Université de Lyon

École doctorale Informatique et Mathémathiques

Découverte interactive et complète de chroniques : application à la co-construction de connaissances à partir de traces

THÈSE

présentée et soutenue publiquement le 30 septembre 2010

pour l'obtention du

Doctorat de l'Université de Lyon I

(spécialité informatique)

par

Damien Cram

Composition du jury

Président:

Rapporteurs:

Fabien Gandon	HDR, chargé de recherche, INRIA Sophia Antipolis
Maguelonne Teisseire	Directeur de Recherche UMR TETIS - CEMAGREF

Examinateurs:

Mireille Ducassé	Professeur des universités, INSA Rennes
Florence Le Ber	HDR, chargé d'enseignement et de recherche, ENGEES Strasbourg

Invité:

Patrick Michels	PDG de Knowings SA

Directeur de thèse:

Alain Mille	Professeur des universités, Université Claude Bernard Lyon I

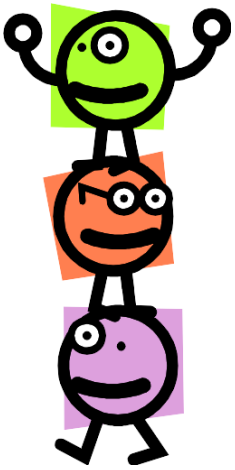
Laboratoire d'InfoRmatique en Images et Systèmes d'information — UMR 5205
Univ. Claude Bernard - Bât. Nautibus - 43, bd du 11 novembre 1918 - 69622 Villeurbanne cedex

FINANCÉ PAR ANR

Cette thèse a été financée par l'Agence Nationale pour la Recherche (ANR)
dans le cadre du projet :

PROCOGEC

PROgiciels COllaboratifs de GEstion des Connaissances



(http://www.procogec.com)

Cette thèse a fait l'objet d'un stage doctoral de six mois à l'étranger au laboratoire DFKI, financé par la région Rhône-Alpes dans le cadre de la bourse EXPLO'RA DOC 2008.

Un grand merci au laboratoire DFKI de Saarbrücken (Allemagne) pour leur participation financière à ce stage doctoral, et particulièrement à Alexander Kröner et toute l'équipe SEMPROM du DFKI, pour leur accueil chaleureux.

German Research Center for Artificial Intelligence

Remerciements

En premier lieu, je tiens à remercier Alain Mille, à qui je dois énormément dans cette thèse, pour ses conseils, pour ses orientations de recherche, pour le temps et la confiance qu'il m'a accordés, pour son écoute dans les moments difficiles, pour sa patience, et pour avoir su maintenir une ambiance de travail agréable entre nous dans les moments plus délicats.

Merci également à toute l'équipe du projet ProCoGeC, à Knowings en particulier pour leur collaboration très efficace, à Philippe et à Mathieu pour le temps et l'énergie qu'ils m'ont accordés, et aussi à Élise, avec qui nous avons franchi les mêmes étapes aux mêmes moments.

Je remercie le laboratoire LIRIS de m'avoir accueilli pour réaliser mon travail de thèse. Un grand merci également au laboratoire DFKI pour leur participation financière à mon stage doctoral, et particulièrement à Alexander Kröner, Michael Schneider et toute l'équipe SemProM du DFKI, pour leur accueil chaleureux.

Merci à tous les membres de l'équipe SILEX pour leurs retours et conseils variés tout au long de cette thèse, et tout particulièrement à Amélie pour son aide précieuse et continue, et sans qui je serais encore à la rédaction de mon premier chapitre.

Merci à toutes les personnes avec qui j'ai partagé ces trois années et qui les ont rendues très agréables : mes co-bureaux Catalin, Denis et Magali, les co-silex, les co-procogec, les co-liris, les co-dfki, etc.

Du fond du coeur, je remercie tous mes proches pour leur soutien. J'envoie un clin d'oeil affectueux à ma filleule fraîchement baptisée, et à son papa qui m'a fait découvrir et aimer la recherche.

Résumé

Cette thèse se situe dans le cadre de l'ingénierie de la dynamique des connaissances et s'intéresse plus particulièrement à la découverte interactive de connaissances dans les traces d'interactions. La gestion de la dynamique des connaissances liée à la mise en place d'un environnement de gestion de connaissances constitue le cadre applicatif principal du travail. Les contributions théoriques concernent d'une part la proposition d'un processus de co-construction de connaissances exploitant les capacités d'apprentissage automatique de la machine et les capacités d'interprétation de l'utilisateur et d'autre part une contribution algorithmique permettant d'exploiter de manière interactive un processus de découverte dans des séquences temporelles d'événements.

Les traces d'interactions sont des informations que les utilisateurs d'un système informatique laissent lors de leurs activités. Ces informations sont collectées volontairement ou non par le concepteur du système. Lors de la collecte, elles sont représentées dans un format expressif dédié à l'ingénierie des traces, le format des traces modélisées, et sont accessibles par l'intermédiaire d'un système de gestion des traces (SBT) qui gère leur stockage. Nous argumentons que ces traces d'interactions sont des conteneurs de connaissances riches en informations contextuelles et qu'il est possible de les utiliser pour inférer des connaissances pertinentes sur l'activité tracée et exploitables par des systèmes d'assistance à l'utilisateur. Nous proposons un processus de co-construction de connaissances à partir de traces, qui est itératif et interactif. L'humain et la machine jouent tour à tour un rôle dans la construction des connaissances : la machine propose des motifs de comportement de l'utilisateur à partir des traces et l'humain valide ces motifs s'il les reconnaît et les juge intéressants. Dans le cas contraire, il formule de nouvelles requêtes à la machine qui lui propose alors de nouveaux motifs, et ainsi de suite. L'idée est d'implémenter un processus de construction de connaissances ascendant qui prenne en compte les aspects dynamique et contextuel de la connaissance. Pour que la machine puisse jouer un tel rôle pro-actif dans la construction, il faut concevoir un algorithme d'extraction de motifs temporels à partir de traces qui soit complet et qui permette de fournir des motifs en temps réel à l'humain, de sorte que le processus prenne la forme d'un dialogue avec la machine.

Une chronique est une structure de motif spécifiant des contraintes temporelles numériques. L'algorithme d'extraction de chroniques fréquentes que nous présentons dans cette thèse pour implémenter ce processus est le premier algorithme d'extraction complète de chroniques à partir de séquences d'événements. Il permet l'interactivité en temps réel avec son utilisateur en affichant les résultats partiels de l'extraction à tout moment. L'algorithme supporte l'intégration de plusieurs types de contraintes temporelles et structurelles permettant à l'utilisateur de faire converger la découverte plus rapidement vers les chroniques d'intérêt. L'algorithme se comporte comme un framework dans la mesure où il peut être configuré pour agir comme les algorithmes d'extraction de chroniques non complets existants, pour découvrir l'ensemble véritablement complet des chroniques fréquentes, ou encore l'ensemble complet des épisodes hybrides fréquents, une certaine forme résumée et simplifiée des chroniques. Lorsqu'il est comparé aux algorithmes existants dans les mêmes conditions, notre algorithme montre des performances tout à fait comparables. L'inconvénient du problème de découverte de chroniques est que l'espace d'exploration s'agrandit exponentiellement avec la longueur des chroniques, si bien qu'il n'est possible de découvrir que des chroniques de faibles longueurs, introduisant la nécessité de réaliser la découverte de manière incrémentale.

La plate-forme `Scheme Emerger`, développée dans le cadre de cette thèse, implémente cet algorithme et une interface graphique de pilotage. `Scheme Emerger` illustre le processus de co-construction de connaissances proposé sur des traces d'activités collaboratives collectées dans la plate-forme `CollaborativeECM`, développée dans le cadre du projet PROCOGEC.

Mots-clés

Gestion de la dynamique de la connaissance, découverte interactive de connaissances, chroniques, traces d'interactions

Abstract

This thesis deals with the engineering of knowledge dynamics and it focuses on the interactive discovery of knowledge from activity traces. The applicative context targeted by this work is the management of the dynamic aspect of knowledge in Knowledge Management Systems (KMS). Two theoretical contributions are presented in this thesis. Firstly, we propose an iterative and interactive process for the co-construction of dynamic knowledge that requires a dialogue and a cooperation of the machine and humans. Secondly, we present an algorithm for the complete discovery of temporal patterns in sequences of events. This algorithm implements the machine proactive behaviour in this process.

Interaction traces are information that users leave when they interact with their environment. This information about users' activities is collected, sometimes intentionally, by the designer of the environment. Interaction traces are represented in an expressive format designed especially for the engineering of interaction traces: the format of modelled traces. Such interaction traces are managed separately in a Trace-Based System (TBS), which can store modelled traces and provides primitive functions to access them. We argue that such interaction traces are potential containers of contextual knowledge about how users behave in their activities mediated by the traced environment. For this reason, interaction traces can be used for building systems that provide contextual assistance to users. We propose an iterative and interactive process for the co-construction of knowledge from traces. In this process, the machine analyses the traces and suggests some behaviour patterns to the human involved in the process. The human validates these patterns if he finds them relevant. If it is not the case, the human elaborates new requests and the machine suggests new candidate patterns, and so on. The idea behind this process was to build a bottom-up knowledge construction approach that takes into account the dynamic and contextual aspects of knowledge. The proactive participation of the machine to this co-construction process implies/requires the development of an algorithm that can extract temporal pattern from interaction traces, that is complete, and that can provide patterns to the human in real time, so that the knowledge co-construction process takes the form of a dialogue between the human and the machine.

Chronicles are patterns that can occur in interaction traces and that contain temporal constraints with numerical bounds. The frequent chronicle mining approach we present in this thesis has been designed to implement the machine's behaviour in this process. This algorithm is the first algorithm for chronicle extraction from a sequence of events that is complete. It allows real time interactivity with its users by returning the partial result set of frequent chronicles, at any time. The algorithm supports temporal and structural user constraints pushing, which allows the human to make the chronicle exploration procedure converge more quickly towards the most interesting chronicles. The algorithm can be configured in a way that makes it return the same non-complete chronicle result set as other existing algorithms in the literature. It can also be configured so as to return the complete frequent chronicle set, or to return the complete set of frequent hybrid episodes. Hybrid episodes are summarized forms of chronicles, with a simpler pattern structure that is easier to understand by humans. When compared to existing chronicle mining algorithms with the same conditions, our algorithm shows equivalent time performances. The main inconvenient of the chronicle discovery problem is that the size of the exploration space depends exponentially on the chronicle length. As a result, it is possible to discover only small chronicles in one shot, which implies the need for an iterative and incremental discovery approach.

The platform `Scheme Emerger` has been developed for the purpose of this PhD project. It implements our complete chronicle discovery algorithm and it provides a graphical user interface for it. We use the platform `Scheme Emerger` to illustrate our knowledge co-construction process with interaction traces of the platform `CollaborativeECM`, which is the collaborative platform that has been developed within the project PROCOGEC.

Keywords

Engineering of knowledge Dynamics, Interactive knowledge discovery algorithm, chronicles, interaction traces

Table des matières

Table des figures

Liste des tableaux

Introduction

Les activités humaines sont largement médiées par les environnements informatiques, que ce soit pour des activités professionnelles ou privées. Chaque activité individuelle étant singulière par nature, l'environnement informatique n'a pas été spécifiquement conçu pour elle, et l'utilisateur exploite les différentes possibilités qui lui sont offertes pour faciliter les différentes tâches constitutives de ses activités. L'appropriation de ces multiples tâches en un tout cohérent pour une activité particulière peut se révéler complexe du fait qu'elles se réalisent en successions d'actions simples à exécuter par l'ordinateur, mais dont le séquencement échappe à l'assistance, faute de modèle de l'activité. Prenons l'exemple de la rédaction d'un rapport au moyen d'un logiciel de traitement de texte. Si l'utilisateur de ce logiciel décide de mettre en italique toutes les occurrences d'un même mot dans son rapport, alors il devra probablement effectuer les actions suivantes : rechercher la première occurrence du mot, puis le mettre en italique, puis rechercher l'occurrence suivante du mot, puis le mettre en italique, et ainsi de suite. Il faut noter que l'on pourrait utiliser un registre de description différent et dire qu'il doit effectuer les actions suivantes : sélectionner l'outil de recherche d'un mot, indiquer le mot cherché, sélectionner l'action de lancement de recherche de l'occurrence suivante, etc. Pour l'utilisateur, ces répétitions de motifs d'actions sont laborieuses et il aimerait sans doute que l'ordinateur intègre son intention de mettre toutes les occurrences d'un même mot en italique comme un service à réaliser globalement. Peut-être que dans le futur, une nouvelle version du logiciel de traitement de texte permettra de le faire, mais les services de ce genre seraient si variés, multiples et s'exprimant dans des registres tellement différents qu'il est illusoire d'imaginer qu'un concepteur les prévoira et les rendra faciles à repérer et à utiliser. Cela est d'autant plus vrai que les usages diffèrent d'un utilisateur à l'autre, d'un logiciel à l'autre, d'une version à l'autre, etc. Les besoins de nouveaux services peuvent se révéler progressivement et le cycle de développement logiciel classique, qui vise à répondre aux besoins généraux, échouera à produire des applications adaptées à chacun. L'environnement informatique doit donc être capable d'intégrer des connaissances sur les usages de chaque utilisateur et permettre des adaptations pour les prendre en compte.

Permettre à la machine d'apprendre des usages individuels pour en découvrir les connaissances utiles à une adaptation progressive est un enjeu important pour la production d'environnements qui soient perçus comme fonctionnant en *intelligence* avec les intentions de l'utilisateur. Toute stratégie pour concevoir de tels environnements nécessite d'acquérir les modèles de connaissances associés. De tels modèles qui régiraient le comportement « intelligent » de l'environnement, s'ils s'appliquaient de la même manière pour tous les utilisateurs, ne pourraient pas convenir à tous les contextes individuels. Pour permettre l'apprentissage progressif de ces connaissances, des techniques d'apprentissage automatique peuvent être mises en place au fil de l'utilisation de l'environnement par son utilisateur. Par exemple, certains logiciels de traitement de texte créent des réseaux bayésiens comme modèles de connaissance pour décider quelles fonctionnalités apparaîtront dans les menus en fonction du contexte. Au fil de l'utilisation du logiciel de traitement de texte, le réseau

bayésien est mis à jour en selon les fonctionnalités utilisées par l'utilisateur. Les éléments qui sont visibles dans les menus évoluent selon ces probabilités conditionnelles d'usage. Le problème avec de tels systèmes est que les différents usages du logiciel ne sont pas différenciés, et si l'utilisateur utilise le même traitement de texte pour écrire des courriers, pour faire un rapport, pour réaliser des étiquettes, il est très probable que les menus disparaissant pour l'une des tâches seraient vraiment intéressants pour d'autres et vice-versa. Par ailleurs, la *logique* d'adaptation échappe complètement à l'utilisateur qui n'a aucun moyen de la contrôler. De façon générale, pour s'approprier au mieux son environnement de travail, l'utilisateur a besoin de comprendre et de s'approprier le comportement de l'environnement en toutes circonstances. En l'absence d'une compréhension d'une logique du comportement de l'environnement et en l'absence d'explication sur les écarts au comportement régulier, l'utilisateur sera encore plus déstabilisé par son environnement que s'il n'avait aucune forme de comportement « intelligent » (agissant à la place de l'utilisateur).

Cette thèse s'inscrit dans le cadre de la conception d'environnements capables d'apprendre des usages propres à chaque utilisateur et de les exploiter en « situation » pour assister l'activité, en particulier lorsque cette activité est collaborative et en cours de construction. L'idée directrice qui est à la source de notre démarche de recherche est que les connaissances nécessaires à l'appropriation individuelle et collective ne peuvent ni ne doivent être construites par le concepteur/utilisateur seul ni ne peuvent et ne doivent être apprises automatiquement par l'environnement seul. Nous proposons une approche de construction de connaissances contextuelles, appelées *signatures de tâche*, qui doit être partagée entre l'humain et la machine, d'où le terme « co-construction » employé dans le titre. Ce partage du processus de construction entre humain et machine vise à exploiter au mieux les points forts des deux parties, c'est-à-dire la capacité de calcul et d'exploration systématique de la machine et la capacité d'interprétation de l'humain des informations produites par la machine. Nous proposons que cette co-construction de connaissances soit alimentée par les traces d'interactions collectées dans l'environnement. L'originalité de notre approche de construction de connaissances est que les traces d'interactions, ne sont pas collectées pour découvrir la signature d'une tâche précise, mais pour permettre la découverte de différentes signatures de tâches que les individus et le collectif cherchent à stabiliser. Dans notre approche, les traces d'interactions sont collectées sans *a priori* sur les connaissances qu'elles permettront de construire, même si naturellement le choix des interactions observées est associé implicitement à une classe d'activité. Ces traces d'interactions décrivent l'histoire d'utilisation de l'environnement par son utilisateur de manière non complètement anticipée et peuvent servir à construire des connaissances portant aussi bien sur l'utilisation des fonctionnalités sélectionnées dans les applications que sur toute autre tâche que l'utilisateur mettrait en oeuvre par une combinaison des possibilités offertes. Pour implémenter ce processus de co-construction de connaissances, il faut doter la machine de la capacité d'explorer les traces d'interactions et d'en extraire des informations qui seront proposées à l'humain comme candidates à devenir des connaissances pour représenter les signatures de tâche. Nous proposons un algorithme qui implémente un comportement pro-actif de la machine dans un tel processus de construction des connaissances. Le formalisme choisi pour la représentation des signatures de tâche est le formalisme des *chroniques*. Les *chroniques* sont des motifs temporels représentant l'apparition de certains événements dans la trace, dans un ordre spécifié par des contraintes temporelles numériques. L'algorithme que nous proposons est, à notre connaissance, le premier algorithme complet d'extraction de chroniques fréquentes à partir de séquences d'événements. Cet algorithme a été conçu pour permettre le plus de contrôle possible par son utilisateur, de sorte qu'il puisse faire converger l'algorithme vers les chroniques potentiellement intéressantes. Les leviers de contrôle sont : l'intégration de contraintes spécifiées par l'humain sur les chroniques recherchées, l'ordre dans lequel les chroniques seront explorées, et le type de contraintes temporelles données en entrée

de l'algorithme. Le problème d'extraction complète de chroniques étant très complexe, nous proposons à l'utilisateur d'avoir accès rapidement aux premières chroniques fréquentes découvertes, de sorte qu'il puisse décider d'interrompre la découverte à tout moment pour modifier la requête ou se satisfaire des chroniques trouvées jusqu'à lors.

Les contributions concernent deux domaines de recherche, étroitement articulés, de la communauté scientifique. La problématique initiale est posée dans le domaine de l'*Ingénierie des Connaissances* (IC) et vise à mettre en place un processus de co-construction de connaissances contextuelles et adaptées au besoin de chaque utilisateur d'un environnement. Le processus que nous proposons nécessite une participation pro-active de la machine dans la construction des connaissances, et pour cela nous présentons un travail algorithmique qui s'inscrit dans le domaine de la *Découverte de Connaissances à Partir de Données* (KDD[1]).

Le **chapitre 1** présente le contexte d'étude de la thèse, le projet PROCOGEC, et situe la problématique de recherche dans le domaine de l'IC. Ce chapitre rappelle brièvement les enjeux de l'IC et parcourt certains de ses travaux en mettant en avant les problématiques liées à l'aspect dynamique de la connaissance. Il pose également les conditions d'une ingénierie de la dynamique des connaissances, c'est-à-dire d'une ingénierie de la connaissance qui prend en compte l'aspect contextuel et évolutif de la connaissance.

Le **chapitre 2** met le focus sur les approches d'ingénierie des connaissances par les traces et sur les processus de construction de connaissances mettant en jeu à la fois l'humain et la machine. Dans ce chapitre, nous proposons un processus interactif de co-construction de connaissances à partir de traces d'interactions, auquel participent l'humain et la machine. Nous donnons ensuite le cahier des charges que doit remplir l'algorithme d'extraction de motifs temporels à partir de traces implémentant le comportement pro-actif de la machine.

Le **chapitre 3** entre dans le domaine de la découverte de connaissances à partir de données. Il dresse l'état de l'art des techniques existantes dans la littérature pour l'extraction de motifs temporels à partir de données temporelles. Ce domaine est parcouru du point de vue des propriétés attendues de l'algorithme implémentant le comportement du système défini en conclusion du chapitre 2. Le chapitre 3 se conclut sur une synthèse des techniques d'extraction abordées et propose d'orienter le travail algorithmique pour obtenir le comportement pro-actif du système à des fins d'extraction de chroniques à partir de traces.

Le **chapitre 4** détaille l'algorithme d'extraction complète de chroniques à partir de traces qui implémente le comportement pro-actif de la machine pour une co-construction des connaissances. Ce chapitre fournit les clés de conception de l'algorithme, prouve sa complétude, étudie sa complexité et propose une évaluation de celui-ci sur des jeux de traces divers.

Le **chapitre 5** concerne la mise en application de l'algorithme proposé sur des traces d'interactions réelles, c'est-à-dire collectées à partir d'environnements tracés et notamment à partir de l'environnement d'étude du projet PROCOGEC. Le problème de la préparation des traces collectées en séquences d'événements adaptées pour l'algorithme est posé et nous proposons des stratégies et des outils pour faciliter ce prétraitement. Nous présentons la plate-forme `Scheme Emerger`, que nous avons développée dans le cadre de la thèse dans le but d'implémenter le processus de co-construction de connaissances, et nous illustrons la mise en oeuvre de ce processus avec la découverte de chroniques à partir des traces d'interactions issues de l'environnement d'étude du projet PROCOGEC.

Le **chapitre 6** revient sur les contributions présentées dans cette thèse, tant dans le domaine de la découverte de connaissances à partir de données qu'en ingénierie des connaissances, et propose une discussion critique de ces contributions par rapport aux objectifs initiaux et en relève les

[1] *Knowledge Discovery from Databases*

limitations. Enfin, nous dressons les perspectives ouvertes par ce travail de thèse.

Chapitre 1

Ingénierie de la dynamique des connaissances

Sommaire

L'ingénierie des connaissances est une discipline qui s'est imposée dans les années 70 avec les questions de l'acquisition et de la représentation des connaissances pour la réalisation de « Systèmes à Base de Connaissances ». Cette discipline s'est développée en étendant la problématique au contexte du web, et en prenant en compte des sources de connaissances sans cesse plus nombreuses (bases de données, bases documentaires, interactions, ontologies, etc.) pour des tâches de plus en plus diverses (aide à la décision, gestion de connaissances, diagnostic, classification, recherche d'informations, etc.). Le projet PROCOGEC, contexte d'étude de cette thèse, se développant dans le cadre de la gestion des connaissances, inscrit cette problématique dans son programme de recherche. Ce premier chapitre vise à présenter le projet PROCOGEC, ses objectifs en matière de gestion flexible des connaissances et leurs implications en matière d'ingénierie de la dynamique des connaissances, cadre de travail qui motive le développement algorithmique d'ingénierie de l'expérience présenté dans cette thèse. Afin de définir ce que nous appelons l'Ingénierie de la Dynamique des Connaissances (IDC), ce chapitre parcourt brièvement le domaine de l'ingénierie des connaissances et dresse les questions de recherche concernant l'aspect dynamique de la connaissance.

1.1 Contexte d'étude : le projet PROCOGEC

Ce travail de thèse a été financé dans le cadre du projet PROCOGEC[2]. PROCOGEC (PROgiciels COllaboratifs de GEstion de Connaissances) est un projet financé par l'Agence Nationale de la Recherche[3] (ANR). Il a pour motivation générale de concevoir et développer des plates-formes capables de médier efficacement les activités collaboratives, c'est-à-dire les activités réalisées en collaboration par un ensemble d'individus regroupés autour de la réalisation d'un projet commun, c'est pourquoi nous nous plaçons dans le cadre de l'émergence de communautés de pratique.

1.1.1 Les communautés de pratique

Le terme *communauté de pratique* désigne un groupe de personnes en interaction qui partagent un intérêt ou une passion pour une certaine pratique. Plus précisément, Wenger (2007) définit les communautés de pratique de la manière suivante :

> Communities of practice are groups of people who share a concern or a passion for something they do and learn how to do better as they interact regularly.

Comme le dit Wenger, les communautés de pratique ne sont pas un phénomène nouveau, même si ce terme est relativement récent. Depuis longtemps, les personnes portant un intérêt à un domaine commun ont cherché à se regrouper pour échanger et partager leur savoir-faire. Par exemple, cette volonté d'échanger peut se traduire par l'organisation de conférences ou l'édition de revues.

Le fonctionnement des communautés de pratique a été bouleversé par l'arrivée des technologies numériques, qui permettent aux individus de se regrouper en réseau facilement et leur offrent de nouveaux moyens d'interagir à distance. Depuis plusieurs années, Des réseaux d'humains se forment spontanément sur le Web pour l'entraide et le partage des connaissances et des informations relatives à des pratiques communes, comme le Club des développeurs francophones[4], Marmiton[5], la Guitar Pickers Association[6], etc. Le terme « communauté de pratique » semble donc plutôt désigner ces communautés développées à l'ère du numérique.

Selon Wenger, les communautés de pratique se caractérisent par l'existence d'un domaine commun à tous les individus (e.g. le développement, la guitare picking, la cuisine, etc.), l'engagement de chaque individu dans la communauté (activités communes, discussion, entraide, partage d'information, etc.) et surtout le fait que chaque individu de la communauté est un praticien du domaine partagé. Les communautés de pratique engagent donc des connaissances de type « savoir-faire ».

Selon cette définition, l'intérêt principal des individus à se regrouper en communauté de pratique, ou d'un individu à intégrer une communauté de pratique existante, est l'apprentissage rapide de connaissances sur le domaine et la pratique. Concrètement, une communauté de pratique peut être concernée par plusieurs activités : la résolution de problème, la documentation, la coordination, le partage d'expériences, etc. Dans leur livre blanc sur les communautés de pratiques visant à assister leur développement dans les entreprises et grandes organisations, Parot et al. (2004) identifient trois types de communautés de pratique. Les communautés de pratique thématiques, probablement la catégorie la plus proche de la définition de Wenger, permettent à ses membres de partager les

[2] http://www.procogec.com/
[3] http://www.agence-nationale-recherche.fr/
[4] http://www.developpez.com/
[5] http://www.marmiton.org/
[6] Association des guitaristes de guitare picking : http://www.gpassociation.com/

connaissances qui resteraient inaccessibles à l'individu sans la démarche collective. Chaque individu tire personnellement profit de cette mise en commun. Les communautés de pratique d'innovation et de progrès sont des communautés d'action commune et de mise en commun des ressources individuelles pour servir et faire progresser un domaine commun unique. Ces communautés fonctionnent plus grâce à la collaboration qu'au simple partage. Enfin, les communautés de pratique projet concernent tout à la fois le partage des connaissances des individus et les collaborations entre individus. Elles ont pour but de réaliser ensemble un objectif commun et les collaborations entre individus relèvent plus de la coordination. Elles s'écartent un peu de la définition de communauté de pratique de Wenger en ce qu'il n'y a pas forcément de domaine unique partagé par tous les individus. En effet, un projet de publication d'une nouvelle version d'un produit informatique peut par exemple nécessiter une action coordonnée de plusieurs domaines comme le marketing et le développement informatique. À part cette absence de domaine unique partagé par tous les individus, ce type de communauté correspond en tout point à une communauté de pratique.

1.1.1.1 Cycle de vie d'une communauté de pratique

Selon Wenger (1998), la vie d'une communauté de pratique peut se décomposer en cinq phases distinctes et successives dans le temps : (cf. figure 1.1)

1. la *communauté potentielle*, durant laquelle les individus résolvent les mêmes problèmes sans partager pour autant les mêmes pratiques,
2. la *coalition des individus*, durant laquelle les individus se trouvent et se reconnaissent mutuellement,
3. la *communauté active*, durant laquelle les individus exercent leurs pratiques collectivement ou avec un souci d'intérêt collectif,
4. la *dispersion*, durant laquelle les individus contribuent de moins en moins,
5. la *communauté remémorable*, durant laquelle plus aucune activité n'est effectuée. Les individus se remémorent la communauté dans leurs nouvelles activités comme un référentiel commun ou comme partie de leur identité.

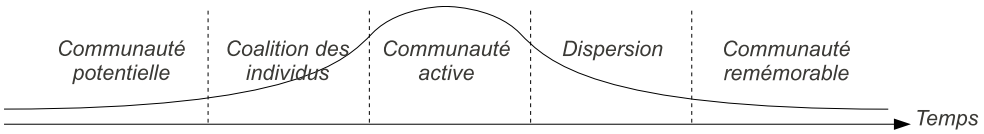


Figure 1.1 – Étapes de développement d'une communauté de pratique, tiré de Wenger (1998)

Les problématiques de gestion des connaissances et d'ingénierie des connaissances qui se posent aux communautés de pratique diffèrent selon la phase dans laquelle se situe une communauté.

1.1.1.2 Problèmes liés à la dynamique d'une communauté de pratique

Le cadre dans lequel les individus d'une communauté de pratique coopèrent et travaillent n'est pas préexistant à cette communauté. Au contraire, ce sont aux membres de cette communauté de faire émerger leurs règles. Ce cadre commun de travail est créé lors de la phase de coalition des individus et évolue ensuite de façon continue, car la communauté vit et intègre sans-cesse

de nouveaux membres, de nouveaux besoins, de nouvelles connaissances et pratiques, en même temps que d'autres disparaissent. Les nouveaux arrivants peinent à s'intégrer dans la communauté, à en appréhender les règles, la philosophie et les éléments moteur. Cette difficulté d'intégration devient un problème pour ces communautés qui sont demandeuses de nouvelles connaissances et de contributions spontanées de la part des individus et peut causer leur dispersion. Un autre obstacle aux communautés de pratique, qui cause généralement leur dispersion, est la présence de redondances voire d'incohérences dans les connaissances véhiculées, si bien que ces communautés de pratiques tendent à souffrir d'un manque d'identité avec le temps, ce qui éloigne leurs membres. De plus, les nouveaux arrivants éprouvent des difficultés à identifier rapidement les connaissances qu'ils sont venus chercher ou l'expert possédant les connaissances intéressantes et ne trouvent alors pas d'intérêt à intégrer cette communauté.

1.1.2 Le projet PROCOGEC : objectifs et enjeux

Le consortium du projet PROCOGEC est né de la volonté de développer une génération de plates-formes de gestion de contenus qui favorisent le partage des connaissances et les collaborations entre individus d'une communauté de pratique, tout en prenant en compte la dynamique de ces communautés. Le constat était que les solutions mises à disposition pour la gestion des contenus en entreprise et assister les collaborations étaient trop souvent parcellaires, ou se focalisaient essentiellement sur le simple partage de document, et n'intégraient pas pleinement l'aspect *collaboratif* des activités qu'elles médiaient. Or cet aspect collaboratif de la gestion des contenus se retrouve dans de nombreux besoins soulevés par les entreprises et organisations dans leurs gestions internes : développer l'efficacité collective, faire exprimer les connaissances, retrouver les connaissances adaptées à une situation donnée, diffuser une information pertinente pro-activement, trouver les bons acteurs face à une situation, animer les communautés, etc.

Le consortium du projet s'est focalisé sur trois objectifs précis instanciant cet objectif général :

1. implémenter des processus souples et des interfaces flexibles pour faire face à l'évolution des communautés de pratique,

2. utiliser les traces d'interactions comme source de connaissances pour une ingénierie de la dynamique des connaissances de la plate-forme,

3. permettre à l'utilisateur de mieux adhérer aux processus de la communauté.

Le partenaire Knowings[7], pilote du projet PROCOGEC, a développé à l'occasion de ce projet le socle logiciel flexible de la plate-forme qui médie les activités collaboratives répondant au premier objectif. Ce socle logiciel s'appelle `CollaborativeECM`. Il est *flexible* dans la mesure où il implémente un très grand nombre de fonctionnalités collaboratives de manière générique et nécessite d'être instancié pour chaque nouvelle communauté. Pour ce faire, des interfaces graphiques propres à chaque communauté (avec bien sûr des possibilité de réutilisation) sont réalisées relativement rapidement et simplement par des *informaticiens consultants* à l'aide d'un certain nombre de technologies souples et dynamiques (e.g. `Javascript`, `FreeMarker`[8], etc.). Ces informaticiens consultants sont au contact permanent de la communauté, connaissent leurs pratiques, s'intéressent à leurs besoins et utilisent ces technologies souples pour réagir très rapidement à une nouvelle demande (sans compilation et sans redémarrage du serveur). Cette association « socle logiciel générique + fichiers de spécialisation à la communauté » constitue la plate-forme de chaque communauté, ces « fichiers

[7] éditeur de plate-forme de gestion collaborative de contenus (`http://www.knowings.com`)

[8] `freemarker.sourceforge.net/`

de spécialisation » agrégeant diverses fonctionnalités collaboratives disponibles dans le socle logiciel au travers des interfaces qu'ils proposent.

Ce socle logiciel permet également le traçage des interactions entre la plate-forme et l'utilisateur, c'est-à-dire qu'il est doté d'un module indépendant et facilement configurable capable de générer des informations en temps réel sur les opérations réalisées par la plate-forme en réponse aux actions des individus dans la plate-forme. Ces informations sont envoyées à un système tiers dédié à la gestion des traces, le SBT (cf. section 2.1.3), et fournissent une description de l'histoire de l'activité des individus à partir de leurs traces d'interactions. Le module de traçage a été implémenté par Knowings sur la base des modèles théoriques (théorie des traces modélisées et Système à Base de Traces, cf. chapitre 2) proposés par le partenaire LIRIS (Laboratoire d'InfoRmatique en Image et Systèmes d'information).

Le deuxième point a été étudié plus particulièrement dans le cadre de la thèse présentée ici. Il consiste à supporter le processus l'émergence de nouvelles pratiques et de nouvelles connaissances dans la communauté, puis à être capable de modifier les modèles de connaissance existants pour intégrer ces nouvelles connaissances. Nous reviendrons plus en détails dans la section 1.4 sur les objectifs de ce travail de thèse liés à cette ingénierie de la dynamique des connaissances.

Le troisième point a fait l'objet de deux directions de recherche dans le projet PROCOGEC. Premièrement, la visualisation interactive des traces d'interactions des utilisateurs par eux-mêmes permet à chaque individu d'observer graphiquement sur une ligne de temps les actions qu'ils a accomplies en comparaison avec celles de ses collaborateurs. Un prototype de visualisation interactive des traces d'interactions a été livré et testé avec succès auprès des terrains d'application qui ont installé la plate-forme `CollaborativeECM`. Non seulement cette visualisation permet d'augmenter la réflexivité des individus (Cram et al. 2007*b*) sur leur activité et l'*awareness* par rapport à l'activité du groupe, mais en plus les éléments visualisés permettent de reproduire des actions passées et de retrouver par ce biais des contenus égarés dans la plate-forme. Deuxièmement, la conception et le développement d'indicateurs de collaboration permettent à l'utilisateur de mieux se situer dans les processus de collaboration et d'évaluer les processus de collaboration mis en place. Ce deuxième volet du troisième point a donné lieu à la création et l'implémentation d'un framework de de calcul d'indicateurs de collaboration à partir de traces (Gendron et al. 2009).

1.2 Ingénierie des connaissances

Afin de mieux comprendre les objectifs généraux d'ingénierie de la dynamique des connaissances (IDC) qui motivent le travail de cette thèse, cette section vise à présenter les principaux concepts de l'ingénierie des connaissances.

1.2.1 Connaissances

Dans la littérature sur l'ingénierie des connaissances, les définitions de la *connaissance* se formulent généralement en fonction des concepts de *données* et d'*information* (cf. figure 1.2). Les données sont les listes de faits bruts alors que les informations sont des données qui ont été transformées en une structure susceptible d'être compréhensible par un humain dans un contexte donné. Enfin, la connaissance est le résultat de la compréhension d'une ou plusieurs informations sur un domaine (Kendal & Creen 2006). Ce qui fait la particularité d'une connaissance, c'est qu'elle permet à un agent, par exemple un humain, d'agir dans une situation donnée, c'est-à-dire d'effectuer des raisonnements menant à une prise de décision (Russell & Norvig 2003). Kayser (1997) définit également

la connaissance comme la capacité d'utiliser l'information pour agir : « il n'y a présomption de connaissance que si la faculté d'utiliser des informations à bon escient est attestée ».

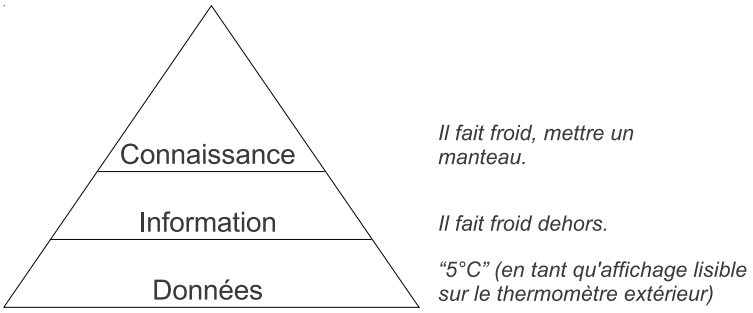


Figure 1.2 – Données, information et connaissance (Kendal & Creen 2006)

La figure 1.2 donne un exemple de la différence entre données, information et connaissance. Pour un opérateur en Bourse, un « trader », les données qui servent à son travail sont les courbes de valeurs des différentes actions, c'est-à-dire les séquences de nombres représentant les valeurs en fonction du temps pour chaque action. Pour cet opérateur, les informations seraient les statistiques sur chaque valeur ou leurs tendances, ou la tendance moyenne pour toutes les actions d'un secteur particulier, ou encore n'importe quelle autre forme résumée de l'évolution de ces actions. Parmi toutes ces informations, certaines s'élèvent au rang de connaissances pour cet opérateur, ce sont celles qui concernent l'entreprise sur laquelle il travaille, ou qui sont relatives au secteur d'activité sur lequel l'opérateur travaille. Une information relative au cour du blé deviendra probablement connaissance dans l'esprit de l'opérateur qui travaille sur une entreprise distributeur de boîtes de céréales puisque cet opérateur agira probablement en fonction de cette information, mais cela restera une simple information pour l'opérateur spécialiste des sociétés de service informatique et encore plus pour toute personne ne travaillant pas en relation avec la Bourse, comme un coiffeur.

Wilson (2002) ajoute un élément de différentiation entre information et connaissance. Pour Wilson, le propre de la connaissance est également d'exister dans l'esprit humain et dans l'esprit humain seulement. Au contraire, les informations sont des formes codées des connaissances présentes dans un esprit humain. Si un humain souhaite échanger ou partager une connaissance, il ne pourra le faire que par l'intermédiaire d'un langage, le plus souvent le langage naturel, en projetant cette connaissance dans ce langage sous forme d'informations. Rien ne peut prouver que la connaissance comprise par l'interlocuteur soit la même. L'information en tant que forme codifiée de la connaissance, c'est aussi ce que Bachimont (2004) appelle « inscription numérique de connaissance » en ingénierie des connaissances, car la « connaissance ne peut s'appréhender techniquement, c'est-à-dire via une ingénierie, qu'à travers les inscriptions qui l'expriment ».

En réalité, cette dimension humaine de la connaissance décrite par Wilson, présentée dans le cadre de l'étude de la gestion des connaissances, est largement violée par de nombreux travaux de la littérature scientifique, notamment dans les domaines de l'apprentissage automatique et de l'ingénierie des connaissances où l'on parle souvent de « connaissance » pour désigner une structure formelle (par exemple une règle ou un réseau bayésien) permettant à la machine d'effectuer des inférences, puis de prendre des décisions, d'agir... Dans la plupart des cas, ces « connaissances machine » sont compréhensibles par tout humain formé à l'interprétation de ces connaissances. C'est le cas par exemple lorsque les connaissances sont modélisées sous forme d'un réseau bayésien et d'une base de règles logiques. Dans d'autres cas, ces connaissances machine ne sont pas compréhensible par l'humain. Par exemple dans un réseau de neurones, ce sont les fonctions d'entrée, les

fonctions d'activation et les poids du biais (Russell & Norvig 2003) qui représentent en interne les connaissances machine. En effet, un être humain serait incapable d'agir dans le domaine concerné au vu des coefficients et fonctions. Pourtant ce sont bien des connaissances pour la machine au sens où la machine est capable d'utiliser ces coefficients et fonctions pour effectuer des tâches d'apprentissage, de classification, de reconnaissance, etc.

Par abus de langage, dans le contexte de ce mémoire, nous distinguerons les *connaissances humaines* et les *connaissances machine*. Les connaissances humaines sont celles qui sont compréhensibles et utilisables par l'homme et les connaissances machine sont celles qui sont utilisables par la machine grâce à un moteur d'inférence interne. Par analogie avec les connaissances humaines, on pourrait dire que les connaissances machine sont des informations « comprises » par la machine. Ces deux catégories ne sont pas disjointes puisqu'un certain nombre de représentations des connaissances sont « comprises » par l'homme et la machine.

1.2.2 Variabilité et dynamique de la connaissance

Une des propriétés de la connaissance est son aspect variable. Cette variabilité de la connaissance, nous la décrivons ici selon trois dimensions : interindividuelle, contextuelle et temporelle. Cette variabilité vaut aussi bien pour les connaissances humaines que pour les connaissances machine.

Ce que nous appelons variabilité interindividuelle réfère à ce qu'une connaissance valable pour un individu ne le sera pas forcément pour un autre. La description d'un domaine donné est différente d'un humain à l'autre. Ainsi la connaissance de ce domaine varie en fonction du point de vue de chacun. Par exemple, la connaissance qu'un individu a d'une voiture Renault Mégane donnée ne sera probablement pas la même selon que cet individu soit propriétaire de cette voiture depuis plusieurs années, garagiste, ou encore concepteur chez Renault. Il y aura probablement un noyau de connaissances commun à ces trois individus concernant cette voiture (e.g. elle est composée de quatre roues, d'un volant, d'un mécanisme de régulation de vitesse, etc.) mais au-delà de ce noyau, seul le garagiste saura que cette voiture dispose d'une boîte de vitesse particulièrement fragile, alors que le concepteur connaîtra le temps de réponse du système de freinage automatique, tandis que le propriétaire saura que l'amortisseur arrière droit grince après quelques kilomètres. Avec ces trois niveaux de connaissances différents sur la voiture, ces trois individus seraient probablement amenés à établir trois diagnostics totalement différents les uns des autres si on leur demandait de trouver la cause d'une même panne donnée. La variabilité de la connaissance n'est pas qu'une question de « rôle » de l'individu vis-à-vis de l'objet. En effet, deux garagistes différents n'auront pas la même expérience de ce modèle de voiture et auront probablement des connaissances différentes à son sujet.

Par variabilité contextuelle, nous entendons la variabilité selon le contexte d'utilisation de la connaissance. Si l'on demandait au même individu de décrire le même domaine pour deux contextes d'utilisation différents, les bases de connaissances qui en résulteraient seraient également très différentes. Par exemple, le diagramme de classes UML d'une voiture sera totalement différent selon que l'on conçoit une application de gestion des pannes ou bien une application de vente de voitures. Dans le premier cas, le diagramme comporterait probablement des informations très techniques (numéro de modèle du carburateur, type d'embrayage, etc) alors que dans le second les informations contenues dans le diagramme seraient celles ayant un impact immédiat sur le prix et sur le choix du client (nombre de kilomètres, couleur, présence d'air conditionné, revêtement des sièges, etc.).

La connaissance présente également une variabilité temporelle car pour un individu donné et un contexte d'utilisation donné une connaissance peut être différente à deux moments différents.

Par exemple, à un moment t_1 donné, un garagiste sera amené à considérer des fausses pistes de diagnostic, voire à changer des pièces inutilement, face à une panne donnée. Si à la date t_2, la même voiture lui est présentée avec la même panne, et que le constructeur a publié entre temps un communiqué signalant un défaut menant à ce type de panne, alors sa connaissance de la voiture aura changé, l'amenant à faire le bon diagnostic tout de suite.

Dans le monde médical, il en va de même pour un médecin habitué à prescrire tel traitement contre telle pathologie jusqu'au jour où une étude ou découverte remette en cause sa connaissance de la maladie. Pour mieux voir cette variabilité temporelle chez le médecin, on pourrait lui demander de traduire ses connaissances de la pathologie sous la forme d'une base de règles de décisions de type Mycin (Buchanan & Shortliffe 1984). On constaterait que les deux bases de règles, avant et après la découverte scientifique, seraient différentes. Il serait peut-être plus juste d'appeler cette variabilité temporelle de la connaissance « évolution de la connaissance » ou encore « dynamique de la connaissance », car généralement l'état des connaissances à une date t_2 ultérieure à t_1 n'est pas en tout point différent de l'état des connaissances à la date t_1. Au contraire, les connaissances à la date t_2 sont les connaissances à la date t_1 altérées par l'apprentissage et l'oubli.

1.2.3 Gestion des connaissances

La gestion des connaissances est une discipline vieille de plusieurs décennies, ayant fait l'objet de diverses études et étant aujourd'hui entrée dans les pratiques courantes de nombreuses entreprises et organisations (Wilson 2002). Cette discipline a une portée non seulement scientifique mais également industrielle, si bien que nous nous autorisons à accepter la définition de Wikipedia comme référentiel. Wikipedia définit la gestion des connaissances comme suit (Wikipédia 2010) : « la gestion des connaissances (en anglais Knowledge Management) est l'ensemble des initiatives, des méthodes et des techniques permettant de percevoir, d'identifier, d'analyser, d'organiser, de mémoriser, et de partager des connaissances entre les membres des organisations, en particulier les savoirs créés par l'entreprise elle-même (ex : marketing, recherche et développement) ou acquis de l'extérieur (ex : intelligence économique) en vue d'atteindre l'objectif fixé. »

L'objectif de la gestion des connaissances pour une organisation telle qu'une entreprise ou une communauté de pratique est d'améliorer l'efficacité des activités en tirant profit de toutes les connaissances présentes en son sein. Les connaissances dont on parle généralement en gestion des connaissances sont des connaissances humaines. De nombreux problèmes se posent lors de la gestion des connaissances. L'un d'eux est que les connaissances présentes au sein d'une organisation sont de diverses natures. Selon la littérature associée, celles-ci peuvent être généralement explicites, implicites. L'enjeu de la gestion des connaissances est alors de mettre en place des politiques permettant d'organiser les connaissances explicites et de faciliter leur accès à tous, favorisant l'explicitation des connaissances stratégiques qui sont à l'état implicite.

Selon Wilson, le terme « gestion des connaissances » n'est pas bien choisi et ne correspond pas à la réalité des activités. En pratique, ce qui est étudié dans cette discipline sont en fait les formes extérieures des connaissances, c'est-à-dire les informations. Le seul qui soit capable de gérer les connaissances est celui qui les possède, c'est-à-dire l'individu. Il ne peut donc pas y avoir de gestion commune et partagé des connaissances puisque les connaissances sont propres à chacun. La seule politique commune possible est la gestion des informations, au sens où nous avons défini l'information en section 1.2.1, c'est-à-dire au sens des formes externalisées et codées des connaissances. Il serait alors plus juste de parler de « gestion des informations ». Dans les entreprises, ces informations étant généralement codifiées sous forme de documents, on pourrait parler de « gestion des documents ». D'ailleurs, les plates-formes informatiques réalisées dans l'optique de faciliter et

d'assister la gestion des connaissances, comme `CollaborativeECM`, sont généralement appelées plates-formes de « *Gestion Électronique Documentaire* » (GED) ou plates-formes de « *Gestion de Contenus* ». Pour la suite, nous retiendrons cependant le terme le plus communément utilisé pour décrire cette discipline, c'est-à-dire la « gestion des connaissances ».

Ce que l'on retiendra de la gestion des connaissances est qu'elle concerne principalement les connaissances que nous avons qualifiées d'humaines et qu'elle concerne pour une grande partie l'étude et la prise en compte des facteurs humains et sociaux lors de la réutilisation de compétences et de savoirs présents au sein de l'organisation.

1.2.4 Ingénierie des connaissances

Au contraire de la gestion des connaissances, la discipline que nous nommons « ingénierie des connaissances » dans ce mémoire concerne principalement les connaissances que nous avons qualifiées de « connaissances machine ». Si nous effectuons ici cette distinction entre gestion des connaissances et ingénierie des connaissances, c'est qu'il n'existe pas de frontière consensuelle dans la littérature scientifique. Parfois l'un désigne ce que nous avons défini ici comme étant l'autre (par exemple la communauté scientifique francophone intitulée *Extraction et Gestion des Connaissances* sous-entend par « gestion « ce que nous appelons « ingénierie »), et d'autres fois, les deux termes peuvent être utilisés de manière interchangeable. Il convient donc de préciser à chaque emploi de l'un de ces deux termes quelle en est la portée.

Ce qu'il ressort de la littérature scientifique, c'est que l'ingénierie des connaissances peut être définie comme un processus de transfert des connaissances humaines vers une certaine forme de base de connaissances (Kendal & Creen 2006). Dans leur état de l'art sur le sujet, Studer et al. (1998) retiennent également prioritairement cette notion de *transfert* comme définition du domaine. Autrement dit, l'ingénierie des connaissances consisterait à traduire, à implémenter, des connaissances humaines en connaissances machine, le but étant de donner les moyens à la machine d'agir dans un contexte donné.

1.2.4.1 Les activités de l'ingénierie des connaissances

Le domaine d'ingénierie des connaissances regroupe plusieurs activités autour de la manipulation des connaissances. Kendal & Creen en voient cinq :

- l'*acquisition des connaissances* consiste à obtenir des connaissances d'un domaine donné à partir de plusieurs sources de connaissances (experts humains, livres, vidéos, etc.).
- La *validation des connaissances* consiste à vérifier sur des cas d'étude que la connaissance acquise est de qualité suffisante pour le domaine concerné et les besoins ciblés.
- La *représentation des connaissances* consiste à trouver un langage de représentation des connaissances adapté aux besoins et à traduire la connaissance acquise dans ce langage. Les énoncés produits par cette activité sont stockés dans un conteneur appelé *base de connaissances*. Étant donnée la distinction sémantique entre « information » et « connaissance » que nous avons faite, le terme le plus approprié serait donc *base d'informations*, mais nous acceptons de parler de base de connaissances en considérant que ces informations codées dans le langage machine et stockées dans la machine peuvent être considérées comme étant « dans l'esprit de la machine » (c'est-à-dire à l'intérieur de la machine) par analogie aux connaissances qui sont des informations comprises par l'homme et dans l'esprit de l'homme.

- Les *inférences* sont les mécanismes de raisonnement implémentés dans la machine qui sont souvent directement associés au langage de représentation des connaissances choisi. Les inférences permettent de générer des informations à partir d'autres informations explicitées dans la base de connaissances.
- L'*explication et la justification* consistent à fournir à l'utilisateur de la base de connaissances des moyens de poser des questions sur le domaine et de justifier à l'aide des informations dans la base les réponses qui y sont apportées.

Pour les tâches qui reviennent à l'ingénieur des connaissances, Russell & Norvig (2003) dressent à peu près la même liste mais à un niveau de grain plus gros : « *étudier un domaine donné* », « *en apprendre les concepts* » et « *donner une représentation formelle des objets et relations qu'il contient* ». Si Russell & Norvig ne parlent pas d'inférence et d'explication, c'est parce qu'il présuppose que le langage de représentation utilisé est un langage logique, et qui possède donc tous les mécanismes d'inférence et d'explication associés.

1.2.4.2 Les Systèmes à Base de Connaissances

Les Systèmes à Base de Connaissances (SBC) sont des systèmes informatiques qui contiennent et manipulent des connaissances pour agir. Les connaissances manipulées par les SBC sont donc des « connaissances machine » (c'est-à-dire « compréhensibles » par la machine). Ces SBC, leur conception, leur construction, leur maintenance et leurs applications sont l'objet d'étude de l'ingénierie des connaissances. La mise en place d'un SBC implique donc de réaliser correctement les cinq activités ci-dessus.

On peut différencier deux types de SBC : les SBC qui raisonnent à partir de modèles de connaissances, et d'autres part les SBC qui raisonnent à partir de l'expérience. Les premiers possèdent en interne une base de connaissances expertes modélisées, censées couvrir l'ensemble des lois qui régissent le monde dans lequel interagit le SBC. C'est le cas du SBC Mycin (Buchanan & Shortliffe 1984), pour lequel l'univers des symptômes et traitements est modélisé sous forme de base de règles. La phase d'acquisition des connaissances pour de tels SBC à base de modèles est préalable à la phase d'action. Il se peut que la base de connaissances soit mise à jour ensuite pour améliorer son comportement, mais les modifications des modèles de connaissances sont coûteuses et ces systèmes ne sont généralement pas prévus pour de telles mises à jour régulières. La pertinence du SBC dans sa tâche est donc directement conséquence de la qualité des modèles qui ont été construits par l'expert. Les SBC qui raisonnent à partir de l'expérience ont pour source principale de connaissances une base d'expériences passées qui peuvent être rappelées et réutilisées à tout moment par le SBC pour faire face à une situation similaire à une situation existante. Le Raisonnement à Partir de Cas (RàPC) formalise cette approche en distinguant une base de *cas* et un cycle de raisonnement en quatre étapes, où un *cas* constitue l'atome d'expérience capitalisé pour le SBC (Riesbeck & Schank 1989). De nombreux travaux de précision et d'amélioration de la méthode RàPC ainsi que d'application dans des domaines concrets se sont reposés sur ce paradigme (Aamodt & Plaza 1994, Mantaras et al. 2005). Les quatre étapes, *les quatre « R »* (Riesbeck & Schank 1989), du raisonnement sont :

Retrieve recherche d'un cas passé similaire à la situation actuelle et présent dans la base de cas,

Reuse adaptation du cas retrouvé à la situation actuelle en le modifiant légèrement et en adoptant pour la situation courante le même comportement que pour la situation passée tout en tenant compte de l'adaptation,

Revise ajustement par l'humain du cas adapté après avoir agi dans la situation courante et constaté quelques problèmes éventuels,

Retain stockage du nouveau cas adapté et révisé dans la base de cas pour une utilisation future.

Ce deuxième type de SBC vise à contourner les problématiques d'acquisition de connaissances, en particulier pour les domaines pour lesquels les experts ont peu de connaissances *a priori* et ne peuvent pas établir un modèle de comportement pertinent pour le SBC. Ainsi, pour ce type de SBC, la base de connaissances permettant d'agir (c'est-à-dire la base de cas) évolue et s'améliore au fur et à mesure que les connaissances du domaine d'application sont découvertes par la pratique. Cependant, même avec ce type d'approche et malgré la volonté de faire évoluer les connaissances du SBC en même temps qu'elles sont découvertes, un certain nombre de connaissances doivent être acquises *a priori*, comme les connaissances de similarité, d'adaptation et du domaine, même si le cycle du RàPC peut être modifié de manière à acquérir également les connaissances d'adaptation au fil des expériences (Cordier 2008).

1.2.5 Acquisition des connaissances

L'acquisition des connaissances est, nous l'avons vu, une étape obligatoire dans la réalisation d'un SBC, sur laquelle nous nous focalisons dans ce travail de thèse. Elle est également déterminante, car ce sont entre autres les connaissances présentes dans le SBC qui produisent en sortie le comportement du SBC. Si les connaissances acquises par le SBC sont de mauvaise qualité, le SBC se comportera de manière « surprenante » et fausse du point de vue de l'humain et ne sera pas utilisé par ce dernier. Par exemple, un SBC de type assistant au diagnostic médical contenant des connaissances de mauvaise qualité diagnostiquera plus souvent les mauvaises pathologies. Cet écart entre les attentes de l'humain et le comportement du SBC en pratique provient de l'écart entre les connaissances humaines présentes dans l'esprit de l'humain et les connaissances machine stockées dans la base de connaissances du SBC et qui prescrivent son comportement.

1.2.5.1 Origine humaine de la connaissance machine

Kendal & Creen (2006) distinguent six catégories[9] de SBC : les systèmes experts, les réseaux de neurones, le Raisonnement à Partir de Cas, les algorithmes génétiques, les agents intelligents, et les Environnement Informatique pour l'Apprentissage Humain. Cette classification peut paraître surprenante dans la mesure où les catégories ne sont pas homogènes, certains systèmes experts, agents intelligents ou Environnement Informatique pour l'Apprentissage Humain pouvant embarquer aussi bien un réseau de neurones ou un algorithme génétique qu'une base logique. Cette classification a probablement été établie par analyse historique des différents domaines des l'Intelligence Artificielle ayant convergé vers les SBC. Elle permet cependant de donner un rapide aperçu de l'état du domaine. Quel que soit le SBC, l'origine des connaissances qu'il manipule et qui conditionne, son comportement est toujours humaine. Par exemple, pour un SBC de type *algorithme génétique*, c'est un expert humain qui aura préalablement étudié le domaine d'application du SBC pour établir les meilleures fonctions d'adaptation (fonction de *fitness*) et de recombinaisons possibles. Dans un SBC de type *Raisonnement à Partir de Cas* , c'est encore un expert humain qui aura entré les mesures de similarité et fonction d'adaptation nécessaires à la recherche et la réutilisation de cas de résolution de problème similaires. De même, pour tout SBC comme Mycin (Buchanan &

[9]Sept en réalité, mais nous avons exclu la catégorie « fouille de données » car nous la voyons davantage comme un moyen d'acquérir des connaissances, quel que soit le type de SBC.

Shortliffe 1984) dont le fonctionnement repose sur la modélisation préalable des connaissances du domaine dans un langage supportant les inférences, c'est-à-dire un langage logique, c'est encore un expert humain qui se chargera d'étudier le domaine et de le traduire en base de connaissances.

1.2.5.2 Acquisition ascendante et descendante des connaissances

Charlet (2009) distingue deux méthodes opposées d'acquisition des connaissances en ingénierie des connaissances : la méthode ascendante et la méthode descendante. La méthode descendante consiste à disposer *a priori* d'un modèle générique de connaissances qui se spécialise ensuite en modèle du domaine. C'est le cas de la méthode CommonKADS (Schreiber 2000), qui propose un framework de connaissances en trois couches (connaissances de la tâche, connaissances d'inférence et connaissances du domaine) et un langage de représentation pour les différentes connaissances que l'expert utilise pour implémenter le SBC cible. L'avantage de ce type d'approche est la réutilisabilité des connaissances. CommonKADS propose une bibliothèque de modèles, un peu à la manière des bibliothèques logicielles. Lors de la réalisation du SBC, l'expert doit sélectionner le meilleur modèle et l'adapter à la tâche du SBC. Le désavantage de l'approche descendante est que le bon modèle doit être présent dans la bibliothèque. De plus le fait que les tâches à réaliser soient pré-caractérisées introduit souvent une limitation dans l'expression de la tâche du SBC cible. À l'inverse, les méthodes ascendantes sont peu réutilisables. Elles visent à réaliser les modèles de connaissances sans aucun *a priori* sur leurs structures et leurs contenus à partir des données brutes du domaine et dans le but seul de caractériser le plus précisément possible la tâche à réaliser. Les outils de découverte de connaissances à partir de données (Fayyad et al. 1996) sont de très bons exemples de ce type d'approches. Au départ du processus d'acquisition des connaissance, seules les données existent, c'est-à-dire les observations factuelles du domaine. Des outils d'analyse pilotés par l'expert permettent d'extraire des motifs récurrents et intéressants dans ces données. De tels motifs sont validés ou non par l'expert et viennent ensuite alimenter le modèle de connaissances du SBC (cf. section 2.2.1).

1.3 Les verrous de l'ingénierie des connaissances

1.3.1 Le goulet d'étranglement de l'acquisition des connaissances

L'une des problématiques qui animent la communauté de l'ingénierie des connaissances, parfois appelée « goulet d'étranglement » de l'ingénierie des connaissances (Charlet 2009), concerne l'acquisition des connaissances. On parle de « goulet d'étranglement » car quel que soit le type de SBC ciblé, nous avons vu que les connaissances qu'il contient sont d'origine humaine et il nécessite pour cela une longue phase d'acquisition de connaissances auprès d'experts humains, avec tous les problèmes que soulève cette démarche : difficultés à capturer précisément l'expertise d'un expert, difficultés à coder cette expertise en langage machine, etc. Cette phase d'acquisition est coûteuse puisqu'elle nécessite de regrouper plusieurs acteurs autour de la même tâche : experts, des informaticiens, utilisateurs, décideurs, etc. De plus, les incompréhensions inhérentes à la communication humaine, qui surviennent entre ces acteurs lors du processus d'acquisition de connaissances, débouchent sur des imperfections du SBC cible. Kendal & Creen (2006) parlent de « champion du projet » pour décrire une personne idéale capable de mener à bien le projet de mise en place du SBC dans une organisation. En effet, celui qui mène le projet de SBC doit à la fois :

- comprendre auprès des décideurs les objectifs du SBC cible,

- comprendre l'ensemble des connaissances des différents experts du domaine,
- être capable de représenter les connaissances formellement et de les coder dans la machine,
- convaincre les utilisateurs de son utilité,
- croire en l'intérêt et la réalisation du projet.

Idéalement, une seule et même personne possède les compétences de tous ces rôles, mais en pratique ces rôles sont joués par les divers acteurs prenant part à l'acquisition des connaissances, ce qui débouche inéluctablement sur des SBC au mieux imparfaits, mais souvent défaillants et inutilisables.

1.3.2 Difficulté de mise à jour des connaissances machine

Nous l'avons vu en section 1.2.2, la connaissance est de nature très variable dans le temps, selon l'individu et le contexte d'utilisation. Cela soulève une grande problématique en ingénierie des connaissances puisque les connaissances machine sont souvent difficile à modifier ou à mettre à jour en pratique. Cette problématique est particulièrement présente dans le cas des ontologies. Une ontologie est une représentation formelle du domaine dans lequel interagit le SBC. Les éléments du monde y sont représentés par des concepts et des relations. En informatique, les ontologies sont souvent des descriptions en logique d'un domaine, et c'est d'ailleurs la principale motivation des *logiques de description* que de formaliser la représentation logique du monde et de permettre de nombreux raisonnements sur les connaissances représentées (Napoli 1997). Les modifications les plus simples consistent à ajouter des concepts à l'ontologie, par exemple pour étendre le champs d'action du SBC à un nouveau sous-domaine. Ajouter un concept revient à créer une nouvelle classe nommée, à définir ses relations avec d'autres classes et ses propriétés. Supprimer un concept de l'ontologie ne revient pas seulement à supprimer la ou les classes concernées. Il faut également supprimer toutes les références à cette classe. Souvent, les modifications à effectuer ne relèvent pas simplement de l'ajout ou de la suppression de concepts, mais des connexions entre concepts (hiérarchie, disjonction, équivalence, concepts définis à partir de constructeur logique et d'autres concepts, etc.), qui impliquent d'éditer l'ontologie en plusieurs endroits. Or de telles modifications sont souvent pour corriger des imperfections de l'ontologie dans sa représentation du monde révélées lors de l'action du SBC dans sa tâche. Corriger ainsi les connaissances requiert d'identifier dans l'ontologie les concepts fautifs et d'opérer les altérations nécessaires. Lorsque l'expert revient sur l'ontologie après un long temps, il lui est très difficile de se remémorer les raisons qui ont motivé ses choix de conception et il peut écraser certains de ces choix sans s'en apercevoir et sans le vouloir. Les ontologies se voulant communautaires, ce problème devient encore plus gênant, car les individus de la communauté peuvent écraser des choix de conception effectués par d'autres sans même en avoir conscience. De telles modifications ne seraient pas aussi problématiques si la consistance de l'ontologie entière n'était pas remise en cause à chaque fois. La moindre modification peut mener à une incohérence et l'origine de cette incohérence est très difficile à localiser dans l'ontologie (Parsia et al. 2005). Cette problématique de mise à jour des ontologies nuit à leur objectif de permettre la formalisation partagée par plusieurs individus et systèmes d'un même domaine, car les connaissances de ce domaine dépendent du point de vue (cf. section 1.2.2) de chacun et de nombreuses altérations de l'ontologie sont nécessaires avant d'obtenir une représentation qui commence à faire consensus.

1.3.3 Prise en compte de l'aspect dynamique des connaissances

Pour prendre en compte l'aspect dynamique des connaissances dans la conception d'un SBC, une possibilité est d'utiliser des structures de représentation plus souples que les ontologies, comme les *taxonomies* ou les *thésaurus*. Les taxonomies permettent de représenter le monde sous forme de hiérarchie de concepts. Il s'agit d'une tentative de classification du monde. Les seuls éléments présents dans la représentation sont les concepts eux-mêmes (les *taxons*) et les relations de subsomption entre ces concepts. Parfois, les thésaurus permettent l'ajout de relations sémantiques entre ces concepts. Une forme encore plus faible de structuration d'un domaine est le *nuage de tags* (on parle parfois de « folksonomie » lorsqu'il est alimenté par plusieurs contributeurs). Le nuage de tags est un ensemble de taxons, sans obligation de relation entre eux. Les efforts à fournir lors de la modification de ces structures sont minces et les éléments de représentation sont accessibles à tout humain n'étant pas expert en modélisation des connaissances. C'est pourquoi ces structures ont beaucoup de succès sur le Web et permettent d'élaborer des bases de connaissances partagées et évolutives. En contrepartie, les possibilités d'inférence et d'action à partir des connaissances sont inversement proportionnelles au degré de structuration des ces connaissances ; moins il y a de relations entre concepts, et moins la machine pourra utiliser ces connaissances pour agir.

Pour les ontologies d'un même domaine ayant été élaborées par deux groupes différents, il existe des techniques d'alignement permettant d'unifier les concepts et relations des différentes ontologies, afin d'obtenir une ontologie unique et partagée d'un même domaine. Ces mêmes techniques d'alignement peuvent être utilisées pour agréger de nouvelles ontologies et augmenter la couverture des connaissances. Mais le résultat en est parfois aussi la perte de spécificité des ontologies agrégées au profit d'une expression commune du domaine. Pour les bases de connaissances logiques, des opérateurs de révision (Alchourrón et al. 1985) formalisent dans quelles conditions des nouvelles connaissances, potentiellement inconsistantes avec les connaissances déjà en base, peuvent être ajoutées à la base en cours. La stratégie consistant à rechercher et supprimer de la base les axiomes inconsistants avec la nouvelle connaissance, c'est-à-dire l'écrasement systématique, n'étant pas toujours la meilleure solution. En effet, pour modifier la base de manière la plus appropriée à la nouvelle vision du monde, des modifications plus subtiles et au cas par cas, guidées par l'expert humain sont plus à recommander.

Les SBC qui reposent sur l'expérience, comme ceux utilisant le RàPC, se veulent plus aptes à prendre en compte l'aspect dynamique de la connaissance. L'ajout de nouvelles connaissances se limite généralement au simple fait de mémoriser les nouveaux cas. En revanche, s'il arrive que le monde change de manière conséquente ou que le problème à résoudre par le RàPC évolue, il se peut que ce soit la structure des cas qui soit à modifier. De plus, en RàPC, une partie des connaissances humaine sur la résolution du problème cible est transférée dans la machine sous forme de mesure de similarité et de connaissances d'adaptation. Ce type de connaissance au sein du RàPC est représenté sous forme de modèle (e.g. fonction de mesure de similarité entre deux cas, base de règles d'adaptation, etc.) et est donc en partie sujet aux mêmes inconvénients de mise à jour que les SBC reposant uniquement sur des modèles de connaissances. C'est pourquoi certaines recherches se focalisent sur les stratégies permettant l'évolution d'un SBC basé sur le RàPC (Cordier et al. 2009, Craw 2009).

1.4 Ingénierie de la dynamique des connaissances à partir de traces

Ce rapide aperçu du domaine de l'ingénierie des connaissances montre d'une part le besoin de construire des connaissances machine dynamiques pour respecter au mieux la nature variable et

évolutive de la connaissance, et d'autre part la grande difficulté à faire face à cette variabilité de la connaissance. Cette difficulté tient principalement du fait que la connaissance est complexe par nature et nécessite des structures complexes pour les implémenter en machine qu'il est difficile de modifier simplement.

L'objectif général du groupe de recherche SILEX[10] est de proposer une ingénierie de la dynamique des connaissances basée sur les traces d'interactions laissées par les humains lors de leur activité. Dans le cadre de cette recherche, c'est la phase d'acquisition des connaissances qui est la plus critique, car c'est à cette phase que se joue le dynamisme de l'ensemble. En effet, supposons que nous avons une base de connaissances évolutive grâce à un processus d'acquisition dynamique, tout SBC construit sur cette base par l'ajout de mécanismes de raisonnement traditionnels sera considéré comme évolutif également. Cependant, l'approche par les traces permet également d'effectuer un raisonnement directement à partir de l'expérience tel le RàPC, mais en se basant sur les traces au lieu des cas.

Ce travail s'inscrit dans une volonté de construction de connaissances machine au sujet d'une activité humaine, collaborative ou non, qui soient de bonne qualité pour être ensuite embarquées dans des SBC proposant de l'assistance aux individus prenant part à cette activité. Pour ce faire, le processus de construction des connaissance doit répondre aux critères suivants :

- *acquisition ascendante des connaissances*, pour que les connaissances construites soient plus précises et plus exactes dans la description du monde, et générées par les besoins de représentation du monde qui ont été rencontrés,
- *en apprentissage continu* à partir de l'expérience, de sorte que le modèle de représentation des connaissances machine soit sans cesse remis en cause à l'épreuve de l'expérience pour s'adapter aux nouvelles situations rencontrées dans le monde.
- *en interaction entre l'humain et la machine*, de sorte que les connaissances acquises soient validées, modifiées au besoin, par un humain, afin d'assumer l'origine humaine de la connaissance machine et d'intégrer cet humain dans l'acquisition des connaissances tout au long de la vie du SBC, et non plus seulement à la phase de conception. Cette interactivité avec l'humain vise également à prendre acte de la variabilité interindividuelle en respectant au mieux le point de vue de l'humain dans la construction des connaissances.

Ce travail de thèse propose d'utiliser les traces d'interactions des utilisateurs comme inscriptions informatiques de l'expérience des utilisateurs et comme conteneurs potentiels de connaissances à découvrir, expliciter et finalement acquérir. Il repose notamment sur les efforts de théorisation de la trace informatique et de sa représentation formelle effectués par l'équipe SILEX au cours de ces dernières années.

[10] Supporting Interaction and Learning by EXperience (Équipe de recherche du Laboratoire d'InfoRmatique en Image et Systèmes d'information, UMR 5205 CNRS). URL : `http://liris.cnrs.fr/equipes?id=44`.

Chapitre 2

Co-construction interactive de connaissances à partir de traces

Sommaire

Le travail algorithmique présenté dans cette thèse prend tout son sens dans le contexte d'un processus de co-construction de connaissances à partir de traces d'interactions. Nous présentons une série de travaux sur l'ingénierie des traces d'interactions permettant d'étudier comment elles sont obtenues (gestion des captures), quels types de connaissances elles peuvent contenir, comment elles sont représentées, et comment elles sont exploitées en vue d'une réutilisation de l'expérience ainsi tracée. Le terme « co-construction » indique que la construction de connaissance est un processus associant humain et machine. Dans ce chapitre nous parcourons des techniques de construction de connaissances impliquant à la fois l'humain et la machine en mettant en avant leurs particularités, leurs points forts et leurs points faibles. Nous proposons ensuite un processus, que nous pensons original, de co-construction de connaissances qui repose sur l'émergence et l'explicitation des connaissances découvertes dans les traces d'interactions, en dressant la liste des propriétés que la partie machine doit exhiber pour être pro-active et que ce processus prenne la forme d'un échange productif avec l'humain dans son activité de découverte de connaissances.

2.1 Ingénierie des traces d'interactions

L'ingénierie des traces est une discipline qui traite des problématiques et méthodes de gestion des traces d'interactions (modélisation, capture, collecte, stockage, requêtage, etc.) laissées par les utilisateurs de systèmes informatiques au cours de leurs activités. Dans cette section, les travaux que nous présentons sur l'ingénierie des traces ont pour la plupart été réalisés par le groupe de recherche SILEX (Supporting Interaction and Learning by EXperience) du laboratoire LIRIS, dans lequel s'inscrit cette thèse. Ces travaux sont d'une grande importance pour la suite du manuscrit, car le processus de construction de connaissances que nous proposons utilise les traces d'interactions pour produire des connaissances et s'appuie sur l'existence de méthodes et d'outils de manipulation des traces d'interactions.

2.1.1 Utiliser les traces d'interactions comme source de connaissances

2.1.1.1 Les notions de traces informatiques et de traces d'interactions : définitions

Ce que nous appelons *trace informatique* est une empreinte laissée par un processus qui s'exécute en interaction avec un système informatique. L'exemple le plus connu de trace informatique est le fichier log. Le fichier log provient de la volonté d'un concepteur de logiciel de produire automatiquement un enregistrement lors de l'exécution du programme à certains passages stratégiques, dans certaines conditions, et généralement sources de problèmes et de pannes. Les historiques des navigateurs Web ou de certains logiciels sont un autre exemple de trace informatique.

Il existe des formalisations mathématiques de la notion de trace, en tant que trace d'exécution d'un automate (Diekert & Rozenberg 1995) et en tant que trace d'exécution d'un programme logique de résolution de problème (Deransart et al. 2007). Ces formalisations permettent entre autre de théoriser la faisabilité de reconstruire ou non un processus à partir des traces qu'il laisse. Cependant, ces formalisations ne peuvent pas s'appliquer telles quelles à la gestion des traces d'interactions, car dans une activité humaine les processus tracés ne peuvent pas être complètement explicites. Nous reviendrons sur ce point après avoir présenté les traces d'interactions.

Dans ce manuscrit, nous définissons les *traces d'interactions* comme des traces informatiques qui proviennent des interactions entre un système informatique et un utilisateur de ce système. Toute trace d'interactions sera donc une trace informatique. Dans la pratique, ces traces dites « d'interactions » sont souvent collectées grâce aux processus informatiques résultant des actions de l'utilisateur. C'est le cas par exemple lorsque l'on capture les interactions de l'utilisateur avec un formulaire : entrer une nouvelle valeur déclenche un événement système qui est « écouté » par un programme informatique de collecte créant la trace. Cependant, cela n'est pas toujours vrai. Il est possible de capturer des traces d'interactions entre un système informatique et son utilisateur sans que ce soit l'un des processus internes au système qui soit tracé. Par exemple, une empreinte informatique peut être créée manuellement par un observateur tiers à chaque interaction significative. Dans ce cas, il y a un processus informatique qui est utilisé pour construire la trace comme dans les autres cas. Seule la "sonde" n'est pas informatique, mais le processus de construction l'est. Pour qu'il y ait trace informatique, il faut un processus informatique la construisant.

2.1.1.2 Les traces d'interactions comme descripteurs de l'activité

Collecter et analyser des traces d'interactions pour comprendre comment les humains se comportent dans leurs environnements d'activité n'est pas nouveau. Sanderson & Fisher (1994) en proposent un état de l'art et une synthèse, et proposent une méthode générique, appelée ESDA

(*Exploratory Sequential Data Analysis*), d'analyse des données séquentielles d'observation pour en inférer des connaissances sur l'utilisation du système. La méthode consiste à collecter les séquences d'observation, à les utiliser pour obtenir des « produits transformés » (*Transformed Products*). Ces « produits transformés » sont des formes interprétées des observations, alimentant les conclusions des analystes de l'activité. Ces méthodes sont très utiles pour les sciences humaines et permettent de créer de la connaissance dans ces disciplines. Les connaissances produites ainsi ne sont pas des « connaissances machine » mais des « connaissances humaines » (courbes, documents, etc.), au sens où nous avons défini ces deux types de connaissance en section 1.2.1. Autrement dit, les connaissances produites par ces méthodes ne sont pas représentées sous une forme exploitable par un moteur d'inférence mais sont interprétables par des humains formés à leur interprétation. Il n'est pas possible d'utiliser directement ces connaissances pour produire des SBC capables d'agir en fonction de celles-ci.

Les traces d'interactions formelles représentent l'histoire des interactions par des données structurées volontairement pour rendre compte de cette histoire interactionnelle. Une machine pourra donc exploiter ces structures pour traiter de l'activité tracée. Puisqu'une trace informatique est obligatoirement structurée explicitement, il n'existe pas de distinction franche en les traces d'interactions non-formelles et les traces d'interactions formelles, les différentes structures de traces offrant plus ou moins d'information exploitable pour rendre compte de l'histoire interactionnelle. Une vidéo de l'activité observée sera considérée comme non formelle car aucun aspect de l'activité filmée n'est représenté par une structure dans la machine (les structures représentent les images et la façon de les reproduire) et la machine ne pourra par exemple pas utiliser directement la vidéo pour inférer que l'utilisateur est dans une situation nécessitant une assistance, ou une fonctionnalité particulière. À l'inverse, si les traces d'interactions sont des séquences d'annotations RDF décrivant l'activité selon une sémantique précise, la machine pourra agir en fonction de ce qui est décrit par les annotations RDF. Les fichiers logs sont entre les deux. Les enregistrements contenus sont présentés selon le même modèle, par exemple une date, un niveau de sécurité, une source, et un commentaire. Le système pourra utiliser la date, le niveau de sécurité et la source pour agir, par exemple déclencher l'arrêt du système en cas de problème sur un module à risque (si la sémantique du niveau de sécurité est explicitée), mais l'essentielle de l'information portée par l'enregistrement sera comprise dans le commentaire formulé par un humain et donc inaccessible directement à la machine. Roussel et al. (2006) proposent une approche par les logs pour tracer les activités de l'utilisateur, mais en réalité, ce que les auteurs appellent « logs » ne sont pas les simples fichiers logs créés par les développeurs « à toutes fins utiles », mais des informations modélisées spécifiquement en provenance d'un système spécialement instrumenté à cet effet.

