Une Méthode à Base de Graphes pour la Corrélation de Messages dans les Logs

Belkacem Serrour, Hamamache Kheddouci

Une Méthode à Base de Graphes pour la Corrélation de Messages dans les Logs

Belkacem Serrour et Hamamache Kheddouci
Laboratoire LIESP, équipe G2AP
Université Claude Bernard Lyon 1, Lyon, France
{bserrour,hkheddou}@bat710.univ-lyon1.fr

***Résumé*—L'analyse de l'historique des conversations (fichiers log) entre services afin de découvrir les protocoles de conversations sous-jacents, aussi appelés protocoles métiers, est devenue un vrai challenge. Le comportement des services web est spécifié dans les protocoles métiers, ce qui montre l'importance de ces derniers. La génération des protocoles métiers passe d'abord par la corrélation des messages dans les logs en conversations. Cette tâche est facilement réalisable si on suppose que les logs contiennent des identifiants qui permettent d'associer chaque message à une conversation. Mais malheureusement, cette information est rarement existante dans les logs. Notre travail s'inscrit dans cette thématique, tel qu'il consiste à corréler les messages des logs des services web en conversations. Pour ce faire, contrairement aux autres approches, aucune supposition n'est faite concernant l'existence de l'identifiant des conversations. Notre approche consiste à modéliser les logs sous forme de graphes et à utiliser des techniques de la théorie des graphes pour extraire les conversations. Par ailleurs, les logs sont souvent incomplets et contiennent des erreurs. Ce qui induit une certaine incertitude des résultats. Afin de prendre en compte cette incertitude, nous introduisons la théorie de l'évidence de Dempster-Shafer. Notre approche a été implémentée et testée sur des logs générés.**

***Index Terms*—Services Web, Protocoles métiers, Corrélation des messages, Analyse des logs.**

I. Introduction

Ces dernières années, la recherche dans le domaine des services web a été très active. Après avoir très avancé dans la composition de services, elle s'oriente actuellement vers la découverte des protocoles de conversations, aussi appelés protocoles métiers. Les services web interagissent les uns avec les autres par le biais d'envoi de messages. Pour aboutir à un but donné, par exemple vendre un produit, le service échange une séquence de messages synchrones avec son client. Cette séquence de messages est appelée *conversation*. Un protocole métier est une spécification de toutes les conversations possibles qu'un service peut entreprendre avec ses partenaires (clients ou d'autres services). De ce fait, on dira que les protocoles métiers spécifient le comportement des services et représentent leurs interfaces dynamiques. Ainsi, ils complètent les interfaces statiques qui ne sont que des descriptions de différentes méthodes des services (spécifiées par le langage WSDL[1], par exemple). La découverte des protocoles de conversations passe par l'analyse des historiques d'exécution (fichiers log). L'analyse des logs, dans le but de découvrir les modèles qui les ont générés, est devenue un vrai challenge. Dans le domaine des services web, l'objectif premier de l'analyse des logs est la découverte des processus ou protocoles métiers. La génération des protocoles métiers passe d'abord par la corrélation des messages dans les logs en conversations. Cette tâche est facilement réalisable si on suppose que les logs contiennent des identifiants qui permettent d'associer chaque message à une conversation. La plupart des approches qui traitent le problème de la découverte dans le domaine des services web supposent l'existence de cette information. Mais malheureusement, cette information est rarement existante dans les logs.

[1]Web Services Description Language

Notre travail s'inscrit dans cette thématique, tel qu'il consiste à corréler les messages des logs des services web en conversations, puis générer automatiquement le protocole métier correspondant. Pour ce faire, contrairement aux autres approches, aucune supposition n'est faite concernant l'existence de l'identifiant des conversations. Notre approche consiste à modéliser les logs sous forme de graphes et à utiliser des techniques de la théorie des graphes pour en extraire les conversations.

Ce présent papier est organisé comme suit. Dans la section 2, on détaillera la problématique et on citera quelques travaux réalisés dans le domaine. On terminera la section par la définition de nos outils de travail. La section 3 sera consacrée aux détails de notre proposition. L'implémentation et les résultats seront présentés dans la section 4. La section 5 conclue le papier.

