisé avec une

technologies multi-agent [72]. Le donjon pédagogique et le SBT sont décrits dans la figure 6.9. L'agent de la station enseignant permet de configurer la collecte de n'importe quel station apprenant en spécifiant les observés des traces des activités qui doivent lui être envoyés. Toutes ces traces peuvent être interprétées et présentées à l'enseignant comme nous allons le montrer dans la suite.

6.3.3 Collecte de M-Traces dans le donjon pédagogique

Comme eMédiathèque, la totalité des modules du donjon pédagogique est équipée pour être observable. Le donjon pédagogique a été conçu pour observer les activités des apprenants de la manière la plus générique possible, indépendamment des scénarios ou des activités d'apprentissage qui peuvent être spécifiées. Cette idée est mise en œuvre en permettant l'observation des différents modules du donjon pédagogique au moyen de *sondes* ou *capteurs logiciel*.

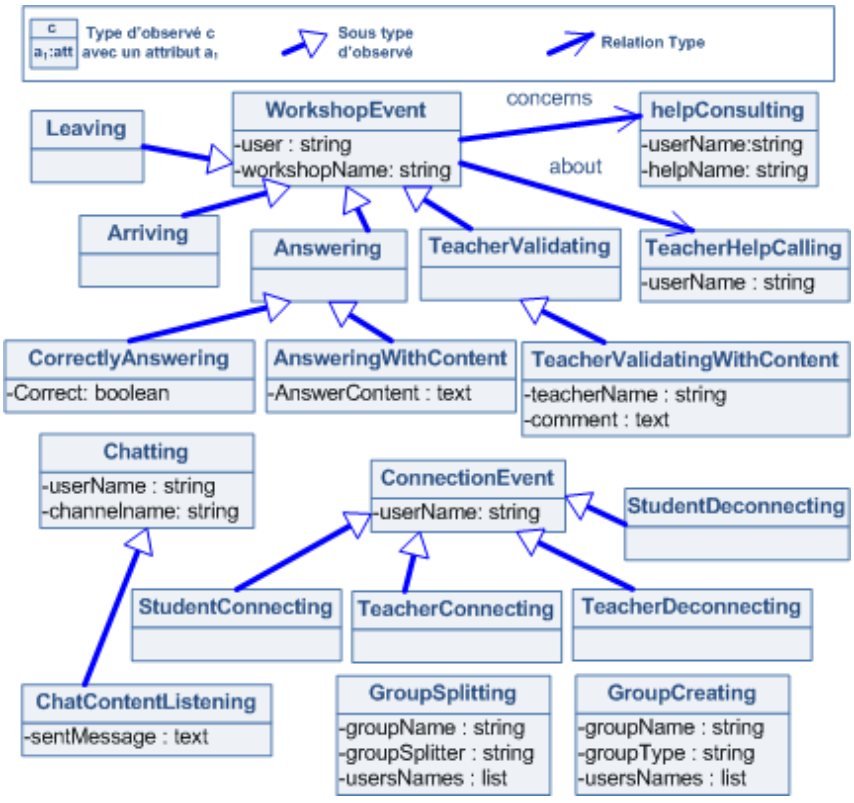

FIGURE 6.10 – Modèle de la trace première du donjon pédagogique

Par exemple, dans le donjon, les actions telles que « entrer dans une salle », « répondre correctement à un questionnaire », « chat » ou « aide consultée » peuvent être tracées conformément au modèle général de la M-Trace première décrit dans la figure 6.10. Chaque type observé du

72. En fait, ceci est justifié par le fait que les informations importantes doivent être collectées du poste de travail de chaque apprenant et envoyer par la suite à la station de l'enseignant (voir [CMHF06] pour plus de détails).

modèle de la trace première est techniquement instancié par une sonde [73]. Cependant, comme le montre la visualisation de la figure 6.11, les traces premières posent deux problèmes majeurs :

- *quantité* : il y a un grand nombre d'information (parfois inutile) venant de tous les clients et des filtrages sont nécessaires pour pouvoir les exploiter. En effet, dans le donjon pédagogique, beaucoup d'apprenants jouent simultanément et toutes ces sondes créent un nombre important d'observés.
- *qualité* : ces sondes fournissent des informations qui sont de trop bas niveau, pas très significatives, souvent difficile à comprendre. C'est pourquoi l'utilisateur (en l'occurence l'enseignant) doit exprimer des traitements spécifiques pour transformer ces observés en une information plus compréhensible.

Le traitement des M-Traces pour obtenir des informations pertinentes des activités des apprenants se fait moyennant des requêtes et des transformations. Le module de SBT du donjon pédagogique donne la possibilité d'exprimer des patterns complexes et des transformations pour traiter la trace première.

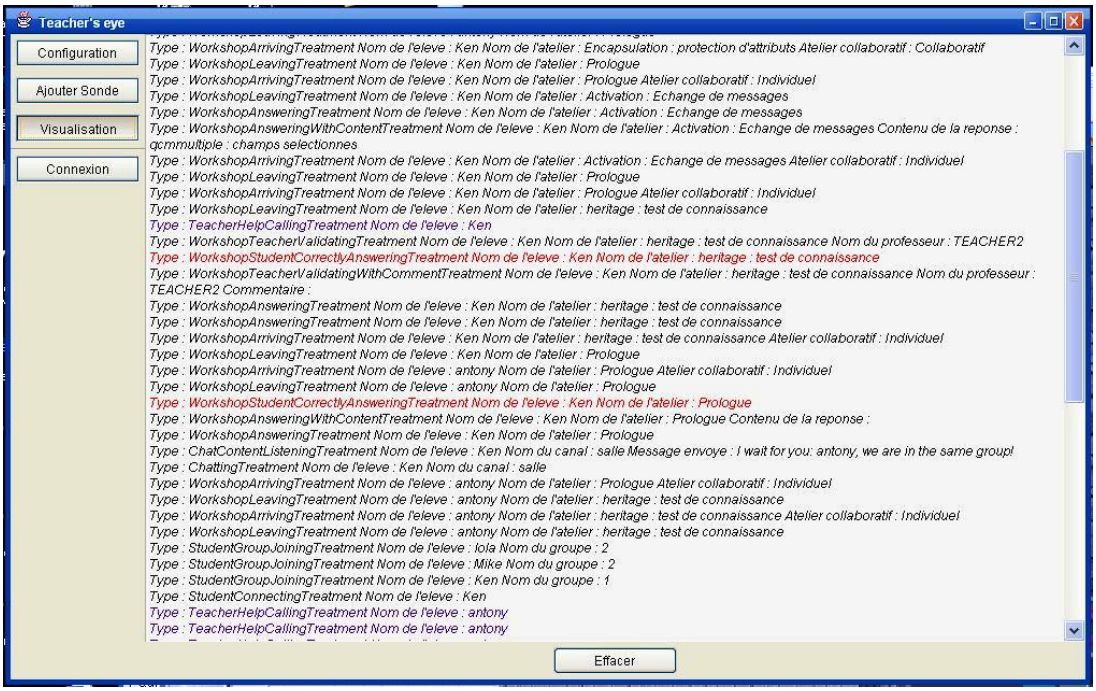

FIGURE 6.11 – Visualisation de trace première dans la plateforme donjon pédagogique

6.3.4 Interrogation et transformation de M-Traces dans le donjon pédagogique

La trace première présentée (cf. figure 6.11) fournit aux utilisateurs les informations élémentaires et est ainsi utile pour observer des faits de base. Cependant, elle n'est pas pertinente

73. D'un point de vue technique, les sondes se comportent comme des capteurs logiciel qui envoient sans interruption au SBT des données observées de chaque client.

quand le niveau de l'abstraction requis est plus élevé. Par exemple, disposer d'un observé statuant qu'un apprenant a consulté une ressource d'aide n'est pas forcément très significatif ou de première importance. Cependant, si la même observation se produit juste après qu'il ait donné une réponse fausse et que ceci ait été suivi d'une bonne réponse dans la même activité, l'enseignant peut être rassuré quant à l'utilité de l'aide proposée. Cet observé peut être déduit et obtenu en exprimant un pattern complexe permettant de fournir une explication de plus haut niveau au sujet de l'activité en cours.

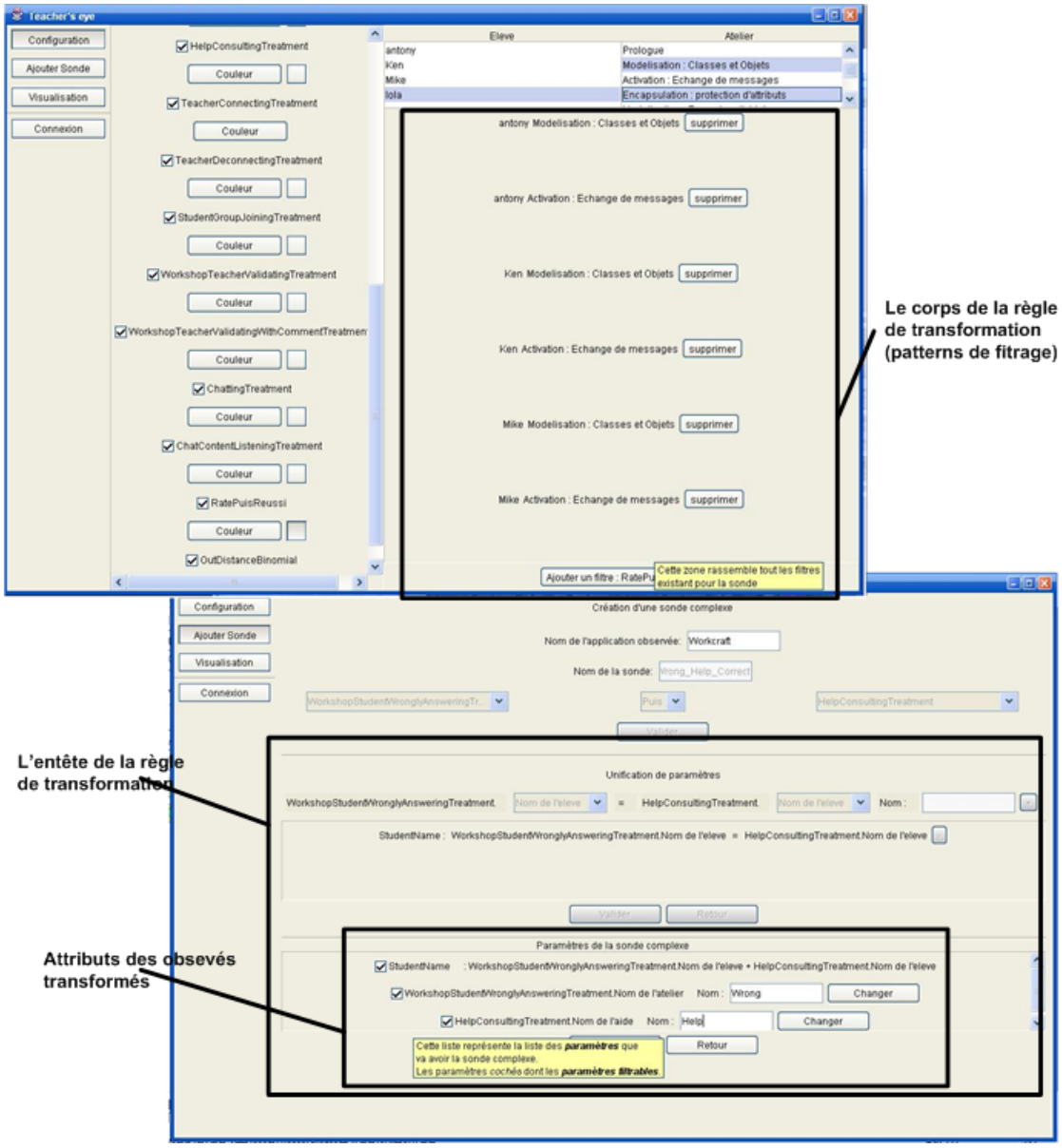

FIGURE 6.12 – Interfaces permettant d'exprimer l'entête et le corps des règles de transformation

Le module SBT du le donjon pédagogique permet d'extraire des observés de la trace première (voir la figure 6.10) en appliquant des patterns et des règles de transformation. La figure 6.12

montre un exemple de définition d'une telle règle de transformation. Les nouveaux observés issus de la transformation sont disponibles dans la plateforme, avec les propriétés inférées de la trace première. En particulier, les nouveaux types construits peuvent être réutilisés pour créer des patterns et des règles plus complexes. Grâce aux règles, l'ensemble des observés et leurs types augmente naturellement, guidés par les besoins d'observation de la plateforme et les usages des utilisateurs (notamment les enseignants). L'observation peut ainsi être finement améliorée en combinant différents types observé dans des patterns s'appliquant sur la M-Trace première.

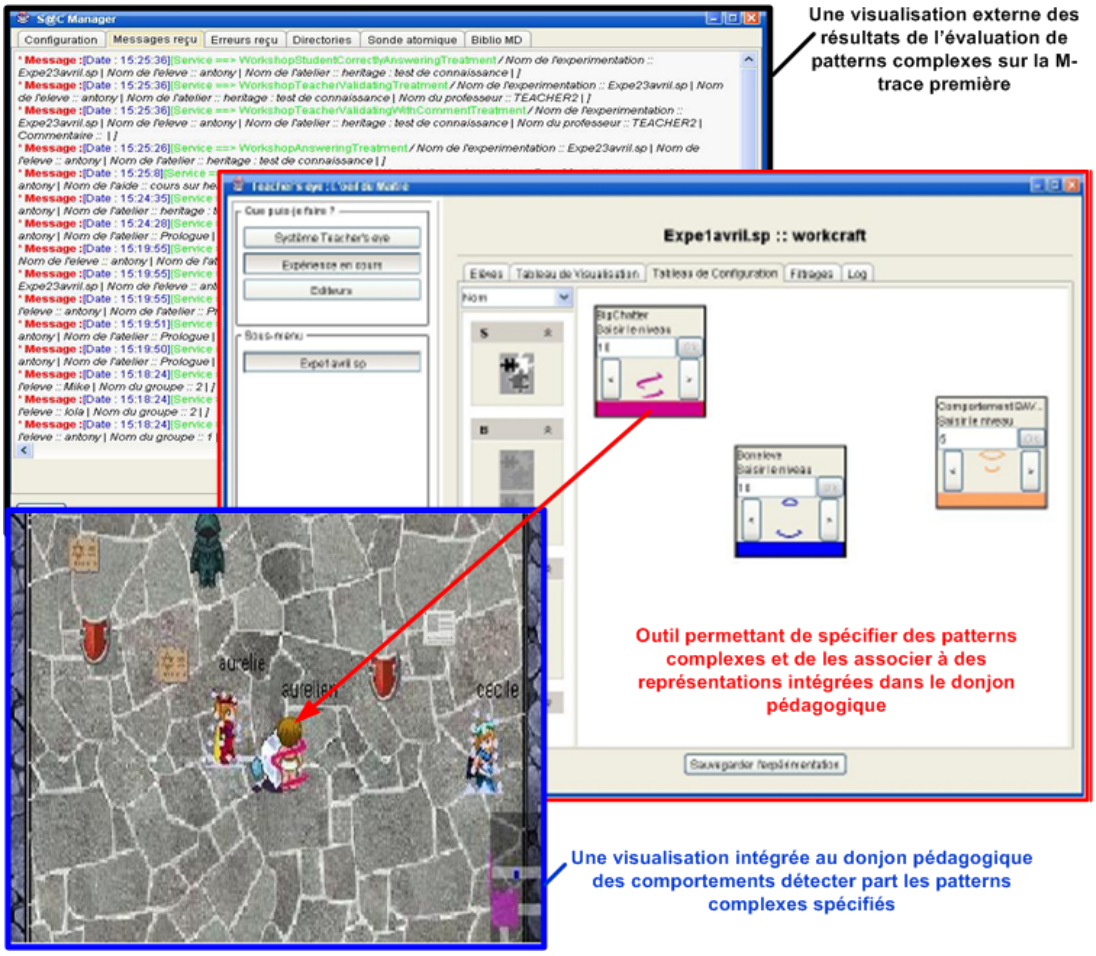

FIGURE 6.13 – Intefaces pour la spécification de patterns de comportements, et leurs visualisations intégrées.

Des patterns plus complexes peuvent être spécifiés pour détecter des types de comportement spécifiques pendant une session d'apprentissage tels que brillant, coopératif, laborieux, bavard, rigoureux, etc. En plus de la visualisation des patterns évalués en temps réel de l'activité (cf. figure 6.13 (Cadre noir)), le donjon pédagogique permet également une représentation intégrée des éléments déduit par la transformation. Le donjon pédagogique permet la détection de certains de ces types de comportement en évaluant des patterns spécifiques sur les traces premières. L'enseignant doit employer une interface externe pour définir les patterns de comportement

(figure 6.13 (cadre rouge)). La figure 6.13 (cadre bleu) montre deux auras autour des avatars des apprenants. Par exemple, la neige tombant sur les deux filles indique qu'elles sont en difficulté et une spirale tournant autour du garçon signifie qu'il est bavard. L'usage du SBT permet ainsi de soutenir une observation personnalisée puisque l'enseignant peut décider de définir certains attributs, puis activer ou désactiver dynamiquement les auras définis selon ses besoins pédagogiques. Le niveau du déclenchement peut également être ajusté à tout moment. Cette visualisation d'aura est intégré directement dans le jeu et fournit une information pertinente aux apprenants et à l'enseignant.

6.4 Discussions et limites des exemples présentés

Les deux cas d'application montrés ne sont pas les seuls exemples de l'utilisation du cadre conceptuel des SBT que nous avons défini dans le chapitre 3. Plusieurs applications de l'approche ont été menées et des Systèmes à Base de Traces modélisées ont été implémentés dans des situations d'activité variées. C'est le cas par exemple du travail de Georgeon [Geo08] sur la mise en place d'un SBT nommé ABSTRACT destiné à l'analyse des comportements de conducteur d'automobile. Le système, mis en place à l'INRETS [74], gère des sources de données traitées pour collecter des traces d'interactions : vidéos des différentes caméras du véhicule, mesures prélevées sur le véhicule, informations sur l'environnement provenant d'un télémètre, données de navigation d'un GPS et les évènements déclenchés par l'expérimentateur.

C'est le cas également dans des projets liés à d'autre EIAH, comme le travail mené dans la cadre de l'EIAH Moodle [75] pour mettre en place un SBT visant la mise en place d'indicateurs d'activités éducatives [DSP+10]. Dans un domaine d'application tout à fait différent le logiciel Advene consacré à la lecture active de vidéos a été instrumenté d'un SBT visant à soutenir l'utilisateur dans la reprise d'activité après une interruption [RPC08].

Dans tous ces exemples d'applications de notre approche, les chercheurs ont dû implémenter de manière *ad-hoc* un SBT au sein de l'environnement concerné par les traces (système observé), le plus souvent en étant contraint de se limiter à l'instrumentation d'un outil unique. La majorité des SBT développés n'implémente qu'une spécialisation de notre méta-modèle (voir par exemple les modèles de trace première des deux exemples présentés). Les implémentations proposées sont loin d'être génériques et utilisent des langages qui ne permettent pas de prendre complètement les traitements pouvant être impliqués par les M-Traces (principalement SQL ou une variante de SQL).

