Message Correlation and Business Protocol Discovery in Service Interaction Logs

Belkacem Serrour[1], Daniel P. Gasparotto[1], Hamamache Kheddouci[1], and Boualem Benatallah[2]

[1] Université de Lyon, Laboratoire LIESP, Bât. Nautibus (ex 710), 43, Bd. du 11 novembre 1918, 69622 Villeurbanne Cedex, France
{bserrour,hkheddou}bat710.univ-lyon1.fr, daniel.gasparotto@gmail.com
[2] CSE, UNSW, Sedney NSW 2052, Australia
boualem@cse.unsw.edu.au

Abstract. The problem of discovering protocols and business processes based on the analysis of log files is a real challenge. The behavior of a Web service can be specified using a Business Protocol, hence the importance of this discovery. The construction of the Business Protocol begins by correlating the logged messages into their conversations (i.e. instances of the business protocol). The accomplishment of this task is easy if we assume that the logs contain the right identifiers, which would allow us to associate every message to a conversation. But in real-world situations, this kind of information rarely exists inside the log files.

Our work consists in correlating the messages present in Web service logs into the conversations they belong to, and then generating automatically the Business Protocol that reflects the messaging behavior perceived in the log. Contrary to other approaches, we do not assume the existence of a conversation identifier. We first model logged message relations using graphs and then we use graph theory techniques to extract the conversations and finally the Business Protocol. Logs are often incomplete and contain errors. This induces some uncertainty on the results. To address this problem, we apply the Dempster-Shafer theory of evidence. Our approach is implemented and tested using synthetic logs.

Keywords: Web services, business protocols, message correlation, log analysis, graph theory.

1 Introduction

Every Web service has an interface (specified by the language WSDL[1], for example). This interface contains the realizable operations, ingoing and outgoing message types, etc. The interface specified by WSDL is only a functional interface, i.e. it describes only the various invocable methods (the ordered sequences of these are not described). To assure the dynamic and behavioral aspect of the Web services, a new interface is proposed in [1][2]: the Business Protocol. A

[1] Web Services Description Language.

Z. Bellahsène and M. Léonard (Eds.): CAiSE 2008, LNCS 5074, pp. 405–419, 2008.

Business Protocol is a specification of all possible conversations that a service can have with its partners[10]. Recent work shows the importance of specifying business protocols and also propose models to represent them. The authors used Final States Machine (FSM) to model the Business Protocol. The motivations to use FSM are that: (i) FSM is a well-known paradigm with established formal foundations, (ii) it is simple and suitable for modeling reactive behaviors, (iii) it embeds the notion of state, which is useful for applications such as monitoring. Business Protocols provide several advantages: (i) They provide developers with information on how to program the client to interact correctly with the service. (ii) They serve as a verification model of behavioral constraints (make sure that the real service corresponds well to the conception constraints). (iii) A Business Protocol is modeled in the form of a Final States Machine (FSM), which is an easily exploitable visual model by the user (addition of new functionalities, constraints, etc.)[4][10].

Business Protocol provides several advantages and is important for Web services paradigm. In spite of this, a key point is how to reconstitute a protocol with no prior specification?

The idea proposed in this paper is to analyze log files of Web services to extract the Business Protocol. The discovery models depend on the size of log file to analyze. More the log file is long more the probability to have all the possible exchanges between the service and his clients is great. Otherwise, if the log to analyze does not contain all the instances of the service, it will validate at least a part of the model.

Analyze of log files is a delicate task because they are often incomplete, uncertain and contain errors. The first difficulty of this task is the detection of errors in logs. There are two types of errors:

Log Incompleteness. In practice, conversation logs are very often incomplete in the sense that they do not contain all the possible conversations allowed by the service protocol. Incompleteness makes it difficult for a model discovery algorithm to discover even simple models.

Noises/Errors/Interruptions. Various approaches try to solve the problem of handling noise in real-world log files, which are often imperfect. In a real-world web service log file, it is normal to find problems such as omitted/lost entries, swapped timestamps, random interruption, etc. In a protocol execution instance, this will add a lot of problematic sequences that should be considered[9]. Below are some examples that illustrate the issues:

Missing Messages: The logging infrastructure may fail to record one or more messages of a conversation. For example, for a conversation *abcde*, we may have *acde* captured in the log, in which *b* is missing. This type of error happens for various reasons, including bugs in the logging infrastructure or performance degradation.