II. Contexte et préliminaires

A. Problématique

La communication dans les service web se fait par échange de messages. Chaque service web a une interface (spécifiée par le langage WSDL, par exemple). Cette interface englobe les opérations réalisées, les types de messages reçus et envoyés par le service, etc. L'interface spécifiée par WSDL n'est qu'une interface fonctionnelle, c-à-d : elle ne décrit que les différentes méthodes invocables (les séquences ordonnées de celles-ci ne sont pas décrites). Pour assurer l'aspect dynamique et comportemental des services web, une nouvelle interface est proposée dans [1][2]. Ces travaux montrent l'importance de l'utilisation des protocoles métiers (Business Protocol) pour compléter l'aspect statique assuré par WSDL et pour décrire le comportement des services web (spécifier les séquences ordonnées de messages échangés, appelées *conversations*). Les auteurs ont montré aussi l'efficacité d'utiliser les automates à états finis pour modéliser les protocoles métiers. La modélisation des services web en protocoles métiers fournit beaucoup d'avantages, entre autres : (i) Les protocoles métiers fournissent aux développeurs des informations pour programmer les clients à interagir correctement avec le service. (ii) Les protocoles métiers servent d'un modèle de vérification de contraintes de conception (s'assurer que le service réel correspond bien aux contraintes de conception). (iii) Un protocole métier est modélisé sous forme d'un automate à états finis, qui est un modèle visuel facilement exploitable par l'utilisateur (ajout de nouvelles fonctionnalités, contraintes, etc) [4][9]. Il est clair que le protocole métier offre beaucoup d'avantages et impose une importance aigüe, mais la difficulté est la question : Comment reconstituer un protocole sans aucune spécification à priori ? L'idée est d'analyser les fichiers log des services web pour en extraire le protocole métier, d'où le terme de la *Découverte* des protocoles métiers. Analyser les fichiers log est une tâche délicate car ces derniers sont souvent incomplets, incertains et contiennent des erreurs. Donc, le processus de découverte doit prendre en compte tous ces aspects. La première

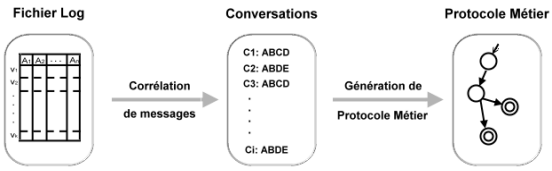

FIG. 1. Processus de la découverte des protocoles métiers.

difficulté concerne la détection des erreurs dans le log. Dans un log d'un service web, les types d'erreur qu'on peut trouver sont les suivants[8] : erreurs dans les conversation et incomplétude des logs. En ce qui concerne les erreurs dans les conversations, on distingue ces trois cas possibles : messages manquants, messages permutés et conversations partielles. La deuxième difficulté de l'analyse des fichiers log est la corrélation des messages dans le log, c-à-d : grouper les messages dans le log en conversations. Cette étape est la première dans le processus de la découverte du protocole métier (voir Figure 1). La corrélation des messages peut être facile à faire si on dispose d'une information dans le log indiquant à quelle conversation appartient chaque message (cette information est appelée identifiant de la conversation, ID-Conv). Mais malheureusement cet ID-Conv n'est pas toujours présent dans les logs.

B. Travaux Similaires

Le problème d'analyse des fichiers log pour la découverte des modèles touche de nombreux domaines. Dans le domaine des services web, les travaux visent la découverte de processus (le workflow) et aussi les protocoles métiers. Le problème du workflow consiste à construire un modèle formel, à partir des logs, représentant le fonctionnement de processus [7][12]. Le modèle est généralement représenté sous forme d'automates finis ou des réseaux de petri. En ce qui concerne le problème de la découverte des protocoles métiers, les auteurs dans [1] et [2] ont bien défini les protocoles métiers et ont montré leur importance. Les mêmes auteurs ont proposé dans [8] une méthode pour la découverte des protocoles métiers et une façon de détecter les erreurs dans les logs. L'idée de leur approche est constituée de trois étapes : identification du bruit, découverte de protocole, raffinement du protocole. Dans [4], les auteurs ont traité un sous problème qui est la découverte des transitions temporisées des protocoles métiers. Les auteurs ont introduit la notion *d'expiration propre* comme indice dans les logs pour trouver les transitions temporisées. Ils se basent dans leur travail sur la notion d'épisode et la durée entre deux messages d'une même conversation pour détecter ces transitions. Mais en revanche, et comme toutes les approches citées plus haut, les auteurs supposent l'existence des identifiants de conversations pour la corrélation de messages. Les auteurs dans [10] ont tenté d'aborder la problématique de corrélation. Ils jugent que les messages des logs doivent être reliés par des règles de corrélation pour former des conversations. L'objectif de ce papier consiste en la recherche des attributs des entrées logs à utiliser pour la corrélation, et à spécifier les règles de corrélation groupant les messages en conversations.

Notre principale contribution est dans ce sens. En effet, elle consiste en la corrélation des messages dans les logs des services web pour la découverte des protocoles métiers, ceci est fait sans la supposition de l'existence au préalable de l'ID-Conv dans le log.

C. Préliminaires

Avant de détailler notre contribution, nous allons tout d'abord définir les notions sur lesquelles on s'appuie. Les définitions des notions de services web sont tirées des papiers de l'équipe de B. Benatallah [1][8].