2.1.1.3 Approches de modélisation des traces d'interactions au sein du groupe SILEX

C'est l'originalité de l'approche MUSETTE (*Modéliser les USages et les Tâches pour Tracer l'Expérience*) de proposer une représentation formelle de l'expérience d'utilisation d'un système par ses utilisateurs (Champin et al. 2003, Champin et al. 2004, Laflaquière & Prié 2009). MUSETTE introduit un *modèle d'utilisation* et un *modèle d'observation*. Le modèle d'utilisation décrit quels objets d'intérêt de l'activité (des entités, des événements et des relations) doivent être présents dans la trace et le modèle d'observation décrit quelles données du système doivent être collectées pour reconstituer un trace conforme au modèle d'utilisation. La trace obtenue est représentée sous forme d'états et de transitions successives décrivant l'activité de l'utilisateur. Une telle trace formelle a prouvé qu'elle pouvait être utilisée par le système pour fournir de l'assistance contextuelle à l'utilisateur (Champin 2003), en appariant à la trace en cours des situations passées similaires à

des *signatures de tâche* (cf. section 2.1.4).

Le formalisme état/transition de MUSETTE, trop contraignant pour représenter l'activité d'utilisation dans certains cas, a été ensuite abandonné au profit du modèle des *traces modélisées*, ou *M-traces* (Settouti et al. 2009), modèle qui laisse plus de liberté dans la description de l'activité. Le principe original du modèle des M-traces reste le même que pour MUSETTE : un modèle associé à chaque trace d'interactions (le *modèle de trace*) décrit les éléments qu'elle contient (cf. section 2.1.2). Les notions d'observation et de modèle d'observation sont abandonnées et l'on ne parle plus que de « collecte de trace » (cf. section 2.1.3). Le modèle de trace doit être créé *a priori*, ce qui peut être critique car il est difficile de prévoir à l'avance quels aspects de l'activité doivent être réifiés dans la trace pour qu'elle contienne les informations nécessaire à la réutilisation ciblée. Des méthodes existent pour créer ce modèle de trace au mieux (Laflaquière 2009) et la notion de SBT a été proposée pour assurer une dynamique de modélisation facile.

2.1.1.4 Les traces d'interactions comme conteneurs de connaissances

Si nous nous efforçons de représenter formellement ces traces d'interactions, c'est qu'elles décrivent l'expérience d'utilisation d'un système. Mille (2009) explique que toute connaissance trouverait sa source dans l'expérience, et que « l'expérience sensible étant par définition non accessible en tant que telle puisque incorporée, au sens propre du terme, seules les traces laissées dans l'environnement (y compris probablement chez le sujet vivant l'expérience) restent des inscriptions d'une connaissance émergente ou/et mobilisée au moment d'une nouvelle expérience ». Les connaissances que contiennent ces traces peuvent être mises à profit aussi bien pour l'analyse de l'activité tracée (Sanderson & Fisher 1994, Georgeon 2008*a*) que pour l'assistance aux utilisateurs dans cette activité (Champin 2003, Dragunov et al. 2005, Cram et al. 2007*a*).

Dans la plupart des cas, les modules d'assistances ne cherchent pas à capitaliser les connaissances contenues dans la trace qui ont été mobilisées pour fournir l'aide. Dans cette thèse, nous nous plaçons dans le cadre de l'assistance. Cependant, notre objectif n'est pas de produire un système d'assistance qui agit directement à partir des traces, mais plutôt de réaliser un processus co-construction de connaissances machine explicites à partir de traces (cf. section 2.3). Ces connaissances explicites sont candidates à devenir des signatures de tâche (cf. section 2.1.4) exploitables par un assistant à la réutilisation et au partage de connaissances.

2.1.2 Les traces modélisées (M-traces)

Cette sous-section présente plus en détail l'approche des *traces modélisées* de Settouti et al. (2009). Settouti et al. définissent formellement les notions d'observé, de relation, de trace et de modèle de trace. Nous le simplifions et résumons très brièvement ici pour donner un aperçu du format de « trace modélisée », format selon lequel seront collectées les interactions capturées par la suite. En résumé, une *trace modélisée* est composée d'un ensemble d'*observés* et du modèle de cette trace : le *modèle de trace*. Un *observé* est un objet typé, constitué de paires attribut/valeur, et possédant une *extension temporelle*. L'extension temporelle est une information obligatoire, elle a pour rôle de situer l'observé dans le temps. Dans les cas les plus courants, cette extension temporelle est une simple date ou un intervalle dans le temps, c'est-à-dire une date de début et une date de fin. Mais le méta-modèle autorise tous les cas possibles de représentation du temps. Enfin, des *relations* binaires peuvent connecter des observés de la trace entre eux.

Le modèle de trace spécifie l'ensemble des types que les observés peuvent instancier. Ces types sont organisés en hiérarchie. Le modèle de trace définit aussi l'ensemble des attributs que chaque type d'observé peut avoir et leurs domaines de valeurs. Il donne aussi le domaine temporel de la

trace, c'est-à-dire les extensions temporelles possibles pour les observés. C'est ce domaine temporel qui dicte si les extensions temporelles sont des dates ou des intervalles, ou autres. Enfin, le modèle de trace spécifie les types de relation, ainsi que leurs domaines et co-domaines.

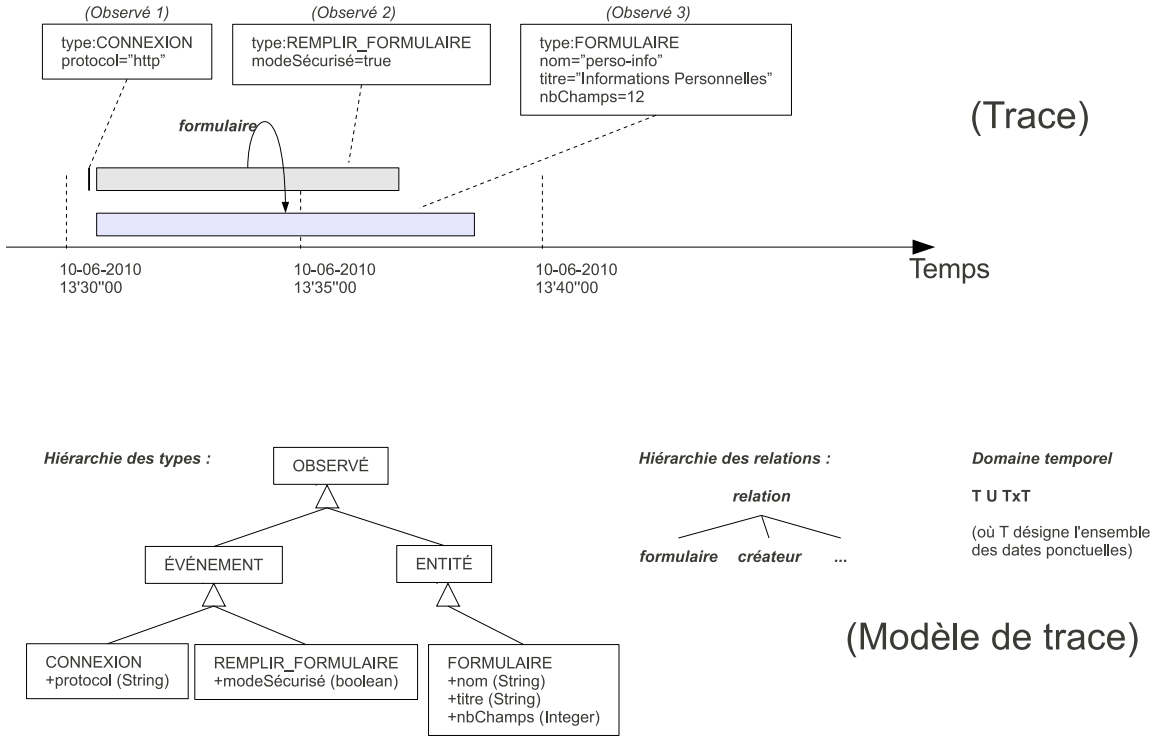


Figure 2.1 – Exemple de M-trace au format spécifié par Settouti et al.. Sur la moitié du haut : la trace ; sur la moitié du bas : son modèle de trace.

La figure 2.1 montre un exemple de M-trace et de son modèle de trace associé. Cette M-trace est composée de trois observés (*Observé* 1, *Observé* 2, *Observé* 3). *Observé* 1 est ponctuellement situé dans le temps (à la date 13′31″00 à peu près) alors que *Observé* 2 et *Observé* 3 persistent dans le temps (de 13′31″00 à 13′37″00 pour *Observé* 2 et de 13′31″00 à 13′38″00 pour *Observé* 3). Le type de *Observé* 1 est `CONNEXION` ; le type de *Observé* 2 est `REMPLIR_FORMULAIRE` ; le type de *Observé* 3 est `FORMULAIRE`. Les paires attributs/valeur sont représentées en dessous des types sous la forme « $attribute = valeur$ ». Enfin, il existe une relation de type `formulaire` entre *Observé* 2 et *Observé* 3, cette relation signifiant que le formulaire cible de *Observé* 2 (c'est-à-dire le formulaire qui est rempli) est *Observé* 3. Cette trace peut se lire comme suit : « l'utilisateur s'est connecté à la plate-forme via le protocole HTTP à 13*h*31, a ensuite rempli en mode sécurisé le formulaire intitulé « Informations Personnelles » (comportant 12 champs) pendant 6 minutes ». Le fait que l'observé de type `FORMULAIRE` persiste une minute de plus que l'observé de type `REMPLIR_FORMULAIRE` peut s'expliquer techniquement dans le système par le fait que l'objet « Formulaire » est informatiquement détruit une minute après la fin du remplissage. Le modèle de trace de cet exemple différencie dans la hiérarchie des types, deux types d'observé génériques : les événements et les entités. Cette distinction peut se faire dans pratiquement tout description d'activité entre les actions qui sont effectuées et les entités (c'est-à-dire les « objets » matériels ou abstraits) qui sont manipulées[11]. Enfin, le domaine temporel de cet exemple ($T \cup T \times T$) signifie que l'extension temporelle de

[11] C'est la même distinction qu'entre les concepts *endurant* et *perdurant* dans les ontologies.

chaque observé peut être soit une date, soit deux dates (interprétées comme « début » et « fin », c'est-à-dire un intervalle).

2.1.3 Architecture du SBT

Le Système à Base de Traces (SBT) centralise la gestion et le stockage des traces. On pourrait l'appeler « Système de Gestion des Base de Traces » en écho aux « Systèmes de Gestion de Base de Données » (SGBD), puisque le rôle du SBT est analogue à celui d'un SGBD pour les données. Les traces sont des données également, mais contrairement au SGBD qui impose que ces données soient modélisées au format relationnelle, le SGBD requiert que les données soient modélisées au format M-trace, c'est-à-dire soient des traces modélisées conformes au modèle de Settouti et al. (2009).

Dans le schéma d'architecture générale du SBT (cf. figure 2.2, tirée de Laflaquière et al. (2006)), on retrouve certaines briques généralement fournies par les SGBD.

Système de collecte Il est à l'interface entre l'environnement tracé et le SBT lui-même. Il est constitué d'une API (*Application Program Interface*) d'entrée, qui fournit des méthodes de création/suppression de M-traces, de création/suppression/modification d'observés et d'éléments du modèle de trace, et prêt à recevoir des messages depuis l'extérieur, c'est-à-dire depuis les sources de traçage. Par exemple, supposons qu'un capteur RFID détecte l'entrée d'une puce RFID dans son champs de détection, alors un module logiciel associé au capteur devra réceptionner cet événement au format brut du capteur et le traduire en observé. Cela présuppose que le modèle de trace de la trace collectée a préalablement été conçu pour accueillir ce type d'observé fourni par la source de traçage ST_1, c'est-à-dire qu'il existe dans le modèle de trace de la trace collectée un type d'observé auquel l'observé créé par ST_1 est conforme. Les sources de traçage sont donc des modules logiciels externes, généralement développés de manière *ad hoc* pour chaque activité tracée, et communiquant avec l'API du système de collecte pour instancier les classes du modèle de trace[12]

Système de transformation Il implémente l'ensemble des fonctionnalités du SBT, c'est-à-dire les sélections et transformations (toute transformation étant nécessairement composée d'au moins un module de sélection de motif à transformer et de la transformation proprement dite). La gestion des transformations est un problème compliqué. Il a été formalisé pour la majeure partie des besoins par Settouti et al. (2009). Le système de transformation est piloté par l'utilisateur du SBT par l'intermédiaire du *système de requête*.

Système de stockage Il a pour rôle de persister les traces collectées et/ou transformées.

Système de requête Il fournit un langage à l'utilisateur du SBT pour sélectionner des traces, des parties de traces, des observés, ou pour effectuer des transformations sur les traces. Il remplit le même rôle que le langage SQL pour les bases de données relationnelles.

Système de visualisation Le système de visualisation n'est pas nécessaire pour faire du SBT un « tout » cohérent, puisqu'il est toujours possible de réaliser un module de visualisation externe plus ou moins élaboré et adapté à des besoins spécifiques en se basant sur le langage de requête. Cependant, un système de visualisation générique et utilisable à des fins de déboggage

[12] La figure 2.2 associe une source de traçage à un capteur, mais une relation de type n-n est possible, bien que ce soit rarement le cas en pratique.

des modèles de traces est particulièrement utile, afin que les informaticiens chargés du traçage ne soient pas obligés de commencer leur travail par un module de visualisation minimaliste.

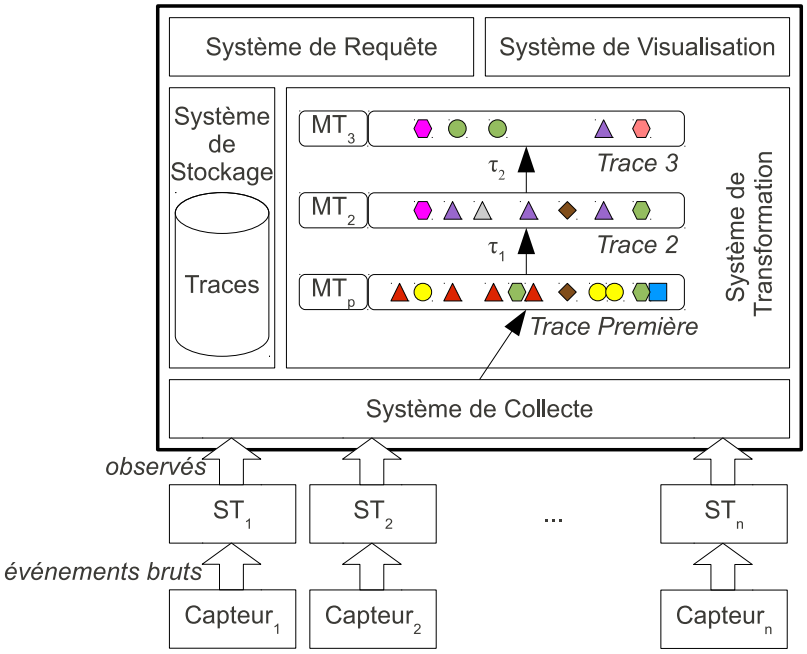


Figure 2.2 – Schéma d'architecture du Système à Base de Traces (« *MT* » signifie « Modèle de Trace » et représente le modèle de la trace à laquelle il est rattaché)

Dans le système de transformation de la figure 2.2, et pour la suite de ce rapport de thèse, on distingue la *trace première* et les *traces transformées*. La *trace première* est la trace modélisée qui est composée des observés en provenance directe du système de collecte, sans qu'aucune transformation n'ait été effectuée. Si en revanche une trace a été obtenue à partir d'observés issus de la collecte par quelque transformation que ce soit, alors cette trace sera dite *trace transformée*. La distinction est importante car les traces premières sont pérennes dans le SBT. Toute trace transformée peut par contre être soit supprimée, voire jamais persistée dans le système de stockage, car il est toujours possible de recalculer une trace transformée à partir de la trace première et des différentes transformations qui lui sont appliquées pour obtenir cette trace transformée.

Si Settouti et al. posent les bases formelles de ce SBT, les problèmes d'API de collecte et d'implémentation du langage de requête en particulier restent entiers. Deux implémentations de ce SBT ont toutefois été développées sur ces bases formelles. La première, nommée KTBS[13], est implémentée au sein de l'équipe SILEX. KTBS vise à implémenter à terme toutes les spécifications du modèle formel des traces modélisées de Settouti et al. et fait face à des problématiques algorithmiques fortes, notamment l'optimisation du temps de calcul des transformations. La deuxième implémentation a été réalisée par la société Knowings dans le cadre du projet PROCOGEC. Il a été conçu avec la volonté pragmatique de n'implémenter dans les spécifications du SBT que ce qui est réalisable dans les critères de fiabilité qu'un client peut attendre d'un produit informatique. Ainsi, ce SBT n'implémente pas de modèle de trace (les observés ont bien un nom de type, mais la gestion des types entre eux doit être effectuée en externe) et les transformations sont réalisées en `Javascript`. L'utilisateur du SBT souhaitant réaliser une transformation de trace doit créer

[13] `http://liris.cnrs.fr/sbt-dev/ktbs/`

un script `Javascript` auquel est donné un ensemble d'objets que l'utilisateur peut manipuler via `Javascript`. Entres autres, ces objets sont la ou les traces sur lesquelles la transformation s'opère et la trace cible. Généralement, l'utilisateur du SBT écrit en `Javascript` une boucle qui itère sur les observés des traces sources et spécifie comment ces observés doivent être réifiés dans la trace cible, e.g. laissés, supprimés, modifiés ou fusionnés avec d'autres observés. L'annexe D montre un exemple d'une telle transformation SBT en `Javascript`.

2.1.4 Notion de signature de tâche

Le concept de signature de tâche a été introduit dans le cadre du modèle MUSETTE et appliqué dans le prototype ARDECO. Une signature de tâche est un objet informatique qui représente l'exécution typique d'une certaine tâche cible réalisée par l'utilisateur dans l'environnement tracé. Elle doit s'apparier avec les morceaux de la trace qui décrivent la réalisation de cette tâche. La signature de tâche est donc une structure de données décrivant quels éléments du modèle de trace doivent apparaître pour considérer que la tâche cible est en train d'être réalisée. Par exemple, dans le cadre de l'utilisation d'une plate-forme collaborative, on pourrait définir une signature de tâche intitulée « mettre à jour ses données personnelles » comme étant l'apparition de trois observés de types respectifs `CONNEXION`, `REMPLIR_FORMULAIRE`, `FORMULAIRE`. Cette signature s'apparierait avec la trace en cours de collecte chaque fois que ces trois observés apparaîtraient dans la trace première, et donc chaque fois que l'utilisateur mettrait à jour ses données personnelles. Plusieurs travaux sur les traces se sont heurtés au problème de représentation de la signature de tâche (Champin 2003, Stuber 2007), mais des solutions simplistes et peu satisfaisantes ont été développées au cas par cas. Si la représentation d'une signature de tâche est problématique, c'est qu'elle doit :

- d'une part permettre de spécifier quels observés doivent apparaître pour être considérés comme appartenant à l'occurrence de la signature,
- d'autre part permettre de spécifier comment ces observés doivent se situer temporellement les uns par rapport aux autres.

Spécifier quels observés doivent apparaître est l'aspect *structurel* du problème de représentation. Comment sélectionner précisément les observés souhaités ? On peut le faire par leurs types, la présence ou non d'un attribut, ou d'une certaine valeur d'attribut, ou d'une certaine relation, ou d'une relation avec un certain domaine. L'important est de pouvoir spécifier des *wildcards*[14] autorisant des éléments à apparaîtrent sans connaître leurs natures. On trouve des pistes pour aborder l'aspect structurel de la signature de tâche dans les langages de requêtes pour données structurées. C'est le cas notamment du langage `SparQL` (Prud'hommeaux & Seaborne 2008) qui permet de sélectionner des triplets dans une base de triplets. Spécifier comment ces observés doivent se situer temporellement les uns par rapport aux autres constitue l'aspect *temporel* du problème de la représentation. Spécifier un ordre d'apparition entre les observés sélectionnés par la signature peut ne pas suffire. En effet, les observés peuvent se suivre mais sur une très longue durée, auquel cas il ne sera pas intéressant de les repérer comme occurrence de la signature. Ils peuvent également se suivre tout en ayant d'autres occurrences de la même signature qui apparaissent entre temps, ou se recouvrir mutuellement, etc. De plus, il peut être intéressant, et souvent même réaliste, de ne pas vouloir spécifier un ordre total entre l'apparition des observés pour cette signature. En représentant les signatures de tâche sous forme d'automates, Zarka et al. (2010) permettent aux signatures d'autoriser différents ordres d'apparition des observés sans permettre de quantifier les

[14] ou encore des *jokers*, souvent représentés dans les langages par « ∗ » ou « ? ».

écarts temporels entre ces observés. Toutes ces questions ont été posées par les scientifiques de la communauté de fouille de séquences (cf. chapitre 3), en particulier par ceux qui se posent le problème de la « découverte d'épisodes à partir de séquences d'événements » (Mannila et al. 1997).

Ces deux aspects de la représentation des signatures de tâche, structurel et temporel, sont traités par le modèle formel de Settouti et al. (2009) en tant que « M-trace pattern », mais aucun algorithme expliquant comment trouver toutes (ou un sous-ensemble) les occurrences de ce *pattern* n'est donné. La question de trouver une représentation de la signature de tâche avec des algorithmes associés de recherche d'occurrences de la signature dans la trace reste entière.

2.2 Collaboration humain/machine dans la construction des connaissances machine

Nous avons vu que l'origine de la connaissance machine est toujours humaine. De fait, l'humain participe nécessairement au processus de construction des connaissances machine, que ce soit par la traduction de connaissances de l'expert du domaine en modèle de connaissances ou dans la conception de SBC qui raisonne à partir de l'expérience. Mais de nombreuses techniques algorithmiques permettent à un SBC de mettre à jour automatiquement ses connaissances par apprentissage à partir de données. Cet apprentissage est « artificiel »[15], c'est-à-dire créé par l'humain, mais lorsque la conception d'un tel SBC se termine, ce SBC bénéficie généralement d'une certaine évolution et autonomie. Cependant, les connaissances apprises automatiquement peuvent ne pas être satisfaisantes pour au moins trois raisons :

- les données ne couvrent pas le domaine de manière représentative et donc les connaissances machine résultantes ne le couvrent pas non plus,
- la base de connaissances mise à jour peut être incohérente (cf. section 1.3.2),
- les situations intéressantes du domaine mais peu fréquentes vont avoir tendance à ne pas être prises en compte par la base de connaissances, qui se veut être une forme résumée.

L'intervention de l'humain, non seulement à la conception du SBC, mais aussi lors de chaque opération d'apprentissage est nécessaire dans l'optique d'un SBC qui se comporte de manière compréhensible par l'humain.

Dans cette section nous abordons deux approches de construction de connaissances mêlant l'humain et la machine. Ces deux approches ne sont pas « disjointes », c'est-à-dire qu'elles ne s'opposent pas l'une à l'autre. Il est nécessaire de les combiner.

2.2.1 Découverte de connaissances à partir de données

La « découverte de connaissances à partir de données » renvoie à ce que nous avons appelé méthode *ascendante* de construction de connaissances (cf. section 1.2.5.2), mais désigne un domaine précis de recherche en informatique[16]. Ce domaine de recherche utilise les techniques d'*apprentissage artificiel*, et d'ailleurs dans les grandes conférences et revues sur de ce domaine[17], de nombreuses

[15] *Machine Learning* en anglais

[16] en anglais, ce domaine s'intitule *Knowledge Discovery from Databases (KDD)*

[17] ACM SIGKDD Conference on Knowledge Discovery and Data Mining,
IEEE International Conference on Data Mining (ICDM),
le journal « Data Mining and Knowledge Discovery, Springer Netherlands »

contributions se focalisent davantage sur les algorithmes d'apprentissage automatique à partir de données que sur le processus d'acquisition de connaissances dans lequel ces algorithmes prennent part. Cependant les méthodes ascendantes de construction de connaissances ne se limitent pas aux méthodes et approches développées dans la communauté *Découverte de connaissances à partir de données* (KDD). Ce qui caractérise les méthodes proposée par la communauté KDD, c'est l'intégration dans le processus de construction de connaissances d'algorithmes de fouille de données, mais cela n'empêche pas qu'une méthode puisse être ascendante sans faire appel à la machine.

2.2.1.1 Le cycle KDD

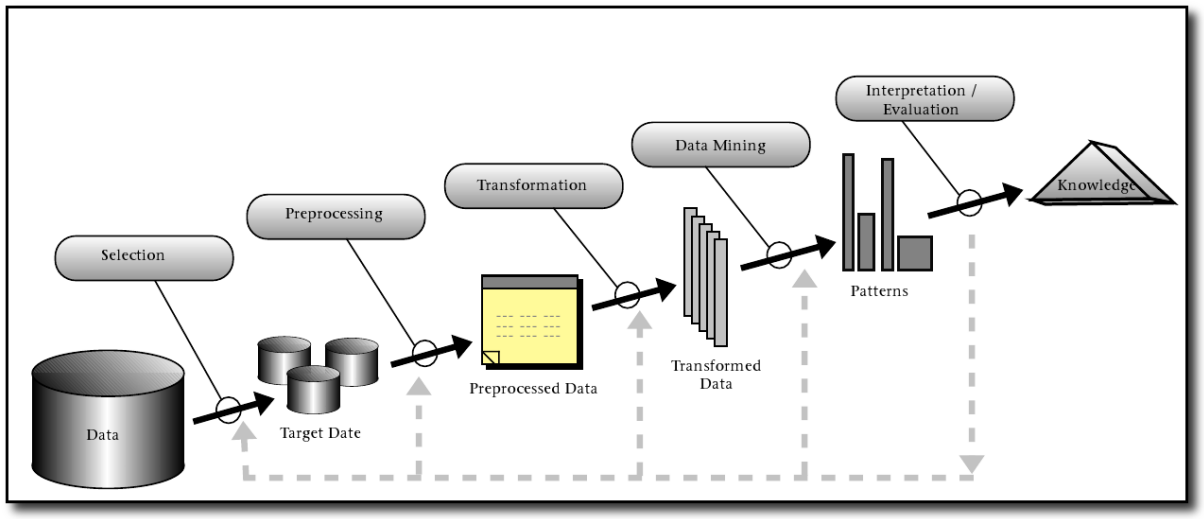


Figure 2.3 – Aperçu des étapes qui composent le processus cyclique de découverte de connaissances (cycle KDD) à partir de données (Fayyad et al. 1996)

Pour Fayyad et al. (1996), ce processus de découverte est interactif et se déroule en plusieurs phases (cf. figure 2.3). Il fait la distinction entre *fouille de données* et *découverte de connaissances*, la fouille de données n'étant que l'une des cinq phases qui compose la découverte de connaissances. L'humain qui prend part à ce processus de découverte de connaissances est communément appelé « utilisateur » par la communauté KDD, mais dans le but de ne pas confondre cet utilisateur qui découvre et construit des connaissances et les utilisateurs d'un environnement tracé (comme les utilisateurs de la plate-forme `CollaborativeECM`), nous désignerons dans ce mémoire tout humain qui participe à un processus de découverte et de construction de connaissances par « *analyste* ». Les cinq phases du processus de Fayyad et al. sont les suivantes.

Sélection Parmi l'ensemble des données accessibles, toutes ne concernent pas le champ de connaissance que l'analyste cherche à construire. L'étape de sélection consiste à ne retenir que les données dont on pense qu'elles sont potentiellement porteuses de connaissances du domaine à expliciter.

Prétraitement Cette étape consiste à supprimer le bruit et les imperfections parmi les données.

Transformation Cette étape consiste à mettre les données dans le format d'entrée de l'algorithme utilisé pour l'extraction de pattern. Il s'agit de choisir parmi les données sources une ou

plusieurs valeurs pour en construire une valeur des données transformées. Cette étape de transformation implique donc une sorte de sélection, c'est pourquoi il est parfois difficile de délimiter les contours de ces trois premières étapes et ces trois premières étapes sont parfois regroupées sous le chapeau unique de la *préparation des données*.

Fouille de données L'étape de fouille de données (*data mining* en anglais) consiste à appliquer des algorithmes d'extraction de motifs ou de modèles à partir de traces. Ces motifs ont pour vocation d'être simples (intelligibles rapidement à l'humain) et intéressants. C'est pourquoi, un algorithme de fouille de données n'extrait que des motifs dépassant un certain seuil d'intérêt par rapport à une mesure d'intérêt. Cette mesure d'intérêt est dans les cas les plus courants la *fréquence*, c'est-à-dire le nombre d'occurrences du motifs dans les données, car le concept de *fréquence* est très générique et peut s'appliquer à pratiquement tout type de motif, qu'il soit temporel, relationnel, sous forme de règle, etc. Cependant d'autres mesures d'intérêt peuvent s'appliquer (par exemple la confiance d'une règle d'association). C'est l'analyste qui définit le seuil d'intérêt qui est donné en entrée de l'algorithme.

Interprétation L'interprétation est une phase d'observation et d'analyse des motifs (*patterns*) retournés par l'algorithme de fouille. Parmi les souvent nombreux motifs retournés, beaucoup sont déjà connus par l'analyste, ou des corrélations triviales entre les données, ou encore redondants entre eux. Seuls quelques uns ont un réel intérêt pour l'analyste. Idéalement, des outils de post-traitement (filtrage, transformation, résumés, etc.) sont mis à disposition de l'analyste pour naviguer plus facilement dans ces données et mieux les comprendre. C'est après cette phase d'analyse des motifs par l'analyste que les motifs ayant été à la fois compris, pertinents et nouveaux pour lui sont considérés comme de la connaissance. Ces connaissances créées sont toujours des connaissances humaines, au sens où elles sont comprises par l'humain, mais souvent aussi des connaissances machine, au sens où le but de la découverte de connaissances est souvent de les capitaliser dans une base de connaissances utilisables par un SBC.

Les flèches de bouclage indiquent que ce processus est un cycle. Le bouclage est opéré par l'analyste. Au vu des motifs extraits par les algorithmes, il peut choisir de revenir sur l'une des étapes antérieures à la fouille de données. Si l'analyste n'est pas satisfait des motifs ou s'il souhaite approfondir un aspect particulier des données, il peut décider de revoir le choix de l'algorithme ou les paramètres d'entrée de celui-ci, ou modifier la préparation des données lors de l'une des trois premières étapes.

Le processus de construction de connaissance du point de vue de la communauté KDD est donc un processus de construction partagé où l'humain et la machine se partagent les tâches pour lesquelles ils sont chacun les meilleurs : l'humain oriente la recherche vers un but via la préparation et le paramétrage puis interprète les motifs retournés, et la machine explore systématiquement et le plus efficacement l'espace des motifs candidats en utilisant sa grande capacité de mémoire et sa rapidité de calcul.

2.2.1.2 Périmètre de bouclage

La plupart des travaux de recherche du domaine KDD visent à chercher des nouvelles structures de motif de manière la plus économique possible en nombre de calculs, en espace mémoire occupé et en temps global. Ces travaux s'inscrivent la plupart du temps dans ce cycle et présupposent des étapes d'interaction avec l'humain lors des phases de préparation et d'interprétation, mais rares sont les algorithmes qui permettent à l'humain d'agir à l'intérieur de la phase *fouille de données*,

c'est-à-dire au sein du processus de découverte automatique de motifs. En général l'analyste doit attendre la fin de l'exécution de l'algorithme de fouille pour disposer des résultats, effectuer une interprétation de ceux-ci et agir en conséquence sur l'une des phases antérieures et relancer la boucle. On dira dans la suite de ce mémoire que plus la phase est antérieure, plus le périmètre de bouclage est grand. Le périmètre de boucle désigne la quantité de travail (de préparation, de configuration ou algorithmique) à effectuer avant d'obtenir de nouveaux résultats en provenance de l'algorithme de fouille.

À la vue du cycle KDD proposé par Fayyad et al., le bouclage le plus court possible consiste à modifier le choix de l'algorithme ou ses paramètres avant de le relancer. Nous qualifions cette boucle de « temps réel » lorsque le temps d'attente de l'analyste avant d'intervenir à nouveau dans le cycle, c'est-à-dire avant de recevoir de nouveaux résultats à interpréter, est nul ou de quelques secondes. Or il n'est pas gagné que le bouclage le plus court qui soit dans le cycle KDD soit temps réel. En effet, de nombreux algorithmes de fouille de données nécessitent plusieurs minutes, heures, voire journées pour s'exécuter et proposer tous les motifs à l'analyste. C'est une pratique courante que de laisser tourner ces algorithmes sur un temps très long puis de revenir ensuite pour analyser les résultats. Dans ces cas-là, l'exécution sur un temps long est nécessaire pour extraire l'ensemble des motifs satisfaisant les paramètres d'entrée. Cependant lorsque le bouclage est temps réel, l'analyste entre en interaction avec la machine, ce qui dans l'esprit de l'analyste peut permettre d'élever le système de découverte au rang d'agent informatique avec lequel il peut discuter pour découvrir des connaissances.

2.2.1.3 Découverte orientée par des connaissances préalables

Cette dernière décennie, un nouveau volet de la découverte de connaissances s'est développé. Michalski (2003) qualifie ce volet de « fouille de connaissances » (*Knowledge Mining*) et explique que les nouveaux défis du domaine ne consiste plus seulement à être capable d'extraire des nouvelles structures de motifs à partir de données (cf. équation 2.1) et à améliorer les performances des algorithmes qui implémentent cette extraction, mais aussi à être capable de tirer profit des motifs extraits pour construire de la connaissance. Pour savoir si un motif fréquent est une connaissance pour l'analyste, il faut connaître d'une part les intentions de l'analyste lors de sa découverte et d'autre part les connaissances qu'il a de ce domaine (cf. équation 2.2).

$$DATA \rightarrow PATTERNS \tag{2.1}$$

$$DATA + PRIOR\_KNOWLEDGE + GOAL \rightarrow NEW\_KNOWLEGDE \tag{2.2}$$

Fauré (2007) applique ce principe en proposant de capitaliser des règles d'association extraites et sélectionnées comme intéressantes par l'analyste dans un réseau bayésien, qui constitue la base de connaissances du domaine dans une représentation probabiliste. Chaque nouvelle règle d'association découverte permet la mise à jour du réseau bayésien. Ce réseau bayésien peut être utilisé ensuite dans le processus d'extraction de règles pour rechercher exclusivement des règles qui seront « nouvelles », c'est-à-dire non déjà résumées dans le réseau bayésien d'entrée. Il peut également être mis à profit dans la présentation des règles d'association extraites à l'analyste (post-traitement) en indiquant le degré d'intérêt de chaque règle au vu des connaissances déjà acquises. Le processus de découverte de règles permet donc d'améliorer le réseau bayésien, et ce même réseau bayésien permet d'améliorer le processus de découverte ; l'ensemble du cycle constitue une boucle vertueuse. Les connaissances données en entrée permettant d'orienter l'exploration vers des candidats intéres-

sants plus rapidement, d'augmenter l'efficacité et la pertinence des motifs extrait, et de les élever automatiquement au rang de connaissance.

2.2.2 Acquisition opportuniste de connaissances

L'*acquisition opportuniste* de connaissances ne désigne pas une méthode d'acquisition de connaissances comme l'est la découverte de connaissances à partir de données. L'*acquisition opportuniste* désigne une stratégie générique, qui peut s'appliquer à n'importe quelle méthode d'acquisition de connaissances, consistant à profiter du fait qu'un SBC nécessite une interaction avec l'humain pour lancer un processus de construction de connaissances visant à mettre à jour la base de connaissances du SBC (Cordier 2008, Badra et al. 2009). Dans la collaboration humain/machine, l'acquisition opportuniste vise à favoriser les échanges entre humain et machine en les initiant dès que l'opportunité s'en présente pour la machine, c'est-à-dire dès que l'humain est au contact d'une interface du SBC. Les travaux portant sur l'acquisition opportuniste de connaissances apportent la preuve que les interactions multiples et répétées au cours de la vie du SBC entre l'humain et la machine servent l'évolutivité et la pertinence toujours plus grande des SBC.

Cordier (2008) applique ce principe dans le cadre du Raisonnement à Partir de Cas (RàPC, cf. section 1.2.4.2). Un SBC utilisant le paradigme du RàPC cherche à résoudre des problèmes nouveaux en s'appuyant sur une base de cas de résolution de problème passés. Une phase critique de ce processus est la phase d'*adaptation* d'un cas de la base proche du problème courant à ce dernier, car la pertinence de l'assistance fournie par le SBC en dépend directement. Cette phase d'adaptation nécessite des connaissances particulières d'adaptation en plus des connaissances générales du domaine. La phase suivant l'adaptation dans le RàPC, c'est-à-dire la *révision*, nécessite une interaction avec l'analyste du SBC pour lui demander de corriger le cas adapté automatiquement par le SBC. Cette phase demande à l'analyste de mobiliser ses propres connaissances sur ce qu'aurait du être l'adaptation de ce cas. Cordier propose de profiter de cette interaction nécessaire avec l'humain pour lancer une boucle d'acquisition de connaissances d'adaptation en cas d'échec de la phase d'adaptation. Au lieu, de simplement traduire ses connaissances de ce qu'aurait du être la phase d'adaptation en nouveau cas à retenir, l'analyste est invité à formaliser les connaissances d'adaptation qu'il a mobilisées pour établir le cas révisé. Les connaissances d'adaptation ainsi formalisées sont retenues dans une base de connaissances d'adaptation et le SBC recommence immédiatement la phase d'adaptation avec ces nouvelles connaissances et ainsi de suite jusqu'à ce que le SBC ne se trompe plus. Ce principe d'acquisition opportuniste de connaissances d'adaptation a été implémenté dans les prototypes IAKA (Cordier et al. 2008), un démonstrateur mathématique de ce principe sans aucun intérêt applicatif, et FRAKAS (Cordier et al. 2007). FRAKAS est le module d'acquisition opportuniste de connaissances d'adaptation d'un SBC assistant le diagnostic médical en cancérologie. Les connaissances d'adaptation sont une base d'assertion sur le domaine formulées en logique propositionnelle.

L'enseignement retenu de ces deux prototypes est que le principe d'acquisition opportuniste est très prometteur, car il permet au SBC de disposer d'une base de connaissances qui soit à la fois à jour et pertinente du point de vue de l'analyste. L'acquisition opportuniste permet de capitaliser l'ensemble des connaissances humaines nécessaires au bon comportement du SBC en fonction des besoins, c'est-à-dire que seules les connaissances nécessaires seront modélisées. Ceci limite la modélisation d'aspects du domaine qui ne sont pas utiles au SBC, ce qui réduit le volume des connaissances machine et le risque d'incohérence entre connaissances. Cependant, lors de l'ajout d'une nouvelle connaissance par le mode opportuniste, il se peut que l'analyste soit amené à résoudre des incohérences dans les connaissances machine entre la connaissance nouvellement acquise et la

base de connaissances déjà en mémoire. Il peut alors devenir très contraignant pour l'analyste de résoudre le conflit,car cela nécessite à la fois d'avoir des connaissances logiques pour comprendre l'origine de l'incohérence et d'avoir du temps pour résoudre le conflit de manière pérenne. En effet, l'acquisition se voulant « opportuniste », c'est-à-dire au fil de l'activité première de l'analyste lorsque que le SBC a besoin de mettre à jour sa base, il n'est pas acceptable que ce processus prenne trop de temps à l'analyste. Quant à l'implémentation de ce principe, il est très difficile d'implémenter les interfaces humain/machine qui supportent ce processus de co-construction de connaissances. Pour IAKA, l'interface est la ligne de commandes et nécessite de l'humain qu'il connaisse la syntaxe du langage, ce qui n'est pas aisé pour des experts qui ne sont pas informaticiens. Pour FRAKAS, l'interface était graphique, mais elle devait représenter les propositions logiques de la base, permettre la formulation de nouvelles formules en utilisant si possible les propositions déjà présentes dans la base, expliquer et afficher les origines des incohérentes en cas de conflit. Réaliser de telles interfaces s'avère compliqué en pratique et n'exonère pourtant pas l'analyste d'être formé dans le langage logique utilisé pour la représentation des connaissances.

2.2.3 Co-construction basée sur les traces

Dans ses travaux sur l'assistance aux utilisateurs d'un environnement informatique, Stuber (2007) propose de construire « à partir de zéro » et en négociation un langage compris à la fois par la machine et l'utilisateur. Pour Stuber, l'*utilisateur* de l'environnement et l'*analyste* ne font qu'un. C'est en effet l'utilisateur de l'environnement qui participe à la construction de connaissances à partir de traces. Ce qui fait guise de connaissance machine ici, c'est le langage lui-même, utilisé à la fois par l'utilisateur/analyste et le système pour *parler* de l'activité.

Nous utilisons le terme « à partir de zéro », car le langage partagé n'est initialisé avec aucun élément. Cependant, l'expérience d'utilisation, qui est représentée par les traces d'interactions, vient alimenter ce processus. Au fil des situations rencontrées par l'utilisateur dans son utilisation du système, il rencontre des situations à problème nécessitant une assistance. À chaque fois que l'utilisateur sollicitera le module d'assistance, un processus d'élaboration de sens commun s'engage entre lui et l'utilisateur. Le langage commun est constitué de signatures de tâche. Au vu de la situation courante, si le langage commun est non vide, alors le module d'assistance proposera la signature de tâche la plus appropriée pour qualifier la situation courante, et ce même module activera l'assistance associée à cette signature de tâche. Si la signature proposée ne convient pas à l'utilisateur, alors l'utilisateur peut modifier la signature proposé par le système pour qu'elle décrive de manière plus juste la situation courante. Si le langage est vide, l'utilisateur élaborera depuis zéro une signature la décrivant. L'élaboration ou la modification de cette signature représentant la situation courante est effectuée par l'utilisateur en visualisant la trace d'interactions, ce qui lui permet de regarder la pertinence de la signature qu'il élabore par rapport à la situation courante et à d'autres situations passées.

À mesure que le système est utilisé, que le module d'assistance est sollicité et modifié, le langage commun est plus riche, plus précis, et plus apte à caractériser les différentes situations d'utilisation qui ont lieu dans le système. Ce langage commun fait en effet guise de connaissance puisque plus le langage sera fourni et plus le système pour agir et fournir de l'assistance pertinente à l'utilisateur. Stuber parle d'émergence de sens entre l'utilisateur et le système car à l'origine le langage commun est vide. Ce processus d'élaboration du langage commun tout au long de la vie du SBC (en l'occurrence ce SBC est le module d'assistance) par confrontation perpétuelle au nouvelles situations, puis à leur révision par l'utilisateur fait penser au RàPC. Ce cycle de raisonnement est appelé « Raisonnement à Partir de l'Expérience Tracée », ou plus simplement

« RàPET » (Mille 2006, Cordier et al. 2009). Il a les mêmes objectifs que le cycle de RàPC, c'est-à-dire construire un SBC capable de faire face à de nouvelles situations dans un domaine où les connaissances *a priori* sont faibles, mais s'efforce d'en amoindrir les inconvénients. En l'occurrence, l'inconvénient majeur du RàPC est qu'il nécessite de définir *a priori* une structure de cas pour tous les cas futurs et que cette structure est difficile à remettre en cause au gré des nouvelles situations rencontrées de la même manière qu'il est difficile de remettre en cause un modèle de connaissances (cf. section 1.2.4.2). Le RàPET autorise une structure de cas différente pour chaque cas. Cette structure de cas est établie par l'utilisateur dans une phase d'*élaboration* préalable à toutes les autres phase du cycle de raisonnement.

Si Stuber propose une approche à bouclage court qui permet la co-construction de connaissances à partir de traces, la contribution du système lors de la génération de nouvelles signatures se limite à effectuer un retour sur les élaborations de signatures effectuées par l'utilisateur. Ce retour consiste à montrer les occurrences dans la trace d'interactions de la signature qu'il élabore afin que l'utilisateur évalue la pertinence de cette signature. Les signatures étant des structures complexes et donc difficiles à manipuler, ce travail d'élaboration peut être laborieux pour l'utilisateur. C'est pourquoi des techniques permettant au système de prendre part pro-activement à l'élaboration dans le RàPET, afin de limiter la tâche de l'utilisateur, restent à proposer.

2.3 Processus itératif de co-construction de signatures de tâche à partir de traces

Dans cette section, nous présentons le processus de co-construction de connaissances à partir de traces dans lequel s'inscrit le travail algorithmique détaillé dans la suite de ce mémoire. Cette approche de co-construction de connaissances s'effectue sur le principe de l'abstraction des traces. C'est pourquoi nous définissons en premier lieu ce concept d'*abstraction de trace* et présentons un travail existant en la matière.

2.3.1 Approche de co-construction de connaissances par abstraction de traces

2.3.1.1 Les niveaux d'abstraction de la trace

Lorsqu'on parle de transformation de traces d'interactions, on parle souvent de « transformation d'abstraction » pour désigner l'opération qui prend une trace constituée d'observés en provenance des capteurs et produit une trace décrivant l'activité à un niveau « plus élevé », c'est-à-dire utilisant des éléments du langage humain. Que voulons nous dire par « abstraction » ? Le `larousse.fr` définit l'abstraction comme une « opération intellectuelle qui consiste à isoler par la pensée l'un des caractères de quelque chose et à le considérer indépendamment des autres caractères de l'objet ». Cette définition n'a pas beaucoup de rapport avec la notion d'abstraction véhiculée d'une transformation d'abstraction.

Dans son analyse des interactions utilisateur/système, Hilbert & Redmiles (2000) distinguent six niveaux d'abstraction différents (cf. figure 2.4). Les interactions de niveaux d'abstraction les plus élevés tendent à décrire l'activité à des niveaux proches du vécu de l'utilisateur dans cette activité (ses buts, ses tâches, etc.) alors que les niveaux les plus bas tendent à décrire les interactions à un niveau proche du logiciel et matériel, c'est-à-dire de l'outil qui médie l'activité de l'utilisateur. Chaque observé d'un niveau d'abstraction est obtenu par composition d'observés du niveau inférieur. Par exemple, un observé du niveau d'abstraction 4 (le *niveau d'interaction abstraite*) de type « entrer une valeur » peut être composé des observés du niveau 3 (le niveau *événements de l'interface*

utilisateur) suivants : « déplacement du pointeur sur le champ », « click sur le champ pour lui donner le focus », « enfoncement de la touche 1 », puis « enfoncement de la touche 0 ». Cette succession d'observés de niveau 3 décrit l'entrée de la valeur 10. De même au niveau 5, l'observé « donner son adresse » est composé des observés de niveaux 4 suivants : « entrer une valeur dans le champ `Rue` », « entrer une valeur dans le champ `Code Postal` », « entrer une valeur dans le champ `Ville` », puis « Valider les champs » (en cliquant sur le bouton `Ok`).

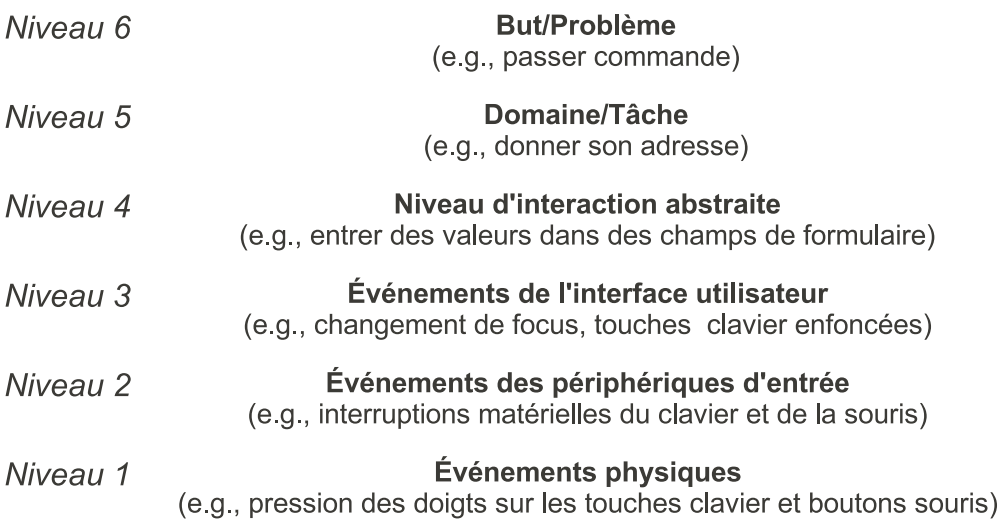


Figure 2.4 – Les six niveaux d'abstraction des interactions utilisateur/système (Hilbert & Redmiles 2000)

Sur la base des travaux de Hilbert & Redmiles (2000), on peut définir les transformations d'abstraction de traces comme une transformation vérifiant les deux propriétés suivantes :

1. la trace transformée est constituée d'observés chacun construit par composition d'un ou plusieurs observés de la trace source,
2. la trace transformée est constituée d'observés répartis temporellement avec un grain plus proche de celui perçu par l'utilisateur que celui de la trace source.

Le point 1 est vrai pour toute transformation de trace, mais insiste sur la notion de « composition » qui définit l'abstraction telle que nous l'entendons en ingénierie des traces. En pratique il existe peu de transformations de traces qui ne vérifient pas le point 2, c'est-à-dire qui ne soient pas construites pour servir la compréhension de la trace transformée par l'humain. Cela s'explique par le fait que les motivations initiales du SBT et des transformations de trace étaient de produire des traces d'interactions servant d'objet informatique descripteur de l'activité qui soit intermédiaire entre le système et l'utilisateur, c'est-à-dire qui soit compris à la fois par la machine et par l'humain.

Le terme « abstraction » n'est peut-être pas le plus adapté au vu de son sens commun dans la langue française, mais nous le conservons car il est adopté par la communauté d'analyse et d'ingénierie des traces. Les termes « interprétation » ou « reformulation » pourraient également être utilisés. L'atelier ABSTRACT, présenté dans la sous-section suivante, permet à l'humain d'effectuer des transformations d'abstraction pour les traces d'interactions, produisant au final une trace abstraite qui décrit l'activité d'un humain à partir de traces de bas niveau d'abstraction.

2.3.1.2 L'atelier ABSTRACT

ABSTRACT[18] (Analysis of Behavior and Situation for menTal Representation Assessment and Cognitive acTivity modeling) est un outil logiciel qui a été développé dans le cadre d'un projet d'analyse de l'activité de conduite à partir des traces d'interactions de celle-ci (Georgeon 2008*a*). Un véhicule instrumenté a été conduit par des sujets d'expérimentation pendant plusieurs heures. Ce véhicule possède de nombreux capteurs (occulomètre, capteur de pression sous chaque pédale, capteur d'angle au volant, de changement de vitesse, etc.). Les données en provenance de ces capteurs ont été collectées et les données numériques ont été transformées par réduction en données symboliques (e.g., une courbe numérique d'angle du volant en fonction du temps donnera des événements symboliques de type *droite*, *gauche* ou *tout-droit*, avec les dates associées après transformation). Des analystes, experts en transport et en psychologie du conducteur, cherchent ensuite à caractériser des modèles de comportement du conducteur.

La plate-forme ABSTRACT est une plate-forme d'analyse des traces symboliques au format RDF. Le format de la trace est inspiré de la théorie des M-traces de Settouti et al. (2009) dans la mesure où :

- chaque trace ABSTRACT possède un modèle de trace,
- les *observés* de la trace ABSTRACT possèdent une extension temporelle (une date ponctuelle),
- les événements possèdent de nombreuses paires attribut/valeur.

Si la plate-forme ABSTRACT a été conçue pour l'analyse des traces de conduites automobile, elle peut sans problème être appliquée à n'importe quel autre trace respectant son format.

Le principe de l'analyse de traces avec ABSTRACT est d'abstraire les traces collectées. Les traces abstraites servent ensuite de support d'analyse de l'activité tracée. Dans le cadre de l'activité de conduite, ces traces peuvent aussi être visualisées en synchronisation avec les captures vidéos de l'activité de conduite, si bien que l'analyste peut confronter les actions abstraites observées dans la trace à leur contexte et aux effets qu'elles ont produits.

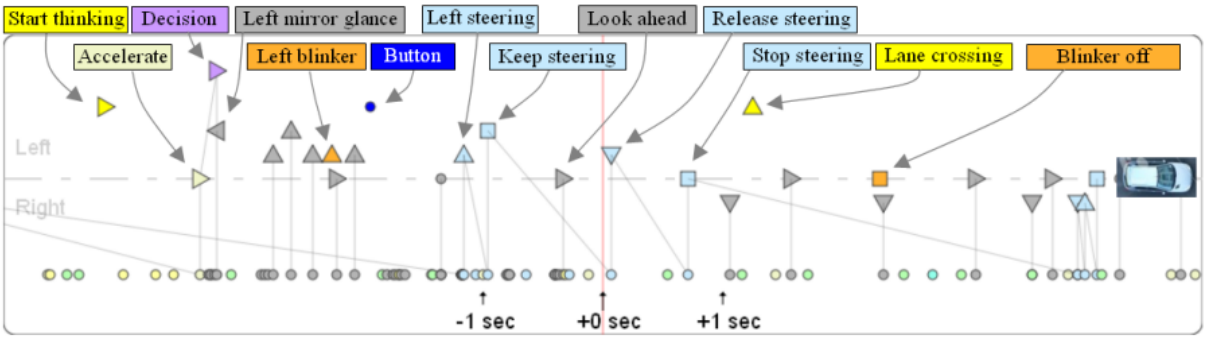


Figure 2.5 – Visualisation d'une trace ABSTRACT et de ses observés transformés

La figure 2.5 montre l'interface de visualisation de trace de ABSTRACT. La trace visualisée sur cette figure est composée d'observés « premiers » (c'est-à-dire appartenant à la trace première) et d'observés abstraits, construits à partir d'observés premiers. Les observés premiers sont les cercles alignés sur le bas du graphique. Les observés abstraits sont tous les observés (cercles, triangles, carrés) situés au dessus de ces observés premiers. Dans ce type de représentation de la trace, l'axe

[18] http://liris.cnrs.fr/abstract/abstract.html

des abscisses représentent le temps linéaire ; l'axe des ordonnées représentent en partie[19] le degré d'abstraction de la trace. Des liens gris relient les observés abstraits aux observés à partir desquels ils sont construits.

Le processus par lequel l'analyste de l'activité de conduite abstrait les traces avec ABSTRACT est cyclique. Il se déroule en trois étapes (Mathern 2006) :

1. l'analyste pense à un modèle de comportement de conduite qu'il souhaite pouvoir caractériser à l'aide des traces. Il formule ce modèle de comportement en motif d'observés de la trace (c'est-à-dire en signature de tâche) à l'aide du langage `SparQL` sur lequel nous reviendrons par la suite.

2. Le système parcourt la trace à la recherche de toutes les occurrences de ce motif et affiche ces occurrences à l'analyste.

3. L'analyste observe ces occurrences de motif sur l'interface de visualisation de trace de la figure 2.5 et en synchronisation avec la vidéo de cette occurrence du motif, puis il juge si les occurrences du motifs qu'il a formulé correspondent au schéma cognitif qu'il avait en tête. S'il s'avère que le motif ne correspond pas, alors l'analyste revient à l'étape 1 et modifie la formulation du motif en fonction. Sinon, le cycle s'arrête et l'analyste utilise le motif pour construire un nouveau type d'observé plus abstrait dans le modèle de trace et instancier ce nouvel observé à chacune des occurrences du motif qu'il représente.

Le langage dans lequel l'analyste formule un motif et dans lequel il spécifie comment créer des observés plus abstraits à partir de ceux-ci est `SparQL`. Le langage `SparQL` est un langage de requête sur des données organisées en triplets (sujet-prédicat-objet) comme le sont les données RDF. `SparQL` permet de sélectionner des triplets dans un ensemble de triplets par des contraintes sur leur sujet, prédicat ou objet. Formuler un motif dans les traces ABSTRACT revient donc à formuler une requête `SparQL`. L'avantage du langage `SparQL` est qu'il permet de créer de nouveaux triplets à partir des triplets requêtés. Une requête `SparQL` contient donc deux parties (cf. figure 2.6) : une partie « sélection » (`WHERE`) et une partie « construction » (`CONSTRUCT`) qui est optionnelle . Les triplets ainsi créés forment la représentation symbolique du nouvel observé plus abstrait.

Une fois ce nouvel observé abstrait ajouté à la trace, l'analyste peut recommencer son cycle d'analyse sur cette nouvelle trace pour trouver ainsi des observés de plus en plus abstraits.

La plate-forme ABSTRACT a beaucoup apporté à l'analyse de l'activité de conduite en ce qu'elle aidait à caractériser formellement à partir des traces des modèles de comportement du conducteur.

Cependant, ce processus d'abstraction de traces présente aussi ses limites. Un observé étant constitué de plusieurs triplets et une trace de très nombreux observés, la quantité de triplets à parcourir par le moteur d'exécution des requêtes `SparQL` est très grand. En conséquence, les performances ne sont pas toujours satisfaisantes, car l'analyste doit souvent attendre plusieurs secondes, voire minutes, pour espérer obtenir un premier retour de la part de l'algorithme. Cela s'explique par le fait que les requêtes formulées par l'expert sont souvent « temporelles », c'est-à-dire qu'elles comportent des contraintes temporelles sur les observés sélectionnés (e.g. tel observé doit apparaître avant tel autre observé, tels observés doivent apparaître dans un intervalle temporel

[19] Il représente bien le degré d'abstraction, mais une différence de position relative par rapport à l'axe pointillé horizontal sera faite selon que l'observé abstrait représente une action ayant un rapport avec la gauche (comme « tourner à gauche », « regarder à gauche ») ou un rapport avec la droite. L'image de la voiture sur la droite de l'axe montre alors quelle est le sens de la marche pour la lecture de ces observés abstraits. Malgré tout, les observés abstraits de degré 2, c'est-à-dire étant construit à partir d'au moins un observé abstrait seront positionnés encore au-dessus de tous ces observé droite/gauche abstraits de degré 1.

```
# Decelerate
PREFIX myfn:    <java:com.ldodds.sparql.>
PREFIX kb:      <http://protege.stanford.edu/kb#>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xs:      <http://www.w3.org/2001/XMLSchema#>

CONSTRUCT {
    ?r1    kb:inferred  _:a.
    _:a    a            kb:Decelerate.
    _:a    kb:date      ?d1
}
WHERE {
    ?r1    a            kb:Accelerator.
    ?r1    kb:subtype   "Threshold_Down".
    ?r1    kb:date      ?d1
}
```

Figure 2.6 – Exemple de requête SparQL. La partie `WHERE` est la partie sélection de triplet et la partie `CONSTRUCT` est la partie qui contruit l'observé abstrait, c'est-à-dire un ensemble de triplets qui constitue cet observé, à partir des triplets sélectionnés. Dans cette requête, l'attribut `a` définit le type d'observé. Cette requête construit un observé de type `kb:decelerate` (arrêt de l'accélération du conducteur) lorsque le seuil d'enfoncement de l'accélérateur par le conducteur n'est plus dépassé. La date de l'observé `kb:decelerate` est la même que celle de l'observé premier `kb:Accelerator`.

d'une minute, etc.). Or, le langage SparQL et ses moteurs ne prévoient aucun traitement particulier pour les aspects temporels des données. Les attributs temporels des observés sont donc traités de manière parfaitement identique à tout autre type d'attribut, et les performances s'en ressentent.

L'autre inconvénient de l'approche ABSTRACT est que dans la co-construction du motif de comportement, le système ne joue aucun rôle génératif du motif. De même que pour le module d'assistance de Stuber basé sur le RàPET (cf. section 2.2.3), le système ne joue pas d'autre rôle que de trouver les occurrences du motif élaboré par l'analyste mais ne joue aucun rôle créatif dans l'élaboration des motifs. En conséquence, il n'est pas possible pour le système de suggérer des motifs qui pourraient susciter un intérêt nouveau pour l'analyste. L'analyste n'effectuera son cycle d'analyse que dans l'optique des modèles de comportement qu'il prévoit et préssent, alors que la trace regorge potentiellement de nombreuses signatures intéressantes et non envisagées par l'analyste.

2.3.2 Proposition d'un processus avec système pro-actif

Nous proposons comme cadre de travail pour la suite de cette thèse, un processus de co-construction de connaissances à partir de trace inspiré du processus ABSTRACT, mais avec un système qui soit pro-actif dans la génération des motifs. De même que ABSTRACT, le processus que nous proposons est itératif, constitué d'abstractions successives. En entrée de ce processus, on donne la trace première d'une activité tracée. Cette trace première est considérée de degré d'abstraction k. La figure 2.7 illustre une itération de ce processus d'abstraction, c'est-à-dire comment une trace de degré d'abstraction k est abstraite en trace de degré $k+1$.

L'itération se déroule comme suit. L'analyste est un humain qui souhaite abstraire les traces d'interactions collectées. Cet analyste peut être l'utilisateur-même de l'environnement tracé. Cette deuxième possibilité implique que l'utilisateur/analyste possède les connaissances informatiques nécessaires à la réalisation du processus d'abstraction, c'est-à-dire certaines compétences en ingénierie des connaissances mais surtout en fouille de données (cf. chapitre 5). L'analyste charge

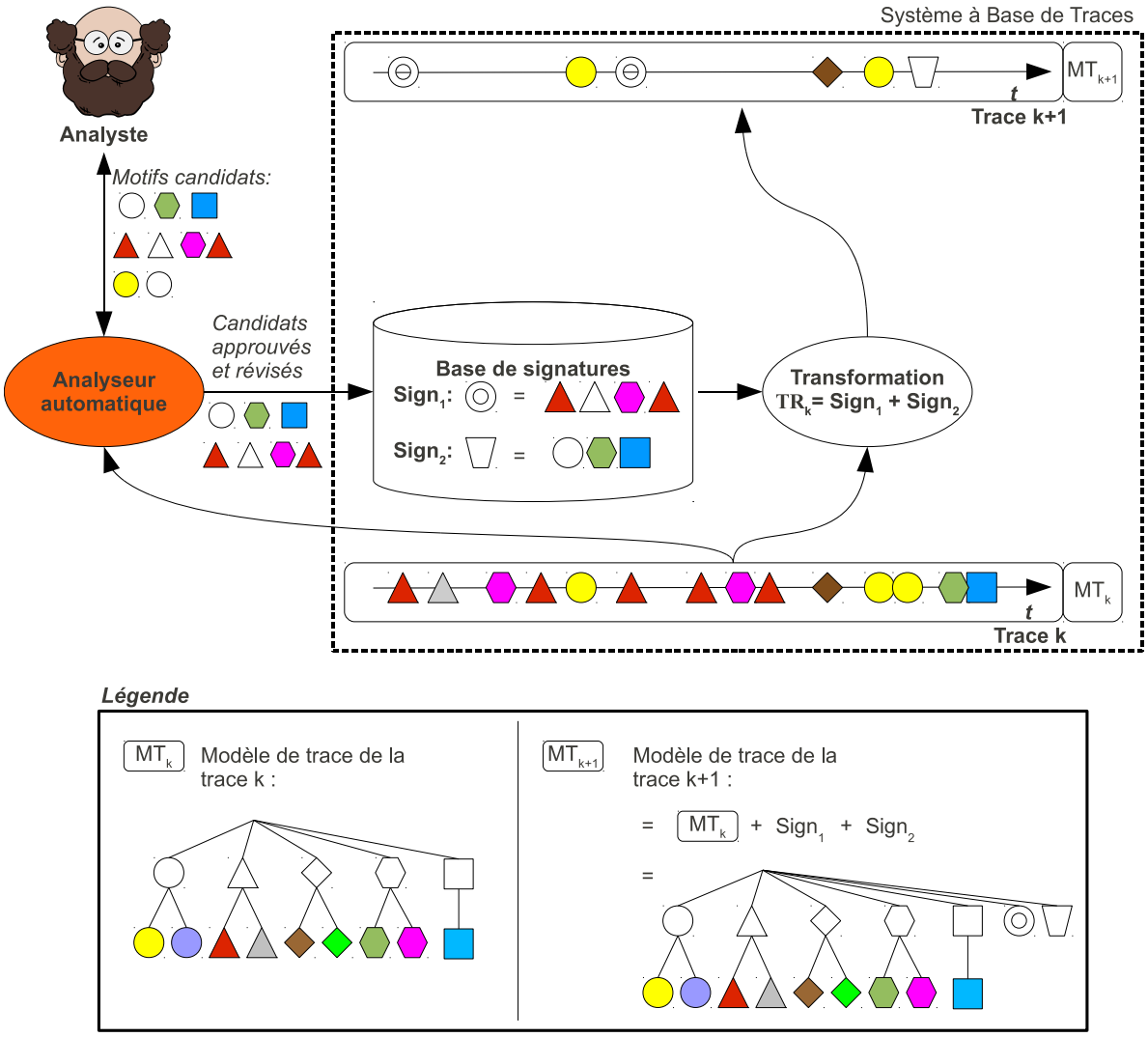


Figure 2.7 – Une itération du processus de co-construction de connaissances à partir de traces (Cram et al. 2008). Dans les modèles de trace, les relations entre types d'observé sont des relations de subsomption. La convention adoptée sur cette figure est de représenter les types d'observé les plus généraux par des symboles blancs. Les spécialisations de chaque type d'observé sont représentées avec le même symbole mais en couleurs.

une trace d'interactions de degré d'abstraction k ($k = 0$ si cette trace est la trace première). Un *analyseur automatique* fouille la trace k et extrait des motifs potentiellement intéressants pour l'analyste, c'est-à-dire des motifs candidats à l'abstraction. Si l'un des motifs candidats fait sens pour l'analyste, c'est-à-dire correspond à l'exécution d'une tâche reconnue par l'analyste, l'analyste peut approuver ce motif ou l'approuver après révision (c'est-à-dire après légère modification). S'il ne l'approuve pas, il formule une requête à l'analyseur automatique. Cette requête peut être plus ou moins contrainte, c'est-à-dire demander aux motifs extraits de respecter plus ou moins de critères. Ces critères demandés par l'analyste peuvent être élaborés au vu des défauts qu'il a constatés concernant les premiers motifs candidats extraits. Les motifs candidats approuvés sont considérés comme étant des signatures de tâche et sont ajoutés à une base de signatures. Cela signifie que la structure des motifs recherchés dans la trace par l'analyseur automatique est une structure de signature de tâche. Pour les nouvelles signatures de tâche ajoutées à la base ($Sign_1$ et $Sign_2$ sur la figure), on définit une transformation de trace TR_k composée de ces deux signatures. Le principe de transformation est de remplacer chaque occurrence d'une signature par un observé d'un nouveau type, chaque signature de la transformation TR_k possédant un nouveau type d'observé propre (ces nouveaux types d'observé sont représentés par le double cercle pour $Sign_1$ et par le trapèze pour $Sign_2$). On obtient ainsi une trace $k+1$ abstraite par les signatures qui composent la transformation TR_k. Les nouveaux types d'observé plus abstraits sont ajoutés au modèle de trace de la trace $k+1$.

Les connaissances construites sont les nouvelles classes d'observé du modèle de trace, chacune d'entre elles pouvant être considérée comme *abstraite* car définie à partir d'observés d'abstraction de niveau inférieur, et en bout de chaîne, à partir d'observés de la trace première. Ces classes abstraites représentent des signatures de tâche. Pour un SBC utilisant le RàPET, une classe abstraite (c'est-à-dire une signature de tâche) est une connaissance machine, car nous avons défini la connaissance machine comme une information permettant au système d'agir et que cette signature de tâche peut justement être utilisée pour agir dans le cadre du RàPET. Pour un SBC utilisant le RàPET, une classe abstraite (ou signature de tâche) est donc une connaissance de la même manière qu'un cas est une connaissance pour un SBC utilisant le RàPC. Le processus présenté ci-dessus et sur lequel nous nous appuyons dans la suite de ce manuscrit est donc, dans le cadre d'un SBC utilisant le RàPET, un processus de co-construction de connaissances machine.

2.3.3 Cahier des charges de l'algorithme implémentant l'analyseur automatique

Du fait de l'existence du Système à Base de Traces développé dans le cadre du projet PROCOGEC, les problématiques liées à la collecte de la trace première, à leur stockage dans le SBC, à leur récupération par des modules externes tel l'analyseur automatique, et les problématiques liées aux transformations ne sont pas abordées dans cette thèse. Nous nous focaliserons uniquement sur la conception d'un tel analyseur automatique capable de recevoir en entrée une trace d'interactions et de proposer en sortie des motifs candidats à l'abstraction.

Plus précisément, l'analyseur automatique est un algorithme de fouille de traces qui doit posséder les propriétés suivantes :

- les données qu'il fouille sont des traces modélisées ou des données dont le format est proche ou assimilable ;
- les motifs recherchés doivent pouvoir être considérés comme des signatures de tâche, c'est-à-dire qu'ils doivent spécifier la présence de certains observés et imposer des contraintes temporelles entre ces observés ;

- l'algorithme doit être complet, c'est-à-dire s'engager à ne « passer sous silence » aucun motif de la trace répondant aux critères de l'analyste. Nous imposons ce critère, car nous voulons qu'un motif potentiellement intéressant pour l'analyste ne puisse pas échapper à l'abstraction sous prétexte qu'il ne fait pas partie de l'arbre d'exploration de motifs de l'algorithme ;
- s'il le souhaite, l'analyste doit pouvoir formuler des contraintes temporelles et structurelles expressives sur les motifs recherchés, pour que l'algorithme oriente sa fouille vers les motifs les plus intéressants pour l'analyste ;
- bouclage temps réel, au sens où nous l'entendons dans la section 2.2.1.2, de sorte que le processus soit assimilable à un discours en temps réel pour l'humain. Le bouclage temps réel implique que l'algorithme de fouille soit très efficace ou qu'il soit construit de telle sorte que des résultats partiels soient accessibles à l'analyste en temps réel.

Le chapitre 3 dresse l'état de l'art des méthodes automatiques d'extraction de motifs à partir de données semblables aux traces.

Chapitre 3

État de l'art sur l'extraction de motifs temporels à partir de traces

Sommaire

Ce chapitre fait le bilan d'un certain nombre de travaux de la littérature concernant la fouille de données temporelles, et plus particulièrement ceux concernant l'extraction de motifs temporels, visant à découvrir des connaissances comportementales dans des contextes et avec des objectifs proches de notre question de recherche. Cet état de l'art ne se veut pas exhaustif mais se propose de présenter un résumé des techniques existantes dans ce domaine par rapport aux objectifs de notre analyseur automatique. Nous présenterons les différents concepts abordés de manière informelle mais en mettant en évidence les propriétés principales qu'ils offrent au regard de nos objectifs. Ces

concepts seront repris et formalisés de manière détaillée pour notre proposition d'algorithme dans le chapitre 4.

La section 3.1 explore les branches du domaine de la fouille de données temporelles qui sont en relation avec les objectifs de notre recherche. En section 3.2, nous rappelons les formulations de problèmes d'extraction de motifs temporels les plus connues et les mieux adaptées à notre approche pour un analyseur automatique. Nous détaillons quelques types de motifs temporels extraits en section 3.3 et les extensions jugées utiles pour les M-traces sont listées en section 3.4. La section 3.5 s'intéresse à mettre en évidence différentes façons de permettre à l'analyste d'interagir efficacement avec un analyseur automatique en fournissant plus de critères (heuristiques) sur les motifs recherchés et d'augmenter ainsi la capacité à contrôler l'intérêt potentiel des motifs extraits. Un aperçu des techniques de fouille interactive est donné en section 3.6 en les situant par rapport à notre approche de construction interactive de connaissances. Enfin, nous effectuons, en section 3.7, la synthèse des techniques présentées pour argumenter sur les choix que nous faisons dans cette thèse (cf. chapitre 4) pour implémenter l'analyseur automatique.

3.1 Fouille de données temporelles

La fouille de données est l'étape du cycle de découverte de connaissances (cf. section 2.2.1) qui consiste à appliquer des algorithmes sur des données pour en extraire des informations qui peuvent potentiellement se révéler être des connaissances intéressantes pour les humains qui les interprètent. Le but de ces algorithmes est de découvrir des motifs, des tendances ou des relations dans les données. La *fouille de données temporelles* se concentre sur les données à caractère temporel et fournit des motifs à caractère temporel. Des données temporelles sont des données auxquelles des attributs temporels sont attachés ou parmi lesquelles un ordre temporel est établi. Faire de la *fouille de données temporelles*, c'est prendre en compte cet aspect temporel dans les méthodes de fouilles et d'expression des motifs.

Plusieurs récents états de l'art sur le domaine permettent de dresser le tableau de ce champ de recherche (Zhao & Bhowmick 2003, Masseglia et al. 2005, Laxman & Sastry 2006, Teisseire 2007). Mooney (2006) et Han & Kamber (2006) distinguent trois catégories de données pouvant faire l'objet de fouille. Les *flux de données* (*data streams*) sont des données extrêmement volumineuses et générées constamment, si bien qu'il n'est pas possible de les stocker. Han & Kamber donnent l'exemple d'un capteur fixé sur un satellite, observant l'espace ou la météo sur la Terre, et communiquant en masse des informations ordonnées temporellement et changeantes. Ce qui caractérise les méthodes d'analyse automatique et de fouille de ces données, c'est qu'une seule passe sur les données est possible à cause du volume et de l'absence de stockage. Les méthodes de préparation employées, telles que l'échantillonnage (*sampling*), l'aggrégation, le *sketching*, les fenêtres glissantes (*sliding window*, etc. (Gaber et al. 2005), visent à réduire le volume des données traitées. Les opérations de fouilles consistent ensuite classiquement à clusteriser, à classifier, compter le support de certains motifs, etc. La recherche en fouille de données temporelles s'est concentrée sur la volonté principale de gérer ce très grand volume de données. Le contexte des données temporelles comme les M-traces n'est pas aussi hostile, car les traces sont mémorisées, et les traces fouillées sont donc permanentes, ce qui nous permet, au besoin, d'y effectuer plusieurs passes.

Les *séries temporelles* représentent la deuxième catégorie de données concernée par la fouille de données temporelles. « Une base de séries temporelles est composée de séquences de valeurs numériques ou symboliques obtenues par des mesures espacées dans le temps de manière régulière » (Han & Kamber 2006). Ces séries temporelles représentent généralement des courbes de mesures échantillonnées comme le cours d'une action. Les méthodes de fouilles de ces données visent

généralement à en extraire des tendances et des similarités. Les travaux produits pour la fouille de séries temporelles sont loin des préoccupations qui nous motivent pour la détection de motifs dans les M-traces, puisque les M-traces sont symboliques, structurées et en aucun cas le résultat de mesures régulières dans le temps, mais plutôt le résultat d'observations d'événements et d'entités pouvant survenir à tout moment dans le cadre d'une activité observée.

La troisième catégorie de données régulièrement concernées par le chapeau « fouille de données temporelles » est constitué par les *bases de séquences*. Une base de séquences est composée de séquences d'événements ou éléments ordonnés, enregistrés avec ou sans notion de temps concrète. Les travaux les plus courants pour cette troisième catégorie de données relèvent de l'extraction de motifs récurrents, intéressants ou périodiques à partir de séquences d'événements. Les travaux portant sur l'extraction de motifs à partir de séquences d'événements nous semblent les plus pertinents vis-à-vis de notre problématique d'implémentation de l'analyseur automatique. En effet, les séquences d'événements sont des structures proches des M-traces, qui sont des collections d'observés temporellement situés. C'est pourquoi nous avons étudié l'extraction de motifs à partir de séquences d'événements (détaillées en section 3.2) et choisi de nous référer à ce problème pour réaliser l'analyseur automatique.

Enfin, de nombreux domaines contenant le terme *fouille* (*mining*) dans leur intitulé sont également proches de notre problématique d'analyse automatique des traces laissées par les utilisateurs d'un système : *Web Usage Mining*, *Log Mining*, etc. En particulier, le *Web Usage Mining* (Srivastava et al. 2000) se focalise sur les séquences d'url naviguées par les utilisateurs au sein d'un même site ou entre plusieurs sites pour détecter des comportement typiques, ou dangereux. Ces méthodes se fondent sur des problèmes, des techniques et des structures de motifs spécifiques à leur domaine d'application et ne peuvent pas servir de base pour l'approche générique que nous cherchons. De tels domaines utilisent des techniques fondamentales de fouilles empreintées à celles associées aux trois catégories que nous avons données dans cette section ou à la fouille de données en générale : classification, clusterisation, prédiction, etc.

3.2 Les problèmes d'extraction de motifs temporels à partir de séquences

Dans cette section, nous étudions trois approches d'extraction de motifs à partir de séquences et nous commentons pour chacun leur pertinence par rapport à notre analyseur automatique cible.