Pour faciliter la mise en place des SBT et ainsi donner les moyens à notre approche de prendre une nouvelle ampleur, un projet de développement d'un framework informatique générique a vu le jour (Atelier pour la gestion de Traces, leurs Exploitations et Représentations - ATER). Il s'agissait de créer une implémentation générique de SBT (un framework) qui pourrait offrir des services de manipulation de traces modélisées nécessaires à l'implémentation d'un SBT :

74. http ://www.inrets.fr/
75. http ://moodle.org/

collecter, transformer, requêter, visualiser, gérer et de partager des traces modélisées. Dans le chapitre suivant, nous allons présenter le développement de la plateforme ATER.

Chapitre 7

Conception et Implémentation d'un Systèmes à base de M-Traces : La plateforme ATER

Sommaire

Dans ce chapitre, nous allons présenter le développement d'une implémentation d'un système à base de traces (SBT), baptisé ATER (Atelier pour la gestion des Trace, leurs Exploitations et Représentations). On commencera par présenter la modélisation de ce système dans le langage UML. Nous exposons ensuite son architecture technique et les différents choix technologiques utilisés. Enfin, nous discuterons les limites du prototype développé et les principales difficultés recontrées.

7.1 Introduction

L'objectif de ce chapitre est de présenter la conception et le développement d'un système à base de traces modélisées générique pouvant être utilisé dans différentes situations et pour de multiples objectifs (analyse *post-hoc*, découverte de connaissances *ad-hoc*, etc.). Contrairement aux cas d'application développés dans le chapitre précédent qui implémenté des modèles de traces spécifiques, la plateforme développée baptisée ATER [76] a été conçue pour être indépendante de toute application qui l'utilise et implémente le méta-modèle de représentation des M-Traces.

L'objectif premier de ce projet de développement est de définir un ensemble d'outils et de méthodes dans le but de faciliter la représentation et la manipulation des traces modélisées. Notre idée est de concevoir ATER comme un *framework*. Littéralement, « framework » signifie « cadre d'applications ». Il représente d'une certaine manière un espace de travail modulaire composé d'un ensemble de bibliothèques permettant le développement rapide d'applications. Un Framework fournit suffisamment de briques logicielles et impose suffisamment de rigueur pour pouvoir produire une application aboutie et facile à maintenir.

Pour ce faire, le développement de la plateforme ATER doit prendre en compte de nombreuses contraintes, qui n'ont pas été considérées dans les implémentations précédentes. Nous citons les principales d'entre elles, notamment les critères qui ont piloté nos développements :

Normaliser les développements : le framework ATER doit permettre de construire toutes les applications basées sur ce framework avec les mêmes technologies, les mêmes normes et les mêmes méthodes. Ceci est une contrainte forte du projet ATER.

Réutilisabilité : le framework ATER doit fournir un point de départ pour le développement des systèmes basés sur ce framework, ce qui permet d'avoir très rapidement une structure de base sur laquelle ces systèmes pourront se baser.

Extensibilité : le framework ATER doit offrir un comportement flexible et extensible, i.e., une application utilisant ce framework pourra facilement être étendue.

Dans la suite de ce chapitre, nous allons présenter le projet de développement de la plateforme ATER. Nous allons commencer par présenter une description non exhaustive de la conception de notre système. Dans la suite, nous présentons l'architecture technique, en évoquant les technologies utilisées pour son implémentation et les fonctionnalités développées. Enfin, après une brève description de l'historique des développements, nous dressons le bilan des réalisations et discutons les résultats obtenus.

7.2 Conception du système ATER

La conception de notre système s'appuie sur la modélisation UML (Unified Modeling Langage). UML est un langage de modélisation objet permettant l'expression et la communication

76. Atelier pour la gestion des Traces, leurs Exploitations et Représentions

de modèles en fournissant un ensemble de symboles (la notation) et de règles qui régissent l'assemblage de ces symboles (la syntaxe et la sémantique) [MG00]. Le processus de développement utilisé pour la modélisation, s'appuie sur une démarche itérative et incrémentale, guidée par les cas d'utilisation (besoins des utilisateurs) et centrée sur l'architecture logicielle de notre système (Figure 7.1). Ainsi, nous allons commencer d'abord par donner l'architecture générale de notre système. Par la suite nous allons déterminer les cas d'utilisation, en identifiant les différents acteurs et activités impliquées par notre application.

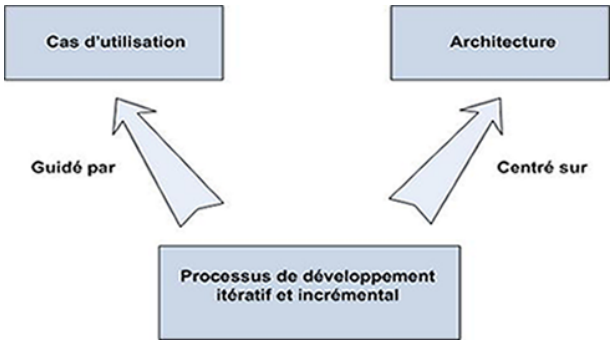

FIGURE 7.1 – Le démarche de développement du Système ATER

7.2.1 Architecture globale de la plateforme ATER

L'architecture de la plateforme ATER distingue différents types d'utilisation du système ATER, qui correspondent aux flèches 1-5 de la Figure 7.2 :

1. La collecte des traces à partir de sources de collecte (XML, texte, vidéo, etc.) avec un stockage sur le poste client. Le résultat de la collecte produit une nouvelle trace qui peut être traitée par le système ATER localement ou à distance. Ceci correspond par exemple au cas d'utilisation où chaque apprenant est tracé par l'EIAH utilisé localement et que l'enseignant peut se connecter à distance pour consulter cette trace.
2. L'utilisateur du système ATER peut gérer les traces, créer et gérer de nouveaux modèles et gérer les sources de collecte.
3. L'utilisateur du système ATER peut aussi visualiser une ou plusieurs traces. Il peut aussi visualiser les résultats d'une requête ou d'une transformation.
4. Un système informatique client du système ATER peut aussi stocker, interroger, transformer des traces et récupérer les résultats.
5. Le système ATER peut exporter et envoyer des M-Traces à une application tierce.

Enfin et comme pour tout système informatique, un administrateur est responsable de la gestion des utilisateurs et leurs droits d'accès. Dans ce qui suit, nous allons présenter les différents acteurs et cas d'utilisation impliqués par notre système.

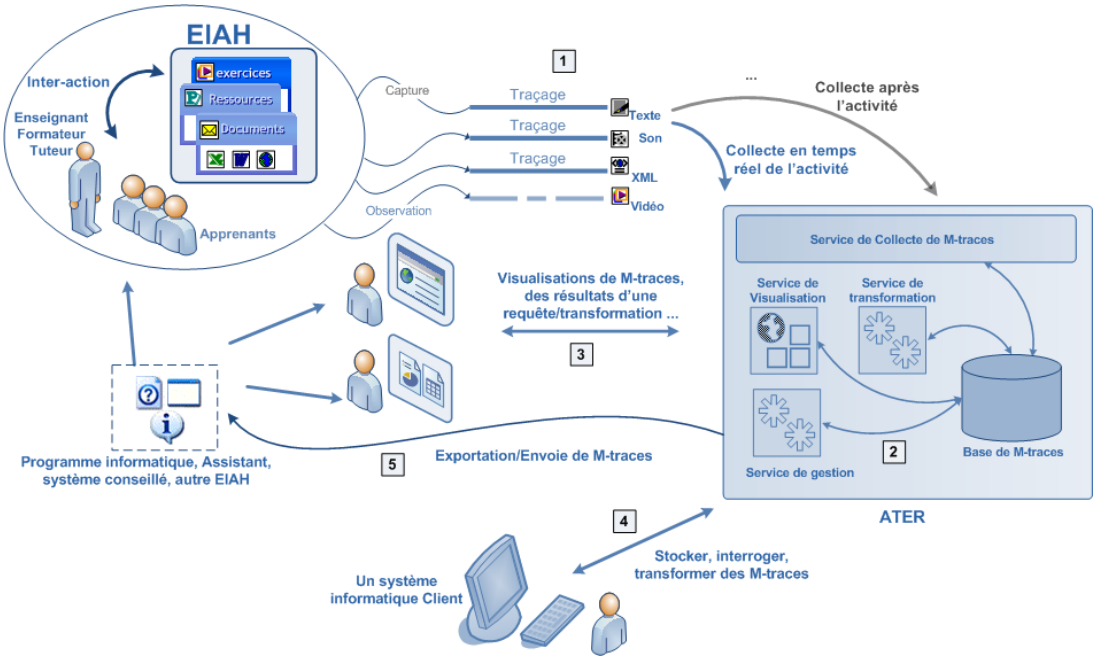

FIGURE 7.2 – Architecture globale du système ATER

7.2.2 Identification des acteurs

Un acteur représente l'abstraction d'un rôle joué par un utilisateur, un groupe d'utilisateurs ou tout simplement par toute autre entité externe qui interagit directement avec le système [MG00]. Pour notre SBT, nous avons pu identifier les acteurs suivants :

- Utilisateur (e.g., Analyste/Chercheur, Concepteur EIAH, Enseignant, Apprenant).
- Administrateur de l'application.
- Application externe utilisant des traces (EIAH).

7.2.3 Identification des cas d'utilisation

Un cas d'utilisation (use case) représente un ensemble d'actions réalisées par le système, en réponse à une action d'un acteur [MG00]. Il exprime les interactions acteurs/système et apporte un résultat observable intéressant pour l'acteur concerné.

La meilleure façon de trouver les cas d'utilisation, est d'essayer de trouver pour chaque acteur les tâches qui peuvent être satisfaites par le système. Pour chaque acteur, nous avons identifié les tâches (activités) suivantes :

1. Utilisateur :
   - Gérer les base de traces ;
   - Collecter des traces ;
   - Importer des traces / modèles de traces ;
   - Exporter des traces / modèles de traces ;
   - Transformer des traces ;

- Visualiser des traces ;
- Interroger des traces ;
- Gérer les modèles de traces ;
- Gérer les modèles de transformation ;
- Gérer les sources de collecte ;

2. Administrateur :
   - Gérer les utilisateurs ;
3. Application utilisant des traces :
   - Demander une transformation ;
   - Interroger une trace ;

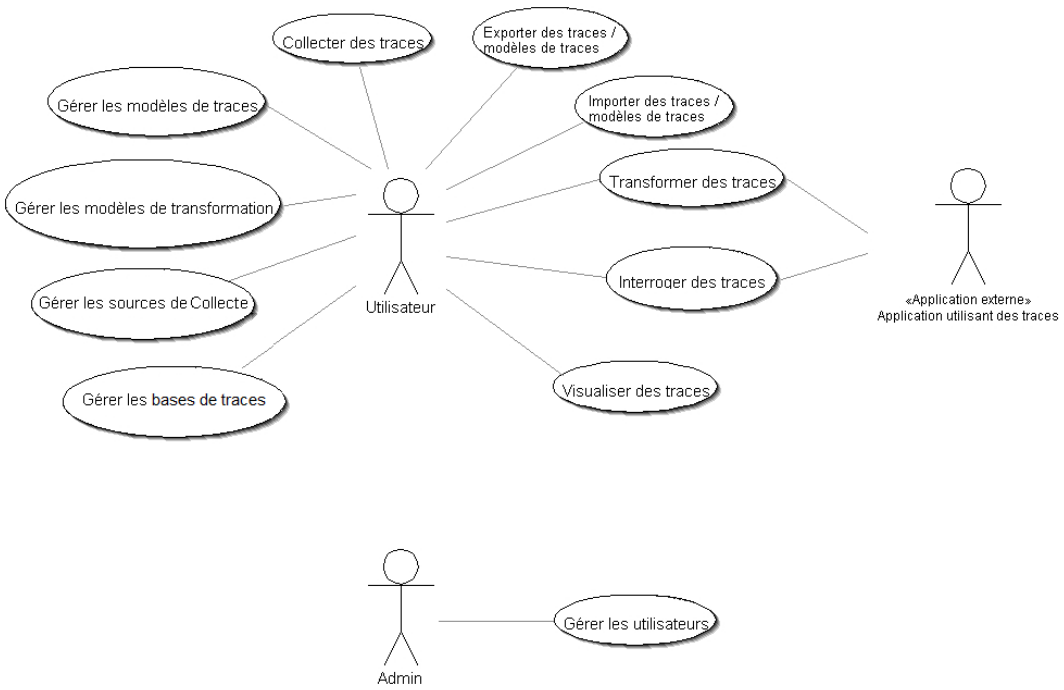

FIGURE 7.3 – Diagramme des cas d'utilisation du système ATER

L'identification des acteurs et de leurs cas d'utilisation est représentée graphiquement sur un diagramme de cas d'utilisation (Figure 7.3). Dans la suite, nous détaillons les différents modules qui composent notre système. Notamment, pour chaque module, nous évoquons et justifions les outils techniques utilisés.

7.3 Modules et technologies utilisées dans la plateforme ATER

ATER se présente comme une application Client/Serveur permettant le stockage de traces modélisées, leur gestion, manipulation (i.e., interrogation et transformation) et visualisation. La Figure 7.4 montre que le système ATER se présente sous la forme d'une plateforme JAVA utilisant différentes technologies que nous détaillons dans la suite.

Le client ATER est une application Eclipse RCP [77] offrant des interfaces de visualisations (traces, modèles, etc.). Le principal objectif du framework RCP est d'établir une plateforme *modulaire* et *extensible*, une propriété qui répond largement à nos contraintes. Son principe, fondé sur la notion de *plugin*, permet la mise à jour ou l'ajout de nouveaux plugins sans besoin de modifier le reste de l'application.

En terme de conception et de modélisation, le client ATER a été développé en se fondant sur la technologies de modélisation d'Eclipse EMF [78] (Eclipse Modeling Framewok). EMF est un framework permettant de modéliser une application et de générer le code java correspondant [79] respectant les patrons de conception les plus utilisés (e.g. factory, adapter, etc.). Le framework EMF offre aussi des possibilités d'implémenter des mécanismes de notifications et de sérialisation. Par exemple, la notification est utilisée dans notre projet pour prévenir les changements survenus sur les traces et leurs éléments.

Les visualisations dans ATER sont développées comme des éditeurs graphiques sous forme de plugins basés sur le framework GEF [80]. Ces plugins de visualisations mettent en œuvre une certaine représentation de la trace, parmi une multitude de représentations possibles. Notre visualisation se voulait générique et classique, mais d'autres plugins peuvent être développés pour des visualisations plus sophistiquées ou plus personnalisées.

Dans ATER, les traces ainsi que leurs modèles sont décrites en RDF, une représentation bien adaptée pour la modélisation mais limitée en terme d'interrogation et de transformation de M-Traces en temps réel (voir le chapitre 5 pour plus de détails). La manipulation des traces et des modèles RDF est faite avec la librairie EODM [81] (EMF Ontology Definition Metamodel) du framework IODT [82] développé par IBM. EODM est un gestionnaire d'ontologies permettant la création et la manipulation d'ontologies OWL/RDFS. EODM inclut également un parseur RDFS/OWL et permet de faire des raisonnements (en mémoire) sur les ontologies définies.

Le client ATER permet des dépôts de bases de M-Traces en local ou à distance via des services web SOAP [83]. Une base de M-Traces stockée à distance sur le serveur ATER se base sur deux technologies :Java Content Repository (JCR) [84] pour le stockage des sources de collectes ou toute autres ressources jugées nécessaires à l'exploitation des M-Traces et Minerva[ZML+06] pour le stockage des modèles et des traces décrites en RDF. Le serveur ATER gère la correspondance et la cohérence entre ces deux dispositifs de stockage.

77. Techniquement, Eclipse RCP est la plateforme sur laquelle sont construits tous les projets Eclipse. Cette plateforme est conçue comme un framework utilisable pour le développement d'applications clientes dites riches. Sur le plan visuel, la notion de *workbench* ainsi que les librairies SWT et JFace permettent la création d'interfaces graphiques de haute qualité (voir http ://www.eclipse.org/rcp/).
78. http ://www.eclipse.org/modeling/emf/
79. Ce code contient les déclarations d'objets métiers, sous forme d'interface et leur implémentation par défaut, ainsi qu'une fabrique d'objets (*factory*) permettant la création des différents objets.
80. Graphical Editing Framwork - http ://www.eclipse.org/gef
81. http ://www.eclipse.org/emft/projects/eodm/
82. Integrated Ontology Development Toolkit (voir http ://www.alphaworks.ibm.com/tech/semanticstk).
83. http ://fr.wikipedia.org/wiki/SOAP
84. http ://jackrabbit.apache.org/

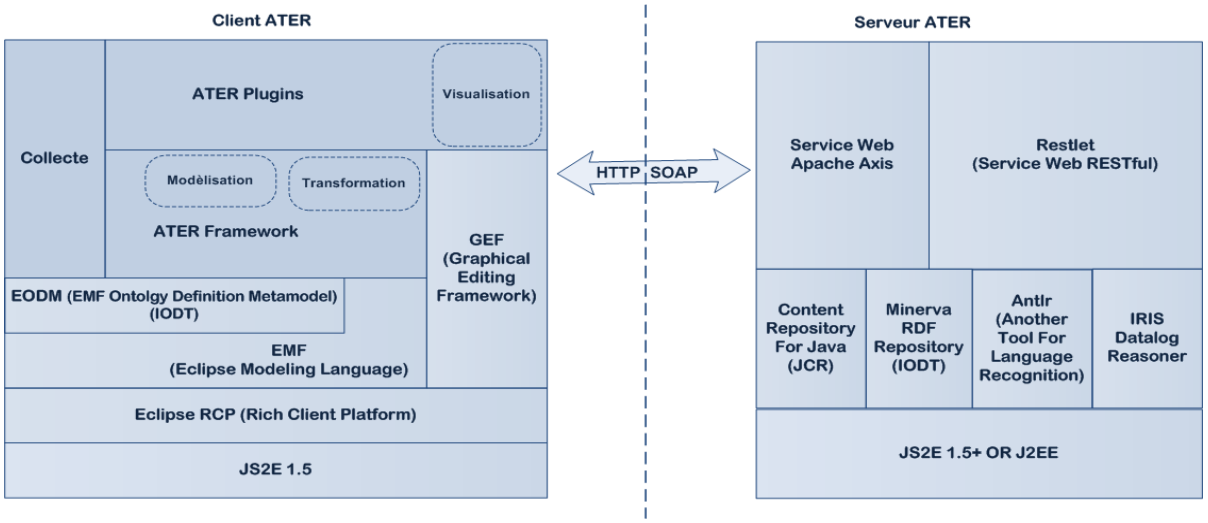

FIGURE 7.4 – Architecture technique du système ATER

Le serveur ATER se présente comme une application Web basée principalement sur les technologies suivantes :

- Le framework Apache Axis [85] nous permettant de mettre en œuvre des communications via des services web, HTTP et SOAP.
- Le framework IODT (Integrated Ontology Development Toolkit) développé par IBM que nous utilisons pour la représentation et manipulation de M-Traces en RDF. IODT offre aussi un entrepôt RDF appelé Minerva[ZML+06], permettant de stocker et d'accéder à distances à des fichiers RDF/OWL et d'exprimer des requêtes SPARQL (dans sa version 1.0).
- Java Content Repository (JCR) que nous utilisons pour le stockage, la gestion et la manipulation des ressources complémentaires (e.g. les sources de collecte, les vidéos contextuelles, les productions, etc.).
- Le framework Restlet [86] permettant l'implémentation de services web Restful. Les services web Restful sont utilisés pour ajouter des observés dans l'entrepôt RDF Minerva.
- Le framework Antlr [87] pour le développement de parseurs et de compilateurs de langages. Nous utilisons ce framework pour implémenter la syntaxe d'une version antérieure de notre langage définie dans [SPC+09].
- Le moteur d'inférence Datalog Iris [88] pour l'évaluation des requêtes et des transformation de M-Traces.