1) Protocole Métier: Un protocole métier (Business Protocol en anglais) est une spécification de toutes les conversations possibles qu'un service peut avoir avec ses partenaires. Dans la suite de ce papier, on notera par BP le Business Protocol. La plupart des papiers qui traitent cette problématique utilisent un Automate à États Finis (AEF) pour modéliser le BP. Nous en ferons de même. D'une manière formelle, un BP peut être vu comme un uplet P=(S,S_0,F,M,T) où :

- S : Ensemble des états du protocole,
- S_0 : État initial,
- F : Ensemble des états finaux,
- M : Ensemble des messages supportés par le service,
- T : Ensemble des transitions.

Un protocole P doit accepter (resp. refuser) une conversation *"c"* si un chemin existe (resp. n'existe pas) de l'état initial à l'un des états finaux.

2) Conversation: Une conversation *"c"* est une séquence de messages échangés entre un service et un client pour aboutir à un but donné. Nous supposons que toutes les conversations commencent par le même message (un message d'authentification, par exemple).

3) Fichier Log: Un fichier Log, ou journal, est un fichier texte (XML, ...) reprenant de façon chronologique l'ensemble des événements qui ont affecté un système informatique. D'une manière formelle, un Log L peut être vu comme une table de n entrées. $L = \sum_{i=1}^{n} \nu_i$, où ν_i est un événement ou encore appelée Entrée Log. Chaque entrée Log est un uplet $\nu = (\nu.A_1, \nu.A_2, ..., \nu.A_k)$ ou $\nu.A_i$ représente la valeur de ν dans l'attribut A_i. Le contenu des Logs des services web diffère de l'un à l'autre suivant le mécanisme de logging. Les informations qu'on peut trouver dans les Logs des services web sont détaillées dans [5] et [6]. Ces informations sont en général :

- T : Timestamp : Date de l'envoi du message,
- S : Sender : L'émetteur du message,
- R : Receiver : Le recepteur du message,
- M : MessageType : Contenu du message.

Dans notre proposition, on se contente de ces quatre attributs contrairement aux autres approches qui supposent l'existence de l'attribut identifiant les conversations (ID-Conv). Dans notre approche, une entrée log est définie comme suit : $\nu = (\nu.T, \nu.S, \nu.R, \nu.M)$.

III. CONTRIBUTION

Dans cette section, on détaillera le principe et les différentes étapes de notre proposition. L'architecture générale de notre approche est illustrée dans la Figure 2. Dans ce qui suit, on détaillera le fonctionnement et le principe de chaque étape.

A. Partitionnement du Log

Un service web peut avoir des conversations avec différents clients. Le fait qu'un service entreprenne plusieurs conversations simultanées avec des clients différents provoque un chevauchement de messages dans le fichier Log (i.e. deux messages qui se suivent dans le Log ne font pas nécessairement partie d'une même conversation). Pour contourner ce problème et trouver les vraies conversations, une première étape s'est avérée nécessaire qui consiste à partitionner le Log en sous-ensembles tels que dans chacun d'eux, on n'aura que des messages échangés entre le service et un client donné. Le partitionnement du Log se base sur deux attributs : le Sender et

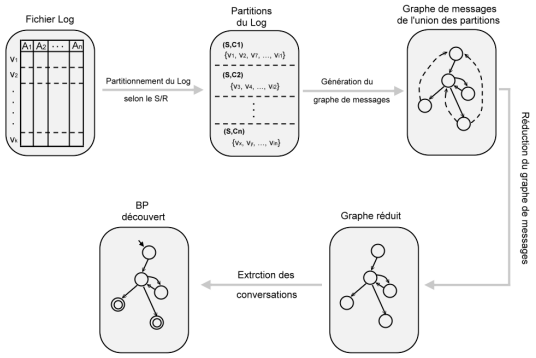

FIG. 2. Les étapes de la proposition.

le Receiver. Toutes les entrées log qui contiennent le même couple d'attributs (S,R) ou (R,S), doivent appartenir au même sous-ensemble.

B. Graphe de messages

Une fois le Log partitionné en sous ensembles C_i, les *MessageType* de chaque partition sont ordonnés selon leurs *Timestamp* (cela indique la séquentialité des messages dans le Log pour chaque client). Un graphe orienté de messages G est construit pour chaque sous-ensemble C_i. Les noeuds du graphe G vont représenter les *MessageType* des entrées Log ν_i et les arcs la séquentialité des messages. L'orientation des arcs dans le graphe de messages est importante. Un arc de A vers B signifie que le message B succède au message A mais pas le contraire. Soit $G(V, A)$ le graphe de messages où V représente l'ensemble des *MessageType* et A l'ensemble des arcs du graphe G tel que : $\forall \nu_1, \nu_2 \in V^2$, $\exists$ un arc de ν_1 à ν_2 ssi : ν_2 succède ν_1 dans la table. Les arcs du graphe G seront pondérés. Le poids d'un arc *a* reliant deux sommets ν_1 et ν_2 représente le nombre de fois que le message ν_2 a succédé à ν_1 (voir Figure 3). Dans la Figure 3, le poids de l'arc entre le sommet *A* et le sommet *B* est de quatre (04). Cela signifie que le sommet *B* a succédé quatre fois au sommet *A*. L'Algorithme 2 illustre la génération du graphe de messages à partir d'une partition donnée C_i. L'union des différents graphes de messages générés représente l'information complète du fichier Log. Dans ce qui suit, on verra quelles sont les informations à extraire de ces graphes et comment le faire.