3.2.1 Fouille de base de transactions

Le premier algorithme d'extraction de motifs séquentiels fréquents à partir de séquences d'événements[20] a été proposé par Agrawal & Srikant (1995). Un an plus tard, les auteurs proposent des extensions et améliorations avec l'algorithme *GSP* (Srikant & Agrawal 1996), premier algorithme référence en matière de fouille de séquences d'événements. Son but est d'extraire des sous-séquences fréquentes dans une base de séquences.

Le tableau 3.1 donne un exemple de base de transactions. Chaque ligne correspond à un ensemble d'articles achetés par un client (identifié par $Customer\_id$). Par exemple, la deuxième ligne du tableau signifie que le client 2 a acheté les articles a et b lors d'une première visite du magasin, puis a acheté l'article c lors d'une visite ultérieure, puis a acheté les articles d, f et g lors d'une autre visite ultérieure. Informellement, une sous-séquence de cette séquence 2 est une

[20] l'intitulé anglais de ce problème est *Sequence Pattern Mining*.

Customer-id	Customer Sequence
1	$\langle(c)(i)\rangle$
2	$\langle(ab)(c)(dfg)\rangle$
3	$\langle(ceg)\rangle$
4	$\langle(c)(dg)(i)\rangle$
5	$\langle(i)\rangle$

Table 3.1 – Exemple de base de séquences (Agrawal & Srikant 1995)

autre séquence construite à partir de la séquence 2, à laquelle on a retiré des articles (les définitions formelles dont on aura besoin sont données au chapitre 4 avec la présentation de notre algorithme). Par exemple, les séquences $\langle(a)(c)(fg)\rangle$ et $\langle(ab)(f)\rangle$ sont des sous-séquences de la séquence 2, alors que les séquences $\langle(f)(a)\rangle$ et $\langle(c)(fgi)\rangle$ ne sont pas des sous-séquences. Ces sous-séquences sont les motifs séquentiels qui sont recherchés par l'algorithme *GSP* dans la base de séquences. Les auteurs définissent ensuite la notion de *support* d'une sous-séquence comme le nombre de séquences de la base qui contiennent cette sous-séquence. Par exemple, la sous-séquence $\langle(c)\rangle$ a un support de 4 dans la base du tableau 3.1, et la sous-séquence $\langle(c)(dg)\rangle$ a un support de 2 (incluse dans les séquences 2 et 4).

Le problème d'extraction de motifs séquentiels dans une base de séquences consiste, pour une fréquence seuil f_{seuil} donnée, à trouver dans la base toutes les sous-séquences dont le support est supérieur ou égal à f_{seuil}. Dans notre exemple, pour $f_{seuil} = 2$, ces sous-séquences fréquentes sont $\langle(c)(dg)\rangle$ et $\langle(c)(i)\rangle$ (et toutes leurs sous-séquences).

C'est sur ce problème d'extraction de motifs séquentiels dans une base de séquences que les principales approches de résolutions ont vu le jour. Les approches *Apriori* (en largeur d'abord) utilisent la propriété d'*antimonotonicité* du support, qui a été exploitée pour la première fois par Agrawal et al. (1993) pour l'extraction de règles d'associations fréquentes. La propriété d'antimonotonicité dit que si un motif séquentiel a un support supérieur à f_{seuil} dans une base de séquences, alors toutes les sous-séquences de ce motif ont *a fortiori* un support supérieur à f_{seuil}. Ainsi, les approches *Apriori* consistent à générer des motifs séquentiels candidats de longueur $n+1$ à partir de motifs de longueur n. Dans cette approche, les deux algorithmes les plus connus, *GSP* et *SPADE*[21] (Zaki 2001), proposent deux méthodes différentes, respectivement une base de séquences horizontale et une base de séquences verticale, la méthode *SPADE* ayant montré des temps de réponse meilleures grâce à un nombre de passes effectués sur la base très faible, mais au prix d'un encombrement mémoire un peu plus important.

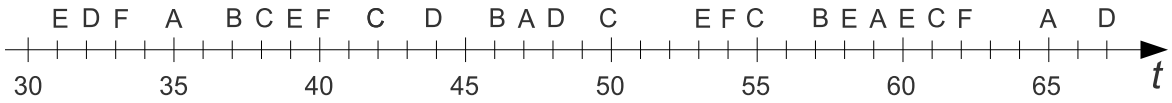
Les approches *en profondeur d'abord* comme *PSP* (Masseglia et al. 1998) et *PrefixSpan* (Pei et al. 2001) disposent généralement en interne d'une représentation en arbre de l'espace d'exploration des motifs. *PrefixSpan* accroît les préfixes des motifs fréquents, en construisant pour chaque nouveau préfixe, des bases de séquences projetées, évitant ainsi d'effectuer une passe de comptage sur la base pour chaque candidat. L'encombrement mémoire est très important, car il faut stocker en mémoire vive une base de séquences projetée pour chaque préfixe fréquent, mais les performances en temps sont compétitives avec les approches *Apriori*, surtout pour les supports faibles. De plus, l'approche par accroissement de préfixes possède des propriétés très intéressantes lorsqu'il s'agit de trouver des motifs séquentiels sous contraintes (cf. section 3.5).

[21] Une implémentation `Java` de *SPADE* est accessible avec le framework de fouille de données temporelles accessible dans le projet Open Source `Scheme Emerger` (`http://sourceforge.net/projects/schemerger/`), sans interface graphique, en complément de l'algorithme de découverte de chroniques présentées dans cette thèse.

Intérêt pour l'analyseur automatique Pour notre analyseur automatique, le problème de la découverte de motifs séquentiels dans des bases de séquences présente l'inconvénient majeur qu'une M-trace n'est pas une base de séquences, mais une séquence longue d'événements temporellement situés. De plus, le modèle de données séquentielles proposé par Agrawal & Srikant (1995) ne permet pas de prendre en compte de manière très précise l'information temporelle. En effet, la seule information temporelle qui existe dans la base de séquences est l'ordre dans lequel les articles et ensembles d'articles ont été achetés. L'algorithme *GSP* permet néanmoins de prendre en compte les dates relatives des groupes d'événements en intégrant notamment des contraintes temporelles d'écart minimum et maximum entre deux achats successifs mais ces contraintes temporelles ne sont pas totalement expressives (cf. section 3.5).

3.2.2 Découverte d'épisodes fréquents à partir de séquences d'événements

Le problème de découverte d'épisodes fréquents à partir de séquences d'événements a été introduit par (Mannila et al. 1997). Au lieu d'être représentées sous la forme d'une base de séquences, les données d'entrée sont une séquence unique d'événements. De plus, les situations temporelles de chaque événement sont maintenant représentées par un attribut *date*. Un événement est donc un couple constitué d'un type d'événement et d'un entier représentant la date.



$\langle (E,31)\ (F,33)\ (A,35)\ (B,37)\ (C,38)\ (E,39)\ (F,40)\ (C,42)\ (D,44)\ (B,46)\ (A,47)\ (D,48)$
$(C,50)\ (E,53)\ (F,54)\ (C,55)\ (B,57)\ (E,58)\ (A,59)\ (E,60)\ (C,61)\ (F,62)\ (A,65)\ (D,67)\ \rangle$

Figure 3.1 – Exemple de séquence d'événements : la séquence $\mathcal{S}_{mannila}$ (Mannila et al. 1997)

La figure 3.1 montre un exemple de séquence d'événements. La séquence se lit comme ceci : « un événement de type E apparaît à la date 31, puis un événement de type F apparaît à la date 33, etc. ». Les motifs recherchés dans une telle séquence sont des épisodes en série, en parallèle, ou hybrides (cf. figure 3.2). Informellement, un épisode en série impose que les types d'événement spécifiés dans l'épisode apparaissent dans le même ordre dans la séquence d'événements que dans l'épisode, alors que pour un épisode parallèle, les types d'événement peuvent apparaître dans n'importe quel ordre. Enfin un épisode hybride spécifie des contraintes d'ordre d'apparition partielles, mais Mannila et al. ne proposent pas de méthode pour découvrir des épisodes hybrides. Étant donnée une mesure de fréquence donnée, c'est-à-dire une fonction qui associe à chaque épisode en série ou parallèle un nombre représentant sa fréquence, et un seuil de fréquence f_{seuil}, le problème de découverte d'épisodes à partir de séquences d'événements consiste à trouver tous les épisodes en série ou en parallèle dont la fréquence est supérieure à la fréquence f_{seuil}.

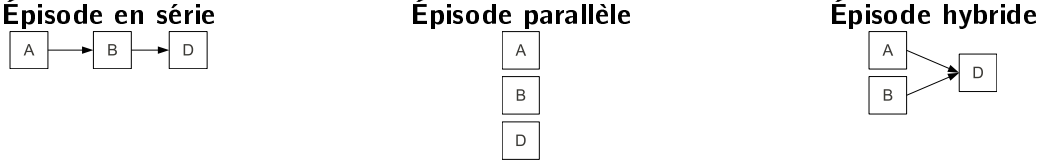


Figure 3.2 – Épisode en série, épisode parallèle et épisode hybride

Le problème de découverte d'épisodes à partir de séquences d'événements est proche du problème d'extraction de motifs séquentiels dans une base de séquences, si bien que les méthodes de résolution utilisées pour les bases de séquences peuvent parfois s'adapter à la découverte d'épisode. Méger (2004) propose une représentation sous forme de liste d'occurrences de la séquence d'événements, semblable à la représentation verticale utilisée par *SPADE*. Les algorithmes proposés par Mannila et al. (1997) utilisent le principe *Apriori* et la même méthode de génération de candidats que *GSP*. Le comptage d'un épisode candidat dans la séquence d'événements, c'est-à-dire le calcul de la fréquence d'un épisode, est un problème à part entière et non trivial et souvent non intuitif. Les mesures de fréquence utilisées peuvent être variées, chacune étant intimement liée à la structure d'épisode recherchée et à l'algorithme de comptage. La section 3.3 revient plus en détail sur certaines d'entre elles.

Intérêt pour l'analyseur automatique La représentation sous forme de séquences d'événements est très proche de la représentation en M-traces, chaque observé ponctuellement situé dans le temps pouvant facilement être assimilé à un événement dont le type est le type de l'observé et la date son extension temporelle. C'est pourquoi c'est le problème qui nous semble le plus pertinent pour bâtir un algorithme de découverte de motifs à partir de traces. Dans les sections suivantes, nous revenons plus en détails sur les travaux de recherche réalisés sur ce problème (section 3.3) et sur les différentes extensions existantes (section 3.4) qui permettent d'étendre le champ de découverte à des séquences d'événements plus structurés, comme le sont les M-traces.

3.2.3 Workflow Mining

Le problème du *Workflow Mining* consiste à découvrir des modèles de processus à partir des logs d'exécution de ces processus. La figure 3.3 illustre ce processus. La table de gauche liste les traces de cinq exécutions d'un même modèle de processus, qui est *a priori* inconnu. Le problème posé par le *Workflow Mining* est de trouver quel modèle de processus est à l'origine de ces traces. Une solution, représentée sous la forme d'un réseau de Petri, est donnée sur la partie droite de la figure, mais plusieurs solutions existent généralement pour un même ensemble de classes.

Le problème du *Workflow Mining* a été introduit par Agrawal et al. (1998). L'état de l'art de van der Aalst et al. (2003) présente une synthèse des problématiques et approches de résolution posées par le *Workflow Mining*. La motivation principale de ce domaine de recherche est d'analyser les modèles de processus en place dans les progiciels d'entreprises pour voir si les traces qu'ils génèrent pourraient par exemple révéler qu'ils sont trop élaborés pour les besoins réels. De manière générale, la comparaison du modèle de processus initial qui génère les traces avec ses versions redécouvertes sont des sources d'information très intéressantes pour la reconception et l'amélioration de ces modèles de processus. Les approches varient selon les formalismes choisis pour la représentation du modèle de processus. L'algorithme « α » (van der Aalst et al. 2004) permet par exemple de découvrir certaines classes de modèles de processus sous la forme de réseau de Petri. L'une des limitations des algorithmes de *Workflow Mining* est que plusieurs modèles de processus peuvent être équivalents pour un même jeu de traces produit. En conséquence, rien ne garantit que parmi tous les modèles équivalents, l'algorithme produira en sortie un modèle plutôt qu'un autre.

Intérêt pour l'analyseur automatique L'idée que les traces d'interactions d'une activité tracée, comme c'est le cas avec la plate-forme `CollaborativeECM`, est un idée séduisante pour la découverte de motif à partir de traces et donc pour notre analyseur automatique. Dans cette optique-là, les motifs temporels recherchés seraient des modèles de processus, par exemple des réseaux de

id trace	trace
1	*ABCD*
2	*ACBD*
3	*ABCD*
4	*ACBD*
5	*AED*

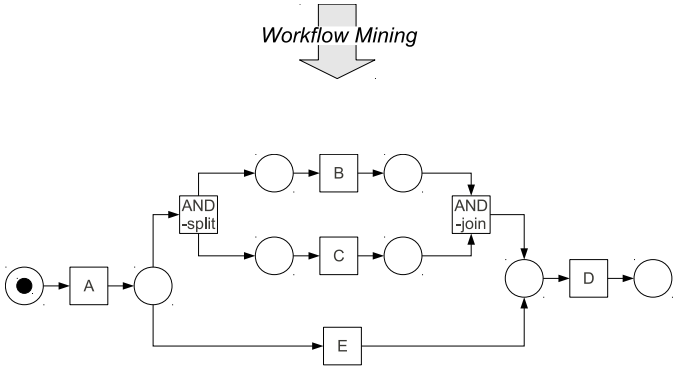


Figure 3.3 – Illustration du problème posé par le *Worflow Mining* (van der Aalst et al. 2004)

Petri. Ces modèles de processus, en tant que motifs temporels, permettraient de représenter des relations complexes entre les observés de la trace : présence facultative d'un certain type d'observé, présence disjonctive de tel type d'observé ou de tel autre type d'observé, etc. Cette piste d'utilisation des techniques de *Workflow Mining* est d'autant plus intéressante pour notre problématique d'extraction de motifs de comportement dans une activité, que nous avons la conviction que les régularités détectées dans la trace proviennent de schémas, inconnus ou pas forcément explicites, mais répétés plus ou moins consciemment par les utilisateurs. Nous pensons que les modèles de processus (tels que les réseaux de Petri) sont adaptés à la représentation des processus dans leur complexité. Cependant, l'hypothèse fondamentale faite par le domaine du *Workflow Mining* est que les traces dont on dispose proviennent toutes d'un et un seul processus. Cette hypothèse n'est pas tenable dans le cadre d'une trace d'interactions contenant potentiellement les traces de plusieurs processus inconnus à découvrir.

3.2.4 Autres problèmes

Bien d'autres travaux se sont focalisés sur la recherche de motifs temporels dans des données temporelles, mais ces travaux n'ont pas fait l'objet de branche de recherche à part entière dans la communauté, car spécifiques à un certain type de motifs ou de données, ou trop peu applicables en situation réelle.

Sun et al. (2003) proposent d'extraire à partir de séquences d'événements (avec attributs) des motifs temporels orientés événement et spécialement dédiés à la prédiction d'un événement à venir, comme le sont les règles d'épisode introduites par Mannila et al. (1997) construites à partir des épisodes fréquents. Magnusson (2000) présente le *T-Pattern* et expose une méthode pour extraire des *T-Patterns* à partir de données séquentielles. Le *T-Pattern* est expressif car il permet de représenter les types d'événement devant apparaître successivement (il s'agit donc d'épisodes en série) ainsi que les intervalles de temps devant séparer deux types d'événement successifs. Malheureusement, la méthode proposée n'est pas fondée formellement et n'est probablement pas

complète. De plus aucune indication sur ses performances n'est donnée. Les algorithmes *SM-Miner* (de Amo et al. 2004) et *MSP-Miner* (de Amo & Furtado 2007) permettent d'extraire des motifs exprimés en logique du premier ordre temporelle. Ces motifs expressifs permettent de représenter des situations complexes dans des données relationnelles temporelles, assimilables à des événements ayant plusieurs attributs. Les formules logiques inductibles à partir de la base de données ne peuvent utiliser que « $\Diamond$ » comme connecteur logique temporel ($a \wedge \Diamond b$ se lit « a, suivi de b quelque part dans le futur ») et les informations temporelles riches et quantifiées qui sont représentées par les M-traces ne peuvent pas être représentées par ces motifs. De plus, le nombre de candidats générés par ces algorithmes est très grand à cause de la puissance expressive des motifs recherchés.

3.3 Découverte à partir de séquences d'événements : mesures de fréquence et motifs

Dans cette section, nous parcourons plus en détail le problème de découverte de motifs à partir de séquences d'événements présentés en section 3.2.2, car c'est ce problème d'extraction de motifs qui est le plus proche des M-traces et des propriétés recherchées pour l'analyseur automatique.

Comme nous l'avons dit, le point critique de ces méthodes d'extraction de motifs à partir de séquences d'événements est le calcul du support, c'est-à-dire de la fréquence des motifs extraits et ce calcul dépend de la structure des motifs recherchés. Dans cette section, nous donnons quelques exemples de motifs et mesures de fréquences de la littérature. La tableau 3.2 récapitule l'ensemble des mesures de fréquences et types de motif traités dans cette section et en propose une comparaison.

3.3.1 Reconnaissance d'occurrences Winepi et Minepi

Minepi et *Winepi*[22] (Mannila et al. 1997) sont les premiers algorithmes publiés de découverte de motifs à partir de séquences d'événements. Les motifs extraits sont des épisodes parallèles et en série. *Minepi* et *Winepi* ont la même architecture générale « génération d'épisodes candidats et comptage » et utilisent la même méthode de génération d'épisodes candidats, mais diffèrent par leurs algorithmes de reconnaissance des occurrences épisodes candidats dans la séquence d'événements. L'algorithme *Minepi* reconnaît l'ensemble des occurrences minimales d'un épisode. Une occurrence d'une épisode est dite *minimale* si elle ne contient aucune autre occurrence de ce même épisode. Par exemple, les événements $(E, 58)(C, 61)(F, 62)$ constituent une occurrence de l'épisode parallèle CEF dans la séquence $\mathcal{S}_{mannila}$ de la figure 3.1, mais ne constituent pas une occurrence minimale puisqu'elle contient l'occurrence $(E, 60)(C, 61)(F, 62)$. En revanche, les occurrences $(C, 38)(E, 39)(F, 40)$ et $(E, 39)(F, 40)(C, 42)$ sont minimales.

Winepi requiert un paramètre d'entrée supplémentaire en plus de la fréquence seuil f_{seuil} : la largeur de fenêtre win. Pour reconnaître un épisode dans la séquence d'événements, *Winepi* fait glisser temporellement sur cette séquence une fenêtre de largeur win et compte le nombre de fenêtres qui contiennent cet épisode. La fréquence *Winepi* d'un épisode α dans une séquence $\mathcal{S}$ pour une largeur de fenêtre win se définit comme ceci :

[22] Des implémentations `Java` de *Minepi* et *Winepi* ont été développés avec le framework de fouille de données temporelles accessible dans le projet Open Source `Scheme Emerger` (`http://sourceforge.net/projects/schemerger/`), sans interface graphique, en complément de l'algorithme de découverte de chroniques présentées dans cette thèse.

Motif et mesure de fréquences	Occurrences	Fréquence
Occurrences *Minepi*	$(E,31)(F,33)(C,38)$, $(C,38)(E,39)(F,40)$, $(F,33)(C,38)(E,39)$, $(E,39)(F,40)(C,42)$, $(C,50)(E,53)(F,54)$, $(E,53)(F,54)(C,55)$, $(F,54)(C,55)(E,58)$, $(E,60)(C,61)(F,62)$	8
Occurrences *Winepi* ($win = 5$)	$(C,38)(E,39)(F,40)$, $(E,39)(F,40)(C,42)$, $(C,50)(E,53)(F,54)$, $(E,53)(F,54)(C,55)$, $(F,54)(C,55)(E,58)$, $(E,58)(C,61)(F,62)$, $(E,60)(C,61)(F,62)$	$\frac{15}{40} = 0,375$*
Occurrences sans recouvrement	$(E,31)(F,33)(C,38)$, $(E,39)(F,40)(C,42)$, $(C,50)(E,53)(F,54)$, $(C,55)(E,58)(F,62)$	4
Épisode sans borne (tus** $= 2$) occurrences distinctes	$(C,38)(E,39)(F,40)$, $(E,39)(F,40)(C,42)$, $(E,53)(F,54)(C,55)$, $(E,60)(C,61)(F,62)$	$\frac{7}{40} = 0,175$
Occurrences distinctes *au plus tôt*	$(E,31)(F,33)(C,38)$, $(E,39)(F,40)(C,42)$, $(C,50)(E,53)(F,54)$, $(C,55)(E,60)(F,62)$	4

* sachant que pour une fenêtre $[t, t')$, une occurrence peut toucher la borne inférieure t mais pas la borne supérieure t', les 15 fenêtres de largeur $win = 5$ contenant l'épisode parallèle CEF sont $[29, 34)$, $[30, 35)$, $[31, 36)$, $[36, 41)$, $[37, 42)$, $[38, 43)$, $[39, 44)$, $[50, 55)$, $[51, 56)$,$[52, 57)$,$[53, 58)$,$[54, 59)$, $[58, 63)$, $[59, 64)$ et $[60, 65)$.

** voir section 3.3.3.

Table 3.2 – Comparaison des fréquences de l'épisode parallèle CEF dans la séquence d'événements $\mathcal{S}_{mannila}$ en fonction de la mesure de fréquence choisie

$$f_{Winepi}(\alpha, \mathcal{S}, win) = \frac{\text{Nombre de fenêtre contenant } \alpha}{\text{Nombre total de fenêtres}}$$

où $T_S - T_E + win - 1$ est le nombre total de fenêtre de largeur win, T_S est la date de début de la séquence et T_E la date de fin ($T_S = 31$ et $T_E = 67$ pour la séquence $\mathcal{S}_{mannila}$).

La mesure f_{Winepi} renvoie donc pour chaque épisode candidat α un nombre entre 0 et 1, alors que *Minepi* renvoie un nombre entier représentant le nombre exact d'occurrences minimales dans la séquence (cf. tableau 3.2). L'ordre de complexité de la reconnaissance des occurrences d'un épisode par *Minepi* ou par *Winepi* est : $O(I \times N)$, où I est la taille de l'épisode et N le nombre d'événements de la séquence.

3.3.2 Reconnaissance d'occurrences sans recouvrement

Laxman et al. (2007) proposent une mesure d'épisodes en série et en parallèle que est basée sur le non recouvrement des occurrences. Les 8 occurrences *Minepi* du tableau 3.2 comportent des recouvrements temporels entre elles. Par exemple, $(E,31)(F,33)(C,38)$ et $(C,38)(E,39)(F,40)$ se recouvrent à la date 38, $(C,50)(E,53)(F,54)$ et $(E,53)(F,54)(C,55)$ se recouvrent sur l'intervalle temporel $[53, 54]$, etc. Laxman et al. proposent un algorithme de reconnaissance du nombre maximal d'occurrences d'un épisode dans une séquence d'événements en évitant tout recouvrement

entre les occurrences. L'algorithme ne reconnaît que des *occurrences distinctes*, c'est-à-dire des occurrences qui ne partagent jamais un même événement avec une autre occurrence. Par exemple, les occurrences $(E, 31)(F, 33)(C, 38)$ et $(C, 38)(E, 39)(F, 40)$ ne sont pas distinctes car elles partagent l'événement $(C, 38)$. Le fait que l'algorithme ne reconnaisse que des occurrences distinctes est une conséquence de la restriction de non recouvrement. Cette propriété d'occurrences nécessairement distinctes permet de parcourir efficacement la séquence d'événements en empêchant de construire des occurrences partielles empruntant des événements à une occurrence en cours de reconnaissance. Ainsi, il n'existe jamais plus d'une occurrence en cours de reconnaissance lors du parcours de la trace et cela se traduit par des performances en temps et en mémoire supérieures à *Minepi* dans la pratique, car le facteur I disparaît, bien que toujours linéaire en $O(N)$, où N est la taille de la séquence.

Pour l'épisode parallèle CEF, les occurrences trouvées par cet algorithme dans la séquence $\mathcal{S}_{mannila}$ (cf. tableau 3.2) montrent que cet algorithme produit une sorte de découpage de la séquence, dans lequel chaque occurrence trouvée représente un morceau de ce découpage.

3.3.3 Reconnaissance d'occurrences d'un épisode sans borne

Casas-Garriga (2003) introduisent une structure d'épisode en série et en parallèle un peu plus élaborée que celle de Mannila et al. : les épisodes sans borne (*Unbounded episode*). Ces épisodes sans borne ajoutent une contrainte d'écart maximum entre deux événements d'une même occurrence. Cet écart maximal est appelé *tus* (*Time Unit Separation*).

Par exemple, pour une valeur de *tus* donnée, un épisode sans borne en série ABC contraint le type d'événement B à apparaître au plus *tus* unités de temps après A, et D à apparaître au plus *tus* unités de temps après B. En revanche, il n'y a pas de fenêtre englobante win comme avec *Winepi*, d'où le « sans-borne ». Pour un épisode sans borne en parallèle ABC, les types d'événement A, B et C peuvent apparaître dans n'importe quel ordre mais doivent toujours respecter l'écart maximal *tus* entre deux événements successifs de l'occurrence. Le calcul de la fréquence d'un épisode sans borne s'effectue de la même manière que pour *Winepi*, mais avec une largeur de fenêtre égale à $(n-1) \times tus$, où n est la longueur de l'épisode. Ainsi la mesure de fréquence prend en compte la longueur de l'épisode pour la taille de la fenêtre coulissante, ce qui n'était pas le cas avec *Winepi* et empêchait de reconnaître des épisodes longs dans une fenêtre de taille fixe win. La méthode de découverte est la même que pour *Minepi* et *Winepi*, c'est-à-dire *Apriori* avec la même méthode de génération.

La correction et la complétude de l'algorithme de découverte ne sont pas étudiés par Casas-Garriga. Méger (2004) montre que l'algorithme de génération associée à cette mesure de fréquence n'est pas complet et propose une approche complète de découverte d'épisodes avec contrainte d'écart maximum, basée sur une exploration en profondeur avec un comptage par liste d'occurrences semblable à *SPADE*.

3.3.4 Reconnaissance de chroniques « au plus tôt »

Les *chroniques* sont des épisodes parallèles auxquels on a ajouté des contraintes temporelles numériques. De manière formelle, les chroniques sont des réseaux de contraintes temporelles (Dechter et al. 1991). La figure 3.4 montre l'apport des chroniques par rapport aux épisodes parallèles. Sur cette figure, la chronique se lit comme ceci : « le type d'événement E doit apparaître entre 1 et 3 unités de temps après le type d'événement C, et le type d'événement F doit apparaître entre 2 et 4 unités de temps après B ». Par propagation des contraintes temporelles sur la chronique (Dechter

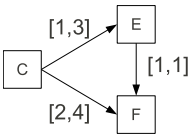
et al. 1991), cela implique que le type d'événement F doit apparaître entre 1 et 1 unités de temps après E (c'est-à-dire exactement 1 unité de temps après E).

Épisode parallèle
(chronique sans contraintes temporelles)

Occurrences distinctes *au plus tôt*
dans $\mathcal{S}_{mannila}$:

$(E, 31)(F, 33)(C, 38)$
$(E, 39)(F, 40)(C, 42)$
$(C, 50)(E, 53)(F, 54)$
$(C, 55)(E, 60)(F, 62)$

Chronique



Occurrences distinctes *au plus tôt*
dans $\mathcal{S}_{mannila}$:

$(C, 38)(E, 39)(F, 40)$
$(C, 50)(E, 53)(F, 54)$

Figure 3.4 – Comparaison entre épisode parallèle et chronique

Pour reconnaître les occurrences d'une chronique dans une séquence d'événements, Dousson et al. (1993) introduisent l'algorithme CRS (*Chronicle Recognition System*), optimisé par Dousson et al. (2007).

Pour une chronique sans contraintes temporelles, c'est-à-dire pour un épisode parallèle, cet algorithme reconnaît des occurrences distinctes (c'est-à-dire sans partage d'événements entre les occurrences) et de façon dite « *au plus tôt* » (Duong 2001). La reconnaissance *au plus tôt* stipule que parmi toutes les occurrences possibles pour un épisode parallèle, CRS choisit les occurrences distinctes par « ordre chronologique d'apparition ». Le tableau 3.2 donne la liste des occurrences *au plus tôt* de l'épisode parallèle CEF en comparaison avec les autres mesures de fréquence. Pour une chronique avec contraintes temporelles, l'algorithme sélectionne de la même manière, mais en veillant à respecter les contraintes. Nous expliquerons plus en détails et sur d'autres exemples les effets de cette mesure de fréquence dans le chapitre suivant (cf. section 4.1.3). À des fins d'illustration, les occurrences distinctes *au plus tôt* de l'épisode parallèle CEF sont reportées dans le tableau comparatif 3.2.

Un algorithme de découverte automatique de chroniques à partir de séquence d'événements a été proposé (Dousson & Duong 1999, Duong 2001), mais cet algorithme n'est pas complet.

3.4 Extensions pertinentes pour les M-traces

Il existe de nombreuses extensions au problème de la découverte de motifs séquentiels à partir de séquences d'événements. Ces extensions permettent de fouiller des séquences d'événements plus élaborés, c'est-à-dire des séquences dont les événements ne sont plus seulement des couples $(type, date)$. Ces extensions sont intéressantes à considérer dans le cadre de la fouille de M-traces, car nous avons vu que les observés d'une M-trace sont des données structurées, comportant plusieurs attributs, et relations et pouvant persister dans le temps. Dans les sous-sections suivantes, nous parcourons quelques extensions du problème de découverte de motifs à partir de séquences d'événements pour discuter leur applicabilité à la structure « M-trace ».

3.4.1 Fouille de séquences avec taxonomie

Une M-trace est une collection d'observés typés. Pour cerner le besoin d'une fouille de séquences avec taxonomie, prenons une M-trace telle que chaque observé est ponctuellement situé dans le temps, telle qu'aucun observé n'a d'attribut ni de relation avec d'autres observés. On pourrait alors facilement représenter cette M-trace par une séquence d'événements, en remplaçant chaque observé de la M-trace par un événement (a, t) où a est le type de l'observé et t une conversion sous forme d'entier de la date de l'observé. Cependant, il resterait encore une partie importante des informations comprises dans la M-trace et non représentée dans la séquence d'événements fouillée. Cette information, c'est le modèle de trace de la M-trace. Le modèle de trace établit une hiérarchie entre les types d'observés. Ainsi, la situation dans laquelle on se trouve avec une telle M-trace est celle de la figure 3.5, c'est-à-dire que la M-trace est alors une séquence d'événements dont les types sont les types d'observé à laquelle est associée la taxonomie des types de la séquence, où chaque relation est de type « is_a ».

$$\langle (c,1)(d,2)(f,4)(c,7)(g,8)(e,9) \rangle$$

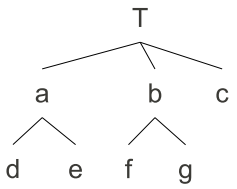


Figure 3.5 – Séquence d'événements avec taxonomie

Les méthodes « classiques » d'extraction de motifs à partir de séquences d'événements n'incluent pas de taxonomie pour les types d'événement. Avec un support minimal de 2 et une mesure de fréquence distincte *au plus tôt*, il n'y a dans la séquence de la figure 3.5 aucun épisode de taille 2 qui soit fréquent. En prenant en compte la taxonomie, l'épisode en série ca possède deux occurrences : $(c,1)(a,2)$ et $(c,7)(e,9)$. Prendre en compte les type d'événement à un niveau hiérarchique plus élevé peut être important pour la fouille de M-traces, car il n'existe pas toujours de corrélations temporelles dans la trace entre les types « feuille » de la taxonomie.

L'algorithme *GSP* (Srikant & Agrawal 1996) propose de prendre en compte les taxonomies dans la fouille de base de séquences en reprenant le principe présenté par Srikant & Agrawal (1995) dans le cadre de l'extraction de règle d'association. Pour des séquences d'événements, ce principe reviendrait à ajouter tous les sur-types de chaque événement à la même date ce que donnerait pour notre exemple la séquence :

$$\langle (c,1)(d,2)(a,2)(f,4)(b,4)(c,7)(g,8)(b,8)(e,9)(b,9) \rangle.$$

On comprend à la vue de cette séquence modifiée que la prise en compte des hiérarchies est un compromis à faire avec les performances de fouille puisque la séquence à fouiller est plus longue et le nombre de types d'événement est plus grand, ce qui augmente aussi le nombre de candidats.

L'algorithme *HYPE* (Plantevit et al. 2006) propose également une solution pour la prise en compte de taxonomies dans le cadre des bases de séquences multidimensionnelles (voir section suivante) et donc *a fortiori* pour des séquences d'événements classiques, facilement adaptable à des séquences d'événements.

Customer_id	Customer_group	City	Age_group	sequence
10	business	Boston	middle	$\langle (bd)\, cba \rangle$
20	professional	Chicago	young	$\langle (bf)\, (ce)\, (fg) \rangle$
30	business	Chicago	middle	$\langle (ah)\, abf \rangle$
40	education	New-York	retired	$\langle (be)\, (ce) \rangle$

Table 3.3 – Base de données séquentielle multidimensionnelle (Pinto et al. 2001)

3.4.2 Fouille de séquences multidimensionnelles

Le besoin qui est à l'origine des travaux sur la fouille de séquences multidimensionnelles est le même que celui de l'analyseur automatique. Une des limitations du problème de découverte de motifs à partir de séquences d'événements est que chaque événement de la séquence ne représente qu'un seul aspect de l'événement du monde réel qu'il représente. En effet, avec le formalisme des séquences d'événements, il est impossible de représenter un événement ayant plusieurs dimensions. Un événement ayant plusieurs dimensions, ne seraient plus de la forme (a, t), où a est le type d'événement et t sa date, mais de la forme $(a_1, \ldots, a_p, t)$, où les a_i sont des valeurs nominales représentant les différentes dimensions de l'événement. Nous avons dit que les observés présents dans les M-traces possèdent souvent plusieurs attributs en plus d'une date. Pour représenter plus fidèlement un observé de M-trace par un événement et pour pouvoir extraire des corrélations entre ces attributs en plus des corrélations temporelles, il faudrait trouver un moyen de représenter tous ces attributs et une procédure de fouille adaptée à cette nouvelle structure.

Par exemple, l'*Observé* 3 de la figure 2.1 pourrait être représenté de façon multidimensionnelle ainsi : $(CONNEXION, http, 811)$, où $CONNEXION$ serait la valeur de l'événement pour la dimension 1, $http$ la valeur pour la dimension 2, et 811 sa date (car à 13h31, c'est la 811^{e} minute de la journée).

On trouve dans la littérature scientifique des travaux qui vont dans le sens de la fouille de séquences multidimensionnelles, mais les représentations multidimensionnelles adoptées ne conviennent pas parfaitement. Le premier travail connu sur cette question est celui de Pinto et al. (2001), dans lequel les auteurs proposent une adaptation de l'algorithme *PrefixSpan* pour l'extraction de motifs séquentiels multidimensionnels à partir d'une base de séquences multidimensionnelle comme celle de la figure 3.3. Le problème d'une telle représentation est double : la représentation en base de séquences ne correspond pas aux M-traces et les attributs multidimensionnels sont attachés aux séquences et non pas aux événements.

Contrairement à Pinto et al., Wojciechowski (2001) propose une représentation multidimensionnelle des événements sous de la forme $(a_1, \ldots, a_p, t)$ (cf. tableau 3.4), c'est-à-dire comme celle que nous avons présentée ci-dessus, et propose un algorithme de découverte de type *Apriori* pour la découverte de motifs dans cette séquence. Wojciechowski ne parle pas de séquence multidimensionnelle mais de séquence d'*événements complexes*. Les épisodes multidimensionnels découverts sont de la forme $(m10, t27, .)(., t54, 2)$, s'interprétant comme suit : « une alarme de type $t27$ sur le module $m10$ et de n'importe quelle sévérité, suivie d'une alarme de type $t54$ sur n'importe quel module et de sévérité 2 ».

L'inconvénient avec cette représentation multidimensionnelle de Wojciechowski est que tous les événements doivent avoir la même structure, e.g. avoir trois attributs, dont le premier a pour domaine $\{m10, m20, m30, m40\}$, le deuxième $\{t27, t54, t30\}$ et le troisième $\{1, 2, 3\}$. Cela n'est pas le cas pour les M-traces car les observés peuvent être des instances de classes différentes et

Time	Module name (M)	Notification type (N)	Severity (S)
105	m10	t27	1
108	m20	t54	2
112	m10	t27	3
113	m20	t30	3
119	m30	t54	2
126	m40	t54	3

$\langle (m10, t27, 1, 105)\ (m20, t54, 2, 108)\ (m10, t27, 3, 112)\ (m20, t30, 3, 113)\ (m30, t54, 2, 119)\ (m40, t54, 3, 126)\rangle$

Table 3.4 – Exemple de séquence d'évènements complexes (Wojciechowski 2001)

avoir des attributs avec des sémantiques et des domaines différents.

Yu & Chen (2005) proposent d'extraire des motifs séquentiels de taille n dans lesquels chaque élément est lui-même un motif séquentiel de taille $n-1$, et ainsi de suite récursivement. Mais comme le dit Teisseire (2007), ces séquences ne sont pas vraiment multidimensionnelles dans la mesure où les différentes dimensions entretiennent un lien hiérarchique très strict (un jour comporte des sessions qui sont elle-mêmes composés de pages visités). Il est difficile de voir comment tirer profit de ce motif pour des collections d'objets aux structures aussi diverses que les observés de M-traces.

Les algorithmes M^2SP et *HYPE* (Plantevit et al. 2006) permettent de découvrir des motifs réellement multidimensionnels, au sens des événements complexes de Wojciechowski, mais avec la possibilité pour *HYPE* de trouver des motifs dont les valeurs pour chaque dimension appartiennent à n'importe quel niveau d'une taxonomie (cf. section 3.4.1). M^2SP et *HYPE* présentent néanmoins les mêmes inconvénients que les événements complexes de Wojciechowski : le nombre de dimensions est le même pour chaque événement et l'interprétation de chaque dimension est la même pour tous les événements. De plus, ils s'appliquent également à des bases de séquences et non à des séquences d'événements, mais devraient pouvoir s'adapter à des séquences d'événements assez simplement.

3.4.3 Événements persistants

Un autre point critique dans la fouille des M-traces par rapport aux problèmes de fouille de séquences existants est la prise en compte de la persistance des événements. Les observés d'une M-trace possèdent chacun une extension temporelle qui est soit une date ponctuelle, auquel cas la représentation classique en séquence d'événements ne pose pas de problème, soit un intervalle de temps. Si les types de corrélation temporelle entre deux événements ponctuels a et b sont simples (a avant b, a en même temps que b, et a après b), ceux entre deux intervalles de temps sont plus élaborés et plus nombreux. Allen (1983) en dénombre treize.

Certaines propositions ont été réalisées pour étendre le problème de la fouille de données temporelles à des événements qui peuvent persister dans le temps, c'est-à-dire de la forme (a, t_d, t_f) au lieu de la forme ponctuelle (a, t). t_d est la date de début de l'événement et t_f est la date de fin. Chen & yi Wu (2006) proposent de se ramener au problème connu des événements ponctuels en découpant tout événement persistant (a, t_d, t_f) en deux événements ponctuels (a^d, t_d) et (a^f, t_f). L'algorithme *T-Apriori* proposé par les auteurs recherche dans cette séquence d'événements ponctuels des motifs appelés *séquences temporelles* très proches des épisodes en série. Le problème avec cette approche est que rien ne lie dans la séquence d'événements ponctuels les événements

de type a^d et les événements de type a^f. Pour l'algorithme d'extraction utilisé, ce sont deux types complètement différents au même niveau que a^d, b^d et c^f par exemple. Du coup, rien ne garantit que les motifs extraits décriront des corrélations entre le début et la fin d'un même type d'événement. De plus, il peut y avoir des erreurs dans l'interprétation de ces motifs par l'humain, car si un motif extrait décrit une relation entre un a^d et un a^f il se peut que le a^f ne soit pas la fin du même événement persistant de type a dont le a^d est le début.

Laxman & Sastry (2006) proposent de se ramener à une séquence d'événements ponctuels de type multidimensionnel. Pour ce faire, avant le processus d'extraction à proprement parler, les auteurs proposent de définir une table à deux entrées référençant des intervalles de durée pour les événements persistants. Par exemple, avec la table 3.5 l'événement persistant $(a, 56, 63)$ sera transformé en l'événement ponctuel multidimensionnel $(a, 2, 56)$, où a est son type, 56 sa date de début et 2 la clé représentant son intervalle de durée. Les durées précises des événements persistants sont ainsi partitionnées par la table des intervalles de durée et regroupées en paquets, ce qui limite la possibilité de trouver des corrélations temporelles fines sur les durées des événements.

Clé	Intervalle de durée
1	$[0, 3]$
2	$[4, 8]$
3	$[9, 15]$
...	...

Table 3.5 – Exemple de table d'intervalles de durée

L'algorithme *IEMiner* (Patel et al. 2008) est le premier algorithme complet d'extraction de motifs temporels à partir d'une base de séquences d'événements persistants. *IEMiner* ne cherche pas à se ramener au problème plus simple des événements ponctuels et ne perd aucune information et peut extraire de manière complète des motifs décrivant les treize types de relations de l'algèbre de Allen. La mesure de fréquence utilisée est celle de la plupart des algorithmes de fouille dans une base de séquences, c'est-à-dire le nombre de séquences de la base satisfaisant le pattern. La question de trouver une mesure de fréquence pour le même motif, mais dans une seule et longue séquence d'événements persistants, reste entière. C'est pourquoi il n'est pas direct d'appliquer *IEMiner* à la découverte complète de motifs temporels avec persistance dans des séquences d'événements persistants comme le sont les M-traces.

3.4.4 Autres domaines de la fouille de données en rapport avec les M-traces

D'autres domaines de la fouille de données peuvent apporter des solutions à la recherche de motifs dans les M-traces. L'un d'entre eux, la *fouille de graphe* (Yan & Han 2002), consiste à extraire des sous-graphes fréquents d'une base de graphes. Jusqu'alors, dans notre état de l'art nous nous sommes concentrés sur l'aspect temporel des M-traces et n'avons rien proposé pour extraire des motifs en prenant en compte les relations qui existent entre les observés. Dans le contexte des M-traces, les techniques de fouille de graphes, où chaque observé est un noeud et chaque arc une relation entre deux observés pourraient s'appliquer. Cependant, nous n'avons pas trouvé de travaux qui utilisent conjointement les techniques de fouille de séquences et de fouille de graphes.

Le domaine de la fouille de données multirelationnelles (Džeroski 2003) est probablement le sous-domaine qui s'intéresse aux données les plus structurées dans la communauté de la fouille de données, et surtout les plus proches de la structure de M-trace en dehors des aspects temporels. Les données multirelationnelles sont les données présentes dans les bases de données relationnelles.

Dans l'algèbre relationnelle, chaque table représente une relation *n-aire*, où *n* est le nombre de colonnes de la table. Le formalisme des M-traces de Settouti et al. (2009) ne supporte pour l'heure que des relations binaires, il faudrait donc représenter chaque type de relation d'une M-trace par un table à deux colonnes. Chaque relation représente un prédicat de la logique du premier ordre et le but de la fouille de données multirelationnelles est d'extraire des motifs recherchés sous la forme de formules de la logique du premier ordre. Le problème de ces méthodes est le même que celui rencontré par de Amo et al. (2004) sur l'extraction de formules de la logique temporelle du premier ordre : l'espace de recherche est très large et le nombre de candidats générés est très grand. Ces techniques passent mal l'épreuve de la mise à l'échelle alors qu'une M-trace peut contenir de très nombreux types et instances de relations. Les logiques temporelles du premier ordre constituent le seul moyen d'utiliser les techniques multirelationnelles conjointement avec des techniques d'extraction de corrélations temporelles dans les traces et comme nous l'avons dit, les techniques d'extraction existantes pour les logiques temporelles n'ont qu'une très faible expressivité temporelle (cf. section 3.2.4).

Une autre piste, plus récente cette fois, concernant la prise en compte des nombreuses relations que contiennent les M-traces est le domaine de la *fouille de réseaux* (*Network Mining*). Ce domaine est né de la volonté d'extraire des relations d'intérêt dans les graphes très volumineux et complexes tels que *Facebook*, mais le problème de découverte n'est pas non plus attaqué sous l'angle de la temporalité, angle prioritaire pour nous.

3.5 Découverte de motifs sous contraintes

Une requête de fouille de données comporte toujours au moins une contrainte sur les motifs à trouver dans un jeu de données. Par défaut, une contrainte est toujours utilisée : le *support minimal*. La fouille consiste alors à trouver tous les motifs qui satisfont le support minimal. De nombreuses autres contraintes que le support minimal peuvent cependant être spécifiées par l'utilisateur de l'algorithme. Ces contraintes peuvent être utilisées pour réduire l'espace d'exploration des motifs candidats et ainsi améliorer les temps de réponse de l'algorithme. Ce qui nous intéresse dans cette thèse, c'est qu'elles peuvent surtout être vues comme un moyen pour l'analyste d'effectuer une guidage fort de l'algorithme vers les motifs d'intérêt. Dans cette section, nous dressons un rapide état de l'art des résultats trouvés par la communauté de fouille de données temporelles sur le sujet.

L'algorithme *GSP*, pionnier de la fouille de données séquentielles, proposait déjà d'inclure des contraintes temporelles d'écart minimal et d'écart maximal entre deux événements successifs et de fenêtre temporelle englobante. Pei et al. (2002) ont été les premiers à théoriser sur l'utilisation des contraintes en fouille de données séquentielles et distinguent sept catégories de contraintes pouvant se simplifier en cinq :

- les *contraintes d'intervalle de temps*, comme celles supportées par *GSP* (Srikant & Agrawal 1996) et *GTC* (Masseglia et al. 2009) permettent de restreindre l'étalement temporel des motifs,
- les *contraintes de longueur* (longueur minimale ou maximale des motifs recherchés),
- les *contraintes d'inclusion (ou de non-inclusion) dans un motif* permettent à l'analyste de spécifier un motif dans lequel tous les motifs recherchés sont (ou ne sont pas) inclus. Si la longueur du motif défini par l'analyste est 1, alors cette contrainte revient à imposer la présence (ou l'absence) d'un type d'événement dans les motifs recherchés,

- les *contraintes d'aggrégat* permettent à l'analyste de spécifier des propriétés aggrégées requises sur les motifs extraits (e.g. un écart temporel moyen maximal ou minimal entre les événements),

- les *contraintes d'expressions régulières* permettent à l'analyste de spécifier de manière simple et précise, grâce à un langage d'expressions régulières, des contraintes structurelles résumées sur les types d'événement apparaissant dans les motifs recherchés et sur leur ordre d'apparition. Pour prendre en compte ce type de contraintes, les algorithmes comme *SPIRIT Garofalakis1999* tirent profit de la dualité entre expressions régulières et automates. Les motifs représentés en logique temporelle du premier ordre se prêtent particulièrement bien à la prise en compte de contraintes de ce type (de Amo & Furtado 2007) pour réduire le très grand nombre de candidats générés dans ces approches.

Généralement, les algorithmes de fouille utilisent la relation d'ordre partielle d'inclusion « $\subseteq$ » entre motifs pour parcourir l'espace des motifs candidats des motifs les plus courts. Les contraintes qui peuvent être prises en compte de manière très efficace par les algorithmes de fouilles sont les contraintes qui vérifient la propriété d'antimonotonicité : « si $motif_1 \subseteq motif_2$ et $motif_2$ vérifie la contrainte C, alors $motif_1$ vérifie la contrainte C ». C'est cette propriété d'antimonotonicité qui est vérifiée par la contrainte de support minimale utilisée dans tout algorithme de fouille. C'est aussi le cas des contraintes de longueur maximale, et d'inclusion dans un motif. Une manière naïve mais très efficace de prendre en compte ces contraintes antimonotones dans les approches de type « générer et compter » consiste à parcourir l'ensemble des motifs candidats générés et de supprimer les motifs ne vérifiant pas ces contraintes avant de compter les autres. Cela économise des comptages de motifs dans les données, opérations qui sont généralement les plus coûteuses lors de la fouille pour des grandes volumes de données. Une manière plus efficace de procéder consiste à tenir compte de ces contraintes dès la phase de génération de sorte à ne jamais générer les motifs qui ne satisferaient pas les contraintes antimonotones.

Une contrainte est dite *préfixe-antimonotone* si pour tout motif séquentiel α vérifiant cette contrainte tous les préfixes de α la vérifient aussi. Les contraintes d'expressions régulières étant *préfixe-antimonotones*, Pei et al. (2007) montrent que les méthodes de fouille séquentielle par accroissement de préfixe comme *PrefixSpan* peuvent prendre en compte les contraintes d'expressions régulière très efficacement (sans générer de motifs candidats non acceptables), en étant capables de prendre en compte *a fortiori* les contraintes qui sont seulement *antimonotones*, car *préfixe-antimonotone* implique *antimonotone*.

3.6 Fouille incrémentale et interactive de séquences

Parmi les travaux que nous avons parcourus sur l'extraction de motifs à partir de données temporelles, certains traitent des aspects *incrémental* et *interactif* de la fouille, aspects qui semblent être particulièrement intéressants dans le cadre d'un processus itératif et interactif de construction de connaissances tel que celui présenté au chapitre 2. Le problème de l'*extraction incrémentale* de séquences consiste en réalité à optimiser les requêtes d'extraction successives de même support lorsque la base de séquences évolue. Au lieu d'effectuer pour une même requête, des passes sur la base de séquences modifiée entière, les techniques d'extraction incrémentale visent à être capable de répondre à la requête sur la base modifiée à partir de l'écart (le *delta*) entre les deux bases et des résultats obtenus à partir en fouillant la première base, celle qui n'est pas encore modifiée. Des techniques ont été proposées, optimisant toujours plus l'espace mémoire requis pour sauvegarder les potentiels futurs fréquents (Parthasarathy et al. 1999, Masseglia et al. 2003, Lin & Lee 2004),

ne gardant que les connaissances les plus utiles lors des phases d'extraction précédentes. Le problème d'extraction *interactive* de séquences est proche du problème de fouille incrémentale et très lié à ce dernier. Il consiste à être capable de résoudre efficacement une requête d'extraction proche d'une requête précédente d'extraction sur un base de séquences qui n'a pas changé entre les deux requêtes, à partir de la première requête et de ses résultats (Wojciechowski 2001). La fouille interactive est très liée à la fouille incrémentale (Lin & Lee 2004), les techniques proposées pouvant parfois gérer à la fois les *delta* entre base de séquences et entre requêtes.

Les travaux sur la fouille interactive de séquences, bien que définis généralement sur des bases de séquences, peuvent être adaptés à une séquence longue et unique d'événements. Ces travaux sont très pertinents dans le cadre d'un analyseur automatique prenant part au processus interactif d'abstraction. Malgré cela, nous considérons que ces travaux relèvent de l'optimisation d'un problème plus basique d'extraction de motifs à partir de traces qui reste à définir. C'est pourquoi nous ne nous y sommes pas intéressés dans cette thèse mais nous les considérons comme primordiaux pour l'amélioration de l'analyseur automatique présenté au chapitre suivant.

3.7 Bilan et choix d'une approche pour notre analyseur automatique

Parmi les techniques d'extraction de motifs à partir de données temporelles, nous avons vu que celles proposées dans le cadre des séquences d'événements sont les plus appropriées au problème de l'extraction de motifs à partir de traces pour notre analyseur automatique, car une M-trace est plus proche d'une séquence d'événements que d'une base de séquences ou que d'un jeu de traces tel qu'analysé par les algorithmes de *Workflow Mining*. Les motifs généralement recherchés dans les séquences d'événements, c'est-à-dire les épisodes en série et les épisodes parallèles, ont une expressivité temporelle relativement simple. Les types d'événement présents dans les motifs peuvent apparaître soit dans une ordre stricte total (épisode en série), soit dans n'importe quel ordre (épisode parallèle). Certains types d'épisodes sont plus expressifs sur les critères de temps à respecter par leurs occurrences : épisodes avec contraintes d'écart en événements, ou avec fenêtre englobante, mais les critères de temps sont généralement les mêmes pour tous les événements de tous les motifs recherchés (on spécifie un écart tus ou une fenêtre win valable pour tous les motifs).

Nous avons choisi la chronique comme structure de motif recherché dans les traces, et donc comme représentation des signatures de tâche introduites au chapitre 2 (section 2.1.4). Nous avons fait ce choix, car les chroniques peuvent exprimer des contraintes temporelles précises sur les événements des occurrences. Cette précision est *a priori* nécessaire pour distinguer deux situations de l'activité mettant en jeu les mêmes événements, mais dans des ordres différents. Nous verrons que ce besoin est justifié, comme l'illustre dans le cas de l'activité de conduite automobile (annexe B). La possibilité de quantifier les bornes des contraintes temporelles nous a paru également intéressante pour pouvoir distinguer deux réalisations d'une même tâche dans une activité à des vitesses différentes. Ces différences de vitesse dans la réalisation des tâches peuvent témoigner d'un niveau d'habitude plus ou moins grand selon l'individu qui la réalise, ce qui peut entrer en considération dans un éventuel module d'assistance qui prenne en compte le niveau de l'individu. L'avantage des chroniques en tant que motif temporel est qu'elles sont pratiques pour effectuer de la reconnaissance fine de situations, aussi bien dans les alarmes d'un réseau de télécommunication (Cordier & Dousson 2000), que dans les comportements des clients (Guillou et al. 2008) et que dans l'activité d'un humain dans un environnement instrumenté (Cram et al. 2009).

Il existe un algorithme de découverte de chroniques à partir de séquences d'événements (Dousson & Duong 1999, Duong 2001), mais cet algorithme n'est pas complet. Cela pose problème pour

notre analyseur automatique, car il pourrait ne pas découvrir certaines chroniques potentiellement intéressantes pour l'analyste selon des critères de sélection arbitraires (ceux qui impliquent la non-complétude de l'algorithme).

Dans le chapitre 4, nous présentons un algorithme complet de découverte de chroniques à partir de traces, où les traces sont des séquences d'événements telles que présentées en section 3.2.2. Dans le choix d'un algorithme pour notre analyseur automatique, nous avons donc privilégié le support des aspects temporels de la M-trace (à l'exception de la persistance des observés), car ces aspects temporels sont primordiaux dans l'observation d'une activité et manquaient fortement dans la représentation des signatures de tâche dans l'équipe SILEX jusqu'à présent. Les aspects structurels de la M-trace (attributs, relations, et hiérarchie des types d'observé) ne sont pas supportés par cet algorithme mais nous avons vu qu'il existe un certain nombre d'extensions possibles pour les prendre en compte à partir d'un algorithme résolvant le problème de base. Sans de telles extensions, c'est lors de la phase de préparation des données, préalable à la phase d'extraction des motifs (cf. section 2.2.1.1), que les informations importantes portées par les M-traces sont traduites dans le format des séquences d'événements accepté en entrée de l'analyseur automatique. Nous détaillerons cette phase de préparation des M-traces dans le chapitre 5.

Chapitre 4

Découverte complète de chroniques à partir de traces

Sommaire

Dans ce chapitre, nous présentons l'algorithme CDA, un algorithme complet de découverte de chroniques à partir de traces qui a été conçu dans le but de répondre aux besoins de notre processus de découverte interactive et itérative de motifs temporels présentés au chapitre 2. Ce chapitre propose une formalisation des concepts relatifs à la découverte de chroniques. Tout au long de ce chapitre nous positionnerons fortement nos définitions et notre approche par rapport au travail de Duong (Dousson & Duong 1999, Duong 2001), qui a proposé une méthode de découverte non complète de chroniques à partir de journaux d'alarmes. L'algorithme CDA que nous proposons traite le problème de la découverte de chroniques de manière complète et dans un cadre plus général que ce qui a été proposé par Duong puisqu'il est possible de choisir les paramètres d'entrée de CDA de telle sorte qu'il découvre exactement et avec des performances comparables les mêmes chroniques que celles retournées par la méthode de Duong, mais il est aussi possible et très simple d'utiliser CDA pour implémenter la découverte d'*épisode hybrides* (Mannila et al. 1997) pour lequel peu de solutions ont été proposées à notre connaissance, ou encore la découverte « réellement » complète de chroniques à partir de traces. Après avoir défini les concepts nécessaires à la présentation de CDA (cf. section 4.1), nous présenterons CDA (cf. section 4.2) et ses propriétés : terminaison, complétude et complexité. Nous expliquerons ensuite comment construire la base de contraintes donnée en entrée de notre algorithme CDA de façon à implémenter les trois cas de découvertes : Duong, *épisode hybrides* et découverte complète (cf. section 4.3). Nous discuterons dans chacun de ces trois cas le problème de la complexité en temps du problème abordé. Dans la section 4.4 nous détaillerons les leviers accessibles à l'utilisateur de l'algorithme qui lui permettront de guider autant que possible le processus de découverte vers les chroniques qui l'intéressent le plus. Enfin, la section 4.5 présente les mesures et les observations que nous avons faites sur notre algorithme CDA afin d'une part de justifier certains choix de conception et d'autre part d'estimer le comportement de CDA dans les situations réelles et ainsi de mieux l'utiliser.

4.1 Définitions

4.1.1 Trace

Définition 1 (Trace)
Une trace *est une séquence d'événements, notée* $\mathcal{S} = \langle(\varepsilon_1, t_1) \dots (\varepsilon_l, t_l)\rangle$. *Chaque paire* (ε_i, t_i) *est appelée* événement, *où* ε_i *est le type d'événement et* t_i *est un entier appelé la date d'événement.*

On note $\mathbb{E}$ l'ensemble de tous les types d'événement qui apparaissent dans la trace $\mathcal{S}$. De plus, on suppose que $\mathbb{E}$ est un ensemble totalement ordonné par une relation d'ordre que l'on notera $\leq_{\mathbb{E}}$. Imposer un ordre total sur les types d'événement est nécessaire pour définir de manière formelle les notions d'épisode et de base de contraintes dans la suite. Cet ordre total peut être choisi de manière complètement arbitraire. Par exemple, on peut choisir l'ordre lexicographique sur les noms de types d'événement.

Exemple
La trace $\mathcal{S}_0$ *de la figure 4.11 de la page 107 (la page 107 est détachable pour faciliter la lecture) est constituée de six événements :*

$$\mathcal{S}_0 = \langle(A, 1)(C, 2)(B, 4)(A, 5)(C, 5)(B, 6)\rangle$$

Pour la trace $\mathcal{S}_0$, *l'ensemble des types d'événement est* $\mathbb{E} = \{A, B, C\}$

Nous appelons *trace* dans ce chapitre ce que Duong appelle *journal log*, et plus généralement ce que l'ensemble de la communauté de fouille de séquences appelle *event sequence*, c'est-à-dire une *séquence d'événements* (cf. section 3.2.2). Cela signifie que l'algorithme que nous présentons dans la suite s'applique à tout jeu de données pouvant être modélisé sous la forme d'une telle séquence d'événements, pas seulement aux traces d'interactions. Réciproquement, tout algorithme proposé dans la littérature pour l'analyse et la fouille de séquences d'événements peut s'appliquer aux traces modélisées ainsi.

4.1.2 Chronique

Une chronique est un motif temporel qui se définit à partir des notions d'épisode et de contrainte temporelle.

Définition 2 (Épisode)
Étant donné un ensemble $\mathbb{E}$ *de types d'événement, un épisode est un n-uplet* $(\varepsilon_1, \ldots, \varepsilon_n)$ *d'éléments de* $\mathbb{E}$*, noté plus simplement* $\varepsilon_1 \ldots \varepsilon_n$*, et tel que* $\forall i, \varepsilon_i \leq_{\mathbb{E}} \varepsilon_{i+1}$.

Un épisode peut être vu comme un mot ordonné sur $\mathbb{E}$. La longueur de l'épisode est définie par le nombre de types d'événement qu'il contient, c'est-à-dire n. Imposer que les types d'événement qui composent l'épisode soient ordonnés assure que pour un épisode donné il n'existe qu'une seule écriture possible.

La littérature sur la fouille de séquences différencie deux types d'épisodes : les épisodes en série et les épisodes parallèles (cf. chapitre 3). La notion d'épisode telle que nous la définissons ici est en fait ce que Mannila et al. (1997) et les autres travaux sur la découverte d'épisodes appellent plus précisément *épisode parallèle*. N'utilisant que cette notion d'épisode parallèle pour les chroniques, nous y référerons dans la suite par le simple terme « épisode ».

Définition 3 (Contrainte temporelle)
Une contrainte temporelle *est un quadruplet* $(\varepsilon, \varepsilon', i^-, i^+)$*, plus souvent noté* $\varepsilon[i^-, i^+]\varepsilon'$*, où* $(\varepsilon, \varepsilon') \in \mathbb{E}^2$*, tel que* i^- *et* i^+ *sont chacun à valeur entière ou à valeur dans l'ensemble* $\{-\infty, \infty\}$*, appelés respectivement* borne inférieure *et* borne supérieure*, tels que* $i^- \leq i^+$.

Les contraintes temporelles sont des motifs temporels élémentaires pouvant s'interpréter comme ceci : « $\varepsilon[i^-, i^+]\varepsilon'$ contraint le type d'événement ε' à apparaître dans un intervalle de temps $[i^-, i^+]$ après ε ». Ainsi, on dira que deux événements (ε, t) et (ε', t') *satisfont la contrainte temporelle* $\varepsilon[i^-, i^+]\varepsilon'$ si et seulement si $t' - t \in [i^-, i^+]$. Une contrainte temporelle du type $\varepsilon[a, \infty]\varepsilon'$ s'interprétera ainsi : « le type d'événement ε' doit apparaître après plus de a unités de temps après le type d'événement ε.

Définition 4 (Chronique)
Étant donné un ensemble $\mathbb{E}$ *de type d'événement, une* chronique *est un couple* $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ *tel que :*

1. $\mathcal{E} = \varepsilon_1 \ldots \varepsilon_n$ *est un épisode composé d'éléments de* $\mathbb{E}$,
2. $\mathcal{T} = \{\tau_{ij}\}_{1 \leq i < j \leq n}$ *est un ensemble de contraintes temporelles sur* $\mathcal{E}$*. On note pour tout* i, j *tel que* $i < j$*,* $\tau_{ij} = \varepsilon_i[\tau_{ij}^-, \tau_{ij}^+]\varepsilon_j$.

On dira que l'épisode $\mathcal{E}$ est l'*épisode* de la chronique $\mathcal{C}$ ou que la chronique $\mathcal{C}$ est *basée sur* l'épisode $\mathcal{E}$. L'élément $\mathcal{E}$ est un épisode au sens où nous l'avons défini, et il convient donc que son écriture $\varepsilon_1 \ldots \varepsilon_n$ respecte l'ordre « $\leq_{\mathbb{E}}$ » existant sur les types. La longueur de la chronique

est définie par la longueur de son épisode $\mathcal{E}$, c'est-à-dire par le nombre d'éléments de l'épisode : $|\mathcal{C}| = |\mathcal{E}| = n$. On peut voir les chroniques telles que définies ci-dessus comme des épisodes parallèles accompagnés d'informations supplémentaires : les contraintes temporelles τ_{ij}.

Exemple
Sur la figure 4.11, La chronique $\mathcal{C}_1$ est de longueur 3 et la chronique $\mathcal{C}_2$ est de longueur 2. ABC est l'épisode de $\mathcal{C}_1$, et $A[1,3]B$, $A[0,1]C$, et $B[-2,-1]C$ sont les trois contraintes temporelles de $\mathcal{C}_1$. La contrainte temporelle $A[1,3]B$ n'est satisfaite que si B apparaît dans la trace $\mathcal{S}_0$ entre 1 et 3 unités de temps après A.

Cette définition des chroniques suscite deux remarques. Premièrement, chaque contrainte temporelle peut être vue comme une chronique de longueur deux. Deuxièmement, il existe une contrainte temporelle τ_{ij} définie pour chaque couple de types d'événement de la chronique $\mathcal{C}$. Dit autrement, les chroniques sont nécessairement *pleines*. Il y a donc dans chaque chronique de longueur n exactement C_n^2 (c'est-à-dire $n \times (n-1)/2$) contraintes temporelles définies. Cela peut être ennuyeux car cela oblige à définir une contrainte temporelle pour chaque couple de types d'événement de la chronique, même si on ne souhaite pas poser de contrainte temporelle pour certains couples. Dans notre définition des chroniques, ne pas poser de contrainte temporelle entre les types d'événement ε_i et ε_j c'est définir la contrainte temporelle $\tau_{ij} = [-\infty, \infty]$. De cette manière, on obtient une équivalence naturelle entre une chronique telle que définie ci-dessus (c'est-à-dire *pleine*) et sa version simplifiée dans laquelle toutes les contraintes temporelles du type $\varepsilon_i[-\infty, \infty]\varepsilon_j$ sont masquées (cf. figure 4.1). Dans la suite, nous supposerons que toutes les chroniques sont pleines, même si elles sont représentées ou écrites dans leurs versions simplifiées.

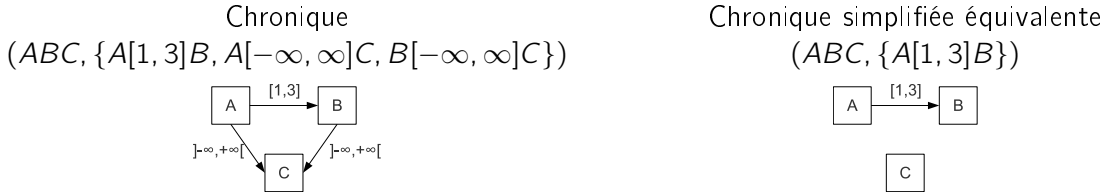


Figure 4.1 – Un exemple de chronique et sa version simplifiée équivalente

Les chroniques sont des motifs qui s'instancient dans la trace en *occurrences*.

Définition 5 (Occurrence)
Étant données une trace $\mathcal{S}$ et une chronique $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ avec $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$, une occurrence *de $\mathcal{C}$ dans la trace $\mathcal{S}$ est un ensemble $(\varepsilon_1, t_1) \dots (\varepsilon_n, t_n)$ de la trace $\mathcal{S}$ qui satisfait les contraintes temporelles de $\mathcal{C}$, c'est-à-dire tel que $\forall i < j, t_j - t_i \in [\tau_{ij}^-, \tau_{ij}^+]$.*

La notation formelle « $(\varepsilon_1, t_1) \dots (\varepsilon_n, t_n)$ » ne représente pas la liste temporellement ordonnée des occurrences des types d'événement ε_i de l'épisode de la chronique $\mathcal{C}$, puisque $\varepsilon_i \leq_{\mathbb{E}} \varepsilon_{i+1}$. Autrement dit, l'occurrence notée ainsi ne vérifie pas $t_i \leq t_{i+1}$. Sur tous les exemples concrets que nous verrons, nous noterons les événements des occurrences dans leur ordre temporel d'apparition dans la trace, ce qui est plus naturel.

Exemple
Soient un ensemble des types d'événement $\mathbb{E} = \{A, B, C\}$ et une trace $\mathcal{S}_0$ sur $\mathbb{E}$ telles que $\mathcal{S}_0 = \langle (A,1)(C,2)(B,4)(A,5)(C,5)(B,6) \rangle$. Les occurrences de la chronique $\mathcal{C}_1 = (ABC, \{A[1,3]B,$ $A[0,1]C,$ $B[-2,-1]C\})$ sont $\langle (A,1)(C,2)(B,4) \rangle$ et $\langle (A,5)(C,5)(B,6) \rangle$ (cf. première partie de

la figure 4.11). ABC est l'épisode de $\mathcal{C}_1$, et $A[1,3]B$, $A[0,1]C$, et $B[-2,-1]C$ sont les trois contraintes temporelles de $\mathcal{C}_1$. La contrainte temporelle $A[1,3]B$ n'est satisfaite que si B apparaît dans la trace entre 1 et 3 unités de temps après A.

4.1.3 Fréquence d'une chronique, la mesure f_{CRS}

Comme dans toute approche de découverte de type « générer et compter », notre approche de découverte nécessite un moyen de quantifier la fréquence d'une chronique. De façon générale, la *fréquence* d'une chronique dans une trace se définit comme le nombre d'occurrences de cette chronique dans la trace. La question de trouver le nombre d'occurrences d'une chronique dans une trace renvoie à la question de définir l'ensemble des occurrences d'une chronique $\mathcal{C}$ dans une trace $\mathcal{S}$.

Cette question ne va pas de soi. En ce qui concerne les épisodes seuls, sans contraintes temporelles, il existe dans la littérature sur la fouille de séquences autant d'algorithmes de découverte que de définitions de la fréquence d'un épisode (cf. section 3.3). Ce même problème subsiste avec les chroniques.

Par exemple, combien y a-t-il d'occurrences de la chronique $\mathcal{C}_2$ dans la trace $\mathcal{S}_0$ sur la figure 4.11 ? $\mathcal{C}_2$ étant composée des deux types d'événement A et C, une façon naïve de répondre à cette question est dans un premier temps de combiner toutes les occurrences de A dans la trace $\mathcal{S}_0$ avec toutes les occurrences de C, et dans un deuxième temps de ne garder que les occurrence ainsi construites qui satisfont la contrainte temporelle $A[0,4]C$. Cela donne sans doute un nombre total d'occurrences de $\mathcal{C}_2$ dans la trace $\mathcal{S}_0$ qui est largement supérieur au nombre total d'occurrence reconnues par un observateur humain. Un humain répondrait probablement qu'il y a 2 occurrences de la chronique $\mathcal{C}_2$ dans la trace $\mathcal{S}_0$.

Nous notons f_{all} la mesure de fréquence qui reconnaît toutes les occurrences d'une chronique dans une trace par la méthode naïve décrite ci-dessus. Cette mesure de fréquence sera utilisée dans la section 4.3 à l'occasion de la construction des bases de contraintes temporelles.

Exemple
Il y a au total quatre occurrences de la chronique $\mathcal{C} = (AB, \{A[-1,4]B\})$ dans la trace $\mathcal{S} = \langle (A,1)(B,3)(A,4)(B,5) \rangle$: $(A,1)(B,3)$, $(A,1)(B,5)$, $(A,4)(B,3)$ et $(A,4)(B,5)$, donc $f_{all}(\mathcal{C}) = 4$. Cependant, dans le contexte de l'analyse de l'activité à partir de traces, un humain en verrait probablement seulement 2 : $(A,1)(B,3)$, $(A,4)(B,5)$.

L'algorithme *Chronicle Recognition System* (CRS) proposé par Dousson et al. (Dousson et al. 1993, Dousson et al. 2007) reconnaît un ensemble de chroniques qui est proche du nombre de chroniques reconnues spontanément par un être humain. Nous ne rentrons pas ici dans les détails de l'algorithme CRS, mais nous retiendrons qu'il reconnaît les occurrences d'une chronique de telle sorte qu'il n'y ait jamais deux occurrences partageant un même événement. D'autres mesures de fréquence existent pour compter des épisodes, qui sont basées sur d'autres principes que celui des occurrences disjointes (cf. section 3.3), mais il n'est pas évident de les réutiliser telles quelles avec des chroniques. De plus, il a été prouvé par Dousson et al. (1993) que CRS possède deux propriétés indispensables à la découverte : sa complexité est linéaire en fonction de la longueur de la trace et il est *monotone* pour la relation d'ordre sur les chroniques « $\preceq$ » que nous définissons dans la section 4.1.4. Une mesure de fréquence f est dite *monotone* pour une relation $\preceq$ si et seulement si $\mathcal{C} \preceq \mathcal{C}'$implique $f(\mathcal{C}) \leq f(\mathcal{C}')$. Cette propriété de monotonicité est fondamentale pour une approche de type « générer et compter » comme la nôtre. Nous noterons la mesure de fréquence CRS f_{CRS} dans la suite.

Exemple
Dans la trace $\mathcal{S}_0 = \langle (A,1)(C,2)(B,4)(A,5)(C,5)(B,6) \rangle$*, la chronique* $\mathcal{C} = (AB, A[-1,3]B)$ *possède au total trois occurrences dans la trace* $\mathcal{S}_0$ *:* $\mathcal{O}_1 = (A,1)(B,4)$*,* $\mathcal{O}_2 = (B,4)(A,5)$*, et* $\mathcal{O}_3 = (A,5)(B,6)$*. Cependant, les occurrences* $\mathcal{O}_1$ *et* $\mathcal{O}_2$ *ont un événement en commun* $(B,4)$*, et les occurrences* $\mathcal{O}_2$ *et* $\mathcal{O}_3$ *ont un événement en commun* $(A,5)$*. Dans la trace* $\mathcal{S}_0$*, CRS parcourt les événements dans leur ordre chronologique. Dès que l'occurrence* $\mathcal{O}_1$ *sera reconnue à la date 4, CRS empêchera l'événement* $(B,4)$ *d'apparaître dans d'autres occurrences. Ainsi, l'occurrence* $\mathcal{O}_2$ *ne sera pas reconnue par* CRS*, alors que l'occurrence* $\mathcal{O}_3$ *le sera. En conclusion, on a* $f_{CRS}(\mathcal{C}) = 2$ *dans la trace* $\mathcal{S}_0$*.*

4.1.4 Base de contraintes temporelles

La notion de « base de contraintes temporelles » a été introduite par Dousson & Duong (1999), mais les bases de contraintes telles que nous les définissons ici en diffèrent sensiblement. On peut cependant définir de manière générale et partagée une base de contraintes temporelles comme un conteneur de contraintes temporelles qui sont utilisées pour la génération de chroniques candidates lors du processus de découverte.

Dans notre définition, la base de contraintes stocke pour chaque couple de types d'événement plusieurs contraintes temporelles dans une structure appelée *graphe de contraintes*, alors que la base de contraintes de Dousson & Duong ne peut stocker qu'une seule contrainte temporelle par paire. Il n'y a donc pas dans leur approche de notion de graphe de contraintes, ni de notion de $\mathcal{D}$-chronique (voir ci-après). Dans un graphe de contraintes, les contraintes temporelles sont les noeuds d'un graphe acyclique orienté dont les arcs représentent des relations d'inclusion de contraintes. On définit de manière naturelle l'inclusion de contraintes temporelles comme suit : la contrainte temporelle $\varepsilon_1[i^-, i^+]\varepsilon_2$ est incluse dans la contrainte temporelle $\varepsilon_1[i^{-\prime}, i^{+\prime}]\varepsilon_2$ si et seulement l'intervalle $[i^-, i^+]$ est inclus dans l'intervalle $[i^{-\prime}, i^{+\prime}]$. Généraliser cette notion d'inclusion aux chroniques de longueurs supérieures à 2 est, nous le verrons dans la sous-section suivante, beaucoup moins naturel. C'est pourquoi nous préférons appeler cette relation d'inclusion sur les contraintes temporelles « relation de descendance », qui se généralise facilement aux chroniques (cf. sous-section suivante). Cette notion de « descendance » provient du treillis des chroniques (cf. figure 4.13, page 107) que l'on peut construire à partir de cette relation généralisée aux chroniques.

4.1.4.1 Descendance/Ascendance de chroniques

Définition 6 (Descendance de contraintes temporelles)
La contrainte temporelle $\varepsilon_1[i^-, i^+]\varepsilon_2$ *est une descendante (resp. ascendante) de la contrainte temporelle* $\varepsilon_1[i^{-\prime}, i^{+\prime}]\varepsilon_2$*, notée* $\varepsilon_1[i^-, i^+]\varepsilon_2 \preceq \varepsilon_1[i^{-\prime}, i^{+\prime}]\varepsilon_2$ *(resp.* $\varepsilon_1[i^-, i^+]\varepsilon_2 \succeq \varepsilon_1[i^{-\prime}, i^{+\prime}]\varepsilon_2$*), si et seulement si* $[i^-, i^+] \subseteq [i^{-\prime}, i^{+\prime}]$ *(resp.* $[i^-, i^+] \supseteq [i^{-\prime}, i^{+\prime}]$*).*

Une implication implicite de la définition de descendance ci-dessus est qu'il est impossible de comparer par la relation de descendance deux contraintes temporelles définies pour deux couples de types d'événement différents. Pour le reste, la relation $\preceq$ hérite des propriétés de la relation d'inclusion sur un ensemble d'intervalles et constitue donc une relation d'ordre partielle sur l'ensemble des contraintes temporelles.

Exemple
La contrainte temporelle $A[1,2]B$ *est descendante de la contrainte temporelle* $A[0,4]B$ *(noté* $A[1,2]B \preceq A[0,4]B$*) puisque* $[1,2] \subseteq [0,4]$*. De même :*

1. $A[2,2]A \preceq A[2,3]A$
2. $A[3,5]C \preceq A[3,5]C$ *(égalité)*
3. $A[0,3]C$ *et* $A[2,4]C$ *ne sont pas comparables par* $\preceq$ *car* $[0,3]$ *et* $[2,4]$ *ne sont pas comparables pour l'inclusion* $\subseteq$.

