Using Graph Aggregation for Service Interaction Message Correlation

Adnene Guabtni[1,2], Hamid Reza Motahari-Nezhad[3], and Boualem Benatallah[1]

[1] The University of New South Wales, Sydney, Australia
{aguabtni, boualem}@cse.unsw.edu.au
[2] National ICT Australia (NICTA), Sydney, Australia
aguabtni@nicta.com.au
[3] HP Labs, Palo Alto, CA, USA
hamid.motahari@hp.com

Abstract. Discovering the behavior of services and their interactions in an enterprise requires the ability to correlate service interaction messages into process instances. The service interaction logic (or process model) is then discovered from the set of process instances that are the result of a given way of correlating messages. However, sometimes, the Correlation Conditions (CC) allowing to identify correlations of messages from a service interaction log are not known. In such cases, and with a large number of message's correlator attributes, we are facing a large space of possible ways messages may be correlated which makes identifying process instances difficult. In this paper, we propose an approach based on message indexation and aggregation to generate a size-efficient Aggregated Correlation Graph (ACG) that exhibits all the ways messages correlate in a service interaction log not only for disparate pairs of messages but also for sequences of messages corresponding to process instances. Adapted filtering techniques based on user defined heuristics are then applied on such a graph to help the analysts efficiently identify the most frequently executed processes from their sequences of CCs. The approach has been implemented and experiments show its effectiveness to identify relevant sequences of CCs from large service interaction logs.

Keywords: SOA, Process mining, Correlation, Aggregation.

1 Introduction

As the number of services in organizations are growing and service interactions are getting more dynamic, there is a significant interest in understanding the relationships and behavior of services in the enterprise. Approaches for discovering the behavior of systems and services (also known as process or workflow discovery [10, 4]) take process instances as input. A process instance is a sequence of service messages corresponding to one execution of a process model. In the context of services, a process instance consists of a sequence of messages that are exchanged by services. However, identifying process instances, i.e. correlating messages so that we know which service messages belong to the same process instance, is not

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 642–656, 2011.

always straightforward. This is because information about correlator attributes may not be known to a monitoring service which overseas and captures service interactions in a log. Although each service internally knows how it correlates its messages with those of its immediate interacting partners, this information may not be well documented, or may be buried in the code of the service which is sometimes outsourced or the documentation may become obsolete. Therefore, it is often necessary to perform an automated message correlation in order to identify process instances.

We consider two messages in a service interaction log as correlated if a *Correlation Condition* (CC) is verified. A typical CC may be the equality of two messages' attributes [8]. When messages generated by the same process instance are correlated, the sequence of their CCs can be considered as the "fingerprint" of such process instance. Therefore, discovering sequences of CCs allows to step ahead towards the identification of process instances.

Motivated by the goal of providing a light-weight approach that can help an analyst to quickly identify relevant sequences of CCs, we propose in this paper an approach using message indexation and aggregations to generate an Aggregated Correlation Graph (ACG) that exhibits all the sequences of CCs identified in a service interaction log. In an ACG graph, each node corresponds to an aggregation of messages, and each edge represents a CC between all pairs of messages in the two nodes. Therefore, the ACG graph represents all correlations of messages in an aggregated representation allowing to quickly identify, using weighted nodes and edges, the most frequent sequences of CCs revealing frequent process executions. The approach has been implemented and we offer an interactive filtering/browsing of the ACG graph helpful for analysts to better understand the way messages are correlated and the potential process models those correlations may reveal. In particular, we make the following contributions:

(1) We propose an approach for service interaction message aggregation based on their CCs. The resulting ACG graph is size-efficient and exhibits all the sequences of CCs identified in the log.
(2) We provide a method, based on graph filtering techniques and user-defined criteria, to efficiently identify relevant sequences of CCs and visually browse them using an interactive ACG graph visualization tool.
(3) We have implemented the approach in a tool available online, and performed experiments on a number of service interaction logs. The experiments show that the generated ACG has a stable size regardless of the size of the log being processed. Furthermore, the interactive ACG visualizer allows analysts to quickly find relevant sequences of CCs and identify process instances.

The rest of the paper is organized as follows: In section 2, assumptions and notations used in this paper are presented. We present the service interaction message log format, and define the notations used in this paper. In section 3, we propose our approach and discuss its strengths and limitations. In section 4, we describe the implementation and discuss its results and applications. In section 5, we discuss related work. Finally, in section 6, we summarize the contribution of this paper and present future work.