Swapping Messages: The order of messages as recorded in the log may differ from the real ordering of messages as exchanged between services. For example, for conversation *abcde*, we may find *acbde* recorded in the log. The order of *b* and

c is swapped. This type of error may be due to the granularity of time stamping, delays in the network infrastructures, etc.

Partial Conversations: We call partial a conversation that is interrupted before its completion. For instance, this can be due to network failure, client abortion or service execution exceptions. For example, if *abcd* is the message sequence of a complete conversation, only *abc* may have been exchanged and recorded in the log.

The second difficulty of log analysis task is the message correlation, i.e. grouping log messages in a set of conversations. This step is the first in the protocol discovery process (see Figure 1). The message correlation task can be easily accomplished if we have an information in logs indicating to which conversation belongs every message (this information is called conversation identifier or *cid*). But in real-world implementation, this information rarely exists in the logs.

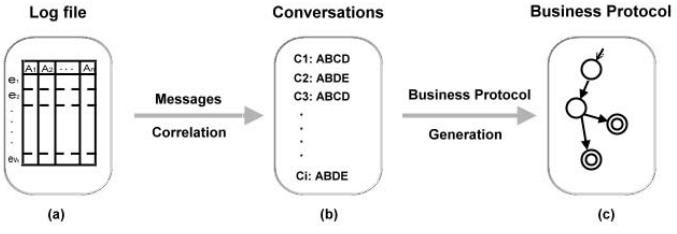

Fig. 1. Business Protocol Discovery Process

This paper is structured as follows. In the next section we give some definitions and we discuss some approaches. Then, in Section 3 we present the details of our approach. Section 4 discusses implementation aspects. We conclude in Section 5.

2 State of the Art

2.1 Definitions

WS Message Log (ML)

Definition 1. *A message log ML is a collection of entries (or events) $e = (cid, s, r, t, m)$, where cid is the conversation identifier, s and r denote the sender and the receiver of message m, and t is the timestamp.*

At this point, our definition diverges from [10]. The log files that we consider are slightly different:

A message log ML is a collection of entries $e = (s, r, t, m)$.

Note that a *cid* is not present as it was in the previous definition. The omission of the conversation identifier will not let us correlate each message to the conversation that it made part. The log is a sequence of events ordered by timestamp and with the identifier of the sender and receiver.

Conversation Log (CL)

Definition 2. *A conversation log CL is a collection of conversations* $CL = \{c_1, c_2, ..., c_n\}$. *Each conversation* $c_i \in CL$ *is a sequence of messages* $c_i = \langle m_1^i, m_2^i, ...m_k^i \rangle$.*(see Figure 1(b)).*

Each client exchanges a set of messages (conversation) to interact with the service.

Business Protocol

Definition 3. *A Business Protocol, as already defined, specifies the potential sequencing of messages exchanged by a particular partner with its other partners to achieve a business goal. It can be modeled as a tuple:*

$$P = (S, s_0, F, M, T)$$

where S is the set of states of the protocol, M is the set of messages supported by the service, $T \subseteq S^2 \times M$ *is the set of transitions,* s_0 *is the initial state, and F represents the finite set of final states. A transition from state* s_i *to state* s_j *triggered by the message m is denoted by the triplet* (s_i, s_j, m)*[10] (see Figure 1(c)).*

Note that in the model, conversations may contain markings indicating whether the message is sent or received by the service. This is easily detected by looking at the sender attribute of the message (If the sender is the service we say that the message is outgoing else the message will be marked as ingoing). In our work, we have not heeded that.

2.2 Approaches

The problem of logs analysis for models discovery relates to several fields. For instance, the main idea of process mining is the analysis of log files to extract knowledge that enhance and improve information systems[5][6][13]. To discover these models, it uses different techniques for the analysis of logs, especially data mining techniques. Data mining is the extraction of interesting (non-trivial, implicit, previously unknown) information or patterns from data in large databases[7].

In the Web services field, the majority of works focus on the process discovery (workflows) and protocol discovery[9][10][11].