On ne peut pas non plus comparer les contraintes temporelles $A[2,4]B$ *et* $A[2,5]C$ *car les couples de types d'événement* (A,B) *et* (A,C) *sont différents.*

La définition de descendance sur les contraintes temporelles se généralise aux chroniques de longueurs supérieures à deux par la définition ci-dessous. De manière informelle, cette définition signifie qu'une chronique $\mathcal{C}$ est une descendante d'une chronique $\mathcal{C}'$ si l'épisode de $\mathcal{C}'$ est inclus dans celui de $\mathcal{C}$ (c'est-à-dire si $\mathcal{E}'$ est un sous-épisode de $\mathcal{E}$) et si chaque contrainte de $\mathcal{C}$ est une descendante de sa contrainte correspondante dans $\mathcal{C}'$. La relation de descendance $\preceq$ définit une relation d'ordre partiel sur l'ensemble des chroniques. On définit l'ascendance entre chroniques comme la relation d'ordre opposée de la descendance.

Définition 7 (Descendance de chroniques)
La chronique $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ *est une descendante de la chronique* $\mathcal{C}' = (\mathcal{E}', \mathcal{T}')$, *noté* $\mathcal{C} \preceq \mathcal{C}'$, *si et seulement si :*

1. $\mathcal{E}' \subseteq \mathcal{E}$, *c'est-à-dire en notant* $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$, *il existe une fonction* f *strictement croissante sur les indices* $1 \dots n'$ *de* $\mathcal{E}'$ *telle que* $\mathcal{E}' = \varepsilon_{f(1)} \dots \varepsilon_{f(n')}$,
2. $\forall i, j$ *tel que* $1 \leq i < j \leq n'$, $[\tau^-_{f(i)f(j)}, \tau^+_{f(i)f(j)}] \subseteq [\tau^{-\prime}_{ij}, \tau^{+\prime}_{ij}]$

Exemple
Sur la figure 4.11, on a $\mathcal{C}_1 \preceq \mathcal{C}_2$ *parce que l'épisode* AC *est un sous-épisode de* ABC *et que* $A[1,1]C \subseteq A[0,4]C$. *Dans ce cas, avec* $\mathcal{E} = ABC$ ($\varepsilon_1 = A$, $\varepsilon_2 = B$, $\varepsilon_3 = C$) *et* $\mathcal{E}' = AC$ ($= \varepsilon_1\varepsilon_3$), $n' = 2$, *la fonction* f *de croissance sur les indices se définit par* $f(1) = 1$ *and* $f(2) = 3$, $[\tau^{-\prime}_{12}, \tau^{+\prime}_{12}] = [1,1]$, *et* $[\tau^-_{f(1)f(2)}, \tau^+_{f(1)f(2)}] = [0,4]$. *Dans cet exemple, on a plus précisément,* $\mathcal{C}_1 \prec \mathcal{C}_2$ *(se lit «* $\mathcal{C}_1$ *est un* descendant strict *de* $\mathcal{C}_2$ *») puisque* $\mathcal{C}_1 \neq \mathcal{C}_2$.

Sur la figure 4.13, on a :

1. $\mathcal{C}_7 \prec \mathcal{C}_4 \prec \mathcal{C}_3$
2. $\mathcal{C}_6 \prec \mathcal{C}_4 \prec \mathcal{C}_3$
3. $\mathcal{C}_5 \prec \mathcal{C}_3$
4. $\mathcal{C}_8 \prec \mathcal{C}_3$

En revanche, les chroniques $\mathcal{C}_7$ *et* $\mathcal{C}_6$ *ne sont pas comparables car les contraintes temporelles* $A[0,1]C$ *et* $A[-3,1]C$ *ne sont pas comparables. De même* $\mathcal{C}_4$, $\mathcal{C}_5$ *et* $\mathcal{C}_8$ *ne sont pas comparables.*

Une autre manière de voir la relation de descendance entre chroniques $\preceq$ est de dire que si $\mathcal{C} \preceq \mathcal{C}'$, alors la chronique $\mathcal{C}$ est « plus stricte », ou « plus contrainte » que la chronique $\mathcal{C}'$.

La relation de descendance $\preceq$ étant un ordre partiel sur l'ensemble des chroniques, on dira qu'une chronique $\mathcal{C}$ est *minimale* (resp. *maximale*) dans un ensemble de chroniques F si et seulement si il n'existe pas de descendant strict (resp. ascendant strict) de $\mathcal{C}$ dans l'ensemble F. Formellement, $\mathcal{C}$ est minimale (resp. maximale) dans F si et seulement si $\forall \mathcal{C}' \in F$, $\mathcal{C}' \prec \mathcal{C}$ est faux (resp. $\mathcal{C}' \succ \mathcal{C}$ est faux). Pour tout ensemble de chroniques F, on note $F^\top$ l'ensemble des chroniques maximales de F.

Exemple
Comme les contraintes temporelles peuvent être considérées comme des chroniques de longueur 2, on lit sur la figure 4.12 (voir page 107) que $A[-1,1]B$, $A[1,3]B$ *et* $A[3,5]B$ *sont minimales dans le graphe* $\mathcal{G}_{AB}$ *et que* $A[-1,5]B$ *est la seule chronique maximale du graphe* $\mathcal{G}_{AB}$*, c'est-à-dire* $\mathcal{G}^{\top}_{AB} = \{A[-1,5]B\}$*. De même* $\mathcal{G}^{\top}_{AC} = \{A[-3,4]C\}$ *et* $\mathcal{G}^{\top}_{BC} = \{B[-4,1]C\}$.

4.1.4.2 Graphe de contraintes temporelles

Les arcs d'un graphe de contraintes temporelles représentent des relations de type « parent/enfant », que nous formalisons ci-dessous.

Définition 8 (`is_child_of`/`is_parent_of`)
Soit F *un ensemble de contraintes temporelles, on dit qu'une contrainte temporelle* τ est un parent de τ' dans F*, noté* τ `is_parent_of` τ'*, si et seulement si :*

1. $\tau' \prec \tau$,
2. *il n'existe pas* τ'' *dans* F *telle que* $\tau' \prec \tau'' \prec \tau$.

On définit la relation `is_child_of` *comme la relation inverse de* `is_parent_of` *:* τ' `is_child_of` τ *si et seulement si* τ `is_parent_of` τ'.

La définition de « τ `is_parent_of` τ' » s'interprète comme ceci : la contrainte temporelle τ est un parent de τ' dans F si τ est un ascendant strict de τ' et qu'il n'existe pas d'autre contrainte temporelle dans F qui « s'intercale » entre τ et τ'.

Exemple
Dans le graphe de contraintes temporelles $\mathcal{G}_{AB}$ *il y a un arc orienté de la contrainte temporelle* $A[-1,3]B$ *vers* $A[-1,1]B$ *parce que* $A[-1,1]B$ `is_child_of` $A[-1,3]B$ *(*$[-1,1] \subset [-1,3]$*), et un arc orienté de* $A[-1,5]B$ *vers* $A[-1,3]B$ *parce que* $A[-1,3]B$ `is_child_of` $A[-1,5]B$ *(*$[-1,3] \subset [-1,5]$*), mais pas d'arc de* $A[-1,5]B$ *vers* $A[-1,1]B$ *parce que* $A[-1,1]B$ *n'est pas un enfant de* $A[-1,5]B$ *dans* $\mathcal{G}_{AB}$ *(il y a* $A[-1,3]B$ *entre les deux).*

Enfin, les graphes de contraintes temporelles se définissent à partir de la relation `is_parent_of`.

Définition 9 (Graphe de contraintes temporelles)
Un graphe de contraintes temporelles est un couple $(\mathcal{T}, \mathcal{R})$ *vérifiant :*

1. $\mathcal{T} = \{\tau_1 \dots \tau_p\}$ *est un ensemble de contraintes temporelles,*
2. $\mathcal{R} = \{(\tau_i^d, \tau_i^a); (\tau_i^d \in \mathcal{T}) \wedge (\tau_i^a \in \mathcal{T}) \wedge (\tau_i^d$ `is_parent_of` $\tau_i^a)\}$.

Un graphe de contraintes temporelles est donc un ensemble de contraintes temporelles muni de toutes ses relations de type `is_parent_of`.

Exemple
Sur la figure 4.12, le graphe $\mathcal{G}_{AB}$ *se définit par :*

1. $\mathcal{T} = \{A[-1,5]B, A[-1,3]B, A[1,5]B, A[-1,1]B, A[1,3]B, A[3,5]B\}$

2.

$$\begin{aligned}\mathcal{R} = \{&A[-1,5]B \texttt{ is\_parent\_of } A[-1,3]B,\\ &A[-1,5]B \texttt{ is\_parent\_of } A[1,5]B,\\ &A[-1,3]B \texttt{ is\_parent\_of } A[-1,1]B,\\ &A[-1,3]B \texttt{ is\_parent\_of } A[1,3]B,\\ &A[1,5]B \texttt{ is\_parent\_of } A[1,3]B,\\ &A[1,5]B \texttt{ is\_parent\_of } A[3,5]B\}.\end{aligned}$$

4.1.4.3 Base de contraintes temporelles

Nous pouvons maintenant définir formellement la notion de base de contraintes temporelles.

Définition 10 (Base de contraintes temporelles)
Soit $\mathbb{E}$ l'ensemble des types d'événement. Une base de contraintes temporelles $\mathcal{D}$ est un ensemble de triplets $(\varepsilon, \varepsilon', \mathcal{G}_{\varepsilon\varepsilon'})$ tels que :

1. *pour tout triplet $(\varepsilon, \varepsilon', \mathcal{G}_{\varepsilon\varepsilon'})$ de $\mathcal{D}$, ε et ε' sont des types d'événement de $\mathbb{E}$ et $\mathcal{G}_{\varepsilon\varepsilon'}$ est un graphe de contraintes temporelles sur le couple $(\varepsilon, \varepsilon')$ (c'est-à-dire des contraintes temporelles du type $\varepsilon[i^-, i^+]\varepsilon'$),*
2. *il n'existe pas deux triplets $(\varepsilon_1, \varepsilon'_1, \mathcal{G}_{\varepsilon_1\varepsilon'_1})$ et $(\varepsilon_2, \varepsilon'_2, \mathcal{G}_{\varepsilon_2\varepsilon'_2})$ dans $\mathcal{D}$ tels que $(\varepsilon_1, \varepsilon'_1) = (\varepsilon_2, \varepsilon'_2)$,*
3. *pour tout triplet $(\varepsilon, \varepsilon', \mathcal{G}_{\varepsilon\varepsilon'})$ de $\mathcal{D}$, $\varepsilon \preceq \varepsilon'$.*

Une base de contraintes temporelles est donc constituée de graphes de contraintes sur des couples de types d'événement. La deuxième condition assure qu'il n'y aura pas deux graphes de contraintes définis dans $\mathcal{D}$ pour le même couple $(\varepsilon, \varepsilon')$ de types d'événement. La troisième condition « $\varepsilon \preceq \varepsilon'$ » parachève la volonté de ne pas définir dans une même base de contraintes deux graphes pour les deux même types d'événement en assurant qu'il n'y aura pas un graphe défini pour le couple $(\varepsilon, \varepsilon')$ et un graphe pour le couple $(\varepsilon', \varepsilon)$. En revanche, il est tout à fait possible de définir dans une base de contraintes un graphe pour un couple du type $(\varepsilon, \varepsilon)$ où les deux types d'événement sont identiques, même si l'exemple de la figure 4.12 ne le montre pas.

Exemple
La base de contraintes temporelles $\mathcal{D}_0$ de la figure 4.12 contient trois triplets : $(A, B, \mathcal{G}_{AB})$, $(A, C, \mathcal{G}_{AC})$ et $(B, C, \mathcal{G}_{BC})$.

Nous étendons la notation $F^\top$ définie pour un ensemble F de contraintes temporelles à une base de contraintes $\mathcal{D}$ de telle sorte que $\mathcal{D}^\top$ dénote l'union des contraintes temporelles maximales de tous les graphes de contraintes de $\mathcal{D}$.

$$\mathcal{D}^\top = \bigcup_{\mathcal{G} \in \mathcal{D}} \mathcal{G}^\top$$

Exemple
Sur la figure 4.12 :

$$\mathcal{D}_0^\top = \{A[-1,5]B, A[-3,4]C, B[-4,1]C\}$$

4.1.4.4 Chroniques construites à partir d'une base de contraintes temporelles

Si le concept de base de contraintes temporelles a été introduit, c'est parce que les chroniques générées par le processus de découverte seront construites à partir de cette base de contraintes. De telles chroniques construites à partir d'une base de contraintes $\mathcal{D}$ sont appelées des $\mathcal{D}$-chroniques.

Définition 11 ($\mathcal{D}$-chronique)
Étant donnée une base de contraintes temporelles $\mathcal{D}$, $(\mathcal{E}, \mathcal{T})$ est une $\mathcal{D}$-chronique *si et seulement si, en notant $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$ et $\mathcal{T} = (\tau_{ij})_{1 \leq i < j \leq n}$, pour toute contrainte temporelle τ_{ij} de $\mathcal{T}$, il existe un triplet $(\varepsilon_i \varepsilon_j, \mathcal{G}_{ij})$ dans $\mathcal{D}$ tel que $\tau_{ij} \in \mathcal{G}_{ij}$.*

Exemple
Étant donnée la base de contraintes temporelles $\mathcal{D}_0$ de la figure 4.12, la chronique $\mathcal{C}_2$ de la figure 4.11 est une $\mathcal{D}_0$-chronique puisque $A[0,4]C \in \mathcal{G}_{AC}$. En revanche, $\mathcal{C}_1$ n'est pas une $\mathcal{D}_0$-chronique car $A[1,3]B \notin \mathcal{G}_{AB}$ et $B[-2,-1]C \notin \mathcal{G}_{BC}$.

La chronique définie par $\mathcal{C} = (ACC, \{A[1,3]C^1, A[1,3]C^2, C^1[0,+\infty]C^2)\})$, où C^1 représente le type d'événement du premier des deux événements de type C acceptés par la chronique $\mathcal{C}$, n'est pas une $\mathcal{D}_0$-chronique puisqu'il n'existe pas dans $\mathcal{D}_0$ de graphe de contraintes pour le couple (C, C).

4.1.4.5 Différence avec les bases de contraintes de Duong

Dans le processus de découverte de chroniques proposé par Duong (Dousson & Duong 1999, Duong 2001), il existe également un concept de base de contraintes temporelles. La différence est que la base de contraintes proposée par Duong ne comporte qu'une seule contrainte temporelle pour chaque couple de types d'événement. Il n'y a donc pas dans les travaux de Duong de notion de graphe de contraintes, ni de relation parent/enfant entre chroniques, ni de $\mathcal{D}$-chronique. Les relations d'ascendance et de descendance sont toutefois introduites par Duong sous la forme de *sous-chroniques* et de *sur-chroniques* d'une chronique donnée.

Avec une telle base de contraintes temporelles, les chroniques découvertes par Duong étant implicitement également des $\mathcal{D}$-chroniques, il ne peut donc être découvert que des chroniques n'ayant qu'un seul type de contraintes par couple de type d'événement pour un épisode donné.

Exemple
Supposons que la base de contraintes temporelles utilisée en entrée de l'algorithme de découverte de Duong soit $\mathcal{D}$ définie par :

$$\begin{aligned}\mathcal{D} = \{&(A, B, \{A[2,4]B\}) \\ &(A, C, \{A[3,5]C\}) \\ &(B, C, \{B[1,1]C\})\end{aligned}$$

Il ne peut alors y avoir comme $\mathcal{D}$-chroniques découvertes par le processus de Duong basées sur

l'épisode ABC que :

$$(ABC, \{\})$$
$$(ABC, \{A[2,4]B\})$$
$$(ABC, \{A[3,5]C\})$$
$$(ABC, \{B[1,1]C\})$$
$$(ABC, \{A[2,4]B, A[3,5]C\})$$
$$(ABC, \{A[2,4]B, B[1,1]C\})$$
$$(ABC, \{A[3,5]C, B[1,1]C\})$$
$$(ABC, \{A[2,4]B, A[3,5]C, B[1,1]C\})$$

Cela signifie, que parmi toutes les chroniques générées contenant les deux types d'événement A et B, toutes contiendront soit la contrainte temporelle $A[2,4]B$ soit aucune contrainte temporelle pour le couple (A,B). Si jamais la contrainte temporelle $A[-10,0]B$ (« B suivie de A dans les 10 unités de temps qui suivent) peut sembler intéressante pour l'analyste (c'est-à-dire l'utilisateur qui mène la découverte[23]*), alors il ne pourra jamais la découvrir.*

Comme le montre cet exemple, c'est au niveau de la base de contraintes temporelles que l'on trouve l'explication de la non-complétude de l'algorithme de découverte proposé par Dousson & Duong. Nous verrons plus loin ce que nous entendons par base de contraintes *complète* et comment la possibilité de pouvoir stocker plusieurs contraintes temporelles par graphe permet de généraliser la découverte de chroniques à la découverte de motifs plus variés (cf. section 3.3).

4.1.5 Énoncé du problème de découverte complète de chroniques à partir de traces

Au vu des définitions précédentes, le problème de découverte complète de chroniques à partir de traces peut se formuler ainsi :

> Étant donnée une trace $\mathcal{S}$, un seuil de fréquence f_{seuil}, et une base de contraintes $\mathcal{D}$, trouver toutes les $\mathcal{D}$-chroniques minimales telles que $f_{CRS}(\mathcal{C}) \geq f_{seuil}$.

Dit autrement, dans l'ensemble de toutes les $\mathcal{D}$-chroniques fréquentes qui existent dans une trace donnée, le problème de découverte complète de chroniques consiste à ne garder que les minimales. Ainsi, si une $\mathcal{D}$-chronique est fréquente alors elle aura nécessairement une $\mathcal{D}$-chronique descendante minimale présente dans l'ensemble retournée par l'algorithme.

Cette formulation du problème de découverte dite « complète » de chronique, n'est en fait qu'une découverte complète de $\mathcal{D}$-chroniques, pour $\mathcal{D}$ donnée. La découverte ne peut être considéré vraiment complète que si la base de contraintes temporelles $\mathcal{D}$ donnée en entrée est considérée comme complète. Le problème de construction d'une base de contraintes temporelles dite « complète » ou d'une base de contraintes d'un autre type est un problème complexe que nous aborderons dans la section 4.3.

[23] Nous rappelons que l'*analyste* est l'humain qui utilise les outils de découverte et de construction de connaissances. Ce terme *analyste* a été préféré au terme *utilisateur* pour ne pas confondre l'utilisateur de l'algorithme de découverte de connaissances à partir de traces (l'analyste) et les utilisateurs des environnements tracés. Ce choix a été effectué au chapitre 2, section 2.2.1.1.

4.2 Méthode de découverte complète de $\mathcal{D}$-chroniques

Cette partie présente la méthode et ses algorithmes permettant de réaliser une découverte complète de chroniques telle que formulée ci-dessus et répondant aux critères d'interactivité que nous avons posés pour la réalisation de l'analyseur automatique de traces (cf. section 2.3.3).

Le processus de découverte de chroniques proposé par Duong consiste à générer et compter des chroniques candidates de longueur n et ayant k contraintes temporelles à partir de la base de contraintes et des chroniques fréquentes de longueur n ayant $k-1$ contraintes temporelles. Cette méthode parcourt donc « en largeur d'abord » l'espace des chroniques candidates. Il s'agit en quelque sorte d'une extension de la méthode développée par Agrawal & Srikant (1995) pour la découverte d'épisodes parallèles à partir de séquences d'événements, sauf que Duong ajoute un sous-niveau d'itération k pour le parcours de l'ensemble des contraintes possibles.

Cette méthode ne peut pas être appliquée dans notre cas, car ayant plus d'un choix de contraintes temporelles par couple de types d'événement, il n'est pas évident de savoir comment générer toutes les chroniques candidates de longueur n ayant k contraintes temporelles.

4.2.1 Générer et compter

La méthode de découverte complète de chroniques proposée dans ce rapport applique également le principe « générer et compter », mais ne considère plus la longueur n de la chronique et son nombre de contraintes temporelles k comme des facteurs d'accroissement des chroniques lors du processus itératif de génération. En effet, avec une base de contraintes temporelles ayant plusieurs contraintes temporelles pour chaque couple de types d'événement, il est toujours possible de générer des chroniques candidates plus contraintes sans augmenter n, ni k, mais seulement en choisissant dans la base de contraintes temporelles une contrainte temporelle plus stricte pour un couple donné de types d'événement de la chronique. Par exemple, à partir de la chronique $(ABC, \{A[-1,5]B\})$ ($n=3$ et $k=1$) et de la base de contraintes $\mathcal{D}_0$ de la figure 4.12, il est possible de générer deux chroniques candidates plus contraintes (c'est-à-dire descendantes) tout en gardant $n=3$ et $k=1$: $(ABC, \{A[1,5]B\})$ et $(ABC, \{A[-1,3]B\})$.

Au lieu de tenter de générer les chroniques candidates en augmentant n et k itérativement, ce qui n'est pas possible dans l'optique de la complétude, nous proposons de générer pour chaque chronique fréquente l'ensemble des chroniques candidates qui lui sont « élémentairement » plus contraintes.

4.2.2 Opérateurs de génération de chroniques candidates plus contraintes

Afin de formaliser cette génération de chroniques candidates « élémentairement » plus contraintes, nous généralisons les relations `is_parent_of` et `is_child_of` définies sur les contraintes temporelles aux $\mathcal{D}$-chroniques. Par « chronique élémentairement plus contrainte » nous entendons chronique *enfant*.

Définition 12 (Relations `is_parent_of`/`is_child_of` sur les $\mathcal{D}$-chroniques)
Étant donnée une base de contraintes temporelles $\mathcal{D}$, la relation `is_child_of` entre deux $\mathcal{D}$-chroniques $\mathcal{C}$ et $\mathcal{C}'$ est définie par : « $\mathcal{C}$ `is_child_of` $\mathcal{C}'$ » si et seulement si :

1. *$\mathcal{C}$ est une descendante stricte de $\mathcal{C}'$ ($\mathcal{C} \prec \mathcal{C}'$),*
2. *il n'existe pas de $\mathcal{D}$-chronique $\mathcal{C}''$ telle que $\mathcal{C} \prec \mathcal{C}'' \prec \mathcal{C}'$.*

Exemple
Sur la figure 4.13 (voir page 107) toutes les chroniques sont des $\mathcal{D}_0$-chroniques. Sur cette figure, on observe que $\mathcal{C}_7 \prec \mathcal{C}_3$. Cependant, $\mathcal{C}_7$ n'est pas une chronique enfant de $\mathcal{C}_3$ car $\mathcal{C}_4$ est une $\mathcal{D}_0$-chronique et que $\mathcal{C}_7 \prec \mathcal{C}_4 \prec \mathcal{C}_3$. Néanmoins, $\mathcal{C}_4$ `is_child_of` $\mathcal{C}_3$ et $\mathcal{C}_7$ `is_child_of` $\mathcal{C}_4$.

Nous proposons ici deux types opérateurs de générations de chroniques enfant à partir d'une $\mathcal{D}$-chronique supposée fréquente $\mathcal{C}$:

`add_`ε' ajoute un type d'événement à $\mathcal{C}$,

`str_`$\varepsilon_i\varepsilon_j$ renforce la contrainte temporelle d'un niveau $\mathcal{C}$ pour le couple $(\varepsilon_i, \varepsilon_j)$.

Les deux sous-sections suivantes décrivent en détails le fonctionnement de ces deux types d'opérateurs. On note $\mathcal{C} = (\mathcal{E}, \mathcal{T})$, $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$ et $\mathcal{T} = (\tau_{ij})_{1 \leq i < j \leq n}$.

Nous prouverons dans la section 4.2.6 que ces deux types d'opérateur combinés permettent la génération complète de toutes les chroniques candidates.

4.2.2.1 L'opérateur `add_`ε'

L'opérateur `add_`ε' crée des chroniques candidates à partir de la chronique supposée fréquente $\mathcal{C}$ en ajoutant le type d'événement ε' à l'épisode de $\mathcal{C}$, c'est-à-dire à $\mathcal{E}$. Cette opération d'ajout est en réalité un peu plus compliquée qu'elle en a l'air puisque les chroniques générées doivent d'une part être également des $\mathcal{D}$-chroniques et doivent d'autre part être pleines, c'est-à-dire qu'elles doivent avoir des contraintes temporelles définies pour chaque couple de types d'événement (cf. section 4.1).

Les nouvelles chroniques générées sont basée sur l'épisode $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$ auquel on a ajouté le type d'événement ε'. Notons $\mathcal{E}'$ l'épisode de longueur $n+1$ obtenu lorsqu'on ajoute ε' à $\mathcal{E}$. Comme l'écriture des épisodes doit toujours s'effectuer en respectant l'ordre $\leq_{\mathbb{E}}$ défini sur $\mathbb{E}$, l'élément ε' ne s'ajoutera pas forcément à la dernière position de $\mathcal{E}'$. Notons donc i' son indice dans le nouvel épisode $\mathcal{E}'$. On a ainsi :

$$\mathcal{E}' = \varepsilon_1 \dots \varepsilon_{i'-1} \varepsilon' \varepsilon_{i'} \dots \varepsilon_n.$$

L'opérateur `add_`ε' est applicable à la chronique $\mathcal{C}$ si et seulement s'il existe dans la base de contraintes $\mathcal{D}$ des graphes de contraintes définis pour tous les nouveaux couples de $\mathcal{E}'$, c'est-à-dire si :

- pour chaque couple $(\varepsilon_i, \varepsilon')$ tel que $i < i'$, le graphe de contraintes $\mathcal{G}_{\varepsilon_i\varepsilon'}$ existe dans $\mathcal{D}$,
- pour chaque couple $(\varepsilon', \varepsilon_i)$ tel que $i' \leq i$, le graphe de contraintes $\mathcal{G}_{\varepsilon'\varepsilon_i}$ existe dans $\mathcal{D}$.

Si l'opérateur `add_`ε' est applicable à la chronique supposée fréquente $\mathcal{C}$, alors il créera des chroniques candidates $\mathcal{C}' = (\mathcal{E}', \mathcal{T}')$ en procédant ainsi :

1. copier les contraintes temporelles de $\mathcal{C}$, c'est-à-dire mettre tous les éléments de $\mathcal{T}$ dans $\mathcal{T}'$,
2. pour chaque couple $(\varepsilon_i, \varepsilon')$ tel que $i < i'$, choisir parmi les éléments maximaux du graphe de contraintes $\mathcal{G}_{\varepsilon_i\varepsilon'}$ une contrainte temporelle et l'ajouter à $\mathcal{T}'$,
3. pour chaque couple $(\varepsilon', \varepsilon_i)$ tel que $i' \leq i$, choisir parmi les éléments maximaux du graphe de contraintes $\mathcal{G}_{\varepsilon'\varepsilon_i}$ une contrainte temporelle et l'ajouter à $\mathcal{T}'$.

L'opérateur `add_`ε' créera donc à partir de la chronique supposée fréquente $\mathcal{C}$ autant de chroniques candidates qu'il existe de combinaisons entre les éléments des $\mathcal{G}^{\top}_{\varepsilon_i\varepsilon'}$ et des $\mathcal{G}^{\top}_{\varepsilon'\varepsilon_i}$. Le nombre total de chroniques candidates créées par `add_`ε' est :

$$\sum_{i=1}^{i'-1} |\mathcal{G}^{\top}_{\varepsilon_i\varepsilon'}| + \sum_{i=i'}^{n} |\mathcal{G}^{\top}_{\varepsilon'\varepsilon_i}|.$$

Dans la pratique, il est rare que les bases de contraintes possèdent plus d'un seul élément maximal par graphe. Dans la section 4.3, les méthodes de constructions de bases de contraintes que nous proposons produisent de tels graphes avec seulement un élément maximal. Dans de telles bases de contraintes, la formule ci-dessus vaut 1 et l'opérateur `add_`ε' ne produit donc qu'une seule chronique candidate. En revanche, il existe autant d'opérateurs du type `add_`ε' qu'il existe de ε' dans $\mathbb{E}$. Dit autrement, il existe $|\mathbb{E}|$ opérateurs de types `add_`ε', mais tous ne seront pas toujours nécessairement applicables pour les raisons évoquées ci-dessus.

Exemple
Prenons l'exemple de la chronique supposée fréquente $\mathcal{C}_3$ de la figure 4.13 ($\mathcal{C}_3 = (AC, \{A[-3,4]C\})$) et de la base de contraintes $\mathcal{D}_0$ de la figure 4.12. Dans ce cas, l'ensemble des types d'événement est $\mathbb{E} = \{A, B, C\}$. Il existe alors trois opérateurs du type `add_`ε' *:*`add_`A*,* `add_`B *et* `add_`C*.*

L'opérateur `add_`B *appliqué à $\mathcal{C}_3$ créera la chronique $\mathcal{C}'_3$ de longueur 3 et dont l'épisode est ABC ($i' = 2$). Pour établir l'ensemble des contraintes temporelles de $\mathcal{C}'_3$, l'opérateur* `add_`B *reprend la contrainte temporelle $A[-3,4]C$ de $\mathcal{C}_3$. Pour le couple (A, B), la contrainte temporelle est à choisir parmi l'ensemble des contraintes maximales du graphe $\mathcal{G}_{AB}$ de $\mathcal{D}_0$, c'est-à-dire parmi $\mathcal{G}^{\top}_{AB}$, c'est-à-dire $A[-1,5]B$. De même pour (B, C), $\mathcal{G}^{\top}_{BC}$ ne contient qu'un seul élément : $B[-4,1]C$. L'opérateur* `add_`B *ne produit donc qu'une seule chronique candidate : $\mathcal{C}'_3 = (ABC, \{A[-1,5]B, A[-3,4]C, B[0,4]C\})$. Sur la figure 4.13, $\mathcal{C}'_3$ est en réalité $\mathcal{C}_5$.*

En revanche, l'opérateur `add_`A *n'est pas applicable à $\mathcal{C}_3$ car il n'existe pas de graphe de contraintes dans $\mathcal{D}_0$ pour le couple (A, A).*

De même, l'opérateur `add_`C *n'est pas applicable à $\mathcal{C}_3$ car il n'existe pas de graphe de contraintes dans $\mathcal{D}_0$ pour le couple (C, C).*

4.2.2.2 L'opérateur `str_`$\varepsilon_i\varepsilon_j$

L'opérateur `str_`$\varepsilon_i\varepsilon_j$ (`str` est l'abréviation de *strengthen*, qui signifie « renforcer » en anglais pour une contrainte, ou aussi « rendre plus stricte ») renforce la contrainte temporelle $\tau_{ij} = \varepsilon_i[\tau^-_{ij}, \tau^+_{ij}]\varepsilon_j$ de la $\mathcal{D}$-chronique supposée fréquente $\mathcal{C}$ en la remplaçant par une contrainte σ_{ij} telle que σ_{ij} `is_child_of` τ_{ij} dans le graphe de contraintes $\mathcal{G}_{\varepsilon_i\varepsilon_j}$.

Si la contrainte temporelle τ_{ij} est déjà minimale dans le graphe $\mathcal{G}_{\varepsilon_i\varepsilon_j}$ alors `str_`$\varepsilon_i\varepsilon_j$ n'est pas applicable à la chronique $\mathcal{C}$. De même, si la base de contraintes $\mathcal{D}$ ne contient pas de graphe de contraintes pour le couple $(\varepsilon_i, \varepsilon_j)$, alors la contrainte `str_`$\varepsilon_i\varepsilon_j$ n'est pas non plus applicable à $\mathcal{C}$, cependant cette situation ne devrait jamais arriver dans le processus de découverte car cela signifierait que la chronique supposée fréquente $\mathcal{C}$ ne serait pas une $\mathcal{D}$-chronique puisqu'elle contiendrait aussi les deux types d'événement ε_i et ε_j.

Pour une $\mathcal{D}$-chronique supposée fréquente $\mathcal{C}$ donnée, l'opérateur `str_`$\varepsilon_i\varepsilon_j$ produira, s'il est applicable à $\mathcal{C}$, q $\mathcal{D}$-chroniques candidates, où q est le nombre de contraintes temporelles enfant de τ_{ij} dans $\mathcal{G}_{\varepsilon_i\varepsilon_j}$. Toutes les chroniques produites par les opérateurs de type `str` sont de longueur n, où n est la longueur de $\mathcal{C}$.

Exemple
Étant donnée la base de contraintes $\mathcal{D}_0$ de la figure 4.12, le seul opérateur de type `str` *applicable à la $\mathcal{D}_0$-chronique $\mathcal{C}_3$ est* `str_AB`*. Dans le graphe de contraintes $\mathcal{G}_{AB}$ de la base de contraintes $\mathcal{D}_0$, la contrainte temporelle $A[-3,4]C$ a deux enfants : $A[-3,1]C$ et $A[0,4]C$. Dans la chronique $\mathcal{C}_3$, en remplaçant la contrainte temporelle $A[-3,4]C$ par l'un de ses enfant dans le graphe $\mathcal{G}_{AB}$ d'une part, l'opérateur* `str_AB` *appliqué à $\mathcal{C}_3$ produit la chronique $(AC, \{A[-3,1]C\}$ (c'est-à-dire $\mathcal{C}_4$), et par son deuxième enfant d'autre part,* `str_AB` *produit la chronique $(AC, \{A[0,1]4\}$ (c'est-à-dire $\mathcal{C}_8$). Au total,* `str_AB` *produit donc deux $\mathcal{D}_0$-chroniques candidates.*

4.2.3 Graphe d'exploration des $\mathcal{D}$-chroniques

L'algorithme `générer_enfants` (cf. algorithme 1) implémente la génération de chroniques candidates à partir des opérateurs `add` et `str` en générant toutes les chroniques enfant d'une chronique supposée fréquente $\mathcal{C}$. Les procédures *appliquer_operateur*(`op`, $\mathcal{C}$) (cf. lignes 4 et 10) appliquent l'opérateur `op` à la chronique $\mathcal{C}$ et retournent l'ensemble des chroniques générées par l'application de cet opérateur `op`.

Algorithme 1 `générer_enfants` : Génération de chroniques candidates.

Entrées: $\mathcal{C} = (\mathcal{E}, \mathcal{T})$;$\mathcal{D}$;$\mathbb{E}$
Sorties: `Candidats`

1: `Candidats` $\leftarrow \emptyset$
2: `candidats_add` $\leftarrow \emptyset$
3: **pour** chaque $\varepsilon' \in \mathbb{E}$ **faire**
4: `Candidats_add` $\leftarrow$ `appliquer_operateur(add_`ε'`,`$\mathcal{C}$`)`
5: **fin pour**
6: `Candidats` $\leftarrow$ `Candidats` $\cup$ `Candidats_add`
7: `Candidats_str` $\leftarrow \emptyset$
8: **pour** chaque $\tau_{ij} \in \mathcal{T}$ **faire**
9: (en notant $\tau_{ij} = \varepsilon_i[i^-, i^+]\varepsilon_j$)
10: `Candidats_str`$\leftarrow$ `appliquer_operateur(str_`$\varepsilon_i\varepsilon_j$`,`$\mathcal{C}$`)`
11: **fin pour**
12: `Candidats` $\leftarrow$ `Candidats` $\cup$ `Candidats_str`
13: **retourner** `Candidats`

La figure 4.13 montre l'arbre d'exploration des $\mathcal{D}_0$-chroniques candidates générées par l'algorithme 1 sur les trois premiers niveaux de profondeur en partant, pour le niveau 0, des chroniques de longueur 2 maximales dans la base de contraintes $\mathcal{D}_0$.

4.2.4 Algorithme de découverte complète de $\mathcal{D}$-chroniques fréquentes minimales

L'algorithme `CDA`[24] (algorithme 3, sur la page détachable 109) implémente la découverte complète de $\mathcal{D}$-chroniques. L'ensemble `Fréquents` contient à tout moment de l'exécution l'ensemble des $\mathcal{D}$-chroniques fréquentes minimales parmi l'ensemble des chroniques traitées par la boucle itérative des lignes 5 à 26. À la fin de l'exécution de l'algorithme, `Fréquents` est donc l'ensemble de toutes les chroniques fréquentes minimales qui sont retournées à l'analyste.

L'ensemble `Candidats` est l'ensemble des chroniques candidates pour lesquelles il faut établir la fréquence et la minimalité. Le principe de fonctionnement général de l'algorithme `CDA` est de

[24] Chronicle Discovery Algorithm

répéter la boucle des lignes 5 à 26 jusqu'à ce que l'ensemble `Candidats` soit vide les instructions suivantes : prendre une chronique candidate $\mathcal{C}$ dans l'ensemble `Candidats` (l. 6) (« prendre » implique « retirer » de l'ensemble `Candidats`), tester sa fréquence dans la trace $\mathcal{S}$ et sa minimalité dans l'ensemble `Fréquents`, et si $\mathcal{C}$ est fréquente et minimale alors ajouter toutes ses $\mathcal{D}$-chroniques enfant dans l'ensemble `Candidats`. L'ensemble `Candidats` est initialisé avec $\mathcal{D}^{\top}$ (l. 4), c'est-à-dire avec l'ensemble des $\mathcal{D}$-chroniques de longueur 2 qui sont les contraintes temporelles maximales des graphes de la base de contraintes $\mathcal{D}$.

Plus précisément, lors d'une itération de la boucle des lignes 5 à 26, l'algorithme `CDA` opère comme suit. De même qu'un ensemble `Fréquents` des $\mathcal{D}$-chroniques minimales est maintenu à jour au cours de l'exécution de `CDA`, un ensemble `Non_Fréquents` des $\mathcal{D}$-chroniques non fréquentes maximales est maintenu à jour. Cet ensemble `Non_Fréquents` est utilisé par la procédure `contient_ascendants` (l. 8), qui teste s'il existe dans `Non_Fréquents` une $\mathcal{D}$-chronique plus générale que $\mathcal{C}$. Si tel est le cas, alors la $\mathcal{D}$-chronique $\mathcal{C}$ ne peut pas être fréquente (par monotonicité de $\preceq$) et l'algorithme reprend la boucle à la prochaine itération. De même, la procédure `contient_descendants` (l. 11) teste si la $\mathcal{D}$-chronique candidate $\mathcal{C}$ possède un descendant qui est fréquent. Si tel est le cas, alors $\mathcal{C}$ est assurée d'être fréquente par monotonicité. Nous ne discuterons pas de méthodes efficaces dans ce mémoire pour l'implémentation des procédures `contient_ascendants` et `contient_descendants`. Nous proposons pour ces deux procédures une implémentation naïve, c'est-à-dire pour `contient_ascendants` (resp. `contient_descendants`) un parcours linéaire des éléments de l'ensemble `Non_Fréquents` (resp. `Fréquents`). Nous verrons en section 4.5.4 que ce choix n'impacte pas de manière significative le temps d'exécution de l'algorithme `CDA`.

Dans le cas contraire, c'est-à-dire si on ne peut pas établir la fréquence de la $\mathcal{D}$-chronique candidate $\mathcal{C}$ à l'aide des deux ensembles `Fréquents` et `Non_Fréquents`, alors il faut compter $\mathcal{C}$. C'est le rôle des lignes 12 à 18. La candidate $\mathcal{C}$ est d'abord comptée dans la trace $\mathcal{S}$ (l. 12) à l'aide de l'algorithme CRS de Dousson et al. (Dousson et al. 1993, Dousson et al. 2007). Deux cas se présentent alors.

1. La fréquence de la candidate $\mathcal{C}$ est supérieure à f_{seuil}. $\mathcal{C}$ est donc fréquente dans la trace $\mathcal{S}$. Dans ce cas, la procédure `ajout_minimal` ajoute la candidate $\mathcal{C}$ en tant que chronique minimale de l'ensemble `Fréquents` (l. 14), c'est-à-dire que les chroniques présentes dans l'ensemble `Fréquents` à ce moment-là et qui sont ascendantes de $\mathcal{C}$ sont supprimées de l'ensemble `Fréquents`, afin de ne garder que les minimales.

2. La fréquence de la candidate $\mathcal{C}$ est strictement inférieure à f_{seuil}. $\mathcal{C}$ n'est donc pas fréquente dans la trace $\mathcal{S}$. Dans ce cas, la procédure `ajout_maximal` ajoute la candidate $\mathcal{C}$ non fréquente à l'ensemble `Non_Fréquents` tout en supprimant de `Non_Fréquents` les chroniques descendantes de $\mathcal{C}$ (l. 16), afin de ne garder que les maximales. Ensuite, la candidate $\mathcal{C}$ n'étant pas fréquente, elle est abandonnée (l. 17) et l'exécution se poursuit à la prochaine itération.

Les lignes 20 à 25 sont exécutées si la $\mathcal{D}$-chronique candidate $\mathcal{C}$ est fréquente, que $\mathcal{C}$ ait été comptée ou non. La procédure `générer_enfants` vue à la section 4.2.2 génère alors toutes les chroniques enfant de la $\mathcal{C}$ (l. 20). Chaque enfant $\mathcal{C}'$ de la chronique $\mathcal{C}$ est ajouté à l'ensemble des chroniques candidates (l. 23) si $\mathcal{C}'$ remplit les deux conditions suivantes :

1. $\mathcal{C}'$ n'est pas déjà fréquente dans `Candidats` (c'est le sens de l'expression `Candidats` $\leftarrow$ `Candidats` $\cup \{\mathcal{C}'\}$),

2. $\mathcal{C}'$ n'a jamais été traitée par une itération précédente de l'algorithme `CDA`, c'est-à-dire $\mathcal{C}' \notin$ `Traitées`. Le but de ce test est de ne jamais mettre deux fois la même chronique dans

l'ensemble `Candidats` au cours de la même exécution de `CDA`, afin d'assurer la terminaison de `CDA`. L'ensemble `Traitées` est maintenu à jour à chaque fois qu'une nouvelle chronique candidate $\mathcal{C}$ est prise dans l'ensemble `Candidats` (l. 7). Nous discuterons dans la section 4.5.4 des problèmes de complexité en mémoire et en temps posés d'une part par le stockage de l'ensemble des chroniques déjà traitées et d'autre part par le test de non appartenance des chroniques $\mathcal{C}'$ à cet ensemble. Il existe des solutions alternatives pour assurer la terminaison de l'algorithme sans stocker l'ensemble des chroniques déjà traitées (cf. section 4.4.2).

4.2.5 Terminaison

Cette partie vise à prouver que l'algorithme `CDA` se termine toujours, quels que soient ses paramètres d'entrée.

Prenons une base de contraintes $\mathcal{D}$, une trace $\mathcal{S}$ et une fréquence seuil f_{seuil}. Notons n_{max} la longueur de la plus longue $\mathcal{D}$-chronique fréquente minimale, si elle existe[25]. Étant donné que la procédure `générer_enfants` (l. 20) ne s'applique qu'à des $\mathcal{D}$-chroniques fréquentes, elle ne s'appliquera alors jamais à des $\mathcal{D}$-chroniques de longueur $n_{max}+1$. En conséquence, il n'y aura jamais de $\mathcal{D}$-chronique de longueur $n_{max}+2$ qui sera générée et ajoutée à l'ensemble `Candidats`. Or, la base de contraintes étant finie, c'est-à-dire contient un nombre fini de graphes finis (nous n'envisageons d'ailleurs pas qu'une base de contraintes puisse être infinie dans le cadre de la découverte de chroniques), il existe un nombre fini de $\mathcal{D}$-chroniques de longueur inférieure à $n_{max}+1$. De plus, ne mettant jamais deux fois la même chronique dans l'ensemble `Candidats` grâce au test $\mathcal{C}' \notin$ `Traitées` (l. 22), il est certain que l'algorithme `CDA` mettra un nombre fini de $\mathcal{D}$-chroniques dans l'ensemble `Candidats`. Comme à chaque itération de `CDA` une $\mathcal{D}$-chronique est retirée de l'ensemble `Candidats` (l. 6), il est certain qu'il y aura un moment où l'ensemble `Candidats` sera vide et donc où la boucle des lignes 5 à 26 se terminera, ce qui prouve la terminaison de `CDA`.

4.2.6 Complétude

Dans cette partie, nous cherchons à démontrer que l'algorithme `CDA` présenté précédemment est effectivement complet, c'est-à-dire que pour une trace $\mathcal{S}$ donnée, pour une base de contraintes $\mathcal{D}$ donnée et pour une fréquence seuil donnée, toute $\mathcal{D}$-chronique fréquente sera présente dans l'ensemble `Fréquents` retourné par `CDA` ou bien aura une chronique descendante qui sera présente dans `Fréquents`. Afin de mener à bien cette preuve, nous devons introduire la notion de *chemin* vers une $\mathcal{D}$-chronique et la notion de *profondeur* d'une $\mathcal{D}$-chronique.

4.2.6.1 Profondeur d'une $\mathcal{D}$-chronique : facteur d'accroissement

Comme pour les notions d'ascendance et de descendance, nous devons d'abord définir les notions de chemin et de profondeur pour les contraintes temporelles, puis les généraliser aux $\mathcal{D}$-chroniques.

Définition 13 (Chemin vers une contrainte temporelle)
Dans un graphe de contraintes $\mathcal{G}$ de la base de contraintes $\mathcal{D}$, le n-uplet de contraintes temporelles $[\tau_0, \ldots, \tau_n]$ est un chemin *vers la contrainte temporelle τ si et seulement les conditions suivantes sont vérifiées :*

1. $\tau = \tau_n$,

[25] Il se pourrait qu'il n'existe pas de $\mathcal{D}$-chronique fréquente dans la trace $\mathcal{D}$ pour la fréquence seuil f_{seuil} donnée et pour la base de contraintes $\mathcal{D}$. Dans ce cas `CDA` se termine bien puisqu'il ne met jamais aucun élément dans l'ensemble `Fréquents`.

2. $\tau_0 \in \mathcal{G}^\top$,

3. $\forall i,\ \tau_i \in \mathcal{G}$,

4. $\forall i, \tau_{i+1}$ `is_child_of` τ_i *in* $\mathcal{G}$.

Définition 14 (Profondeur d'une contrainte temporelle)
La profondeur *d'une contrainte temporelle τ dans un graphe de contraintes τ tel que $\tau \in \mathcal{G}$ est définie comme étant la longueur du plus court chemin vers τ dans $\mathcal{G}$, c'est-à-dire le nombre de contraintes temporelles du plus court chemin vers τ, moins* 1.

La profondeur d'une contrainte temporelle τ dans son graphe $\mathcal{G}$ se note $prof(\tau, \mathcal{G})$.

$$prof(\tau, \mathcal{G}) = |[\tau_0, \ldots, \tau_n]| - 1$$

où $[\tau_0, \ldots, \tau_n]$ est le plus court chemin vers τ dans $\mathcal{G}$.

Les notions de profondeur et de chemin pour des contraintes temporelles se comprennent mieux graphiquement. Sur un schéma de graphe de contraintes, un chemin vers τ correspond à un parcours depuis l'une des contraintes maximales du graphe jusqu'à τ et la profondeur d'une contrainte temporelle correspond au plus petit nombre d'arcs à traverser depuis l'une des contraintes maximales du graphe pour l'atteindre.

Exemple
Dans le graphe de contraintes $\mathcal{G}_{AB}$ de la base de contraintes $\mathcal{D}_0$ de la figure 4.12, la seule contrainte maximale est $A[-1,5]B$ (c'est-à-dire $\mathcal{G}_{AB}^\top = \{A[-1,5]B\}$), donc tous les chemins de ce graphe partent nécessairement de $A[-1,5]B$. L'ensemble des chemins vers la contrainte $A[-1,1]B$ est $\{[A[-1,5]B, A[-1,3]B, A[-1,1]B]\}$ (un seul chemin). L'ensemble des chemins vers la contrainte $A[1,3]B$ $\{[A[-1,5]B, A[-1,3]B, A[1,3]B], [A[-1,5]B, A[1,5]B, A[1,3]B]\}$ (deux chemins, tous deux de longueur 2). Les différentes profondeurs de ce graphe sont :

- $prof(A[-1,5]B, \mathcal{G}_{AB}) = 0$,
- $prof(A[-1,3]B, \mathcal{G}_{AB}) = 1$,
- $prof(A[1,5]B, \mathcal{G}_{AB}) = 1$,
- $prof(A[-1,1]B, \mathcal{G}_{AB}) = 2$,
- $prof(A[1,3]B, \mathcal{G}_{AB}) = 2$,
- $prof(A[3,5]B, \mathcal{G}_{AB}) = 2$.

Définition 15 (Graphe de contraintes régulier/Base de contraintes régulière)
Une graphe de contraintes $\mathcal{G}$ est dit régulier *si pour toute contrainte temporelle τ de $\mathcal{G}$, tous les chemins vers τ sont de la même longueur.*

Une base de contraintes temporelles est dite régulière *si tous ses graphes de contraintes sont réguliers.*

Exemple
La figure 4.2 montre deux configurations de base de contraintes, l'une régulière (à gauche de la figure) et l'autre non (à droite). En effet, dans le graphe de droite, il y a deux chemins qui mènent à la contrainte temporelle encerclée. Le premier chemin, représenté en pointillés, est de longueur 2.

Le deuxième chemin, représenté en gras, est de longueur 3. *Nous pouvons définir la profondeur de cette contrainte temporelle, c'est* 2. *En revanche, les deux chemins menant à cette contrainte étant de longueurs différentes, le graphe de droite n'est pas régulier. À l'inverse, toutes les contraintes du graphe de gauche ont des chemins de longueurs égales, ce qui définit un graphe régulier.*

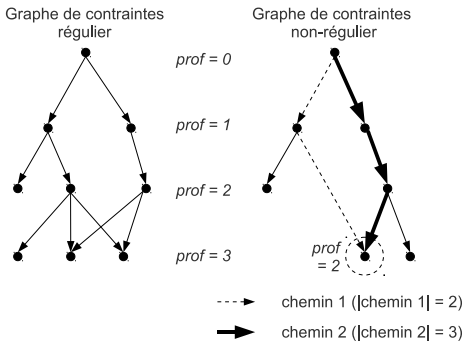


Figure 4.2 – Configuration schématique d'un graphe de contraintes régulier en comparaison avec un graphe de contraintes non régulier

Le théorème suivant découle naturellement de la définition de graphe régulier.

Théorème 1
Soit $\mathcal{G}$ un graphe de contraintes régulier et τ une contrainte temporelle de $\mathcal{G}$. Pour tout chemin $[\tau_0, \ldots, \tau_n]$ vers la contrainte τ dans le graphe $\mathcal{G}$, la profondeur de τ est égale à la longueur de ce chemin, moins 1.

Si nous avons introduit la notion de graphe régulier, c'est pour utiliser le résultat du théorème 1. L'intérêt du théorème 1 et des graphes réguliers est qu'il est possible de connaître la profondeur d'une contrainte temporelle τ à partir d'un seul des chemins qui mènent à τ, contrairement à la définition de la profondeur dans le cas général qui requiert de connaître tous les chemins qui mènent à τ pour n'en garder que le plus court.

En conséquence, la démonstration de la complétude ci-après ne sera valable que dans le cas des bases de contraintes régulières. Cela peut paraître limitatif, mais dans la pratique les bases de contraintes les plus communément utilisées pour la découverte de chroniques, comme les trois bases de contraintes présentées en section 4.3, sont toutes régulières.

Dans un graphe régulier, le théorème 2 donne la relation entre la profondeur d'une contrainte temporelle et ses contraintes enfants.

Théorème 2
Soit $\mathcal{G}$ un graphe de contraintes régulier, et soient τ et τ' deux contraintes temporelles du graphe $\mathcal{G}$ telles que τ' `is_child_of` τ, alors $prof(\tau', \mathcal{G}) = prof(\tau, \mathcal{G}) + 1$.

Preuve
Soit $[\tau_0 \ldots \tau_n]$ un chemin vers τ dans $\mathcal{G}$, c'est-à-dire que $\tau_n = \tau$. Comme τ' `is_child_of` τ et par définition d'un chemin, *on peut construire le chemin suivant vers τ' : $[\tau_0 \ldots \tau\tau']$. Comme nous sommes dans un graphe régulier, on peut calculer la profondeur directement à partir de n'importe quel chemin et donc en particulier à partir des deux chemins $[\tau_0 \ldots \tau]$ et $[\tau_0 \ldots \tau\tau']$. Ce qui nous*

autorise le calcul suivant :

$$\begin{aligned} prof(\tau', \mathcal{G}) &= |[\tau_0 \dots \tau\tau']| - 1 \\ &= (|[\tau_0 \dots \tau]| + 1) - 1 \\ &= (|[\tau_0 \dots \tau]| - 1) + 1 \\ &= prof(\tau, \mathcal{G}) + 1. \end{aligned}$$

Avant de pouvoir passer à la démonstration de la complétude, il convient de généraliser les notions de profondeur et de chemin aux $\mathcal{D}$-chroniques.

Définition 16 (Profondeur d'une $\mathcal{D}$-chronique)
Étant donnée une base de contraintes $\mathcal{D}$, la profondeur d'une $\mathcal{D}$-chronique $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ est définie par :

$$prof(\mathcal{C}, \mathcal{D}) = (|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}} prof(\tau, \mathcal{G}_{\varepsilon\varepsilon'}) \text{ (où } \tau = \varepsilon[\tau^-, \tau^+]\varepsilon')$$

Une conséquence intéressante de la formule et qui est largement utilisée dans la preuve du théorème ci-dessous est que pour toute contrainte temporelle τ appartenant à un graphe de contraintes $\mathcal{G}$ de la base $\mathcal{D}$, on a $prof(\tau, \mathcal{G}) = prof(\tau, \mathcal{D})$, c'est-à-dire que les définitions de profondeur pour les $\mathcal{D}$-chroniques sont applicables aux $\mathcal{D}$-chroniques de longueur 2, c'est-à-dire aux contraintes temporelles. On a en particulier : pour toutes contraintes temporelles τ appartenant $\mathcal{D}^\top$, $prof(\tau, \mathcal{D}) = prof(\tau, \mathcal{G}) = 0$.

Exemple
Sur le graphe de $\mathcal{D}_0$-chroniques de la figure 4.13, en appliquant la formule de la définition ci-dessus, on a $prof(\mathcal{C}_3, \mathcal{D}_0) = 0$ (ce qui était attendu puisque $\mathcal{C}_3 \in \mathcal{D}_0^\top$). On a également en appliquant la formule de la profondeur ci-dessus :

$$\begin{aligned} &prof(\mathcal{C}_3, \mathcal{D}_0) = 0, \\ &prof(\mathcal{C}_4, \mathcal{D}_0) = prof(\mathcal{C}_5, \mathcal{D}_0) = prof(\mathcal{C}_8, \mathcal{D}_0) = 1, \\ \text{et } &prof(\mathcal{C}_6, \mathcal{D}_0) = 2. \end{aligned}$$

Théorème 3
Soient $\mathcal{C}$ et $\mathcal{C}'$ deux $\mathcal{D}$-chroniques telles que $\mathcal{C}'$ `is_child_of` $\mathcal{C}$, alors $prof(\mathcal{C}', \mathcal{D}) = prof(\mathcal{C}, \mathcal{D}) + 1$.

Preuve
$\mathcal{C}'$ `is_child_of` $\mathcal{C}$, on a donc $\mathcal{C}' \prec \mathcal{C}$, et donc $length(\mathcal{C}') \geq length(\mathcal{C})$. Deux cas se distinguent alors : « $length(\mathcal{C}') = length(\mathcal{C})$ » et « $length(\mathcal{C}') > length(\mathcal{C})$ ». On note $\mathcal{C} = (\mathcal{E}, \mathcal{T})$, avec $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$ et $\mathcal{T} = (\tau_{ij})_{1 \leq i < j \leq n}$. De même $\mathcal{C}' = (\mathcal{E}', \mathcal{T}')$, avec $\mathcal{E}' = \varepsilon'_1 \dots \varepsilon'_{n'}$ et $\mathcal{T}' = (\tau'_{ij})_{1 \leq i < j \leq n'}$.

Cas 1 : **length**$(\mathcal{C}')$ = **length**$(\mathcal{C})$. *Dans ce cas $\varepsilon = \varepsilon'$, c'est-à-dire $\varepsilon_1 \dots \varepsilon_n = \varepsilon'_1 \dots \varepsilon'_{n'}$ car d'une part $\varepsilon' \subseteq \varepsilon$ (par définition de la relation de descendance $\mathcal{C}' \preceq \mathcal{C}$) et d'autre part ε et ε' sont de même longueur (car la longueur d'une chronique est égale par définition à la longueur de son épisode et que $n = n'$). Soit n la longueur de $\mathcal{C}$ et de $\mathcal{C}'$. On note $\mathcal{T} = (\tau_{ij})_{1 \leq i < j \leq n}$ et $\mathcal{T}' = (\tau'_{ij})_{1 \leq i < j \leq n}$. Comme $\mathcal{C}'$ est une descendante de $\mathcal{C}$, on a par définition de la descendance pour tout couple d'indices (i, j) vérifiant $1 \leq i < j \leq n$, $\tau'_{ij} \preceq \tau_{ij}$. De plus, comme $\mathcal{C}$ et $\mathcal{C}'$ sont des $\mathcal{D}$-chroniques, on a $\tau_{ij} \in \mathcal{G}_{ij}$ et $\tau'_{ij} \in \mathcal{G}_{ij}$, où $\mathcal{G}_{ij} \in \mathcal{D}$ est le graphe de contraintes temporelles pour le couple de types d'événement $(\varepsilon_i, \varepsilon_j)$ (qui est égal au couple $(\varepsilon'_i, \varepsilon'_j)$). Là encore, soit $\tau'_{ij} = \tau_{ij}$, soit $\tau'_{ij} \prec \tau_{ij}$.*

- *Dans le cas* $\tau'_{ij} = \tau_{ij}$*, que nous appellerons « cas (i) », on a de toute évidence* $prof(\tau_{ij}, \mathcal{G}_{ij}) = prof(\tau'_{ij}, \mathcal{G}_{ij})$.
- *Dans le cas* $\tau'_{ij} \prec \tau_{ij}$*, que nous appellerons « cas (ii) », on a en fait nécessairement* τ'_{ij} `is_child_of` τ_{ij} *dans le graphe* $\mathcal{G}_{ij}$*, car sinon il existerait une contrainte temporelle* τ''_{ij} *dans le graphe* $\mathcal{G}_{ij}$ *telle que* $\tau'_{ij} \prec \tau''_{ij} \prec \tau_{ij}$*, et alors en remplaçant* τ'_{ij} *par* τ''_{ij} *dans la chronique* $\mathcal{C}'$ *on obtiendrait alors une* $\mathcal{D}$*-chronique* $\mathcal{C}''$ *telle que* $\mathcal{C}' \prec \mathcal{C}'' \prec \mathcal{C}$*, ce qui est impossible puisque* $\mathcal{C}'$ `is_child_of` $\mathcal{C}$.

On a donc τ'_{ij} `is_child_of` τ_{ij}*, et donc d'après le théorème 2,* $prof(\tau'_{ij}, \mathcal{G}_{ij}) = prof(\tau_{ij}, \mathcal{G}_{ij}) + 1$*. Donc pour chaque couple* (i,j)*, on a soit (i)* $\tau'_{ij} = \tau_{ij}$ *et* $prof(\tau'_{ij}, \mathcal{G}_{ij}) = prof(\tau_{ij}, \mathcal{G}_{ij})$*, soit (ii)* $\tau'_{ij} \prec \tau_{ij}$ *et* $prof(\tau'_{ij}, \mathcal{G}_{ij}) = prof(\tau_{ij}, \mathcal{G}_{ij}) + 1$*. Si on est dans le cas (ii) pour tous les couples* (i,j)*, alors* $\mathcal{C} = \mathcal{C}'$*, ce qui n'est pas possible. Donc il existe au moins un couple* (i_1, j_1) *dans le cas (ii). Or il ne peut exister qu'un seul de ces couples. En effet, il existerait sinon un autre couple* (i_2, j_2) *dans le cas 2, alors on aurait à la fois* $\tau'_{i_1 j_1} \prec \tau_{i_1 j_1}$ *et* $\tau'_{i_2 j_2} \prec \tau_{i_2 j_2}$*. En construisant alors la chronique* $\mathcal{C}''$ *identique à* $\mathcal{C}$ *mais en remplaçant juste* $\tau_{i_1 j_1}$ *par* $\tau'_{i_1 j_1}$ *on aurait* $\mathcal{C}' \prec \mathcal{C}'' \prec \mathcal{C}$*, ce qui n'est pas possible puisque* $\mathcal{C}'$ `is_child_of` $\mathcal{C}$*. En conséquence on peut écrire la formule suivante :*

$$\begin{aligned}
prof(\mathcal{C}', \mathcal{D}) &= (|\mathcal{E}'| - 2) + \sum_{\tau \in \mathcal{T}'} prof(\tau, \mathcal{G}), \text{ où } \mathcal{G} \text{ est le graphe auquel appartient } \tau \\
&= (|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}'} prof(\tau, \mathcal{G}) \\
&= (|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}', \tau \neq \tau'_{i_1 j_1}} prof(\tau, \mathcal{G}) + prof(\tau'_{i_1 j_1}, \mathcal{G}_{\varepsilon_{i_1} \varepsilon_{j_1}}) \\
&= (|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}, \tau \neq \tau_{i_1 j_1}} prof(\tau, \mathcal{G}) + (prof(\tau_{i_1 j_1}, \mathcal{G}_{\varepsilon_{i_1} \varepsilon_{j_1}}) + 1) \\
&= (|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}} prof(\tau, \mathcal{G}) + 1 \\
&= prof(\mathcal{C}, \mathcal{D}) + 1.
\end{aligned}$$

Ce premier cas $length(\mathcal{C}') = length(\mathcal{C})$ *correspond en fait au cas où l'on a appliqué l'opérateur* `str_`$\varepsilon_i \varepsilon_j$ *à la* $\mathcal{D}$*-chronique* $\mathcal{C}$ *pour obtenir* $\mathcal{C}'$.

Cas 2 : **length**$(\mathcal{C}')$ > **length**$(\mathcal{C})$ *En tenant un raisonnement similaire à celui du premier cas, on déduit successivement que* $\mathcal{E} \subset \mathcal{E}'$*, que* $|\mathcal{E}'| = |\mathcal{E}| + 1$*, que* $\mathcal{E}' = \varepsilon_1 \ldots \varepsilon_{i^*-1} \varepsilon^* \varepsilon_{i^*} \ldots \varepsilon_n$*, c'est-à-dire que l'épisode* $\mathcal{E}'$ *est l'épisode* $\mathcal{E}$ *auquel on a ajouté le type d'événement* ε^* *à la position* i^**. On a donc le calcul suivant :*

$$\begin{aligned}
prof(\mathcal{C}', \mathcal{D}) &= (|\mathcal{E}'| - 2) + \sum_{\tau \in \mathcal{T}'} prof(\tau, \mathcal{G}), \text{ où } \mathcal{G} \text{ est le graphe auquel appartient } \tau \\
&= ((|\mathcal{E}| + 1) - 2) + \sum_{\tau \in \mathcal{T}} prof(\tau, \mathcal{G}) + \sum_{1 \leq i < i^*} prof(\tau'_{ii^*}, \mathcal{G}_{ii^*}) + \sum_{i^* \leq i < n} prof(\tau'_{i^* i}, \mathcal{G}_{i^* i}),
\end{aligned}$$

où les τ'_{ii^*} *et les* $\tau'_{i^* i}$ *sont les contraintes temporelles de la* $\mathcal{C}'$ *pour tous les couples* $(\varepsilon_i, \varepsilon^*)$ *et* $(\varepsilon^*, \varepsilon_i)$*, c'est-à-dire sont les contraintes temporelles pour tous les couples impliquant* ε^**. Avec un raisonnement similaire à celui du premier cas, on obtient que les* $prof(\tau'_{ii^*}, \mathcal{G}_{ii^*})$ *et les* $prof(\tau'_{i^* i}, \mathcal{G}_{i^* i})$ *valent* 0 *car sinon on pourrait construire des* $\mathcal{D}$*-chroniques* $\mathcal{C}''$ *telles que* $\mathcal{C}' \prec \mathcal{C}'' \prec \mathcal{C}$*, ce qui est*

impossible par définition de $\mathcal{C}'$ `is_child_of` $\mathcal{C}$. *Donc :*

$$\begin{aligned} prof(\mathcal{C}', \mathcal{D}) &= ((|\mathcal{E}| + 1) - 2) + \sum_{\tau \in \mathcal{T}} prof(\tau, \mathcal{G}) + 0 + 0 \\ &= ((|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}} prof(\tau, \mathcal{G})) + 1 \\ &= prof(\mathcal{C}, \mathcal{D}) + 1. \end{aligned}$$

Ce deuxième cas « $length(\mathcal{C}') > length(\mathcal{C})$ *» correspond en fait au cas où l'on a appliqué l'opérateur* `add_`ε^* *à la* $\mathcal{D}$*-chronique* $\mathcal{C}$ *pour obtenir* $\mathcal{C}'$.

Dans tous les cas, on a bien $prof(\mathcal{C}', \mathcal{D}) = prof(\mathcal{C}, \mathcal{D}) + 1$, *si* $\mathcal{C}'$ `is_child_of` $\mathcal{C}$.

Une manière de comprendre la profondeur d'une $\mathcal{D}$-chronique $\mathcal{C}$ est de voir la profondeur comme le nombre d'opérateurs à appliquer à n'importe quel élément de $\mathcal{D}^\top$ pour obtenir $\mathcal{C}$ (cf. figure 4.13). C'est le théorème 3 qui motive les efforts de formalisation consentis dans cette partie. En effet, ce que donne ce théorème, c'est en réalité le facteur d'accroissement de l'approche « générer et compter » de la découverte complète de chroniques. Ce facteur d'accroissement c'est donc la profondeur. Chaque fois que l'on génère des $\mathcal{D}$-chroniques candidates à partir d'une $\mathcal{D}$-chronique fréquente de profondeur k, la profondeur des chroniques générées est $k + 1$.

4.2.6.2 Preuve de complétude

Pour prouver la complétude de `CDA` conformément à l'énoncé du problème de découverte complète de $\mathcal{D}$-chroniques établi en section 4.1.5, on utilise la notion de profondeur d'une $\mathcal{D}$-chronique pour démontrer le prédicat $P(k)$ ci-dessous par récurrence, sur la profondeur k.

Théorème 4
Le prédicat $P(k)$ *ci-dessous est vrai pour tout entier k supérieur ou égal à* 1 *:*

> $P(k)$ *: Si* $\mathcal{C}$ *est une* $\mathcal{D}$*-chronique fréquente de profondeur k, alors toutes les* $\mathcal{D}$*-chroniques parent* $\mathcal{C}'$ *de* $\mathcal{C}$ *seront ajoutées à l'ensemble* `Candidats` *à un moment donné de l'exécution de* `CDA`.

Preuve
On démontre le prédicat $P(k)$ *par récurrence sur k.*

$P(1)$. *Si* $\mathcal{C}$ *est une* $\mathcal{D}$*-chronique fréquente telle que* $prof(\mathcal{C}, \mathcal{D}) = 1$, *alors tous ses parents sont de profondeur* 0 *d'après le théorème 3. Comme l'ensemble* `Candidats` *est initialisé avec toutes les* $\mathcal{D}$*-chroniques qui sont les contraintes temporelles maximales de la base de contraintes, c'est-à-dire avec l'ensemble* $\mathcal{D}^\top$ *(qui regroupe les* $\mathcal{D}$*-chroniques de profondeur* 0*), toutes les chroniques parentes de* $\mathcal{C}$ *seront bien dans l'ensemble* `Candidats`, *ce qui prouve* $P(1)$.

$P(k+1)$. *Supposons que le prédicat* $P(k)$ *soit vrai pour un entier k tel que* $k \geq 1$. *Alors* $\mathcal{C}$ *est une* $\mathcal{D}$*-chronique fréquente de profondeur* $k+1$, *donc tous les parents* $\mathcal{C}'$ *seront des* $\mathcal{D}$*-chroniques de profondeur k (cf. théorème 3). Notons* $P' = \{\mathcal{C}'_1, \ldots, \mathcal{C}'_p\}$ *l'ensemble des* $\mathcal{D}$*-chroniques parent de* $\mathcal{C}$. *Comme* $\mathcal{C}$ *est fréquente, toutes les chroniques de l'ensemble* P' *sont fréquentes aussi, par monotonicité, et donc l'hypothèse «* $P(k)$ *est vrai » s'applique à chaque* $\mathcal{C}'_i$ *de* P'. *L'union des* $\mathcal{D}$*-chroniques parent de* $\mathcal{C}'_i$ *pour tout i constitue l'ensemble* P'' *des* $\mathcal{D}$*-chroniques « grand-parent » de* $\mathcal{C}$. *D'après* $P(k)$, *toutes les* $\mathcal{D}$*-chroniques de l'ensemble* P'' *seront ajoutées par* `CDA` *à l'ensemble* `Candidats`. *Donc pour chaque chronique* $\mathcal{C}''$ *de* P'', $\mathcal{C}''$ *étant fréquente également par monotonicité, la procédure* `générer_enfants` *(l. 20) sera appliquée à* $\mathcal{C}''$. *Donc chaque chronique* $\mathcal{C}'$ *de l'ensemble*

P' sera générée au moins une fois par la procédure `générer_enfants`*. La première fois que chaque $\mathcal{C}'$ de l'ensemble P' sera générée par* `générer_enfants`*, $\mathcal{C}'$ passera le test $\mathcal{C}' \notin$* `Traitées` *et sera ajoutée à l'ensemble* `Candidats`*, ce qui prouve $P(k+1)$.*

Finalement $P(1)$ est vrai et si $P(k)$ est vrai alors $P(k+1)$ est vrai, donc $P(k)$ est vrai pour tout k supérieur à 1.

Le théorème 5 prouve la complétude de l'algorithme `CDA`.

Théorème 5
Soient $\mathcal{S}$ une trace, $\mathcal{D}$ une base de contraintes et f_{seuil} une fréquence seuil. Soit $\mathcal{C}$ une $\mathcal{D}$-chronique fréquente qui soit minimale dans l'ensemble de toutes les $\mathcal{D}$-chroniques fréquentes dans $\mathcal{S}$. Alors la $\mathcal{D}$-chronique $\mathcal{C}$ sera présente dans l'ensemble `Fréquents` *retourné par l'algorithme* `CDA` *à la fin de son exécution.*

Preuve
D'après le théorème 4, toutes les chroniques parent de $\mathcal{C}$ seront présentes dans `Candidats` *à un moment de l'exécution de* `CDA`*. $\mathcal{C}$ sera donc générée et ajoutée à l'ensemble* `Candidats`*. $\mathcal{C}$ sera donc prise par la procédure* `prendre_candidat_suivant` *(l. 6) et sera nécessairement comptée dans la trace $\mathcal{S}$ (l. 12), car étant fréquente elle ne peut pas avoir de chronique ascendante non fréquente dans* `Non_Fréquents` *(cf. l. 8), et étant minimale elle ne peut pas avoir de chronique descendante fréquente dans* `Fréquents` *(cf. l. 11). Étant fréquente, $\mathcal{C}$ sera ajoutée par la procédure* `ajout_minimal` *en tant que chronique minimale de l'ensemble* `Fréquents` *(l. 14) et n'en sera jamais supprimée par cette même procédure* `ajout_minimal`*, toujours parce que $\mathcal{C}$ est minimale. En conséquence, $\mathcal{C}$ sera présente dans l'ensemble* `Fréquents` *à la fin de l'exécution de* `CDA`*.*

4.2.7 Estimation de la complexité

Dans cette section nous donnons un aperçu de la complexité temporelle de l'algorithme `CDA` en estimant la taille du graphe d'exploration de $\mathcal{D}$-chroniques (cf. figure 4.13). Nous aborderons le problème de la complexité en mémoire, liée notamment à l'ensemble `Traitées`, dans la section 4.5.4.

La taille du graphe d'exploration des chroniques dépend fortement de la base de contraintes temporelles $\mathcal{D}$. Prenons un épisode de longueur n, noté $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$. À partir de cet épisode, il est possible de construire en tout C_n^2 couples de types d'événement $(\varepsilon_i, \varepsilon_j)$. Supposons maintenant qu'il existe dans la base $\mathcal{D}$, un graphe de contraintes $\mathcal{G}_{ij}$ pour chacun de ces couples de types d'événement, et chacun des ces graphes contient p contraintes temporelles. Il est alors possible de créer $p^{C_n^2}$ $\mathcal{D}$-chroniques basées sur le seul épisode $\mathcal{E}$.

Le nombre d'épisodes de longueur n qu'il est possible de créer à partir de types d'événement de $\mathbb{E}$, est le nombre de n-combinaisons avec remises d'un ensemble à $|\mathbb{E}|$ éléments, c'est-à-dire $C^n_{|\mathbb{E}|+n-1}$. Soit $N(n)$ le nombre de $\mathcal{D}$-chroniques de longueur n existant dans une telle base $\mathcal{D}$. On a :

$$N(n) = C^n_{|\mathbb{E}|+n-1} \times p^{C_n^2} \tag{4.1}$$

Sur la formule 4.1 on lit deux facteurs. Le facteur $C^n_{|\mathbb{E}|+n-1}$ est lié aux épisodes des $\mathcal{D}$-chroniques. Ce facteur représente la complexité du problème de découverte d'épisodes parallèles de longueur n parmi $|\mathbb{E}|$ types d'événement. Le facteur $p^{C_n^2}$ est lié aux contraintes temporelles des $\mathcal{D}$-chroniques. On lit donc sur la formule 4.1 que la complexité du problème de découverte complète de $\mathcal{D}$-chroniques est la même que celle du problème de découverte d'épisodes parallèles à partir de

traces (Mannila et al. 1997) avec laquelle on combine le facteur de complexité propre aux $\mathcal{D}$-chroniques : $p^{C_n^2}$. Le cas de la découverte d'épisodes parallèles correspond au cas de la découverte de chroniques avec une seule et même contrainte possible pour chaque couple : $[-\infty, +\infty]$. On a donc pour ce problème de découverte d'épisodes parallèle $p = 1$; on retrouve bien la complexité de ce problème : $C^n_{|\mathbb{E}|+n-1}$.

Soit maintenant n_{max} la longueur de la plus longue $\mathcal{D}$-chronique fréquente minimale découverte par `CDA`. Il n'y aura alors jamais de chroniques de longueur plus grande que $n_{max}+1$ dans l'ensemble `Candidats`, et donc une borne supérieure de la taille N_{total} du graphe d'exploration est :

$$\begin{aligned} N_{total} &= N(2) + N(3) + \ldots + N(n_{max}+1) \\ &= O(N(n_{max}+1)) \\ &= O(C^{n_{max}+1}_{|\mathbb{E}|+n_{max}} \times p^{C^2_{n_{max}+1}}) \\ &= O(p^{C^2_{n_{max}}}) \\ &= O(p^{\frac{n_{max}\times(n_{max}+1)}{2}}). \end{aligned}$$

Finalement,

$$N_{total} = O(p^{n^2_{max}}). \tag{4.2}$$

La complexité affichée par la formule 4.2 donne un majorant du nombre total de chroniques qui devront être traitées par `CDA`. Dans la pratique, lors de l'exécution de l'algorithme, toutes les $\mathcal{D}$-chroniques ne seront pas générées. En effet, l'algorithme `CDA` ne parcourra que les branches qui sont fréquentes, les autres seront abandonnées. De plus, les bases de contraintes n'ont pas forcément de graphes de contraintes pour tous les couples de types d'événement, et tous les graphes ne contiennent pas nécessairement p contraintes. Il arrive souvent que plusieurs graphes de contraintes ne contiennent en réalité qu'une ou deux contraintes.

Ce qu'il faut retenir de cette section, c'est que le paramètre critique du problème de découverte complète de $\mathcal{D}$-chroniques est la longueur n_{max} de la chronique la plus longue qui sera découverte. De plus, la dépendance en n_{max} est exponentiellement proportionelle à n^2_{max}. Cela signifie que lors des mises en pratiques de la découverte complète de $\mathcal{D}$-chroniques, il faudra trouver des stratégies pour gérer les effets de cette complexité, notamment par le biais des contraintes utilisateur, par exemple pour permettre à l'analyste de limiter l'espace de recherche à une certaine longueur maximale n_{max}. Dans ce cas, le problème de découverte complète de chroniques devient alors : « extraire toutes les $\mathcal{D}$-chroniques fréquentes minimales dont la longueur est inférieure à n_{max} ».

4.3 Construction de la base de contraintes temporelles

Dans cette section, nous donnons trois stratégies pour construire la base de contraintes temporelles $\mathcal{D}$. Selon la stratégie utilisée, la base de contraintes $\mathcal{D}$ créée diffère, et donc l'ensemble des $\mathcal{D}$-chroniques découvertes diffère également. Selon la base de contraintes $\mathcal{D}$, l'ensemble des chroniques découvertes peut être le même que l'ensemble des $\mathcal{D}$-chroniques découvertes par la méthode de Duong (2001), ou bien peut être l'ensemble complet des $\mathcal{D}$-chroniques fréquentes minimales (cf. section 4.3.2), ou encore l'ensemble des épisodes hybrides fréquents minimaux (cf. section 4.3.3)

pour le problème de découverte d'épisodes hybrides à partir de séquences d'événements posé par Mannila et al. (1997).