85. http ://ws.apache.org/axis/
86. http ://www.restlet.org/
87. http ://www.antlr.org/
88. http ://www.iris-reasoner.org/

7.3.1 Une vue d'ensemble du Code Source de la plateforme ATER

Il n'y a pas encore de version stable (i.e. une *release*) du système ATER. Cependant, le code source sous LGPL et les packages compilés sont disponibles sur le serveur SVN du liris à l'adresse (https ://svn.liris.cnrs.fr/tt/TRUNK/ATER/). Le source se compose de 4 parties organisées sous forme de projets Eclipse :

- Le noyau du framework (org.ater.framework.core) est le projet relatif à la manipulation du modèle de trace, la trace en RDF et des transformations SPARQL. Ce projet à des dépendances avec le package IODT de manipulation du langage RDF notamment EODM.
- Le projet (org.ater.framework) a des dépendances avec le noyau (org.ater.framework.core). Il contient les packages suivants :
  - Package de stockage et de gestion de bases de M-Traces (org.ater.framework.bases)
  - Package de sécurité (identification et gestion des utilisateurs, privilèges, groupes d'utilisateurs, etc.) (org.ater.framework.security)
  - Package de recherche et d'interrogation (org.ater.framework.query)
  - Package de gestion de sources de collecte (org.ater.framework.ressources)
  - Package de gestion de catégories et des relations pour organiser les M-Traces et les modèles (org.ater.framework.categories) (org.ater.framework.relations)
  - Package de transformation de M-Traces (org.ater.framework.transformation)
  - Package de description et de calculs d'indicateurs (org.ater.framework.indicators)
- Le projet (org.ater.server) est composé de trois packages : (org.ater.server.axis) pour l'implémentation des accès via des services web à JCR et Minerva ; (org.ater.server.perser) qui implémente le parseur pour notre langage de transformation ; (org.ater.server.restful) permettant des ajouts de traces, de modèles de trace et d'observés dans un fichier RDF de l'entrepôt RDF Minerva ; (org.ater.server.engine) permettant la conversion des requêtes et des transformations en langage Datalog et leurs évaluations dans le raisonneur Iris.
- Une application Eclispe RCP (org.ater.client) qui se base sur les précédentes parties. C'est une interface utilisateur permettant d'exécuter et d'utiliser les différents services. Elle peut être vue comme une interface d'administration du framework puisqu'elle permet de se connecter aux différentes bases de M-Traces.

7.3.2 Fonctionnalités du prototype

Les copies d'écrans que nous allons montrer présente l'exécution du client ATER ainsi que le serveur ATER déployé sur un serveur web Apache Tomcat sous le système d'exploitation Linux Ubuntu.

Le client ATER implémente différentes vues organisées en 4 perspectives. Le cadre noire de la figure 7.5 montre la liste des modules de la plateforme. Le Client ATER permet la création de bases de M-Traces en local ou à distance (voir le cadre orange de la figure 7.5). Techniquement, une base de M-Traces dans ATER est un couplage entre des nœuds JCR qui pointent sur des fichiers RDF (traces et modèles) dans l'entrepôt RDF Minerva. L'exemple de la figure 7.5 montre

quatre bases de M-Traces dont trois sont locales et une qui est sur le serveur.

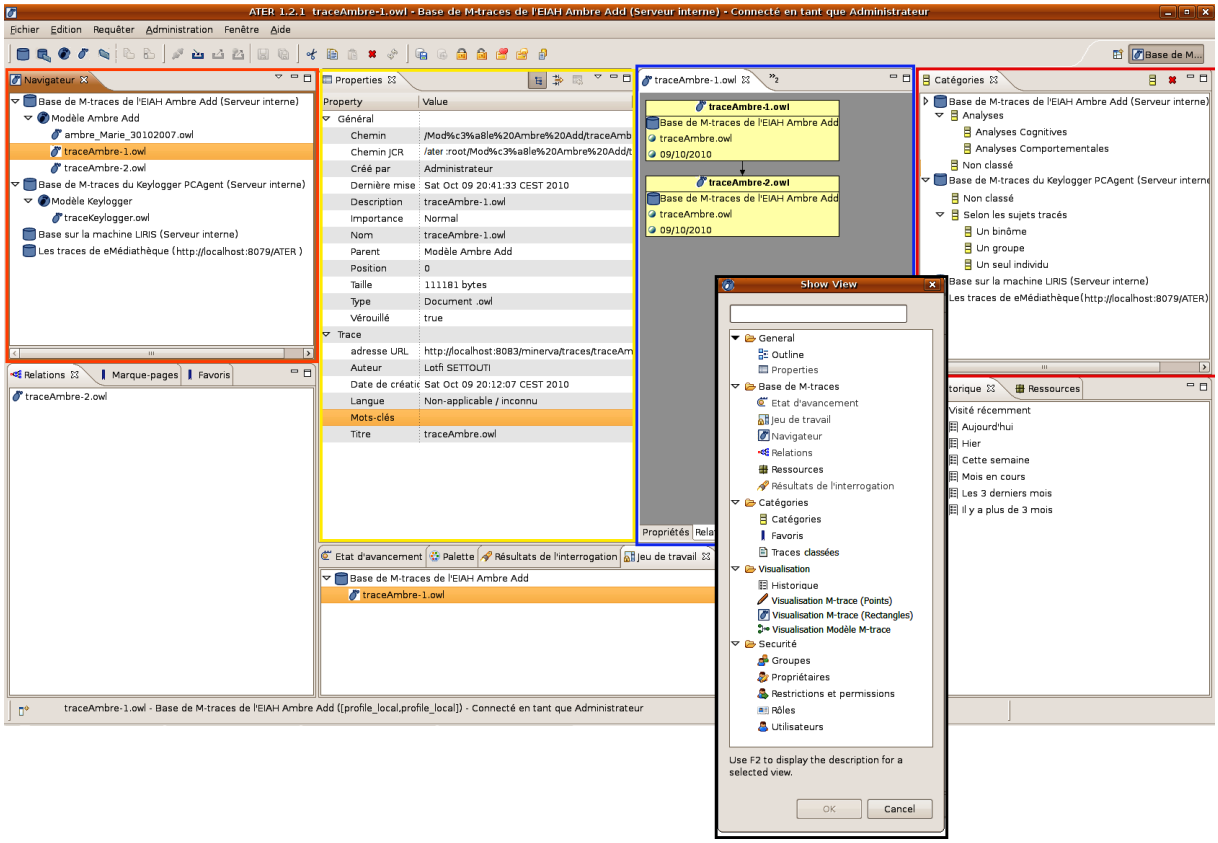

FIGURE 7.5 – Copie d'écran du client ATER avec la liste des différents modules développées

Le client ATER permet aussi de rajouter à la base de M-Traces des modèles et des M-Traces. Pour les modèles de traces, s'ils sont dans le format RDF de ATER, ils sont directement ajoutés à Minerva mais aussi à JCR (voir le cadre jaune de la figure 7.5). Les M-Traces sont ajoutées dans une base et associées à un modèle. Des sources externes (appelées ressources) peuvent être ajoutées à la base de M-Traces et stockées dans l'entrepôt JCR. Ces ressources peuvent être de différents types (vidéos, XML, texte, etc.). ATER permet de les visualiser avec leurs lecteurs associés. Cependant, la synchronisation temporelle entre ces sources et les M-Traces n'est pas encore implémentée.

Le client ATER permet de créer des catégories pour organiser les M-Traces et les ressources stockées dans les bases de M-Traces (voir le cadre rouge de la figure 7.5). Lors de l'ajout de modèles, de traces ou de ressources, une ou plusieurs catégories peuvent être associées à ces éléments. Ceci permet par la suite de classer les différents éléments dans la base de M-Traces et facilite l'accès et la recherche lorsque le nombre de M-Traces et des ressources associées est important (par exemple les traces d'une classe). ATER permet également de lier deux M-Traces entre elles ou des M-Traces à des ressources et de visualiser graphiquement ces liens (voir le cadre bleu de la figure 7.5). Ceci facilite leurs gestions permettant de visualiser l'historique des

traitements, de réifier des liens de versioning ou de précédence temporelle (par exemple une M-Trace est la suite d'une autre, etc.). Ces relations lient des éléments de la base de M-Traces et sont évidement différentes des relations entre observés dans une même M-Trace.

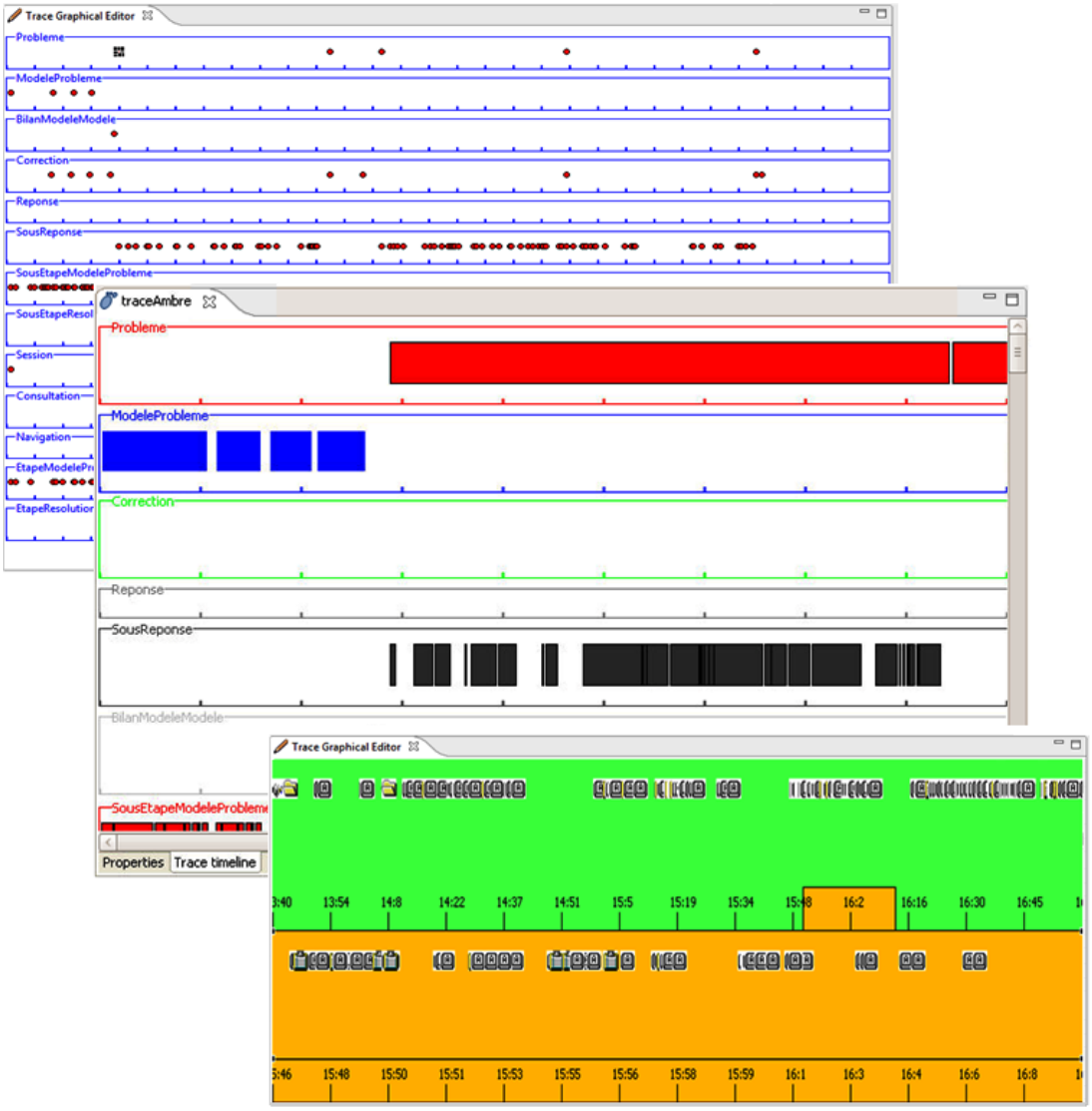

FIGURE 7.6 – Trois visualisations de M-Traces dans ATER. Les exemples des visualisations 1 et 2 concernent la M-Trace de l'EIAH Ambre-add. La troisième visualisation représente la M-Trace d'un keylogger.

Le client ATER permet différentes visualisations de M-Traces. Quatre *plugin* de visualisation ont été développés : une visualisation de modèle de traces (figure 7.7) et trois visualisations de traces (figure 7.6). L'éditeur de visualisation présenté en haut de la figure 7.6 représente une trace sous forme de diagramme, constitué de plusieurs figures représentant des timelines. Ces figures représentent un axe du temps, équipées d'une règle graduée de gauche à droite, soit selon une unité temporelle (heure, minute, seconde) ou tous simplement par une relation d'ordre. Chacune

de ces timelines représente une classe d'observé. Les observés sont disposés selon leurs valeurs temporelles sur la ligne de temps et représentés par des figures descriptives (points, ellipse, icône, etc.).

La seconde visualisation présentée au milieu de la figure 7.6 est une amélioration de la première en deux points : elle permet de représenter la durée des observés et de configurer, en plus des couleurs, l'échelle de temps de la règles visualisée (e.g, une règle de 5 minutes, ou de 30 secondes). La troisième visualisation présentée en bas de la figure 7.6 permet de mettre en œuvre un mécanisme de zoom. La partie verte permet la visualisation de tous les observés de la M-Trace alors que la partie orange représente juste les observés sur un sous intervalle de la première partie.

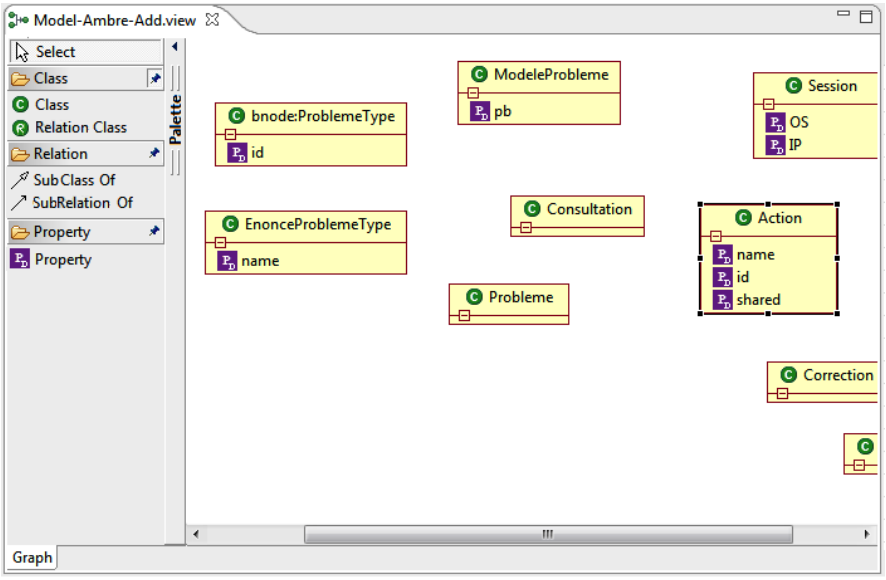

FIGURE 7.7 – La visualisation de modèle de traces dans ATER

La vue *propriété* de ATER (présentée dans la figure 7.5) permet de manipuler les propriétés des M-Traces sélectionnées mais aussi les observés sélectionnés dans les différentes visualisations de la figure 7.6. D'autres vues peuvent être utilisées en parallèle à l'éditeur, comme la vue *outline* qui permet la mise à jour d'une structure d'arbre représentant la trace visualisée et ses observés. Ceci permet donc en plus de la visualisation, d'éditer les traces en permettant l'ajout et la suppression d'observés ou la modification de certaines de leurs propriétés. Cependant, différentes améliorations peuvent être ajoutées à ces plugins, notamment un service de zoom performant permettant l'analyse d'observés temporellement condensés, et des fonctions d'édition plus intuitives pour les mises à jours manuelles de M-Traces, d'observés ou de modèles de traces.

Le client ATER dispose d'une gestion avancée des accès aux bases de M-Traces (figure 7.8). ATER permet la gestion d'utilisateurs, de groupes d'utilisateur, de rôles, des propriétaires des M-Traces. Il permet aussi de gérer les privilèges de chacun d'entre eux sur n'importe quelle base de M-Traces ou élément lui appartenant (i.e. modèles, traces et ressources).

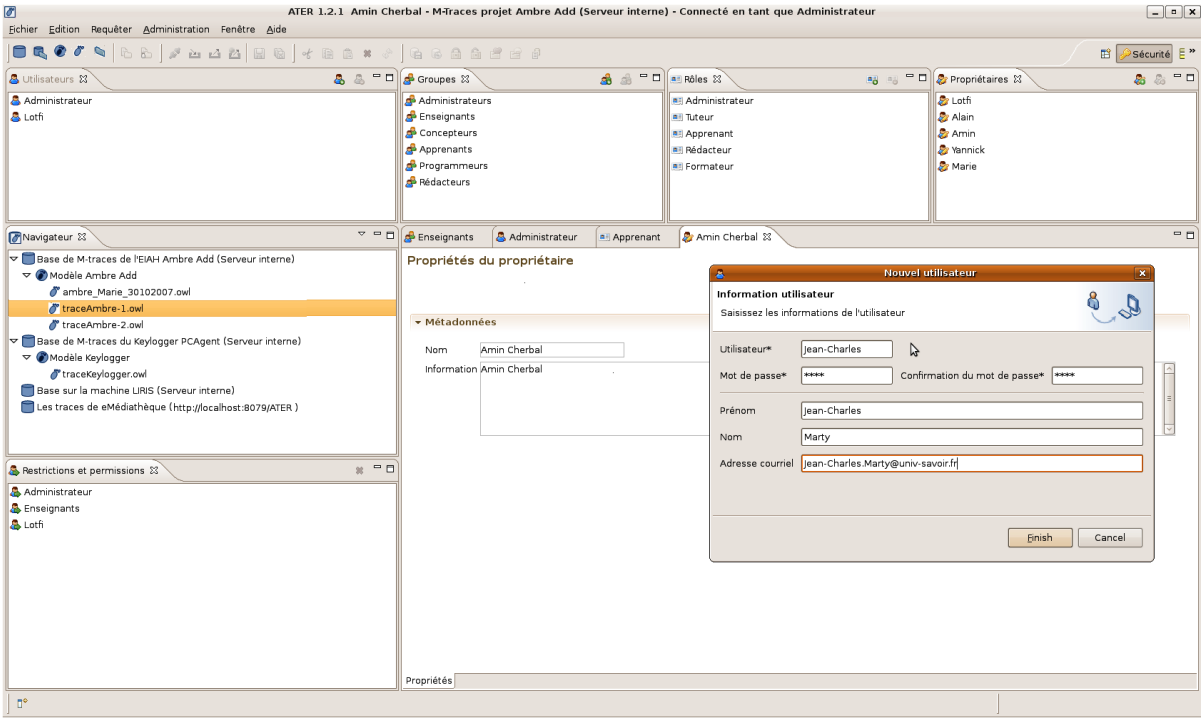

FIGURE 7.8 – La gestion des utilisateurs, groupes et les privilèges dans ATER

7.4 Les différents incréments du projet ATER et historique des développements

La plateforme ATER a été le cadre de plusieurs stages se conformant tous au cycle de développement itératif et incrémental. L'architecture technique d'Eclipse sur laquelle est fondée ATER permet naturellement de soutenir un développement incrémental [89] de modules appelés *plugins*. Nous résumons dans le suite les différents incréments développés, leur intégration dans la plateforme et leur état d'aboutissement.