In this paper, we are interested in the protocol discovery problem.

In our research domain, there are two kinds of problems: protocol discovery using conversation logs (messages log with conversations identifiers) and using message logs (messages log without conversations identifiers). A lot of efforts have been put in the resolution of the first problem. On the other hand, the problem of having message logs without conversation identifiers is a research field that has been gaining attention recently[11].

Protocol Discovery. The solution in [9] uses conversation logs as input (messages log with conversations identifiers). The conversations are subdivided in smaller sequences and an automatic bottom threshold is applied taking advantage of the information about the conversations. These sequences are building blocks of an oriented graph, called Message Graph. It is initially overgeneralized, because of the way it is built, meaning that refinement is needed to reach expected results. After the removal of erroneous paths (using an enhanced version of the splitting method originally), the Message Graph is then converted into an FSM, minimized using Hopcroft's algorithm and consumed by final user-refinement in order to generate an acceptable Discovered Protocol.

Message Correlation. B. Benatallah *et al*. in [11], bring up this specific problem of protocol discovery *without* message correlation information. The authors define rules (celled Correlation Rules) between messages log to construct conversations. The objective of this work consists in the identification attributes to use for the correlation. For this objective, they use some heuristics. After this, they try to specify the rules and composite rules to apply on these attributes. The authors gave some other directions for solving the problem, and making it the sole article found by us that deals with the entire process of business protocol discovery starting from uncorrelated message log files.

Due to difficulty of this task (message correlation in logs), all papers which treat this problem suppose the existence of *cid*, which facilitates the conversations extraction. Our main contribution consists in the correlation of log messages without assuming the existence of conversation identifiers and then to generate the Business Protocol.

3 Protocol Discovery Framework

The general architecture of our approach is described in Figure 2. It has as objective the Finite State Machine (FSM) that describes the protocol that would have generated the log file.

The log file to be analyzed is generated by the logging infrastructure of the service. In this log, one can find all messages exchanged by the service and its clients (sent and received messages). It is assumed that every client begins a conversation with the service by a single starting message (e.g. login message).

In following, we describe each step in the process of discovery framework.

3.1 Pre-partitioning the Log File

Each message of the log is generated by one partner (client/server) and received by another (client/server). This step consists of grouping the messages exchanged between a same pair client/server. New sub-log files containing these groups are created.

The log partitioning aims to eliminate the conversation overlaps in logs. When a service begins several simultaneous conversations with different clients causes

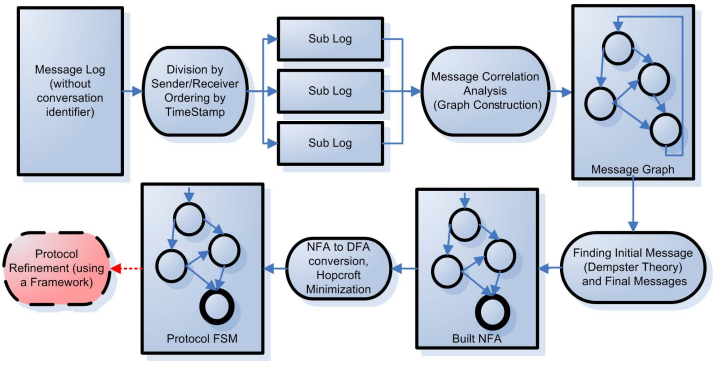

Fig. 2. Business Protocol Discovery from Message Log Files

overlaps of conversations (e.g. two messages which follow each other in this log can belong to different conversations).

Using **S** and **R**, it is possible to divide the log into pairs as $\{\mathbf{S_i},\mathbf{R_j}\}$ and with entries ordered by **T** (see example in Tables 1 and 2).