4.3.1 Base de contraintes temporelles de Duong

Nous résumons ici la méthode de construction de base de contraintes expliquée et détaillée par Duong (2001). Cette base de contraintes est celle utilisée en entrée de l'algorithme de découverte de Duong pour générer les chroniques. Nous l'avons vu, cette base de contraintes n'est pas complète puisqu'elle ne stocke qu'une seule contrainte temporelle par couple de types d'événement. Nous résumons cependant sa méthode de construction ici, car d'une part il est intéressant de savoir comment l'algorithme `CDA` peut être utilisé pour retrouver les mêmes contraintes temporelles que Duong, et d'autre part parce que la méthode de construction de la base de contraintes complète s'inspire de la méthode de Duong.

La méthode de construction de la base de contraintes de Duong se déroule en quatre phases. Elle requiert d'avoir en entrée l'ensemble $\mathbb{E}$ des types d'événement qui apparaissent dans la trace $\mathcal{S}$, la trace $\mathcal{S}$ elle-même ($\mathbb{E}$ peut toujours être calculé à partir de $\mathcal{S}$), une fréquence seuil f_{seuil} et une note $\mathcal{Q}$, c'est-à-dire un nombre réel compris entre 0 et 1, dont nous verrons l'usage ci-après.

Premièrement, pour chaque couple $(\varepsilon, \varepsilon')$ de types d'événement de $\mathbb{E}$ tels que $\varepsilon \leq_{\mathbb{E}} \varepsilon'$, on construit l'ensemble complet de toutes les occurrences de l'épisode $\varepsilon\varepsilon'$, noté $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$, dans la trace $\mathcal{S}$, c'est-à-dire toutes les occurrences de la chronique $(\varepsilon\varepsilon', \{\})$ dans la trace $\mathcal{S}$. Par « toutes les occurrences », nous entendons l'ensemble reconnu par l'algorithme de reconnaissance de chroniques CRS, plus celles qui ne sont pas reconnues par CRS, c'est-à-dire l'ensemble des occurrences obtenu en combinant toutes les occurrences du type d'événement ε et toutes les occurrences du type d'événement ε'. Formellement, $\mathcal{O}^{all}_{\varepsilon\varepsilon'} = \{(\varepsilon, t)(\varepsilon', t'); (\varepsilon, t) \in \mathcal{S} \text{ et } (\varepsilon', t') \in \mathcal{S}\}$.

Exemple
Dans la trace $\mathcal{S}_0$ (cf. figure 4.11), $\mathcal{O}^{all}_{AB} = \{(A,1)(B,4), (A,1)(B,6), (A,5)(B,4), (A,5)(B,6)\}$.

Deuxièmement, on construit à partir de $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$ la liste, notée $\mathcal{A}_{\varepsilon\varepsilon'}$, des amplitudes des occurrences du couple $(\varepsilon, \varepsilon')$. Formellement, $\mathcal{A}_{\varepsilon\varepsilon'} = \langle t - t'; (\varepsilon, t)(\varepsilon', t') \in \mathcal{O}^{all}_{\varepsilon\varepsilon'} \rangle$.

Exemple
Dans la trace $\mathcal{S}_0$, $\mathcal{A}_{AB} = \langle 3, 5, -1, 1 \rangle$, donc $\mathcal{A}_{AB} = \langle -1, 1, 3, 5 \rangle$ une fois triée.

Troisièmement, on construit à partir de l'ensemble $\mathcal{A}_{\varepsilon\varepsilon'}$ trié, un ensemble de contraintes temporelles candidates en appliquant la note $\mathcal{Q}$, que nous appellerons « note minimale de couverture ». L'exemple ci-dessous explique comment utiliser la note $\mathcal{Q}$ et la trace $\mathcal{S}_0$ pour générer les contraintes temporelles candidates.

Exemple
Dans la trace $\mathcal{S}_0$, supposons que $\mathcal{Q} = 0.5$ (ou $\mathcal{Q} = 50\%$). Comme on a $|\mathcal{A}_{AB}| = 4$, le minimum de couverture de 0.5 signifie que les contraintes temporelles candidates devront couvrir $|\mathcal{A}_{AB}| \times \mathcal{Q} = 2$ occurrences de l'ensemble $\mathcal{O}^{all}_{AB}$. Pour cela, on fait glisser une fenêtre de largeur $|\mathcal{A}_{AB}| \times \mathcal{Q} - 1 = 1$ sur la liste $|\mathcal{A}_{AB}|$ triée. En faisant glisser une fenêtre de largeur 1 sur $|\mathcal{A}_{AB}|$, on obtient les intervalles suivants : $[-1, 1]$, $[1, 3]$, et $[3, 5]$. À partir des intervalles ainsi générés, on construit les trois contraintes temporelles candidates pour le couple (A, B) : $A[-1, 1]B$, $A[1, 3]B$ et $A[3, 5]B$. On est assuré que chacune de ces trois contraintes temporelles couvrira exactement deux occurrences de l'épisode AB dans la trace $\mathcal{S}_0$, car chaque élément de la liste $\mathcal{A}_{AB}$ représente exactement une occurrence de $\mathcal{O}^{all}_{AB}$.

Quatrièmement, parmi les contraintes candidates générées, un critère de sélection de contraintes permet de choisir celle qui sera retenue comme étant l'unique contrainte de la base $\mathcal{D}$ pour le couple (A, B). Ce critère de sélection de contraintes peut être :

- les contraintes dont l'amplitude est la plus faible, dans ce cas nos trois contraintes candidates pour le couple (A, B) seraient sélectionnées car elles ont toutes une amplitude de 2,
- les contraintes dont les bornes sont les moins éloignées de 0, auquel cas la contrainte $A[-1, 1]B$ serait sélectionnée,
- les contraintes dont les deux bornes sont du même signe et les plus proches de 0, auquel cas $A[1, 3]B$ serait sélectionnée.

Dans la pratique, ces trois critères de sélection sont combinés pour ne sélectionner qu'une seule contrainte temporelle par couple $(\varepsilon, \varepsilon')$.

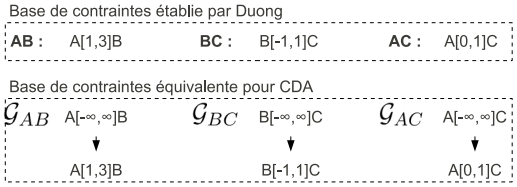


Figure 4.3 – Base de contraintes selon Duong pour la trace $\mathcal{S}_0$, $f_{seuil} = 2$ et $\mathcal{Q} = 0.5$ et sa base équivalente adaptée pour l'algorithme CDA

La figure 4.3 montre deux bases de contraintes temporelles. Celle du haut de la figure montre celle obtenue par le processus de construction ci-dessus à la trace $\mathcal{S}_0$. Toutefois, il existe une différence entre le processus de découverte de Duong et notre algorithme CDA. L'algorithme de Duong autorise les chroniques générées à n'avoir aucune contrainte pour certains couples de types d'événement de la chroniques, alors que CDA ne génère que des chroniques dites *pleines*, c'est-à-dire ayant des contraintes temporelles définies pour tous les couples. Ne pas poser de contraintes temporelles sur un couple $(\varepsilon, \varepsilon')$ revient à poser la contrainte temporelle $\varepsilon[-\infty, +\infty]$. Donc pour pouvoir retrouver les mêmes chroniques que Duong avec CDA, nous devons adapter la base de contraintes obtenue par la méthode ci-dessus en ajoutant pour chaque couple la contrainte temporelle $[-\infty, +\infty]$ en tant que contrainte parent. Cela donne la base de contraintes équivalentes du bas de la figure 4.3. On note que la base de contraintes créée ainsi est régulière, et donc que l'algorithme CDA peut s'appliquer à cette base de contraintes tout en étant complet.

Le deuxième intérêt de faire le lien entre la méthode de Duong et la découverte complète est que la formule 4.1 donne la taille de l'espace d'exploration du problème de découverte selon Duong, et donc un majorant de sa complexité temporelle. En effet, dans le cas de la découverte de Duong, le nombre de contraintes temporelles par graphe dans la base de contraintes est 2,c'est-à-dire $p = 2$. En conséquence, la complexité du problème de découverte de Duong est :

$$O(C^{n_{max}}_{q+n_{max}-1} \times 2^{n^2_{max}}).$$

Pour la découverte de chroniques à partir d'une trace, utiliser la méthode de Duong revient à « disqualifier » *a priori* certaines chroniques qui ne seront jamais générées du fait de l'incomplétude de la base de contraintes. De plus, le choix de l'unique contrainte temporelle pour chaque couple se fait automatiquement à partir d'une note $\mathcal{Q}$ et d'un critère de sélection, et ce choix peut paraître arbitraire aux yeux de l'analyste qui ne comprend pas pourquoi certaines contraintes potentiellement intéressantes sont disqualifiées de la découverte.

4.3.2 Construction de la base de contraintes temporelles complète

Afin de ne pas disqualifier certaines contraintes *a priori*, et donc de pouvoir générer toutes les $\mathcal{D}$-chroniques fréquentes, nous voulons laisser le choix à l'analyste d'utiliser en entrée de la découverte une base de contraintes complète, c'est-à-dire une base de contraintes contenant toutes les contraintes τ telles que $f_{CRS}(\tau) \geq f_{seuil}$. C'est l'objet de la méthode décrite dans cette section.

Pour cela, la méthode que nous présentons reprend la méthode de Duong (2001), mais diffère à partir de l'étape trois. Une fois créée la liste $\mathcal{A}_{\varepsilon\varepsilon'}$ pour le couple $(\varepsilon, \varepsilon')$, l'idée est de faire glisser sur la liste $\mathcal{A}_{\varepsilon\varepsilon'}$ triée une fenêtre de largeur $f_{seuil} - 1$ pour obtenir toutes les contraintes temporelles qui ont exactement f_{seuil} occurrences dans $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$, puis une fenêtre de largeur f_{seuil} pour obtenir toutes les contraintes temporelles qui ont exactement $f_{seuil} + 1$ occurrences dans $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$, puis une fenêtre de largeur $f_{seuil} + 1$ pour obtenir toutes les contraintes temporelles qui ont exactement $f_{seuil} + 2$ occurrences dans $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$, et ainsi de suite jusqu'à une fenêtre de largeur $|\mathcal{A}_{\varepsilon\varepsilon'}|$. L'ensemble de toutes les contraintes temporelles pour le couple $(\varepsilon, \varepsilon')$ obtenues par cette méthode doit ensuite être organisé en un graphe de contraintes temporelles.

Exemple
Appliquée à la trace $\mathcal{S}_0$ de la figure 4.11 pour le couple (A, B) avec $f_{seuil} = 2$, cette procédure donne :

En glissant une fenêtre de largeur 2 sur $\mathcal{A}_{AB}$: $A[-1,1]B, A[1,3]B, A[3,5]B$
fenêtre de largeur 3 : $A[-1,3]B, A[1,5]B$
fenêtre de largeur 4 : $(= |\mathcal{A}_{AB}|)$: $A[-1,5]B$

Organisées en graphe de contraintes, ces six contraintes temporelles donnent le graphe de contraintes $\mathcal{G}_{AB}$ montré sur la figure 4.12.

L'algorithme `CCDC` (cf. algorithme 2) construit la base de contraintes en appliquant cette méthode.

Algorithme 2 `CCDC` : Complete Constraint-Database Construction.

Entrées: $\mathcal{S}$; $\mathbb{E}$; f_{seuil}
Sorties: $\mathcal{D}$

1: $\mathcal{D} \leftarrow \emptyset$
2: **pour** chaque $(\varepsilon, \varepsilon') \in \mathbb{E} \times \mathbb{E}$ **faire**
3: **si** $\varepsilon >_{\mathbb{E}} \varepsilon'$ **alors**
4: continuer la boucle à la prochaine itération (l. 2)
5: **fin si**
6: $\mathcal{O}^{all}_{\varepsilon\varepsilon'} \leftarrow \{\langle(\varepsilon, t)(\varepsilon', t')\rangle | (\varepsilon, t) \in \mathcal{S}\ et\ (\varepsilon', t') \in \mathcal{S}\ et\ (\varepsilon, t) \neq (\varepsilon', t')\}$
7: $\mathcal{A}_{\varepsilon\varepsilon'} \leftarrow$`trier` $(\{(t' - t) | \langle(\varepsilon, t)(\varepsilon', t')\rangle \in \mathcal{O}^{all}_{\varepsilon\varepsilon'}\})$
8: **pour** $k = f_{seuil}$ à $|\mathcal{A}_{\varepsilon\varepsilon'}|$ **faire**
9: $\mathcal{K}^k_{\varepsilon\varepsilon'} \leftarrow \{\varepsilon[\mathcal{A}_{\varepsilon\varepsilon'}[i], \mathcal{A}_{\varepsilon\varepsilon'}[i + k - 1]]\varepsilon' | 0 \leq i \leq |\mathcal{A}_{\varepsilon\varepsilon'}| - k + 1\}$
10: **fin pour**
11: $\mathcal{K}_{\varepsilon\varepsilon'} \leftarrow \mathcal{K}^{f_{seuil}}_{\varepsilon\varepsilon'} \cup \mathcal{K}^{f_{seuil}+1}_{\varepsilon\varepsilon'} \cup \ldots \cup \mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|}_{\varepsilon\varepsilon'}$
12: Transformer $\mathcal{K}_{\varepsilon\varepsilon'}$ en un graphe de contraintes temporelles $\mathcal{G}_{\varepsilon\varepsilon'}$
13: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{G}_{\varepsilon\varepsilon'}$
14: **fin pour**
15: **retourner** $\mathcal{D}$

Avec cette méthode, toutes les contraintes temporelles contenues dans le graphe de contraintes $\mathcal{G}_{\varepsilon\varepsilon'}$ seront assurées d'avoir au moins f_{seuil} occurrences dans l'ensemble $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$. Cependant, cela n'assure pas toutes les contraintes temporelles de $\mathcal{G}_{\varepsilon\varepsilon'}$ d'être fréquentes pour la mesure de fréquence f_{CRS}. Si on définit la nouvelle mesure de fréquence f_{all} d'une contrainte temporelle τ du graphe $\mathcal{G}_{\varepsilon\varepsilon'}$ comme étant le nombre d'occurrences qu'elle a dans l'ensemble $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$, alors nécessairement $f_{CRS}(\tau) \leq f_{all}(\tau)$, car $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$ contient toutes les occurrences possibles pour l'épisode $\varepsilon\varepsilon'$, donc contient *a fortiori* toutes les occurrences possibles pour la contrainte temporelle τ, et donc *a fortiori* pour les occurrences de la contraintes temporelle τ qui sont reconnues par l'algorithme CRS. En conséquence, il peut y avoir dans le graphe de contraintes $\mathcal{G}_{\varepsilon\varepsilon'}$ des contraintes temporelles qui ne soient pas fréquentes pour f_{CRS}, mais toutes les contraintes fréquentes pour f_{CRS} y sont. On peut tenter de supprimer toutes les contraintes temporelles non fréquentes du graphe $\mathcal{G}_{\varepsilon\varepsilon'}$ en les comptant une à une, mais dans la pratique on constate que peu de contraintes sont dans ce cas et que ce n'est pas avantageux de supprimer ces contraintes.

Cette méthode s'applique très bien aux traces « courtes », comme notre trace exemple $\mathcal{S}_0$. Pour des traces plus longues, ce que nous seront amenés à rencontrer avec les traces d'activité, il n'est pas concevable d'appliquer cette méthode complète telle quelle. En effet, supposons qu'une trace réelle contienne 100 occurrences du type d'événement A et 100 occurrences du type d'événement B. Cela ferait alors potentiellement $100 \times 100 = 10000$ contraintes temporelles pour le couple (A, B) dans la base de contraintes. Le temps d'élaboration du graphe $\mathcal{G}_{AB}$ serait d'une part très long, et la génération de chroniques contenant le couple (A, B) serait très coûteuse aussi. Pour remédier à cela, on peut introduire la contrainte de fenêtre win (de l'anglais *window*), limitant l'étalement temporel des occurrences. Par exemple, si $win = 10$, alors toutes les occurrences $\langle (A, t_A)(B, t_B) \rangle$ de l'épisode AB telles que $|t_A - t_B| > 10$ sont supprimées de $\mathcal{O}^{all}_{AB}$, supprimant ainsi pour les longues traces s'étalant sur de très nombreuses unités de temps une très grande partie des contraintes temporelles du graphe.

4.3.3 Construction de la base de contraintes temporelles hybride

L'analyste peut ne pas être intéressé par la pleine expressivité offerte par le formalisme des chroniques et vouloir découvrir des motifs temporels dans la trace qui contiennent moins d'informations. Les *épisodes hybrides*, présentés par Mannila et al. (1997), sont de tels motifs temporels. Les épisodes hybrides sont des épisodes parallèles auxquels s'ajoute un ensemble de contraintes d'ordre entre les types d'événement de l'épisode. Mannila et al. argumentent que les épisodes hybrides sont des motifs temporels qu'il est intéressant de découvrir dans de nombreuses données séquentielles, comme les journaux d'alarmes dans les réseaux de télécommunication, mais ne proposent pas de méthode propre pour la découverte de tels motifs.

On peut remarquer qu'un épisode hybride est en réalité une chronique particulière. En effet, un épisode hybride est une chronique constituée des mêmes types d'événement telle que pour chaque contrainte d'ordre de l'épisode hybride (cf. figure 4.4), il existe une contrainte temporelle $\varepsilon[0, +\infty]\varepsilon'$ dans la chronique. Pour chaque couple $(\varepsilon, \varepsilon')$ de l'épisode hybride n'ayant pas de contrainte d'ordre, il existe dans la chronique une contrainte temporelle $\varepsilon[-\infty, +\infty]\varepsilon'$.

Exemple
La figure 4.4 montre sur un exemple l'équivalence qu'il y a entre un épisode hybride et une chronique. L'épisode hybride est représenté sur la gauche. Il est constitué de l'épisode parallèle AABC et des quatre contraintes d'ordre suivantes :

- *B doit suivre A,*

- *C doit suivre A,*
- *D doit suivre B,*
- *D doit suivre C.*

Autrement dit, cet épisode hybride s'interprète comme suit : « A, suivi de B et C dans n'importe quel ordre, suivis de D ». Cet épisode hybride se représente par la chronique suivante :

$$(ABCD, \{A[0,+\infty]B, A[0,+\infty]C, A[-\infty,+\infty]D, B[0,+\infty]D, B[-\infty,+\infty]C, C[0,+\infty]D\}).$$

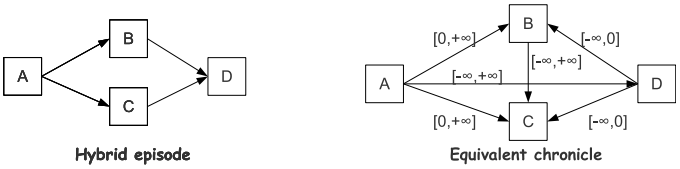


Figure 4.4 – Un épisode hybride et sa chronique équivalente

L'exemple ci-dessus, montre que pour ce qui est de la représentation graphique du même motif temporel, la représentation sous forme d'épisode hybride est bien plus lisible pour l'humain, c'est d'ailleurs l'intérêt des épisodes hybrides. En revanche, notre algorithme ne supportera en interne que la représentation de ce motif sous forme de chronique. C'est pourquoi dans la suite nous représenterons graphiquement chaque chronique équivalente d'un épisode hybride sous la forme de cet épisode hybride, même si en interne il s'agira toujours d'une chronique.

À partir de ce constat d'équivalence entre épisode hybride et chronique, il est très simple d'établir la base de contraintes temporelles qui permettra la découverte complète d'épisodes hybrides dans une trace en utilisant notre algorithme CDA. La base de contraintes hybride est construite ainsi :

1. pour chaque couple de types d'événement $(\varepsilon, \varepsilon')$ tel que $\varepsilon <_{\mathbb{E}} \varepsilon'$, construire un graphe de contraintes composé des trois contraintes suivantes : $[-\infty, +\infty]$, $[-\infty, 0]$ et $[0, +\infty]$,
2. pour chaque couple de types d'événement $(\varepsilon, \varepsilon)$, construire un graphe de contraintes composé de l'unique contrainte $[0, +\infty]$.

Le paramètre de complexité p de la formule 4.1 vaut donc 3 et la complexité du problème de découverte complète d'épisodes hybrides est $O(C^{n_{max}}_{q+n_{max}-1} \times 3^{n^2_{max}})$.

Exemple
La figure 4.5 montre la base de contraintes, notée $\mathcal{D}_{hyb}$, obtenue par le processus de construction de base de contraintes hybrides ci-dessus. pour l'ensemble $\mathbb{E}_0$ des types d'événement de la trace $\mathcal{S}_0$, c'est-à-dire $\mathbb{E}_0 = \{A, B, C\}$.

4.4 Permettre le guidage de CDA par l'analyste

L'algorithme CDA présenté précédemment permet la découverte complète de $\mathcal{D}$-chroniques par l'analyste à partir d'une trace $\mathcal{S}$, d'une base de contraintes $\mathcal{D}$ et d'une fréquence minimale f_{seuil}. L'algorithme CDA produit des $\mathcal{D}$-chroniques en temps réel accessibles à l'analyste grâce à l'ensemble `Fréquents`, qui est toujours en perpétuelle élaboration. Cependant, au vu de la complexité du

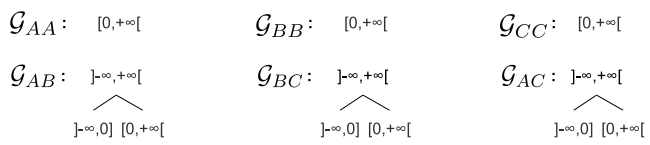


Figure 4.5 – Base de contraintes $\mathcal{D}_{hyb}$ pour la découverte complète d'épisodes hybrides

problème de découverte complète de $\mathcal{D}$-chroniques, il n'est pas raisonnable d'attendre la fin du processus de découverte complète de $\mathcal{D}$-chroniques et l'analyste doit se satisfaire des premiers résultats présents dans l'ensemble `Fréquents`. C'est pourquoi, il est important que l'analyste puisse avoir un fort contrôle sur le processus de découverte. En effet, un fort contrôle par l'analyste permettra à `CDA` de converger plus rapidement vers les solutions les plus intéressantes pour l'analyste sans attendre la fin de l'exécution de `CDA`.

Dans cette section, nous expliquons comment l'algorithme `CDA` présenté précédemment peut être légèrement adapté et utilisé par l'analyste pour que ce dernier ait le contrôle nécessaire sur le déroulement de son exécution. Le premier type de contrôle que nous abordons est l'ajout de contraintes utilisateur à l'exécution de `CDA` ; le deuxième type de contrôle est le choix de l'heuristique de parcours des $\mathcal{D}$-chroniques candidates.

4.4.1 Introduction de contraintes utilisateur

Une contrainte utilisateur est un paramètre d'entrée donné par l'analyste qui pose une condition d'acceptabilité sur les $\mathcal{D}$-chroniques retournées par la machine. Nous devrions appeler ces contraintes des « *contraintes analyste* » car nous avons choisi de désigner l'utilisateur de l'algorithme par *analyste*, mais l'expression *contrainte utilisateur* étant admise par la communauté de la fouille de données, nous la garderons dans la suite. De nombreux travaux existent sur l'introduction de contraintes utilisateur dans un processus de fouille de séquences d'événements telles que les traces (cf. section 3.5). Il n'existe pas dans la littérature scientifique de travaux connus qui étudient l'ensemble des contraintes utilisateur qu'il est possible de spécifier sur des chroniques, mais il est très simple d'adapter les contraintes existantes pour les épisodes en parallèle et en série.

Nous avons vu en section 3.5 que les contraintes anti-monotones étaient les plus simples et les plus avantageuses à prendre en compte dans la découverte d'épisodes car elles peuvent être poussées à chaque étape de la génération d'épisodes et réduire ainsi à chaque étape la taille de l'espace des motifs candidats. Dans le cas des $\mathcal{D}$-chroniques, la relation d'ordre que nous suivons lors de la génération de $\mathcal{D}$-chroniques candidates à partir de $\mathcal{D}$-chroniques fréquentes est la relation de descendance « $\preceq$ », et non plus la relation d'inclusion d'épisode « $\subseteq$ ». Ainsi, nous générons des $\mathcal{D}$-chroniques candidates $\mathcal{C}'$ à partir d'une $\mathcal{D}$-chronique fréquente $\mathcal{C}$ telle que $\mathcal{C}' \preceq \mathcal{C}$, alors que la découverte d'épisodes à partir de traces consiste à générer des épisodes candidats ε' à partir d'un épisode fréquent ε tel que $\varepsilon' \supseteq \varepsilon$. Ainsi, les contraintes anti-monotones appliquées à la découverte de $\mathcal{D}$-chroniques deviennent des contraintes « monotones » par le simple fait que l'accroissement de motifs lors de la génération respecte l'ordre de droite à gauche pour la relation $\supseteq$ et de gauche à droite pour notre relation $\preceq$. Ainsi les contraintes avantageuses pour la découverte de $\mathcal{D}$-chroniques sont les contraintes « monotones » pour la relation « $\preceq$ ».

La première des contraintes utilisateur, déjà rendue disponible par l'algorithme `CDA` tel que présenté précédemment, est la contrainte de fréquence minimale f_{seuil}. Cette contrainte est habituelle pour un algorithme de fouille de données car elle est souvent la contrainte minimale de toute requête de fouille. De plus elle est monotone pour la relation $\preceq$, cela a été démontré par Dousson

& Duong (1999). La contrainte f_{seuil} est aussi extrêmement coûteuse à vérifier, car elle nécessite souvent, comme dans le cas de l'algorithme de comptage CRS, d'effectuer pour chaque candidat une passe sur l'ensemble des données, en l'occurrence sur la trace.

Nous proposons d'adapter l'algorithme CDA (cf. algorithme 3) en modifiant la ligne 22 de telle sorte que pour chaque chronique candidate enfant générée $\mathcal{C}'$, on vérifie aussi si cette candidate vérifie les contraintes monotones, c'est-à-dire en remplaçant la condition :

$$\mathcal{C}' \notin \texttt{Traitées}$$

par :

$$\texttt{satisfait\_contraintes\_monotones}\ (\mathcal{C}')\ \&\&\ \mathcal{C}' \notin \texttt{Traitées},$$

où la procédure `satisfait_contraintes_monotones` $(\mathcal{C}')$ renvoie vrai si toutes les contraintes monotones posées par l'analyste sont vérifiées. Comme nous le verrons dans les sous-sections qui viennent, vérifier si une chronique $\mathcal{C}'$ satisfait une contrainte autre que f_{seuil} est souvent assez direct, très rapide, et nécessite la chronique $\mathcal{C}'$ seule. Au contraire, vérifier la contrainte f_{seuil} nécessite l'ensemble de la trace $\mathcal{S}$, ce qui est très coûteux. Afin de vérifier les contraintes les moins coûteuses d'abord, la vérification de ces contraintes monotones autres que f_{seuil} doit se faire pour chaque chronique candidate avant la vérification de f_{seuil}. C'est pourquoi nous avons choisi d'introduire la vérification des contraintes monotones comme condition d'ajout des chroniques générées à l'ensemble `Candidats` (l. 22) des chroniques qui seront potentiellement comptées.

4.4.1.1 La contrainte *win*

La contrainte *win*, dite « contrainte de fenêtre englobante », contraint les chroniques découvertes à ne pas s'étaler trop temporellement. Une chronique $\mathcal{C}$ satisfait la contrainte *win* dans la trace $\mathcal{S}$ si chaque contrainte temporelle $\varepsilon_i[\tau_{ij}^-, \tau_{ij}^+]\varepsilon_j$ de la chronique $\mathcal{C}$ est incluse dans l'intervalle $[-win, +win]$. Une chronique $\mathcal{C}$ vérifiant une telle contrainte est assurée d'avoir chacune de ses occurrences incluses dans une fenêtre temporelle de largeur *win*. En effet, prenons une occurrence $(\varepsilon_1, t_1)\dots(\varepsilon_n, t_n)$ de la chronique $\mathcal{C}$ dans la trace $\mathcal{S}$. Comme $[\tau_{ij}^-, \tau_{ij}^+]$ est inclus dans l'intervalle $[-win, +win]$, on a $-win \leq \tau_{ij}^- \leq t_i - t_j \leq \tau_{ij}^+ \leq +win$ et donc $|t_i - t_j| \leq win$. La contrainte *win* n'est pas monotone, mais il est cependant très simple de la prendre en compte grâce à la base de contraintes $\mathcal{D}$. En effet, il suffit de supprimer de chaque graphe de la base de contraintes $\mathcal{D}$ toutes les contraintes temporelles τ qui ne sont pas incluses dans $[-win, +win]$. Pour les contraintes temporelles ayant des bornes infinies, comme les bases de contraintes hybrides et les bases de contraintes de Duong adaptées à CDA, il est possible d'intégrer la contrainte *win* dans la base de contraintes en remplaçant toutes les bornes « $-\infty$ » par « $-win$ » et toutes les bornes « $+\infty$ » par « $+win$ ».

La contrainte de fenêtre englobante est très pratique, voire nécessaire, lorsqu'il s'agit de découvrir des chroniques dans des traces comportant beaucoup d'événements et s'étalant temporellement sur plusieurs minutes ou plusieurs heures et que l'on souhaite découvrir des chroniques de l'ordre de quelques secondes.

On peut également intégrer une contrainte de fenêtre englobante directement au processus de comptage des chroniques de telle sorte que l'algorithme de reconnaissance des occurrences d'une chronique $\mathcal{C}$ dans une trace $\mathcal{S}$, c'est-à-dire l'algorithme CRS dans notre cas, ne reconnaisse que les occurrences à l'intérieur d'une fenêtre temporelle de largeur *win*. Pour cela, il suffit de modifier l'algorithme CRS de façon à disqualifier, lors du parcours chronologique des événements de la trace, les occurrences en cours de reconnaissance qui s'étalent sur une durée supérieure à

win. Pour appliquer une contrainte de fenêtre englobante à l'algorithme de Duong (2001), il est plus simple de procéder de cette manière que d'agir sur la base de contraintes, car comme nous l'avons vu en section 4.3.1, la découverte selon Duong (2001) génère des chroniques qui ne sont pas nécessairement *pleines*, c'est-à-dire qui n'ont pas nécessairement une contrainte définie pour chaque couple de types d'événement. C'est pourquoi les mesures comparatives entre l'algorithme de Duong et `CDA`, présentées dans l'annexe A et discutées lors de l'évaluation de notre algorithme (cf. section 4.5), ont exceptionnellement été obtenues par cette deuxième méthode. Dans toutes les autres mesures sur `CDA` et dans la plate-forme `Scheme Emerger` qui implémente `CDA`, la contrainte *win* a été appliquée en agissant sur la base de contraintes et non sur l'algorithme de comptage.

4.4.1.2 La contrainte n_{sup}

La contrainte n_{sup}, dite de « longueur maximale », contraint les chroniques découvertes à ne pas être trop longues. On dit que la chronique $\mathcal{C}$ satisfait la contrainte de longueur maximale n_{sup} si $|\mathcal{C}| \leq n_{sup}$, c'est-à-dire si $|\mathcal{E}| \leq n_{sup}$ (en notant $\mathcal{C} = (\mathcal{E}, \mathcal{T})$). La contrainte n_{sup} est monotone pour la relation « $\preceq$ », c'est-à-dire si $\mathcal{C} \preceq \mathcal{C}'$ et si $\mathcal{C}$ satisfait la contrainte de longueur maximale n_{sup} alors $\mathcal{C}'$ satisfait la contrainte de longueur maximale n_{sup}. Vérifier la satisfiabilité de $\mathcal{C}$ pour la contrainte de longueur maximale n_{sup} est immédiat et ne requiert pas de passe sur la trace $\mathcal{S}$. Vérifier cette contrainte peut donc s'effectuer à l'intérieur de la procédure `satisfait_contraintes_monotones`. Comme la complexité de la découverte complète de chroniques est exponentiellement dépendante en n_{max}^2, la contrainte n_{sup} est d'une importance capitale pour l'analyste car elle permet de faire en sorte que le processus de découverte ne « s'englue » pas dans la génération de chroniques trop longues (cf. section 4.5.5).

4.4.1.3 La contrainte d'ascendance $\mathcal{C}^\top$

La contrainte d'ascendance permet à l'analyste de s'assurer que les chroniques découvertes comporteront certains types d'événement désirés et certaines contraintes temporelles. On dit qu'une chronique candidate $\mathcal{C}$ respecte la contrainte d'ascendance $\mathcal{C}^\top$ si $\mathcal{C} \preceq \mathcal{C}^\top$, c'est-à-dire si $\mathcal{C}^\top$ est une $\mathcal{D}$-chronique ascendante de $\mathcal{C}$. Par exemple, si l'analyste souhaite que le type d'événement A et le type d'événement B apparaissent dans les chroniques découvertes et que les chroniques découvertes respectent la contrainte temporelle $C[3,5]D$ alors cela revient à poser la contrainte d'ascendance $\mathcal{C}^\top = (ABCD, \{C[3,5]D\})$. Théoriquement, une seule contrainte d'ascendance à la fois peut être posée, car poser les contraintes d'ascendance $\mathcal{C}_1^\top$ et $\mathcal{C}_2^\top$ reviendrait à poser la seule contrainte d'ascendance $\mathcal{C}^\top$ où $\mathcal{C}^\top$ est « l'intersection » des chroniques $\mathcal{C}_1^\top$ et $\mathcal{C}_2^\top$, où l'intersection de chroniques serait définie comme l'union des épisodes et l'intersection de leurs contraintes temporelles.

Exemple
Poser en même temps les contraintes d'ascendance $\mathcal{C}_1^\top = (ACD, \{C[0,5]D\})$ *et* $\mathcal{C}_2^\top = (BCD, \{C[3,8]D\})$ *reviendrait à poser l'unique contrainte d'ascendance* $\mathcal{C}^\top = (ABCD, \{C[3,5]D\})$.

Poser en même temps les contraintes d'ascendance $\mathcal{C}_1^\top$ *et* $\mathcal{C}_3^\top = (BCD, \{B[3,8]D\})$ *reviendrait à poser l'unique contrainte d'ascendance* $\mathcal{C}^\top = (ABCD, \{B[3,8]D, C[0,5]D\})$.

La contrainte d'ascendance n'est pas monotone, elle est anti-monotone, c'est-à-dire si $\mathcal{C}' \preceq \mathcal{C}$ et si $\mathcal{C}$ satisfait la contrainte d'ascendance $\mathcal{C}^\top$ alors $\mathcal{C}'$ satisfait également la contrainte d'ascendance $\mathcal{C}^\top$. On ne peut donc pas prendre en compte cette contrainte d'ascendance dans la procédure `satisfait_contraintes_monotones`. En revanche, il existe tout de même un moyen très efficace et très direct de prendre en compte cette contrainte. En effet, les $\mathcal{D}$-chroniques candidates $\mathcal{C}'$ étant

générées à partir de leurs chroniques fréquentes ascendantes $\mathcal{C}$, il suffit d'initialiser l'ensemble `Candidats` avec la $\mathcal{D}$-chronique $\mathcal{C}^\top$, c'est-à-dire en remplaçant la ligne `Candidats` $\leftarrow \mathcal{D}^\top$ par `Candidats` $\leftarrow \mathcal{C}^\top$ (l. 4 de l'algorithme 3).

4.4.1.4 La contrainte de non-ascendance $\mathcal{C}^\neg$

La contrainte de non ascendance permet à l'analyste de spécifier un ensemble de types d'événement et de contraintes temporelles qu'il ne souhaite pas retrouver dans les chroniques découvertes. La contrainte de non-ascendance se spécifie par le moyen d'une liste $\mathcal{C}^\neg$ de chroniques. Ensuite, on dit que la $\mathcal{D}$-chronique $\mathcal{C}$ vérifie la contrainte de non-ascendance $\mathcal{C}^\neg$ si pour toute chronique $\mathcal{C}_i$ de la liste $\mathcal{C}^\neg$, $\mathcal{C} \succeq \mathcal{C}_i$, c'est-à-dire si la $\mathcal{D}$-chronique $\mathcal{C}$ n'est la descendante d'aucune chronique de la liste $\mathcal{C}^\neg$.

Par exemple, si l'analyste souhaite découvrir des chroniques qui ne contiennent pas le type d'événement A, le type d'événement B et qui ne laissent pas la possibilité au type d'événement D de suivre C, alors l'analyste posera la contrainte de non-ascendance $\mathcal{C}^\neg = \{(A, \{\}), (B, \{\}), (CD, \{C[0, +\infty]D\})\}$.

La contrainte de non-ascendance est monotone et peut donc être prise efficacement par la méthode `satisfait_contraintes_monotones`.

4.4.1.5 Autres contraintes utilisateur

Les contraintes présentées ci-dessus sont les plus utiles à l'analyste pour avoir un contrôle fort sur l'exécution du processus de découverte de connaissances. Nous n'avons pas cherché à en implémenter davantage pour l'heure. Cependant, tout autre contrainte monotone ne requiérant pas de passe sur la trace $\mathcal{S}$ peut très facilement être prise en compte par ajout à la procédure `satisfait_contraintes_monotones`. Il est possible de prendre en compte toute autre contrainte utilisateur qui ne soit ni monotone ni anti-monotone de manière naïve en testant la satisfiabilité de la $\mathcal{D}$-chronique candidate $\mathcal{C}$ avant de compter sa fréquence dans la trace $\mathcal{S}$ (l. 12).

4.4.2 Heuristiques de parcours du graphe d'exploration

Un autre moyen laissé à l'analyste pour agir sur le processus de découverte de $\mathcal{D}$-chroniques est d'intervenir sur l'ordre dans lequel les $\mathcal{D}$-chroniques candidates sont sélectionnées dans l'ensemble `Candidats` par la procédure `prendre_candidat_suivant`. Lors de la conception de l'algorithme `CDA`, nous avons décidé de stocker l'ensemble des $\mathcal{D}$-chroniques candidates déjà traitées, c'est-à-dire dont la fréquence a été testée dans la trace $\mathcal{S}$ lors d'une itération de la boucle des lignes 5 à 26, dans l'ensemble `Traitées`. Grâce à cet ensemble `Traitées`, nous pouvons ainsi vérifier que chaque nouvelle $\mathcal{D}$-chronique générée n'a pas déjà été traitée et ainsi assurer la terminaison de `CDA`. Une autre solution pour éviter de stocker l'ensemble des $\mathcal{D}$-chroniques déjà traitées serait de définir un ordre total, noté $\preceq_{total}$ sur l'ensemble des chroniques et de s'assurer que les $\mathcal{D}$-chroniques sont traitées dans l'ordre. Ainsi lorsque les $\mathcal{D}$-chroniques enfant $\mathcal{C}'$ sont générées à partir de la $\mathcal{D}$-chronique fréquente $\mathcal{C}$ (l. 20) il suffit de générer uniquement les $\mathcal{D}$-chroniques $\mathcal{C}'$ telles que $\mathcal{C} \preceq_{total} \mathcal{C}'$ et de les ajouter à l'ensemble `Candidats`, si elles n'y sont toutefois pas déjà présentes, sans avoir à s'inquiéter de savoir si certaines de ces chroniques enfant $\mathcal{C}'$ ont déjà été ajoutées puis supprimées antérieurement de l'ensemble `Candidats`. Une telle stratégie aurait économisé la mémoire prise par l'ensemble `Traitées`. De plus une telle stratégie est possible puisqu'on peut facilement définir un ordre total sur les $\mathcal{D}$-chroniques, en notant $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ et $\mathcal{C}' = (\mathcal{E}', \mathcal{T}')$, par « $\mathcal{C} \preceq_{total} \mathcal{C}'$ » si et seulement si :

$\mathcal{E} <_{\mathbb{E}} \mathcal{E}'$,
ou $\mathcal{E} =_{\mathbb{E}} \mathcal{E}'$ et, en notant $\mathcal{E} = \mathcal{E}' = \varepsilon_1 \dots \varepsilon_n$, on a $[\tau_{12}^-, \tau_{12}^+] <_{c^{tes}} [\tau_{12}^{-\prime}, \tau_{12}^{+\prime}]$,
ou $[\tau_{12}^-, \tau_{12}^+] =_{c^{tes}} [\tau_{12}^{-\prime}, \tau_{12}^{+\prime}]$ et $[\tau_{13}^-, \tau_{13}^+] <_{c^{tes}} [\tau_{13}^{-\prime}, \tau_{13}^{+\prime}]$,
ou $[\tau_{13}^-, \tau_{13}^+] =_{c^{tes}} [\tau_{13}^{-\prime}, \tau_{13}^{+\prime}]$ et $[\tau_{14}^-, \tau_{14}^+] <_{c^{tes}} [\tau_{14}^{-\prime}, \tau_{14}^{+\prime}]$,
$\vdots$
ou $[\tau_{n-1,n}^-, \tau_{n-1,n}^+] \leq_{c^{tes}} [\tau_{n-1,n}^{-\prime}, \tau_{n-1,n}^{+\prime}]$.

où « $\leq_{\mathbb{E}}$ » désigne dans les lignes ci-dessus la relation d'ordre lexicographique définie sur $\mathbb{E}^*$ (l'ensemble des épisodes construits sur $\mathbb{E}$) par extension de la relation « $\leq_{\mathbb{E}}$ » que nous avions définie sur $\mathbb{E}$, et « $<_{c^{tes}}$ » est la relation d'ordre sur les intervalles définie par $[a, b] \leq_{c^{tes}} [c, d] \Leftrightarrow a < c$, ou $a = c$ et $b \leq d$.

Exemple
Soient $\mathcal{C} = (ABC, \{A[2,3]B, A[0,3]C, B[-2,0]C\})$, $\mathcal{C}' = (ABC, \{A[2,3]B, A[1,4]C, B[-1,1]C\})$ *et* $\mathcal{C}'' = (ABC, \{A[2,3]B\})$. *On a* $\mathcal{C}'' <_{c^{tes}} \mathcal{C} <_{c^{tes}} \mathcal{C}'$.

Avec une relation d'ordre total sur les chroniques telle que « $\preceq_{total}$ » il est possible de se passer de stocker toutes les chroniques déjà traitées. En revanche, si l'on souhaite avoir le choix de traiter les $\mathcal{D}$-chroniques candidates dans un ordre non total ou dépendant du contexte, c'est-à-dire si on souhaite implémenter la procédure `prendre_candidat_suivant` (l. 6) autrement que par la relation « $\preceq_{total}$ », alors il faudra stocker toutes les $\mathcal{D}$-chroniques déjà traitées. Les autres stratégies que nous avons implémentées pour la sélection par la procédure `prendre_candidat_suivant` de la prochaine $\mathcal{D}$-chronique à traiter sont :

LIFO (Last In First Out) la dernière $\mathcal{D}$-chronique ajoutée à l'ensemble `Candidats` est choisie en premier,

FIFO (First In First Out) la plus ancienne $\mathcal{D}$-chronique ajoutée à l'ensemble `Candidats` est choisie en premier,

Random la prochaine $\mathcal{D}$-chronique choisie de façon aléatoire.

Si l'on choisit d'utiliser l'une des trois stratégies ci-dessus, il faut utiliser l'ensemble `Traitées`, mais si on utilise la stratégie « $\preceq_{total}$ », alors ce n'est pas la peine. Lors de l'évaluation de l'algorithme `CDA` (cf. section 4.5), nous montrons que dans les 60 à 120 premières secondes d'exécution de `CDA`, c'est-à-dire pour la fourchette de temps correspondant à une itération du processus interactif de découverte temps réel, les effets de l'ensemble `Traitées` ne se font pas sentir par l'analyste en terme de performance.

Avec cette architecture d'algorithme pour `CDA`, la stratégie de sélection de la prochaine $\mathcal{D}$-chronique utilisée par la procédure `prendre_candidat_suivant` est appelée « heuristique », car cette stratégie régit l'ordre dans lequel le graphe d'exploration de $\mathcal{D}$-chroniques candidates (cf. figure 4.13) est parcouru. Pour l'heure nous n'avons implémenté que les quatre stratégies ci-dessus. Ces stratégies sont « objectives » dans la mesure où elles ne dépendent que de la trace $\mathcal{S}$. Une possibilité intéressante offerte par cette architecture est d'implémenter des heuristiques de parcours « subjectives », c'est-à-dire des stratégies de sélection pour `prendre_candidat_suivant`

qui prennent en compte des connaissances de l'analyste. Ces connaissances sont acquises au cours du processus de sélection de la $\mathcal{D}$-chronique candidate qui est potentiellement la plus intéressante à traiter pour l'analyste (Faure et al. 2006), et traiter ainsi en premier les $\mathcal{D}$-chroniques les plus intéressantes pour lui.

4.5 Évaluation de CDA

Dans cette section, nous présentons une première évaluation de l'algorithme CDA en termes de temps et de complétude des chroniques retournées. L'objet de cette section est d'expliciter les propriétés factuelles de CDA par rapport à la méthode de découverte de chroniques existantes (Duong 2001). Ce n'est pas l'objet de cette section de discuter de la valeur subjective des chroniques retournées par notre algorithme CDA et des moyens à mettre en oeuvre par l'analyste pour parvenir à établir des chroniques intéressantes. La sous-section 4.6 propose un résumé des propriétés de CDA à retenir pour une bonne utilisation de notre méthode de découverte complète de chroniques. Nous abordons ces propriétés dans les sous-sections suivantes.

Toutes les mesures présentées dans ce mémoire on été réalisées avec la configuration suivante : Linux Ubuntu 9.10, 2GB RAM, Intel(R) Core(TM)2 CPU T5500 @ 1.66GHz. Tous les algorithmes abordés ont été implémentés et exécutés en Java 5.

4.5.1 Temps limite d'interactivité

Avant de commencer l'évaluation de l'algorithme CDA, nous introduisons le concept de *temps limite d'interactivité*. L'algorithme CDA a été réalisé pour implémenter l'analyseur automatique du cycle itératif et interactif de co-construction de signatures de tâche. Nous avons vu dans le cahier des charges de cet analyseur automatique (cf. section 2.3.3) que l'algorithme devait permettre un *bouclage temps réel*, c'est-à-dire fournir des motifs à l'analyste en temps réel, dès la première seconde de fouille et continuellement jusqu'à ce que tous les motifs soient extraits, de sorte que la temporalité des échanges entre l'analyste et la machine soit celle d'un dialogue. Le bouclage temps réel implique une hypothèse, c'est que l'analyste n'attendra pas très longtemps avant de passer à une itération suivante du cycle de co-construction. En effet, au bout d'un certain temps, de l'ordre de la minute, l'analyste comprendra qu'il n'aura plus beaucoup d'intérêt à laisser l'algorithme s'exécuter plus longtemps, car :

- soit les motifs renvoyés sont intéressants et l'analyste s'en contente pour créer des abstractions,
- soit les motifs renvoyés sont inintéressants et l'analyste comprend qu'il faut améliorer la requête pour trouver des motifs intéressants.

C'est cette durée d'attente avant un nouveau lancement de la requête par l'analyste que nous appelons *temps limite d'interactivité*. Dans les expériences qui suivent, nous cherchons à évaluer si l'algorithme présente des bonnes propriétés pour une exécution qui ne dépasse pas ce temps limite d'interactivité. Ce temps limite d'interactivité varie fortement selon les analystes et les requêtes, mais nous avons décidé dans les expériences suivantes qu'il était de 60 à 120 secondes.

4.5.2 Comparaison avec l'algorithme de Duong

Le premier volet de l'évaluation est la comparaison de notre algorithme avec la méthode de découverte existante de Duong (2001), cet algorithme étant le seul algorithme d'apprentissage automa-

tique de chroniques à partir de traces à notre connaissance. Pour cette comparaison, nous voulions savoir d'une part si notre algorithme `CDA` retournait bien les même chroniques que l'algorithme de Duong lorsque la base de contraintes de Duong adaptée (cf. section 4.3.1) est donnée en entrée de `CDA`, et d'autre part si le temps d'exécution de la découverte était comparable pour les deux algorithmes.

Pour cela, nous avons appliqué trois requêtes différentes sur des traces différentes avec des paramètres différents, et nous avons cherché d'une part à constater si l'ensemble des chroniques retournées était le même pour les deux algorithmes et d'autre part à comparer le temps d'exécution de chaque algorithme pour chacune de ces requêtes. Les paramètres des requêtes comme la fréquence seuil f_{seuil}, la contrainte de longueur maximale n_{max} et la fenêtre temporelle englobante win ont été choisis de manière empirique de telle sorte que les temps d'exécution des deux algorithmes soient acceptables. Pour chacune des trois requêtes, nous avons utilisé la même base de contraintes pour les deux algorithmes, c'est-à-dire la base de contraintes décrite en section 4.3.1 en entrée de l'algorithme de Duong et cette même base « *adaptée à `CDA`* » (voir figure 4.3 pour la notion de base de Duong adaptée à `CDA`) en entrée de l'algorithme `CDA`. Ces trois requêtes sont :

`Requête 1` recherche dans la trace exemple de ce chapitre $\mathcal{S}_0$ (cf. figure 4.11) toutes les chroniques ayant au moins deux occurrences ($f_{seuil} = 2$) sans autre contrainte,

`Requête 2` recherche dans la trace $\mathcal{S}_{mannila}$ (cf. figure 3.1) toutes les chroniques fréquentes au moins deux fois ($f_{seuil} = 2$) dans une fenêtre de 5 unités de temps ($win = 5$), avec une note $\mathcal{Q} = 0,5$ pour la base de contraintes d'entrée. $\mathcal{S}_{mannila}$ est la trace exemple de 25 événements donnée par Mannila et al. dans leur papier fondateur de découverte d'épisodes à partir de séquences d'événements (Mannila et al. 1997),

`Requête 3` recherche dans la trace $\mathcal{S}_{driving}$ l'ensemble des chroniques ayant au moins trois occurrences ($f_{seuil} = 3$) dans une fenêtre temporelle d'une seconde. $\mathcal{S}_{driving}$ est une trace d'interactions obtenue à partir de l'instrumentation par des capteurs d'une voiture appartenant à l'INRETS[26]. Cette trace est celle d'un individu conduisant la voiture instrumentée pendant deux minutes. Elle contient 266 événements étalés sur 127590 unités de temps, où l'unité de temps est la milliseconde (donc la largeur de fenêtre temporelle est $win = 1000$). Ces événements ont été obtenus par discrétisation des courbes de valeurs en fonction du temps générées par le capteurs de la voiture. Par exemple, la courbe des valeurs de l'angle du volant a été discrétisée en ne retenant que des valeurs d'intérêt : l'angle revient à 0, l'angle du volant dépasse une valeur seuil sur la droite ou sur la gauche, etc. $\mathcal{S}_{driving}$ contient en tout 28 types d'événement ($|\mathbb{E}| = 28$).

Prendre en compte la contrainte win n'est pas prévu par l'algorithme de découverte de chroniques tel que présenté par Duong mais il est facile de la prendre en compte dans le méthode de découverte en la poussant dans l'algorithme de reconnaissance CRS. En effet, lors de la reconnaissance par CRS, chaque fois qu'une occurrence en cours de reconnaissance dépasse en amplitude temporelle la valeur win, cette occurrence est disqualifiée du processus de reconnaissance, et donc de découverte (cf. section 4.4.1.1).

Pour chaque requête, nous avons également effectué un post-traitement de l'ensemble des chroniques fréquentes retournées par l'algorithme de Duong afin de ne garder que les chroniques fréquentes minimales, comme l'algorithme `CDA`.

Nous avons constaté que chaque algorithme retourne pour les trois requêtes exactement le même ensemble de chroniques fréquentes. À titre indicatif, cet ensemble est de taille 1 pour la `Requête 1` et vaut $\{(ABC, \{A[-1,3]BA[-3,1]CB[-2,1]C)\}\}$. Pour la `Requête 2` et la `Requête 3`, les ensembles de chroniques fréquentes minimales avaient respectivement 6 et 19 chroniques, et la taille de la chronique la plus longue était 4 pour la `Requête 2` et la `Requête 3`.

[26] Institut national de recherche sur les transports et leur sécurité (`www.inrets.fr`).

	Requête 1	Requête 2	Requête 3
Duong (2001)	$47ms$	$2,64s$	$7.04s$
CDA	$290ms$	$2,31s$	$10.4s$

Table 4.1 – Comparaison en temps pour trois requêtes entre CDA et l'algorithme de Duong.

Le tableau 4.1 montre les résultats de cette première comparaison. En annexe A, nous avons appliqué ce principe de comparaison de manière plus complète sur la trace $\mathcal{S}_{mannila}$ en faisant varier les trois paramètres f_{seuil}, win et $\mathcal{Q}$. Le tableau 4.1 est représentatif des résultats que nous avons obtenus et présentés en annexe. En effet, d'une part l'algorithme le plus rapide n'est pas le même selon les valeurs des trois paramètres. Cependant, on constate que dans les cas où la durée totale de découverte est supérieure à $500ms$, c'est-à-dire pour les cas où l'analyste pourrait percevoir une différence d'exécution, l'algorithme CDA est en moyenne un peu plus rapide que l'algorithme de Duong (il est précisément $1,11$ fois plus rapide en moyenne pour nos mesures).

Nous avons voulu également comparer les évolutions respectives des durées de l'algorithme de Duong et de CDA en fonction de $|\mathcal{S}|$, la longueur de la trace $\mathcal{S}$ donnée en entrée. Pour ne faire varier d'une mesure à l'autre que ce paramètre, nous avons procédé comme suit. Nous avons choisi $\mathcal{S}_{mannila}$ comme trace d'entrée, avec comme fréquence minimale $f_{seuil} = 3$. Pour créer des séquences plus longues, nous avons appliqué des requêtes sur une trace $\mathcal{S}_n$ obtenue en concaténant n fois la trace $\mathcal{S}_{mannila}$. Afin de s'assurer que le processus interne reste le même, nous avons appliqué une fréquence seuil pour chaque trace $\mathcal{S}_n$ de $n \times f_{seuil}$. Nous nous sommes également assurés qu'il n'était pas possible de découvrir des chroniques fréquentes à cheval sur deux occurrences successives de $\mathcal{S}_{mannila}$ dans $\mathcal{S}_n$ en assurant un trou de 31 unités de temps entre deux occurrences successives de $\mathcal{S}_{mannila}$ et en posant la contrainte $win = 31$.

La figure 4.6 montre le résultat obtenu pour cette expérience pour un *temps limite d'interactivité* de $70s$. Avec une telle expérience, l'évolution de la durée de découverte est linéaire avec le même facteur de linéarité pour l'algorithme de Duong et pour CDA. Ce résultat était attendu car le processus de génération de chroniques, que ce soit celui de l'algorithme de Duong ou celui de CDA, ne dépend en rien de la longueur de la trace $\mathcal{S}$. Seul le comptage du nombre d'occurrences de chaque chronique dépend de $|\mathcal{S}|$. Dousson et al. ont prouvé que cette dépendance est linéaire, et nous l'observons sur la figure 4.6. Même si ce résultat ne fait que confirmer ce que nous attendions, il nous permet de visualiser et de mesurer le fait que pour des séquences de plus en plus longues et pour une durée de découverte tout à fait « normale », c'est-à-dire de l'ordre des secondes, la durée d'exécution de CDA se confond avec celle de Duong.

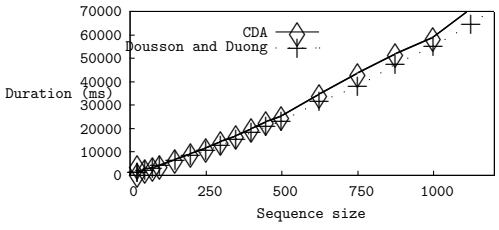


Figure 4.6 – Comparaison de la durée de découverte entre l'algorithme de Duong et CDA en fonction de la longueur de la trace

De manière générale, le comportement de CDA est le même que celui de l'algorithme de Duong lorsqu'ils sont tous les deux appliqués dans les mêmes conditions, c'est-à-dire sur la même trace,

avec des paramètres d'entrée égaux et des bases de contraintes équivalentes.

4.5.3 Observation des résultats de fonction de la base de contraintes temporelles

Le but de cette section est simplement d'observer à titre comparatif, l'ensemble complet des chroniques fréquentes minimales que peut retourner l'algorithme CDA en fonction du type de base de contraintes donnée en entrée. Pour ce faire, nous avons appliqué CDA à la trace $\mathcal{S}_{mannila}$, avec une fréquence minimale $f_{seuil} = 3$ et une contrainte de fenêtre englobante $win = 3$.

Le premier des trois types de base de contraintes dont nous proposons une méthode de construction en section 4.3 est la base de contraintes de Duong adaptée à CDA. La figure 4.7 montre l'ensemble des chroniques fréquentes minimales retourné par CDA lorsque la base de contraintes de Duong adaptée est donnée en entrée. Cet ensemble de chroniques est donc le même ensemble que l'ensemble des chroniques retourné par l'algorithme de Duong, suivi d'une suppression des chroniques fréquentes non minimales.

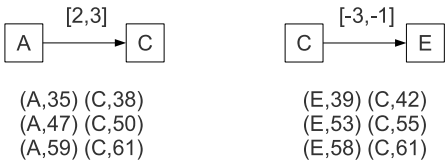


Figure 4.7 – Chroniques fréquentes minimales dans $\mathcal{S}_{mannila}$, $f_{seuil} = 3$, $win = 3$, retournées par CDA avec la base de contraintes de Duong adaptée (construite avec $\mathcal{Q} = 0.5$). C'est aussi l'ensemble de chroniques fréquentes minimales retourné par l'algorithme de Duong.

La figure 4.8 montre l'ensemble complet de toutes les chroniques fréquentes minimales, lorsque la base de contraintes en entrée est la base de contraintes hybride. C'est donc aussi l'ensemble des épisodes hybrides minimaux fréquents.

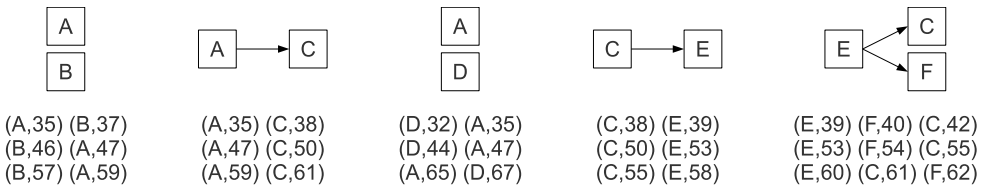


Figure 4.8 – Épisodes hybrides fréquents minimaux dans $\mathcal{S}_{mannila}$, $f_{seuil} = 3$, $win = 3$, c'est-à-dire les chroniques fréquentes minimales retournées par CDA avec la base de contraintes hybride

Enfin, la figure 4.9 montre l'ensemble des chroniques fréquentes minimales retourné par CDA lorsque la base de contraintes complète est donnée en entrée.

Cette comparaison des chroniques fréquentes minimales retournées par CDA en fonction de la base de contraintes d'entrée donne quelques informations. En ce qui concerne les résultats de l'algorithme de Duong (figure 4.7), en plus du problème déjà soulevé de n'avoir qu'une seule contrainte temporelle possible par couple de types d'événement, nous nous apercevons sur cette expérience, qu'il n'est pas capable de détecter de chroniques de taille 3 dans la trace $\mathcal{S}_{mannila}$ alors que les deux autres figures prouvent qu'il existe pourtant de tels motifs dans $\mathcal{S}_{mannila}$. Nous constatons ensuite que l'ensemble des épisodes hybrides fréquents minimaux (figure 4.8) d'une trace est moins important en nombre que l'ensemble des chroniques avec la base de contraintes

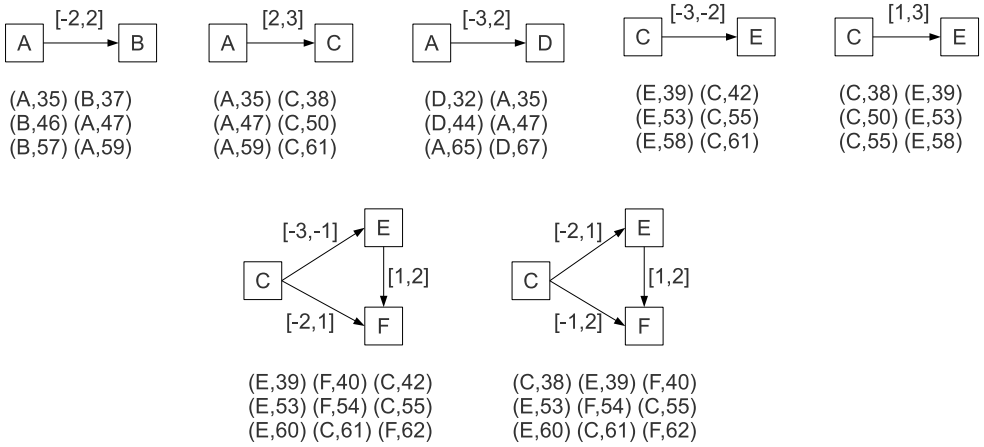


Figure 4.9 – Chroniques fréquentes minimales dans $\mathcal{S}_{mannila}$, $f_{seuil} = 3$, $win = 3$, retournées par CDA avec la base de contraintes complète.

complète (figure 4.9). Pourtant, l'ensemble des épisodes concernés, c'est-à-dire AB, AC, AD, CE et CEF, est le même dans les deux cas. La différence est que la base complète permet de proposer des variations pour les épisodes CE et CEF, ce qui permet d'avoir plus de résultats et d'obtenir une description plus fine des motifs présents dans la trace. La contre-partie est le risque d'une surcharge d'informations dans l'ensemble des chroniques résultats. Cette surcharge d'information concerne tant le nombre de chroniques retournées, comme nous venons de le voir, que la quantité d'information présente dans chaque chronique retournée. En effet, dans le cas d'une base hybride, les chroniques retournées peuvent se représenter sous la forme d'un épisode hybride et ne contiennent donc pas de contraintes numériques, parfois difficiles à interpréter. Par exemple dans le cas de la base complète, il existe deux chroniques basées sur l'épisode CEF, chacune définissant trois contraintes temporelles à deux bornes. Il n'est pas facile de faire directement la différence entre ces deux chroniques, alors que la base hybride permet de ne proposer qu'un seul motif basé sur CEF, ce motif n'ayant que deux informations temporelles simples : « C après E » et « F après E ». Pour un analyste qui cherche à effectuer une analyse fine des motifs présents dans la trace, cela peut être intéressant, mais pour un analyste qui effectue une première analyse de la trace, il est très probable que la base de contraintes hybride sera un meilleur compromis.

4.5.4 Traitement limitatif de CDA, complexité en mémoire, et ensemble Traitées

Dans cette section nous montrons par les mesures que le traitement limitatif, c'est-à-dire le traitement qui prend le plus de temps, de CDA est le comptage (l. 12 de l'algorithme 3). Pour cela, nous avons exécuté CDA sur la trace $\mathcal{S}_{driving}$ avec une base de contraintes hybride en entrée, et $win = 1000$ (c'est-à-dire 1 seconde) comme contrainte de fenêtre englobante. Nous avons fait varier les valeurs de la fréquence seuil f_{seuil} de 2 à 10 et de la contrainte de longueur maximale de 2 à 10 également, soit au total 81 exécutions de CDA. Nous avons fixé pour cette expérience le *temps limite d'interactivité* à 60s et nous avons interrompu l'exécution à chaque fois que la durée dépassait 60 secondes. Pour chacune de ces exécutions nous avons mesuré le temps d'exécution global de la découverte complète de chroniques et le temps dédié au comptage des chroniques, c'est-à-dire le temps uniquement dédié à l'exécution de la ligne 12 (cf. algorithme 3) de CDA, c'est-à-dire le temps total mis par l'algorithme CRS pour reconnaître toutes les occurrence d'une

chronique candidate dans la trace.

Nous avons mesuré que dans tous les cas, le comptage représentait entre 97% et 99% du temps de découverte total. Plus la durée est longue et plus les mesures tendaient vers 97%, ce qui signifie que plus la découverte est longue et plus un autre traitement prenait de l'importance. Nous avons observé ce même comportement sur deux traces et avec d'autres paramètres et contraintes utilisateur.

Ce que nous tirons de ces observations, c'est que dans la fourchette de temps limite d'interactivité, le traitement limitant est le comptage des chroniques dans la trace. Ce résultat est un résultat habituel dans la communauté de la fouille de données, dans laquelle il est très courant de rencontrer des algorithmes pour lesquels l'enjeu est souvent de limiter à tout prix le nombre de comptages de motifs générés.

Dans le cas de notre algorithme `CDA`, cette propriété justifie certains choix naïfs dans sa conception. Par exemple, nous testons de manière naïve la présence de chroniques minimales dans l'ensemble `Fréquents` (via la procédure `contient_descendants`), et maximales dans l'ensemble `Non_Fréquents` (via la procédure `contient_ascendants`). Les mesures sur le temps de comptage nous montrent qu'il ne sert à rien pour l'instant d'optimiser ces deux procédures, car cela n'impacterait pas de manière significative le temps global de découverte. L'autre choix naïf, plus critique encore, est celui de stocker la totalité des chroniques déjà traitées dans l'ensemble `Traitées` par une itération de `CDA` (l. 7) et de tester systématiquement la présence des chroniques candidates générées dans cet ensemble (l. 22). En effet, plus les itérations de `CDA` se succèdent, plus l'ensemble `Traitées` grandit et donc plus le test de non appartenance d'une chronique $\mathcal{C}'$ est coûteux. Combiné à cela, plus les itérations se succèdent, plus les chroniques traitées par `CDA` ont tendance à être longues et plus les chroniques enfant candidates $\mathcal{C}'$ générées ont tendance à être nombreuses. Cela signifie qu'avec le temps, la boucle des lignes 21 à 25 effectue plus d'itérations et donc plus de tests de non appartenance.

Au vu des tests de temps de comptage effectués, nous pouvons dire que dans la fourchette de temps limite d'interactivité, les effets de ces implémentations naïves sont négligeables et non perceptibles pour l'analyste. En revanche, en s'approchant des 60 secondes de durée et en allant au-delà, le temps imparti aux tests de non-appartenance est de plus en plus grand en proportion du temps de découverte global. Le coût du comptage étant tellement grand (et d'autant plus grand que la trace $\mathcal{S}$ est longue), le nombre de chroniques stockées dans l'ensemble `Traitées` est largement trop faible pour déclencher une erreur de mémoire dans le temps limite d'interactivité. Sur toute la durée de cette thèse, nous n'avons jamais rencontré de problème de dépassement de mémoire lors de tous nos tests sur `CDA`, car à mesure que les itérations de `CDA` se succèdent, les itérations sont de plus en plus espacées, du fait du temps de plus en plus long nécessaire aux tests de non-appartenance, et donc l'ensemble `Traitées` croît de plus en plus lentement. En d'autres termes, au-delà du temps limite d'interactivité, le problème de temps lié aux tests de non-appartenance prend le dessus sur le problème de mémoire. Dans le cadre du cycle de co-construction présenté au chapitre 2, l'analyste ne se trouvera pas dans cette situation car dans le temps limite d'interactivité, il n'y a pas de problème de mémoire, ni de problème de temps excessif dépensé pour les tests de non-appartenance à l'ensemble `Traitées`.

4.5.5 Impact des paramètres d'entrée

Dans la section 4.2.7, nous avons estimé que la taille de l'espace d'exploration des $\mathcal{D}$-chroniques candidates était dépendante de $p^{n_{max}^2}$, où n_{max} est la longueur de la plus longue chronique fréquente minimale et p le nombre de contraintes par graphe de contraintes dans la base de contraintes $\mathcal{D}$.

Pour mieux cerner l'impact d'une telle dépendance, prenons l'exemple de la base de contraintes $\mathcal{D}_0$ (cf. figure 4.12), pour laquelle $p = 6$. Il existe $6^3 = 216$ $\mathcal{D}_0$-chroniques basées sur l'épisode ABC. Supposons maintenant que l'événement D soit présent dans la base $\mathcal{D}_0$, et qu'il y ait des graphes de contraintes dans $\mathcal{D}_0$ pour les couples (A, D), (B, D) et (C, D). Il existerait alors $6^{C_4^2} = 6^6 = 46656$ $\mathcal{D}_0$-chroniques basées sur l'épisode $ABCD$. Le processus de découverte pourrait alors passer un temps très long à générer et compter des chroniques basées sur l'épisode $ABCD$ uniquement. Ce que cette dépendance en $p^{n_{max}^2}$ implique, c'est que l'algorithme aura tendance à « s'engluer » dans la génération de chroniques trop longues si l'on n'y fait rien.

Dans les faits, l'algorithme CDA n'explorera pas l'ensemble des $\mathcal{D}$-chroniques candidates de longueur n_{max}. En effet, seules seront explorées les $\mathcal{D}$-chroniques de taille n_{max} ayant une chronique parent de taille $n_{max} - 1$ fréquentes. Par exemple, si l'épisode parallèle ABC n'est pas fréquent, c'est-à-dire si la chronique $(ABC, \{\})$ n'est pas fréquente, alors jamais les chroniques basées sur des épisodes contenant ABC, par exemple $(AABC, \{\})$, $(ABBC, \{\})$, $(ABCC, \{\})$, $(ABCD, \{\})$, etc, ne seront explorées. Par la théorie, nous ne pouvons pas estimer plus finement le nombre de $\mathcal{D}$-chroniques qui seront effectivement traitées. Les mesures effectuées sur CDA données en annexe A nous confirment que la durée d'exécution globale de CDA est fortement dépendante en n_{max}. En effet, sur les 140 mesures de temps effectuées, on observe une variation très sensible de la durée d'exécution de CDA en fonction de n_{max}. Le tableau 4.2 montre le temps d'exécution moyen de CDA en moyenne pour nos mesures données en annexe. Le cas $n_{max} = 6$ n'a pas été reporté dans ce tableau car il correspond à seulement 7 mesures de notre annexe. Comme nous avons fixé le *temps limite d'interactivité* à 120 secondes pour chaque exécution de CDA lors de nos mesures, les seules mesures qui ont été retenues pour n_{max} sont celles pour lesquelles la durée de CDA est en dessous de 120 secondes ; ces mesures-là ne sont pas représentatives de la réalité. Le tableau montre sur les valeurs de 2 à 5 de n_{max}, que l'évolution de la durée en fonction de n_{max} est bien de nature exponentielle. Ceci nous pousse à adopter le principe suivant pour la découverte complète de chroniques :

> Il existe une valeur critique de longueur de chroniques, notée $n_{acceptable}$, pour laquelle découvrir l'ensemble complet des chroniques de longueur supérieure à $n_{acceptable}$ nécessitera un temps trop long pour être acceptable par l'analyste alors que découvrir l'ensemble complet des chroniques de longueur inférieure à $n_{acceptable}$ sera quasiment instantané pour l'analyste. Cette valeur critique $n_{acceptable}$ variera entre 2 et 6 en fonction des paramètres d'entrée de l'algorithme CDA.

Nous énonçons ce principe sans chercher à caractériser précisément la valeur de $n_{acceptable}$, qui dépendra du point de vue de l'analyste et du contexte. Au vu du tableau 4.2, il est probable que cette valeur soit $n_{acceptable} = 4$ dans le cas de la découverte interactive de chroniques à partir de la trace $\mathcal{S}_{mannila}$.

n_{max}	2	3	4	5
Nombre de mesures	54	11	31	37
Durée moyenne	26ms	116ms	1,87s	95,7s

Table 4.2 – Résumé de la durée d'exécution de CDA en fonction du paramètre n_{max}.

À partir des mesures de temps effectuées sur la trace $\mathcal{S}_{mannila}$ et présentées en annexe A, il serait possible de visualiser et d'analyser l'impact des autres paramètres d'entrée de l'algorithme CDA : $\mathcal{S}$,$\mathcal{D}$,f_{seuil}, win et n_{sup}. Nous avons d'ailleurs étudié en particulier l'impact de $|\mathcal{S}|$, la longueur de la trace sur la durée, en section 4.5.2. Pour les autres paramètres, l'étude des relations de

dépendance entre leurs valeurs et la durée de CDA ne permettrait pas de faire ressortir des modèles de comportement simple à partir mesures effectuées en annexe. En réalité, il est plus judicieux de voir l'impact de chacun des paramètres d'entrée sur le paramètre n_{max}, qui est de loin le plus limitant car la durée de la découverte en dépend exponentiellement. n_{max} n'est pas un paramètre d'entrée, mais n_{max} dépend directement des paramètres d'entrée. En effet, plus la fréquence seuil f_{seuil} diminue, plus il existera de chroniques fréquentes longues, et plus n_{max} augmentera, et enfin la durée augmentera. Il en va de même pour les autres paramètres d'entrée. Nous avons représenté ce réseau de dépendances sur la figure 4.10. Ce réseau a été élaboré de manière empirique à la suite de nos différentes mesures et observations et non pas de manière automatique, ni théorique. Il montre que quelles que soient les variations opérées sur les paramètres d'entrée, elles n'auront d'impact sur la durée finale qu'au travers de leur impact sur la valeur de n_{max}.

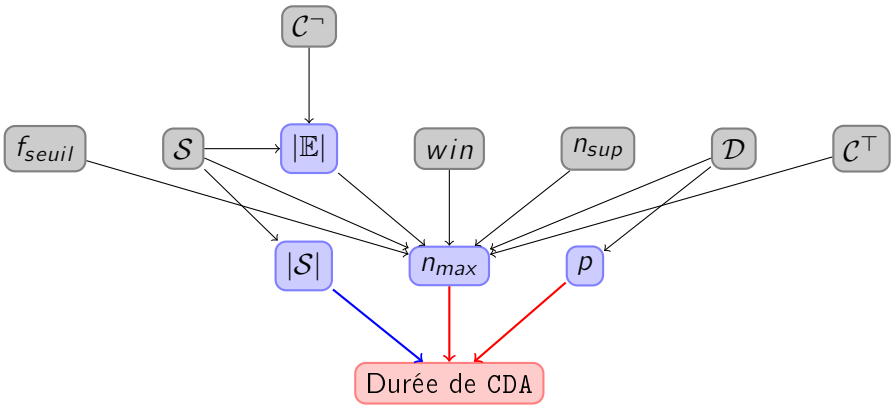


Figure 4.10 – Réseau de dépendances entre les paramètres de CDA influant sur la durée de son exécution. Les boîtes grises contiennent les paramètres d'entrée de CDA, configurables par l'analyste. Les boîtes bleues contiennent les paramètres intermédiaires. La boîte rouge contient le paramètre cible de notre analyse : la durée d'exécution de l'algorithme CDA. Les flèches représentent les dépendances entre les paramètres. Les flèches bleues représentent des dépendances linéaires, les flèches rouges des dépendances critiques (polynomiales ou exponentielles), et les flèches noires des dépendances non quantifiables.

Réduire le nombre de types d'événement $|\mathbb{E}|$ entrant en jeu dans la découverte, *via* la contrainte de non ascendance, influe sur n_{max} dans la mesure où cela peut réduire le nombre de chroniques fréquentes minimales, et donc peut-être n_{max}. La contrainte d'ascendance réduit l'espace d'exploration de $\mathcal{D}$-chroniques candidates (cf. figure 4.13) dans la mesure où cette contrainte revient à débuter l'exploration à une $\mathcal{D}$-chronique racine donnée en entrée par l'analyste (cf. section 4.4.1.3). De même que pour la plupart des autres paramètres d'entrée, elle n'influera significativement sur la durée totale que dans la mesure où elle réduira la valeur de n_{max}. Autrement dit, en notant $\mathcal{C}_{racine}$ la chronique de départ de la contrainte d'ascendance, cette contrainte ne réduira n_{max} que s'il n'existe aucune $\mathcal{D}$-chronique fréquente descendante de $\mathcal{C}_{racine}$ de longueur égale à n_{max}.