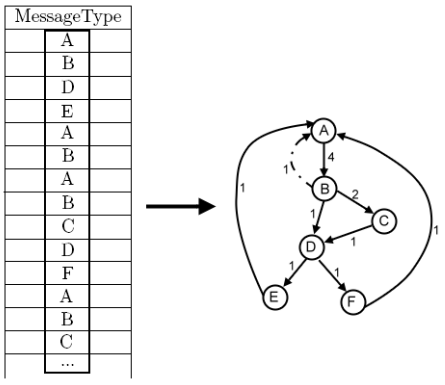

FIG. 3. Graphe de Messages.

1) Sommet initial: La première information qu'on doit extraire dans le graphe est le message initial des conversations (sachant que toutes les conversations commencent par le même message, mais on le connait pas à priori). Les fichiers Log peuvent toujours contenir des erreurs. Donc, rien ne garantit que le premier message lu dans le Log soit le message initial (cela peut être dû à une erreur au début du fichier Log ou encore le Log à analyser n'est qu'un fragment non complet ; Par conséquent, le premier message dans ce fichier n'a aucune certitude d'être le message initial). Les erreurs dans le graphe de messages sont représentées par des arcs non fréquents (de poids faibles). Un arc représente une séquence de deux messages. Une séquence de messages qui est rarement répétée dans le log est soit une erreur, soit une séquence d'une vraie conversation mais non fréquente. Nous verrons plus loin dans ce papier comment différencier les erreurs et les conversations non fréquentes. Pour décider si un arc est de poids faible ou pas, un seuil de fréquence doit être calculé. Ce seuil est calculé selon le poids de l'arc le plus élevé. Par des simulations, nous avons fixé ce seuil à 15% du plus grand poids des arcs. Pour trouver le message initial dans le graphe (sommet initial), un ensemble de critères doivent d'être combinés. Avant de citer ces critères, on précédera par quelques définitions. Soit $G = (V, A)$ un graphe orienté. Pour chaque sommet v du graphe G, on définit : Som(*v*) : la somme des poids de tous les arcs entrants et sortants au sommet *v*, Som$^+$(*v*) : la somme des poids des arcs sortants au sommet *v* et Som$^-$(*v*) : la somme des poids des arcs entrants au sommet *v*.

Sommet de Som élevé. Dans le graphe de messages, un sommet qui a un *Som* élevé indique que le message est répété plusieurs fois dans le log. Un message initial est certainement de *Som* élevé car il figure dans toutes les conversations entre le service et le client. Un sommet de *Som* maximum n'est pas nécessairement le message initial puisque il peut appartenir à une boucle qui augmente son *Som*. C'est pour cela que d'autres critères à déterminer s'avèrent être nécessaires.

Sommet sans boucle unitaire fréquente. Une boucle unitaire dans le graphe de messages signifie que le message se succède à lui même plusieurs fois dans le Log. Or le message initial ne se succède pas dans le Log sauf dans le cas d'erreur. Toutefois comme les erreurs sont aléatoires, le poids de la boucle de sommet initial sera faible. Donc, un sommet avec une boucle de poids élevé ne peut pas être

Algorithm 1 Graphe de messages.

Require: Partition C_i
Ensure: Graphe de messages
  while (NonVide(C_i)) **do**
    s_1 = Liste(j) ; {Un message dans C_i}
    s_2 = Liste(j+1) ; {Un message qui succède s_1}
    if NonExiste(s_1) **then**
      Créer(s_1) ; {Créer le noeud s_1 du graphe}
    end if
    if NonExiste(s_2) **then**
      Créer(s_2) ; {Créer le noeud s_2 du graphe}
    end if
    $d^+(s_1) + +$; {Incrémentation du degré sortant de s_1}
    $d^-(s_2) + +$; {Incrémentation du degré entrant de s_2}
    if ExisteArc(s_1, s_2) **then**
      PoidsArc ++ ; {Incrémentation du poids de l'arc entre s_1 et s_2}
    else
      CréerArc(s_1, s_2) ;
      PoidsArc(s_1, s_2)=1 ;
    end if
  end while

	Score non normalisé			Score normalisé		
	P1	P2	P3	P1	P2	P3
A	990	3	3	25.05	42.88	30
B	992	1	2	26.12	14.28	20
E	417	1	2	10.97	14.28	20
D	922	1	1	24.26	14.28	10
F	479	1	2	12.60	14.28	20

TAB. I
SCORE DES SOMMETS SELON CHAQUE CRITÈRE.