Table 1. In a first view of a log file, each line is an entry and we observe the Sender (S), Receiver (R), Timestamp (T) and MessageType (MT) already identified and n other unknown attributes

Logfile					
#	**S**	**R**	**T**	**MT**	Other Attrib. (unknown meaning yet)
1	$\mathbf{C_1}$	$\mathbf{S_1}$	**12**	**INIT**	$\{A_1, A_2, ..., A_n\}$
2	$\mathbf{C_2}$	$\mathbf{S_1}$	**15**	**INIT**	$\{A_1, A_2, ..., A_n\}$
3	$\mathbf{S_1}$	$\mathbf{C_1}$	**17**	**WLCM**	$\{A_1, A_2, ..., A_n\}$
4	$\mathbf{S_1}$	$\mathbf{C_2}$	**16**	**WLCM**	$\{A_1, A_2, ..., A_n\}$
5	$\mathbf{C_1}$	$\mathbf{S_1}$	**20**	**REQA**	$\{A_1, A_2, ..., A_n\}$
...	...	...	...	...	...

Table 2. Partitioned Logs

Log File Divided by $\{\mathbf{S_i},\mathbf{R_j}\}$ and Sorted by **T**					
#	**S**	**R**	**T**	**MT**	Other Attrib. (unknown meaning yet)
Partition $\{\mathbf{S_1},\mathbf{C_1}\}$					
1	$\mathbf{C_1}$	$\mathbf{S_1}$	**12**	**INIT**	$\{A_1, A_2, ..., A_n\}$
3	$\mathbf{S_1}$	$\mathbf{C_1}$	**17**	**WLCM**	$\{A_1, A_2, ..., A_n\}$
5	$\mathbf{C_1}$	$\mathbf{S_1}$	**20**	**REQA**	$\{A_1, A_2, ..., A_n\}$
...	...	...	...	...	...
Partition $\{\mathbf{S_1},\mathbf{C_2}\}$					
2	$\mathbf{C_2}$	$\mathbf{S_1}$	**15**	**INIT**	$\{A_1, A_2, ..., A_n\}$
4	$\mathbf{S_1}$	$\mathbf{C_2}$	**16**	**WLCM**	$\{A_1, A_2, ..., A_n\}$
...	...	...	...	...	...

After the grouping, we can order the log by timestamp and observe the message flows of each specific client with a specific server.

From now on, a simplification in the notation of the sub-log files will be used, i.e. $\{INIT, WLCM, REQA, \ldots\}$ and $\{INIT, WLCM, REQB, \ldots\}$, two large sequences of messages, each related to particular client and server pair. To simplify, we use in our examples symbolic sequences as {A, B, C, D, A, B, D, E, A, ...}, where each letter is a reference to a message type.

3.2 Message Graph

Once sub-logs are identified, we can now build a graph that represents the transitions between events and their frequency. The oriented weighted graph is denoted as $G(V, E)$, where:

- $V = \{v_1, ..., v_n\}$ is the set of vertices (messages, in this case, with n different types),
- $E = \{e_1, ..., e_k\}$ is the set of edges (correlations) ($e = (u, v)$ marks a correlation between two message types u and v, means that the message v succeed the message u in the sub-log (not in log file source)).

Both the frequency of vertices $f(v_i)$ and edges $g(e_i)$ are counted.

- $f(v_i)$: number of times that message v_i appears in the log file (for example, in figure 3, $f(A)$= 49, means that the message A appears 49 times in the log).
- $g(e_i)$: $e_i = (u, v)$, number of times that the message v succeed u in the log file (for example, in figure 3, $g(A, B) = 46$, means that the message B succeed A 46 times in log).

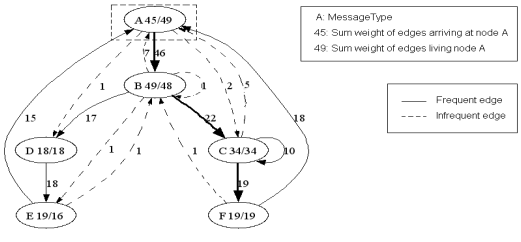

Fig. 3. Message Graph

In this graph, there is only one node for each message type. In the Figure 3, the message graph contains all the messages of the log. The creation of the graph G is given by Algorithm 1:

Algorithm 1. Message Graph

Require: *subLog*
Ensure: G
  $v \leftarrow subLog[0]$
  $f(v) \leftarrow 1$
  for $i = 0, i < (len(Log) - 1)$ */*for every event logged*/* **do**
    $v \leftarrow subLog[i]$
    $v' \leftarrow subLog[i + 1]$
    if $v \notin V$ **then**
      add v to V
    end if
    if $v' \notin V$ **then**
      add v' to V
    end if
    $f(v') \leftarrow f(v') + 1$
    if $(v, v') \notin E$ **then**
      add (v, v') to E
    end if
    $g(v, v') \leftarrow g(v, v') + 1$
  end for
  return G

The union of the various generated message graphs (see Figure 3) represents the complete information contained in the Log file.