Incrément 1 : le socle de base (2007)

Le développement de la plateforme ATER a commencé en 2007 dans le cadre d'un stage PFE effectué au laboratoire LIRIS de deux élèves ingénieurs de l'université de Tlemcen (Algérie) [90]. Ce premier incrément s'est focalisé sur le développement des modules permettant :

- la création de bases de M-Traces en local ou à distance.
- la collecte et l'importation de traces modélisées dans une base de M-Traces

89. En effet, le objectif principal des technologies Eclipse était d'établir une plateforme *modulaire* et *extensible*. Au cœur du projet Eclipse se trouve un « noyau », qui est une implémentation conforme aux spécifications OSGi (Open Services Gateway initiative), le noyau d'Eclipse est responsable de la gestion du cycle de vie des modules intégrant la plateforme.

90. Suite à une demande acceptée d'un financement de stage auprès du « PPF e-praxis ».

- l'association à ces M-Traces aux différentes ressources utilisées (e.g., sources de collecte, vidéo contextuelle, productions de l'apprenant),
- le classement des M-Traces et des ressources associées en catégories et sous-catégories.
- la mise en relation entre M-Traceset historique des ajouts et des modifications dans une base de M-Traces.
- la visualisation et édition de M-Traces.

Les requêtes et les transformations ont été mises en œuvre en se basant sur le langage SPARQL et le framework IODT (figure 7.9). Des requêtes sur l'entrepôt de contenus JCR peuvent être effectuées (figure 7.9). En effet, JCR est également interrogeable à travers des requêtes XPath et/ou SQL.

FIGURE 7.9 – La vue pour exprimer des requêtes SPARQL dans ATER

Incrément 2 : les modules de visualisation de M-Traces (2008)

Dans la suite des développements de la plateforme ATER, trois plugins ont été développés par des étudiants en Master M1 informatique de l'Université Lyon 1 dans le cadre de leurs stages TER (Travail d'Étude et de Recherche). Les plugins développés concernait un éditeur graphique de modèle de trace (figure 7.7), la visualisation de trace à base de son modèle (figure 7.6) et un éditeur d'expressions de calculs d'indicateurs à base de traces. Les deux premiers plugins sont intégrés dans la plateforme, le troisième module pour le calculs d'indicateur n'a pas été finalisé. La durée des stages étant très courte (seulement un mois), les plugins développés requièrent une

maintenance et des mises à niveau du code pour prendre en compte l'évolution des technologies utilisées (e.g. la nouvelle version de GEF [91]).

Incrément 3 : le moteur de transformation de M-Traces (2009)

Le troisième incrément s'est construit sur deux résultats : la formalisation de notre langage d'interrogation et de transformation de M-Traces [SPC+09] et la spécification technique des protocoles de communications pour les SBT dans le cadre du projet ITHACA [92] [CCPS09]. Sur la base de ces deux résultats, un stage de PFE ingénieur de 4 mois a démarré à l'université de Tlemcen (Département informatique). Le résultat de stage s'est soldé par l'implémentation d'un moteur d'interprétation de requêtes et de transformations de traces fondé sur Datalog et Restlet. L'ajout des observés et des modèles dans l'entrepôt RDF Minerva se fait par des services Web RESTful. Ces triplets RDF sont ensuite convertis dans le langage Datalog de la bibliothèque *Iris* [93] pour l'exécution d'une requête ou transformation. La conversion des M-Traces RDF en Datalog se fait suivant les algorithmes 1 et 2 définis pour une formalisation précédente de notre langage dans [SPC+09].

Les requêtes et les transformations sont analysées et parsées grâce à un parseur (généré grâce au framework Antlr) implémentant la grammaire du langage telle qu'elle a été définie dans [SPC+09]. Cependant, le prototype développé n'est pas intégré totalement dans la plateforme ATER, et implémente que partiellement le langage puisque seule la conjonction de patterns est développée.

Les requêtes et les transformations sont traduites en programmes Datalog grâce une implémentation des algorithmes 3, 4 et 5 définis dans [SPC+09]. L'exécution d'un programme Datalog (correspondant à une requête ou une transformation) permet d'obtenir les résultats (i.e., les substitutions ou la M-Trace transformée). Cependant, la conversion de ces résultats en RDF n'est pas encore développée.

7.5 Bilan et limitations courantes du prototype ATER

Pour l'heure, ATER supporte l'expression de modèles de trace comme des ontologies RDF décrivant des concepts et des relations. Les traces décrites par un modèle de trace ontologique sont constituées d'instances représentant leurs observés. Son architecture fournit le stockage et la gestion avancée de modèles, de traces et de sources de collecte.

Dans ATER, la collecte n'est pas générique et doit être codée spécifiquement pour chaque application. Elle peut être exécutée en temps réel ou en différé et permet de passer d'une ou plusieurs sources de collecte quelconques à une trace première stockée dans une base de traces RDF. Chaque collecte dans ATER est organisée en deux étapes : la première consiste en la description du modèle de la trace première contraignant et structurant la trace collectée. La

91. Graphical Editing Framework
92. Interactive Traces for Humain Awareness in Collaborative Annotation
93. http ://www.iris-reasoner.org/

seconde concerne l'instanciation du modèle pour créer la trace en interprétant les sources de collecte de manière *ad-hoc*. Étant donné la diversité des sources de collecte et des techniques pouvant être utilisées dans la collecte, rendre la seconde étape générique est difficile et n'est pas solutionné dans de notre projet. Pour l'heure, nous avons implémenté trois routines de collecte pour collecter les traces issues de l'EIAH « Géonote [94] », l'EIAH « Ambre-add [95] » et le logiciel « PC-Agent Keylogger [96] ». Les sources de collecte générées par ces trois applications étaient sous le format XML. Notre expérience lors de la mise en œuvre de différentes collectes a montré que sa complexité est dépendante d'au moins deux aspects : le format des sources de données (XML, BDD, text, etc.) et les synchronisations temporelles nécessaires pour produire une M-Trace cohérente par rapport au domaine temporel considéré.

Une fois la trace première instanciée, le système permet la gestion intégrée et générique des traces : l'utilisateur peut spécifier des transformations à l'aide de règles de transformation (les transformations étant exprimée en SPARQL ou dans notre langage dans sa version ancienne définie dans [SPC+09]). Il peut également bénéficier des services de visualisation dans les traces modélisées. Cependant, la plateforme ATER présente un certain nombre de limitations. En effet, l'implémentation actuelle de la plateforme ATER est un prototype de recherche et est loin d'être un produit fini. Comme tel, certains aspects ou fonctionnalités ne sont pas essentielles pour faire la preuve des concepts proposés. Nous allons lister les limites actuelles de la plateforme. Aucune des limites discutées n'est inhérente à l'architecture du prototype développé, qui a été conçu pour être facilement extensible.

- Il n'y a pas de zoom intelligible qui prend en compte les propriétés de la M-Trace (densité des observés, chevauchement temporel, etc.). Les visualisations développées sont trop primaires pour permettre des navigations intuitives dans les traces stockées.
- Il n'y pas de visualisation des résultats d'une requête ou d'une transformation qu'elle soit exprimée en SPARQL ou dans notre langage (défini dans [SPC+09]).
- Il n'y pas d'édition de modèle de trace. Le plugin développé dans ATER permet juste de visualiser les modèles mais ne permet pas de les éditer ou modifier. La création d'un modèle de trace revient à importer un fichier RDF représentant le modèle de trace. Donc, un éditeur RDF externe doit être utilisé (e.g. Protégé [97]) pour créer le modèle avant de l'importer dans la plateforme.
- Certains plugins ne sont pas intégrés totalement dans la plateforme. Par exemple, la troisième visualisation de M-Traces (figure 7.6) est actuellement liée au modèle de trace du keylogger et n'est pas intégrée totalement à la plateforme. Le plugin pour le calcul d'indicateurs n'est pas lié au modèle de donnée de ATER et les expressions de calculs décrites ne sont pas sauvegardées dans la base de M-Traces.
- Il n'y a pas d'implémentation du langage d'interrogation et de transformation de M-Traces

94. http ://praxis.inrp.fr/praxis/projets/geomatique/geonote/
95. http ://bat710.univ-lyon1.fr/nguin/ambre.html
96. un logiciel espion qui a la particularité d'enregistrer les touches frappées sur le clavier sous certaines conditions.
97. http ://protege.stanford.edu/

présenté dans cette thèse (chapitre 4). Les résultats présentés étant très récents, ils ne sont pas encore implémentés dans la plateforme. Il existe cependant une implémentation partiel de la version antérieure du langage définie dans [SPC+09].

7.6 Discussions autour des difficultés du projet ATER

Bien que la plateforme ATER soit fonctionnelle et permette d'offrir, comme nous l'avons montré, un certains nombre de services pour la gestion du cycle de vie des traces, des modèles et des ressources associés, elle n'est actuellement utilisée dans aucun projet. Ce projet de développement peut même être considéré comme un échec puisque aucun développement future n'est prévu que ce soit pour faire évoluer les fonctionnalités de la plateforme ou maintenir, tester et corriger son code. Les raisons en sont de nature diverses : certaines sont liées intrinsèquement au risque inhérent à toute gestion de projet informatique, certaines d'autres sont de nature technique et sont dépendantes des technologies utilisées, enfin d'autres sont liées aux caractéristiques et challenges relevés par notre approche. Nous discutons dans la suite les principales raisons de ces difficultés.

- *Disponibilité d'une version stable.* Le prototype ATER présenté n'est toujours pas en version *stable* ou *finale* et certains modules ou fonctionnalités doivent être corrigées ou complétées. Ceci est dû a de nombreuses raisons :
  - Certaines technologies ont énormément évolué parfois avec des changements conséquents ne gardant pas une cohérence totale ou une interopérabilité entre les versions. C'est le cas du framework GEF utilisé pour les visualisations qui à chaque nouvelle version nécessitait des efforts considérables de migration. Les différents modules implémentés n'ayant pas été développés au même moment, ils ont engendré des dépendances conflictuelles très couteuses en temps pour les résoudre.
  - Certaines erreurs n'ont pas été jusqu'à maintenant corrigées. Par exemple, un bug découvert notamment dans JCR Jackrabbit fait perdre certaines sauvegardes.
- *Extensibilité.* L'extension de la plateforme et la prise en main de son code source est une tâche fastidieuse. L'usage des technologies Eclipse (EMF, GEF, etc.) connues pour leur documentation rudimentaire, couplées aux technologies du web sémantique (issues de la recherche) complique certainement cette tâche. Bien que la conception faite de la plateforme utilise les bonnes pratiques en termes de modélisation (l'usage de patrons de conception, méta-modélisation, abstraction, excellent découpage en packages en minimisant les dépendances, etc.), le code source à étendre reste très volumineux. ATER nécessite également une bonne appropriation de technologies ayant parfois des logiques d'utilisation disparates (e.g. JCR et Minerva).
- *Le prix de la généricité.* Il est bien connu que la généricité des outils nécessite plus d'efforts pour « ajuster » ces derniers à l'activité spécifique que l'on souhaite réaliser. Ceci est évidemment le principal inconvénient retenu à l'encontre de cette classe d'outils dits génériques qui offrent une multitude de possibilités. D'ailleurs, ces outils s'efforcent de plus

en plus à offrir des possibilités poussées de paramétrage, de configuration ou d'adaptation selon les besoins et les activités. Dans notre contexte, il s'agissait d'offrir des services génériques de représentation, d'interrogation et de transformation de M-Traces collectées de différentes sources. Si les « ajustements » pour permettre la spécialisation des représentations de M-Traces, l'usage de requêtes, l'expression de transformations semblent être acquis ou atteignables [98], il est d'une complexité plus élevée dans le cas de la collecte. Notre expérience dans l'équipe Silex et dans le projet « personnalisation des EIAH » nous a fait constater qu'offrir des mécanismes de collecte intelligibles non seulement pour les développeurs mais aussi pour des utilisateurs moins experts est une nécessité. ATER a montré qu'il était impératif de disposer de protocoles de collecte intuitifs, faciles à mettre en œuvre tout en étant utilisables dans de multiples situations. ATER ne disposant pas de telles facilités a été délaissé par beaucoup de prétendants à son usage.

– *Gestion de projet.* La gestion du projet de développement de la plateforme a sûrement manqué de moyens et d'expérience dans sa gestion. Les coûts et les temps de développement alloués à la conception et l'implémentation de ATER ont été d'une part sous-estimées par manque d'expérience et d'une autre part évaluées selon les modestes moyens dont nous disposions. Le manque de temps et de personnes dédiés spécifiquement à la maintenance et l'extensibilité de la plateforme n'a fait que précipiter l'inévitable pour les prototypes de recherche : leurs délaissements.
– *Performance et optimisation.* Nos premiers essais avec des M-Traces peu volumineuses ont montré rapidement les limites des technologies existantes. L'usage d'un entrepôt RDF couplé à l'entrepôt documentaire JCR a augmenté considérablement les temps d'accès et de réponse. Même en ne considérant que les entrepôt RDF, comme il a été constaté dans la communauté du web sémantique (e.g. [HH07]), ces technologies RDF ne disposent pas encore des optimisations efficaces pour prendre en compte des données volumineuses [HH07]. Par exemple, pour traiter en continu, partie par partie les traces RDF volumineuses issues des réseaux sociaux (voir [BBC+09]). Le web sémantique pose d'ailleurs en partie la même question que celle relevée par la problématique des M-Traces : de disposer de langages monotones (comme celui que nous avons défini dans le chapitre 4) pour rendre des services plus appropriés dans le cas de flux temporels explicitement modélisés et traités au fil de l'eau.
– *Problématique théorique.* Les langages utilisés dans ATER ne permettent pas de prendre en charge la diversité des situations auxquelles a été confronté la plateforme ATER (des analyses *post-hoc*, des transformations en quasi-temps réel, la prise en charge des relations temporelles, etc.). Notre implémentation nécessitait des avancées théoriques sur la sémantique des langages pour les M-Traces qui n'existait pas encore. Les derniers résultats que nous avons obtenus et présentés dans le chapitre 4 nous permettent de penser qu'il est maintenant possible de disposer de langages efficaces prenant en compte réellement les

98. Sur le plan de la modélisation, la thèse de Julien Laflaquière [Laf09] donne certaines recommandations et des éléments dans ce sens.

caractéristiques et les particularités des M-Traces.

Quatrième partie

Conclusion et perspectives

Chapitre 8

Conclusions et perspectives

Sommaire

Cet ultime chapitre sera consacré à une conclusion étendue résumant l'ensemble de notre travail de thèse. Nous dresserons dans un premier temps un bilan de nos propositions en mettant en avant les principaux apports. Dans un second temps, nous détaillerons les perspectives que ce travail offre quant à de futurs travaux de recherche sur les systèmes à base de M-Traces ainsi que les langages associés.

8.1 Bilan et principaux apports de cette thèse

Ce travail de thèse a pour objectifs d'étudier et de proposer des outils permettant l'exploitation des traces informatiques. Il a également la particularité d'être à l'intersection de deux domaines de recherche : (1) le premier est le domaine de l'ingénierie des EIAH reconnu pour être un vivier pour les usages de traces et qui a vu une intensification des travaux concernant cette notion ces dernières années ; (2) le second concerne l'Ingénierie des Connaissances (IC) notamment les langages permettant de représenter et de manipuler les traces. Bien que la problématique des traces s'inscrive dans le cadre des EIAH, notre contribution principale relève bien de l'Ingénierie des Connaissances. Nous allons dans la suite résumer les principaux aspects relevés et traités dans cette thèse.

8.1.1 Contexte, problématique et objectifs de ce travail de thèse

Comme nous l'avons mentionné dans le chapitre 1, ce travail de thèse s'inscrit dans le cadre du projet « personnalisation des EIAH ». Ce projet et plus généralement les besoins autour des traces dans les EIAH montrent bien que les systèmes existants sont divers mais limités puisqu'ils sont généralement très ciblés pour répondre à un besoin spécifique et sont ainsi peu génériques. Ce chapitre a montré également qu'exploiter les traces issues d'une activité pour y découvrir des éléments permettant d'agir sur l'environnement d'apprentissage nécessite auparavant une théorisation de la notion de trace pour construire enfin les outils et les langages associés.

Le chapitre 2 détaille cette problématique et a permis de relever les propriétés et caractéristiques requises par l'exploitation des traces notamment pour les EIAH. Si les traces constituent le matériau de nombreuses études *post-hoc* dans le cadre d'expérimentations impliquant des EIAH, elles sont également l'objet d'utilisation *ad-hoc* notamment pour élaborer des *connaissances* permettant des actions de personnalisation. Cette élaboration de *connaissances* nécessite souvent des outils et des langages œuvrant de manière duale pendant et après l'activité tracée pour permettre cette personnalisation, i.e., permettre des exploitations des traces en temps réel ou différé de leurs collectes.

Même si les environnements informatiques (pour l'apprentissage humain en particulier) ont été très souvent tracés pour permettre une émergence de sens à partir des interactions observées, il n'existe que peu de travaux proposant de donner à la trace numérique d'interaction le statut d'objet informatique à part entière élaborant les connaissances liées à cette émergence. Les contributions (Chapitres 3, 4) que nous avons proposées vont dans ce sens, en assumant les propriétés que l'on peut attendre d'un tel objet trace et en proposant les représentations, les langages et les mécanismes d'exploitation associés.

Ces outils et langages visés par le cadre théorique proposé, baptisés *Systèmes à Base de Traces modélisées (SBT)*, vus comme une nouvelle classe de systèmes à base de connaissances, nécessitent une définition poussée des concepts et des langages permettant de les mettre en œuvre. Cette définition s'est étalée sur deux chapitres principaux (Chapitres 3, 4) présentant respectivement le cadre conceptuel et formel des SBT.