4.6 Bilan

L'algorithme CDA présenté dans ce chapitre a été conçu pour implémenter les propriétés nécessaires à la découverte interactive et itérative de chroniques à partir de traces de manière complète, c'est-à-dire sans disqualifier *a priori* certaines chroniques du processus de découverte. Pour permettre à

l'analyste d'influer sur le processus de découverte, plusieurs contraintes sont supportées par CDA : f_{seuil}, win, n_{sup}, la contrainte d'ascendance, et la contrainte de non ascendance. L'analyste a également un fort contrôle sur le processus en choisissant la base de contraintes $\mathcal{D}$ à partir de laquelle seront construites les chroniques découvertes et en choisissant l'ordre dans lequel les chroniques candidates seront testées grâce à la procédure `prendre_candidat_suivant`. Le problème de découverte complète de $\mathcal{D}$-chroniques est très complexe puisque l'espace des candidats est dépendant de $p^{n_{max}^2}$. Lorsque $\mathcal{D}$ est la base de Duong (2001), nous avons $p = 2$ et nos mesures montrent que notre algorithme CDA se comporte dans ce cas globalement comme l'algorithme de Duong. Pour la plupart des requêtes, la découverte complète totale sera trop longue pour être acceptable par l'analyste. Cependant, l'ensemble `Fréquents` des chroniques fréquentes minimales contient à tout moment l'ensemble des solutions en cours de construction. Dans les cas de découverte trop longue, c'est cet ensemble partiel de solutions qui sert de retour à l'analyste pour élaborer sa prochaine requête. Dans une fourchette de temps limite d'interactivité, que nous avons fixée entre 60 et 120 secondes selon les mesures, l'algorithme CDA a une dépendance linéaire en $|\mathcal{S}|$ et le comptage des $\mathcal{D}$-chroniques candidates dans la trace $\mathcal{S}$ est dans cette fourchette de temps le traitement limitatif de CDA. C'est pourquoi nous avons essayé par le biais de l'introduction de contraintes monotones et des procédures `contient_descendants` et `contient_ascendants` de limiter les opérations de comptage. La durée de la découverte complète de chroniques ne dépend des autres paramètres d'entrée que par le biais de leur influence sur n_{max}. Cette lourde dépendance en fonction de n_{max} implique que la découverte complète sera totale de manière instantanée pour une valeur de n_{max} et non acceptable pour la valeur $n_{max} + 1$. Cela signifie aussi que la contrainte n_{sup} sera la contrainte la plus utile pour l'analyste, car elle sera son moyen le plus direct pour influer sur la durée de la découverte en empêchant le processus de « s'engluer » dans la génération de chroniques candidates trop longues.

(Page à détacher)

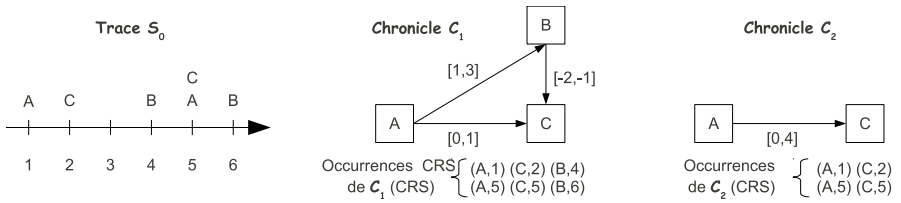


Figure 4.11 – Trace $\mathcal{S}_0$, et deux chroniques $\mathcal{C}_1$ et $\mathcal{C}_2$ telles que $\mathcal{C}_1 \preceq \mathcal{C}_2$

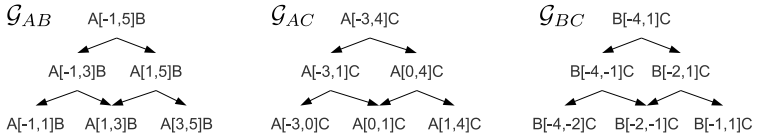


Figure 4.12 – $\mathcal{D}_0$: un exemple de base de contraintes temporelles

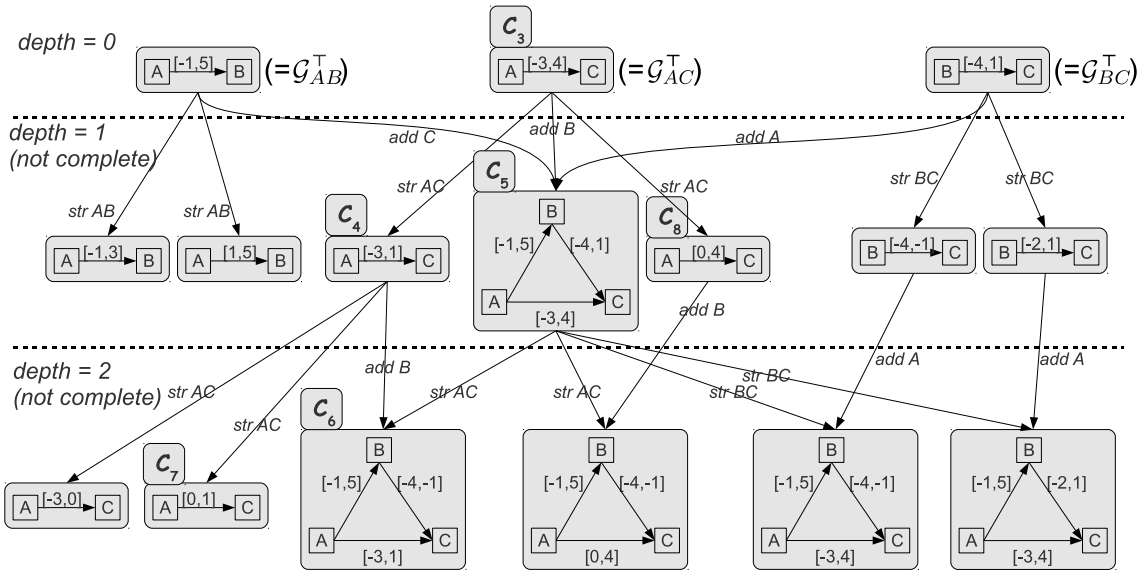


Figure 4.13 – Génération de $\mathcal{D}_0$-chroniques candidates sur les trois premiers niveaux de profondeur en initialisant le niveau 0 avec les chroniques de l'ensemble $\mathcal{D}_0^{\top}$. Les niveaux de profondeur 1 et 2 sur cette figure sont incomplets. Chaque arc représente l'application d'un opérateur de type `add` ou `str_`.C haque arc relie également une $\mathcal{D}_0$-chronique à ses $\mathcal{D}_0$-chroniques enfant.

(Page à détacher)

Algorithme 3 CDA : Chronicle Discovery Algorithm

Entrées: $\mathcal{S}$; $\mathcal{D}$; f_{seuil}
Sorties: Fréquents

1: Fréquents $\leftarrow \emptyset$
2: Non_Fréquents $\leftarrow \emptyset$
3: Traitées $\leftarrow \emptyset$
4: Candidats $\leftarrow \mathcal{D}^{\top}$
5: **répéter**
6: $\mathcal{C} \leftarrow$ prendre_candidat_suivant (Candidats)
7: Traitées $\leftarrow$ Traitées $\cup \{\mathcal{C}\}$
8: **si** contient_ascendants (Non_Fréquents,$\mathcal{C}$) **alors**
9: Aller à la prochaine itération à la ligne 5
10: **fin si**
11: **si** $\neg$contient_descendants (Fréquents,$\mathcal{C}$) **alors**
12: $f_{CRS}(\mathcal{C}) \leftarrow$compter_CRS $(\mathcal{C}, \mathcal{S})$
13: **si** $f_{CRS}(\mathcal{C}) \geq f_{seuil}$ **alors**
14: ajout_minimal ($\mathcal{C}$,Fréquents)
15: **sinon**
16: ajout_maximal ($\mathcal{C}$,Non_Fréquents)
17: Aller à la prochaine itération à la ligne 5
18: **fin si**
19: **fin si**
20: Enfants($\mathcal{C}$) $\leftarrow$ générer_enfants $(\mathcal{C})$
21: **pour** chaque $\mathcal{C}' \in$ Enfants($\mathcal{C}$) **faire**
22: **si** $\mathcal{C}' \notin$ Traitées **alors**
23: Candidats $\leftarrow$ Candidats $\cup \{\mathcal{C}'\}$
24: **fin si**
25: **fin pour**
26: **jusqu'à** Candidats $= \emptyset$
27: **retourner** Fréquents

Chapitre 5

Mise en situation réelle : co-construction de chroniques à partir des traces avec l'outil Scheme Emerger

Sommaire

L'algorithme CDA présenté au chapitre précédent a été conçu dans l'optique d'implémenter l'analyse automatique de traces qui prend part au processus interactif de co-construction de connaissances présenté au chapitre 2. Dans ce chapitre, nous abordons les problématiques liées à la mise en oeuvre concrète de ce processus de co-construction de chroniques à partir de traces modélisées gérées dans un Système à Base de Traces (SBT). La section 5.1 propose des solutions génériques pour transformer une trace modélisée, qui est au format *M-trace* (cf. section 2.1.2), stockée dans le SBT en une trace au format *séquence d'événements* utilisable en entrée de l'algorithme CDA, facilitant l'étape de préparation des données habituelle en fouille de données. La section 5.2 décrit l'outil Scheme Emerger, qui exploite l'algorithme CDA, que nous avons développé dans le but de démontrer le processus de co-construction interactive de connaissances. La section 5.3 illustre

l'utilisation de cet outil pour la préparation des M-traces du SBT issues de l'instrumentation de la plate-forme CollaborativeECM du projet PROCOGEC. Un scénario concret de co-construction interactive de chroniques dans les traces réelles de la plate-forme CollaborativeECM est détaillé pour illustrer concrètement un tel processus dans le contexte du projet PROCOGEC.

5.1 Préparations des M-traces pour l'algorithme CDA

Tout processus de découverte de connaissances comprend une phase de préparation de données préalable à la fouille de données (cf. section 2.2.1). Dans le cas de la fouille de trace, l'activité est tracée et un SBT collecte l'ensemble des observés sous la forme d'une M-trace (ou trace modélisée, définie en section 2.1.2). La phase de préparation des données consiste à sélectionner une trace modélisée que l'on souhaite analyser dans le SBT et à la transformer en une trace qui sera acceptée en entrée de l'algorithme CDA (cf. figure 5.1), c'est-à-dire en une trace telle que définie formellement en section 4.1.

La phase de préparation des données fait toujours partie intégrante du processus de découverte de connaissance (Han & Kamber 2006) et c'est le même humain, appelé « analyste » sur la figure 5.1, qui mène la phase de préparation et de découverte. Cet analyste doit donc avoir une connaissance du format des traces acceptées par Scheme Emerger, du format des traces modélisées et des techniques de préparation présentées dans la suite.

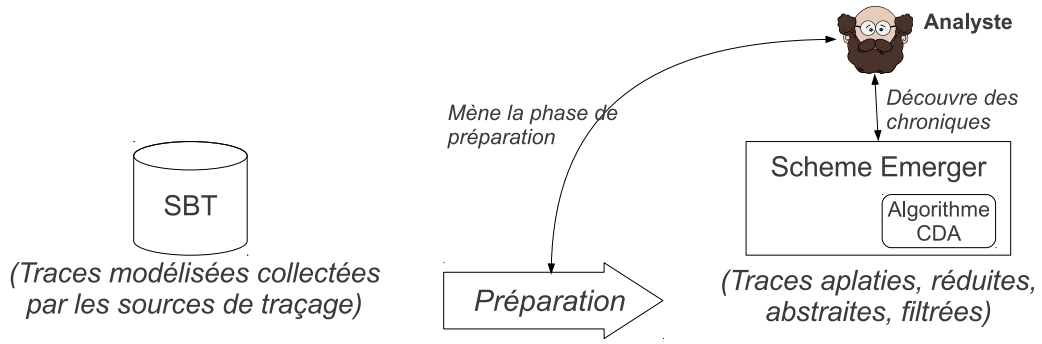


Figure 5.1 – Préparation de traces modélisées du SBT en traces compatibles pour Scheme Emerger

Nous traitons les différents aspects de la préparation des traces modélisées. L'aspect le plus important à toute préparation de données est la conversion de format. La conversion de format consiste à transformer les données disponibles pour l'analyse, c'est-à-dire les données au format *M-trace* , de façon à les fournir dans le format d'entrée tel que prévu par l'algorithme de fouille, c'est-à-dire le format *séquence d'événements*. Dans notre cas, nous proposons une démarche générique « d'aplatissement » pour cette conversion de format que nous détaillons dans la section 5.1.1. Les autres opérations (réduction, sélection, abstraction sémantique) peuvent être réalisées par le SBT grâce à l'exploitation des méthodes de transformation, ce qui rend ces opérations explicites et réutilisables. Ces opérations sont détaillées dans les sous-sections suivantes. La section 5.1.5 synthétise le processus de préparation des M-traces et situe dans ce processus les outils qui ont été développés à cet effet.

5.1.1 Aplatissement des M-traces

L'aplatissement est l'opération qui s'occupe de la conversion des traces modélisées telles qu'elles sont représentées dans le SBT (M-traces) vers un format des données telles que prévues en entrée de l'algorithme CDA (séquence d'événements). Le formalisme des M-traces a été conçu pour décrire avec le plus d'expressivité possible le déroulement observé d'un processus. Comme nous l'avons vu en section 2.1.2, une M-trace comprend un ensemble d'observés, chaque observé étant l'instance d'une classe du modèle de trace. Chaque observé d'une M-trace possède une estampille temporelle (intervalle), est caractérisé éventuellement par plusieurs attributs et peut être en relation avec un ou plusieurs autres observés. Une trace au format *séquence d'événements* telle qu'attendue en entrée de l'algorithme CDA est constituée d'événements ponctuellement situés dans le temps (pas d'intervalles) et typés, sans attributs ni relations entre événements. Dans ce format, il n'existe pas non plus de relation conceptuelle entre les types d'événement de la trace ; ils sont tous au même niveau. C'est en partie la raison pour laquelle nous parlons de l'« aplatissement » des M-traces du SBT. La figure 5.2 illustre ce problème d'aplatissement. La M-trace à préparer est notée « T^{SBT} » et la trace résultant de l'opération d'aplatissement est notée « T^{CDA} ». La trace T^{CDA} est compatible avec l'algorithme CDA ; c'est donc une séquence d'événements telle que définie en section 4.1 que l'on a notée « $\mathcal{S}$ » tout au long du chapitre précédent.

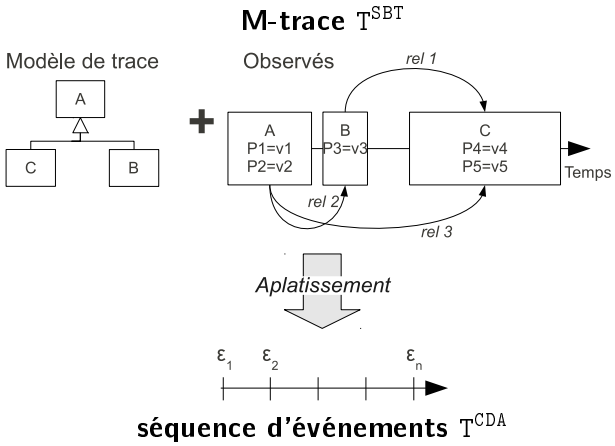


Figure 5.2 – Schéma général de l'opération d'aplatissement d'une M-trace, notée T^{SBT}, ayant hiérarchie de types, attributs, relations et persistance en une séquence d'événements, notée T^{CDA}. (Il s'agit d'un schéma, les ϵ_i sont sans rapport avec les observés de la M-trace sur cette figure)

L'aplatissement d'une M-trace relève de plusieurs sous-problèmes d'aplatissement.

- L'*aplatissement temporel* consiste à convertir la persistance des observés en événements ponctuellement situés dans le temps.
- L'*aplatissement des types d'observé* consiste à convertir les classes d'observé du modèle de trace de la M-trace T^{SBT} et leurs relations conceptuelles en un ensemble « plat » $\mathbb{E}$ de types d'événement pour la trace préparée T^{CDA}.
- L'*aplatissement des attributs* consiste à convertir les attributs des observés en événements n'ayant pas d'attribut.
- L'*aplatissement des relations* consiste à convertir les relations des observés en événements n'ayant pas de relation.

Nous traitons ces sous-problèmes d'aplatissement séparément dans la suite, mais en situation réelle l'aplatissement qui sera réalisé sera une combinaison de ceux-ci.

L'aplatissement peut sembler être simplement un problème de conversion d'un format à un autre, mais les implications du changement de format peuvent être importantes. Le formalisme des traces modélisées est un langage riche pour décrire une trace d'interactions et le langage de description des séquences d'événements est bien plus pauvre. Convertir une trace exprimée dans le formalisme des traces modélisées en une séquence d'événements peut engendrer des pertes d'informations importantes. Nous décrivons chacune des opérations d'aplatissement dans les sous-sections qui suivent et nous discutons les pertes d'informations qu'ils engendrent.

5.1.1.1 Aplatissement temporel

Pour gérer l'aplatissement temporel, nous proposons deux solutions très simples. Pour faciliter leur explication, nous supposons que chaque observé n'a ni attribut, ni relation, de sorte que l'aplatissement temporel soit la seule opération d'aplatissement nécessaire. Un tel observé sans attribut ni relation est de la forme $o = (O, t_d, t_f)$, où O est le type de l'observé, t_d sa date de début et t_f sa date de fin. La première solution pour gérer l'aplatissement temporel est d'interpréter tous les observés de la M-trace comme étant ponctuels et de remplacer chaque observé de la trace $\mathrm{T}^{\mathrm{SBT}}$ par un unique événement (O, t_d) de la trace $\mathrm{T}^{\mathrm{CDA}}$. La date de début de l'observé t_d est retenue comme la date ponctuel d'occurrence de l'observé, mais la date t_f peut être retenue également, où n'importe quelle combinaison de t_d et t_f, par exemple $(t_d + t_f) \div 2$. La deuxième solution est celle employée par Chen & yi Wu (2006) pour la fouille de séquences d'événements persistants. Elle consiste à convertir chaque observé persistant de la trace $\mathrm{T}^{\mathrm{SBT}}$ en deux événements ponctuels (O_d, t_d) et (O_f, t_f) représentant deux instants particulièrement important dans la vie de l'observé o : son début et sa fin. Cette deuxième solution consiste donc à créer deux types d'événement O_d et O_f à partir du seul type d'observé O.

La perte d'information temporelle résultant de la première solution est évidente : on perd l'information de persistance. La deuxième solution engendre également une perte d'information car même s'il y a deux événements (o_d, t_d) et (o_f, t_f) représentant les instants marquants de la vie de l'observé o, les types d'événement o_d et o_f sont au même niveau que tous les autres types d'événement de l'ensemble $\mathbb{E}$ pour la trace $\mathrm{T}^{\mathrm{CDA}}$. Rien ne lie ces deux types d'événement sémantiquement, si bien que la trace résultant peut prêter à confusion dans la pratique. Par exemple, la trace modélisée de la figure 5.3 représente une activité dans laquelle deux observés de type A apparaissent, le deuxième étant temporellement incluse dans le premier. Dans la trace préparée résultant de la deuxième solution d'aplatissement temporel on obtient deux occurrences du type A_d et deux occurrences du types A_f. Comme il n'y a aucune information dans la trace préparée qui relie les occurrences de A_d à leurs occurrences de A_f respectives, il peut y avoir une ambiguïté dans l'interprétation de la trace préparée, ne sachant plus quelle occurrence de A_f est liée à quelle occurrence de A_d. Heureusement cette configuration de chevauchement temporel de deux observés du même type est rare dans la pratique. De plus, nous avons supposé dans cette section que les observés n'ont ni relation ni attribut.

5.1.1.2 Aplatissement des types d'observé

L'aplatissement des types d'observé est simple. Pour chaque type d'observé du modèle de trace de la trace $\mathrm{T}^{\mathrm{SBT}}$, on définit un type d'événement pour la trace $\mathrm{T}^{\mathrm{CDA}}$ dont le nom est le même que celui du type d'observé. Ensuite, pour chaque observé apparaissant dans la trace $\mathrm{T}^{\mathrm{SBT}}$, on crée exactement un événement dans la trace $\mathrm{T}^{\mathrm{CDA}}$ (cf. figure 5.6) dont la date est celle qui résulte de la stratégie

Figure 5.3 – Préparation par aplatissement temporel

d'aplatissement temporel appliquée. Sur la figure 5.6, la stratégie d'aplatissement temporel choisie est la sélection de la date de début de l'observé.

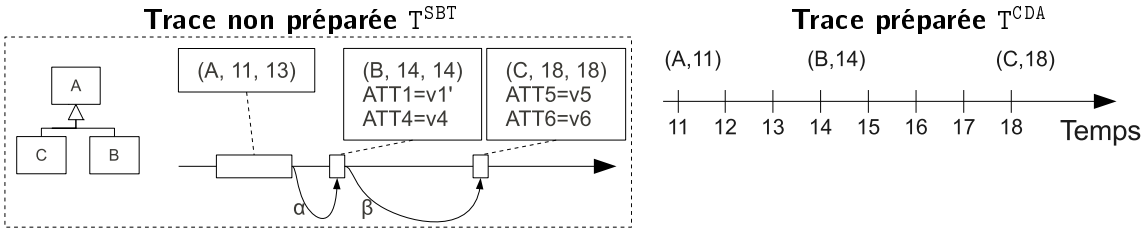


Figure 5.4 – Préparation par aplatissement des types d'observé

Dans la trace préparé T^{CDA}, les types d'événement ne sont pas hiérarchisés ni reliés par quelque relation conceptuelle que ce soit, contrairement aux types d'observé de la trace non préparée T^{SBT} accompagnée de son modèle de trace. C'est toute la description ontologique des concepts apparaissant dans la M-trace T^{SBT} que l'on ne retrouve pas dans la trace préparée T^{CDA}. Cependant, il est toujours possible de garder la hiérarchie des types à côté de la trace T^{CDA}, et de l'utiliser après la fouille pour l'interprétation des motifs par l'humain, mais cette hiérarchie des types ne peut pas être utilisée « en natif » au sein de l'algorithme CDA pour produire des chroniques possédant des types d'événement appartenant à plusieurs niveaux de la hiérarchie.

5.1.1.3 Aplatissement des attributs

Pour aplatir les attributs d'un observé de la trace T^{SBT}, il suffit de remplacer chaque paire attribut/valeur de l'observé par un nouveau type d'événement la représentant. La figure 5.5 montre l'effet d'un tel aplatissement sur la trace modélisée T^{SBT}, en supposant que le choix d'aplatissement temporel effectué sur cette figure est celui de retenir uniquement la date de début de l'observé et en ne montrant pas les événements résultant de l'aplatissement des types d'observé, c'est-à-dire $(A, 11)$ et $(B, 14)$. L'observé A ayant trois attributs, trois événements de trois types différents ont été créés dans la trace T^{CDA}. Les noms des types d'événement créés par aplatissement des attributs sont de la forme `[TYPE_OBS]-[NOM_ATT]-[VAL_ATT]`. Autrement dit, les noms sont la concaténation du nom du type de l'observé, du nom de l'attribut et de sa valeur.

La perte d'information de l'aplatissement des attributs tel que présenté ci-dessus est double. Premièrement, plus rien n'indique dans la trace T^{CDA} que les événements créés à partir du même observé appartiennent à la même « unité d'observation ». Autrement dit, la notion de container pour l'ensemble de ces événements n'est pas réifié dans la trace T^{CDA}. Cependant, l'humain pourra dans la plupart des cas facilement inférer à la vue de la trace T^{CDA} quels groupes d'événements proviennent du même observé car ces événements seront regroupés temporellement à la même date. Cette perte d'information là n'est donc pas la plus gênante. Deuxièmement, cet aplatissement des attributs

Figure 5.5 – Préparation par aplatissement des attributs

implique un aplatissement des valeurs des attributs et donc une perte d'information sur les relations que peuvent avoir ces valeurs entre elles. Ces relations entre valeurs sont pourtant nombreuses. Par exemple, pour des valeurs numériques, le nombre 9 est plus proche du nombre 10 que du nombre 20. Dans le cas d'une activité en plein air, on pourrait avoir un observé, noté `CondExt`, représentant les conditions extérieures de l'activité. Par exemple, un des attributs de l'observé `CondExt` pourrait être la température `apr`. Supposons que deux observés du type `CondExt` apparaissent dans la trace T^{SBT}, que pour le premier l'attribut `tpr` aura la valeur 15 et pour le deuxième la valeur 16, alors deux types d'événement seront générés : `CondExt-tpr-15` et `CondExt-tpr-16`. Ces deux types d'événement seront « au même niveau » dans la trace T^{CDA} que tous les autre types d'événement, c'est-à-dire traités exactement de la même façon que le type d'événement `CondExt-tpr-35` par exemple, ou qu'un type d'événement de toute autre nature, alors que sémantiquement `CondExt-tpr-15` est bien plus proche de `CondExt-tpr-16` que de `CondExt-tpr-35`. Il y aura dans la trace T^{CDA} autant de types d'événement dont le nom commencera par `CondExt-tpr` que de valeurs pour l'attribut `tpr` dans la trace T^{SBT}, alors que pour l'analyse, seules quelques intervalles de valeurs pour l'attribut `tpr` seront pertinents. Une façon de surmonter ce problème est de réaliser pour toutes les valeurs de l'attribut `tpr` un clustering en amont de l'aplatissement, afin de ne retenir pour `tpr` que quelques valeurs significatives pour l'analyste, par exemple `chaud`, `tiède` et `froid`. De telle techniques de réduction de domaines de valeur sont couramment utilisées lors de la préparation de données en découverte de connaissances (Han & Kamber 2006).

5.1.1.4 Aplatissement des relations

Les relations qui lient les observés les uns aux autres s'aplatissent dans le même esprit que les attributs, mais de manière récursive. La figure 5.6 illustre l'aplatissement des relations. Par souci de simplicité, les événements obtenus par aplatissement des attributs ne sont pas représentés sur la trace préparée de la figure. Étant donné un observé o apparaissant à la date t (après aplatissement temporel), pour chaque relation rel de l'observé o vers un autre observé o' apparaissant à la date t', l'aplatissement des relations de niveau de profondeur 1 consiste à créer autant d'événements dans la trace T^{CDA} que l'observé o' a d'attributs. Pour chaque événement ainsi créé, sa date est t, c'est-à-dire la date de l'observé o et le nom de son type d'événement est de la forme :

`[TYPE_OBS]-[NOM_REL]-[TYPE_OBS_CIBLE]-[NOM_ATT]-[VAL_ATT]`.[27]

L'aplatissement des relations de niveau de profondeur 2 consiste à appliquer ce même principe à l'observé cible o' pour chacune de ses relations vers des observés o''. Les noms des types d'événement créés dans la trace T^{CDA} sont de la forme :

[27] Pour chaque aplatissement des relations et des attributs, nous proposons pour les types d'événement ces noms « à rallonge » par défaut, mais l'analyste est libre de choisir un nom de type plus court et plus parlant.

[TYPE_OBS]-[NOM_REL]-[TYPE_OBS1]-[NOM_REL_1]-[TYPE_OBS2]-[NOM_ATT]-[VAL_ATT],

où [NOM_ATT] et [VAL_ATT] représentent les noms et les valeurs des attributs de l'observé o''. On peut ainsi, si on le souhaite, aplatir les relations au niveau de profondeur n en appliquant récursivement ce même principe n fois aux relations des observés cible. L'aplatissement des relations au niveau n consiste à aplatir les attributs des observés qui sont connectés à l'observé o par n relations. L'aplatissement des attributs tel que nous l'avons vu dans la section précédente est en fait un aplatissement des relations au niveau 0.

Sur la figure 5.6, l'observé de type A dans la trace non préparée a produit deux événement par son aplatissement de niveau 1 : (A-α-B-ATT1-v1', 11) et (A-α-B-ATT4-v4, 11), et deux événements par son aplatissement de niveau 2 : (A-α-B-β-C-ATT5-v5, 11) et (A-α-B-β-C-ATT6-v6, 11). Les deux événements (B-β-C-ATT5-v5, 14) et (B-β-C-ATT6-v6, 14) dans la trace T^{CDA} résultent de l'aplatissement au niveau 1 de la relation β l'observé B dans la trace T^{SBT}.

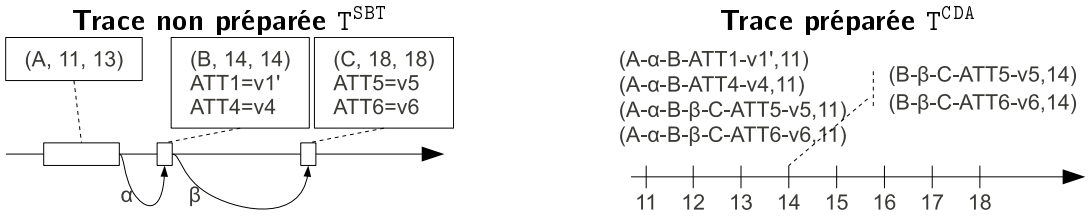


Figure 5.6 – Préparation par aplatissement des relations

L'aplatissement des relations au niveau n d'un observé de la trace T^{SBT} crée un grand nombre d'événements dans la trace T^{CDA}. Supposons que chaque observé possède r relations et s attributs, alors l'aplatissement au niveau 1 créera $r \times s$ nouveaux événements dans la trace T^{CDA}, l'aplatissement au niveau 2 créera $r \times r \times s$ événements, et $r^n \times s$ au niveau n.

En ce qui concerne la perte d'information engendrée par cette stratégie d'aplatissement des relations, on retrouve le même inconvénient qu'avec l'aplatissement des attributs : les containers des observés concernés par l'opération disparaissent. Par cet aplatissement, la situation temporelle (c'est-à-dire la date de début et la date de fin) des observés reliés à l'observé aplati n'est pas réifiée dans les événements de la trace T^{CDA} qui sont générés. Par exemple, sur la figure 5.6, les intervalles de temps [14, 14] pour l'observé B et [18, 18] pour l'observé C ne sont pas réifiés dans l'aplatissement des relations de l'observé A. Le plus souvent, cela n'est pas ennuyeux dans la pratique puisque les observés qui sont la cible des relations sont des observés de type « entité » (par exemple un document, une page Web, un formulaire, etc) qui ne se situent temporellement que via d'autres observés de nature « actions » (lire, remplir, charger, etc) ayant une relation vers eux (cf. section 5.3.2 pour un exemple).

5.1.2 Sélection

Lors de la préparation des M-traces, une étape de sélection est souvent nécessaire. La *sélection*, ou *filtrage*, consiste à choisir quels types d'observé (voire parfois quels observés instanciés), quels attributs d'observé et quels relations entre observés doivent être gardés dans la trace sur laquelle sera effectuée la fouille. L'opération de sélection n'altère pas le format de la trace à laquelle elle s'applique : une opération de sélection appliquée à une M-trace donne une M-trace et une opération de sélection appliquée à une séquence d'événements, c'est-à-dire une trace compatible

avec l'algorithme CDA, est une séquence d'événements. L'opération de sélection peut se situer avant la phase d'aplatissement ou après.

Au vu du nombre d'événements créés dans la trace T^{CDA} par l'aplatissement des M-traces, la phase de sélection est souvent nécessaire car sinon la taille de la trace T^{CDA} limiterait l'efficacité de l'algorithme CDA. Dans la pratique, l'opération de sélection est effectuée avant l'aplatissement. Au lieu d'aplatir tous les attributs de chaque observé et toutes ses relations à un niveau n puis de filtrer T^{CDA} ensuite, ce qui serait très fastidieux, l'analyste va sélectionner pour chaque observé qu'il souhaite aplatir les attributs d'intérêt et les relations d'intérêt au niveau souhaité. Cette sélection est faite en connaissance des propriétés de l'algorithme CDA (cf. section 4.5), notamment en connaissance de sa dépendance linéaire en $|T^{CDA}|$ due au comptage.

5.1.3 Équilibrage des échelles temporelles

La phase d'*équilibrage des échelles temporelles* est une opération de réduction des M-traces qui répond à un aspect bien particulier des données de type « traces modélisées d'activité ». Nous avons vu que l'activité observée est tracée par plusieurs sources de traçage qui alimentent la trace première de l'activité stockée et maintenue à jour dans le SBT (cf. section 2.1.3). Dans plusieurs activités, c'est le cas avec une activité médiée par une application Web comme CollaborativeECM, chaque source de traçage produit des observés avec une échelle de temps propre. Lorsque les échelles de temps de chaque source sont d'ordres différents, cela pose un problème pour la trace. Par exemple, la plate-forme CollaborativeECM possède deux sources de traçage principales, parmi d'autres. Le première source de traçage, que nous noterons s_{req}, trace les requêtes reçues par le serveur d'application qui héberge CollaborativeECM ; la deuxième, que nous noterons $s_{services}$, trace les services internes du système qui sont exécutés suite aux actions de l'utilisateur de CollaborativeECM. En général, une action réalisée par l'utilisateur (e.g. click sur un bouton ou sur un lien) produira zéro ou une requête envoyée vers le serveur, et donc zéro ou un observé sera produit par s_{req}. En fonction de la requête reçue, zéro ou plusieurs services système sont appelés à la chaîne : création d'un contenu, attribution de son nom, attribution de ses aspects (versionnable, éditable, multilingue, etc.). La source $s_{services}$ produira autant d'observés que de services appelés, et ces services étant appelés à la chaîne par le système, les observés produits seront espacés d'un temps de l'ordre de quelques millisecondes. Par contraste, les observés de la source s_{req} sont espacés d'un temps de l'ordre de quelques secondes ou minutes. On se retrouve donc dans la situation où les observés de la M-trace décrivent des processus de deux échelles temporelles différentes : l'une (s_{req}) est à l'échelle temporelle de l'utilisateur, et l'autre ($s_{services}$) est à l'échelle du système. Dans la M-trace, ce problème d'échelles différentes donne l'impression à l'analyste que les observés produits par $s_{services}$ arrivent « par paquets » après les observés produits par s_{req}.

La figure 5.7 illustre ce problème. On suppose que cette phase d'équilibrage s'opère sur les séquences d'événements et non sur les M-traces, c'est-à-dire qu'elle s'opère après l'aplatissement des mêmes traces. Les événements sont donc ponctuellement situés dans le temps. Sur la trace du haut de la figure, les événements représentés par un rond proviennent d'une source de traçage, notée s_{rond} et les événements en triangle proviennent d'une autre source de traçage $s_{triangle}$. Les différentes couleurs représentent les différents types d'événement. L'échelle de temps de la source s_{rond} n'est pas la même que celle de $s_{triangle}$. Les événements de la source s_{rond} apparaissent assez régulièrement et un après l'autre. Les événement de la source $s_{triangle}$ apparaissent avec un espacement temporel très court par rapport à ceux de la source s_{rond}. Dans notre exemple, s_{req} est représentée par s_{rond} sur la figure et $s_{services}$ est représentée par $s_{triangle}$.

Cette configuration pose un problème si l'on souhaite analyser la trace au niveau de grain de

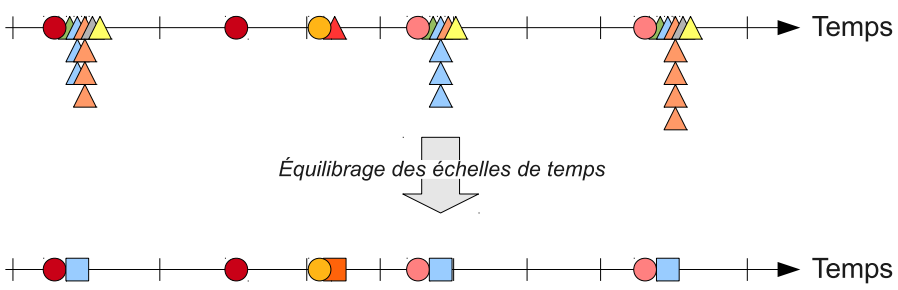


Figure 5.7 – Équilibrage des échelles temporelles

la source s_{rond}, car alors pour n événements dans ce niveau de grain à analyser (5 événements du niveau de grain de s_{rond} sur la figure), il y aura beaucoup plus d'événements du niveau de grain le plus petit (26 du niveau de grain de $s_{triangle}$ sur la figure). Les événements du niveau de grain le plus petit sont très nombreux et détériorent de beaucoup l'efficacité de CDA pour deux raisons : 1) la trace est beaucoup plus longue et donc la découverte aussi (dépendance linéaire) et 2) il y aura beaucoup de chroniques fréquentes entre événements de la source $s_{triangle}$ qui n'intéressent pas l'analyste. Il n'est pas non plus acceptable pour l'analyste de filtrer tous les événements de la source $s_{triangle}$, car ils sont porteurs d'information potentiellement intéressantes concernant le comportement des actions capturées par la source $s_{triangle}$.

Ce que nous appelons « équilibrer les échelles temporelles » des différentes sources de traçage consiste à faire en sorte qu'il y ait le même grain d'événement pour chaque source. Dans l'exemple de notre figure, cela revient à transformer chaque paquet d'événements de la source $s_{triangle}$ en un seul événement d'un nouveau type qui le représente. Sur la trace du bas de la figure nous avons représenté ces nouveaux événements par des carrés. En tout il y a sur la figure quatre paquets de $s_{triangle}$ (cf. figure 5.8), dont un de taille 1, et il y a donc quatre événements carrés. Ces quatre événements carrés sont de deux types différents (trois carrés bleus et un carré orange). Cela signifie que les trois paquets représentés par le carré bleus ont été considérés comme étant de même nature pour l'analyse au niveau de grain de $s_{triangle}$ et que le paquet représenté par la carré orange (la paquet de taille) était d'une autre nature.

Pour effectuer un tel rééquilibrage des échelles, nous proposons une méthode générale qui consiste à transformer chaque « paquet d'événements » tel que perçu à l'échelle de temps du niveau de grain le plus gros par un unique événement qui le représente. Les étapes de cette méthode sont :

1. déterminer la source dont le grain est trop fin et que l'on souhaite rééquilibrer,
2. déterminer le seuil d'étalement temporel qui caractérise un paquet d'événements et extraire les différents paquets de la trace à l'aide de ce seuil,
3. définir des classes de paquets et des règles de classification pour les paquets,
4. définir pour chaque classe de paquet un nouveau type d'événement,
5. remplacer chaque paquet par un événement dont le type d'événement est celui nouvellement créé qui représente sa classe et dont la date est celle du premier événement du paquet,
6. recommencer ce processus itérativement avec les autres sources de traçage à rééquilibrer.

Pour extraire les paquets à partir du seuil d'étalement temporel (étape 2), on parcourt la trace chronologiquement et chaque fois que la distance entre deux événements successifs de la source

à rééquilibrer est d'une durée supérieure au seuil on considère qu'ils appartiennent à deux paquets différents. C'est ce seuil d'étalement temporel qui quantifie le « grain » auquel l'analyste souhaite analyser la trace.

Exemple
Pour la trace de la figure 5.7, on souhaite rééquilibrer les événements de la source $s_{triangle}$ avec un grain qui vaut une unité de temps. Avec ce grain on extrait les quatre paquets de la figure 5.8. Ensuite on définit deux classes de paquets : la classe « bleue » et la classe « noire », et on définit les règles de classification suivantes : ($nb_{[COULEUR]}$ désigne le nombre d'événements dont le type est représenté par une triangle de couleur [COULEUR])

1. *si $nb_{rouge} = 1$ alors la classe du paquet est « noire »,*
2. *si $nb_{rouge} = 0$ et $nb_{vert} = 1$ et $nb_{jaune} = 1$ et $nb_{orange} + nb_{bleu} \geq 3$ alors la classe du paquet est « bleue ».*

Avec ces deux règles on peut classer les paquets 1, 3 et 4 dans la classe « bleue » et le paquet 2 dans la classe « noire ». Il ne reste plus qu'à remplacer les paquets par les types représentant leurs classes, c'est-à-dire des carrés de la couleur correspondante.

La trace obtenue par ce processus contient peu d'événements et contient l'essentiel (de manière résumée par les événements carrés) des informations apportées par la source de traçage $s_{triangle}$. Elle pourra donc être analysée efficacement par l'algorithme CDA.

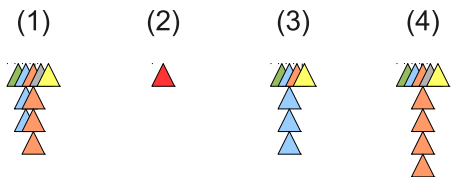


Figure 5.8 – Les quatre paquets d'événements obtenus à partir de la trace de la figure 5.7 avec un seuil de 1 unité de temps

Ces règles de classification peuvent être établies à la main comme dans notre exemple ou avec l'aide d'algorithme de classification tels que C4.5 (Quinlan 1993) sur la base du nombre d'événements de chaque types dans un paquet. Des algorithmes de clustering tels que K-Means (Lloyd 2003) ou Cobweb (Fisher 1987) peuvent aussi être utilisés pour l'apprentissage des classes de paquet qu'il est pertinent de distinguer.

5.1.4 Réduction des traces par les transformations décrites manuellement

En plus des transformations d'aplatissement, de sélection et d'équilibrage temporel qui peuvent s'appliquer à la préparation de toute trace modélisée de manière générique, l'analyste souhaitera effectuer des transformations propres au domaine de l'activité qui est tracée en fonction du type de motif qu'il recherche dans la trace. Ces transformations décrites manuellement ont pour but d'« abstraire » autant que possible la trace qui sera fouillée par l'algorithme CDA, cette abstraction ayant la double vertu de réduire le nombre d'événements que la trace fouillée contiendra et de décrire l'activité dans un langage le plus proche possible de celui des motifs recherchés par l'analyste.

Pour mieux comprendre, prenons l'exemple de la plate-forme CollaborativeECM et d'un analyste qui cherchera à élaborer avec l'algorithme CDA des motifs de collaboration autour d'un document. La M-trace première collectée par les sondes et stockée dans le SBT sera composée

entre autres d'observés d'appel aux méthodes des services internes au système et d'observés de chargements successifs des pages Web concernées. Pour l'analyste, ces actions sont de trop bas niveau par rapport à son étude de l'activité. Il préférera travailler sur des traces dont les observés décrivent les actions de plus haut niveau suivantes : « création d'un document »,« consultation d'un document »,« modification d'un document », « commentaire sur un document », « navigation », « évaluation d'un document », etc., chacun de ces observés pouvant être obtenu par transformation en fonction des observés de la M-trace première. Pour ce faire, l'analyste peut utiliser le mécanisme de transformation de M-traces du SBT. Le SBT développé dans le cadre du projet PROCOGEC permet d'effectuer ces transformations en `Javascript`.

5.1.5 Flot de préparation complet

La figure 5.1.5 résume les différentes étapes de la préparation des M-traces en séquences d'événements compatibles avec `CDA` et `Scheme Emerger`. Les traces représentées sur fond blanc sont des M-traces et les traces sur fond gris sont des séquences d'événements. La trace première collectée par les sondes de traçage et stockée dans le SBT est transformée par l'analyste grâce au système de transformation du SBT. La trace abstraite résultante peut également être stockée dans le SBT. Les opérations de sélection et d'aplatissement sont réalisées ensemble sur cette trace abstraite par l'outil `sbt2schemerger` (cf. figure 5.10), c'est-à-dire que l'outil permet de sélectionner les éléments (types, attributs ou relation) à aplatir. Pour l'aplatissement temporel, l'outil `sbt2schemerger` ne laisse pas le choix, la date de début des observés persistants est systématiquement retenue comme la date des événements aplatis. Enfin, l'équilibrage des échelles de temps n'a pas encore fait l'objet de la réalisation d'une interface graphique. À l'heure actuelle, cette opération est réalisée en `Java` par le programme `TimeScaleAdapter`, qui prend en entrée une trace, un seuil d'étalement temporel et un objet de type `RulesClassifier` contenant les règles de classification décrites manuellement en langage `Java`. L'absence d'interface graphique pour l'équilibrage temporel n'a pas été gênante pour le déroulement du projet PROCOGEC car les règles d'équilibrage ont été réalisées une fois pour toute pour les traces de `CollaborativeECM` et appliquées automatiquement à chaque exécution du flot de préparation des traces.

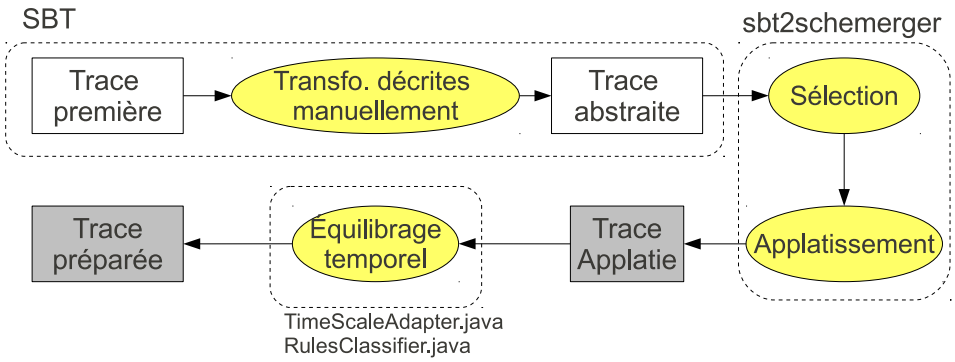


Figure 5.9 – Flot de préparation des M-traces en séquences d'événements compatible avec `CDA` et `Scheme Emerger`

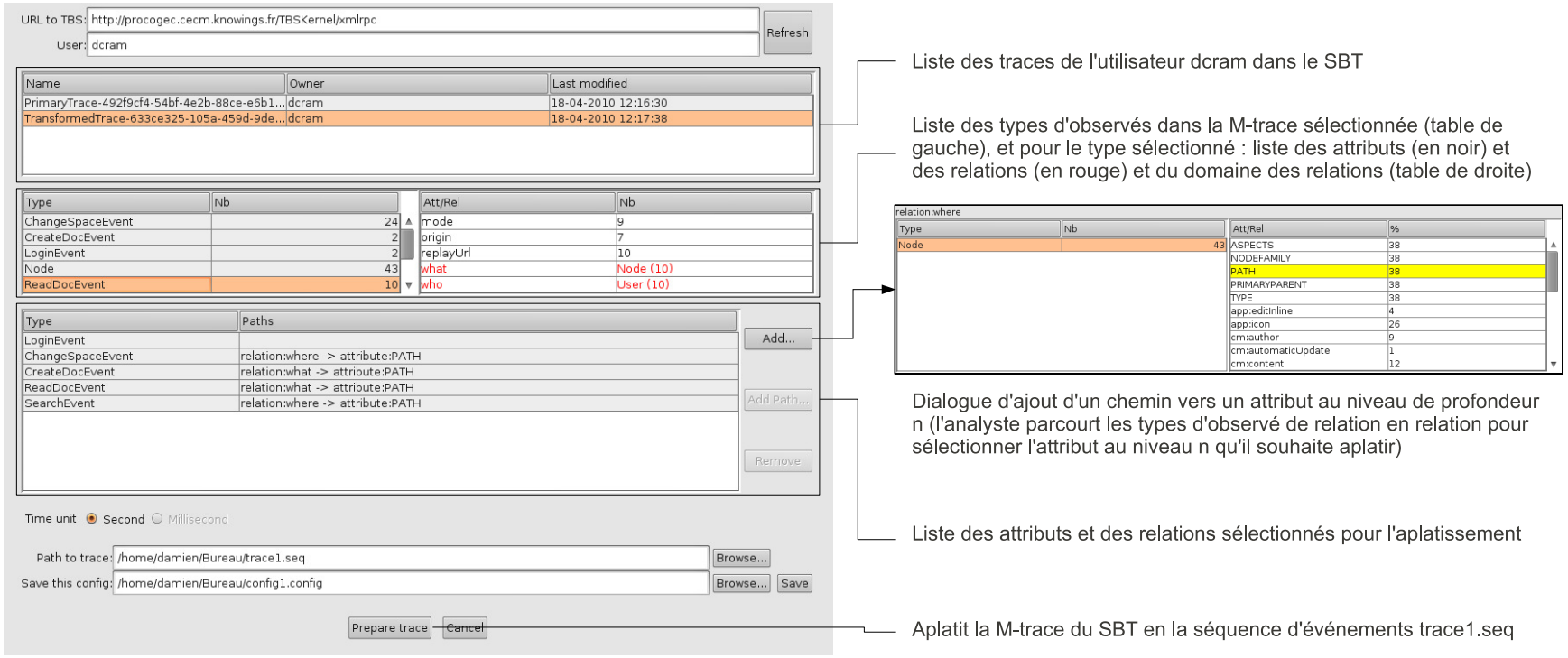


Figure 5.10 – Interface graphique de l'outil de sélection et d'aplatissement `sbt2schemerger`

5.2 L'outil Scheme Emerger : une plate-forme de découverte interactive de chroniques à partir de traces

La plate-forme Scheme Emerger a été développée dans le but de produire une interface graphique pour CDA et plus généralement d'implémenter le processus d'ingénierie interactive de connaissances présenté au chapitre 2. Le terme « schème » désigne à l'origine une structure représentant un modèle de comportement pour l'humain (Beth & Piaget 1966). Dans la cadre de l'analyse de l'activité à partir de traces d'interactions, ce terme réfère plus généralement à n'importe quelle structure de motif représentant une succession d'observés dans la trace d'activité. Dans notre cas un « schème » est donc une chronique.

La plate-forme Scheme Emerger a été développée avec Java 5. C'est une plate-forme basée sur Eclipse RCP[28] (Rich Client Platform) et bénéficie donc de tous les avantages de cette technologie, dont son interface versatile en vues et éditeurs, son architecture modulaire basée sur OSGi[29], sa portabilité sur les plate-forme Windows, Linux et Mac OS, et enfin sa bibliothèque graphique SWT (Standard Widget Toolkit) utilisant les bibliothèques natives de chaque système. En conséquence, l'architecture en plug-in de Eclipse RCP, Scheme Emerger est lui-même un plug-in et peut être installé sur n'importe quelle distribution de Eclipse, notamment le très célèbre environnement de développement Eclipse IDE. Une autre conséquence de ce choix technologique est qu'il est facile de contribuer à Scheme Emerger en développant un plug-in OSGi. La dernière version de la plate-forme Scheme Emerger est accessible en open source sur SourceForge à l'adresse suivante :

http://sourceforge.net/projects/schemerger/.

La plate-forme Scheme Emerger permet de créer et visualiser des bases de contraintes temporelles des trois types différents vus en sections 4.3, de créer, éditer et visualiser des chroniques, de visualiser une trace de type « séquence d'événements », c'est-à-dire une trace telle que définie en section 4.1, de visualiser l'ensemble des types d'événement qui apparaissent dans la trace et de modifier leur apparence dans la plate-forme (forme, couleur et étiquette), de lancer l'algorithme CDA sur la trace souhaitée avec la base de contraintes souhaitées en entrée. Des contraintes structurelles peuvent être spécifiées par l'utilisateur de la plate-forme Scheme Emerger. Ces contraintes sont le minimum de fréquence des chroniques découvertes, la contrainte n_{sup}, la contrainte win, la contrainte d'ascendance $\mathcal{C}^{\top}$ et la contrainte de non-ascendance $\mathcal{C}^{\neg}$. L'ensemble Fréquents des chroniques fréquentes en cours de construction est visible à tout moment par l'analyste et mis à jour en temps réel ou avec une fréquence paramétrable par l'analyste. L'algorithme CDA peut être arrêté et relancé à tout moment avec de nouveaux paramètres. Les occurrences des chroniques fréquentes découvertes sont visualisables dans la trace et les chroniques découvertes peuvent être rééditées manuellement par l'analyste, soit dans le but de les sauvegarder, soit dans le but de les réutiliser en tant que contrainte $\mathcal{C}^{\neg}$ ou $\mathcal{C}^{\top}$ pour une prochaine exécution de l'algorithme CDA.

La figure 5.11 présente l'interface graphique de Scheme Emerger. La figure 5.12 met le focus sur l'éditeur de requête de Scheme Emerger, afin de mieux voir quelles sont les contraintes utilisateur implémentées. Dans Scheme Emerger, on parle de « séquences » plutôt que de traces, car les traces manipulées sont des séquences d'événements, comme nous l'avons vu. Ainsi, tout algorithme capable de découvrir des motifs à partir de séquences d'événements peut être appliqué à ces séquences et faire l'objet de l'implémentation d'un plug-in de Scheme Emerger (cf. chapitre 6.

[28] http://wiki.eclipse.org/Rich_Client_Platform
[29] www.osgi.org

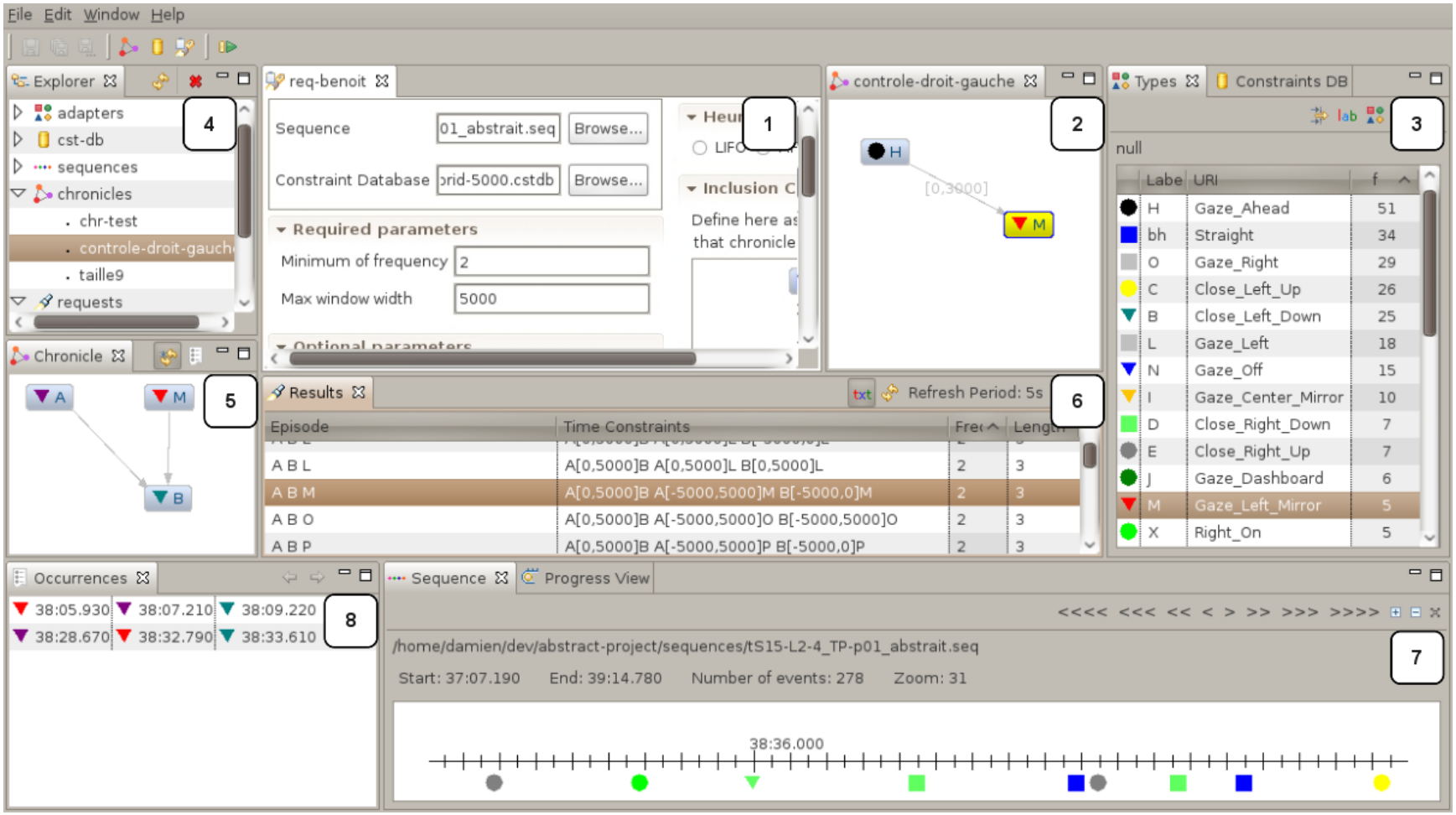


Figure 5.11 – Interface graphique de la plate-forme `Scheme Emerger`. La vue 4 affiche les ressources du projet sauvegardées et manipulées par l'analyste : bases de contraintes, chroniques, traces, requêtes. Les onglets 1 et 2 sont les éditeurs de requêtes et de chroniques. La vue 7 affiche la trace en cours d'analyse, dont l'ensemble des types est listé dans la vue 3. La vue 6 liste en temps réel les chroniques découvertes par l'algorithme `CDA`. La vue 5 affiche la chronique sélectionnée dans la vue 6, alors que la vue 8 liste ses occurrences dans la trace. Un clic sur l'une de ces occurrences déplace le focus de la vue de la trace (vue 7).

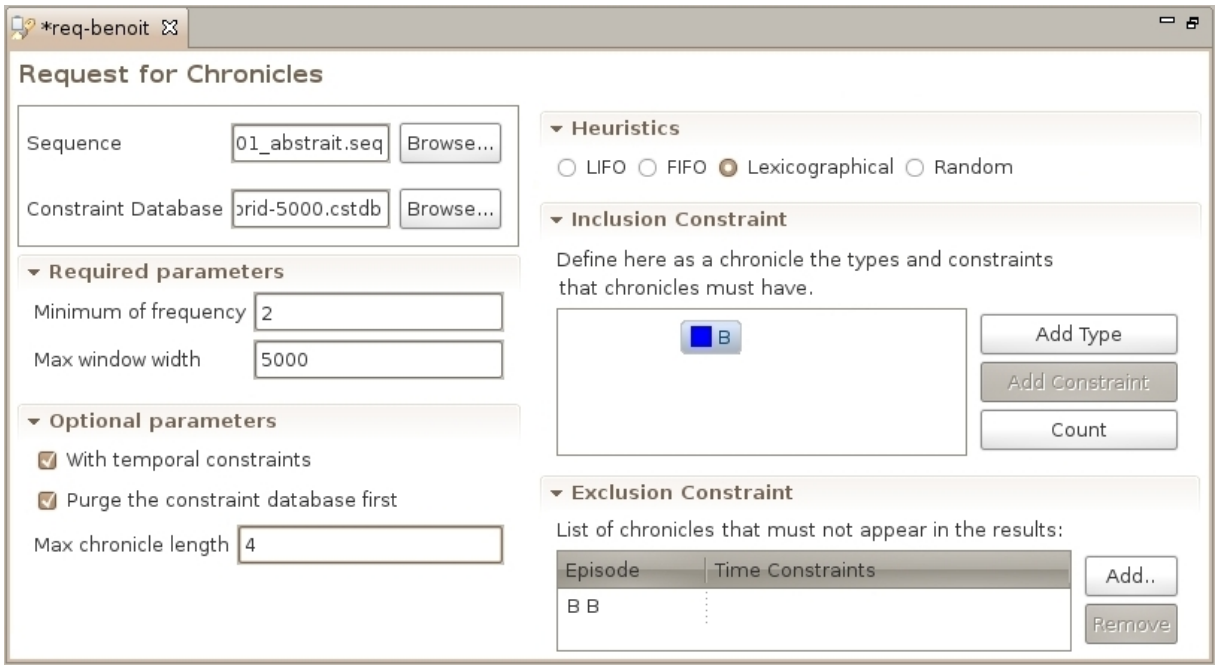


Figure 5.12 – Éditeur de requête de la plate-forme `Scheme Emerger`. L'encadré contient le lien vers la trace à fouiller et le lien vers la base de contraintes à donner en entrée de l'algorithme `CDA`. L'analyste peut spécifier la fréquence minimale f_{seuil} et donner la contrainte win qu'il souhaite dans le champ `Max window width`. La contrainte n_{sup} est saisie dans le champ `Max chronicle length` et l'analyste a le choix de découvrir des chroniques sans contrainte temporelle (c'est-à-dire des épisodes parallèles) ou avec contraintes temporelles. Il peut également « purger » la base de contraintes qui est donnée dans le champ `Constraint Database`, c'est-à-dire de supprimer de la base toutes les contraintes temporelles dont la fréquence est inférieure à f_{seuil}. La section `Heuristic` indique l'heuristique qui sera utilisée pour le parcours dans l'espace des chroniques candidates (cf. section 4.4.2). La section `Inclusion Constraint` contient la chronique $\mathcal{C}^{\top}$ qui sera utilisée pour la contrainte d'ascendance et la sections `Exclusion Constraint` donne la liste $\mathcal{C}^{\neg}$ des chroniques pour la contrainte de non-ascendance. Ces deux sections restent vident dans le cas où l'analyste ne souhaite pas utiliser ces contraintes.

5.3 Exemple de découverte de chroniques à partir de traces

5.3.1 Description du scénario

Afin d'illustrer le processus de découverte sur des traces d'interactions réelles (c'est-à-dire collectées à partir d'une activité), nous avons mis l'algorithme CDA et la plate-forme Scheme Emerger en situation de découverte de chroniques à partir des traces d'interactions collectées dans la plate-forme CollaborativeECM développée dans le cadre du projet PROCOGEC. La découverte interactive de chroniques à partir de traces a également été illustrée sur un autre domaine d'application que les activités médiées par un système informatique (cf. annexe B).

Pour les besoins d'expérimentation de l'ensemble des partenaires du projet PROCOGEC, la plate-forme CollaborativeECM a été déployée en version de test et avec un jeu de fonctionnalités assez limité par rapport à la version en cours de commercialisation. Dans cette version, la plate-forme permet de créer des espaces et des sous-espaces, où chaque espace est un conteneur de documents de manière analogue à un répertoire de fichiers. Il y a plusieurs utilisateurs de la plate-forme. Chaque utilisateur possède un compte propre sur la plate-forme et tout utilisateur peut créer espaces, sous-espaces et contenus accessibles à tous sans notion de droits[30], de restriction d'accès ou de rôle.

La trace que nous analysons est collectée à partir des sources de traçage de cette plate-forme. Il s'agit de la trace d'un utilisateur ayant créé un espace personnel et un sous-espace de bibliographie. Cet utilisateur a été créé et joué par nous. Cet utilisateur a ajouté un certain nombre d'articles de recherche au format pdf. Dans d'autres espaces, cet utilisateur a créé d'autre contenus. Plus tard, ce même utilisateur se connecte à nouveau pour accéder à des documents qu'il avait déposés sur la plate-forme.

Les chroniques que nous cherchons à retrouver et à caractériser à partir de cette trace sont celles que nous savons avoir introduites dans les traces en jouant cet utilisateur. Ces « chroniques témoin » que nous avons introduites sont au nombre de deux :

1. La chronique Login_Espace_Perso correspond à l'action de s'authentifier sur la plate-forme et de naviguer jusqu'à son espace personnel,

2. La chronique Recherche_Document correspond à l'action de naviguer rapidement d'espace en espace, en passant éventuellement plusieurs fois par les mêmes espaces, pour trouver un contenu que l'utilisateur a égaré.

5.3.2 La préparation des traces de CollaborativeECM

Le but de cette section est d'illustrer comment se passe le processus de préparation des traces au format SBT en une séquence d'événements dans un cas concret : les traces de la plate-forme CollaborativeECM. La séquence d'événements obtenue par ce processus de préparation est notée $\mathcal{S}_{cecm}$. La trace première des observés de cet utilisateur a été collectée par les sources de traçage de la plate-forme CollaborativeECM et stockée dans le SBT. Cette trace première contient au total 4877 observés et 3779 relations, les observés ayant chacun entre 2 et 20 attributs. À titre indicatif, le tableau ci-dessous donne la répartition des types d'observé dans cette trace première.

[30] La plate-forme CollaborativeECM permet la gestion des droits de manière fine et intuitive, mais dans la version déployée dans la cadre du projet PROCOGEC, la gestion des droits a été désactivée.

Type d'observé	Nombre
`AuditRepoEvent`	107
`ContextWebEvent`	259
`HttpWebEvent`	572
`Node`	1423
`Page`	259
`Search`	160
`SearchRepoEvent`	160
`Session`	839
`User`	1098

Les noms de types qui se terminent par `Event` sont des observés ayant une sémantique d'événement pour l'être humain, ce qui fait au total 1101 observés de type « événement » dans la trace.

Les étapes de transformation que nous avons appliquées sont les suivantes et dans cet ordre : transformations par le SBT décrites manuellement, aplatissement temporel, puis sélection et aplatissement des attributs et des relations par l'outil `sbt2schemerger`.

5.3.2.1 Transformations par le SBT décrites manuellement

Les transformations décrites manuellement qui ont été appliquées sont les transformations SBT d'abstraction de traces qui ont été réalisées dans le cadre des expérimentations du projet PROCOGEC. Ces transformations sont au nombre de 5. Le détail des transformations est donné par leur code `Javascript` (voir annexe D pour un exemple). Ces 5 transformations sont :

- la transformation *ChangeSpaceEvent* crée un observé de type `ChangeSpaceEvent` à partir d'observés de types `HttpWebEvent`, `ContextWebEvent` et `Page`. Cet observé de type `ChangeSpaceEvent` représente la navigation d'un espace à un autre, alors que les observés de plus bas niveau `HttpWebEvent`, `ContextWebEvent` et `Page` décrivent l'enchaînement des requêtes HTTP et des pages qui sont tracés lors de cette navigation.

- La transformation *LoginEvent* crée un observé de type `LoginEvent` à partir d'observés de type `HttpWebEvent`, `Page` (la page d'authentification) et `AuditRepoEvent` (l'appel au service d'authentification). Cet observé représente l'authentification d'un utilisateur de la plate-forme `CollaborativeECM` et le début de sa session.

- La transformation *ReadDocEvent* crée un observé de type `ReadDocEvent` à partir des observés de type `HttpWebEvent`, `ContextWebEvent` et `AuditRepoEvent`. De nombreux cas sont à prendre en compte dans la transformation parce que d'une part un document peut être lu par l'intermédiaire de plusieurs protocoles différents, et d'autre part il peut être lu par le système (c'est-à-dire générer un `AuditRepoEvent` de type « lecture ») sans avoir été réellement lu par l'utilisateur. L'observé de type `ReadDocEvent` généré représente la lecture d'un contenu par l'utilisateur quel qu'en soit le moyen.

- La transformation *CreateDocEvent* crée un observé de type `CreateDocEvent` à partir d'observés de type `AuditRepoEvent` traduisant la création d'un nouveau contenu par l'utilisateur.

- La transformation *SearchEvent* crée un observé de type `SearchEvent` à partir d'observés de types `SearchRepoEvent` et des nombreux observés de type `Node` (les contenus trouvés au format de `CollaborativeECM`) entrant en jeu dans la recherche. Cet observé représente

l'exécution d'une recherche via le formulaire de recherche par l'utilisateur et des documents retournés par la recherche.

Chaque nouvel observé ainsi créé décrit les actions réalisées par l'utilisateur avec un langage plus proche de l'activité que les observés de la trace première, qui décrivent l'exécution de services et de protocoles. Chaque observé de la trace première et de la trace transformée de type « événement », c'est-à-dire dont le nom se termine par `Event`, est relié à un observé de type `User` décrivant l'utilisateur qui a effectué l'action par la relation `who`. De même, des relations de type `what` relient les opérations effectuées aux documents et aux espaces qui en sont les objets. Par exemple, un observé de type `CreateDocEvent` est relié au contenu crée, un observé de type `Node`, par une relation `what`.

Cette opération de préparation de traces par transformations manuellement décrites consiste également en un filtrage puisque seuls les observés transformés seront gardés, afin de limiter le nombre d'événements de la trace finale et de limiter le bruit. La tableau ci-dessous donne la répartition des types d'observé dans la trace ainsi transformée.

Type d'observé	Nombre
`ChangeSpaceEvent`	93
`CreateDocEvent`	9
`LoginEvent`	9
`Node`	214
`ReadDocEvent`	14
`Search`	23
`SearchEvent`	23
`User`	148

Cela donne une trace transformée de 533 observés, dont 148 de type « événement », et de 385 relations. On note qu'il existe beaucoup moins d'observés de type `SearchEvent` dans la trace transformée (23) que d'observés `SearchRepoEvent` dans la trace première (160). Cela s'explique par le fonctionnement interne de navigation dans les pages par la plate-forme `CollaborativeECM`. En effet, certaines pages présentent une liste de contenus qui sont en réalité le résultat d'une recherche interne de type `SearchRepoEvent`. Comme cela est invisible pour l'utilisateur et que la trace transformée veut décrire l'activité telle que perçue par l'utilisateur autant que possible, ces recherches systèmes ont été filtrées par les transformations SBT. De même, ces transformations SBT font également office d'équilibrage temporel (cf. section 5.1.3) puisque les successions d'observés système provenant de la source d'audit des services de `CollaborativeECM`, c'est-à-dire les observés de type `AuditRepoEvent` et `SearchRepoEvent`, ont été réduites par les transformations présentées ci-dessus en un et un seul observé représentant une action de l'utilisateur.

5.3.2.2 Aplatissement temporel

L'aplatissement temporel que nous appliquerons est le plus simple qui soit : sélection de la date de début de l'observé comme étant la date de l'événement dans la séquence d'événements. Comme les observés de la trace de `CollaborativeECM` sont par nature ponctuels, cela ne pose aucun problème de perte d'information.

Nous appliquons également une deuxième transformation temporelle consistant à changer le domaine temporel de la trace en secondes au lieu des millisecondes. Cela consiste à diviser toutes les informations de date par 1000. Une conséquence d'une telle transformation est l'augmentation du nombre d'événements qui apparaîtront à la même date dans la séquence d'événements fouillée,

ce qui ne pose pas de problème d'un point de vue théorique pour la découverte complète de chroniques tant que deux événements du même type n'apparaissent pas à la même date. De plus les observés de la trace transformée par les transformations manuelles étant tous dans l'échelle temporelle de l'utilisateur, il en résulte que peu d'événements apparaîtront en même temps que d'autres.

5.3.2.3 Sélection et aplatissement des attributs et relations

Pour aplatir les relations et les attributs, nous avons utilisé l'outil `sbt2schemerger`, que nous avons développé et présenté un peu plus haut. De nombreux attributs des observés de la trace transformée donnent des informations système peu intéressantes pour la découverte de motifs dans l'activité de l'utilisateur. Par exemple ces informations systèmes sont pour les observés de type `Node` sa famille, son type et ses aspects internes (auditable, versionnable, referenceable, etc), alors que les informations potentiellement utiles sont son nom, sa description et son chemin dans la hiérarchie des espaces et des contenus. En réalité, peu d'attributs vont être retenus pour la trace finale.

Type	Paths
LoginEvent	
CreateDocEvent	
CreateDocEvent	relation:what -> attribute:PATH
ChangeSpaceEvent	
ChangeSpaceEvent	relation:where -> attribute:PATH
ReadDocEvent	
ReadDocEvent	relation:what -> attribute:PATH
SearchEvent	
SearchEvent	relation:where -> attribute:PATH

Figure 5.13 – Capture d'écran de `sbt2schemerger` pour la préparation des traces de `CollaborativeECM`

La capture d'écran de l'outil `sbt2schemerger` (cf. figure 5.13) montre les choix d'aplatissement et de filtrage qui ont été effectués. Pour chaque observé de type `LoginEvent`, on crée un événement dont le type est `LoginEvent`, c'est-à-dire qu'on effectue un aplatissement du type d'observé `LoginEvent` (cf. section 5.1.1.2). On effectue également des aplatissements de types pour les types d'observé `CreateDocEvent`, `ChangeSpaceEvent`, `ReadDocEvent` et `SearchEvent`. En plus de cela, pour les observés de type `CreateDocEvent` la ligne `relation:what -> attribute:PATH` signifie qu'on effectue un aplatissement des relations de niveau 1 (cf. section 5.1.1.4) avec le chemin :

`CreateDocEvent_what_node_PATH_[valeur_de_PATH]`.

Dit autrement, pour tout observé o de type `CreateDocEvent`, on crée un événement à partir de la valeur de l'attribut `PATH` de l'observé qui est relié à o par la relation `what` (tous les observés reliés à un observé de type `CreateDocEvent` par la relation `what` étant de type `Node`). On procède de même avec les autres types d'observé `ChangeSpaceEvent`, `ReadDocEvent` et `SearchEvent` selon le chemin correspondant décrit sur la figure 5.13. La figure 5.14 illustre l'aplatissement défini par les règles de la figure 5.13 sur quelques observés.

Il y aura donc au total un événement dans la séquence d'événements pour chaque observé de type `LoginEvent`, deux événements pour chaque observé de type `CreateDocEvent`, et de même deux événements pour chaque observé de type `ChangeSpaceEvent`, `ReadDocEvent` et `SearchEvent`, soit un total de $9 + 2 \times 9 + 2 \times 93 + 2 \times 14 + 2 \times 23 = 287$ événements dans la séquence d'événements qui sera fouillée.

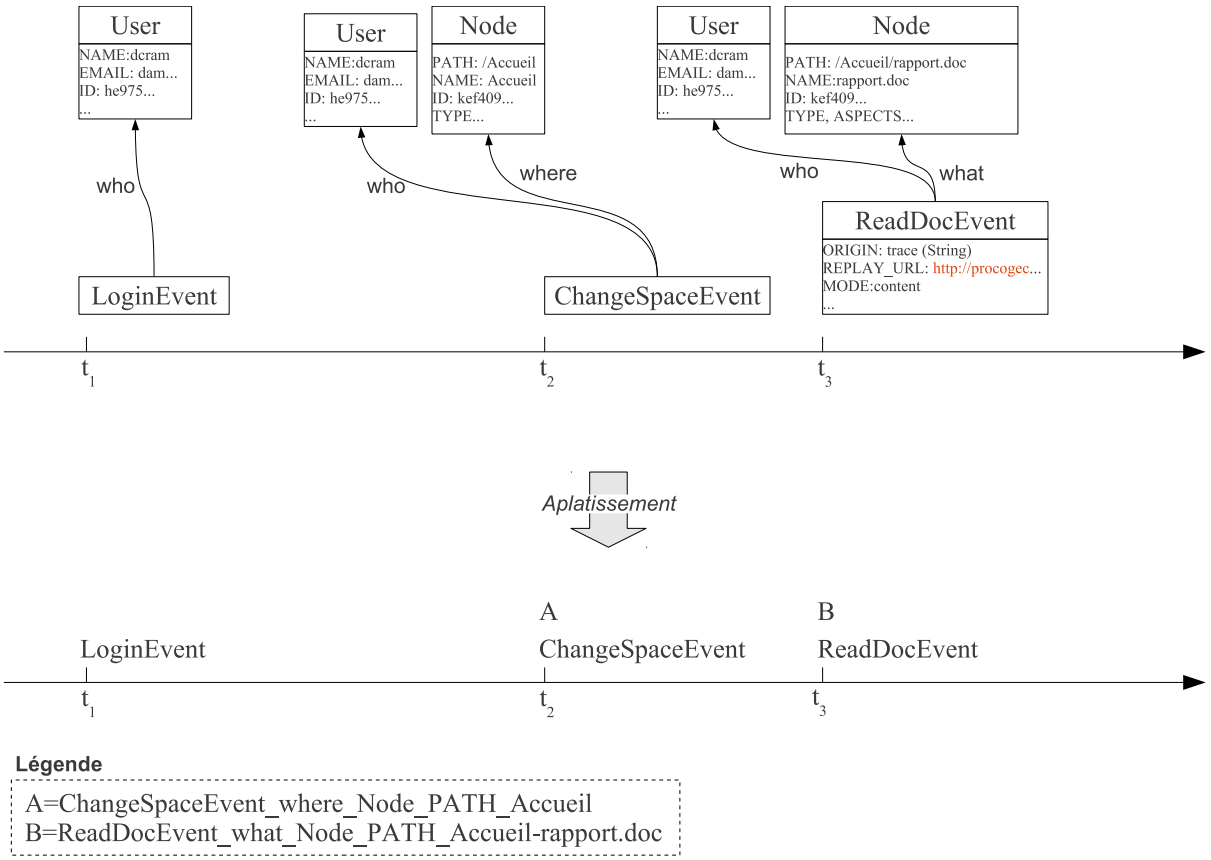


Figure 5.14 – Exemple de l'aplatissement des traces de `CollaborativeECM`

Le choix de créer 2 événements par observé de types `ReadDocEvent`, `ChangeSpaceEvent`, `ReadDocEvent` et `SearchEvent` traduit notre volonté de découvrir des chroniques mettant en jeu à la fois des actions générales et des actions sur des contenus et espaces spécifiques. Avec un tel choix, il sera par exemple possible de découvrir une chronique mettant en jeu la navigation vers l'espace d'accueil (`ChangeSpaceEvent_where_Node_PATH_Accueil`) suivi d'une navigation vers n'importe quel espace (`ChangeSpaceEvent`). L'inconvénient est que l'algorithme aura tendance à découvrir des chroniques de longueur 2 basées sur `ChangeSpaceEvent_where_Node_PATH_Accueil` et `ChangeSpaceEvent` avec la contrainte temporelle $[0, 0]$.

Si par exemple on avait souhaité analyser l'utilisation des documents de la plate-forme en fonction de leurs types, on n'aurait pas choisi l'attribut `PATH` pour l'aplatissement mais l'attribut `TYPE`[31]. Ainsi, il y aurait moins de types d'événement différents dans la trace préparée, mais il y aurait également eu moins de précision dans la description de l'activité de l'utilisateur.

L'annexe C présente la trace $\mathcal{S}_{cecm}$ obtenue en sortie de ce processus de préparation de la trace collectée dans la plate-forme `CollaborativeECM`. C'est cette trace $\mathcal{S}_{cecm}$ qui est analysée dans la section suivante.

[31] Le type des documents est renseigné dans la trace non pas par le type des observés qui les représentent mais par un attribut nommé `TYPE`.