Initial Message Type. The first information that we should extract in message graph is the Initial Message Type. We must be able to infer which message is used in the beginning of the protocol execution and which messages end an execution (always considering noise and interruptions). Log files can always contain errors. Thus, nothing guarantees that the first message read in log is the initial message (it can be due to an error at the beginning of the log file or the log to analyze is only a not complete fragment; Consequently, the first message in this file has no certainty to be the initial message).

To find the initial message, we should rely on some evidences.

From the graph built in the previous step, we can extract evidences witch help to discover the initial message of the protocol:

1. P_1 : Frequency of the node: $Score(C_v) = f(v)$. Surely, all conversations must make use of the initial message. It will show a high frequency of observations when compared to other normal messages. Although, we must consider the cases of messages loops, that could increase the occurrence of regular messages when compared to the frequency of the initial message.
2. P_2 : Highest sum of frequencies of infrequent edges that arrive to the node. $Score(C_v) = \sum g(k, v), k \in V, g(k, v) < \phi(threshold)$. Another good evidence is that every time there is an interruption in the protocol, the next message seen from the client will be an initial message (a new instance of

protocol execution). We use the sum of the frequency values of all edges arriving to a message node that are below a minimum frequency threshold. In other words, we take all the weak edges arriving to a node and count their appearances. The best candidate will be the one with the highest value.

3. P_3 : Node with *weight degree out* higher than *weight degree in*. The partitioning of the log into subsets avoids returned edges of last messages of various clients towards the initial node. $Score(C_v) = weight\_degree\_out(v) - weight\_degree\_in(v)$.
4. P_4 : No Self Loops: We assume that there is no self-loop in the initial message of a protocol, so every time we find a frequent one, the message will not be a candidate.

These evidences do not always point to the correct candidate(s) of initial message. The criteria have their own situations where the chosen node is not the real initial message, so we combine the results of the evidences to achieve a better outcome. To do so, we use Dempster-Shafer's mathematical theory of evidence, based on belief functions and plausible reasoning, which is used to combine separate pieces of information (evidence) and to calculate the probability of an event. Detailed explanations this the theory can be found in [12] and a practical use of it in [3].

Each of the three criteria (P_1, P_2, P_3) to be combined gives a score to the node in the candidates set (a score function is associated to every criterion).

The set of candidate nodes, $U = \{C_1, C_2, ..., C_k\}$, to be initial node, are all nodes of the graph except those having self loops.

In order to apply DS' theory, we need to normalize the scores. The normalization equation is shown in eq. 1:

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

By the theory, the uncommitted belief (UB) is defined as an amount of belief that is not assigned to the focal elements by the evidence.

$$m'(C_i) = \left(\frac{m(C_i)}{\sum_{i=1}^{n} m(C_i)} \right) \times (100 - UB) \tag{2}$$

This equation allows revaluing the amount of belief associated with each cluster after the introduction of the uncertainty. $m(C_i)$ is an original mass assigned to the candidate C_i without considering the uncertainty. $m'(C_i)$ is the new mass that considers for the uncertainty. UB is the uncertainty related to the criterion. $\sum m(C_i)$ is the total amount of belief affected to all the focal elements before considering the uncertainty. Note that, by the equation 2, after the revaluation $\sum m(C_i) + UB = 100$ [3].

To calculate the UB for each criterion, it was carried out as follows. We take a log file of which we know the starting message. For each criterion, we test if it manages to find the true starting message. We take each criterion alone and we do 100 simulations on this criterion. For each execution, we look if the

criterion finds the right starting node. The UB associated with this criterion will the number of time that it fails to find the right node (failure rate to find the right node). The UB founded by our simulations are these:

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

The Dempster combination rule is provided by the theory to let us pool evidences from a variety of sources. This rule aggregates bodies of evidence two by two and at each run, and gives a new combined one. The combination rule is commutative and associative.

$$m(A) = m_1 \otimes m_2 = \frac{\sum_{B \cap C = A} m_1(B) \cdot m_2(C)}{\sum_{B \cap C = \phi} m_1(B) \cdot m_2(C)} \tag{3}$$

Note that the criterion weighting can be considered when combining evidences in order to prioritize the most significant ones. After combining the criteria and associating a score to each candidate message, our algorithm selects the highest value as the initial message of the business protocol.