8.1.2 Cadre Conceptuel et architecture des systèmes à base de M-Traces

Notre première contribution consiste en la définition d'un cadre conceptuel pour les systèmes à base de traces modélisées. Le cadre conceptuel que nous avons présenté (Chapitre 3) considère les traces numériques comme étant des objets informatiques à part entière, manipulables en tant que tels. Notre vision, qui constitue un cas singulier dans l'ensemble des travaux existants sur les traces, se concrétise dans un cadre conceptuel par la définition du concept de *trace modélisée* (ou M-Trace) et les services sous-jacents à son traitement. L'idée centrale est qu'une trace numérique doit être modélisée *a priori* pour dépasser les limites sémantiques habituellement considérées (par exemple pour les traces de type log). Pour cela le cadre conceptuel fournit la définition de modèle de trace, comme support à la représentation concrète et explicite de toute trace. Sur la base de ces définitions, le cadre des SBT définit les services permettant la manipulation des traces modélisées :

- Un service de collecte qui alimente d'une manière ou d'une autre le SBT à partir de données issues de l'interaction d'un utilisateur avec son système.
- Un service de requête et transformation des traces modélisées, qui globalement consiste à préparer les données à leur exploitation en tant que traces d'activité.

Le cadre conceptuel que nous avons présenté permet de disposer d'une fonction d'organisation des pratiques à base de traces, dans le sens où il oriente la démarche et la logique des étapes nécessaires pour manipuler les traces. Par ailleurs, ce cadre est indispensable également pour évaluer les pratiques et les résultats notamment dans le cadre du projet personnalisation des EIAH. Plus concrètement, le cadre conceptuel permet d'une part d'étudier les outils utilisant des traces en repérant les différentes exploitations possibles, et d'autre part, de construire une nouvelle classe d'outils pour la manipulation de traces homogènes et interopérables. D'un point de vue technique, on peut alors :

- modéliser les traces et spécifier les traitements dans les systèmes existants et ainsi à long terme de pouvoir s'échanger, partager et réutiliser des traces à des fins de capitalisation de pratiques (voir notamment le chapitre 6).
- construire un cadre logiciel (Framework), qui pourra être spécialisé, intégré ou utilisé pour construire divers outils ou services à base de traces (voir le chapitre 7).

Si la solidité du cadre conceptuel est acquise notamment à travers deux exemples de deux EIAH exploitant notre cadre (Chapitres 6) mais également à travers différents travaux effectués dans le cadre du projet *personnalisation des EIAH*, il manque encore à notre approche les langages permettant de prendre en compte complètement les caractéristiques des M-Traces et les mécanismes nécessaires pour son exploitation. Pour mettre en œuvre ces langages, nous avons défini un cadre formel définissant la sémantique requise et attendue pour l'interrogation et la transformation de M-Traces.

8.1.3 Cadre formel des langages de requêtes et de transformations de M-Traces

Notre seconde contribution concerne la définition formelle des concepts présentés dans le cadre conceptuel. Le Chapitre 4 a permis de définir les langages sur lesquels se fondent les systèmes à base de traces modélisées. En se basant sur la définition de la notion de trace modélisée, nous avons d'abord défini la syntaxe de notre langage pour les requêtes en donnant la syntaxe des *patterns* et les transformations en donnant la syntaxe des *templates*. De manière informelle, la notion de *patterns* et *templates* représentent respectivement les moyens d'obtenir et de construire les observés de la M-Trace.

Nous avons choisi une approche déclarative pour la définition de la sémantique des langages de requêtes et de transformations. Le problème étudié par notre sémantique déclarative peut être décrit à un niveau très abstrait comme suit : étant donné un ensemble de règles de transformation (appelé aussi transformation) et un flux entrant d'observés (ainsi que leurs relations et valeurs d'attribut associés) de la M-Trace qui sont collectés et ne sont pas dérivés ou transformés par les règles, nous voulons avoir tous les éléments qui sont dérivés par les règles de transformation. Cette approche couvre donc également la question de trouver les réponses à une requête étant donné un pattern et un flux entrant d'observés de la M-Trace.

La sémantique déclarative de notre langage est donnée sous forme d'une théorie des modèles dans le style de Tarski (i.e. définie récursivement sur la structure des expressions du langage) et adaptée aux exigences spécifiques des transformations de M-Traces (e.g. les divers aspects temporels des patterns de requête, les expressions dans les templates de règle de transformation, etc.).

Notre proposition d'une sémantique déclarative pour les transformations s'est articulée autour de différentes aspects comme le résumé la table 4.1 et s'est organisée comme suit :

1. Nous avons d'abord présenté les concepts de base de notre sémantique notamment la notion de *substitution*, d'*interprétation* et de *satisfaction* d'un pattern. Ces notions ont été volontairement définies de manière très générale pour pouvoir prendre en compte les différents cas étudiés (cf. tableau 8.1).
2. Sur la base de ces notions, nous avons défini la sémantique des requêtes simples exprimées avec QML^{-} dans le cas d'une évaluation ponctuelle et continue.
3. Sur la base de ces défintions, nous avons défini la semantique des allo-transformations, i.e. construisant une nouvelle M-Trace à partir d'une M-Trace existante. En se fondant sur la satisfaction de transformation simple exprimée avec QML^{-}, nous avons défini la sémantique des transformations dans le cas d'une évaluation ponctuelle et continue.
4. Ensuite, nous avons défini la sémantique des auto-transformations d'une M-Trace. Nous avons défini la sémantique des transformations complexes exprimées avec QML^{w} dans le cas d'une évaluation continue en proposant une théorie du point fixe. En effet, la théorie des modèles a la particularité d'admettre de nombreux modèles pour une transformation donnée. Pour donner une sémantique précise pour les requêtes et les transforma-

tions exprimées dans $TQML^w$, nous avons défini le cas minimal et intuitif pour les auto-transformations complexes en proposant une théorie du point fixe, fondée sur la théorie des modèles. Lorsque les caractéristiques non-monotones comme la négation sont combinées avec la récursivité des règles dans une auto-transformation de M-Trace, des techniques bien connues de stratification ont été utilisées [99] et étendues. En se limitant à des transformations stratifiables, nous avons prouvé également que la sémantique proposée permet d'éviter de tels cas et viellé à ce qu'une interprétation unique existe.

5. Enfin, nous avons donné les preuves des théorèmes énoncés dans notre cadre formel. Nous avons montré que l'interprétation des transformations stratifiées est bien définie et sans ambiguïté. Plus important encore, nous avons démontré que la sémantique spécifiée par la théorie des modèles et l'interprétation de point fixe « fait sens » dans le cas de M-Traces infinies, i.e. il doit être possible d'évaluer les requêtes et des transformations de façon continue, de telle manière que les réponses et les M-Traces soient transformées « au fil de l'eau » et ne doivent pas attendre la fin du flux d'observés.

Langage de transformation	**type de transformation**	**mode d'évaluation**	**étude réalisée**
$TQML^-$	Auto-transformation	Continue	Cas étudié faisant partie des auto-transformations complexes définies dans la section 4.11
		Ponctuelle	Cas trivial en utilisant les définitions de la section 4.11 et la relation de satisfaction définie dans la section 4.10.
	Allo-transformation	Continue	Cas étudié dans la section 4.10
		Ponctuelle	Cas étudié dans la section 4.10
$TQML^w$	Auto-transformation	Continue	Cas étudié dans la section 4.11
		Ponctuelle	Cas non étudié puisque trivial en se fondant sur les sections 4.10 et 4.11
	Allo-transformation	Continue	Cas non étudié mais qui peut être considéré comme trivial en comparaison aux auto-transformations.
		Ponctuelle	Cas trivial en utilisant les définitions de la section 4.11 et la relation de satisfaction définie dans la section 4.10.

TABLE 8.1 – Résumé des cas étudiés dans le chapitre 4.

L'ensemble des cas considérés dans cette thèse est résumé dans le tableau 8.1. Nous nous sommes attaqué au cas le plus complexe des auto-transformations complexes qui peut être considéré comme un cadre plus général pour les (allo) transformations. Il serait sans doute possible

99. Il s'agit d'une approche commune et bien connue en programmation logique introduite d'abord dans [ABW88].

de considérer que les autres cas que l'auto-transformation complexe en continue en sont des cas particuliers plus simples et que si le langage $TQML^w$ a une sémantique démontrée pour une auto-transformation en continue, alors il peut s'appliquer à tous les autres cas. Malheureusement, dans le cadre de cette thèse, nous n'avons pas eu le temps d'en faire la démonstration formelle.

8.1.4 Réification du cadre formel avec les langages existants

Suite à notre formalisation, notre troisième contribution a consisté à faire le tour des travaux se rapportant d'une manière directe ou indirecte aux langages pouvant être exploités dans les SBT. Différents formalismes ont été présentés et discutés notamment pour représenter, interroger et transformer des M-Traces.

Les formalismes considérés concernent les langages dans les systèmes de base de données relationnelles (SGBDR), les système de gestion de flux de données (SGFD), les système de traitements d'événements complexes (CEP), les systèmes à base de règles (SBRs) et les entrepôts XML et RDF.

Le tableau 8.2 résume les discussions faites dans le chapitre 5. Les entrées dans le tableau utilisant l'échelle $++$, $+$, 0, $-$, $--$ permettent d'indiquer dans quelle mesure chaque approche prend en compte une certaine caractéristique des M-Traces.

Comme on peut le constater dans le tableau 8.2, aucune de ces technologies ne permet une réification directe de notre sémantique dans les langages proposés. La mise en œuvre de notre cadre formel nécessite la définition d'une sémantique opérationnelle conforme à notre sémantique déclarative. Ce travail a été en partie fait pour une version antérieure de notre langage (non nomotone) présentée dans [SPC+09] dans le cadre du langage Datalog [AHV95]. Cependant, pour la nouvelle version du langage présentée dans cette thèse, il faudra spécifier une sémantique opérationnelle conforme à ces nouvelles exigences et résultats.

8.1.5 Cas d'application du cadre conceptuel des systèmes à base de M-Traces et Implémentation

Pour faire la preuve de l'usage du cadre conceptuel proposé, nous avons présenté deux illustrations de notre approche dans le contexte des EIAH. Nous avons présenté deux utilisations concrètes de la notion de systèmes à base de traces modélisées. Le premier usage concernait l'application dans le cadre d'une plateforme d'apprentissage et de collaboration nommée *eMédiatheque*. Le second usage concerne l'exploitation des traces dans une plateforme d'apprentissage par le jeu nommée le *donjon pédagogique*.

Cependant, ces deux usages à l'instar de la majorité des SBT développés n'implémentent qu'une spécialisation de notre méta-modèle et sont loin d'être génériques. Pour faciliter la mise en place des SBT et ainsi donner les moyens à notre approche de prendre une nouvelle ampleur, un

	SGBDR	SGFD	CEP	SBRs	XML	RDF
Modèles pour les M-Traces						
Hiérarchies de types et de relations	0	0	− (avec des exceptions)	− (conversion sous forme d'objets)	0	++
Aspects Temporels	0	+	0 (avec des exceptions)	−	−	+
Requêtes, transformations et sémantiques pour les M-Traces						
Négation	− (l'aspect temporel)	+	+	− (l'aspect temporel)	− − (l'aspect temporel)	0 (l'aspect temporel)
Occurrence et détection temporelles	−	− (dépend de l'opérateur relation-to-stream)	++	− − (laissé au programmeur)	−	−
Transformation de M-Traces	−	− −	0	+	0	+
Sémantique formelle	+ (précise mais peu intuitive pour les M-Traces)	+ (précise mais peu intuitive pour les M-Traces)	0 (les données des observés ne sont pas considérées)	− (sous forme de programmes impératifs)	+ (peu intuitive pour les M-Traces)	+ (peu intuitive pour les M-Traces)

TABLE 8.2 – Synthèse sur la comparaison entres les différentes technologies pouvant être utilisées pour l'exploitation des M-Traces.

projet de développement d'un framework informatique générique a vu le jour : le projet d'Atelier pour la gestion de Traces, leurs Exploitations et Représentations (ATER). Il s'agissait donc de créer une implémentation générique de SBT qui pourrait offrir des services de manipulation de traces modélisées nécessaires à l'implémentation d'un SBT : collecter, transformer, requêter, visualiser, gérer et partager des traces modélisées. La présentation de cette plateforme en termes de conception et fonctionnalités a été faite dans le chapitre 7.

8.2 Futures travaux et les perspectives de recherche autour des traces

Plusieurs perspectives à différents niveaux peuvent être recensées à la fin de ce travail de thèse : tout d'abord le prolongement concret du travail réalisé sur la formalisation que nous considérons dans une échéance à court terme. Ensuite le prolongement du travail de la communauté « trace » sur des questions de recherche qui nous semblent primordiales pour le futur des exploitations de traces et les prochaines extensions de l'approche des SBT. Cette dernière est discutée dans une perspective d'un travail à long terme.

Perspectives à court terme

La perspective la plus évidente concerne l'intégration dans le cadre formel des allo-transformations complexes. Ce cas de figure, bien que plus simple que celui des auto-transformations complexes, a été omis volontairement dans notre formalisation. En effet, dans notre démarche pour traiter les différents facettes soulevées par les M-Traces, nous avons choisi de nous attaquer au cas le plus complexe (et que nous considérons comme étant le plus général) en nous basant sur plusieurs résultats obtenus et bien établis dans la communauté des langages de représentation de connaissances. Ceci nous permet également de disposer de résultats qui permettront de traiter plus rigoureusement, et en cohérence avec les langages existants, les cas de figure non traités. Nous pensons, en plus des allo-transformations complexes de M-Traces, notamment aux fonctions d'agrégation (`count`, `sum`, etc.) qui présentent probablement le chantier le plus important non traité dans cette thèse. Nous pensons également qu'il est important de lever la restriction aux transformations stratifiables, ce qui a comme conséquence de compliquer davantage l'évaluation des transformations. Nous pensons notamment aux approches définies dans ce sens et qui ont été étudiées en profondeur dans les travaux de recherche liés à la programmation logique et les bases de données déductives comme les modèles bien fondés (*well-founded model*[VGRS91]) ou les modèles stables (*stable models*[GL88]).

Une seconde perspective concerne la définition d'une sémantique opérationnelle de notre langage. L'objectif de la sémantique déclarative que nous avons définie est de décrire la sémantique des transformations de manière précise, formelle et intuitive, ne faisant pas référence à une mise en œuvre concrète du langage. Cette description doit servir de référence pour vérifier l'exactitude et l'exhaustivité des opérationnalisations et des implémentations du langage proposé. Elle doit faire office d'une spécification formelle pour les utilisateurs cherchant à obtenir une compréhension précise du langage proposé. Cependant, pour faciliter les implémentations et les optimisations du langage, il est courant et intéressant de définir des sémantiques opérationnelles conformes à la sémantique déclarative. Cette sémantique opérationnelle se focalisera plus sur les algorithmes et les techniques effectives permettant d'obtenir les résultats escomptés.

La définition d'une telle sémantique opérationnelle reste la perspective la plus intéressante à notre formalisation. En outre, l'effort déployé pour mettre en œuvre cette sémantique peut ne pas être très important. Par exemple, comme nous l'avons mentionné lors de la discussion des langages réifiant les M-Traces et leurs transformations, le cadre théorique des langages RDF et SPARQL se prête naturellement à la définition d'une telle sémantique. D'ailleurs, c'est le cadre qui a été choisi par l'équipe Silex qui s'est déjà engagée dans le développement d'un moteur SBT [CCPS09] basé sur une sémantique opérationnelle (en cours de définition) fondée sur RDF/RDFS et SPARQL.

Concernant toujours cette seconde perspective, on peut également considérer le cadre des bases de données déductives et le langage Datalog [100], que nous avons déjà utilisé et éprouvé

100. Nous sommes notamment intéressé par une forme générale de Datalog communément appelé ASP (Answer Set Programming) [GL91]

pour opérationnaliser une version antérieure de notre langage présentée dans [SPC+09]. Nous tenons à signaler également que cette version et formalisation précédente de notre langage, bien qu'elle ne répondait pas à toutes les exigences relevées par les M-Traces contrairement à la formalisation présentée dans cette thèse, a permis le développement d'un moteur SBT par la société Knowings, un moteur SBT qui fait actuellement partie de son offre commerciale [101]. On peut ainsi espérer dans le future le développement d'un nouveau moteur SBT conforme à notre sémantique déclarative.

Perspectives à long terme

Nous pensons que ce travail de thèse n'est qu'un modeste début et qu'il n'est que le socle de base sur lequel il faudra construire de nouveaux services de raisonnement et de nouvelles techniques d'exploitation. Nous pensons notamment aux travaux de l'équipe Silex sur la découverte de connaissances à partir de M-Traces et à la découverte continue de chroniques étudiée et formalisée dans la thèse de Damien Cram [Cra10]. Il serait intéressant d'étudier l'intégration de ces travaux et plus généralement de l'intégration des travaux de fouille continue dans notre cadre.

Une autre perspective à long terme peut également concerner l'exploitation et l'évaluation de nouveaux types de raisonnement pour les M-Traces. Notre formalisation de l'interrogation et la transformation de M-Traces met en œuvre des mécanismes de raisonnement qui relèvent du domaine des raisonnements déductifs et temporels. Cependant, il existe d'autres différents types de raisonnement [Moh04], qui n'ont pas été abordés : nous pensons notamment à l'étude des capacités du raisonnement par similarité (ou raisonnement par analogie) et le raisonnement inductif afin de compléter et d'étendre les transformations définies pour les M-Traces.

101. http ://www.knowings.com/Nos-Actualites.18+M537a1eb546c.0.html ?&tx_belinknews_pi1[news]=36

Bibliographie

[Aas08] Jans Aasman. Unification of geospatial reasoning, temporal logic, & social network analysis in event-based systems. In *DEBS '08 : Proceedings of the second international conference on Distributed event-based systems*, pages 139–145, New York, NY, USA, 2008. ACM.

[ABB+04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. Stream : The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab, 2004.

[ABW88] K. R. Apt, H. A. Blair, and A. Walker. *Towards a theory of declarative knowledge*, pages 89–148. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[ABW03] A. Arasu, S. Babu, and J. Widom. The cql continuous query language : Semantic foundations and query execution, 2003.

[AC06] Raman Adaikkalavan and Sharma Chakravarthy. Snoopib : interval-based event specification and detection for active databases. *Data Knowl. Eng.*, 59(1) :139–165, 2006.

[ACc+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora : a new model and architecture for data stream management. *The VLDB Journal*, 12(2) :120–139, 2003.

[AE04] Asaf Adi and Opher Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2) :177–203, 2004.

[AFK+07] Nikolaos Avouris, G. Fiotakis, G. Kahrimanis, M. Margaritis, and V. Komis. Beyond logging of fingertip actions : analysis of collaborative learning using multiple sources of data. *Special Issue : Usage Analysis in Learning Systems : Existing Approaches and Scientific Issues*, 18(2) :231–250, April 2007.

[AG08] Renzo Angles and Claudio Gutierrez. The expressive power of sparql. In *International Semantic Web Conference*, pages 114–129, 2008.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11) :832–843, 1983.

[AMK04] Nikolaos Avouris, Meletis Margaritis, and Vassilis Komis. Modelling interaction during small-group synchronous problem-solving activities : The synergo approach. *2nd International Workshop on Designing Computational Models of Collaborative Learning Interaction, ITS2004, 7th Conference on Intelligent Tutoring Systems, Maceio, Brasil*, September 2004.

[AMM+07] Jean-Michel Adam, Sandra Michelet, Christian Martel, Jean-Pierre David, and Guéraud Viviane. Une infrastructure logicielle pour instrumenter l'expérimentation des eiah,. *Actes de la 3ième conférence en Environnement Informatique pour l'Apprentissage Humain (EIAH 2007),*, 2007.