5.3.3 Scénario de découverte interactive

Après préparation des traces collectées dans la plate-forme `CollaborativeECM`, nous obtenons donc la trace $\mathcal{S}_{cecm}$ comportant 264 événements de 32 types différents (cf. annexe C). L'analyste utilise la plate-forme `Scheme Emerger` sur la trace $\mathcal{S}_{cecm}$ dans le but de caractériser les actions de recherche de documents égarés dans la plate-forme et la navigation vers l'espace personnel, c'est-à-dire que l'analyste cherche à expliciter les chroniques `Login_Espace_Perso` et `Recherche_Document` (cf. section 5.3.1). Voici un exemple des étapes successives qui mènent à la découverte de ces deux chroniques. Les étapes de découverte interactive détaillées ci-dessous sont résumées dans la tableau 5.2.

`Étape 1` L'analyste doit d'abord construire la base de contraintes temporelles à partir de laquelle seront construites les chroniques générées par l'algorithme `CDA`. La construction d'une base de contraintes de Duong, d'une base de contraintes hybride ou d'une base de contraintes complète à partir d'une trace peut se faire très facilement dans `Scheme Emerger` grâce à l'assistant de construction de base de contraintes. Comme il n'a pas d'idée précise sur la composition des chroniques `Login_Espace_Perso` et `Recherche_Document`, il choisit de construire une base de contraintes temporelles hybrides, ce qui permet de caractériser les ordres d'apparition des types d'événement des chroniques générées avec moins de précision qu'avec une base de contraintes complète. Dans `Scheme Emerger`, la construction de la base de contraintes hybrides nécessite de spécifier la contrainte de fenêtre englobante *win*, de telle sorte qu'en interne les bornes infinies des contraintes temporelles soient remplacées par la valeur de la contrainte *win*. Ainsi en interne, au lieu d'avoir les trois intervalles $[-\infty, +\infty]$, $[-\infty, 0]$ et $[0, +\infty]$ dans chaque graphe, il y aura les trois intervalles $[-win, +win]$, $[-win, 0]$ et $[0, +win]$. Pour notre scénario, l'analyste spécifiera 60 comme valeur pour la contrainte *win*, c'est-à-dire une fenêtre englobante d'une minute. De plus, il choisira 1 comme fréquence minimale des contraintes temporelles de la base. Grâce à cette valeur, `Scheme Emerger` comptera toutes les contraintes temporelles de la base de contraintes et ne retiendra que celles qui apparaîtront au moins une fois dans la trace $\mathcal{S}_{cecm}$. Cela permet d'optimiser les requêtes futures qui seront effectuées avec cette base de contraintes. Purger ainsi la base de contraintes nécessite un temps de comptage. À titre indicatif, cette étape de purge a été mesurée à 14 secondes dans notre scénario. La base de contraintes obtenue est notée $\mathcal{D}_{\text{hyb-60 f-1}}$.

`Étape 2` L'analyste lance une requête sur la trace $\mathcal{S}_{cecm}$ avec une fréquence minimale de 2 et la base de contraintes $\mathcal{D}_{\text{hyb-60 f-1}}$.

`Étape 3` Après quelques secondes d'exécution de l'algorithme `CDA` et à la vue des résultats partiels de la fouille affichés dans la vue `Results`, l'analyste s'aperçoit que l'algorithme fouille les chroniques candidates basées sur des épisodes de longueur supérieure à 6 et contenant plusieurs fois le type `nav` (voir le tableau 5.1 pour le nom complet des types). Comme l'analyste souhaite caractériser précisément la navigation vers l'espace personnel, le type d'événement `nav` est trop général car il ne spécifie pas quels espaces sont traversés. C'est pourquoi l'analyste ajoute l'événement `nav` à la liste des types événements interdits (en ajoutant la chronique $(\texttt{nav}, \{\})$ à la contrainte de non-ascendance $\mathcal{C}^{\neg}$) et relance la requête ainsi modifiée.

`Étape 4` Après quelques secondes, l'analyste remarque que l'algorithme passe du temps à fouiller les chroniques de longueur 5 contenant plusieurs fois les types `acc` (navigation vers l'espace d'accueil) et `test` (navigation vers l'espace de test). Pour chacun de ces deux événements, il ne fait pas sens d'en avoir deux dans la chronique recherchée. C'est pourquoi l'analyste empêche qu'il y ait deux fois le type `acc` dans les chroniques découvertes en ajoutant la chronique $(\texttt{acc acc}, \{\})$ à la contrainte $\mathcal{C}^{\neg}$. De même il ajoute la chronique $(\texttt{test test}, \{\})$ à la contrainte $\mathcal{C}^{\neg}$ pour éviter que le type `test` apparaisse deux fois. De plus, l'analyste pose la contrainte de longueur maximale sur les chroniques avec pour valeur 4, afin d'éviter que l'algorithme `CDA` « s'englue » dans la recherche

Nom raccourci	Nom complet
nav	ChangeSpaceEvent
log	LoginEvent
acc	ChangeSpaceEvent:/Accueil
test	ChangeSpaceEvent:/Accueil/Test
dam	ChangeSpaceEvent:/Accueil/Test/Damien

Table 5.1 – Légende des noms de type de la trace $\mathcal{S}_{cecm}$.

de chroniques trop longues. L'analyste relance l'algorithme.

Étape 5 L'analyste observe les résultats partiels et constate que de nombreuses chroniques de navigations diverses sont découvertes. Afin de focaliser la recherche de chroniques sur des chroniques plus intéressantes, l'analyste demande à la plate-forme de trouver des chroniques contenant les types d'événement log et dam (navigation vers l'espace Damien). Pour ce faire, l'analyste ajoute la chronique (log dam, {}) comme contrainte d'ascendance $\mathcal{C}^{\top}$. Il relance l'algorithme.

Étape 6 L'analyste observe dans les résultats partiels des chroniques intéressantes basées sur l'épisode log acc test dam. Ces chroniques sont intéressantes car elles mettent en jeu tous les espaces intermédiaires entre l'accueil et l'espace personnel. Il modifie donc la contrainte d'ascendance $\mathcal{C}^{\top}$ en y ajoutant ces quatre événements. Il relance la requête.

Étape 7 L'algorithme retourne au bout de quelques secondes trois chroniques fréquentes minimales (c'est-à-dire que l'algorithme a fourni tous les résultats). L'analyste observe ces trois chroniques et constate que l'une d'entre elles fait sens dans le cadre de la navigation vers l'espace personnel car elle positionne le type d'événement log en premier, puis la navigation vers l'espace d'accueil (acc), puis la navigation vers l'espace de test (test) et enfin la navigation vers l'espace personnel dam. Cette chronique étant satisfaisante, l'analyste décide de l'enregistrer. Cette chronique peut être considérée comme la chronique Login_Espace_Perso que nous avions introduite dans la trace $\mathcal{S}_{cecm}$ et que l'analyste cherchait à caractériser. Dans le cas où aucune des chroniques minimales retournées ne convient parfaitement à l'analyste, celui-ci aurait pu éditer l'une d'entre elles, puis l'enregistrer, dans le but d'utiliser cette chronique ultérieurement pour faire de la reconnaissance de tâches ou pour faire une transformation d'abstraction de la trace $\mathcal{S}_{cecm}$. La fonctionnalité de Scheme Emerger qui permet à l'analyste de voir en temps réel dans la trace l'ensemble des occurrences de la chronique qu'il édite permet à l'analyste de terminer manuellement et facilement le travail de découverte. La chronique Login_Espace_Perso ainsi découverte est montrée sur la figure 5.15.

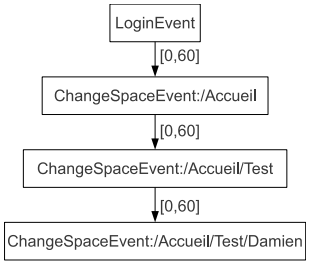


Figure 5.15 – Chronique Login_Espace_Perso finale élaborée avec Scheme Emerger à partir de la trace $\mathcal{S}_{cecm}$ par le processus de découverte interactive de chroniques

La découverte peut se poursuivre sur d'autres étapes par exemple pour découvrir des chroniques de navigation plus longues commençant par la chronique Login_Espace_Perso. Étant donné que

Ét.	f_{seuil}	win	n_{sup}	$\mathcal{C}^{\top}$	$\mathcal{C}^{\neg}$	Results
1	Création de la base de contraintes hybrides, avec $win = 60$					
2	2	60				trop de `nav`
3	2	60			`nav`	trop de `test` et de `dam`
4	2	60			`nav`, `test test`, `dam dam`	l'algorithme « s'englue »
5	2	60	4		`nav`, `test test`, `dam dam`	chroniques de navigation diverses
6	2	60	4	(`log dam`, {})	`nav`, `test test`, `dam dam`	chroniques intéressantes basées sur l'épisode `log acc test dam`
7	2	60	4	(`log acc test dam`, {})	`nav`, `test test`, `dam dam`	`Login_Espace_Perso` découverte

Table 5.2 – Résumé des étapes de découverte interactive de la chronique `Login_Espace_Perso` à partir de la trace $\mathcal{S}_{cecm}$. Pour raccourcir, certaines chroniques sans contraintes temporelles sont notées $\varepsilon_1 \ldots \varepsilon_n$ au lieu de $(\varepsilon_1 \ldots \varepsilon_n, \{\})$ dans ce tableau.

la requête est déjà fortement contrainte par l'analyste à la fin de l'étape 7 par les contraintes $\mathcal{C}^{\neg}$ et $\mathcal{C}^{\top}$, il est possible d'augmenter la longueur maximale n_{sup} des chroniques découvertes pour voir quelles actions suivent les occurrences de la chronique, c'est-à-dire quelles actions l'utilisateur de `CollaborativeECM` effectue lorsqu'il est arrivé dans son espace d'accueil.

De même pour la découverte de la chronique `Recherche_Document`, après quelques itérations du processus de découverte interactive l'analyste découvre la chronique de la figure 5.16. La chronique `Recherche_Document` telle que représentée sur la figure s'interprète comme suit : « une navigation vers l'espace d'accueil (`acc`), suivie d'une navigation vers un espace quelconque (`nav`), suivie d'une navigation vers l'espace d'accueil, suivie d'une navigation vers un espace quelconque ». Il est à noter que les contraintes temporelles de cette chronique `Recherche_Document` sont basées sur l'intervalle $[1, 60]$ au lieu de $[0, 60]$ pour la chronique `Login_Espace_Perso`. L'explication de ceci est que la chronique découverte par l'analyste était initialement la même que la chronique `Recherche_Document` de la figure à l'exception que les contraintes temporelles étaient celles de la base de contraintes, c'est-à-dire $[0, 60]$. Cependant, au lieu d'enregistrer cette chronique telle quelle, l'analyste a choisi d'éditer cette chronique manuellement et de modifier les intervalles des contraintes en $[1, 60]$, afin d'assurer que deux événements successifs de types `acc` et `nav` n'apparaissent à la même date. Cette modification a été effectuée par l'analyste en connaissance des choix effectués lors de la phase de préparation de la trace. L'un de ces choix était de créer pour chaque action de navigation d'un espace à l'autre une paire d'événements telle que le premier événement soit de type `nav` pour représenter la navigation de manière générique et le deuxième de type `ChangeSpace[Espace]` pour la navigation spécifique vers l'espace cible. Si l'analyste n'avait pas remplacé les bornes inférieures des contraintes temporelles de la chronique par des 1, alors `Recherche_Document` aurait eu de nombreuses occurrences sans intérêt construites sur les paires d'événements évoquées ci-dessus résultant de la préparation des traces. D'autres chroniques possibles pour représenter le motif `Recherche_Document` auraient été l'alternance de `acc` et de `nav` trois fois, ou encore quatre fois, mais contrairement aux expressions régulières le formalisme des

chroniques ne permet pas de spécifier les occurrences multiples de certains types d'événement de la chronique.

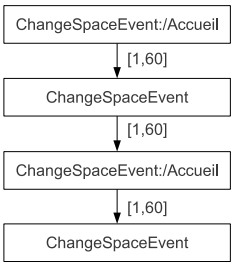


Figure 5.16 – Chronique Recherche_Document finale élaborée avec Scheme Emerger à partir de la trace $\mathcal{S}_{cecm}$ par le processus de découverte interactive de chroniques.

Dans ce scénario de découverte interactive de chroniques comme dans d'autres scénarios (cf. annexe B), les chroniques découvertes ont été élaborées morceau par morceau et étape après étape par l'analyste à partir des résultats proposés par l'algorithme CDA. Avec ce scénario, on s'aperçoit que le processus de découverte interactive de connaissances à partir de traces relève plus de la co-construction de chroniques entre l'analyste et le système par l'intermédiaire de la plate-forme Scheme Emerger que de la découverte à proprement parler (cf. discussion, chapitre 6) .

5.3.4 Autres jeux de données et autres scénarios

Avec ce scénario, il est difficile de montrer l'intérêt du choix du formalisme des chroniques par rapport aux épisodes en série pour représenter les schèmes d'activité. Les chroniques découvertes avec ce scénario n'exploitent pas pleinement les possibilités d'expressivité offertes par le formalisme des chroniques puisque les chroniques que nous avons découvertes sont, à peu de choses près, de simples épisodes en série. Les tâches médiées par un environnement informatique comme la plate-forme CollaborativeECM ne laissent en général que peu de liberté aux utilisateurs quant à l'ordre dans lequel les sous-tâches peuvent être effectuées. En effet, l'utilisateur n'a généralement pas d'autres choix que de suivre les instructions et les formulaires dans l'ordre où ils apparaissent. Pour tirer profit de l'expressivité des chroniques lors de l'analyse des activités collaboratives à partir des traces de la plate-forme CollaborativeECM, il faudrait que les traces contiennent des types d'observé pouvant apparaître dans des ordres différents selon les occurrences d'une même tâche. On ne peut donc pas créer de telles traces à partir de sondes qui observent uniquement l'exécution de la plate-forme informatique qui médie l'activité. Or la plate-forme CollaborativeECM ne comporte à l'heure actuelle que de telles « sondes système », qu'il s'agisse de la source $s_{services}$ d'audit des services internes ou de la source de traçage s_{req} qui trace les pages qui se succèdent.

En revanche, lorsque que l'environnement tracé ne contraint pas trop l'ordre dans lequel les actions de l'utilisateur peuvent être effectuées, il est généralement plus facile de découvrir des schèmes d'activité sous forme de chroniques qui ne soient pas assimilables à des épisodes en série. C'est le cas avec l'activité de conduite automobile (cf. annexes B). En effet, lorsque l'utilisateur d'une voiture conduit, il peut effectuer certaines actions dans l'ordre de son choix. Par exemple, il peut regarder dans son rétroviseur central puis freiner, ou bien freiner puis regarder dans son rétroviseur central[32]. Pourtant, quel que soit l'ordre d'apparition de ces deux événements, il s'agit dans les deux cas de la réalisation d'une tâche de freinage par l'utilisateur de la voiture. Nous

[32] la voiture ne contraint pas l'utilisateur à regarder dans son rétroviseur avant de freiner.

illustrons en annexe B comment la plate-forme `Scheme Emerger` aide à découvrir des chroniques intéressantes à partir des traces de conduite qui ne sont pas de simples épisodes en série.

Chapitre 6

Discussion et perspectives

Sommaire

Dans ce chapitre, nous discutons la contribution algorithmique de cette thèse et le processus plus général de co-construction de connaissances dans lequel il s'inscrit. La discussion s'organisera en quatre parties. Tout d'abord, nous reviendrons sur nos contributions en synthétisant les points importants à retenir et en montrant les points de satisfaction et les questions soulevées qu'il convient maintenant d'étudier. La deuxième partie propose une discussion purement algorithmique sur les limitations de notre algorithme d'extraction de chroniques à partir de traces, les optimisations et extensions possibles. La troisième partie revient sur notre choix d'implémenter

notre analyseur automatique par un solveur du problème d'extraction de chroniques et discute ce choix à la lumière des premières expérimentations et observations que nous avons effectuées sur les traces en provenance des domaines applicatifs. Enfin, la quatrième partie commente le processus de co-construction de connaissances que nous avons proposé et implémenté, et tente de l'évaluer par rapport à son objectif initial de gérer la dynamique de la connaissance.

6.1 Rappel et bilan des contributions

La démarche de recherche à l'origine du travail présenté dans cette thèse trouve sa motivation dans le besoin de créer des Systèmes à Base de Connaissances qui soient pertinents et évolutifs, c'est-à-dire qui soient capables de représenter avec précision et à tout moment le domaine d'application dans lequel il agit. Cette condition d'évolutivité des SBC pose le problème de la gestion de la dynamique de la connaissance, qui est stratégique dans le domaine de l'Ingénierie des Connaissances (IC). Nous avons vu que les travaux sur le cycle du Raisonnement à Partir de l'Expérience Tracée (RàPET), développé au sein de l'équipe SILEX, visent à concevoir un nouveau type de SBC qui permet d'une part l'apprentissage continu de connaissances contextuelles sur une activité, appelées *signatures de tâches*, et d'autre part la réutilisation et l'adaptation de ces connaissances dans le contexte de l'activité en cours, à des fins d'assistance. Deux contributions de recherche sont présentées dans cette thèse. L'une s'inscrit dans le domaine de l'IC et consiste à proposer un processus de co-construction de signatures de tâches pertinentes dans l'optique d'intégrer ces signatures dans un SBC utilisant le RàPET. L'autre contribution s'inscrit dans le domaine de la découverte de connaissances à partir de données et consiste à proposer un algorithme d'extraction complète de signatures de tâche (des chroniques) à partir de traces (des séquences d'événements) visant à implémenter la partie pro-active du processus de co-construction proposé. La plate-forme `Scheme Emerger` a été développée dans l'optique d'implémenter d'abord cet algorithme, puis le processus de construction de connaissances dans lequel il prend part.

6.1.1 Processus interactif de co-construction de connaissances

L'originalité du processus de co-construction de connaissances que nous avons proposé par rapport aux méthodes classiques d'acquisition de connaissances est de partager l'élaboration des connaissances construites entre l'humain et la machine. Cette répartition des rôles entre l'humain et la machine dans la construction est effectuée de manière à bénéficier des points forts de chaque partie, c'est-à-dire d'une part de la capacité de la machine à explorer systématiquement de nouveaux motifs et d'autre part de la capacité humaine à interpréter et à valider les motifs en fonction de leur pertinence dans un contexte donné. Le processus de construction est progressif, il opère par itérations successives mettant en jeu d'une part l'extraction de signatures de tâche à partir de traces et d'autre part les transformations d'abstraction des traces à partir de ces signatures extraites. Autrement dit, les connaissances sont produites par l'humain à l'aide de la participation pro-active de la machine. Les connaissances produites par ce processus sont d'un haut niveau d'abstraction par rapport aux traces premières alimentant le processus, qui décrivent l'activité des utilisateurs avec un niveau d'abstraction bas, dans un langage proche des outils tracés.

Même si nous n'avons pas effectué de validation par l'expérimentation de la qualité des connaissances produites par ce processus (voir la section 6.4.2 pour une discussion sur ce point), les premiers tests effectués sur une seule itération du processus (c'est-à-dire sans transformation d'abstraction) ont montré que l'on était capables de retrouver par ce processus des connaissances dont on savait à l'avance la présence dans les traces.

6.1.2 Algorithme de découverte complète de chroniques à partir de traces

Le comportement pro-actif de la machine dans ce processus est réalisé par l'*analyseur automatique*. Pour implémenter l'analyseur automatique, nous avons décidé de représenter les signatures de tâche par le formalisme des chroniques et nous avons proposé un algorithme complet d'extraction de chroniques à partir de traces. À notre connaissance, cet algorithme est le premier algorithme d'extraction de chroniques à partir de séquences d'événements qui soit complet. L'algorithme proposé explore les chroniques candidates une à une et expose à l'*analyste* (l'humain qui participe au processus de co-construction de connaissances) l'ensemble partiel des chroniques fréquentes en cours de découverte, à la manière d'un algorithme anytime, de telle sorte que le bouclage entre l'analyste et l'analyseur automatique peut se faire en temps réel, et que le processus interactif prend la forme d'une discussion entre l'analyste et l'analyseur automatique.

L'algorithme proposé agit également comme un *framework* de découverte dans la mesure où il est fortement configurable en entrée. Selon la base de contraintes temporelles choisie en entrée, le type de chroniques extraites peut varier. En effet, si l'on utilise la base de contraintes hybrides, les motifs extraits sont des *épisodes hybrides*, une forme simplifiée des chroniques qui est plus facile à lire pour l'humain, qui nécessite moins de temps de calcul et qui produit moins de motifs fréquents en sortie. Avec une base de contraintes possédant des contraintes ayant des bornes numériques variées, comme le sont la base de contraintes complète et la base de contraintes de Duong (2001), les motifs extraits sont de véritables chroniques. L'analyste peut ainsi choisir le type de base de contraintes donnée en entrée selon qu'il souhaite extraire des motifs à expressivité fine (les chroniques) ou des motifs plus résumés (les épisodes hybrides). De plus, l'algorithme supporte un certain nombre de contraintes utilisateur : la fenêtre englobante win, la contrainte de longueur maximale, la contrainte de non ascendance et la contrainte de descendance. Ces contraintes utilisateur permettent à l'analyste de réduire l'espace d'exploration des chroniques et de faire converger l'algorithme plus rapidement vers les chroniques d'intérêt pour lui. Enfin, l'algorithme proposé permet de choisir l'ordre dans lequel les chroniques candidates sont sélectionnées lors de l'exploration. Nous n'avons pas encore implémenté de fonction très élaborée (FIFO, LIFO, sélection aléatoire, sélection par ordre lexicographique) pour la sélection des candidats mais cette possibilité ouvre la porte au guidage de l'exploration des chroniques candidates par des connaissances apprises auprès de l'analyste au fil des itérations.

La complexité du problème de découverte est très grande car les chroniques sont des motifs très structurés. L'extraction totale (c'est-à-dire jusqu'à la fin de l'exécution de l'algorithme) de chroniques par notre algorithme est comparable en durée à l'algorithme existant de Duong, voire légèrement plus rapide. Par le biais de la contrainte de longueur maximale, l'analyste peut agir sur le temps de découverte totale en limitant l'espace d'exploration à des chroniques courtes.

6.1.3 Une plate-forme Open Source et modulaire de découverte de motifs temporels

La plate-forme `Scheme Emerger` a été développée dans le cadre de cette thèse et vise à implémenter, dans un premier temps, l'extraction de motifs temporels à partir de traces guidée par l'analyste. Dans un deuxième temps, elle vise à implémenter le processus de co-construction de signatures de tâches, s'appuyant sur l'extraction de motifs temporels. Ce deuxième objectif n'est pas encore réalisé car il manque à la plate-forme l'interface graphique qui permet de configurer et de visualiser les transformations de traces à partir des signatures de tâche. Cependant, l'API `Java` que nous avons développée permet ces transformations en langage `Java`, donc le processus peut être réalisé mais seulement par l'intermédiaire de la rédaction de code `Java`, ce qui ne permet pas le bouclage

temps réel.

L'API `Java` que nous avons développée pour implémenter `CDA` supporte déjà plusieurs autres algorithmes, comme *WINEPI*, *MINEPI*, *SPADE*, l'extraction de motifs pour des événements persistants et la découverte incomplète de chroniques selon Dousson & Duong. Son architecture modulaire `OSGi` sous forme de plugins a notamment été choisie pour que d'autres contributions puissent relativement simplement être ajoutées et faire de `Scheme Emerger` une plate-forme capitalisant et centralisant un certain nombre de travaux sur la fouille de séquences de la même manière que la plate-forme `Weka`[33] (Hall et al. 2009) centralise de nombreux algorithmes de fouille de données (non temporelles) et que la plate-forme `ProM`[34] (Van Dongen et al. 2005) centralise des algorithmes de *Workflow Mining*.

Au sein de l'équipe SILEX, la plate-forme `Scheme Emerger` est maintenant appliquée à d'autres données séquentielles que les traces d'interaction. Il est en cours d'application à la découverte de la structure d'un morceau musicale à partir de la représentation sous forme de séquences d'événements de celui-ci. Dans le formalisme des séquences d'événements, chaque événement représente une note du morceau, le type d'événement étant le nom de la note et la date étant le nombre d'unités de temps écoulées depuis le début lors de l'apparition de la note[35]. Le formalisme des chroniques est très adapté à la caractérisation fine d'un segment de morceau, car elle peut spécifier l'ordre et le temps précis qui sépare deux notes consécutives. Comme il n'est pas possible de trouver des chroniques longues pour des raisons de complexité algorithmique, le même principe d'abstraction d'occurrences est utilisé. Pour une chronique ayant quatre notes, on remplacera dans la séquence d'événements tous les occurrences de cette chronique par une nouvelle « note abstraite », c'est-à-dire par une note qui n'existe pas en réalité mais qui représente l'enchaînement de ces quatre notes. En procédant ainsi itérativement et fidèlement au cycle d'abstraction que nous avons proposé, il est possible d'extraire d'un morceau non seulement les segments de notes récurrents et fréquents, mais également la structure du morceau à l'échelle macroscopique en décrivant des motifs dans l'ordre d'apparition des segments. Une telle analyse de morceaux de musique est plus appropriée à des morceaux ne comportant pas ou peu de notes simultanées. Des instruments comme la guitare ou le piano peuvent produire plusieurs notes en même temps, mais les instrument tels que la flûte ne peuvent en produire qu'une seule, laissant ainsi apparaître clairement la mélodie qui est jouée. L'annexe E montre comment traduire une partition de musique en une séquence d'événements.

6.2 Limites et optimisation de l'algorithme d'extraction de chroniques proposé

Dans cette section, nous commentons certains aspects de l'algorithme `CDA` d'extraction complète de chroniques à partir de traces que nous avons détaillé au chapitre 4.

6.2.1 Optimiser le comptage des chroniques

En section 4.5.4, nous avons constaté que le traitement limitatif de l'algorithme `CDA`, c'est-à-dire l'opération qui prend l'essentiel du temps de découverte, est le comptage des chroniques candidates (ligne 12 de l'algorithme 3), qui représente entre 97% et 99% du temps de découverte total selon

[33] `http://www.cs.waikato.ac.nz/~ml/weka/`

[34] http ://prom.win.tue.nl/tools/prom/

[35] Une unité de temps correspond à la note la plus courte en durée de tout le morceau. Par exemple, si la note la plus courte est une double croche, alors une unité de temps représente un quart de *temps* dans le tempo du morceau analysé. Ainsi une simple croche durera deux unité de temps, une noire quatre unités de temps, etc.

nos mesures. Dans l'optique de réduire le temps global de découverte, cela donne deux premières pistes. La première consiste à réduire le nombre de chroniques candidates comptées et la deuxième à optimiser la méthode de comptage.

Pour réduire le nombre de chroniques comptées, nous avons déjà mis en place une stratégie de vérification de non-fréquence d'une chronique candidate par l'intermédiaire de l'ensemble `Non_Fréquents`, ou de la fréquence d'une chronique candidate par l'intermédiaire de l'ensemble `Fréquents`. Une autre optimisation consisterait à vérifier avant le comptage que la chronique générée n'est pas inconsistante, puisque cela impliquerait nécessairement qu'elle ne peut pas avoir d'occurrence et donc qu'elle est non-fréquente. Comme nous avons constaté dans la pratique que très peu de chroniques générées sont inconsistantes et que par ailleurs la complexité du test de consistance est $O(n^3)$ (Dechter et al. 1991), où n est la longueur de la chronique, nous avons décidé de ne pas retenir ce test.

Pour la méthode de comptage, nous avons utilisé l'algorithme de reconnaissance CRS de Dousson et al. (1993) et optimisé par Dousson et al. (2007), dont la complexité est $O(K \times N)$, où K est le nombre d'hypothèses (ou d'automates) instanciées lors de la reconnaissance et N la longueur de la trace. CRS effectue donc la reconnaissance des occurrences d'une chronique candidate en effectuant une passe sur la trace. À chaque nouvelle chronique candidate, l'algorithme `CDA` effectue donc une nouvelle passe sur la trace via l'algorithme CRS. Cette manière de faire est assez naïve, car pour déterminer les occurrences d'un motif m en fouille de données, il est généralement préférable de tirer profit des connaissances acquises lors d'une passe antérieure sur un motif m' parent. C'est notamment le principe de comptage par liste d'occurrences, qui est utilisé par *SPADE* dans les bases de fréquences (cf. section 3.2.1) et par *WinMiner* (Méger 2004) pour l'extraction d'épisodes sans borne (cf. section 3.3.3) dans les séquences d'événements. Le problème est que pour la reconnaissance *au plus tôt* des occurrences d'une chronique, qui est le type de reconnaissance que nous avons choisi, car il est supporté par CRS, la méthode par liste d'occurrences ne peut pas s'appliquer. En effet, dans la trace $(A,1)(B,2)(B,4)$, les occurrences de la chronique $\mathcal{C}_1 = A[0,3]B$ sont $\mathcal{O}_1^{CRS} = \{(A,1)(B,2)\}$ et les occurrences de la chronique $\mathcal{C}_2 = A[2,3]B$ sont $\mathcal{O}_2^{CRS} = \{(A,1)(B,4)\}$. On a bien $\mathcal{C}_2 \preceq \mathcal{C}_1$, mais on n'a pas $\mathcal{O}_2^{CRS} \subseteq \mathcal{O}_1^{CRS}$, et il n'existe aucun moyen pour de déterminer $\mathcal{O}_2^{CRS}$ par jointure de $\mathcal{O}_1^{CRS}$ avec quelque autre liste d'occurrences d'une chronique de longueur 2.

Pour pouvoir appliquer le principe de reconnaissance par liste d'occurrences et comparer son efficacité, il faudrait une procédure de reconnaissance qui soit d'une part antimonotone (nécessaire pour les approche de type *Apriori*) et d'autre part qui vérifie la condition $\mathcal{O}_2 \subseteq \mathcal{O}_1$ pour toute paire de chroniques telle que $\mathcal{C}_2 \preceq \mathcal{C}_1$. La procédure de reconnaissance *au plus tôt* ne répond pas au deuxième critère et la reconnaissance d'occurrences minimales de type « *Minepi* » (pour la reconnaissance minimale, on a également $\mathcal{O}_1^{min} = \{(A,1)(B,2)\}$ et $\mathcal{O}_2^{min} = \{(A,1)(B,4)\}$), et donc pas $\mathcal{O}_2^{min} \subseteq \mathcal{O}_1^{min}$ non plus. Dans l'état de nos recherches, la seule procédure de reconnaissance qui vérifie ces deux critères est la procédure de reconnaissance des occurrences totale, c'est-à-dire celle qui définit la mesure de fréquence f_{all} introduite en section 4.1.3. Pour cette procédure de reconnaissance on a $\mathcal{O}_1^{all} = \{(A,1)(B,2),(A,1)(B,4)\}$ et $\mathcal{O}_2^{all} = \{(A,1)(B,4)\}$, et donc on a bien $\mathcal{O}_2^{all} \subseteq \mathcal{O}_1^{all}$. Comme argumenté en section 4.1.3, le problème avec la procédure de reconnaissance totale est qu'elle reconnaît beaucoup plus d'occurrences qu'un observateur humain en dénombrerait. Une piste pour utiliser la méthode de comptage par liste d'occurrences est d'opérer en deux temps pour chaque chronique candidate $\mathcal{C}_2$ générée à partir de sa chronique parent $\mathcal{C}_1$: d'abord déterminer $\mathcal{O}_2^{all}$ par filtrage de $\mathcal{O}_1^{all}$, puis déterminer $\mathcal{O}_2^{CRS}$ à partir de $\mathcal{O}_2^{all}$. Cette méthode nécessite d'une part d'effectuer deux passes sur une liste d'occurrences parent et d'autre part de stocker pour chaque chronique fréquente deux listes d'occurrences : $\mathcal{O}^{all}$ et $\mathcal{O}^{CRS}$, ce qui implique

un encombrement mémoire important, surtout pour l'ensemble $\mathcal{O}^{all}$ qui comporte un grand nombre d'occurrences. Il est nécessaire d'envisager d'utiliser la contrainte *win* pour diminuer fortement le nombre d'occurrences de l'ensemble $\mathcal{O}^{all}$ et de donner une taille raisonnable à l'ensemble $\mathcal{O}^{all}$. Cette méthode n'est pas vraiment dans l'esprit des méthodes par liste d'occurrences telle que *SPADE*, car il n'y a pas de jointure effectuée d'une chronique à sa chronique enfant obtenue par renforcement d'une contrainte temporelle (opérateur $\texttt{str\_}\varepsilon\varepsilon'$), mais uniquement des filtrages successifs sur $\mathcal{O}^{all}$. Pour les calcul des occurrences d'une chronique enfant obtenue par l'ajout d'un type d'événement (opérateur $\texttt{add\_}\varepsilon'$), cette jointure est cependant possible.

Nous avons mesuré que le « goulet d'étranglement » de notre algorithme CDA est le comptage des chroniques candidates dans la séquence d'événements. Il est donc prometteur de tenter cette méthode par listes d'occurrences, afin d'éviter d'effectuer des passes de comptage sur la séquence d'événements et de voir si la vitesse de découverte grandirait sensiblement. Il faudrait également étudier l'impact de cette méthode sur l'occupation de la mémoire, que nous pensons être grand du fait de la taille de l'ensemble $\mathcal{O}^{all}$ pour chaque chronique.

6.2.2 Complexité de l'espace des candidats et complexité réelle

La complexité de l'algorithme CDA est une limitation à l'extraction et la découverte de chroniques dont les longueurs sont grandes. La dépendance exponentielle en n^2 (où n est la longueur de la plus longue chronique trouvée) rend impossible de trouver des chroniques de longueur supérieure à un seuil, qui est de l'ordre de 4 ou 5 selon les traces et les requêtes. Pour être exact, cette complexité est propre à l'espace des chroniques candidates et non pas à l'algorithme CDA. Par ailleurs, dans le cadre de la découverte non-complète de chroniques de Dousson & Duong (1999), cet espace d'exploration présente la même dépendance exponentielle en n^2 et nous avons constaté que l'algorithme CDA était comparable en performance à celui de Dousson & Duong, voire un peu plus rapide.

Ce que nous mettons en avant dans cette discussion est que l'algorithme CDA n'explore en réalité qu'une petite partie de ces chroniques candidates. Pour le comprendre, prenons par exemple une trace de longueur 10, ayant quatre types d'événement différents : A, B, C et D. Prenons le cas le plus critique en nombre de chroniques candidates parcourues, c'est-à-dire le cas où la fréquence seuil est $f_{seuil} = 1$. Le nombre de chroniques fréquentes de longueur 4 dans cette trace est déterminé par le nombre d'occurrences à quatre événements présentes au moins une fois (car $f_{seuil} = 1$) dans cette trace. Ce nombre est le nombre de 4-combinaisons possibles à partir d'un ensemble à 10, c'est-à-dire $C_{10}^4 = 210$. Supposons que la base de contraintes en entrée soit la base de contraintes pour la découverte d'épisodes hybrides (celle de la figure 4.5). Le nombre d'épisodes parallèles de taille 4 qu'il est possible de construire à partir de $\mathbb{E} = \{A, B, C, D\}$ est $C_{|\mathbb{E}|+n-1}^n = C_7^4 = 35$ (cf. section 4.2.7). Pour chacun de ces 35 épisodes parallèles, le nombre de chroniques basées sur cet épisode est $n_{paires\ diff}^3$, où $n_{paires\ diff}$ est le nombre de paires de types d'événement différents de l'épisode parallèle ($n_{paires\ diff} = 0$ pour l'épisode $AAAA$, $n_{paires\ diff} = 4$ pour $AABB$, $n_{paires\ diff} = 6$ pour $ABCD$, etc). À part pour les épisodes $AAAA$, $BBBB$, $CCCC$ et $DDDD$, $n_{paires\ diff}$ vaut au moins 3. Cela veut dire que pour les 31 épisodes parallèles restant le nombre de chroniques basées sur chaque épisode est supérieur à 3^3, c'est-à-dire 27. Donc le nombre de chroniques de longueur 4 qu'il est possible de construire à partir de $\mathbb{E} = \{A, B, C, D\}$ et de cette base de contraintes est largement supérieur à $31 \times 27 = 837$, et *a fortiori* largement supérieur à 210, le nombre de chroniques qui peuvent être fréquentes dans cette trace.

Cette petite comparaison entre le nombre de chroniques présentes dans l'espace des candidats et le nombre maximum de chroniques instanciées dans la trace illustre le fait que la différence

est d'autant plus large que f_{seuil} et $|\mathbb{E}|$ sont grands. Cela signifie que dans la réalité, l'algorithme CDA parcourra bien moins de chroniques candidates que le nombre estimé par notre étude de la complexité. L'ordre de grandeur du temps de découverte complète *totale* (c'est-à-dire jusqu'à la fin de l'exécution de l'algorithme) réel est donc bien en dessous de l'ordre de grandeur $p^{n_{max}^2}$ affiché par la complexité de l'espace d'exploration, ce qui signifie que la complexité de l'algorithme CDA, quoique toujours fortement dépendant en n_{max}, ne sera pas aussi catastrophique qu'attendue au vu de la complexité du problème.

6.2.3 Encombrement mémoire

L'algorithme CDA requiert également un espace mémoire important. En tout, 4 ensembles de chroniques sont maintenus au cours de l'exécution de CDA. Nous avons constaté dans la pratique que les ensembles des chroniques fréquentes minimales `Fréquents` et des chroniques non-fréquentes maximales `Non_Fréquents` contiennent très peu de chroniques (jamais plus de quelques dizaines pour les expérimentations présentées dans cette thèse et effectuées en dehors), ce qui est logique étant donné que lorsqu'une chronique est ajoutée à ces ensembles, elle implique généralement la suppression d'une ou plusieurs chroniques qui deviennent non minimales et non maximales. Nous considérons que ces deux ensembles contribuent fortement à l'efficacité de CDA, puisqu'ils évitent de nombreuses passes sur la trace pour compter des candidates, sans pour autant occuper un espace mémoire important. Les ensembles `Candidats` des chroniques candidates dont il faut tester la fréquence et `Traitées` des chroniques déjà traitées sont de plus en plus volumineux au fil des itérations de CDA, particulièrement `Traitées` qui s'agrandit d'une chronique à chaque itération. Ces deux ensembles sont nécessaires dans la mesure où nous avons décidé de traiter les chroniques candidates dans un ordre laissé au choix de l'analyste (*FIFO*, *LIFO*, aléatoire, lexicographique, etc.). Si l'ordre choisi est l'ordre lexicographique sur les chroniques (cf. section 4.4.2), alors il est possible de n'utiliser aucun de ces deux ensembles. En effet, en procédant en profondeur d'abord « strictement », c'est-à-dire en traitant récursivement les chroniques enfant générées à partir d'une chronique fréquente, alors on pourrait se passer d'un ensemble `Candidats`. Les chroniques enfants générées seraient tout de même gardées en mémoire pour les appels récursifs. En revanche, si les chroniques sont traitées dans l'ordre lexicographique, alors en générant les chroniques enfant d'une chronique fréquente, on pourrait s'assurer de ne pas générer les chroniques enfants qui précèdent cette chronique fréquente pour l'ordre lexicographique.

Le désavantage de fixer un tel ordre lexicographique de parcours des chroniques est que cela fermerait la porte à un ordre de parcours « intelligent », où des connaissances de l'analyste seraient prises en compte pour que les chroniques les plus intéressantes soient les premières traitées. Si l'on dispose de telles connaissances, l'analyste aurait à trancher le compromis suivant avant chaque lancement de l'algorithme CDA : soit il souhaite permettre l'ordre de parcours « intelligent » et dans ce cas l'ensemble `Traitées`, qui occupe beaucoup d'espace mémoire, est utilisé par CDA, soit il ne souhaite pas utiliser ces connaissances et l'ordre de parcours choisi est l'ordre lexicographique, ce qui permet d'économiser l'espace mémoire nécessaire à l'ensemble `Traitées`. Pour l'heure, le programme `Java` qui implémente CDA ne permet pas de choisir la deuxième option. Une légère modification du programme est en effet nécessaire pour pouvoir prendre en compte l'ordre lexicographique sans utiliser l'ensemble `Traitées`.

6.2.4 Post-traitement

Fayyad et al. (1996) expriment l'idée que les motifs recherchés par les algorithmes de fouille doivent posséder une structure simple, de sorte qu'ils soient faciles à comprendre pour l'analyste et qu'ils

représentent un résumé concis d'un ensemble de données potentiellement volumineux et très structuré. Les règles d'association et les épisodes en série et en parallèle rentrent dans cette catégorie. Ces structures peuvent être comprises rapidement et simplement par un individu non spécialiste de la fouille de données, car elles sont relativement intuitives. Les chroniques, nous l'avons vu, ont une structure d'épisode parallèle à laquelle sont ajoutées des contraintes temporelles. Ces informations supplémentaires complexifient la structure de motif et le nombre de motifs de longueur n qu'il est possible de créer pour un même jeu de données est multiplié par un facteur de type p^{n^2}. En recherchant des motifs structurés comme les chroniques, on a d'une part plus de motifs candidats à traiter, mais d'autre part plus de motifs fréquents. Le niveau de description étant plus fin, les différences entre les motifs fréquents trouvés sont plus petites, voire pas toujours significatives. Par exemple on trouvera dans une trace les trois chroniques fréquentes de taille $n = 2$ suivantes : $A[0, 1]B$, $A[1, 2]B$ et $A[2, 3]B$, alors que pour une extraction d'épisodes en série, on n'aurait obtenu qu'un seul motif : l'épisode AB. Plus n est grand, plus le nombre de chroniques fréquentes pour un même épisode est grand. On retrouve ce problème du très grand nombre de motifs extraits avec toute structure de motif qui a une expressivité forte, comme les motifs multi-séquentiels de de Amo & Furtado (2007) basés sur la logique du premier ordre.

Dans ce genre de situation, des motifs nombreux et semblables sont présentés à l'analyste qui peut s'y perdre. Des outils de navigation dans ces motifs (classement, filtrage, etc.) peuvent aider, mais un post-traitement de l'ensemble des motifs trouvés peut également être mis en place pour réduire le nombre des motifs et augmenter leur intérêt et leur niveau de synthèse sur les données qu'ils représentent. Dans le cadre des chroniques fréquentes minimales, nous n'avons pour l'heure rien implémenté dans `Scheme Emerger`, mais nous pensons qu'une fusion de motifs fréquents ayant des contraintes temporelles adjacentes est possible et souhaitable. Avec notre exemple, ce post-traitement fusionnerait les trois chroniques de longueur 2 $A[0, 1]B$, $A[1, 2]B$ et $A[2, 3]B$ en une seule chronique $A[0, 3]B$, qui est plus « synthétique » car la contrainte $A[0, 3]B$ est plus lâche que les trois autres. De même, un regroupement en grappes basé sur une mesure de similarité entre chroniques pourrait aider l'analyste. Cependant, il est à noter que pour les traces et les requêtes que nous avons effectuées dans cette thèse, les chroniques extraites étaient certes nombreuses, mais pas suffisamment pour empêcher de trouver les quelques chroniques pertinentes parmi elles.

6.2.5 Fouille interactive et incrémentale

Nous avons vu au chapitre 3 qu'il existait des techniques pour ne pas avoir à effectuer tous les calculs de nouveau à chaque mise à jour de la trace. Dans nos exemples d'extraction de chroniques, nous avons appliqué notre algorithme `CDA` à des traces immuables, sans se poser le problème incrémental. Dans la réalité, les traces proviennent d'un environnement tracé (par l'intermédiaire d'un SBT), qui alimente la trace à mesure que l'activité se déroule. Dans l'optique de créer des modules utilisant le RàPET pour l'assistance à l'utilisateur de cet environnement on pourrait souhaiter que les signatures de tâche entrant en jeu dans le RàPET (cf. section 2.2.3) soient toujours à jour par rapport à la trace. Dans ce cas, il faudrait modifier l'architecture de l'algorithme `CDA` en reprenant des techniques de fouille incrémentale existant dans la littérature (cf. section 3.6), par exemple en gardant en mémoire des chroniques dont on a mémorisé les occurrences et qui pourraient devenir fréquentes. Cependant, ce besoin de signatures de tâche à jour en temps réel à mesure que la trace évolue ne nous paraît pas évident. Dans l'esprit de l'ingénierie de la dynamique des connaissances que nous ciblons dans cette thèse, si les signatures de tâche ne sont pas à jour au point que le module d'assistance basé sur le RàPET se comporte étrangement, alors l'humain peut relancer un cycle d'apprentissage de nouvelles signatures de tâche avec `Scheme Emerger` à partir de la trace

modifiée. De la sorte, les signatures seraient maintenues régulièrement par intervention humaine, sans être pour autant rigoureusement à jour du point de vue d'une requête de fouille.

Le problème *interactif*, c'est-à-dire le problème de la recherche efficace de chroniques fréquentes pour une requête de fouille R_{k+1} à partir des résultats d'une requête R_k effectuée sur la même trace, nous paraît plus stratégique dans l'optique de réaliser un processus de co-construction de connaissances humain/machine. Le but est d'optimiser le nombre de chroniques intéressantes retournées à l'humain en un minimum de temps, afin d'augmenter la pertinence des réponses de la machine lors du dialogue qui s'instaure. De plus, les requêtes successives effectuées dans le processus itératif de co-construction proposé en section 2.3 par l'analyste sont proches de ce principe et sont ainsi propices à l'utilisation de technique de fouille dite *interactive*. On pourrait tenter d'adapter les techniques existantes pour les épisodes aux chroniques, mais l'utilisation des techniques de fouille interactive nécessite que les résultats de la requête précédente soient tous connues. Notre algorithme `CDA` étant très complexe, nous avons choisi pour celui-ci une architecture de type *anytime* permettant de se satisfaire des résultats partiels pour des requêtes dont la complétude pourrait nécessiter plusieurs heures, jours ou années. Ce choix nous empêche de mettre en place les techniques de fouille interactive existantes telles quelles, bien qu'il soit toujours possible de tirer profit des connaissances acquises lors de requêtes précédentes pour améliorer le traitement de la requête en cours. En résumé, la mise en place de techniques *interactives* d'extraction de chroniques n'est pas compatible avec l'architecture de type *anytime* que nous avons souhaitée pour permettre l'utilisation de l'algorithme `CDA` malgré sa grande complexité.

6.2.6 Extensions de l'algorithme à l'extraction de chroniques encore plus structurées

Des techniques existent pour extraire des motifs temporels à partir de séquences (cf. section 3.4), dont les événements sont plus structurés que le modèle de représentation (ε, t) sur lequel nous nous sommes appuyés. Ces techniques sont intéressantes dans notre cas, car les séquences d'événements constituent une représentation très simplifiée des M-traces et les différentes techniques existantes dans la littérature peuvent permettre de fouiller des traces dont le format se rapproche de celui des M-traces. Un tel format pourrait supporter la présence d'attributs dans les événements, comme les observés, grâce à la fouille de séquences multidimensionnelles, ou bien la présence de relations entre événements comme il existe des relations entre observés (encore que nous ne connaissions pas de travaux sur la fouille de données séquentielles multirelationnelles), ou encore l'extraction de chroniques à partir de séquences d'événements persistants, comme les observés d'une M-trace. Pour tous ces points, on pourrait adapter les extensions existantes pour l'extraction d'épisodes au problème de l'extraction de chroniques. L'inconvénient est que chaque extension implémentée se fera au prix d'une structure de motif encore plus complexe que la structure de chroniques, engendrant une complexité encore plus grande et un nombre encore plus grand de motifs extraits retournés à l'analyste.

Dans la mesure où nous sommes capables de proposer un processus de préparation de M-traces en séquences d'événements (cf. chapitre 5), nous pouvons renoncer à supporter ces extensions coûteuses en natif dans l'algorithme d'extraction. De plus, comme l'algorithme CRS de reconnaissance de chroniques dans la trace supporte la reconnaissance de chroniques plus structurées que celles que nous avons vues (support des chroniques avec attributs et relations vers des entités extérieures), nous proposons d'utiliser l'algorithme `CDA` pour la découverte de contraintes temporelles et d'élaborer « manuellement » les attributs et relations pertinentes en utilisant un comptage de chroniques structurées avec CRS. Cette découverte hybride, à la fois *manuelle* (pour les attributs

et relations) et *automatique* (pour les contraintes temporelle) signifie que l'analyste peut spécifier en tant que contrainte d'entrée les types d'événement qu'il souhaite ainsi que leurs attributs et relations au même titre que les contraintes temporelles souhaitées, mais la partie « générative » de la découverte, c'est-à-dire la partie automatique, se limitera à l'extraction de contraintes temporelles.

6.3 Sur le choix de l'extraction de chroniques pour l'analyseur automatique

Dans cette section, nous discutons le choix de l'extraction complète de chroniques comme représentation du problème à résoudre pour l'analyseur automatique permettant le comportement pro-actif recherché du système dans la co-construction de signatures de tâches. Les discussions ci-dessous reprennent certains éléments abordés précédemment dans la discussion sur l'algorithme, mais du point de vue de l'ingénierie des connaissances plutôt que du point de vue algorithmique.

6.3.1 Complexité algorithmique de l'approche choisie

Nous venons de voir que la complexité algorithmique du problème d'extraction de chroniques choisi pour l'analyseur automatique explose en n, la longueur des chroniques extraites, même si la durée réelle d'extraction des chroniques est en pratique bien plus faible que la complexité $O(p^{n^2})$ de l'espace des chroniques candidats. Après implémentation de l'analyseur automatique et illustration sur les traces de `CollaborativeECM` et ABSTRACT, nous considérons que cette complexité extrême n'est pas un obstacle pour l'implémentation du cycle de co-construction de connaissances présentés en fin de chapitre 2, dans la mesure où ce cycle applique des transformations de traces basées sur les chroniques extraites. Cela nécessite juste que l'humain qui prend part à ce cycle soit informé de cette complexité et qu'il sache qu'il n'est pas possible de trouver des chroniques trop longues (de longueurs supérieures à 4 ou 5) et qu'il ne doit pas le tenter. Trouver des motifs de longueur 3 ou 4 par abstractions successives, cela revient à trouver des motifs plus longs (cf. figure 2.7). Pour l'heure `Scheme Emerger` ne permet pas d'implémenter ces transformations, ce qui ne permet pas d'illustrer notre propos par des chroniques dont les types d'événement sont abstraits, eux-mêmes obtenus par transformation d'événements de plus bas niveau d'abstraction.

6.3.2 Représenter les signatures de tâche par les chroniques

Les signatures de tâche sont des motifs instanciés dans la trace représentant l'exécution d'une tâche par l'utilisateur de l'environnement tracé (cf. chapitre 2). Choisir le problème de l'extraction de chroniques comme problème d'extraction de signatures de tâche pour notre analyseur automatique, c'est choisir de représenter les signatures de tâche par des chroniques. Nous avons argumenté ce choix par le fait que les chroniques ont une forte expressivité temporelle permettant de faire la différence entre deux signatures impliquant les mêmes types d'événement dans des ordres partiellement différents ou avec des constantes de temps différentes. Les chroniques possèdent cependant quelques inconvénients. Les chroniques sont des motifs comportant beaucoup d'informations, et pour des chroniques comportant plusieurs types d'événement et plusieurs contraintes temporelles il est difficile à l'humain qui intervient dans le cycle de co-construction de connaissances de comprendre rapidement les chroniques retournées par l'analyseur automatique. La principale difficulté d'interprétation pour l'humain est de comprendre dans quel ordre doivent apparaître les différents types d'événement de la chronique dans la trace, ce qui n'apparaît pas immédiatement au vu des différentes contraintes. Pour faciliter la compréhension des chroniques dans `Scheme Emerger`, nous

avons implémenté un mécanisme de vue simplifiée sur les chroniques. Dans cette vue simplifiée, les types d'événement de la chronique sont positionnés dans l'espace de telle sorte que pour toute paire de types d'événement $(\varepsilon, \varepsilon')$, si la contrainte temporelle entre $\varepsilon[i^-, i^+]\varepsilon'$ est telle que i^- et i^+ sont positifs, alors ε' est située à droite de ε. Si i^- et i^+ sont tous les deux négatifs alors ε' est à gauche de ε, s'ils sont de signes différents alors et ε' et ε sont placés à la même hauteur, à la manière d'un épisode parallèle[36]. Il est aussi possible de simplifier davantage la visualisation des chroniques en supprimant l'affichage des bornes temporelles. Avec la vue simplifiée de la chronique, l'analyste peut se faire plus rapidement une idée des contraintes d'ordre impliquées par les contraintes temporelles (cf. figure 6.1). Il peut alterner entre vue simplifiée de la chronique et vue complète et revenir aux détails pour une lecture plus fine de la chronique.

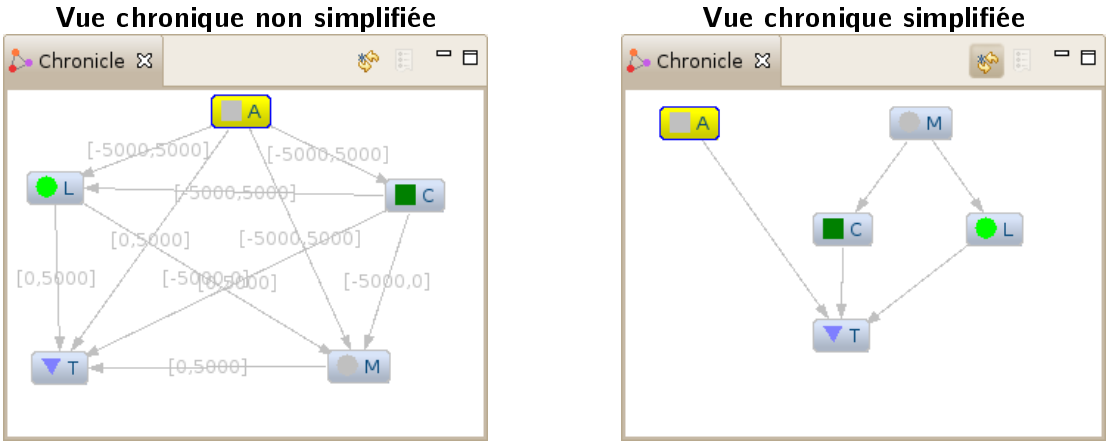


Figure 6.1 – Simplification de la visualisation des chroniques dans `Scheme Emerger`

Une autre difficulté pour l'analyste est la compréhension de certains concepts nécessaires à la bonne conduite de la découverte : contrainte d'ascendance, de non-ascendance, etc. Le concept le plus difficile à appréhender est selon nous la base de contraintes. L'analyste doit comprendre ce qu'est un graphe de contraintes, quelles sont les trois options pour le construire, leurs implications en terme de complexité et de temps d'attente, la purge de cette base de contraintes avant chaque requête, comment les chroniques candidates sont générées à partir de cette base de contraintes, etc. Nous voyons deux possibilités pour contourner ce problème. L'une est de masquer l'existence de cette base de contraintes lors de la découverte pour une utilisation en mode « simplifié » de `Scheme Emerger`, mais cela ne permet pas une conduite experte et optimale de l'extraction. L'autre est de former les utilisateurs de `Scheme Emerger` à ces concepts et à l'algorithme `CDA`.

6.3.3 Simplifier la structure des chroniques

Nous l'avons vu dans la discussion sur l'algorithme lui-même, cette complexité de la structure de chronique ne vaut pas seulement pour l'humain, elle vaut aussi pour le parcours de l'espace des motifs candidats. En choisissant de découvrir des motifs plus simples que les chroniques, comme par exemple des épisodes hybrides, on faciliterait d'une part la compréhension de ces motifs par l'humain et d'autre part on accélérerait le processus d'extraction en diminuant le nombre de candidats. C'est pour cette double raison que pour une première analyse de la trace, nous avons préféré commencer

[36] Le positionnement des types d'événement est effectué automatiquement dans `Scheme Emerger` par des algorithmes de « layout » de graphe, fournis avec le plugin `Zest` d'Eclipse (`http://www.eclipse.org/gef/zest/`).

par la recherche de motifs plus simples : les épisodes hybrides, aussi bien pour les traces du projet ABSTRACT que pour les traces de `CollaborativeECM`.

Sur les traces de `CollaborativeECM`, les chroniques que nous avons découvertes sont même assimilables à des épisodes en série, c'est-à-dire à des motifs imposant un ordre total sur les types d'événement qu'il met en jeu, ce qui pose la question de l'intérêt de rechercher des motifs aussi expressifs que des chroniques. Sur les traces ABSTRACT, nous avons cependant pu extraire des motifs intéressants avec contraintes d'ordre partielles. Nous pensons que plus l'activité tracée est « prescrite » par son environnement et moins il y a d'intérêt à pouvoir exprimer des contraintes d'ordre partielles. Dans le cas de `CollaborativeECM`, les tâches sont ainsi prescrites par la plate-forme. Par exemple, il est impossible de valider un formulaire sans avoir rempli préalablement tel champ. L'ordre des actions est imposé par la plate-forme. Dans une activité de conduite, l'utilisateur est plus libre : il peut contrôler son rétroviseur avant ou après de changer de ligne, même s'il est conseillé de le faire avant.

Dans la majorité des cas où l'on n'a pas besoin de la précision fine que permettent les chroniques avec plusieurs contraintes temporelles numériques, comme c'est le cas avec notre analyse des traces de la plate-forme `CollaborativeECM`, nous pensons qu'il serait judicieux d'extraire des motifs plus simples, comme les épisodes hybrides ou des chroniques n'autorisant par exemple qu'une voire deux contraintes temporelles numériques par motif. En créant un algorithme optimisé pour supporter en natif l'extraction de ces types de motifs plus simples, nous aurions probablement obtenu des performances meilleures que pour l'extraction d'épisodes hybrides en utilisant `CDA`.

6.3.4 Les chroniques co-construites, des connaissances ?

Nous avons vu que ce qui caractérise les connaissances, machine ou humaines, c'est de pouvoir être mobilisées pour agir, et que différents travaux ont montré qu'on peut utiliser des chroniques pour la reconnaissance de situation, aussi bien dans les alarmes d'un réseau de télécommunication (Cordier & Dousson 2000), que dans les comportements des clients (Guillou et al. 2008) et que dans l'activité d'un humain dans un environnement instrumenté (Cram et al. 2009). Lorsqu'une situation s'apparie avec une partie d'une chronique gardée en mémoire par le système, Cram et al. (2009) considèrent que la signature de tâche représentée par la chronique est en train de se jouer dans l'environnement et propose une assistance adaptée à la situation. Un tel système d'assistance à base de signatures de tâche est cependant assez rigide : chaque situation dans l'environnement s'appariant avec une chronique déclenchera l'action d'assistance relative à cette chronique. Nous pensons que chaque situation nouvelle dans l'environnement est différente des situations passées qui s'apparient pourtant à la même signature de tâche. C'est cette singularité supposée de chaque situation que le Raisonnement à Partir de l'Expérience Tracée (RàPET) cherche à prendre en compte. Par analogie au RàPC, de la même manière que chaque nouveau cas est différent des cas passés similaires et nécessite donc une adaptation à partir des cas passés similaires, l'assistance produite par le SBC utilisant le RàPET doit adapter la signature et ses occurences passées à la situation courante. Stuber (2007) formalise un cycle de RàPET qui met en jeu les signatures de tâche en tant que connaissances accumulées par le SBC. Cependant, le formalisme de représentation des signatures de tâche utilisé par Stuber est assez primitif et ne tient pas compte du temps, ce qui rend très imprécise la reconnaissance de situation à partir d'une trace. La capacité du formalisme des chroniques à permettre facilement la reconnaissance totale ou partielle de situations dans les traces fait de lui un très bon candidat à la représentation des signatures de tâche, et élève les chroniques au rang de connaissances dans le cycle de RàPET au même titre que les cas passés sont des connaissances pour un SBC utilisant le RàPC.

6.3.5 Combler le manque d'expressivité des chroniques

Certains manques d'expressivité des chroniques peuvent être traités « en natif » par l'algorithme d'extraction en implémentant certaines extensions existantes dans la littérature (cf. section 6.2.6). D'autres manques peuvent être comblés sans modifier l'algorithme et la représentation sous forme de chroniques. C'est le cas de l'expression de la disjonction entre types d'événement. Nous avons vu au chapitre 3 que les modèles de processus peuvent être considérés comme des motifs temporels et qu'il est possible d'en découvrir à partir de traces. Dans un modèle de processus, par exemple un réseau de Petri, il est possible d'exprimer la disjonction entre événements. Il est par exemple possible de dire : « A, doit être suivi de B ou de C, puis de D ». Le formalisme des chroniques ne permet pas d'exprimer la disjonction. Comme la disjonction est cependant très commode pour représenter fidèlement une situation, nous proposons de représenter la disjonction entre types d'événement en définissant un groupe de plusieurs chroniques représentant tous les cas possibles. Par exemple, notre exemple ci-dessus serait représenté par deux chroniques (équivalentes à des épisodes en série dans ce cas), l'une stipulant « A, suivi de B, suivi de D », et l'autre « A, suivi de C, suivi de D ». Une chronique « disjonctive » est ainsi la disjonction de plusieurs chroniques. C'est ce que nous avons appliqué dans la reconnaissance de tâches lors de la réalisation d'une recette de cuisine (Cram et al. 2009). Un autre manque de représentation pouvant être comblé avec la même stratégie est la prise en compte d'expression régulière. Un motif supportant les expressions régulières permettrait de dire par exemple : « A, suivi de (B suivi de C) n fois, suivi de D », ce qui n'est pas le cas des chroniques. Un tel motif peut être représenté par la disjonction des $p+1$ chroniques suivantes :

- A suivi de D ;
- A, suivi de B, suivi de C, suivi de D ;
- A, suivi de B, suivi de C, suivi de B, suivi de C,suivi de D ;
- ...
- A, suivi de B, suivi de C, ... (p fois), suivi de D,

où p est le nombre maximum de répétitions de B et C qu'il est possible de reconnaître. L'inconvénient de cette approche est qu'il n'est pas possible de supporter un nombre indéterminé de répétitions de sous-motifs. Il faut fixer à l'avance le nombre p de répétitions autorisées et créer les $n+1$ chroniques associées.

6.4 Sur le processus de co-construction de connaissances

6.4.1 Dynamicité de l'approche proposée ?

Nous avons proposé notre cycle de co-construction de connaissances à la fin du chapitre 2, après avoir exprimé la volonté et le besoin de gérer la dynamique de la connaissance. L'approche de construction de traces modélisées abstraites à partir de traces modélisées collectées, par la transformation d'occurrences de signatures de tâche co-élaborées entre l'humain et la machine, semble néanmoins présenter au moins un défaut partagé avec les approches que nous avons qualifiées de « trop statiques » dans le chapitre 1. En effet, avant de pouvoir initier le processus d'abstraction de traces, il faut disposer d'une première trace modélisée : la *trace première*. Nous avons vu que cette trace première est collectée à partir des sources de traçage qui transforment les événements générés par les capteurs de l'environnement tracé en observés de la trace première, c'est-à-dire en

instances de classes d'observé définies dans le modèle de trace de la trace première. Cela signifie que le modèle de trace première doit être créé *a priori*, avant de pouvoir collecter le moindre observé et avant que le processus de co-construction de connaissances puissent débuter. Ce modèle de trace première doit être créé de toute pièce par l'humain, de la même manière que dans une approche d'ingénierie plus « classique » de la connaissance un expert du domaine de connaissances créera un modèle de connaissance permettant à un SBC d'agir dans ce domaine. Cela semble être contradictoire avec la volonté première qui motive notre approche de construction évolutive des connaissances plutôt que *a priori*.

S'il est vrai que ce modèle de trace première doit être créé de toute pièce par l'humain, sa construction par l'humain s'avère en réalité bien moins problématique que la construction d'un modèle de connaissances par un expert du domaine. En effet, dans ce deuxième cas, le modèle de connaissances créé doit décrire le plus fidèlement possible le domaine d'action du SBC pour les tâches intelligentes que le SBC doit réaliser. Les connaissances créées par l'expert doivent être « prêtes à l'emploi » pour le SBC, ce qui signifie que leur niveau de description est déjà abstrait. Pour créer un tel modèle de connaissances abstrait prêt à l'emploi, comme une ontologie, nous avons vu d'une part que le contexte d'utilisation et le point de vue de l'humain qui est en charge de sa construction influent grandement sur la qualité du modèle final, et que d'autre part la moindre modification de l'environnement ou du cahier des charges du SBC nécessite un réingénierie complète du modèle. Cette problématique est plus faible pour la construction du modèle de trace première, car le modèle de trace première est généralement construit par *mapping* des événements produits par les capteurs au format des traces modélisées. Le modèle de trace première décrit l'activité tracée au niveau des interactions de bas niveau. Le modèle de trace première à construire modélise ces interactions de bas niveau, et étant de bas niveau d'abstraction la construction de ce modèle laisse normalement moins de place à l'interprétation et à la subjectivité. Si de nouveaux objets tracés apparaissent dans l'environnement, alors la modification du modèle de trace première est immédiate puisqu'il suffit dans la plupart des cas de simplement rajouter une ou plusieurs classes d'observé correspondant aux événements produits par ce capteur. C'est ensuite le processus interactif et itératif qui gère de manière dynamique la construction des connaissances abstraites qui régiront le comportement du SBC utilisant le RàPET.

6.4.2 Évaluation de notre approche de découverte interactive de connaissances

La question de l'évaluation des SBC est abordée par Bachimont (2004). Pour Bachimont, tout travail d'ingénierie des connaissances (IC) est à la croisée de deux activités de natures différentes. D'une part, l'IC vise à modéliser les connaissances du domaine d'application dans lequel elle s'applique et d'autre part elle consiste à développer les outils théoriques et formelles qui permettent cette modélisation. Bachimont constate qu'il n'existe pas de protocole d'expérimentation propre à l'IC. Soit on considère un travail d'IC du point de vue théorique et formel, auquel cas la validation consiste à prouver la validité et l'efficacité des procédés mis en oeuvre, soit on considère ce travail du point de vue du domaine dans lequel les procédés ont été appliqués, auquel cas la validation consistera à emprunter à la discipline en question les méthodes d'expérimentation et d'évaluation.

Autrement dit, dans notre cas la validation du processus de co-construction de connaissances comporte deux volets. Le premier consiste à valider les techniques formelles qui ont été élaborées et à les évaluer. C'est ce que nous avons fait en prouvant la complétude, la terminaison et en estimant puis en observant la complexité de l'algorithme d'extraction de motif. Le deuxième volet de la validation consisterait à observer les utilisateurs de la plate-forme `CollaborativeECM` avec et sans module d'assistance basé sur le RàPET utilisant les connaissances co-construites. Cela serait la

seule manière d'évaluer par la pratique la pertinence des connaissances co-construites et de valider ou non l'hypothèse à la base de ce travail de thèse stipulant qu'une ingénierie des connaissances qui prend en compte la dynamique des connaissances débouche sur des SBC plus justes et plus évolutifs. Dans le cadre du projet PROCOGEC, la route qui mène à un assistant utilisant ces connaissances est encore longue. Le SBT qui gère les traces de la plate-forme `CollaborativeECM` est arrivé tardivement et aucun module assistant les tâches des utilisateurs basé sur les traces n'a pu être réalisé, à l'exception d'une visualisation générique des traces de la plate-forme `CollaborativeECM`. De plus la plate-forme `CollaborativeECM` n'a toujours pas été déployée en situation réelle chez ses utilisateurs finaux[37], si bien que l'on n'a pas encore pris connaissance des besoins en matière d'assistance. Le consortium du projet PROCOGEC, qui arrive à son terme en juillet 2010, réfléchit actuellement à la poursuite des travaux initiés dans PROCOGEC sur `CollaborativeECM`, y compris la co-construction de signatures de tâche et leur mise en oeuvre dans un SBC utilisant le RàPET, de façon à évaluer en pratique les concepts développés.

Pour l'heure, en l'absence de SBC d'assistance utilisant le RàPET qui serait évalué par expérimentation avec les utilisateurs de `CollaborativeECM`, les seules évaluations que nous avons pu produire consistent d'une part à montrer qu'il est possible de retrouver par notre approche des motifs que nous avions expressément joués dans `CollaborativeECM` et d'autre part à montrer l'intérêt potentiel des chroniques extraites dans le cadre d'un module d'assistance pour `CollaborativeECM`. Pour les traces du projet ABSTRACT, nous avons pu aller un peu plus loin en extrayant de la trace de conduite des motifs d'intérêt, sous forme de chroniques, dont on soupçonnait la présence dans les traces et dont l'expression du motif extrait était novatrice par rapport aux autres représentations du même motif existant dans la littérature scientifique (cf. annexe B).

[37] Une version a été déployé chez le partenaire GDF en juin 2010, mais les tâches réalisées provenaient non pas d'une réelle activité des emplyés de GDF mais à l'exécution de scénarios prévus à l'avance et joués par les utilisateurs.

Conclusion

Les contributions de recherche présentées dans cette thèse concernent deux domaines de recherche : l'ingénierie des connaissances et la découverte de connaissances à partir de données. Des approches et techniques empruntées à ces deux domaines ont été mises en synergie pour aborder la problématique de l'acquisition souple et interactive de connaissances pertinentes, issues de l'expérience, et plus généralement la problématique de l'ingénierie de la dynamique de la connaissance. Nous avons proposé un processus de co-construction de connaissances, entre l'humain et la machine, qui est inspiré des approches d'ingénierie de la connaissance à partir de l'expérience, tel le Raisonnement à Partir de Cas (RàPC), en prenant les traces d'interactions comme conteneurs de connaissances à expliciter. Pour implémenter le comportement pro-actif de la machine dans ce processus de co-construction, nous avons utilisé des techniques de découverte de connaissances à partir de données temporelles et proposé un algorithme d'extraction complète de chroniques à partir de traces. Cet algorithme, développé dans l'optique d'être appliqué aux traces d'interactions, peut également s'appliquer à n'importe quel jeu de données pouvant être représenté sous la forme d'une séquence d'événements. Enfin, la plate-forme `Scheme Emerger` constitue la contribution technologique de la thèse. Cette plate-forme implémente les contributions théoriques proposées en proposant d'une part une interface graphique pour l'algorithme d'extraction complète de chroniques et d'autre part en assistant le processus itératif et interactif de co-construction de connaissances auquel l'algorithme est associé, même si les transformations de traces ne sont pas encore intégrées.

Le processus de construction de connaissances que nous avons proposé permet l'acquisition interactive de connaissances facilitant ainsi la prise en compte de leur dynamique. Cette acquisition de connaissances ne servirait à rien si le Système à Base de Connaissances associé ne prenait pas en compte la dynamique de la connaissance pour les autres tâches de l'IC que sont, entre autres : la capitalisation des connaissances, leur requêtage, leur réutilisation en situation. Les connaissances découvertes par ce processus sont des signatures de tâche, une forme de connaissance contextuelle utilisable par un SBC implémentant le Raisonnement à Partir de l'Expérience Tracée (RàPET) comme principe de raisonnement et d'action en situation. Le RàPET propose en effet un cycle de raisonnement qui permet de prendre en compte cette dynamique de la connaissance en élaborant à la volée les épisodes correspondant à une signature de tâche pour les réutiliser en les adaptant à la situation courante. Les systèmes utilisant le RàPET développés au sein de l'équipe SILEX sont prometteurs en terme de flexibilité d'usage et de capacité à *suivre* les évolutions des connaissances dans les tâches d'assistance à une activité qui évolue.

Du point de vue du domaine de la découverte de connaissances à partir de données, ce que nous retenons de cette thèse est qu'il est avantageux de considérer l'ensemble du processus de découverte afin de concevoir la partie algorithmique de ce processus comme *ouverte* au pilotage par l'analyste qui cherche à découvrir les connaissances à partir de l'expérience tracée.

La recherche sur les algorithmes est fondamentale pour automatiser plus efficacement la recherche de motifs et pour trouver des nouvelles structures de motifs. Toutefois, les travaux publiés

dans ce domaine présentent souvent trop brièvement le contexte d'ingénierie de la connaissance dans lequel ils s'inscrivent. Ce contexte joue pourtant un rôle capital puisque l'on ne demandera pas les mêmes propriétés à un algorithme d'extraction de motifs à partir de données selon les exigences du processus d'ingénierie des connaissances dans lequel il s'intègre. Dans notre approche, nous retenons en particulier un résultat : il est possible d'effectuer la découverte de motifs structurés comme les chroniques au prix d'un espace d'exploration des motifs candidats plus grand, et donc au prix d'une plus grande complexité temporelle de l'algorithme de découverte. Si nous ne considérions pas un processus itératif de construction de signatures de tâche de plus en plus abstraites, il serait difficile de justifier une telle approche qui présente une complexité temporelle très grande. Dans le cadre d'un tel processus, notre algorithme permet au bout de quelques échanges entre la machine et l'humain d'aboutir à des chroniques valables, intéressantes pour l'humain et utilisables par la machine dans le cadre du RàPET.

Annexe A

Comparaison entre l'algorithme de Duong et CDA

Le tableau ci-dessous affiche les résultats comparatifs mesurés en appliquant l'algorithme de découverte de Duong et CDA à la trace $\mathcal{S}_{mannila}$ (cf. figure 3.1) de Mannila et al. (1997). Les mesures ont été effectuées en exécutant ces deux algorithmes avec des fréquences seuil f_{seuil} variant de 10 à 1, avec des contraintes de fenêtre englobante variant de 35 à 5 (de 5 en 5) et avec des notes de couverture minimale de contraintes temporelles $\mathcal{Q}$ variant de 0.1 à 0.9. Au total nous avons donc lancé ces deux algorithmes $10 \times 7 \times 9 = 630$ fois chacun.

Chaque fois que la durée d'exécution de l'un des deux algorithmes dépassait 120 secondes, nous avons interrompu le processus et nous n'avons pas retenu la mesure. Chaque fois que le nombre de chroniques fréquentes était 0 nous n'avons pas non plus retenu la mesure. Au final, 140 mesures ont été retenues, c'est-à-dire 140 combinaisons des trois paramètres f_{seuil}, win et $\mathcal{Q}$ ont donné au moins une chronique fréquente dans un temps de moins de 120 secondes.

Le tableau ci-dessous donne les mesures que nous avons obtenues par ce procédé. Les trois premières colonnes représentent les trois paramètres d'entrée des deux algorithmes. La colonne « $\mathbb{E}_{fréquents}$ » donne le nombre de types d'événement fréquents dans $\mathcal{S}_{mannila}$ pour une valeur de f_{seuil} donnée (en tout il y a 6 types d'événement dans la trace $\mathcal{S}_{mannila}$). La colonne « n_{max} » donne la longueur de la chronique la plus longue de l'ensemble des chroniques fréquentes retournées. La colonne « Nb » donne le nombre de chroniques fréquentes minimales pour chaque exécution. Les colonnes « Duong » et « CDA » donnent respectivement les durées d'exécution (en millisecondes) des algorithmes de Duong et CDA. Enfin, la colonne « Ratio » donne la valeur du ratio $\frac{durée\ Duong}{durée\ \texttt{CDA}}$. Une valeur de ce ratio inférieure à 1 signifie que l'algorithme de Duong a été plus rapide que CDA. Inversement, un ratio supérieur à 1 signifie que CDA a été plus rapide.