	Score sans l'incertitude			Score avec l'incertitude		
	P1	P2	P3	P1	P2	P3
A	25.05	42.88	30	11.72	38.60	25.5
B	26.12	14.28	20	11.75	12.85	17
E	10.97	14.28	20	04.93	12.85	17
D	24.26	14.28	10	10.92	12.85	8.5
F	12.60	14.28	20	05.68	12.85	17
UB	0	0	0	55	10	15

TAB. II
INTRODUCTION DES FACTEURS D'INCERTITUDE UB POUR CHAQUE CRITÈRE.

un sommet initial.

Sommet qui a plus d'arcs faibles entrants. Les arcs de poids faible dans le graphe représentent les erreurs dans le Log. Les types d'erreurs qu'on peut avoir dans un Log sont les trois suivants : perte de message, swap de message et conversation interrompue. Pour le troisième type d'erreur, à chaque fois qu'une conversation est interrompue, il y en aura une nouvelle qui commence. Donc, on aura des arcs de sommets où la conversation est coupée vers le sommet initial (début d'une nouvelle conversation). Par conséquent, le sommet initial aura beaucoup de chances d'avoir des arcs entrants non fréquents.

Sommet de Som$^+$ supérieur au Som$^-$. On construit un graphe de l'union de tous les graphes des sous-ensembles log (graphe de message de tout le log). Un sommet du graphe résultant doit avoir la particularité que son *Som*$^+$ soit supérieur au *Som*$^-$. Cela est dû au fait de partitionner le Log en sous-ensembles, ce qui évite les arcs de retours des derniers messages de différents clients vers le sommet initial.

2) Combinaison de Dempster-Shafer: La théorie de Dempster-Shafer [3][11] a été initialement développée par Arthur Dempster, puis elle fut étendue par Glenn Shafer. La théorie de Dempster-Shafer est une approche qui permet de combiner différentes sources d'informations ou critères (appelées évidences) dans le but de parvenir à une décision dans une situation caractérisée par un fort degré d'incertitude. La théorie fournit aussi les outils nécessaires pour combiner entre les différentes évidences et donner à ces évidences différentes pondérations suivant leur importance dans la prise de décision finale, leur qualité et leur pertinence. Les critères cités ci-dessus doivent être combinés pour trouver le sommet initial. Pour ce faire, la formule de combinaison de Dempster-Shafer est appliquée pour les trois critères suivants :

- P_1 : Sommet de *Som* élevé,
- P_2 : Sommet qui a plus d'arcs faibles entrants,
- P_3 : Sommet qui a le *Som*$^+$ supérieur au *Som*$^-$.

Le quatrième critère, Sommet sans boucle, ne rentre pas dans la formule de combinaison car il est certain (un sommet avec boucle unitaire est certainement un sommet non initial). Chacun des trois critères à combiner donne un score aux sommets de l'ensemble des candidats. L'ensemble des candidats U=$\{c_1, c_2, ..., c_k\}$ est l'ensemble de tous les sommets du graphe excepté des sommets avec boucle. C'est à dire, les sommets qui peuvent être sommet initial.

Fonction de score. Une fonction de score est associée à chaque critère. Cette fonction dépend du critère en lui même. Voici les fonctions de score de chaque critère :

- $P_1 : F(c_i) =$ *Som* de sommet c_i,
- $P_2 : F(c_i) =$ Nombre d'arcs de poids faible entrant au sommet c_i,
- $P_3 : F(c_i) =$ Différence entre le *Som*$^+$ et le *Som*$^-$ du sommet c_i.

Le candidat qui a le meilleur score pour chaque critère P_i est pris comme sommet initial trouvé par ce critère P_i. (voir Tab I). Dans le tableau I, pour le critère P_1 : Sommet de *Som* élevé, le candidat B a le meilleur score. score(B)=*Som*(B)=992. Donc c'est le candidat B qui est sommet initial suivant le critère P_1. Le candidat A est le sommet initial trouvé suivant les critères P_2 et P_3 avec un score de 3 pour chacun. Pour pouvoir appliquer la formule de combinaison de Dempster-Shafer, les scores des sommets de chaque critère doivent être normalisés (somme des scores égal à 100). Cela donne une masse à chaque sommet, par la formule suivante :

$$m(c_i) = \frac{Score(c_i)}{\sum_{i=1}^{n} Score(c_i)} \times 100$$

Facteur d'incertitude UB. Avant d'appliquer la formule de combinaison de Dempster-Shafer, on doit d'abord calculer un facteur d'incertitude UB[2] pour chaque critère P_i. Ceci est effectué grâce à des simulations (voir partie Implémentation). Les facteurs d'incertitude trouvés pour chaque critère sont les suivants. P_1 : UB = 55, P_2 : UB = 10 et P_3 : UB = 15. Une fois les UB calculés, on recalcule les scores des sommets pour chaque critère en introduisant le facteur d'incertitude UB (voir Tab II). Les masses des candidats avec les facteurs d'incertitude sont calculées comme suit :

$$m'(c_i) = \left(\frac{m(c_i)}{\sum_{i=1}^{n} m(c_i)} \right) \times (100 - UB)$$