2 Assumptions and Notations

2.1 Event Log Format and Sample

Process discovery techniques usually assume that service interaction logs have certain format that is useful for analysis. The most common format consists of mono-valued attributes describing service interaction messages. A message is generated by a service and represents an explicit or implicit transition in a process execution. Each message has a set of attributes and their associated values. We consider the set of attributes in a log as $\mathcal{A} = \{a_1, a_2, ..., a_i\}$ and the set of messages in a log as $\mathcal{M} = \{m_1, m_2, ..., m_j\}$. While it is difficult to ensure a global clock with infinite precision to have a total or partial order for messages generated by disparate services, we propose to consider the order that the messages have been inserted in the event log. Therefore, we assume in this work that one centralized log file is used, having exclusive write access to insure that we will always have messages inserted one by one as they are captured. Thus, we define a total order function $\prec$ for messages which corresponds to the order they have been inserted into the event log. The following notations are also used in this paper:

- $\forall\ m\ \in\ \mathcal{M}$, $\mathcal{A}(m) \subset \mathcal{A}$ is the set of attributes represented in the message m.
- $\forall a \in \mathcal{A}(m)$, $\mathcal{V}(a, m)$ is the value of the attribute a in the message m.
- $\mathcal{V} = \{v_1, ..., v_m\}$ is the set of values assigned to attributes of messages in $\mathcal{M}$.

2.2 Message Correlation

Two messages $m_i, m_j\ \in\ \mathcal{M}$ are considered as correlated if a Correlation Condition (CC) is verified. We follow our previous work [4] for the definition of a CC. We consider a CC as a relation between attribute's values of messages based on a correlation function $cf : \mathcal{V} \longmapsto \mathcal{V}$ such that $\mathcal{V}(a_i, m_i) = cf(\mathcal{V}(a_j, m_j))$. For simplicity reasons, we assume in this paper that $cf(x) = x$.

The correlation of two messages m_i and $m_j \in \mathcal{M}$ is denoted $m_i \lll m_j$, in which $\lll$ is the correlation relation, iff $m_i \prec m_j$ and $\exists\ a_k \in \mathcal{A}(m_i)$ and $a_l \in \mathcal{A}(m_j)$ having $\mathcal{V}(a_k, m_i) = \mathcal{V}(a_l, m_j)$.

The Correlation Condition (CC) of two correlated messages can be atomic or composite depending on the number of couples of attributes of the two messages having equal values. The representation of an atomic CC is denoted as the equality of two attributes: $a_i = a_j$ and we note $\mathcal{ACC}$ the set of all possible unique atomic CCs for a given set of attributes. A composite CC is a set of atomic CCs verified for the same couple of correlated messages. We note a composite CC as the conjunction of atomic correlation conditions: $a_i = a_j\ \wedge\ a_k = a_l\ \wedge$ ... For both atomic and composite correlation conditions, we generalize the definition of a correlation condition as follows:

Definition 1 (Correlation Condition). *We note the CC of two correlated messages as a function* $CCond : (\mathcal{M}, \mathcal{M}) \longmapsto P(\mathcal{ACC})$ *such that* $\forall m_i, m_j \in \mathcal{M}\ /\ m_i \lll m_j$, $CCond(m_i, m_j)$ *is the set of* ***all*** *atomic correlation conditions verified by the correlated events* m_i *and* m_j.