Les lignes grisées représentent les exécutions pour lesquelles la durée de l'algorithme CDA était inférieure à 500 millisecondes. Nous avons fait cette distinction car on remarque en général que l'algorithme CDA est plus efficace relativement à celui de Duong lorsque les durées sont plus longues.

f_{seuil}	win	$\mathcal{Q}$	$\mathbb{E}_{fréquents}$	n_{max}	Nb	Duong	CDA	Ratio
5	10	0.7	2	2	1	20	100	0,20
5	10	0.8	2	2	1	15	73	0,21
5	10	0.9	2	2	1	20	68	0,29
5	15	0.6	2	2	1	12	62	0,19
5	15	0.7	2	2	1	10	50	0,20

... continue à la page suivante...

f_{seuil}	win	$\mathcal{Q}$	$\mathbb{E}_{fréquents}$	n_{max}	Nb	Duong	CDA	Ratio
5	15	0.8	2	2	1	10	44	0,23
5	15	0.9	2	2	1	9	42	0,21
5	20	0.4	2	2	1	14	32	0,44
5	20	0.5	2	2	1	9	37	0,24
5	20	0.6	2	2	1	10	41	0,24
5	20	0.7	2	2	1	14	54	0,26
5	20	0.8	2	2	1	8	40	0,20
5	20	0.9	2	2	1	9	52	0,17
5	25	0.4	2	2	1	23	35	0,66
5	25	0.5	2	2	1	16	13	1,23
5	25	0.6	2	2	1	3	29	0,10
5	25	0.7	2	2	1	3	12	0,25
5	25	0.8	2	2	1	3	12	0,25
5	25	0.9	2	2	1	4	11	0,36
5	30	0.4	2	2	1	3	46	0,07
5	30	0.5	2	2	1	3	12	0,25
5	30	0.6	2	2	1	4	18	0,22
5	30	0.7	2	2	1	3	12	0,25
5	30	0.8	2	2	1	3	11	0,27
5	30	0.9	2	2	1	3	24	0,12
5	35	0.4	2	2	1	4	62	0,06
5	35	0.5	2	2	1	3	12	0,25
5	35	0.6	2	2	1	3	13	0,23
5	35	0.7	2	2	1	3	11	0,27
5	35	0.8	2	2	1	2	22	0,09
5	35	0.9	2	2	1	3	12	0,25
4	5	0.4	5	2	1	3	11	0,27
4	5	0.5	5	2	1	3	13	0,23
4	5	0.6	5	2	2	5	25	0,20
4	5	0.7	5	2	3	8	31	0,26
4	5	0.8	5	2	3	7	36	0,19
4	5	0.9	5	2	4	15	76	0,20
4	10	0.3	5	2	2	5	24	0,21
4	10	0.4	5	2	3	8	40	0,20
4	10	0.5	5	3	3	77	143	0,54
4	10	0.6	5	4	3	928	1314	0,71
4	10	0.7	5	4	2	1890	2272	0,83
4	10	0.8	5	4	3	3013	2638	1,14
4	10	0.9	5	4	2	4219	2155	1,96
4	15	0.2	5	2	1	2	8	0,25
4	15	0.3	5	3	2	51	84	0,61
4	15	0.4	5	4	3	986	1146	0,86
4	15	0.5	5	4	3	1089	1250	0,87
4	15	0.6	5	4	4	3753	4005	0,94

. . . continue à la page suivante. . .

f_{seuil}	win	$\mathcal{Q}$	$\mathbb{E}_{fréquents}$	n_{max}	Nb	Duong	CDA	Ratio
4	15	0.7	5	5	1	116725	140089	0,83
4	15	0.8	5	5	1	121116	73169	1,66
4	15	0.9	5	5	1	126259	75815	1,67
4	20	0.2	5	2	2	3	15	0,20
4	20	0.3	5	3	3	56	91	0,62
4	20	0.4	5	4	2	1679	2046	0,82
4	20	0.5	5	4	4	3770	3947	0,96
4	20	0.6	5	5	1	114657	137131	0,84
4	20	0.7	5	5	1	124138	148817	0,83
4	20	0.8	5	5	1	124244	148539	0,84
4	20	0.9	5	5	1	122689	75613	1,62
4	25	0.2	5	2	3	5	23	0,22
4	25	0.3	5	4	3	1088	1235	0,88
4	25	0.4	5	4	4	3652	3926	0,93
4	25	0.5	5	5	1	107521	129942	0,83
4	25	0.6	5	5	1	124654	150163	0,83
4	25	0.7	5	5	1	125330	149364	0,84
4	25	0.8	5	5	1	120227	145847	0,82
4	25	0.9	5	5	1	121723	144795	0,84
4	30	0.2	5	2	3	6	22	0,27
4	30	0.3	5	4	3	1061	1190	0,89
4	30	0.4	5	4	4	3657	3889	0,94
4	30	0.5	5	5	1	114688	138820	0,83
4	30	0.6	5	5	1	124744	148217	0,84
4	30	0.7	5	5	1	124197	149869	0,83
4	30	0.8	5	5	1	117957	141053	0,84
4	30	0.9	5	5	1	115927	138673	0,84
4	35	0.2	5	3	2	51	78	0,65
4	35	0.3	5	4	3	1058	1179	0,90
4	35	0.4	5	5	3	73511	69528	1,06
4	35	0.5	5	5	1	115158	139826	0,82
4	35	0.6	5	5	1	126344	149575	0,84
4	35	0.7	5	5	1	123373	146518	0,84
4	35	0.8	5	5	1	117100	140406	0,83
4	35	0.9	5	5	1	115628	139535	0,83
3	5	0.2	6	2	1	1	7	0,14
3	5	0.3	6	2	2	4	14	0,29
3	5	0.4	6	3	3	50	86	0,58
3	5	0.5	6	3	3	52	86	0,60
3	5	0.6	6	3	3	58	92	0,63
3	5	0.7	6	3	6	158	222	0,71
3	5	0.8	6	4	6	1535	492	3,12
3	5	0.9	6	4	5	1732	288	6,01
3	10	0.1	6	2	1	1	7	0,14

. . . continue à la page suivante. . .

f_{seuil}	win	$\mathcal{Q}$	$\mathbb{E}_{fréquents}$	n_{max}	Nb	Duong	CDA	Ratio
3	10	0.2	6	2	2	3	18	0,17
3	10	0.3	6	3	5	93	155	0,60
3	10	0.4	6	4	4	820	938	0,87
3	10	0.5	6	4	6	2912	3158	0,92
3	10	0.6	6	5	5	21727	26257	0,83
3	10	0.7	6	5	4	72152	76965	0,94
3	15	0.1	6	2	1	1	7	0,14
3	15	0.2	6	3	5	98	157	0,62
3	15	0.3	6	4	3	1038	1275	0,81
3	15	0.4	6	5	11	34807	34494	1,01
3	15	0.5	6	5	6	52147	63245	0,82
3	20	0.1	6	2	2	3	15	0,20
3	20	0.2	6	4	7	780	853	0,91
3	20	0.3	6	5	13	28271	26661	1,06
3	20	0.4	6	5	4	51168	63511	0,81
3	25	0.1	6	2	2	3	15	0,20
3	25	0.2	6	4	6	949	1067	0,89
3	25	0.3	6	5	11	47821	54128	0,88
3	30	0.1	6	2	2	3	14	0,21
3	30	0.2	6	4	6	944	1047	0,90
3	30	0.3	6	5	6	48118	57819	0,83
3	35	0.1	6	2	2	3	15	0,20
3	35	0.2	6	4	10	1892	2096	0,90
3	35	0.3	6	5	6	48583	57911	0,84
2	5	0.1	6	2	1	1	8	0,12
2	5	0.2	6	3	5	40	78	0,51
2	5	0.3	6	4	7	578	538	1,07
2	5	0.4	6	4	14	852	936	0,91
2	5	0.5	6	5	15	2638	2308	1,14
2	5	0.6	6	5	10	5748	5957	0,96
2	5	0.7	6	5	10	23818	23735	1,00
2	5	0.8	6	5	5	57742	18512	3,12
2	5	0.9	6	5	5	64857	6052	10,72
2	10	0.1	6	2	3	5	21	0,24
2	10	0.2	6	4	16	1925	2256	0,85
2	10	0.3	6	6	29	78048	70932	1,10
2	15	0.1	6	4	6	637	762	0,84
2	15	0.2	6	6	56	121971	119457	1,02
2	20	0.1	6	4	10	1257	1429	0,88
2	25	0.1	6	4	18	2678	2936	0,91
2	30	0.1	6	4	18	2724	2940	0,93
2	35	0.1	6	4	18	2718	2882	0,94
1	5	0.1	6	6	15	9737	10462	0,93
1	5	0.2	6	6	16	16320	15249	1,07

. . . continue à la page suivante. . .

f_{seuil}	win	$\mathcal{Q}$	$\mathbb{E}_{fréquents}$	n_{max}	Nb	Duong	CDA	Ratio
1	5	0.3	6	6	18	32648	26964	1,21
1	5	0.4	6	6	18	70910	80275	0,88
1	5	0.5	6	6	17	133265	140521	0,95

FIN

Ce tableau comparatif montre que l'algorithme de Duong et CDA produisent les mêmes chroniques fréquentes minimales pour toutes les mesures avec des durées variables mais généralement équivalentes en fonction des paramètres d'entrée.

	Toutes mesures	$< 500ms$	$> 500ms$
min	0,06	0,06	0,71
max	10,72	6,01	10,72
moy	0,79	0,43	1,11

Table A.2 – Comparaison des ratios en fonction du seuil de durée de 500 millisecondes.

Le tableau A.2 montre un résumé de ces mesures en fonction de la durée totale de découverte. Il montre que pour les découvertes qui ont duré plus de $500ms$, c'est-à-dire pour les découvertes où une différence de durée pourrait être perçue par un utilisateur, CDA est plus rapide en moyenne que l'algorithme de Duong.

Annexe B

Application de `Scheme Emerger` aux traces de conduite instrumentée

Nous illustrons ici la procédure de découverte interactive de schèmes avec l'aide de l'outil `Scheme Emerger` à partir de traces d'activité de conduite automobile. Ce travail est présenté sous la forme d'un extrait d'un article qui a été soumis au journal international *Expert Systems: The Journal of Knowledge Engineering*[38]. Cet extrait montre comment l'analyste de l'activité de conduite peut redécouvrir et caractériser le schème d'un changement de ligne effectué par le conducteur, l'étude de ce schème ayant fait l'objet de beaucoup de travaux publiés dans la littérature sur l'étude des transports. La chronique qui représente le schème découvert dans cet exemple étant un épisode hybride, elle exploite bien la possibilité offerte par les chroniques de spécifier des contraintes temporelles entre certains événements du motif et de ne pas en définir sur certains autres.

Début de l'extrait

We have analyzed two minutes of the trace of a human driving the instrumented car of the INRETS. [The INRETS's instrumented car] tracks many actions of the driver: the steering wheel angle changes, the pressure changes on foot commands, hand commands like indicators and lights, but also the changes of gazing direction thanks to an eye tracker. A first battery of data transformations, like interest point selection in curves, is performed on the raw trace in order to obtain the trace $\mathcal{S}_{driving}$.

We explain how the analyst would use `Scheme Emerger` to discover chronicles in $\mathcal{S}_{driving}$. We assume that the analyst first wants to characterize the scheme "change of lane" in $\mathcal{S}_{driving}$, i.e. schemes that describe all actions needed to move the car from one lane to another. The situation of change of lane is a recurrent case study of driver schemes since a long time in ergonomics (McKnight & Adams 1970) and several systems have been proposed to model the driver behavior in such situations (Salvucci 2004, McCall et al. 2005), but ABSTRACT[39] (Georgeon et al. 2007, Georgeon 2008*b*) is the only one to do that from driver interaction traces (Henning et al. 2007). One issue with the scheme "change of lane" is to detect when the driver crosses the lane intentionally so as to deactivate the LDW (Lane Departure Warning System).

As the analyst does not know much about "change of lane" schemes, he intends to start with hybrid episode discovery rather than the fully expressive complete chronicle discovery. Tab. B.1 gives a summary of the iterative steps explained below.

[38] `http://www.wiley.com/bw/journal.asp?ref=0266-4720`
[39] `http://liris.cnrs.fr/abstract`

`Step 1.` The analyst builds a hybrid constraint database from the constraint database $\mathcal{D}_{hyb}$ wizard with 5000 as a window width constraint win. As the time unit of $\mathcal{S}_{driving}$ is the millisecond, this means that all future discovered chronicles will be $\mathcal{D}_{hyb}$-chronicles and thus contained in a time window of five seconds. The analyst opens a new request editor, sets 2 as the minimum frequency and runs the request.

`Step 2.` The analyst notices that after a few seconds the algorithm has discovered two chronicles: one based on the episode $BBBC$, one on the episode $BBBCC$, where `B` stands for the event type `Close_Left_Down` and C for `Close_Left_Up`. `Close_Left_Up` occurs in the trace when the driver starts to rotate the steering wheel to the left; `Close_Left_Down` means that the steering wheel comes back to its center position from the left. The analyst notices that after a few seconds, the algorithm fixes on these two chronicles, probably because it might be testing with no success different combinations of temporal constraints for their children. According to Formula. 4.1, there are for each length-6 episode $3^{C_6^2} = 14348907$ combinations of temporal constraints of a hybrid constraint database ($p = 3$). That is why the algorithm is probably stuck to length-6 children. The analyst thus decides to set the n_{sup} constraint to 4. Furthermore, the analyst decides that B (`Close_Left_Down`) is an interesting event type for "change of lane" chronicles and tells the system that B must appear in the results by adding B to the inclusion constraint (cf. Figure B.1). Similarly, it is not significant to have twice the event type B in the same chronicle for a change of lane. The analyst adds the chronicle BB (with no temporal constraint) as a chronicle that must not occur in the results. This way, the analyst tells the algorithm to search for chronicles that have exactly one B.

`Step 3.` The analyst notices that the algorithm returns too many chronicles having twice the event C and constraints the algorithm to return chronicles with one C only as he did with B. The analyst runs the request again.

`Step 4.` The analyst prevents the 2-chronicle DD and the 1-chronicle A from appearing in the results because it makes no sense, and runs the request again.

`Step 5.` The analyst observes that many chronicles based on the episode $BCDE$ are frequent (D =`Close_Right_Down` and E =`Close_Right_Up`) and decides that it makes really sense to have these events in the final results and constraints the algorithm to discover chronicles that are stricter than $BCDE$ thanks to the inclusion constraints. The analyst runs the algorithm again.

`Step 6.` The analyst explores the results and notices that the chronicle $(BCDE, \{B[-5000, 0]C\}$ is frequent and has the most sense-making occurrences in the trace. He edits this chronicle from the result view by double clicking on it, adds the temporal constraint $E[-5000, 0]D$ to the chronicle and stores it to the file system. We name this chronicle $\mathcal{C}_{lane}$ (cf. framed sub-chronicle of Figure B.3). It could be read as "A rotation of the steering wheel to the left, followed by a return to the straight position, and (before or after) a rotation of the steering wheel to the right, followed by a return to the straight position". This discovered chronicle and its occurrences are shown on Figure B.2.

`Further steps.` The analyst may still not be able to take advantage of the chronicle $\mathcal{C}_{lane}$ to predict the driver's intention of lane changing. To go further with the analysis and try to discover predictor elements of lane changing, the analyst can continue this iterative discovery process and set $\mathcal{C}_{lane}$ as the inclusion constraint, and relax the n_{sup} constraint to 5 or 6 since the request is already strongly constrained with the inclusion of $\mathcal{C}_{lane}$. With this new request, after a particular focus on the event type `Far_Left` (driver gazing back far at the left), and after some `Scheme Emerger` iterations, the analyst will find in $\mathcal{S}_{driving}$ that two occurrences of the event type `Far_Left` preceding $\mathcal{C}_{lane}$ is a good predictor of an imminent change of lane (cf. Figure B.3).

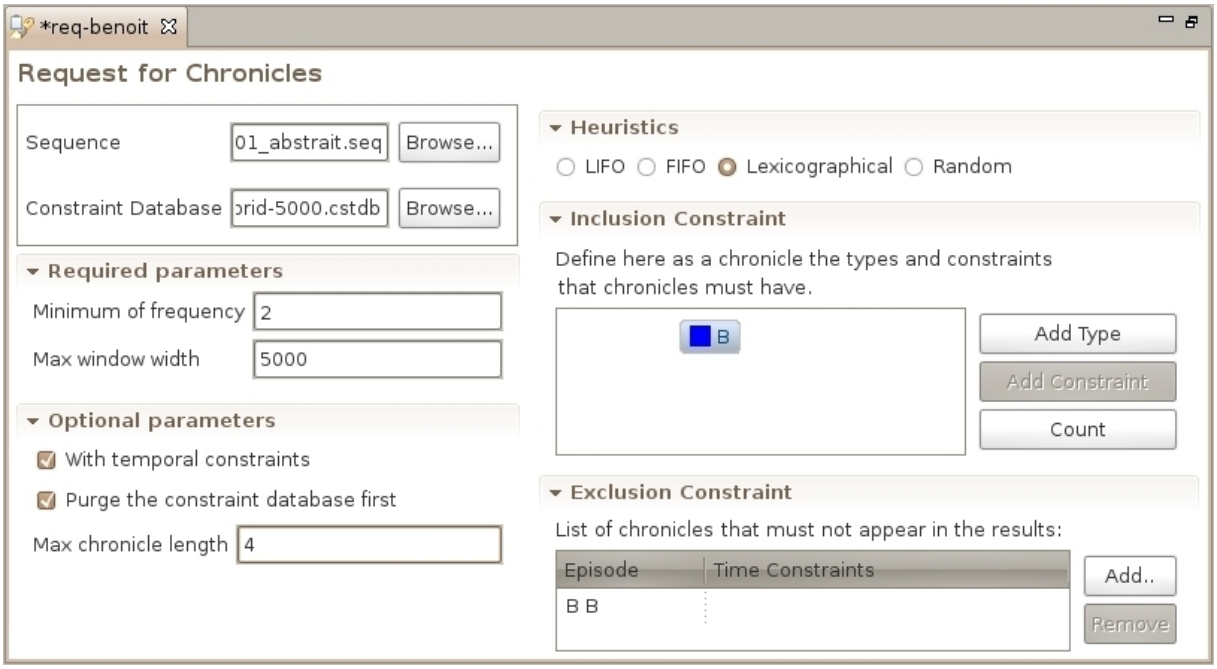


Figure B.1: The request editor of `Scheme Emerger` after adding inclusion constraint, the exclusion constraint and the n_{sup} constraint. The above mining request applies on the trace `tS15-L2-4_TP-p01_abstrait.seq` with the constraint database `cst-db/hybrid-5000.cstdb` as input. It requests for all minimal chronicles that have at least two occurrences and that are contained in a time window of 5000 time units (i.e. milliseconds). The section "Optional parameters" sets the n_{sup} constraint to 4, asks for chronicles with temporal constraints (i.e. chronicles and not parallel episodes), and asks for purging all non frequent temporal constraints for the constraint database if any before running the chronicle discovery process. The section "Heuristics" allows the analyst to choose in which order chronicles are processed, i.e. in which order they are taken from the `Fréquents` set (line 6 of Algo. 3). The section "Inclusion Constraint" defines the parent chronicle of all returned chronicles. For instance on the figure, all returned chronicles must have at least one `B`. The section "Exclusion Constraint" defines a list of chronicles that must have no stricter chronicles in the results. For instance on the figure, the analyst forbids the discovery algorithm to explore chronicles that are stricter than the episode `BB` with no temporal constraint. In other words, it refrains discovered chronicles from containing two or more `B`s.

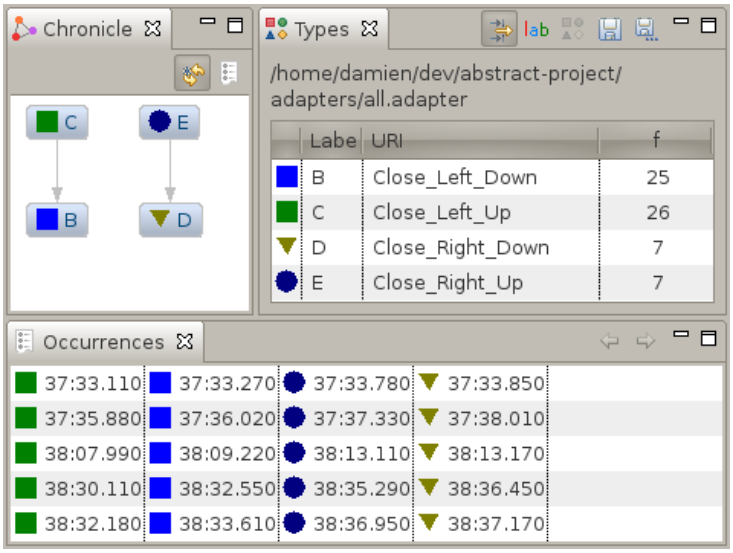


Figure B.2: $\mathcal{C}_{lane}$ and its occurrences.

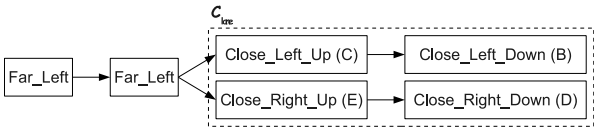


Figure B.3: The discovered chronicle for a change of lane ($\mathcal{C}_{lane}$) and its predictors `Far_Left`.

Step	f_{seuil}	win	n_{sup}	Incl.	Excl.	Results
1	2	5000	none	none	none	too many chronicles with two Bs
2	2	5000	4	B	BB	too many chronicles with two Cs
3	2	5000	4	BC	BB, CC	Two many results with two Ds, A makes no sense
4	2	5000	4	BC	BB, CC, DD, A	BCDE makes sense
5	2	5000	4	BCDE	BB, CC, DD, A	$(BCDE, \{B[-5000, 0]C\})$ is frequent and sense-making
6	$(BCDE, \{B[-5000, 0]C\}$ is edited and modified into $\mathcal{C}_{lane}$ manually, then stored					
7+	2	> 5000	> 4	search for predictors of $\mathcal{C}_{lane}$		

Table B.1: Summary of requests and results for each step of the iterative discovery of car driving schemes with `Scheme Emerger`.

We stop the scenario here, but there are many other things the analyst can do with `Scheme Emerger`. First, we notice in the occurrence set of Figure B.2 that the driver always changes to the left lane (left rotation followed by right rotation), and never to the right, which makes the analyst

think that the way the counting algorithm CRS recognizes the occurrences (cf. Section 4.1.3) does not match the expected occurrences. To improve the pertinence, the analyst could decide to define two different chronicles, one for changing to the left lane specifically, and the other for the right, both being descendant of $\mathcal{C}_{lane}$. To do that, the analyst can decide to stop searching for hybrid episodes and to start searching more expressive chronicles, by first generating the complete constraint database as explained in Section 4.3.2 and setting it as the input constraint database, and then setting $\mathcal{C}_{lane}$ as the inclusion constraint. This would lead to chronicles that have more precise temporal constraints and that would have accurate recognized occurrences.

Fin de l'extrait

Annexe C

Exemple de séquence d'événements : la trace `CollaborativeECM`

Cet annexe donne le contenu de la trace $\mathcal{S}_{cecm}$ qui est analysée pour la découverte de chronique avec `Scheme Emerger`. La figure C.1 donne le résumé des 32 types d'événement qui apparaissent dans la trace $\mathcal{S}_{cecm}$. Au total, la trace $\mathcal{S}_{cecm}$ contient 261 événements.

Types

/home/damien/dev/cecm-project/adapters/default.adapter

Label	URI	f
sp	ChangeSpaceEvent	93
spa	ChangeSpaceEvent:/Accueil	25
spd	ChangeSpaceEvent:/Accueil/Test/Damien	24
spt	ChangeSpaceEvent:/Accueil/Test	23
s	SearchEvent	23
r	ReadDocEvent	14
cr	CreateDocEvent	9
login	LoginEvent	9
spus	ChangeSpaceEvent:/Accueil/Espaces Utilisateurs	5
sptst	ChangeSpaceEvent:/Accueil/Test/Damien/Damien Test	5
spbib	ChangeSpaceEvent:/Accueil/Test/Damien/Biblio Chroniques	4
rcr	ReadDocEvent:/Accueil/Test/Damien/CR_09-04-2010.ods	4
spinv	ChangeSpaceEvent:/Accueil/Espace Invité	3
rcst	ReadDocEvent:/Accueil/Test/Damien/Contraintes_de_reunion___bus.html	3
spdimo	ChangeSpaceEvent:/Accueil/dimocode	2
rdous	ReadDocEvent:/Accueil/Test/Damien/Biblio Chroniques/DOUSSON_2007_Chronicle Recognition Improvement using Temporal focusing and Hierarchization.pdf	2
spdcr	ChangeSpaceEvent:/Accueil/Espaces Utilisateurs/dcram	1
spma	ChangeSpaceEvent:/Accueil/modelArchive	1
cragr	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/AGRAWAL_95_Mining Sequential Patterns.pdf	1
crd93	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/DOUSSON_1993_Situation Recognition-- Representation and Algorithms.pdf	1
crd99	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/DOUSSON_1999_Discovering Chronicles with Numerical Time Constraints from Alarm Logs for Monit	1
crd07	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/DOUSSON_2007_Chronicle Recognition Improvement using Temporal focusing and Hierarchization.p	1
crlax	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/LAXMAN_2006_A survey of temporal data mining.pdf	1
crman	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/MANNILA_1997_DMKD_Discovery of Frequent Episodes in Event Sequences.pdf	1
crduo	CreateDocEvent:/Accueil/Test/Damien/Biblio Chroniques/VUDUONG_2001_PhD_découverte-de-chroniques-à-partir-de-journaux-d'alarmes.pdf	1
crcst	CreateDocEvent:/Accueil/Test/Damien/Contraintes_de_reunion___bus.html	1
crhor	CreateDocEvent:/Accueil/Test/Damien/horaires_ligne_9.pdf	1
rd93	ReadDocEvent:/Accueil/Test/Damien/Biblio Chroniques/DOUSSON_1993_Situation Recognition-- Representation and Algorithms.pdf	1
rman	ReadDocEvent:/Accueil/Test/Damien/Biblio Chroniques/MANNILA_1997_DMKD_Discovery of Frequent Episodes in Event Sequences.pdf	1
rhor	ReadDocEvent:/Accueil/Test/Damien/horaires_ligne_9.pdf	1
ract	ReadDocEvent:/Accueil/Test/activation tracage.doc	1
rpub	ReadDocEvent:/Accueil/dimocode/communautes/nouvelle_expertise-1/Fichiers/pub-c8c385de-e9c8-4f50-96d6-2eb391232a71.html	1
cg	AuditImport	0
A	AuthenticationService.authenticate	0

Figure C.1 – Liste des types de la trace $\mathcal{S}_{cecm}$ et de leur nombre d'occurrences

La figure ci-dessous donne le contenu de la trace $\mathcal{S}_{cecm}$, dans le format de `Scheme Emerger`.

Pour réutiliser cette trace dans la plate-forme `Scheme Emerger`, il suffit d'effectuer un copier-coller des lignes ci-dessous dans un fichier dont le nom se termine par « `.txt` » et de le placer dans le répertoire `sequences` du projet `Scheme Emerger`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<sequence time-unit="second">
  <event time="0" type="ChangeSpaceEvent"/>
  <event time="0" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="109" type="ChangeSpaceEvent"/>
  <event time="109" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="123" type="ChangeSpaceEvent"/>
  <event time="123" type="ChangeSpaceEvent:/Accueil"/>
  <event time="448" type="LoginEvent"/>
  <event time="692" type="ChangeSpaceEvent"/>
  <event time="692" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="702" type="ChangeSpaceEvent"/>
  <event time="702" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="751" type="CreateDocEvent"/>
  <event time="751" type="CreateDocEvent:/Accueil/Test/Damien/horaires_ligne_9.pdf"/>
  <event time="756" type="SearchEvent"/>
  <event time="1051" type="ChangeSpaceEvent"/>
  <event time="1051" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1168" type="ChangeSpaceEvent"/>
  <event time="1168" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1360" type="ChangeSpaceEvent"/>
  <event time="1360" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1378" type="ChangeSpaceEvent"/>
  <event time="1378" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1386" type="ChangeSpaceEvent"/>
  <event time="1386" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1778" type="ChangeSpaceEvent"/>
  <event time="1778" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1787" type="ReadDocEvent"/>
  <event time="1787" type="ReadDocEvent:/Accueil/Test/Damien/CR_09-04-2010.ods"/>
  <event time="1832" type="ReadDocEvent"/>
  <event time="1832" type="ReadDocEvent:/Accueil/Test/Damien/CR_09-04-2010.ods"/>
  <event time="2727" type="ReadDocEvent"/>
  <event time="2727" type="ReadDocEvent:/Accueil/Test/activation␣tracage.doc"/>
  <event time="2810" type="ChangeSpaceEvent"/>
  <event time="2810" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="3382" type="ChangeSpaceEvent"/>
  <event time="3382" type="ChangeSpaceEvent:/Accueil"/>
  <event time="3395" type="ChangeSpaceEvent"/>
  <event time="3395" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="3400" type="ChangeSpaceEvent"/>
  <event time="3400" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="4208" type="CreateDocEvent"/>
  <event time="4208" type="CreateDocEvent:/Accueil/Test/Damien/Contraintes_de_reunion___bus.html"/>
  <event time="4209" type="SearchEvent"/>
  <event time="4631" type="ReadDocEvent"/>
  <event time="4631" type="ReadDocEvent:/Accueil/Test/Damien/CR_09-04-2010.ods"/>
  <event time="4644" type="ReadDocEvent"/>
  <event time="4644" type="ReadDocEvent:/Accueil/Test/Damien/horaires_ligne_9.pdf"/>
  <event time="4678" type="ReadDocEvent"/>
  <event time="4678" type="ReadDocEvent:/Accueil/Test/Damien/Contraintes_de_reunion___bus.html"/>
  <event time="4679" type="ReadDocEvent"/>
  <event time="4679" type="ReadDocEvent:/Accueil/Test/Damien/Contraintes_de_reunion___bus.html"/>
  <event time="63848" type="ChangeSpaceEvent"/>
  <event time="63848" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="64132" type="ChangeSpaceEvent"/>
  <event time="64132" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="64165" type="ChangeSpaceEvent"/>
  <event time="64165" type="ChangeSpaceEvent:/Accueil"/>
  <event time="64206" type="ReadDocEvent"/>
  <event time="64206" type="ReadDocEvent:/Accueil/dimocode/communautes/nouvelle_expertise-1/Fichiers/pub-c8c..."/>
  <event time="65470" type="LoginEvent"/>
  <event time="75971" type="ChangeSpaceEvent"/>
  <event time="75971" type="ChangeSpaceEvent:/Accueil"/>
  <event time="76019" type="ReadDocEvent"/>
  <event time="76019" type="ReadDocEvent:/Accueil/Test/Damien/Contraintes_de_reunion___bus.html"/>
  <event time="84997" type="ChangeSpaceEvent"/>
  <event time="84997" type="ChangeSpaceEvent:/Accueil"/>
  <event time="85118" type="ChangeSpaceEvent"/>
  <event time="85118" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="85143" type="ChangeSpaceEvent"/>
  <event time="85143" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="85252" type="ReadDocEvent"/>
  <event time="85252" type="ReadDocEvent:/Accueil/Test/Damien/CR_09-04-2010.ods"/>
  <event time="85440" type="ChangeSpaceEvent"/>
  <event time="85440" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="325775" type="ChangeSpaceEvent"/>
  <event time="325775" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1268583" type="LoginEvent"/>
  <event time="1268583" type="ChangeSpaceEvent"/>
  <event time="1268583" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1268608" type="ChangeSpaceEvent"/>
  <event time="1268608" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1268695" type="LoginEvent"/>
  <event time="1268757" type="ChangeSpaceEvent"/>
  <event time="1268757" type="ChangeSpaceEvent:/Accueil/Test"/>
```

```
  <event time="1274679" type="LoginEvent"/>
  <event time="1289805" type="ChangeSpaceEvent"/>
  <event time="1289805" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1289812" type="ChangeSpaceEvent"/>
  <event time="1289812" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1289864" type="SearchEvent"/>
  <event time="1289870" type="ChangeSpaceEvent"/>
  <event time="1289870" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1289874" type="ChangeSpaceEvent"/>
  <event time="1289874" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1289877" type="ChangeSpaceEvent"/>
  <event time="1289877" type="ChangeSpaceEvent:/Accueil/Test/Damien/Damien␣Test"/>
  <event time="1289895" type="ChangeSpaceEvent"/>
  <event time="1289895" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1289898" type="ChangeSpaceEvent"/>
  <event time="1289898" type="ChangeSpaceEvent:/Accueil/Test/Damien/Damien␣Test"/>
  <event time="1289932" type="ChangeSpaceEvent"/>
  <event time="1289932" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1289959" type="ChangeSpaceEvent"/>
  <event time="1289959" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1289962" type="ChangeSpaceEvent"/>
  <event time="1289962" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1290104" type="ChangeSpaceEvent"/>
  <event time="1290104" type="ChangeSpaceEvent:/Accueil/Test/Damien/Damien␣Test"/>
  <event time="1290223" type="ChangeSpaceEvent"/>
  <event time="1290223" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1290226" type="ChangeSpaceEvent"/>
  <event time="1290226" type="ChangeSpaceEvent:/Accueil/Test/Damien/Damien␣Test"/>
  <event time="1290245" type="ChangeSpaceEvent"/>
  <event time="1290245" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1290247" type="ChangeSpaceEvent"/>
  <event time="1290247" type="ChangeSpaceEvent:/Accueil/Test/Damien/Damien␣Test"/>
  <event time="1290250" type="ChangeSpaceEvent"/>
  <event time="1290250" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1290274" type="SearchEvent"/>
  <event time="1290293" type="SearchEvent"/>
  <event time="1290296" type="ChangeSpaceEvent"/>
  <event time="1290296" type="ChangeSpaceEvent:/Accueil/Test/Damien/Biblio␣Chroniques"/>
  <event time="1290554" type="CreateDocEvent"/>
  <event time="1290554" type="CreateDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/DOUSSON_1993_Situation␣Rec..."/>
  <event time="1290555" type="SearchEvent"/>
  <event time="1290607" type="CreateDocEvent"/>
  <event time="1290607" type="CreateDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/DOUSSON_2007_Chronicle␣Rec..."/>
  <event time="1290607" type="SearchEvent"/>
  <event time="1290655" type="CreateDocEvent"/>
  <event time="1290655" type="CreateDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/DOUSSON_1999_Discovering␣Chr..."/>
  <event time="1290656" type="SearchEvent"/>
  <event time="1290750" type="CreateDocEvent"/>
  <event time="1290750"
type="CreateDocEvent:/Accueil/Test/Damien/Biblio
Chroniques/VUDUONG_2001_PhD_Decouverte␣de␣chronique␣a␣partir␣de
journaux␣d-alarmes.pdf"/>
  <event time="1290750" type="SearchEvent"/>
  <event time="1290863" type="CreateDocEvent"/>
  <event time="1290863" type="CreateDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/AGRAWAL_95_Mining␣Sequential..."/>
  <event time="1290863" type="SearchEvent"/>
  <event time="1290931" type="CreateDocEvent"/>
  <event time="1290931" type="CreateDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/LAXMAN_2006_A␣survey␣of␣te..."/>
  <event time="1290932" type="SearchEvent"/>
  <event time="1291054" type="CreateDocEvent"/>
  <event time="1291054" type="CreateDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/MANNILA_1997_DMKD_Discovery␣of..."/>
  <event time="1291055" type="SearchEvent"/>
  <event time="1291104" type="ChangeSpaceEvent"/>
  <event time="1291104" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1291140" type="ChangeSpaceEvent"/>
  <event time="1291140" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1291147" type="ChangeSpaceEvent"/>
  <event time="1291147" type="ChangeSpaceEvent:/Accueil/dimocode"/>
  <event time="1291164" type="ChangeSpaceEvent"/>
  <event time="1291164" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1291167" type="ChangeSpaceEvent"/>
  <event time="1291167" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1291172" type="ChangeSpaceEvent"/>
  <event time="1291172" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1291176" type="ChangeSpaceEvent"/>
  <event time="1291176" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1291177" type="ChangeSpaceEvent"/>
  <event time="1291177" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1291195" type="SearchEvent"/>
  <event time="1291220" type="SearchEvent"/>
  <event time="1291238" type="ChangeSpaceEvent"/>
  <event time="1291238" type="ChangeSpaceEvent:/Accueil/modelArchive"/>
  <event time="1291243" type="ChangeSpaceEvent"/>
  <event time="1291243" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1291246" type="ChangeSpaceEvent"/>
  <event time="1291246" type="ChangeSpaceEvent:/Accueil/Espace␣Invite"/>
  <event time="1293203" type="LoginEvent"/>
  <event time="1293203" type="ChangeSpaceEvent"/>
  <event time="1293203" type="ChangeSpaceEvent:/Accueil/Espace␣Invite"/>
  <event time="1293207" type="ChangeSpaceEvent"/>
  <event time="1293207" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293221" type="ChangeSpaceEvent"/>
```

```
  <event time="1293221" type="ChangeSpaceEvent:/Accueil/Espaces␣Utilisateurs"/>
  <event time="1293230" type="ChangeSpaceEvent"/>
  <event time="1293230" type="ChangeSpaceEvent:/Accueil/Espaces␣Utilisateurs/dcram"/>
  <event time="1293237" type="ChangeSpaceEvent"/>
  <event time="1293237" type="ChangeSpaceEvent:/Accueil/Espaces␣Utilisateurs"/>
  <event time="1293239" type="ChangeSpaceEvent"/>
  <event time="1293239" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293243" type="ChangeSpaceEvent"/>
  <event time="1293243" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293247" type="ChangeSpaceEvent"/>
  <event time="1293247" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1293253" type="LoginEvent"/>
  <event time="1293253" type="ChangeSpaceEvent"/>
  <event time="1293253" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1293259" type="ChangeSpaceEvent"/>
  <event time="1293259" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293278" type="LoginEvent"/>
  <event time="1293278" type="ChangeSpaceEvent"/>
  <event time="1293278" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293282" type="ChangeSpaceEvent"/>
  <event time="1293282" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293284" type="ChangeSpaceEvent"/>
  <event time="1293284" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1293290" type="ChangeSpaceEvent"/>
  <event time="1293290" type="ChangeSpaceEvent:/Accueil/Test/Damien/Biblio␣Chroniques"/>
  <event time="1293299" type="ChangeSpaceEvent"/>
  <event time="1293299" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293313" type="SearchEvent"/>
  <event time="1293319" type="ChangeSpaceEvent"/>
  <event time="1293319" type="ChangeSpaceEvent:/Accueil/Espaces␣Utilisateurs"/>
  <event time="1293322" type="ChangeSpaceEvent"/>
  <event time="1293322" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293324" type="ChangeSpaceEvent"/>
  <event time="1293324" type="ChangeSpaceEvent:/Accueil/Espace␣Invite"/>
  <event time="1293327" type="ChangeSpaceEvent"/>
  <event time="1293327" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293330" type="ChangeSpaceEvent"/>
  <event time="1293330" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293333" type="ChangeSpaceEvent"/>
  <event time="1293333" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1293337" type="ChangeSpaceEvent"/>
  <event time="1293337" type="ChangeSpaceEvent:/Accueil/Test/Damien/Biblio␣Chroniques"/>
  <event time="1293339" type="ChangeSpaceEvent"/>
  <event time="1293339" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293348" type="ChangeSpaceEvent"/>
  <event time="1293348" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293370" type="ChangeSpaceEvent"/>
  <event time="1293370" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293384" type="ChangeSpaceEvent"/>
  <event time="1293384" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293408" type="SearchEvent"/>
  <event time="1293424" type="ChangeSpaceEvent"/>
  <event time="1293424" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293434" type="SearchEvent"/>
  <event time="1293447" type="SearchEvent"/>
  <event time="1293538" type="ReadDocEvent"/>
  <event time="1293538" type="ReadDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/DOUSSON_2007_Chronicle␣Recogni..."/>
  <event time="1293552" type="ReadDocEvent"/>
  <event time="1293552" type="ReadDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/DOUSSON_1993_Situation␣Recogni..."/>
  <event time="1293592" type="SearchEvent"/>
  <event time="1293601" type="ReadDocEvent"/>
  <event time="1293601" type="ReadDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/MANNILA_1997_DMKD_Discovery␣of..."/>
  <event time="1293633" type="ChangeSpaceEvent"/>
  <event time="1293633" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293635" type="ChangeSpaceEvent"/>
  <event time="1293635" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293644" type="SearchEvent"/>
  <event time="1293683" type="ChangeSpaceEvent"/>
  <event time="1293683" type="ChangeSpaceEvent:/Accueil/Espaces␣Utilisateurs"/>
  <event time="1293689" type="ChangeSpaceEvent"/>
  <event time="1293689" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293699" type="SearchEvent"/>
  <event time="1293733" type="LoginEvent"/>
  <event time="1293733" type="ChangeSpaceEvent"/>
  <event time="1293733" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293736" type="ChangeSpaceEvent"/>
  <event time="1293736" type="ChangeSpaceEvent:/Accueil/Test"/>
  <event time="1293738" type="ChangeSpaceEvent"/>
  <event time="1293738" type="ChangeSpaceEvent:/Accueil/Test/Damien"/>
  <event time="1293741" type="ChangeSpaceEvent"/>
  <event time="1293741" type="ChangeSpaceEvent:/Accueil/Test/Damien/Biblio␣Chroniques"/>
  <event time="1293745" type="ChangeSpaceEvent"/>
  <event time="1293745" type="ChangeSpaceEvent:/Accueil"/>
  <event time="1293755" type="SearchEvent"/>
  <event time="1293767" type="SearchEvent"/>
  <event time="1293773" type="ReadDocEvent"/>
  <event time="1293773" type="ReadDocEvent:/Accueil/Test/Damien/Biblio␣Chroniques/DOUSSON_2007_Chronicle␣Recogn..."/>
  <event time="1293833" type="ChangeSpaceEvent"/>
</sequence>
```

Annexe D

Exemple de transformation SBT décrite manuellement en `Javascript`

Dans le cadre du projet PROCOGEC, dans lequel s'inscrit cette thèse, un Système à Base de Traces (SBT) a été développé. L'architecture de ce SBT est conforme à l'architecture des SBT que nous avons donnée en section 2.1.3. Il implémente donc notamment les transformations de traces. Dans ce SBT, les transformations de trace sont décrites manuellement par l'utilisateur du SBT à l'aide du langage `Javascript`. Afin de donner une idée du code nécessaire pour implémenter une transformation manuellement dans le SBT, nous donnons dans cet annexe deux exemples de transformation des traces de la plate-forme `CollaborativeECM` (voir l'annexe C pour un extrait de cette trace) : l'un est celui d'une transformation simple et courte (*LoginEvent*), l'autre est celui d'une transformation plus élaborée (*ReadDocEvent*).

Le premier exemple ci-dessous décrit la transformation *LoginEvent*, qui crée un observé de type `LoginEvent` à partir de l'observé de type `AuditRepoEvent` (l'appel au service d'authentification). Cet observé représente l'authentification d'un utilisateur de la plate-forme `CollaborativeECM` et le début de sa session. Un objet `trace` est fourni par le SBT dans l'API `Javascript` de transformation. Cet objet est la trace sur laquelle s'opérera la transformation codée. La transformation *LoginEvent* ci-dessous consiste à effectuer une boucle sur les observés (`obsel`) de cette trace et à créer quasiment à l'identique cet observé dans la trace transformée s'il s'agit d'un observé de type `AuditRepoEvent` et de nom `AuthenticationService.authenticate`. Cette transformation est donc un exemple de filtrage.

```
/* Copyright Knowings 2010 */
/* Auteurs: P. Rivarola et M. Bodin*/

/* parcours de la trace */
for (var i = 0; i < trace.obsels.length; i++)
{
   var obsel = trace.obsels[i];
   if(obsel.type == 'AuditRepoEvent' && obsel.name == 'AuthenticationService.authenticate'){

      for(var k = 0; k < obsel.fromRelations.length; k++){
         if(obsel.fromRelations[k].type == 'who'){
            var user = result.copyObsel(obsel.fromRelations[k].to);

            var login_event = result.createObsel();
            login_event.setType('LoginEvent');
            login_event.setName('Ouverture de session pour ' + user.name);
            login_event.setStartDate(obsel.startDate);
            login_event.setEndDate(obsel.endDate);
            result.addObsel(login_event);

            result.addObsel(user);
            login_event.createRelationTo('who', user.name, user);
         }
      }
   }
}
```

Le deuxième exemple de transformation, plus élaboré, est celui de la transformation *ReadDocEvent*. La transformation *ReadDocEvent* crée un observé de type `ReadDocEvent` à partir des observés de type `HttpWebEvent`, `ContextWebEvent` et `AuditRepoEvent`. De nombreux cas sont à prendre en compte dans cette transformation, car d'une part un document peut être lu par l'intermédiaire de plusieurs protocoles différents, et d'autre part il peut être lu par le système (c'est-à-dire générer un `AuditRepoEvent` de type « lecture ») sans avoir été réellement lu par l'utilisateur. L'observé de type `ReadDocEvent` généré représente la lecture d'un contenu par l'utilisateur quel qu'en soit le moyen.

Les 60 premières lignes de la transformation définissent les expressions régulières qui permettent de connaître le mode d'accès en lecture à un document à partir de l'url `HTTP` du protocole d'accès. La fonction `getMode` retourne ce mode d'accès à partir de l'url et des expressions régulières. La transformation consiste ensuite à faire une boucle sur les observés de la trace à transformer et à créer des observés nouveaux de type `ReadDocEvent` à chaque fois que l'observé source contient une url d'accès à un document. Les nouveaux observés sont construits de telle sorte que le type d'accès au document lu, retourné par la fonction `getMode`, est renseigné par l'attribut `mode` de ces observés.

```
/* Copyright Knowings 2010 */
/* Auteurs: P. Rivarola et M. Bodin*/

/* Expressions regulieres autorisees pour les URLs */

var content_regexps = [
   /\/(download|d|guestDownload|gd)\/.*/i,
   /\/(c|command|cecm)\/pack\/viewSpace\?.*content\=\/?reference\/workspace.*/i,
   /\/(r|resource|guestresource|gr)\/.*\/doc\?name=.*/i
];

var form_regexps = [
   /\/(c|command|cecm)\/form\/showpublication\?.*/i,
   /\/(c|command|cecm)\/actions\/run\?action=[^s]*showPublication.*/i,
   /\/(c|command|cecm)\/pack\/viewSpace\?.*content\=\/?dialog\/showPublication.*/i
];

var descriptor_regexps = [
   /\/(r|resource|guestresource|gr)\/.*\/descriptor\?.*document\=workspace.*/i,
   /\/(c|command|cecm)\/pack\/viewSpace\?.*content\=\/?descriptor.*document=workspace\:.*/i,
   /\/(t|template|guestTemplate|gt)\/.*/i
];

var webdav_regexps = [
   /\/(webdav|nrwebdav)\/.*/i
];

var preview_regexps = [
   /\/(s|wcs|service|wcservice)\/api\/node\/.*\/content\/thumbnails\/(webpreview|pdf).*/i
];

var regexps =
content_regexps.concat(form_regexps).concat(descriptor_regexps).concat(webdav_regexps)
                                                                .concat(preview_regexps);

var http_obsels = new Object();
var targets = new Object();
var created_obsels = [];

var getMode = function(url){
   for(var k = 0; k < content_regexps.length; k++){
      if(regexps[k].test(url)){
         return "content";
      }
   }
   for(var k = 0; k < form_regexps.length; k++){
      if(regexps[k].test(url)){
         return "form";
      }
   }
   for(var k = 0; k < descriptor_regexps.length; k++){
      if(regexps[k].test(url)){
         return "descriptor";
      }
   }
   for(var k = 0; k < webdav_regexps.length; k++){
      if(regexps[k].test(url)){
         return "webdav";
      }
   }
```

```
   for(var k = 0; k < preview_regexps.length; k++){
      if(regexps[k].test(url)){
         return "preview";
      }
   }
   return null;
};

/* parcours de la trace en remontant les evenements */
for (var i = trace.obsels.length - 1; i >= 0 ; i--)
{
   var obsel = trace.obsels[i];

   /* memorisation des acces Web correspondant a des URL de lecture */
   if(obsel.type == 'HttpWebEvent'){

      http_obsels[obsel.attributes.threadId] = null;
      targets[obsel.attributes.threadId] = new Object();
      if(obsel.attributes.length > 0 && obsel.attributes['url']){
         var url = obsel.attributes['url'];
         for(var k = 0; k < regexps.length; k++){
            if(regexps[k].test(url)){
               http_obsels[obsel.attributes['threadId']] = obsel;
               logger.debug("Accepted URL: " + url);
            }
         }
      }
   }
   else if(obsel.type == 'AuditRepoEvent' && obsel.name == 'ContentService.getReader'
                                          && http_obsels[obsel.attributes['threadId']]){
      http_event = http_obsels[obsel.attributes['threadId']];

      if(http_event.startDate <= obsel.startDate && http_event.endDate >= obsel.endDate){
         logger.debug("Found read access");

         for(var j = 0; j < obsel.fromRelations.length; j++){
            if(obsel.fromRelations[j].type == 'performedOn' && obsel.fromRelations[j].name == 'before'){
               var entity = obsel.fromRelations[j].to;
               var node_family = entity.attributes["NODEFAMILY"] ? entity.attributes["NODEFAMILY"] : "";

               if(entity.attributes["sys:node-uuid"]
                     && !(targets[obsel.attributes['threadId']][entity.attributes["sys:node-uuid"]])
                     && node_family
                     && (node_family == 'document'
                              || node_family[0] == 'document'
                              || node_family[1] == 'document')){
                  var read_event = result.createObsel();
                  read_event.setType('ReadDocEvent');
                  read_event.setName('http');
                  read_event.setStartDate(obsel.startDate);
                  read_event.setEndDate(obsel.endDate);
                  read_event.putAttribute('replayUrl', http_event.attributes['url']);
                  read_event.putAttribute('mode', getMode(http_event.attributes['url']));
                  created_obsels.push(read_event);

                  var document = result.copyObsel(entity);
                  read_event.createRelationTo('performedOn', entity.name, document);
                  created_obsels.push(document);

                  for(var k = 0; k < obsel.fromRelations.length; k++){
                     if(obsel.fromRelations[k].type == 'performedBy'){
                        var user = result.copyObsel(obsel.fromRelations[k].to);
                        read_event.createRelationTo('performedBy', user.name, user);
                        created_obsels.push(user);
                     }
                  }
                  /* on memorise les documents pour lesquels on a deja
                     cree un evenement pour cette requete HTTP */

                  targets[obsel.attributes['threadId']][entity.attributes["sys:node-uuid"]] = read_event;
               }
               else if(node_family
                           && (node_family == 'thumbnail'
                                    || node_family[0] == 'thumbnail'
                                    || node_family[1] == 'thumbnail')){
                  var read_event = result.createObsel();
                  read_event.setType('ReadDocEvent');
                  read_event.setName('http');
                  read_event.putAttribute('replayUrl', http_event.attributes['url']);
                  read_event.putAttribute('mode', 'preview');
                  created_obsels.push(read_event);

                  var document = result.createObsel();
                  document.setType('Node');
                  var path = entity.attributes['PATH'].split('/');
                  document.setName(path[path.length - 2]);
                  document.putAttribute('cm:name', document.name);
                  var node_ref = entity.attributes['PRIMARYPARENT'].split(/[\:\/]*\//);

                  document.putAttribute('sys:store-protocol', node_ref[0]);
                  document.putAttribute('sys:store-identifier', node_ref[1]);
                  document.putAttribute('sys:node-uuid', node_ref[2]);
```

```
                read_event.createRelationTo('performedOn', document.name, document);
                created_obsels.push(document);

                for(var k = 0; k < obsel.fromRelations.length; k++){
                   if(obsel.fromRelations[k].type == 'performedBy'){
                      var user = result.copyObsel(obsel.fromRelations[k].to);
                      read_event.createRelationTo('performedBy', user.name, user);
                      created_obsels.push(user);
                   }
                }

                /* on memorise les documents pour lesquels on a deja cree
                   un evenement pour cette requete HTTP */
                targets[obsel.attributes['threadId']][node_ref[2]] = read_event;
             }
          }
       }
    }
 }

/* add obsels to the trace in chronologic order */
for(var i = created_obsels.length - 1; i >= 0; i--){
   result.addObsel(created_obsels[i]);
}
```

Annexe E

Représentation d'une partition de musique sous forme de séquence d'événements

La séquence d'événements ci-dessous, donnée au format de trace de `Scheme Emerger`, représente les trente premiers événements de la partition donnée sur la figure E.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence time-unit="second" type="ie">
    <event type="do" time="0" />
    <event type="lab" time="1" />
    <event type="fa" time="2" />
    <event type="si" time="3" />
    <event type="do" time="4" />
    <event type="do" time="8" />
    <event type="lab" time="9" />
    <event type="fa" time="10" />
    <event type="do" time="11" />
    <event type="mib" time="12" />
    <event type="reb" time="14" />
    <event type="do" time="16" />
    <event type="lab" time="17" />
    <event type="fa" time="18" />
    <event type="si" time="19" />
    <event type="do" time="20" />
    <event type="do" time="24" />
    <event type="lab" time="25" />
    <event type="fa" time="26" />
    <event type="do" time="27" />
    <event type="reb" time="28" />
    <event type="do" time="30" />
    <event type="sib" time="32" />
    <event type="fa" time="33" />
    <event type="reb" time="34" />
    <event type="do" time="35" />
    <event type="sib" time="36" />
    <event type="lab" time="40" />
    <event type="fa" time="41" />
    <event type="re" time="42" />
    ...
</sequence>
```

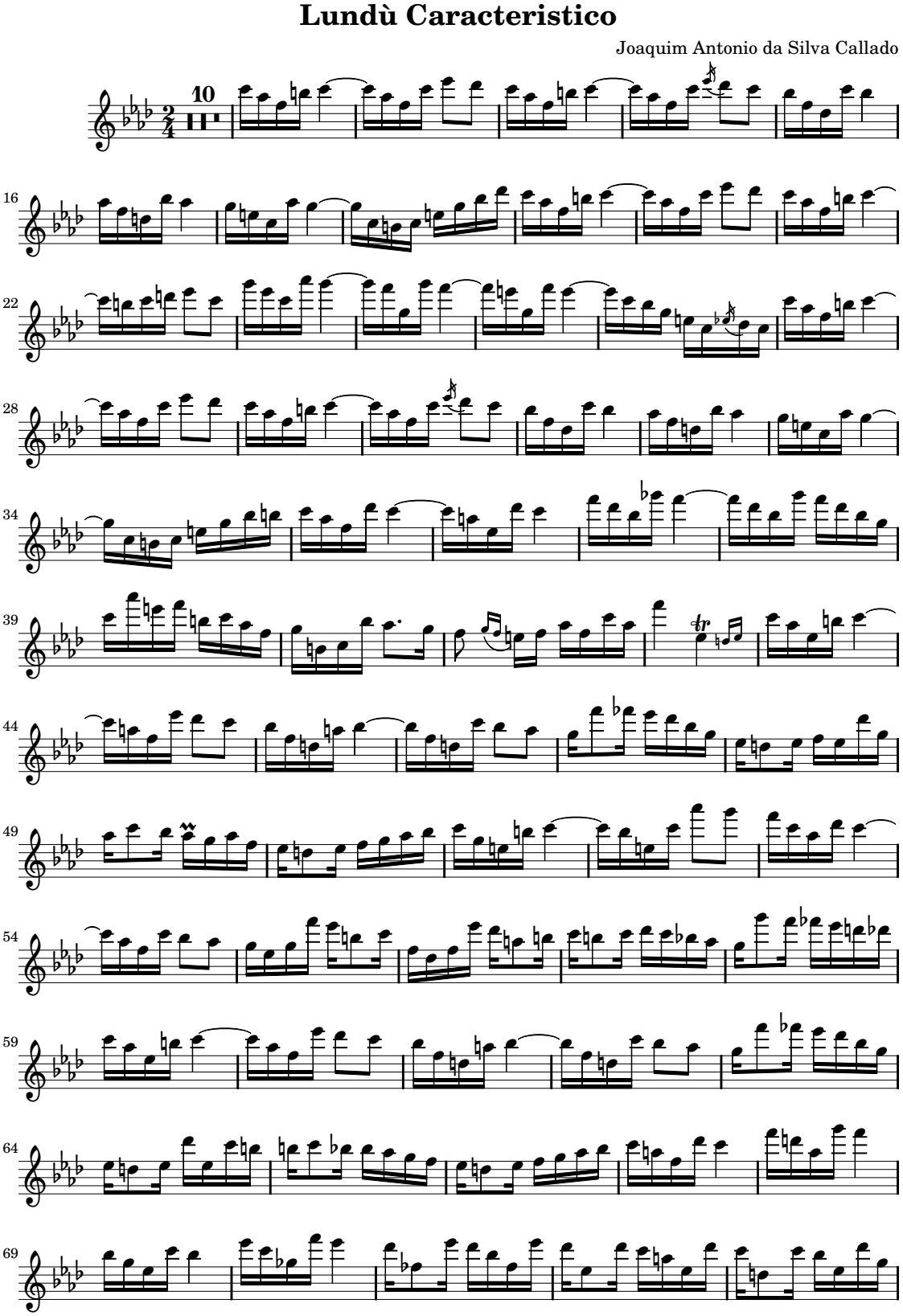


Figure E.1 – Extrait d'une partition de musique

Bibliographie

Aamodt, A. & Plaza, E. (1994), 'Case-based reasoning : foundational issues, methodological variations, and system approaches', *AI Commun.* **7**(1), 39–59.

Agrawal, R., Gunopulos, D. & Leymann, F. (1998), Mining process models from workflow logs, *in* 'EDBT '98 : Proceedings of the 6th International Conference on Extending Database Technology', Springer-Verlag, London, UK, pp. 469–483.

Agrawal, R., Imielinski, T. & Swami, A. N. (1993), Mining association rules between sets of items in large databases, *in* P. Buneman & S. Jajodia, eds, 'Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data', pp. 207–216.

Agrawal, R. & Srikant, R. (1995), Mining sequential patterns, *in* P. S. Yu & A. S. P. Chen, eds, 'Eleventh International Conference on Data Engineering', IEEE Computer Society Press, Taipei, Taiwan, pp. 3–14.

Alchourrón, C., G
"ardenfors, P. & Makinson, D. (1985), 'On the logic of theory change : Partial meet contraction and revision functions', *Journal of symbolic logic* pp. 510–530.

Allen, J. F. (1983), 'Maintaining knowledge about temporal intervals', *Commun. ACM* **26**(11), 832–843.

Bachimont, B. (2004), Pourquoi n'y a-t-il pas d'expérience en ingénierie des connaissances ?, *in* N. M. (ed), ed., 'Actes de la conférence "Ingénierie des connaissances (IC2004)"', Presses Universitaires de Grenoble, Lyon.

Badra, F., Cordier, A. & Lieber, J. (2009), Opportunistic Adaptation Knowledge Discovery, *in* Springer, ed., '8th International Conference on Case-Based Reasoning (ICCBR 2009)', pp. 60–74.

Beth, E. W. & Piaget, J. (1966), *Mathematical epistemology and psychology*, Gordon & Breach Science.

Buchanan, B. & Shortliffe, E. (1984), *Rule-based expert systems : the MYCIN experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Reading, MA.

Casas-Garriga, G. (2003), Discovering unbounded episodes in sequential data, *in* 'PKDD', pp. 83–94.

Champin, P.-A. (2003), ARDECO : an assistant for experience reuse in CAD, *in* B. Fuchs & A. Mille, eds, 'From Structured Cases to Unstructured Problem Solving Episodes For Experience-Based Assistance (Workshop 5 of ICCBR'03)'.

Champin, P.-A., Prié, Y. & Mille, A. (2003), MUSETTE : Modeling USEs and Tasks for Tracing Experience, *in* 'Workshop 5 'From Structured Cases to Unstructured Problem Solving Episodes For Experience-Based Assistance', ICCBR'03', pp. 279–286.

Champin, P.-A., Prié, Y. & Mille, A. (2004), MUSETTE : a framework for Knowledge from Experience, *in* 'EGC'04, RNTI-E-2 (article court)', Cepadues Edition, pp. 129–134.

Charlet, J. (2009), 'L'ingénierie des connaissances : Développements, résultats et perspectives pour la gestion des connaissances médicales.'. mémoire HDR, Université Pierre et Marie Curie.

Chen, Y.-L. & yi Wu, S. (2006), Mining temporal patterns from sequence database of interval-based events, *in* 'FSKD', pp. 586–595.

Cordier, A. (2008), Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning, Thèse de doctorat en informatique, Université Lyon 1. `http://liris.cnrs.fr/publis/?id=3776`.

Cordier, A., Fuchs, B., Lana de Carvalho, L., Lieber, J. & Mille, A. (2008), Opportunistic Acquisition of Adaptation Knowledge and Cases - The IakA approach , *in* M. M. K.-D. Althoff, R. Bergmann & A. Hanft, eds, 'European Conference on Case-Based Reasoning (ECCBR'08)', Springer, pp. 150–164.

Cordier, A., Fuchs, B., Lieber, J. & Mille, A. (2007), Acquisition de connaissances du domaine d'un système de RÀPC : une approche fondée sur l'analyse interactive des échecs d'adaptation - le système FRAKAS, *in* 'Séminaire Raisonnement à partir de cas, Plate Forme AFIA 2007', pp. 57–70.

Cordier, A., Mascret, B. & Mille, A. (2009), Extending Case-Based Reasoning with Traces, *in* 'Grand Challenges for reasoning from experiences, Workshop at IJCAI'09'.

Cordier, M.-O. & Dousson, C. (2000), Alarm driven monitoring based on chronicles, *in* 'proc. of the 4th Sumposium on Fault Detection Supervision and Safety for Technical Processes (SafeProcess)', pp. 286–291.

Cram, D., Fuchs, B., Prié, Y. & Mille, A. (2008), An approach to User-Centric Context-Aware Assistance based on Interaction Traces, *in* 'MRC2008 : fifth International Workshop on Modeling and Reasoning in Context'.

Cram, D., Jouvin, D. & Mille, A. (2007*a*), 'Visualisation interactive de traces et réflexivité : application à l'eiah collaboratif synchrone emediatheque', *STICEF, numéro spécial sur l'analyse des traces d'interactions dans les EIAH* **14**.

Cram, D., Jouvin, D. & Mille, A. (2007*b*), Visualizing Interaction Traces to improve Reflexivity in Synchronous Collaborative e-Learning Activities, *in* A. C. Limited, ed., '6th European Conference on e-Learning', pp. 147–158.

Cram, D., Kroener, A. & Mille, A. (2009), Using patterns in object's memories to make plan-driven help systems more flexible, *in* '1rst International Workshop on Digital Object Memories, in 5th International Conference on Intelligent Environments (IE'09)'.

Craw, S. (2009), Agile case-based reasoning : A grand challenge towards opportunistic reasoning from experiences, *in* 'Grand Challenges for reasoning from experiences, Workshop at IJCAI'09'.

de Amo, S. & Furtado, D. A. (2007), 'First-order temporal pattern mining with regular expression constraints', *Data Knowl. Eng.* **62**(3), 401–420.

de Amo, S., Furtado, D. A., Giacometti, A. & Laurent, D. (2004), An apriori-based approach for first-order temporal pattern mining, *in* 'Proceedings of the 19th Brazilian Symposium on Databases', pp. 48–61.

Dechter, R., Meiri, I. & Pearl, J. (1991), 'Temporal constraint networks', *Artif. Intell.* **49**(1-3), 61–95.

Deransart, P., Ducassé, M. & Ferrand, G. (2007), Une sémantique observationnelle du modèle des boîtes pour la résolution de programmes logiques, *in* 'Troisièmes Journées Francophones de Programmationpar Contraintes (JFPC07)', INRIA Rocquencourt, Rocquencourt / France France. `http://hal.inria.fr/inria-00151138/PDF/33.pdf`.

Diekert, V. & Rozenberg, G. (1995), *The Book of Traces*, World Scientific Publishing Co., Inc., River Edge, NJ, USA.

Dousson, C., Gaborit, P. & Ghallab, M. (1993), Situation recognition : Representation and algorithms, *in* 'IJCAI', pp. 166–174.

Dousson, C. & Duong, T. V. (1999), Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems, *in* 'IJCAI', pp. 620–626.

Dousson, C., Maigat, P. L. & R&d, F. T. (2007), Chronicle recognition improvement using temporal focusing and hierarchization, *in* 'IJCAI', pp. 324–329.

Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L. & Herlocker, J. L. (2005), Tasktracer : a desktop environment to support multi-tasking knowledge workers, *in* 'IUI '05 : Proceedings of the 10th international conference on Intelligent user interfaces', ACM, New York, NY, USA, pp. 75–82.

Duong, M. T. V. (2001), Découverte de chroniques à partir de journaux d'alarmes. Application à la supervision de réseaux de télécommunications, PhD thesis, Institut National Polytechnique de Toulouse.

Džeroski, S. (2003), 'Multi-relational data mining : an introduction', *SIGKDD Explor. Newsl.* **5**(1), 1–16.

Faure, C., Delprat, S., Boulicaut, J.-F. & Mille, A. (2006), Iterative bayesian network implementation by using annotated association rules, *in* 'Proc. 15th Int. Conf. on Knowledge Engineering and Knowledge Management EKAW'06', LNAI, Springer, pp. 326–333.

Fauré, C. (2007), Découverte de réseau pertinents par l'implémentation d'un réseau bayésien : application à l'industrie aéronautique, PhD thesis, Institut National des Sciences Appliquées de Lyon.

Fayyad, U., Piatetsky-shapiro, G. & Smyth, P. (1996), 'From data mining to knowledge discovery in databases', *AI Magazine* **17**, 37–54.

Fisher, D. H. (1987), 'Knowledge acquisition via incremental conceptual clustering', *Machine Learning* **2**(2), 139–172.

Gaber, M. M., Zaslavsky, A. & Krishnaswamy, S. (2005), 'Mining data streams : a review', *SIGMOD Rec.* **34**(2), 18–26.

Gendron, E., Marty, J.-C. & Carron, T. (2009), New usages for knowledge management through collaborative platforms, *in* 'ICKM - International Conference on Knowledge Management'.

Georgeon, O. (2008*a*), Analyse de traces d'activité pour la modélisation cognitive : application à la conduite automobile, Thèse de doctorat en psychologie cognitive, Université Lyon 2.

Georgeon, O., Henning, M. J., Bellet, T. & Mille, A. (2007), Creating Cognitive Models from Activity Analysis : A Knowledge Engineering Approach to Car Driver Modeling, *in* 'International Conference on Cognitive Modeling', Taylor & Francis, pp. 43–48.

Georgeon, O. L. (2008*b*), 'Analyzing traces of activity for modeling cognitive schemes of operators', *AISB Quarterly* **127**, 1–3.

Guillou, X. L., Cordier, M.-O., Robin, S. & Rozé, L. (2008), Chronicles for on-line diagnosis of distributed systems, *in* 'ECAI', pp. 194–198.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009), 'The weka data mining software : an update', *SIGKDD Explor. Newsl.* **11**(1), 10–18.

Han, J. & Kamber, M. (2006), *Data Mining : Concepts and Techniques, Second Edition*, Morgan Kaufmann.

Henning, M. J., Georgeon, O. & Krems, J. F. (2007), The quality of behavioral and environmental indicators used to infer the intention to change lanes, *in* 'International Driving Symposium on Human Factors in Driver Assessment, Training, and Vehicle Design'.

Hilbert, D. M. & Redmiles, D. F. (2000), 'Extracting usability information from user interface events', *ACM Comput. Surv.* **32**(4), 384–421.

Kayser, D. (1997), *La représentation des connnaissances*, Hermès.

Kendal, S. & Creen, M. (2006), *An Introduction to Knowledge Engineering*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Laflaquière, J. (2009), Conception de système à base de traces numériques pour les environnements informatiques documentaires , Thèse de doctorat en informatique, Université de Technologie de Troyes. `http://liris.cnrs.fr/publis/?id=4559`.

Laflaquière, J. & Prié, Y. (2009), *Réutilisation de l'expérience : modèles et applications*, Hemès Science Publishing, chapter Musette : Modéliser les Usages et les Tâches pour Tracer l'Expérience.

Laflaquière, J., Settouti, L. S., Prié, Y. & Mille, A. (2006), A trace-based System Framework for Experience Management and Engineering, *in* 'Second International Workshop on Experience Management and Engineering (EME 2006) in conjunction with KES2006'.

Laxman, S. & Sastry, P. S. (2006), 'A survey of temporal data mining', *Sadhana* **31**, 137–198.

Laxman, S., Sastry, P. S. & Unnikrishnan, K. P. (2007), A fast algorithm for finding frequent episodes in event streams, *in* 'KDD '07 : Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, New York, NY, USA, pp. 410–419.

Lin, M.-Y. & Lee, S.-Y. (2004), 'Interactive sequence discovery by incremental mining', *Inf. Sci. Inf. Comput. Sci.* **165**(3-4), 187–205.

Lloyd, S. (2003), 'Least squares quantization in pcm', *Information Theory, IEEE Transactions on* **28**(2), 129–137.

Magnusson, M. S. (2000), 'Discovering hidden time patterns in behavior : T-patterns and their detection.', *Behav Res Methods Instrum Comput* **32**(1), 93–110.

Mannila, H., Toivonen, H. & Verkamo, A. I. (1997), 'Discovery of frequent episodes in event sequences', *Data Mining and Knowledge Discovery* **1**(3), 259–289.

Mantaras, R. L. D., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., Cox, M. T., Forbus, K., Keane, M., Aamodt, A. & Watson, I. (2005), 'Retrieval, reuse, revision and retention in case-based reasoning', *Knowl. Eng. Rev.* **20**(3), 215–240.

Masseglia, F., Cathala, F. & Poncelet, P. (1998), The psp approach for mining sequential patterns, *in* 'PKDD '98 : Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery', Springer-Verlag, London, UK, pp. 176–184.

Masseglia, F., Poncelet, P. & Teisseire, M. (2003), 'Incremental mining of sequential patterns in large databases', *Data Knowl. Eng.* **46**(1), 97–121.

Masseglia, F., Poncelet, P. & Teisseire, M. (2009), 'Efficient mining of sequential patterns with time constraints : Reducing the combinations', *Expert Syst. Appl.* **36**(2), 2677–2690.

Masseglia, F., Teisseire, M. & Poncelet, P. (2005), *Sequential Pattern Mining : A Survey on Issues and Approaches*, Information Science Publishing, pp. 1028–1032.

Mathern, B. (2006), Analyser l'activité de conduite automobile : méthodologie et atelier logiciel associé, Master's thesis, Université Claude Bernard Lyon 1.

McCall, J. C., Trivedi, M. M., Wipf, D. & Rao, B. (2005), 'Lane change intent analysis using robust operators and sparse bayesian learning', *Computer Vision and Pattern Recognition Workshop* **0**, 59.

McKnight, A. J. & Adams, B. B. (1970), *Driver Education Task Analysis. Volume I : Task Descriptions.*, National Technical Information Service.

Michalski, R. S. (2003), Knowledge mining : A proposed new direction, *in* 'Proceedings of the Sanken Symposium on Data Mining and Semantic Web', Osaka University, Japan.

Mille, A. (2006), 'From case-based reasoning to traces-based reasoning', *Annual Reviews in Control* **30**(2), 223–232. Journal of IFAC.

Mille, A. (2009), Traces numériques d'interaction comme support réflexif de co-construction de sens, *in* 'Actes du colloque de l'Association pour la Recherche Cognitive (ARCo'09), Interprétation et problématiques du sens', Rouen.

Mooney, C. H. (2006), The Discovery of Interacting Episodes and Temporal Rule Determination in Sequential Pattern Mining, PhD thesis, Flinders University of South Australia.

Méger, N. (2004), Recherche automatique des fenetres temporelles optimales des motifs séquentiels, PhD thesis, Institut National des Sciences Appliquées de Lyon.

Napoli, A. (1997), Introduction aux logiques de description, Technical report, INRIA. `ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-3314.pdf`.

Parot, S., Tahli, F., Monin, J.-M. & Sebal, T. (2004), 'Livre blanc, les communautés de pratique, analyse d'une nouvelle forme d'organisation & panorama des bonnes pratiques'. `http://www.knowings.com/knowings/news.nsf/TBVSchDoc/C46DC3193A7FBDA4C1256F5600325844?OpenDocument&Highlight=0,e.,etudes`.

Parsia, B., Sirin, E. & Kalyanpur, A. (2005), Debugging owl ontologies, *in* 'WWW '05 : Proceedings of the 14th international conference on World Wide Web', ACM, New York, NY, USA, pp. 633–640.

Parthasarathy, S., Zaki, M. J., Ogihara, M. & Dwarkadas, S. (1999), Incremental and interactive sequence mining, *in* 'CIKM', pp. 251–258.

Patel, D., Hsu, W. & Lee, M. L. (2008), Mining relationships among interval-based events for classification, *in* 'SIGMOD '08 : Proceedings of the 2008 ACM SIGMOD international conference on Management of data', ACM, New York, NY, USA, pp. 393–404.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. & Hsu, M.-C. (2001), 'Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth', *icde* **00**, 215.

Pei, J., Han, J. & Wang, W. (2002), Mining sequential patterns with constraints in large databases, *in* 'CIKM '02 : Proceedings of the eleventh international conference on Information and knowledge management', ACM, New York, NY, USA, pp. 18–25.

Pei, J., Han, J. & Wang, W. (2007), 'Constraint-based sequential pattern mining : the pattern-growth methods', *J. Intell. Inf. Syst.* **28**(2), 133–160.

Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q. & Dayal, U. (2001), Multi-dimensional sequential pattern mining, *in* 'CIKM '01 : Proceedings of the tenth international conference on Information and knowledge management', ACM, New York, NY, USA, pp. 81–88.

Plantevit, M., Laurent, A. & Teisseire, M. (2006), Hype : mining hierarchical sequential patterns, *in* 'DOLAP '06 : Proceedings of the 9th ACM international workshop on Data warehousing and OLAP', ACM, New York, NY, USA, pp. 19–26.

Prud'hommeaux, E. & Seaborne, A. (2008), SPARQL query language for RDF, Technical report, World Wide Web Consortium.

Quinlan, J. R. (1993), *C4.5 : programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Riesbeck, C. K. & Schank, R. C. (1989), *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA.

Roussel, N., Tabard, A. & Letondal, C. (2006), All you need is log, *in* 'WWW2006 Workshop on Logging Traces of Web Activity : The Mechanics of Data Collection'.

Russell, S. & Norvig, P. (2003), *Artificial Intelligence : A Modern Approach, 2nd Edition*, Prentice Hall Series in Artificial Intelligence, Pearson Education.

Salvucci, D. D. (2004), 'Inferring driver intent : A case study in lane-change detection', *Human Factors and Ergonomics Society Annual Meeting Proceedings* **48**, 2228–2231(4).

Sanderson, P. & Fisher, C. (1994), 'Exploratory sequential data analysis : foundations', *Human-Computer Interaction* **9, No 3**, 251–317.

Schreiber, G. (2000), *Knowledge engineering and management : the CommonKADS methodology*, the MIT Press.

Settouti, L. S., Prié, Y., Champin, P.-A., Marty, J.-C. & Mille, A. (2009), 'A trace-based systems framework : Models, languages and semantics'. `http://hal.inria.fr/inria-00363260/en/`.

Srikant, R. & Agrawal, R. (1995), Mining generalized association rules, *in* 'VLDB '95 : Proceedings of the 21th International Conference on Very Large Data Bases', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 407–419.

Srikant, R. & Agrawal, R. (1996), Mining sequential patterns : Generalizations and performance improvements, *in* P. M. G. Apers, M. Bouzeghoub & G. Gardarin, eds, 'Proc. 5th Int. Conf. Extending Database Technology, EDBT', Vol. 1057, Springer-Verlag, pp. 3–17.

Srivastava, J., Cooley, R., Deshpande, M. & Tan, P.-N. (2000), 'Web usage mining : Discovery and applications of usage patterns from web data', *SIGKDD Explorations* **1**(2), 12–23.

Stuber, A. (2007), Co-construction de sens par négociation pour la réutilisation en situation de l'expérience tracée - Vers le partage et l'échange d'expérience collective, PhD thesis, Université Claude Bernard Lyon 1.

Studer, R., Benjamins, V. R. & Fensel, D. (1998), 'Knowledge engineering : Principles and methods', *Data and Knowledge Engineering* **25**(1-2), 161–197.

Sun, X., Orlowska, M. E. & Zhou, X. (2003), Finding event-oriented patterns in long temporal sequences., *in* 'PAKDD', pp. 15–26.

Teisseire, M. (2007), Autour et alentours des motifs séquentiels, Hdr, Université Montpellier II - Sciences et Techniques du Languedoc. `http://tel.archives-ouvertes.fr/tel-00203628/en/`.

van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G. & Weijters, A. J. M. M. (2003), 'Workflow mining : a survey of issues and approaches', *Data Knowl. Eng.* **47**(2), 237–267.

van der Aalst, W., Weijters, T. & Maruster, L. (2004), 'Workflow mining : Discovering process models from event logs', *IEEE Transactions on Knowledge and Data Engineering* **16**(9), 1128–1142.

Van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A. & Van der Aalst, W. (2005), 'The ProM framework : A new era in process mining tool support', *Applications and Theory of Petri Nets 2005* pp. 444–454.

Wenger, E. (1998), 'Communities of practice : Learning as a social system', *Systems thinker* **9**(5), 1–5.

Wenger, E. (2007), 'Communities of practice, a brief introduction'. `http://www.ewenger.com/theory/index.htm`.

Wikipédia (2010), 'Gestion des connaissances — wikipédia, l'encyclopédie libre'. `http://fr.wikipedia.org/w/index.php?title=Gestion_des_connaissances&oldid=53288964` [En ligne ; Page disponible le 26-mai-2010].

Wilson, T. (2002), 'The nonsense of knowledge management', *Information Research* **8**(1), 8–1.

Wojciechowski, M. (2001), Interactive constraint-based sequential pattern mining, *in* 'ADBIS '01 : Proceedings of the 5th East European Conference on Advances in Databases and Information Systems', Springer-Verlag, London, UK, pp. 169–181.

Yan, X. & Han, J. (2002), gspan : Graph-based substructure pattern mining, *in* 'ICDM '02 : Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)', IEEE Computer Society, Washington, DC, USA, p. 721.

Yu, C. C. & Chen, Y. L. (2005), 'Mining sequential patterns from multidimensional sequence data', **17**(1), 136–140.

Zaki, M. J. (2001), 'Spade : An efficient algorithm for mining frequent sequences', *Machine Learning* **42**(1/2), 31–60.

Zarka, R., Amélie Cordier, A., Corvaisier, F. & Mille, A. (2010), Providing assistance by reusing episodes stored in traces : a case study with sap business objects explorer, *in* '18ème Atelier "Raisonnement à Partir de Cas"'. Strasbourg.

Zhao, Q. & Bhowmick, S. S. (2003), Sequential pattern mining : A survey, Technical report, Nanyan Technological University, Singapore. `http://citeseer.ist.psu.edu/733677.html`.