[And98] J. H. Andrews. Theory and practice of log file analysis. Technical report, Technical Report 524, Department of Computer Science, University of Western Ontario, 1998.

[Ara06] Arvind Arasu. *Continuous queries over data streams.* PhD thesis, Thesis (Ph. D.)–Stanford University, 2006.

[AS96] Fabio N. Akhras and John A. Self. A process-sensitive learning environment architecture. In *Intelligent Tutoring Systems*, pages 430–438, 1996.

[AS99] F. N. Akhras and John A. Self. Modeling the process, not the product, of learning. In Sp Lajoie, editor, *Computers as cognitive tools.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1999.

[Bal94] Nicolas Balacheff. *Didactique et Intelligence Artificielle.* Balacheff N. et Vivet M. Eds La pensée sauvage., 1994.

[Bal96] S Balbo. Ema : An automatic analysis mechanism for the ergonomic evaluation of user interface. Technical report, CSIRO-DIT Technical Report, no. 96/44., 1996.

[Bat95] Peter C. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Trans. Comput. Syst.*, 13(1) :1–31, 1995.

[BB07] Michal Barla and Mária Bieliková. Estimation of user characteristics using rule-based analysis of user logs. in : Data mining for user modeling. In *Proc. of Workshop held at the Int. Conf. on User Modeling UM 2007, Corfu*, pages 5–14, 2007.

[BBC+09] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Continuous queries and real-time analysis of social semantic data with c-sparql. In *SDoW2009*, volume 520 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. online http ://ceur-ws.org/Vol-520/paper02.pdf.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS '02 : Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, New York, NY, USA, 2002. ACM.

[BBEP05] James Bailey, François Bry, Michael Eckert, and Paula-Lavinia Patranjan. Flavours of xchange, a rule-based reactive language for the (semantic) web. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, *Rules and Rule Markup Languages for the Semantic Web, First International Conference*, volume 3791 of *Lecture Notes in Computer Science*, pages 187–192. Springer, 2005.

[BCm06] Roger S. Barga and Hillary Caituiro-monge. Event correlation and pattern detection in cedr. In *In Proc. Int. Workshop Reactivity on the Web*, pages 919–930. Springer, 2006.

[BD06] Tharrenos Bratitsis and Angelique Dimitracopoulou. Monitoring and analyzing group interactions in asynchronous discussions with the dias system. In Yannis A. Dimitriadis, Ilze Zigurs, and Eduardo Gómez-Sánchez, editors, *CRIWG*, volume 4154 of *Lecture Notes in Computer Science*, pages 54–61. Springer, 2006.

[BE07] François Bry and Michael Eckert. Rule-based composite event queries : The language xchange^{eq} and its semantics. In *Web Reasoning and Rule Systems, First International Conference, RR 2007*, pages 16–30, 2007.

[Bec04] Dave Beckett. Rdf/xml syntax specification (revised). W3C recommendation, World Wide Web Consortium, 2004.

[Bel90] Victoria Bellotti. A framework for assessing applicability of hci techniques. In *INTERACT '90 : Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 213–218, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co.

[Ber02] Bruno Berstel. Extending the rete algorithm for event management. In *TIME '02 : Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)*, page 49, Washington, DC, USA, 2002. IEEE Computer Society.

[Ber07] Anders Berglund. Xml path language (xpath) 2.0. w3c recommendation, world wide web consortium, 2007. http ://www.w3.org/TR/xpath/, 2007.

[BG04] Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0 : Rdf schema. w3c recommendation, world wide web consortium, 2004.

[BGHS95] Albert Badre, M Guzdial, Scott E. Hudson, and Paulo J. Santos. A user interface evaluation environment using synchronized video, visualizations,and event trace data. *J. of Software Qual.*, 4, 1995.

[BGJ08] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming sparql - extending sparql to process data streams. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 448–462. Springer, 2008.

[BI00] Denis Bouhineau and Jean-François Nicaud. I. mplementation of the algebraic

layer of the new aplusix with rewriting rules,. In *Workshop associé à ITS'2000 "Learning algebra with the computer", Université du Québec à Montréal*, 2000.

[BJM04] Joseph E. Beck, P. Jia, and Jack Mostow. Automatically assessing oral reading fluency in a computer tutor that listens. In *Technology, Instruction, Cognition and Learning*, volume 1, pages 61 – 81, 2004.

[BKF+07] Irina Botan, Donald Kossmann, Peter M. Fischer, Tim Kraska, Dana Florescu, and Rokas Tamosevicius. Extending xquery with window functions. In *VLDB '07 : Proceedings of the 33rd international conference on Very large data bases*, pages 75–86. VLDB Endowment, 2007.

[BKK04] Martin Bernauer, Gerti Kappel, and Gerhard Kramler. Composite events for xml. In *WWW '04 : Proceedings of the 13th international conference on World Wide Web*, pages 175–183, New York, NY, USA, 2004. ACM.

[BL97] Michael Baker and Kristin Lund. Promoting reflective interactions in a computer-supported collaborative learning environment. In *Journal of Computer Assisted Learning*, volume 13, pages 175–193, 1997.

[BL98] Tim Berners-Lee. Notation 3. http ://www.w3.org/DesignIssues/Notation3.html, 1998.

[BL00] Tim Berners-Lee. Primer : Getting into rdf & semantic web using n3. http ://www.w3.org/2000/10/swap/Primer, 2000.

[BM05] Nicolas Balacheff and Alain Mille. Personnalisation des environnements informatiques pour l'apprentissage humain. Document du projet disponible sur http ://cluster-isle-eiah.liris.cnrs.fr/, 2005.

[BM07] Luca Botturi and Riccardo Mazza. Monitoring an online course with the gismo tool : a case study. *Journal of Interactive Learning Research*, 18(2) :251–265, 2007.

[Boa07] Scott Boag. Xquery 1.0 : An xml query language. w3c recommendation, world wide web consortium. http ://www.w3.org/TR/xquery/, 2007.

[BP97] Daniel Billsus and Michael J. Pazzani. Learning and revising user profiles : The identification of interesting web sites. *Machine Learning*, 27 :313–331, 1997.

[Bru01] Peter Brusilovsky. Adaptive hypermedia. *User Model. User-Adapt. Interact.*, 11(1-2) :87–110, 2001.

[BV00] B. Barros and M F Verdejo. Analyzing student interaction processes in order to improve collaboration. the degree approach. *(IJAIED) International Journal of Artificial Intelligence in Education*, 11 :221–241, 2000.

[BW01] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3) :109–120, 2001.

[CA08] Sharma Chakravarthy and Raman Adaikkalavan. Events and streams : harnessing and unleashing their synergy ! In *DEBS '08 : Proceedings of the second international conference on Distributed event-based systems*, pages 1–12, New York, NY, USA, 2008. ACM.

[CAV06] CAViCoLA. Computer-based analysis and visualization of collaborative learning activities. Action du réseau d'Excellence Européen Kaleidoscope, accessible à : http ://www.noe-kaleidoscope.org, consulté le 01/01/08, 2006.

[CBFA96] S. Carroll, S. Beyerlein, M. Ford, and D. Apple. The learning assessment journal as a tool for structured reflection in process education. In *FIE '96 : Proceedings of the 26th Annual Frontiers in Education*, pages 310–313, Washington, DC, USA, 1996. IEEE Computer Society.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. Telegraphcq : Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[CCG05] G Chabert, T Carron, and L. Gagnière. L'observation des usages dans un contexte éducatif : pratiques pédagogiques et nouvelles formes de communication éducative émergentes ? In *Actes du colloque international ISD'05*, 2005.

[CCPS09] Pierre-Antoine Champin, Damien Clauzel, Yannick Prie, and Karim Sehaba. D1.4-a rapport d'implémentation du module de gestion des m-traces. Rapport d'avancement ANR, Janvier 2009.

[CD97] Elizabeth Chang and Tharam S. Dillon. Automated usability testing. In *INTERACT '97 : Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, pages 77–84, London, UK, UK, 1997. Chapman & Hall, Ltd.

[CDD07] B Charlier, N Deschryver, and Peraya Daniel. Apprendre en présence et à distance : Une définition des dispositifs hybrides. In *Revue Distances et Savoirs*, volume 4, 2007.

[CDP06] B. Charlier, N. Deschryver, and D Peraya. Apprendre en présence et à distance - une définition des dispositifs hybrides. In *Dans Distances et Savoirs*, volume 4, pages 489–490., 2006.

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. Niagaracq : a scalable continuous query system for internet databases. In *SIGMOD '00 : Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 379–390, New York, NY, USA, 2000. ACM.

[CFP+04] Corinna Cortes, Kathleen Fisher, Daryl Pregibon, Anne Rogers, and Frederick Smith. Hancock : A language for analyzing transactional data streams. *ACM Trans. Program. Lang. Syst.*, 26(2) :301–338, 2004.

[CGJ+02] Chuck Cranor, Yuan Gao, Theodore Johnson, Vlaidslav Shkapenyuk, and Oliver Spatscheck. Gigascope : high performance network monitoring with an sql interface. In *SIGMOD '02 : Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 623–623, New York, NY, USA, 2002. ACM.

[CGT08] Françoise Le Calvez, Hélène Giroire, and Gérard Tisseau. Design of a learning environment in combinatorics based on problem solving : Modeling activities, problems and errors. *I. J. Artificial Intelligence in Education*, 18(1) :59–94, 2008.

[Cha02] Jean Charlet. L'ingénierie des connaissances : Développements, résultats et perspectives pour la gestion des connaissances médicales. Mémoire d'habilitation à diriger des recherches, Université Pierre et Marie Curie., 2002.

[Cha03] Pierre-Antoine Champin. Ardeco : an assistant for experience reuse in computer aided design. pages 287–294, 2003.

[Cha04] Marie-Anne Chabin. Document trace et document source. la technologie numérique change- t-elle la notion de document ? *Information - Interaction - Intelligence*, 4 :141–158, 2004.

[Che90] Jolly Chen. Providing intrinsic support for user interface monitoring. In *INTERACT '90 : Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 415–420, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co.

[Cho07] Christophe Choquet. Ingénierie et réingénierie des eiah, l'approche redim. Mémoire de Habilitation à Diriger des Recherches en Informatique, Décembre 2007.

[CI06] Christophe Choquet and Sebastien Iksal. Usage tracking language : A metalanguage for modeling tracks in tel systems. In M. Helfert & B. Shishkov (Eds.) In : J. Filipe, editor, *Proceedings of the First International Conference on Software and Data Technologies, ICSOFT'06*, page 133–138. Berlin, Germany : Springer, 2006.

[CI07] Christophe Choquet and Sebastien Iksal. Modeling tracks for the model driven reengineering of a tel system. *(JILR) Journal of Interactive earning Research, Special Issue Usage Analysis in Learning Systems : Existing Approaches and Scientific Issues*, 18(2) :161–184, 2007.

[CJM07] Damien Cram, Denis Jouvin, and Alain Mille. Visualizing interaction traces to improve reflexivity in synchronous collaborative e-learning activities. In Academic Conferences Limited, editor, *6th European Conference on e-Learning*, pages 147–158, October 2007.

[CJSS03] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope : a stream database for network applications. In *SIGMOD '03 : Pro-*

ceedings of the 2003 ACM SIGMOD international conference on Management of data, pages 647–651, New York, NY, USA, 2003. ACM.

[CKAK94] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. Composite events for active databases : Semantics, contexts and detection. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 606–617. Morgan Kaufmann, 1994.

[CL04] Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. In *EMSOFT '04 : Proceedings of the 4th ACM international conference on Embedded software*, pages 147–154, New York, NY, USA, 2004. ACM.

[Cla99] James Clark. Xsl transformations (xslt) version 1.0. w3c recommendation, world wide web consortium,. http ://www.w3.org/TR/xslt, 1999.

[CM94] S. Chakravarthy and D. Mishra. Snoop : an expressive event specification language for active databases. *Data Knowl. Eng.*, 14(1) :1–26, 1994.

[CM01] James Clark and Makoto Murata. Relax ng specification. http ://relaxng.org/. ISO/IEC 19757-2 :2003., 2001.

[CMH08] Thibault Carron, Jean-Charles Marty, and Jean-Mathias Heraud. Teaching with game based learning management systems : Exploring and observing a pedagogical dungeon. In *Simulation & Gaming Special issue on eGame and Adaptative eLearning*, volume 39(3), pages 353–378, Sept 2008.

[CMHF06] Thibault Carron, Jean-Charles Marty, Jean-Mathias Heraud, and Laure France. Helping the teacher to re-organize tasks in a collaborative learning activity : An agent-based approach. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006, 5-7 July 2006, Kerkrade, The Netherlands*, pages 552–554. IEEE Computer Society, 2006.

[Con03] IMS Global Learning Consortium. Ims learning design specification. http ://www.imsglobal.org/learningdesign/index.html, 2003.

[CP02] Pierre-Antoine Champin and Yannick Prié. ModÉliser l'utilisateur ou l'utilisation ? *Document Virtuel Personalisable 2002*, pages 97–102, 2002.

[CPM03] Pierre-Antoine Champin, Yannick Prié, and Alain Mille. Musette : Modelling uses and tasks for tracing experience. In Béatrice Fuchs and Alain Mille, editors, *workshop From structured cases to unstructured problem solving episodes - WS 5 of ICCBR'03*, pages 279–286, Trondheim (NO), 2003.

[CPY98] Ming-Syan Chen, Jong Soo Park, and Philip S. Yu. Efficient data mining for path traversal patterns. *IEEE Trans. on Knowl. and Data Eng.*, 10(2) :209–221, 1998.

[Cra10] Damien Cram. *Découverte interactive de chroniques : application à la co-construction de connaissances à partir de traces.* Thèse de doctorat en informatique, Université Claude bernard Lyon 1, 2010.

[Cyg05] Richard Cyganiak. A relational algebra for SPARQL. 2005.

[Dav99] Brian D. Davison. Web traffic logs : An imperfect resource for evaluation. In *Proceedings of th INET'99 Conference*, June 1999. `http://www.isoc.org/inet99/proceedings/4n/4n1.htm`.

[DBBO96] Pierre Dillenbourg, Mike Baker, Agnes Blaye, and Claire O'Malley. The evolution of research on collaborative learning. In *E. Spada & P. Reiman (Eds) Learning in Humans and Machine : Towards an interdisciplinary learning science*, pages 189–211. Elsevier, Oxford, 1996.

[Des01] Christophe Després. *Modélisation et Conception d'un Environnement de Suivi Pédagogique Synchrone d'Activités d'Apprentissage à Distance.* PhD thesis, Thèse de Doctorat de l'Université du Maine, 2001.

[DGLC07] Gregory Dyke, Jean Jack Girardot, Kristin Lund, and Annie Corbel. Analysing face to face computer-mediated interactions. In *EARLI'07 , Budapest, Hungary*, 2007.

[Die95] Volker Diekert. *The Book of Traces.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.

[Dim04] A. Dimitracopoulou. State of the art on interaction analysis : Interaction analysis indicators. ICALTS JEIRP Deliverable D.26.1. Kaleidoscope network of excellence. Disponible sur internet : www.rhodes.aegean.gr/LTEE/kaleidoscope-icalts., 2004.

[DSP+10] Tarek Djouad, Lotfi-Sofiane Settouti, Yannick Prié, Christophe Reffay, and Alain Mille. Un système à base de traces pour la modélisation et l'élaboration d'indicateurs d'activités éducatives individuelles et collectives. mise à l'épreuve sur moodle. *revue TSI*, March 2010.

[Dyk09] Gregory Dyke. *A model for managing and capitalising on the analyses of traces of activity in collaborative interaction.* PhD thesis, Ecole des Mines de Saint-Etienne, 2009.

[EA08] Prud'hommeaux Eric and Seaborne Andy. Sparql query language for rdf, Junuary 2008.

[Esp] EsperTech. Event stream intelligence : Esper & nesper. http ://esper.codehaus.org.

[FH90] Janet Finlay and Michael D. Harrison. Pattern recognition and interaction models. In Dan Diaper, David J. Gilmore, Gilbert Cockton, and Brian Shackel, editors, *INTERACT*, pages 149–154. North-Holland, 1990.

[FH05] Mingyu Feng and Neil T. Heffernan. Informing teachers live about student learning : Reporting in assistment system. In *Workshop on Usage Analysis in Learning Systems, Proceedings of Artificial Intelligence in Education*, Amsterdam, 2005.

[FHM+06] Laure France, Jean-Mathias Heraud, Jean-Charles Marty, Thibault Carron, and Joseph Heili. Monitoring virtual classroom : Visualization techniques to observe student activities in an e-learning system. In *ICALT*, pages 716–720. IEEE Computer Society, 2006.

[Fis01] Gerhard Fischer. User modeling in human-computer interaction. *User Model. User-Adapt. Interact.*, 11(1-2) :65–86, 2001.

[FM77] Charles Forgy and John P. McDermott. Ops, a domain-independent production system language. In *IJCAI*, pages 933–939, 1977.

[FMCL06] Enrique Frias-Martinez, Sherry Y. Chen, and Xiaohui Liu. Survey of data mining approaches to user modeling for adaptive hypermedia, 2006.

[FMV07] C Ferraris, C. Martel, and L Vignolet. Helping teachers in designing cscl scenarios : a methodology based on the ldl language. In *Computer Supported Collaborative Learning (CSCL)*, 2007.

[For82] Charles Forgy. Rete : A fast algorithm for the many patterns/many objects match problem. *Artif. Intell.*, 19(1) :17–37, 1982.

[FS96] Carolanne Fisher and Penelope Sanderson. Exploratory sequential data analysis : exploring continuous observational data. *interactions*, 3(2) :25–34, 1996.

[Fur08] Tim Furche. *Implementation of Web Query Language Reconsidered : Beyond Tree and Single-Language Algebras at (Almost) No Costs.* PhD thesis, PhD Thesis, Institute for Informatics, University of Munich, 2008.

[FWZ01] A. Foss, W. Wang, and O. R. Zaiane. A non-parametric approach to web log analysis. *Proceedings of Workshop on Web Mining in First International SIAM Conference on Data Mining, Chicago*, pages 41–50, 2001.

[GA02] Antony Galton and Juan Carlos Augusto. Two approaches to event definition. *DEXA '02 : Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 547–556, 2002.

[GAC06] Vihang Garg, Raman Adaikkalavan, and Sharma Chakravarthy. Extensions to stream processing architecture for supporting event processing. In Stephane Bressan, Josef Küng, and Roland Wagner, editors, *Database and Expert Systems Applications*, volume 4080 of *Lecture Notes in Computer Science*, pages 945–955. Springer Berlin / Heidelberg, 2006.

[GAP+04] V. Guéraud, J.-M Adam, J.-P Pernin, G Calvary, and J.-P David. L'exploitation d'objets pédagogiques interactifs à distance : le projet formid. In *STICEF, vol. 11, ISSN : 1764-7223, www.sticef.org.*, 2004.

[Gas03] Katrin Gassner. The influence of knowledge modeling on the communication process. *KI*, 17(1) :31–, 2003.

[GB07] L. Gagnière and M Bétrancourt. La méthode d'allo-confrontation comme incitation métacognitive dans des situations d'apprentissage. In *Actes des 4èmes journées d'étude en Psychologie Ergonomique*, 2007.

[GD92] Stella Gatziu and Klaus R. Dittrich. Samos : an active object-oriented database system. pages 23–39, 1992.