Formule de combinaison. Le sommet initial sera donné par la formule de combinaison de Dempster-Shafer. La combinaison des critères se fera deux à deux. Cela ne cause pas problème puisque la combinaison de Dempster-Shafer est commutative et associative. Soit m_1 et m_2 les masses associées respectivement aux critères P_1 et P_2. La nouvelle masse combinée des critères P_1 et P_2 d'un candidat A est donnée par la formule suivante :

$$m(A) = m_1 \otimes m_2 = \frac{\sum_{B \cap C = A} m_1(B) \cdot m_2(C)}{\sum_{B \cap C \neq \emptyset} m_1(B) \cdot m_2(C)}$$

Après l'application de la formule de combinaison, le candidat qui a le meilleur score sera le sommet initial (voir Tableau III). Dans l'exemple du tableau III, c'est le sommet A qui a la plus grande masse m_A =45.43, et donc il est supposé être le sommet initial.

C. Conversations candidates

Une fois le sommet initial trouvé, la deuxième partie consiste à chercher les conversations candidates. Pour ce faire, on procède en deux étapes en commençant d'abord à faire une réduction du graphe de messages, puis on cherche tous les chemins du sommet initial aux sommets finaux.

[2]UB : Uncommitted Belief

	P1	P2	P3	Combinaison
A	11.72	38.60	25.5	45.43
B	11.75	12.85	17	14.98
E	04.93	12.85	17	13.14
D	10.92	12.85	8.5	10.06
F	05.68	12.85	17	13.34
UB	55	10	15	03.05

TAB. III
COMBINAISON DES CRITÈRES.

1) Réduction du graphe de messages: La réduction de graphe ne touchera que les arcs. Les sommets étant les MessageType dans le Log, ce qui fait qu'ils sont tous valides. Deux types d'arcs seront supprimés.

Les arcs non fréquents. Comme déjà expliqué plus haut, les arcs non fréquents du graphe de messages représentent soit des erreurs dans le log, soit des séquences de conversations rarement effectuées par le service. Nous devons supprimer du graphe les arcs représentants les erreurs et bien évidemment laisser ceux des conversations non fréquentes. Pour ce faire, on procède comme suit :

- Si un sommet n'est connecté au graphe que par un arc non fréquent, la suppression de cet arc isolera le sommet du reste du graphe. Or les sommets représentent les MessageType des entrées log, et donc ils doivent tous apparaître dans les conversations du protocole recherché. Dans ce cas cet arc ne sera pas supprimé, il représente une séquence d'une conversation non fréquente.
- Si un sommet est connecté au graphe au moins par un arc fréquent, tous les arcs de poids faibles entrants à ce sommet seront considérés comme des erreurs (sachant que les erreurs sont des swap ou des pertes de messages dans les conversations, donc les arcs non fréquents représentants ces erreurs relieront les sommets (MessageType) de ces mêmes conversations). Dans ce cas, les arcs non fréquents seront supprimés.

Les arcs entrants au sommet initial. Un sommet initial représente le premier message des conversations, par conséquent, il ne doit pas contenir de prédécesseurs. Ce qui fait que tous les arcs entrants à ce sommet seront supprimés.

2) Chemins du graphe réduit : Après la localisation du sommet initial et la réduction du graphe de messages, on cherche les sommets finaux du graphe représentants les messages de fin des conversations. Cela est facilement réalisable. Les sommets finaux sont, tout simplement, des sommets n'ayant pas de successeurs (arcs sortants). L'ensemble des chemins du graphe réduit partant du sommet initial vers les sommets finaux représente l'ensemble des conversations du fichier log. A cette étape, on a réussi à spécifier l'ensemble des conversations et à les représenter sous forme d'un graphe de messages.

IV. IMPLÉMENTATION

Pour valider notre approche, nous avons développé un simulateur reprenant les étapes de la contribution. Le simulateur est programmé sous le langage python. L'idée de la validation de notre proposition est inspirée du papier de Van der Aalst [12]. La Figure 4 illustre le principe et la démarche de la validation. Comme illustré dans la Figure 4, l'idée consiste à générer un fichier Log à partir d'un Business Protocol connu, puis à appliquer les techniques de l'approche proposée pour découvrir le BP depuis ce Log généré. Une comparaison entre le BP source et le BP découvert permettra de décider de la validité de la contribution.