Final Message Types. Now with the initial message, finding the final messages that terminate the protocol execution is trivial: we search for all the messages that have a well supported (above the threshold ϕ) edge going toward the initial message, meaning that in the log file, after this specific message, the next one is most probably the initial message.

3.3 Deriving the Business Protocol from the Message Graph

The Message Graph, after the removal of arcs below a user-defined threshold ϕ and those going toward the initial node, is *overgeneralized*, meaning that it accepts incorrect paths (see Figure 4).

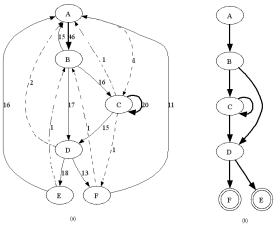

Fig. 4. (a): Message Graph, (b): Reduced Message Graph

In [10], the chosen solution is to perform a graph splitting. They check all the possible paths in the Message Graph and see if it appears in the CL. If not, the path was mistakenly accepted and should be removed from the graph. We did the inverse by building a *overfittet* FSM to our "conversations" and doing a state merging on it.

Candidates to be Conversations. With the information on the initial and final messages of the protocol, we can now extract from the Message Log what we can call "candidates" to conversations.

1. At every occurrence of an initial message, we start extracting the subsequence until finding a next initial message or reaching the end of the log. After this step we have a "candidate" conversation log file, see Figure 5(a).
2. At this point, we have to solve the problem of loop generalization: sequences as $\{A, B, C, B, C, D\}$ and $\{A, B, C, B, C, B, C, D\}$ cannot be seen as different possible executions of the business protocol. To address these situations, we transform each candidate conversation into a small Message Graph. In a graph like this, the loops are described as a way that it can be infinitely executed (see Figure 5(b)).
3. Now the conversations, as message graphs, are somehow generalized, so we group the now-identical ones and sum their frequencies.
4. It can also be supposed that the conversations $\{A, B, C, D, E, F, E, F, G\}$ and $\{A, B, C, B, C, D, E, F, E, F, G\}$ are identical executions, as they share the same set and the edges set E of the first is fully contained in the second and in the same order. So, as an arc will be missing in the graph of the first conversation $(C \rightarrow B)$, but the properties are satisfied, we can join them, as described in Figure 5(c) and is a part of Algorithm 2.

Algorithm 2. Building the NFA

```
Require: ini,ends,cCL //candidates to be conversations
Ensure: FSM
  for all c in cCL do
    if last_event(c) in ends then
      mg(c)=Message_Graph(c)
    else
      del c
    end if
  end for
  for all c in cCL do
    for all c' in cCL do
      if c'! = c and (mg(c) == mg(c') or mg(c') ⊂ mg(c)) then
        f(c) = f(c) + f(c')
        del mg(c'),c'
      end if
    end for
  end for
  for all c in cCL do
    NFA.append_unique(bind_to_NFA(convert_to_FSM(mg(c))))
  end for
  return NFA
```

Finally, after reducing the number of different conversations in the conversation log, we are going to have a refined conversation log file with frequency annotation.

Create and convert a NFA into a DFA (Determinization). Every small Message Graph of every conversation candidate is transformed into a DFA by doing a simple translation, messages becoming arcs and transitions becoming states on the DFA. All the DFAs are binded together using an epsilon transition

from an initial state, as depicted in Figure 5(d). With this binding, it becomes an NFA and will suffer a *determinization* process to become a DFA with its similar states merged, while still accepting the same language.

In the theory of computation, the subset construction is a standard method for converting a nondeterministic finite automaton (NFA) into a deterministic finite automaton (DFA) which recognizes the same formal language. The algorithm used to do the conversion is called subset algorithm. Its description and theory analysis is not in the scope of our work and can be seen in 1979's work of E. Hopcroft in [8].