[GD94] Stella Gatziu and Klaus R. Dittrich. Detecting composite events in active database systems using petri nets. In Widom, J. and Chakravarthy, S., editors, *Fourth International Workshop on Research Issues in Data Engineering : Active Database Systems, Houston, Texas, Feb. 14-15, 1994*, pages 2–9. IEEE-CS, 1994.

[Geo08] Olivier Georgeon. *Analyse de traces d'activité pour la modélisation cognitive : application à la conduite automobile*. PhD thesis, Université lyon 2, 2008.

[GGKL07] Sven Groppe, Jinghua Groppe, Dirk Kukulenz, and Volker Linnemann. A SPARQL Engine for Streaming RDF Data. In *Proceedings 3rd International Conference on Signal-Image Technology & Internet-Based Systems (SITIS 2007)*, pages 154–161, Shanghai, China, December 16 - 19 2007. This paper received an honorable mention at the SITIS'07 Conference.

[GHV05] Claudio Gutiérrez, Carlos A. Hurtado, and Alejandro A. Vaisman. Temporal rdf. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web : Research and Applications, Second European Semantic Web Conference*, volume 3532 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005.

[GHV07] Claudio Gutierrez, Carlos A. Hurtado, and Alejandro A. Vaisman. Introducing time into rdf. *IEEE Trans. Knowl. Data Eng.*, 19(2) :207–218, 2007.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs : combining logic programs with description logic. In *WWW '03 : Proceedings of the 12th international conference on World Wide Web*, pages 48–57, New York, NY, USA, 2003. ACM.

[GJS92a] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In *SIGMOD '92 : Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 81–90, New York, NY, USA, 1992. ACM.

[GJS92b] Narain H. Gehani, H. V. Jagadish, and Oded Shmueli. Composite event specification in active databases : Model & implementation. In *VLDB '92 : Proceedings of the 18th International Conference on Very Large Data Bases*, pages 327–338, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[GJS93] Narain H. Gehani, H. V. Jagadish, and Oded Shmueli. Compose : A system

for composite specification and detection. In *Advanced Database Systems*, pages 3–15, London, UK, 1993. Springer-Verlag.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP)*, pages 1070–1080. MIT Press, 1988.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9 :365–385, 1991.

[GM93] G. Gay and J. Mazur. The utility of computer tracking tools for user-centred design. *Educational Technology*, 34 (3)(34) :45–59, 1993.

[GMD+04] P Gray, M Mclead, S Drapear, M Creasz, and R Thomas. A distributed usage monitoring system. In *In Computer-Aided Design of User Interfaces*, pages 121–132, 2004.

[GO03] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2) :5–14, 2003.

[GO05] Lukasz Golab and M. Tamer Özsu. Update-pattern-aware modeling and processing of continuous queries. In *SIGMOD '05 : Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 658–669, New York, NY, USA, 2005. ACM.

[GP00] Charles F. Goldfarb and Paul Prescod. *The XML Handbook (Second Edition)*. 2000.

[GS01] Johannes Gehrke and Praveen Seshadri. Towards sensor database systems. pages 3–14, 2001.

[Guz93] M Guzdial. Deriving software usage patterns from log files. Technical report, Tech. Rept. GIT-GVU-93- 41., 1993.

[HAB04] J. Hardy, M. Antonioletti, and S.P. Bates. Learner tracking : Tools for discovering learner behaviour. *The IASTED International Conference onWeb-based Education (WBE 2004)*, 2004.

[Hay04] Patrick Hayes. Rdf semantics. W3C recommendation, World Wide Web Consortium, 2004.

[HB97] Julie A. Hatcher and Robert G. Bringle. Reflections : Bridging the gap between service and learning. *Journal of College Teaching*, 45 :153–158, 1997.

[HBM04] Cecily Heiner, Joseph E. Beck, and Jack Mostow. Lessons on using its data to answer educational research questions. In *Proceedings of the ITS2004 Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes*, pages 1 – 9, August 2004.

[HDJ+94] P Holt, S Dubs, M Jones, , and J Greer. *The state of student modelling.*, volume NATO-ASI Series F, Springer-Verlag. Student Modelling : The Key to Indivi-

dualized Knowledge-Based Instruction, In Greer, J.E. and MeCalla, G.I.(Eds), 1994.

[Her02] Jean-Mathias Heraud. *PIXED : Une approche collaborative de l'expérience et l'expertise pour guider l'adaptation des hypermedias.* PhD thesis, Université Claude Bernard Lyon1, 2002.

[HFM04] Jean-Mathias Heraud, Laure France, and Alain Mille. Pixed : an its that guides students with the help of learners' interaction log. In *7th International Conference on Intelligent Tutoring Systems (Workshop Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes)*, pages 57–64, 2004.

[HH07] Olaf Hartig and Ralf Heese. The sparql query graph model for query optimization. In *ESWC '07 : Proceedings of the 4th European conference on The Semantic Web*, pages 564–578, Berlin, Heidelberg, 2007. Springer-Verlag.

[HLK+04] Jurgen Heller, Mark Levene, Kevin Keenoy, Cord Hockemeyer, and Dietrich Albert. An e-learning perspective of cognitive and pedagogical aspects of trails. Kaleidoscope Deliverable D22.1.1 Final Version 1.0, July 2004.

[HM02] Jean-Mathias Héraud and Alain Mille. Pixed : une approche collaborative de l'expérience et l'expertise pour guider l'adaptation hypermédia en enseignement à distance. *in Informatique. Université Lyon1 : Villeurbanne*, 2002.

[HM03] Jean-Mathias Héraud and Alain Mille. Pixed : assister l'apprentissage à distance par la réutilisation de l'expérience. *in Proceedings de l'Atelier Raisonnement à Partir de Cas, Plateforme AFIA'03 Laval, L. Jean Editor*, 2003.

[HMFC05] Jean-Mathias Heraud, Jean-Charles Marty, Laure France, and Thibaut Carron. Helping the interpretation of web logs : Application to learning scenario improvement. *Workshop AIED'05, Amsterdam*, 2005.

[HMMD08] Andreas Harrer, Alejandra Martínez-Monès, and Angelique Dimitracopoulou. *Users' Data Collaborative and Social Analysis*, chapter Technology-Enhanced Learning Principles and Products, pages 175–193. Humanities, Social Sciences and Law. Nicolas Balacheff, Sten Ludvigsen,Ton de Jong, Ard Lazonder, Sally Barnes, Lydia Montandon, 2008.

[Hoo97] Kristina Hook. Evaluating the utility and usability of an adaptive hypermedia system. In *IUI '97 : Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 179–186, New York, NY, USA, 1997. ACM.

[HPSB+] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl : A semantic web rule language combining owl and ruleml. http ://www.w3.org/Submission/SWRL/.

[HR00] David M. Hilbert and David F. Redmiles. Extracting usability information from user interface events. *ACM Comput. Surv.*, 32(4) :384–421, 2000.

[HRR97] David M. Hilbert, J.e Robbins, and David F. Redmiles. Supporting ongoing user involvement in development via expectation-driven event monitoring. Technical report, Tech Report UCI-ICS-97-19, Dept. of Information and Computer Science, Univ. of California, Irvine., 1997.

[Hul04] C. D. Hulshof. Log file analysis. In L.Kimberly Kempf (Ed.), Encyclopedia of Social Measurement Elsevier, 577-583., 2004.

[HV02] Annika Hinze and Agnès Voisard. A parameterized algebra for event notification services. In *TIME '02 : Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)*, page 61, Washington, DC, USA, 2002. IEEE Computer Society.

[IA05] IA. Interaction analysis supporting teachers and students self-regulation. http ://www.rhodes.aegean.gr/ltee/kaleidoscope-ia/, 2005.

[ICA04] ICALTS. Interaction & collaboration analysis' supporting teachers & students' selfregulation. http ://www.rhodes.aegean.gr/ltee/kaleidoscope

[ILO] ILOG. Ilog jrules. http ://www.ilog.com/products/jrules.

[JBo] JBoss.org. Drools. http ://www.jboss.org/drools.

[Jer04] Patrick Jermann. *Computer Support for Interaction Regulation in Collaborative Problem-Solving*. PhD thesis, Thèse de doctorat, Genève, 2004.

[JSM01] Patrick Jermann, Amy Soller, and Martin Muhlenbrock. From mirroring to guiding : A review of the state of art technology for supporting collaborative learning. In *Euro-CSCL*, pages 324–331, 2001.

[KAD04] Martha Koutri, Nikolaos Avouris, and Sophia Daskalaki. A survey on web usage mining techniques for web-based adaptive hypermedia systems. In Hershey Idea Publishing Inc., editor, *in S. Y. Chen and G. D. Magoulas (ed), Adaptable and Adaptive Hypermedia Systems*, 2004.

[Kay97] Daniel Kayser. *La représentation des connaissances*. Hermes, 1997.

[Kay07] Michael Kay. Xsl transformations (xslt) version 2.0. w3c recommendation, world wide web consortium,. http ://www.w3.org/TR/xslt20/, 2007.

[KC04] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf) : Concepts and abstract syntax. W3C recommendarion, World Wide Web Consortium, 2004.

[Kim90] Won Kim. *Introduction to object-oriented databases*. MIT Press, Cambridge, MA, USA, 1990.

[KKP01] Alfred Kobsa, Jurgen Koenemann, and Wolfgang Pohl. Personalised hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16(2) :111–155, 2001.

[Kob01] Alfred Kobsa. Generic user modeling systems. *User Model. User-Adapt. Interact.*, 11(1-2) :49–63, 2001.

[Kra07] Jurgen Kramer. *Continuous Queries over Data Streams – Semantics and Implementation*. PhD thesis, In GI-Edition-Lecture Notes in Informatics (LNI) : Ausgezeichnete Informatikdissertationen 2007, 2007.

[KS86] R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1) :67–95, 1986.

[Lab02] Jean-Marc. Labat. Eiah : Quel retour d'informations pour le tuteur ? In *Colloque Technologies de l'Information et de la Connaissance dans l'Enseignement supérieur et l'industrie (TICE'02),*, pages 81–88, 2002.

[Laf09] Julien Laflaquière. *Conception de système à base de traces numériques pour les environnements informatiques documentaires*. Thèse de doctorat en informatique, Université de Technologie de Troyes, January 2009.

[Lar09] Larousse. Larousse.fr - encyclopédie et dictionnaires larousse en ligne. http ://www.larousse.fr/encyclopedie/nom-commun-nom/trace/98060, 2009. En ligne ; Page disponible le 22-juillet-2009.

[LeP05] LePetitRobert. Version électronique du nouveau petit robert, dictionnaire alphabétique et analogique de la langue française, 2005. En ligne ; Page disponible le 22-décembre-2005.

[LIS07] LISTEN. The listen project,. accessible à : http ://www.cs.cmu.edu/ listen/, consulté le 01/02/08., 2007.

[Liu06] Bing Liu. *Web Data Mining, Exploring Hyperlinks, Contents and Usage Data*. Springer, December 2006.

[LL99] Mark Levene and George Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, London, UK, 1999.

[Llo87] J. W. Lloyd. *Foundations of logic programming ; (2nd extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[LMAVMV07] V Luengo, D Mufti-Alchawafa, C Vu Min, and L Vadcard. Design of adaptative surgery learning environment with bayesian network. In *In INTED2007. International Technology, Education and Development Conference. Valence, Espagne.*, 2007.

[LP98] Andreas Lecerof and Fabio Paternò. Automatic support for usability evaluation. *IEEE Trans. Softw. Eng.*, 24(10) :863–888, 1998.

[LPV06] Jean-Marc Labat, Jean-Phillip Pernin, and Guéraud Viviane. Contrôle de l'activit de l'apprenant : suivi, guidage pédagogique et scénarios d'apprentissage. In *In Environnements informatiques pour l'Apprentissage Humain*, page 69–96. IC2 Series. Hermes Lavoisier (eds), 2006.

[LS08] David Luckham and Roy Schulte. Event processing glossary. http ://complexe-vents.com/ ?p=361, May 2008.

[LWZ04] Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Query languages and data models for database sequences and data streams. In *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, pages 492–503. VLDB Endowment, 2004.

[LY08] Yue-Shi Lee and Show-Jane Yen. Incremental and interactive mining of web traversal patterns. *Inf. Sci.*, 178(2) :287–306, 2008.

[MA] Software MS Analog. rulecore(r) complex event processing (cep) server. http ://www.rulecore.com.

[MA01] Jack Mostow and Gregory Aist. Evaluating tutors that listen : An overview of project listen. In K. Forbus andP. Feltovich, editor, *Smart Machines in Education : The coming revolution in educational technology.*, pages 169 – 234. MIT/AAAI Press, 2001.

[Mag96] M.S. Magnusson. Hidden real-time patterns in intra- and inter-individual behavior : Description and detection. *European Journal of Psychological Assessment*, 12(2) :112–123, 1996.

[MALV07] D Mufti Alchawafa, V Luengo, and L Vadcard. Un modèle informatique pour la production de rétroactions épistémiques. l'exemple d'un environnement d'apprentissage en chirurgie,. In *EIAH 2007, Lausanne*, 2007.

[Mar05] Alejandra Martinez. Library of interaction analysis methods. Deliverable of the ICALTS JEIRP., 2005.

[May09] Madeth May. *Using Tracking Data as Reflexive Tools to Support Tutors ans Learners in Distance Learning Situations :an Application to Computer-Mediated Communications.* PhD thesis, Insa de Lyon, 2009.

[MBC+02] Jack Mostow, Joseph E. Beck, R. Chalasani, Andrew Cuneo, and P. Jia. Viewing and analyzing multimodal human-computer tutorial dialogue : A database approach. In *ITS 2002 Workshop on Empirical Methods for Tutorial Dialogue Systems*, June 2002.

[McD86] D McDermott. Tarskian semantics, or no notation without denotation]. pages 167–169, 1986.

[MCS99] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. Creating adaptive web sites through usage-based clustering of urls. In *KDEX '99 : Proceedings of the 1999 Workshop on Knowledge and Data Engineering Exchange*, page 19, Washington, DC, USA, 1999. IEEE Computer Society.

[MD03] Riccardo Mazza and Vania Dimitrova. Coursevis : Externalising student information to facilitate instructors in distance learning. In *Proceedings of the Inter-*

national conference in Artificial Intelligence in Education (AIED 2003), pages 279–286, 2003.

[MD05] Riccardo Mazza and Vania Dimitrova. Generation of graphical representations of student tracking data in course management systems. In *IV '05 : Proceedings of the Ninth International Conference on Information Visualisation*, pages 253–258, Washington, DC, USA, 2005. IEEE Computer Society.

[MD07] Riccardo Mazza and Vania Dimitrova. Coursevis : A graphical student monitoring tool for supporting instructors in web-based distance courses. *International Journal of Man-Machine Studies*, 65(2) :125–139, 2007.

[MDL+00] Bamshad Mobasher, Honghua Dai, Tao Luo, Yuqing Sun, and Jiang Zhu. Integrating web usage and content mining for more effective personalization. In *EC-Web*, pages 165–176, 2000.

[ME01] Douglas Moreto and Markus Endler. Evaluating composite events using shared trees. In *IEE Proceedings — Software*, page 2001, 2001.

[MG00] Pierre-Alain Muller and Nathalie Gaertner. *Modélisation objet avec UML*. Eyrolles, 2000.

[MGP07] Madeth May, Sébastien George, and Patrick Prévôt. Tracking, analyzing, and visualizing learners' activities on discussion forums. In *WBED'07 : Proceedings of the sixth conference on IASTED International Conference Web-Based Education*, pages 649–656, Anaheim, CA, USA, 2007. ACTA Press.

[MGP08] Madeth May, Sébastien George, and Patrick Prévôt. Students' tracking data : An approach for efficiently tracking computer mediated communications in distance learning. In *The 8th IEEE International Conference on Advanced Learning Technologies, ICALT 2008, July 1st- July 5th, 2008, Santander, Cantabria, Spain*, pages 783–787. IEEE, 2008.

[MGRD03] A Martinez, Y Dimitriadisand E Gomez, B Rubia, and P DelaFuente. Combining qualitative evaluation and social network analysis for the study of classroom social interactions. *Computers and Education*, pages 353–368, december 2003.

[MH69] J. Mccarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4 :463–502, 1969.

[MHCF04] Jean-Charles Marty, Jean-Mathias Heraud, Thibaut Carron, and Laure France. A quality approach for collaborative learning scenarios. *Learning Technology Newsletter of IEEE, Vol.6,Issue 4*, pages 46–48, 14 -16 December 2004.

[MHFC07] Jean Charles Marty, Jean-Mathias Heraud, Laure France, and Thibault Carron. Matching the performed activity on an educational platform with a recommended pedagogical scenario : a multi source approach. *(JILR) Journal of Interactive earning Research, Special Issue Usage Analysis in Learning Systems : Existing Approaches and Scientific Issues*, 18(2) :161–184, 2007.

[Mil06a] Alain Mille. From case-based reasoning to traces-based reasoning. *Annual Reviews in Control*, 30(2) :223–232, October 2006. Journal of IFAC.

[Mil06b] Alain Mille. Raisonner partir de l'exprience tracée, "le storytelling : concepts, outils et applications". In *sous la direction de Eddie Soulier, Traité IC2, Série Informatique et SI, Hermes Science*, 2006.

[Min74] Marvin Minsky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[MM04] Frank Manola and Eric Miller. Rdf primer. W3C recommendation, World Wide Web Consortium, 2004.

[MM09] Jean-Charles Marty and Alain Mille. *Analyse de traces et personnalisation des EIAH.* Hermes Sciences Publishing, 2009.

[Moh04] K.P. Mohanan. Types of reasoning :relativizing the rational force of conclusions. http ://courses.nus.edu.sg/course/ellkpmoh/critical/reason.pdf, 2004.

[Mos04] Jack Mostow. Some useful design tactics for mining its data. In *Proceedings of the ITS2004 Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes*, pages 20 – 28, August 2004.

[MP04] Jean-Pierre Meunier and Daniel Peraya. Introduction aux théories de la communication. analyse sémio-pragmatique de la communication médiatique. In *2ème édition revue et augmentée,*, Bruxelles, 2004. De Boeck.

[MP06] Alain Mille and Yannick Prié. Une théorie de la trace informatique pour faciliter l'adaptation dans la confrontation logique d'utilisation/logique de conception. In *Dans 13eme Journées de Rochebrune - Traces, Enigmes, Problèmes : Emergence et construction du sens - Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels*, Rochebrune, 2006.

[MPT99] F. Masseglia, P. Poncelet, and M. Teisseire. Using data mining techniques on web access logs to dynamically improve hypertext structure. *SIGWEB Newsl.*, 8(3) :13–19, 1999.

[MR93] M Macleod and R Rengger. The development of drum : a software tool for video-assisted usability evaluation. In *Paper presented at the BCS HCI Conference*, 1993.

[MS00] E Morse and M Steves. Collablogger : A tool for visualizing groups at work. In *Proceedings of WETICE2000, Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises*, pages 104–109. IEEE Computer Society, 2000.

[MSR07] Anne Meier, Hans Spada, and Nikol Rummel. A rating scheme for assessing the quality of computer-supported collaboration processes. *International Journal of Computer-Supported Collaborative Learning*, 2(1) :63–86, March 2007.