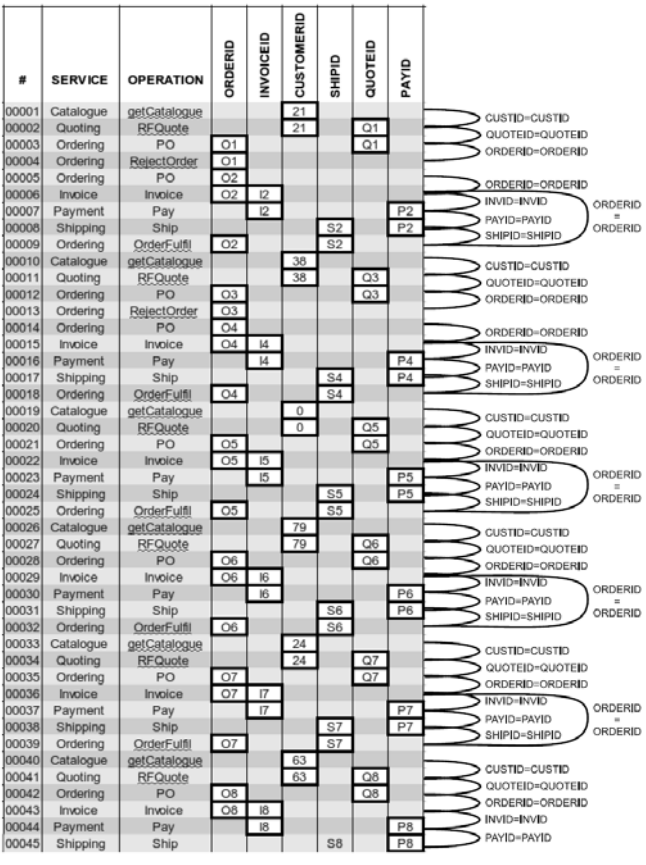

Fig. 1. Example of a real world service interaction log with correlated messages

The example of figure 1 illustrates the message correlations and their corresponding correlation conditions in a small portion of a real world service interaction log related to the Retailer services, respectively "Catalogue", "Quoting", "Ordering", "Invoice", "Payment" and "Shipping" services. For example, the two messages "0001" and "0002" describing respectively the invocation of the catalog and the quoting services, are correlated with the CC CUSTID=CUSTID.

3 Proposed Approach

3.1 Philosophy, Definitions and Properties

Correlation Conditions (CCs) are meant to describe the correlation of two messages. Having a set of correlated messages, it is possible to build a graph in which nodes are messages and edges are correlations. We call such a graph 'Correlation Graph'. Each edge is described using the CC correlating its source and destination nodes. Building such a graph for the entire event log generated a large

graph when large event logs are considered as the number of nodes is equal to the number of messages in the log. Moreover, processing such a graph for process discovery can be computationally expensive.

We propose in this paper a novel approach to efficiently discover and evaluate all CCs from large event logs by using message indexation and aggregation and build an Aggregated Correlation Graph (ACG). The objective is that the ACG graph aggregates and exhibits all the sequences of message correlations (described using their corresponding CCs) identified in a service interaction log using a size-efficient single graph. We define such ACG as follows:

Definition 2 (Aggregated Correlation Graph (ACG)). *We define the ACG graph as a weighted and oriented graph in which nodes and edges have the following properties and notations:*

- ***Nodes properties:*** *Every node is associated with a set of messages. The set of nodes of the ACG is noted* $\mathcal{N}(ACG)$.
- ***Edges properties:*** *Every edge is associated with a set of message correlations. The set of edges of the ACG is noted* $\mathcal{Ed}(ACG)$. *An edge is oriented and links two nodes. Two nodes can be linked with at most one edge. If two nodes* n_1, $n_2 \in \mathcal{N}(ACG)$ *are linked with an edge* $e \in \mathcal{Ed}(ACG)$, *we note* $ed(n_1, n_2) = n_1 \curvearrowright n_2$ *and we say* n_1 *is correlated to* n_2. *The weight* $\mathcal{W}(e)$ *of an edge* $e \in \mathcal{Ed}(ACG)$ *is equal to the number of event correlations associated to it.*
- ***Constraints on message-node association:*** *Every message in the log is associated with one single node in the ACG. Two correlated messages cannot be associated with the same node in the ACG.*
- ***Constraints on message correlation-edge association:*** *All message correlations associated to the same edge share the same correlation condition. Such a correlation condition is noted* $\mathcal{CCond}(ed)$ *for an edge* $ed \in \mathcal{Ed}(ACG)$. *Every correlation of two messages is associated with one single edge in the ACG.*
- ***Root node:*** *The ACG contains necessary one default node called root node where any message* m_i *is placed if* $\nexists\ m_j \prec m_i\ /\ m_j \lll m_i$.

In the Aggregated Correlation Graph, each node corresponds to a set of messages and each edge represents correlations between all pairs of messages in the two sets of messages[1]. The edge's weight represents the number of pairs of correlated messages in the two sets of messages contained in its source and destination nodes. The edge's type corresponds to their CC, ensuring that all correlations represented by the same edge have the same CC.