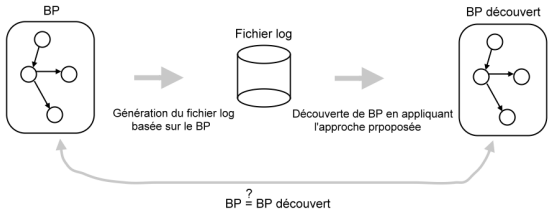

FIG. 4. Principe de validation de l'approche.

A. Génération du fichier Log

L'absence des logs réels nous a obligé à utiliser des fichiers log générés lors de nos simulations. Les fichiers Log sont générés à partir d'un Business Protocol. Ce qui fait que les Logs résultants contiennent des conversations issues du BP source. Puisque les Logs réels sont souvent imparfaits et contiennent des erreurs, et pour que les Logs générés soient représentatifs de la réalité, des erreurs doivent être introduites dans ces derniers. Pour ce faire, deux probabilités P et P-cut, paramétrables par l'utilisateur, sont introduites.

- P : représente la probabilité d'erreur dans le Log. Une erreur est une perte de message ou un swap de message,
- P-cut : représente le taux de conversations interrompues dans le Log.

B. Étapes de l'approche

Une fois le fichier log généré, les étapes suivantes de notre contribution sont mises en oeuvre.

1) Partitionnement du Log : Comme détaillé dans la section précédente, la première étape de la contribution consiste à partitionner le Log en sous-ensembles selon le Sender/Receiver. Le résultat de cette étape est un ensemble de partitions. Dans notre simulation, aucun résultat de cette étape ne sera affiché, puisque elle est considérée comme une étape intermédiaire.

2) Génération du graphe de messages: La deuxième étape de la contribution consiste à générer le graphe de message à partir des partitions. La figure 5 représente un exemple de graphe de messages.

Label des noeuds et des arcs. Les noeuds et les arcs du graphe

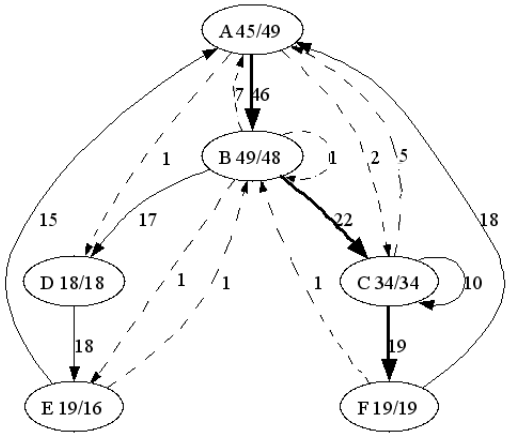

FIG. 5. Génération du graphe de messages.

sont marqués avec des labels. Le label d'un noeud consiste en un MessageType et un couple de som^+ et som^-. Le label d'un arc est son poids (pour rappel, le poids d'un arc est la fréquence de succession entre deux sommets). Le graphe de messages résultant

contient deux types d'arcs : fréquents et non fréquents. Dans la Figure 5, les arcs fréquents sont représentés par des traits continus et les arcs non fréquents par des traits discontinus.

Seuil de fréquence. Pour décider si un arc est de fréquence élevée ou non, on utilise un seuil de fréquence. Ce seuil est calculé selon le poids de l'arc le plus élevé. Dans notre simulateur, ce taux est paramétrable par l'utilisateur. Par des simulations, on a fixé le taux à un sixième (1/6) du plus grand poids. Seuil fréquence = PoidsMax(Arc)/6. Si on prend l'exemple du graphe de la Figure 5, le poids de l'arc le plus élevé est celui entre le sommet A et B. PoidsArc(A,B)=46, donc le seuil= 46/6 = 7. Tous les arcs du graphe qui ont un poids inférieur à 7 seront considérés comme des non fréquents.

3) Sommet initial: La troisième étape de l'approche est la localisation du sommet initial dans le graphe de messages (message de début des conversations). Pour ce faire, on a utilisé la théorie de l'évidence de Dempster-Shafer pour les trois critères :

- P_1 : Sommet de degré élevé,
- P_2 : Sommet qui a le plus d'arcs faibles entrants,
- P_3 : Sommet qui a le degré sortant supérieur au degré entrant.

Le facteur d'incertitude. Un facteur d'incertitude est calculé pour chaque critère grâce à des simulations. Ce calcul est fait sur la base des simulations sur 100 Logs différents. A chaque simulation, on fait varier la probabilité d'erreur P, la probabilité d'interruption P-cut et le nombre de clients dans le Log. Pour chaque exécution, on compte le nombre de fois qu'un critère a trouvé le sommet initial. L'UB de chaque critère représente le taux d'échec lié au choix du sommet initial. Les UB trouvés lors nos simulations sont :

- P_1 : UB = 55,
- P_2 : UB = 10,
- P_3 : UB = 15.