[MsSS97] Masoud Mansouri-samani, Morris Sloman, and Morris Sloman. Gem - a generalised event monitoring language for distributed systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4, 1997.

[MZ97] Iakovos Motakis and Carlo Zaniolo. Temporal aggregation in active database rules. In *SIGMOD '97 : Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 440–451, New York, NY, USA, 1997. ACM.

[NB91] J.S. Noblitt and S.K Bland. Tracking the learner in computer aided langage learning. In *Dans B.F. Freeed(dir), Foreign Langage Acquisition and the classroom.*, pages 120–132, 1991.

[Ngo10] Diem Pham Thi Ngoc. Calcul d'indicateur pour la ré-inigénierie de scénario pédagogique en utilisant utl et dcl4utl. *3 ème Rencontres Jeunes Chercheurs en EIAH*, pages 67–72, 2010.

[Nic94] J-F Nicaud. Modélisation en eiao : les modèles d'aplusix. *In : Didactique et intelligence artificielle, Balacheff N. et Vivet M. (Eds.), La pensée sauvage*, pages 67–112, 14 -16 December 1994.

[NICK09] Diem Pham Thi Ngoc, Sebastien Iksal, Christophe Choquet, and Evelyne Klinger. Utl-cl : A declarative calculation language proposal for a learning tracks analysis process. *Advanced Learning Technologies, IEEE International Conference on*, 0 :681–685, 2009.

[OB06] Magalie Ollagnier-Beldame. *Traces d'interactions et processus cognitifs en activité conjointe : Le cas d'une co-rédaction médiée par un artefact numérique.* PhD thesis, Thèse en Sciences Cognitives de l'Université Claude Bernard (Lyon 1). 247p., 2006.

[OHR94] Gary M. Olson, James D. Herbsleb, and Henry H. Rueter. Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Hum.-Comput. Interact.*, 9(4) :427–472, 1994.

[Pé03] Roger T. Pédauque. Document : forme, signe et relation, les re-formulations du numérique. Archive Ouverte en Sciences de l'Information et de la CommunicationHAL - hal.archives-ouvertes.fr, avril 2003.

[Pat05] Paula-Lavinia Patranjan. *The Language XChange : A Declarative Approach to Reactivity on the Web.* PhD thesis, PhD Thesis (Dissertation). University of Munich, Germany., July 2005.

[Per03] Daniel Peraya. De la correspondance au campus virtuel : formation à distance et dispositifs médiatiques. In D. (Ed.) In Charlier, B. & Peraya, editor, *Technologie et innovation en pédagogie. Dispositifs innovants de formation pour l'enseignement supérieur.*, pages 79–92, Bruxelles, 2003. De Boeck.

[Per05] Jean-Phillip Pernin. Cse, un modèle de traitement de traces. Rapport de recherche CLIPS-IMAG, 2005.

[PHMAZ00] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, and Hua Zhu. Mining access patterns efficiently from web logs. In *PADKK '00 : Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 396–407, London, UK, 2000. Springer-Verlag.

[PKK95] Wolfgang Pohl, Alfred Kobsa, and Oliver Kutter. User model acquisition heuristics based on dialogue acts. In *In International Workshop on the Design of Cooperative Systems*, pages 471–486, 1995.

[PMSMS09] David Peterson, Ashok Malhotra, C. M. Sperberg-McQueen, and Thompson Henry S. W3c xml schema definition language (xsd) 1.1 part 2 : Datatypes, Junuary 2009.

[Pol07] Axel Polleres. From sparql to rules (and back). In *WWW '07 : Proceedings of the 16th international conference on World Wide Web*, pages 787–796, New York, NY, USA, 2007. ACM.

[Pop72] Karl Popper. Philosophical comments on tarski's theory of truth. with Addendum, Objective Knowledge, Oxford : 319-340. Rev. Ed. 1979„ 1972.

[Pos43] Emile .L. Post. Formal reductions of the general combinatorial decision problem. American Journal of Mathematics. 69 : 197–215, 1943.

[PPPS03] Dimitrios Pierrakos, Georgios Paliouras, Christos Papatheodorou, and Constantine D. Spyropoulos. Web usage mining as a tool for personalization : A survey. *User Modeling and User-Adapted Interaction*, 13(4) :311–372, 2003.

[PRR+99] Catherine Plaisant, Anne Rose, Gary Rubloff, Richard Salter, and Ben Shneiderman. The design of history mechanisms and their use in collaborative educational simulations. In *Proceedings of the Computer Support for Collaborative Learning, CSCL'99*, pages 348–359, 1999.

[Prz88] Teodor Przymusinsik. On the declarative semantics of deductive databases and logic programs. *In Jack Minker, editor, Foundations of Deductive Databases and Logic Programming, chapter 5, Morgan Kaufmann,*, page 193–216, 1988.

[PS08] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. W3C Recommendation 15 January 2008, World Wwide Web, 2008.

[PW90] S.G. Paris and P. Winograd. How metacognition can promote academic learning and instruction. In *In Dimensions of Thinking and Cognitive Instruction,*, page 15–51. Beau Fly Jones et Lorna Idol ed., Hillsdale, Lawrence Erlbaum Associates, 1990.

[RC03] C. Reffay and T. Chanier. Mesurer la cohésion d'un groupe d'apprentissage en formation à distance. In *Environnements Informatiques pour l'Apprentissage Humain (EIAH'2003)*, page 367–378, 2003.

[RCNB08] Christophe Reffay, Thierry Chanier, Muriel Noras, and Marie-Laure Betbeder. Contribution à la structuration de corpus d'apprentissage pour un meilleur partage en recherhce. In *Sticef : Sciences et Technologies de l'information et de la communication pour l'education et la formation*, 2008.

[Ren00] D Renié. Apport d'une trace informatique dans l'analyse du processus d'apprentissage d'une langue seconde ou étrangère. In *In Duquette, L. & Laurier, M. (dirs) Apprendre une langue dans un environnement multimédia.*, pages 281–301, Outremont, Canada, 2000. Les Éditions Logiques.

[Ron98] Claudia Roncancio. Toward duration-based, constrained and dynamic event types. In *ARTDB '97 : Proceedings of the Second International Workshop on Active, Real-Time, and Temporal Database Systems*, pages 176–193, London, UK, 1998. Springer-Verlag.

[Ros06] Riccardo Rosati. Integrating ontologies and rules : Semantic and computational issues. In *Reasoning Web*, pages 128–151, 2006.

[RPC08] Bertrand Richard, Yannick Prié, and Sylvie Calabretto. Towards a unified model for audiovisual active reading. In *Tenth IEEE International Symposium on Multimedia.*, pages 673–678, December 2008.

[RTL06] N Roussel, A Tabard, and C Letondal. All you need is log. In *Proc. WWW2006 Workshop on Logging Traces of Web Activity : The Mechanics of Data Collection*, 2006.

[RV07] C. Romero and S. Ventura. Educational data mining : A survey from 1995 to 2005. *Expert Syst. Appl.*, 33(1) :135–146, 2007.

[RVZB09] Cristóbal Romero, Sebastián Ventura, Amelia Zafra, and Paul de Bra. Applying web usage mining for personalizing hyperlinks in web-based adaptive educational systems. *Comput. Educ.*, 53(3) :828–840, 2009.

[SB05] Marco Seiriö and Mikael Berndtsson. Design and implementation of an eca rule markup language. In *Proceedings of International Conference on Rules and Rule Markup Languages for the Semantic Web, Galway, Ireland (10th–12th November 2005)*, volume 3791 of *LNCS*, pages 98–112, 2005.

[SBD+05] Simone Stumpf, Xinlong Bao, Anton N. Dragunov, Thomas G. Dietterich, Jonathan L. Herlocker, Kevin Johnsrude, Lida Li, and Jianqiang Shen. The tasktracker system. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 1712–1713. AAAI Press / The MIT Press, 2005.

[Sch07] Simon Schenk. A sparql semantics based on datalog. In *KI '07 : Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*, pages 160–174, Berlin, Heidelberg, 2007. Springer-Verlag.

[Sel87] J. Self. Student models : what use are they ? In *Proceedings of the IFIP TC3 Working Conference*, 1987.

[Ser02] Alexandre Serres. Quelle(s) problématique(s) de la trace ? Communication du séminaire du CERCOR, 13/12/2002, Traces et corpus dans les recherches en SIC. Sciences de l'Information et de la Communication., 2002. Texte en ligne : http ://archivesic.ccsd.cnrs.fr/sic00001397.htm.

[SF94] Penelope M. Sanderson and Carolanne Fisher. Exploratory sequential data analysis : foundations. *Hum.-Comput. Interact.*, 9(4) :251–317, 1994.

[SGT97] Adelheit Stein, Jon Atle Gulla, and Ulrich Thiel. Making sense of users' mouse clicks : Abductive reasoning. In *and Conversational Dialogue Modeling.. Sixth International Conference on User Modeling. Chia*, pages 89–100. Springer Wien, 1997.

[SLH+07] J Schoonenboom, M Levene, J Heller, K Keenoy, and M Turcsányi-Szabó. Trails in education : Technologies that support navigational learning,. In *In : Rotterdam/Taipei, Sense publishers.*, 2007.

[SLR95] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. Seq : A model for sequence databases. pages 232–239, 1995.

[SMJM05] Amy Soller, Alejandra Martinez, Patrick Jermann, and Martin Muhlenbrock. From mirroring to guiding : A review of state of the art technology for supporting collaborative learning. *IJAIED International Journal of Artificial Intelligence in Education*, 15 (4) :261–290, 2005.

[SNL] Sandia-National-Laboratories. Jess, the rule engine for the java(tm) platform. http ://www.jessrules.com/.

[SPC+09] Lotfi Sofiane Settouti, Yannick Prié, Pierre-Antoine Champin, Jean-Charles Marty, and Alain Mille. A trace-based systems framework : Models, languages and semantics. http ://hal.inria.fr/inria-00363260/en/, 2009.

[SPMM07] Lotfi Sofiane Settouti, Yannick Prié, Jean-Charles Marty, and Alain Mille. Vers des systèmes à base de traces modélisées pour les eiah. Technical Report RR-LIRIS-2007-016, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/Ecole Centrale de Lyon, May 2007.

[SPMM09] Lotfi-Sofiane Settouti, Yannick Prié, Jean-Charles Marty, and Alain Mille. A trace-based system for technology-enhanced learning systems personalisation. In *The 9th IEEE International Conference on Advanced Learning Technologies*, July 2009.

[SR90] Young-Chul Shim and C. V. Ramamoorthy. Monitoring and control of distributed systems. pages 672–681, 1990.

[SS05] J. Stefanov and E. Stefanova. Analysis of the usage of the virtuoso system. In *AIED workshop on Usage analysis in learning systems,*, 2005.

[SSSM05] César Sánchez, Matteo Slanina, Henny B. Sipma, and Zohar Manna. Expressive completeness of an event-pattern reactive programming language. In *In Proc. Int.*

Conf. on Formal Techniques for Networked and Distrib. Systems, pages 529–532. Springer, 2005.

[SW04] Utkarsh Srivastava and Jennifer Widom. Flexible time management in data stream systems. In *PODS '04 : Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 263–274, New York, NY, USA, 2004. ACM.

[SWM04] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide. W3C recommendation, World Wide Web Consortium., 2004.

[SZZA01] Reza Sadri, Carlo Zaniolo, Amir Zarkesh, and Jafar Adibi. Optimization of sequence queries in database systems. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 71–81, New York, NY, USA, 2001. ACM.

[SZZA04] Reza Sadri, Carlo Zaniolo, Amir Zarkesh, and Jafar Adibi. Expressing and optimizing sequence queries in database systems. *ACM Trans. Database Syst.*, 29 :282–318, June 2004.

[Tar33] Alfred Tarski. The concept of truth in the languages of the deductive sciences. (Polish), Prace Towarzystwa Naukowego Warszawskiego, Wydzial III Nauk Matematyczno-Fizycznych 34, Warsaw ; reprinted in Zygmunt 1995, pp. 13–172 ; expanded English translation in Tarski 1983, pp. 152–278., 1933.

[TCG+93] Abdullah Uz Tansel, James Clifford, Shashi Gadia, Sushil Jajodia, Arie Segev, and Richard Snodgrass, editors. *Temporal databases : theory, design, and implementation.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.

[Tch02] Pierre Tchounikine. Pour une ingénierie des envrionnements informatiques pour l'apprentissage humain. *Revue I3*, 2(1) :59–95, 2002.

[TGNO92] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases. In *SIGMOD '92 : Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 321–330, New York, NY, USA, 1992. ACM.

[TRA04] TRAILS. Personalised and collaborative trails of digital and non-digital learning objects. accessible à : http ://www.dcs.bbk.ac.uk/trails/index.html, 2004.

[Van88] K. VanLehn. Student modeling. *In M. C. Polson & J. J. Richardson (Eds.), Foundations of intelligent tutoring systems, awrence Elbaum Associates Publishers.*, pages 55–78, 1988.

[VEK76] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4) :733–742, 1976.

[VGRS91] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3) :619–649, 1991.

[WBG08] Karen Walzer, Tino Breddin, and Matthias Groch. Relative temporal constraints in the rete algorithm for complex event detection. In *DEBS '08 : Proceedings of the second international conference on Distributed event-based systems*, pages 147–155, New York, NY, USA, 2008. ACM.

[WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD '06 : Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418, New York, NY, USA, 2006. ACM.

[Wen87] Etienne Wenger. *Artificial intelligence and tutoring systems : computational and cognitive approaches to the communication of knowledge.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[Wik09] Wikipédia. Trace — wikipédia, l'encyclopédie libre. http ://fr.wikipedia.org/wiki/Trace, 2009. En ligne ; Page disponible le 22-juillet-2009.

[WM06] A. Weerasinghe and A. Mitrovic. Facilitating deep learning through self-explanation in an open-ended domain. In *International journal of Knowledge-based and Intelligent Engineering systems, IOS Press.*, 2006.

[WPB01] Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2) :19–29, 2001.

[WRGD07] Walker White, Mirek Riedewald, Johannes Gehrke, and Alan Demers. What is "next" in event processing ? In *PODS '07 : Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 263–272, New York, NY, USA, 2007. ACM.

[WX04] W3C-XML. Extensible markup language (xml) 1.1. w3c recommendation. http ://www.w3.org/TR/2004/REC-xml11-20040204/, February 2004.

[WXS01] W3C-XML-Schema. Xml schema 1.1. http ://www.w3.org/XML/Schema, 2001.

[Yac05] Kalina Yacef. The logic-ita in the classroom : A medium scale experiment. *I. J. Artificial Intelligence in Education*, 15(1) :41–62, 2005.

[YJGMD97] Tak W Yan, Matthew Jacobsen, Hector Garcia-Molina, and Umeshwar Dayal. From user access patterns to dynamic hypertext linking. Technical report, Stanford, CA, USA, 1997.

[Zai02] O.R. Zaiane. Building a recommender agent for e-learning systems. In *In Proceedings of the 7th International Conference on Computers in Education (ICCE02)*, 2002.

[ZML+06] Jian Zhou, Li Ma, Qiaoling Liu, Lei Zhang, Yong Yu, and Yue Pan. Minerva : A scalable owl ontology storage and inference system. In Riichiro Mizoguchi,

Zhongzhi Shi, and Fausto Giunchiglia, editors, *The Semantic Web – ASWC 2006*, volume 4185 of *Lecture Notes in Computer Science*, pages 429–443. Springer Berlin / Heidelberg, 2006.

[ZS01] Dong Zhu and Adarshpal S. Sethi. Sel, a new event pattern specification language for event correlation. *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 586 – 589, 2001.

[ZS02] Yunyue Zhu and Dennis Shasha. Statstream : statistical monitoring of thousands of data streams in real time. In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 358–369. VLDB Endowment, 2002.

[ZZ01] Osmar Zaane and Osmar R. Zaïane. Web usage mining for a better web-based learning. In *In Proc. of Conference on Advanced Technology for Education*, pages 60–64, 2001.

Résumé

Ce travail de thèse s'inscrit dans le cadre du projet « personnalisation des environnements informatiques pour l'apprentissage humain (EIAH) » financé par la région Rhône-Alpes. La personnalisation des EIAH est essentiellement dépendante de la capacité à produire des traces pertinentes et exploitables des activités des apprenants interagissant avec un EIAH. Dans ce domaine, l'exploitation des traces relève explicitement plusieurs problématiques allant de sa représentation de manière normalisée et intelligible à son traitement et interprétation en temps différé ou en temps réel au moment même de l'apprentissage. La multiplication des pratiques et des usages des traces requiert des outils génériques pour soutenir leurs exploitations. L'objectif de cette thèse est de définir les fondements théoriques et pratiques de tels outils génériques permettant l'exploitation des traces d'interaction. Ceci nous a amené à définir la notion de Systèmes à Base de Trace modélisées : une classe de systèmes à base de connaissances facilitant le raisonnement et l'exploitation des traces modélisées. L'approche théorique proposée pour construire de tels systèmes s'articule autour de deux contributions : (1) La définition d'un cadre conceptuel définissant les concepts, l'architecture et les services mobilisés par les SBT. (2) La définition d'un cadre formel pour les systèmes à base de traces modélisées. Plus précisément, la proposition d'un langage pour l'interrogation et la transformation de trace modélisées à base de règles permettant des évaluations ponctuelles et continues. La sémantique formelle de ce langage est définie sous forme d'une théorie des modèles (dans le style de Tarski) et d'une théorie de point fixe, deux formalismes habituellement utilisés pour décrire la sémantique formelle des langages de représentation de connaissances. Nous avons présenté également deux utilisations concrètes de notre approche. Le premier usage concerne une plateforme d'apprentissage et de collaboration nommée *eMediatheque* pour permettre la réflexivité des activités des apprenants. Le second usage concerne l'exploitation des traces pour l'enseignant dans le cadre d'une plateforme d'apprentissage par le jeu nommée le *donjon pédagogique*.

Abstract

This thesis is funded by the Rhône-Alpes Region as a part of the project « Personalisation of Technology-Enhanced Learning (TEL) Systems ». Personalising TEL Systems is, above all, dependent on the capacity to produce relevant and exploitable traces of individual or collaborative learning activities. In this field, exploiting interaction traces must deal with several problems ranging from its representation in a normalised and intelligible manner to its processing and interpretation in continuous way during the ongoing TEL activity. The proliferation of exploitations of traces has raised the need of generic tools to support their representation and manipulation. The main objective of this thesis is to define the theoretical and practical foundations of such generic tools. To do that, we have defined the notion of Trace-Based System (TBS) as a kind of knowledge-based system whose main source of knowledge is a set of trace of user-system interactions. This thesis investigates practical and theoretical issues related to TBS framework, covering the spectrum from concepts, services and architecture involved by such systems (conceptual framework) to language design over declarative semantics (formal framework). The central topic of our framework is the development of a high-level trace transformation language supporting deductive rules as an abstraction and reasoning mechanism for traces. The declarative semantics for such language is defined by a (Tarski-style) model theory with accompanying fixpoint theory. We illustrate the proposed framework in the context of two TEL systems : (1) A synchronous and collaborative e-learning platform called *eMediatheque* implementing the concept of virtual classrooms. TBS is exploited to enable reflexivity of learners activities. (2) A Game Based Learning Management System called *The pedagogical dungeon* where the TBS is employed to enable teacher to be aware of the ongoing activities.