The correlations of messages are represented in the ACG graph using oriented edges mainly for two reasons. Firstly, time is important in correlation discovery as if a message m_1 is correlated with a message m_2 and $m_1 \prec m_2$, the edge representing such a correlation would be oriented in consequence (from the node containing m_1 to the node containing m_2). The second reason is that the ACG

[1] Only binary correlations are represented (correlations of pairs of messages).

graph is the result of an aggregation of messages into sets of messages (nodes of the ACG) and an edge in the ACG represents correlations between all pairs of messages in the two sets of messages. Without oriented edges, we won't be able to decide which messages would be aggregated. For example, let assume $m_1 \lll m_2$ are correlated with a CC c_1 and $m_3 \lll m_4$ are correlated with the same CC c_1. Should we aggregate m_1 with m_3 and m_2 with m_4 or should we aggregate m_1 with m_4 and m_2 with m_3? The orientation of the edges makes the decision easier as the sources of edges are aggregated together and the destinations of edges are aggregated also together.

Role of the Root Node in ACG. When processing each message sequentially, some messages may not be correlated to messages previously placed into the ACG. Such messages are associated by default with the root node which corresponds to all non correlated messages (singles) and all messages being the starting messages of process instances (starters). Without using such a root node, we would create a separate node for every message not correlated to previous messages which can be very frequent and thus can lead to a larger ACG. By aggregating those events in the root node, we do not loose any information and we reduce drastically the number of nodes in ACG.

Additionally, the weight of an edge corresponds to the number of couples of messages associated with, on one hand its source node and on the other hand its destination node, being correlated with the CC of the edge. Therefore, the more a CC is verified for couples of messages, the bigger is the weight of the corresponding edge(s)[2].

3.2 Step By Step Scenario of Building the ACG

In this section we describe a step by step scenario of building the ACG from a real world service interaction log as illustrated in figure 2. The ACG is built gradually by processing every message sequentially. At each step, an overview of the messages being processed as well as the resulting ACG under construction. Each node of the ACG is labeled using the service operation of its members (messages). The root node has an additional label corresponding to the service operation of its source message (within the root node).

Let's start with the first message in the service interaction log. In the example below, message 0001 is placed in the root node as this is the default node for any message that is not correlated with previously placed messages. As this is the first message to place in the ACG, it is obviously placed in the root node.

A second message, 0002, is correlated with message 0001 with a CC "curtomerid=customerid". However, there is no node in the ACG linked to the root node with correlation condition "curtomerid=customerid". The message 0002 is then placed in a newly created node and that node is linked to the root node using an edge labeled with the CC "curtomerid=customerid". The new

[2] Many edges within the ACG could have the same CC. This is made possible if such CC is involved in many different processes.

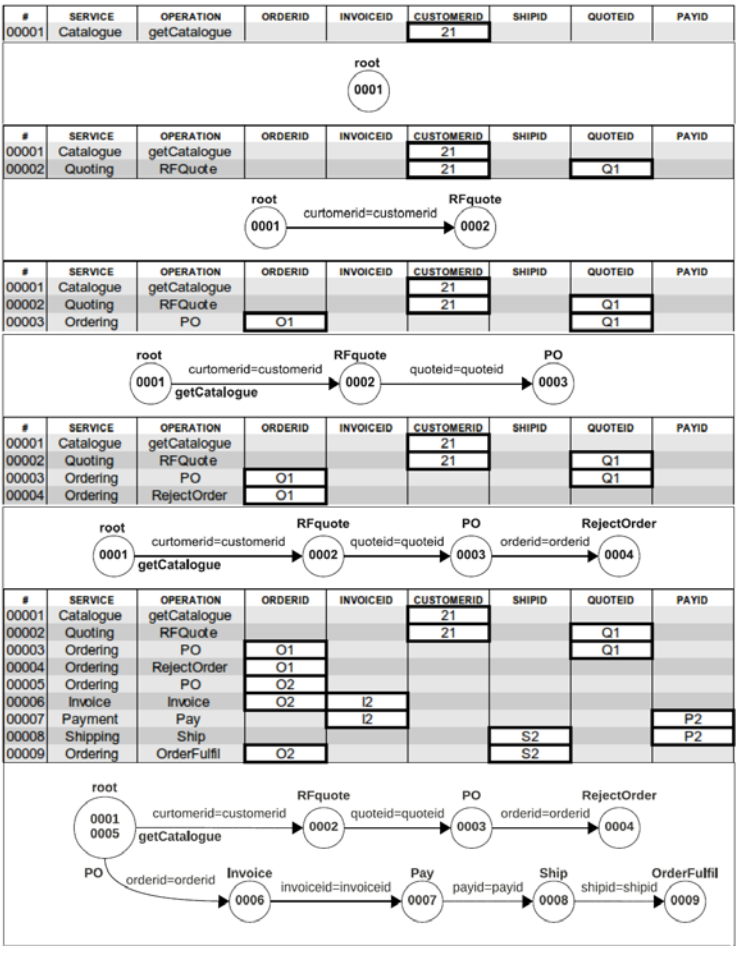

Fig. 2. Step by step scenario of building the ACG

node is labeled with the service operation generating message 0002, which is "RFquote". A similar scenario concerns the next messages 0003 and 0004 as described in the illustration below. Message 0005 is not correlated to any previous message. Therefore, it is placed in the root node. Then, messages 0006 to 0009 are placed in newly created nodes for each of them. Each time a message is placed in an existing node, its size is incremented and its inner and outer edges have their weight incremented. The heavier the edges are, the more frequent are their associated correlations.