Combinaison de Dempster-Shafer. Ayant les scores normalisés avec l'incertitude de chaque candidat et pour chaque critère, il ne reste qu'à appliquer la formule de combinaison de Dempster-Shafer. Le candidat qui a le meilleur score sera pris comme sommet initial (voir la Figure 6). Dans cet exemple, c'est le candidat A qui a le meilleur

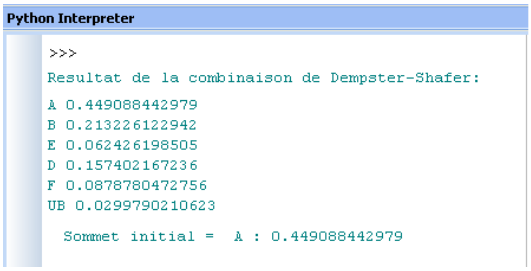

FIG. 6. Sommet initial trouvé après la combinaison des critères.

score (score(A)=0.44).

4) Réduction du graphe de messages: La quatrième étape de l'approche consiste à réduire le graphe de messages. La réduction ne touchera que les arcs de graphe. Selon le seuil défini plus haut, les arcs non fréquents seront supprimés car il représentent des erreurs.

V. CONCLUSION

La corrélation des messages dans les logs des services web afin de découvrir des protocoles métiers est une étape très importante. C'est pourquoi il est nécessaire de trouver une manière automatique de corréler les messages, sans disposer au préalable d'une information dans les logs (identifiants de conversations) qui les groupe en conversations. Pour résoudre le problème de la corrélation de messages, nous avons pensé à modéliser les fichiers log sous forme de graphes. La première étape consistait à partitionner le log en sous-ensembles n'incluant que des messages échangés entre le service et un client donné. Ceci permet d'éviter le chevauchement de messages de différents clients. Ensuite, générer le graphe de messages des partitions représentant le séquencement des messages, et faire apparaître quelques caractéristiques des services web en propriétés de graphes. Nous avons pu caractériser quatre critères liés aux logs des services web, qui nous ont permis de transformer le graphe de messages bruité en un graphe réduit contenant les conversations. Pour prendre en compte l'incertitude des résultats, nous avons introduit la théorie de l'évidence de Dempster-Shafer. La combinaison de Dempster-Shafer nous a permis de combiner les résultats de différents critères pour trouver le message de débuts des conversations. Le résultat auquel nous aboutissons est non seulement la spécification des conversations, mais aussi leur représentation sous forme de graphe. Ceci permet de faciliter la deuxième étape qui consiste à générer le protocole métier. Comme perspectives, nous envisageons de prendre en compte des logs distribués de services composés (communication entre plusieurs services) dans le but de corréler les messages et de découvrir les protocoles métiers.

RÉFÉRENCES

[1] B.Benatallah, F. Casati, and F. Toumani. Analysis and management of web service protocols. In *ER*, pages 524–541, 2004.

[2] B. Benatallah and H. R. Motahari-Nezhad. Servicemosaic project : modeling, analysis and management of web services interactions. In *APCCM '06 : Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, pages 7–9. Australian Computer Society, Inc., 2006.

[3] L. Dekar and H. Kheddouci. A cluster based mobility prediction scheme for ad hoc networks. volume 6, pages 168–194. Elsevier Science Publishers B. V., 2008.

[4] D. Devaurs, F. De Marchi, and M. S. Hacid. Caractérisation des transitions temporisées dans les logs de conversation de services web. In *EGC*, pages 45–56, 2007.

[5] S. Dustdar and R. Gombotz. Discovering web service workflows using web services interaction mining. volume 1, pages 256–266, 2006.

[6] R. Gombotz, K. Baïna, and S. Dustdar. Towards web services interaction mining architecture for e-commerce applications analysis. International Conference on E-Business and E-Learning, Sumaya University for Technology , Amman, Jordan., 2005.

[7] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Trans. on Knowl. and Data Eng.*, 18(8) :1010–1027, 2006.

[8] H. Motahari, B. Benatallah, and R. Saint-Paul. Protocol discovery from imperfect service interaction data. In *Proceedings of the VLDB 2006 Ph.D. Workshop*, 2006.

[9] H. Motahari, R. Saint-Paul, B. Benatallah, and F. Casati. Protocol discovery from web service interaction logs. In *ICDE 07 : Proceedings of the IEEE International Conference on Data Engineering*. IEEE Computer Society, 2007.

[10] H. Motahari, R. Saint-Paul, B. Benatallah, F. Casati, and P. Andritsos. Message correlation for conversation reconstruction in service interaction logs. In *Technical Report*, 2007.

[11] K. Sentz and S. Ferson. Combination of evidence in dempster-shafer theory. Technical report, Sandia National Laboratories, SAND 0835, 2002.

[12] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining : Discovering process models from event logs, 2004.