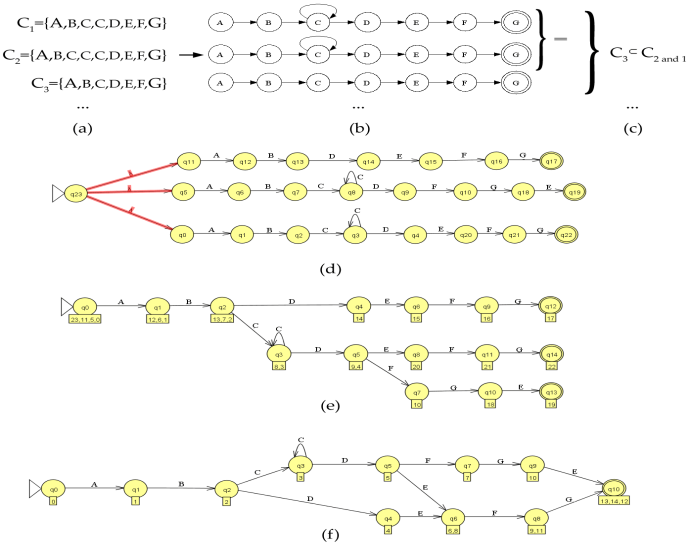

Fig. 5. in (a) we have the list of candidate conversations, in (b) the message graphs of each conversation, in (c) the process of joining the similar sequences, in (d) the NFA binding the messages through the state of the initial message and with the ϵ transitions highlighted, in (e) the NFA after the determinization (subset algorithm), now a DFA and in (f) the final DFA (Hopcroft's minimization)

Ulmann-Hopcroft's minimization algorithm and cycle detection. Ullman-Hopcroft's DFA state minimization algorithm reduces the equivalent states without modifying the accepted language. In Figure 5(d), we have the DFA minimized after the execution of Hopcroft's algorithm.

The idea of the last two algorithms is to minimize the Discovered Protocol without modifying the accepted executions. The generalization is assured when the Message Graph for each conversation candidate was created.

3.4 Refining the Discovered Protocol

Mainly, there are two reasons why user-assisted refinements will be needed: In the presence of noise and, as we use frequency thresholds to separate between

noisy and correct conversations, the latter could be rejected and first ones could be accepted. Also, depending on the desired complexity of the modeled protocol, user may want to simplify its description.

4 Implementation

The best way to contribute to the already existing effort to formalize Business Protocols would be to implement our conversation and protocol discovery algorithm in a framework for the management and analysis of Web service interactions. However, in the first step, we coded an isolated solution using the Python language to execute and simulate our ideas.

For the creation of synthetic logs, simulated Business Protocols were used as inputs, message logs were generated containing multiple clients, the timestamps of the messages and the message names. Of course, the conversation identifier attribute was omitted from the log file. We use the following variables to create the synthetic log:

- Protocol model with probabilities on choices of messages,
- Number of desired conversations,
- Probability of random interruption in an execution of the protocol,
- Probability of random error (swap or omission) in the log file,
- Number of clients and servers of services making part of the log.

The errors were inserted by executing all the desired conversations (with random interruptions) and then making another run through the log, swapping and erasing events in the desired proportion.

(a) P1

P1: Frequency of Nodes			
Node	Score	Norm.	+UB(55%)
A	990	25.05	11.72
B	992	26.12	11.75
D	922	24.26	10.92
E	417	10.97	04.93
F	449	12.60	05.68

(b) P2

P2: Freq. of weak inc. arcs			
Node	Score	Norm.	+UB(10%)
A	3	42.88	38.60
B	1	14.28	12.85
D	1	14.28	12.85
E	1	14.28	12.85
F	1	14.28	12.85

(c) P3

P3: Degree_out > Degree_in			
Node	Score	Norm.	+UB(15%)
A	3	30	25
B	2	20	17
D	2	20	17
E	1	10	8.5
F	2	20	17

(d) Final Score

Final Results	
Node	Final Score
A	**45.43**
B	14.98
D	10.06
E	13.14
F	13.34
UB	03.05

Fig. 6. Application of the Dempster-Shafer Theory

Protocol models were taken from the literature so they could be as realistic as possible.

Once with the synthetic business protocols and their log files generated, we were able to run our protocol discovery algorithm to analyze it.

4.1 Simulation and Results

As an example of simulation, we take a Business Protocol. 1000 conversations were generated with 30% of probability of swaps or deletions in a message, and 10% of the conversations were randomly interrupted. An empirical, user-defined, general bottom threshold of 25% of the highest arc frequency was used for deciding if an arc was considerable or not.

Figure 4(a) show the corresponding Message Graph. The results of the Dempster-Shafter analysis, for finding the initial message, can be seen in Figure 6 (d). Note that the node C does not appear in the set of candidates nodes because it has self loops. After finding initial message, the message graph is reduced by removing the infrequent edges (see Figure 4 (b)).

The UB (uncommitted belief) of each Criterion was calculated based on the observation of the number of times that the criterion had a correct answer, taken different configurations.

The overall results of the Discovered Protocols were satisfactory: even when inserting a few interleaving, the results were as expected (Discovered Protocols very similar to the originals). The *overgeneralization* versus *overfitting* issue was well pondered by the form we did the generalization, and the accepted conversations were at most of the time the same as in the original tested protocols.

5 Conclusion and Perspectives

Message correlation in logs to discover models is a very important step for protocol discovery. That is why it is necessary to find an automatic way to correlate messages, without having an information in logs (conversations identifiers) which groups them in conversations.

Our contribution consists of two parts: Message Correlation in logs into conversations, without conversations identifiers, and generating the Business Protocol in the form of FSM (Final States Machines).

We modeled logs using graphs and we made use of graph theory techniques to extract the conversations and build the Business Protocol. We have extracted four evidences to convert our initial message graph into a reduced one. For this, we apply Dempster-Shafers mathematical theory of evidence.

As perspectives, we intend to consider other attributes for the correlation. There are yet new problems in this domain that need to be explored: as now the composition of web services are becoming a popular procedure, it should be thought about ways to solve the problem when multiple logs with communications from multiple cooperative services are taken.

References

1. Benatallah, B., Casati, F., Toumani, F.: Analysis and Management of Web Service Protocols. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 524–541. Springer, Heidelberg (2004)
2. Benatallah, B., Motahari, H.: Servicemosaic project: Modeling, analysis and management of web services interactions. In: Third Asia-Pacific Conference on Conceptual Modelling (APCCM 2006), vol. 53, pp. 7–9 (2006)
3. Dekar, L., Kheddouci, H.: A cluster based mobility prediction scheme for ad hoc networks. Ad Hoc Networks 6(2), 168–194 (2008)
4. Devaurs, D., De marchi, F., Hacid, M.S.: Caractérisation des transitions temporisées dans les logs de conversation de services web. In: Extraction et Gestion des Connaissances (EGC 2007), vol. RNTI-E-9, pp. 45–56 (January 2007)

5. Dustdar, S., Gombotz, R.: Discovering web service workflows using web services interaction mining. International Journal of Business Process Integration and Management 1(4), 256–266 (2006)
6. Greco, G., Guzzo, A., Pontieri, L.: Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering 18(8), 1010–1027 (2006)
7. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems (2000)
8. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. In: SIGACT News, 2nd edn., vol. 32(1), pp. 60–65 (March 2001)
9. Motahari, H., Benatallah, B., Saint-Paul, R.: Protocol discovery from imperfect service interaction data. In: Proceedings of the VLDB 2006 Ph.D. Workshop (September 2006)
10. Motahari, H., Saint-Paul, R., Benatallah, B., Casati, F.: Protocol discovery from web service interaction logs. In: ICDE 2007: Proceedings of the IEEE International Conference on Data Engineering (April 2007)
11. Motahari, H., Saint-Paul, R., Benatallah, B., Casati, F., Andritsos, P.: Message correlation for conversation reconstruction in service interaction logs. Technical Report, University of Trento and University of New South Wales (2007)
12. Sentz, K., Ferson, S.: Combination of evidence in dempster-shafer theory. Technical report, Sandia National Laboratories (2002)
13. van der Aalst, W., Weijters, A., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)