3.3 Algorithm for Building the ACG

Building the ACG requires a method to aggregate the correlated messages and generate the expected graph without necessary generating the correlation graph

("non aggregated"). Assuming that the ACG is built by processing messages one by one, sorted by their total order from a service interaction log, the proposed approach follows some rules when processing a message m:

- If m is correlated to previous messages $m_1, m_2, ..., m_p$ which are necessary already associated with existing nodes $n_1, n_2, ..., n_q$ in the ACG then two cases are possible:
  - *CASE 1:* Every message in $\{m_1, m_2, ..., m_p\}$ is associated to a distinct node in $\{n_1, n_2, ..., n_p\}$. In that case, m is associated with a node n such that $\forall 1 < i < p$, $\mathcal{CCond}(\mathcal{ed}(n_i, n)) = CCond(m_i, m)$ and the label of n is the name of the service generating m. If such a node does not exist in the ACG, then it is inserted and its correlations to the adequate other nodes are also added.
  - *CASE 2:* $\exists\ n_i \in \{n_1, n_2, ..., n_q\}$ such that two or more messages from $\{m_1, m_2, ..., m_p\}$ are associated with it. If those messages are correlated to m using the same CC and n_i has the service generating m as label, the first case applies. If those messages are correlated to m using different CCs, each of them is then moved from its existing node to a new node inheriting all incoming edges of the original node. This is done for every node verified until the first case can be applied.
- If m is not correlated to any of the previous messages then it is associated with a predefined node in the ACG called *root node*.

Following those rules, it is possible to gradually build the ACG graph by processing the service interaction log message by message. At any stage of the ACG construction, each message to be processed is matched with the nodes of the ACG (sets of messages) to apply the above rules.

The algorithm described below associates messages with their corresponding nodes using a single parsing of the event log. Additional notations concerning the relation between a message and nodes sharing some of its attribute's values are used as follows:

> $\forall m \in \mathcal{M}$, if $\exists m' \in \mathcal{M}$ associated with n such that $m' \lll m$, we note $n \lll m$ as an order relation between a node and a message meaning that the node has to be correlated to the node associated with the message. In that case, we also note $CCond(n, m) = CCond(m', m)$ the correlation condition between the node n and the node associated to a message m.

For each message m in the message log, the proposed algorithm allows to identify an existing node n in the ACG or add a new node to the ACG which can be associated to m with respect to the ACG properties previously defined in section 3.1 of this paper.

Algorithm 1. Building the ACG graph

Require: $\mathcal{M}$ as the event stream sorted by time
Ensure: Building the Aggregated Correlation Graph of all messages in $\mathcal{M}$
for all $m \in \mathcal{M}$ **do**
  if $\exists n \in \mathcal{N}(ACG)$ such that
    $\forall\, n' \in \mathcal{N}(ACG) \;/\; \exists ed \in \mathcal{E}d(ACG),\ ed = n' \curvearrowright n,$
      $n' \lll m$ and $CCond(ed) = CCond(n', m)$
    and
    $\forall\, n' \in \mathcal{N}(ACG) \;/\; n' \lll m,$
      $\exists ed \in \mathcal{E}d(ACG) \;/\; ed = n' \curvearrowright n$ and $CCond(n', m) = CCond(ed)$ **then**
    m is associated with n
    for all $n' \in \mathcal{N}(ACG) \;/\; n' \lll m$ **do**
      $W(n' \curvearrowright n))++$ {The weight of the edge between the two nodes is increased as a new correlation has been associated to it.}
    end for
  else if $\exists cn \in \mathcal{N}(ACG) \;/\; cn \lll m$ **then**
    Create a new node n in $\mathcal{N}(ACG)$
    for all $n' \in \mathcal{N}(ACG) \;/\; n' \lll m$ **do**
      Add a new edge $n' \curvearrowright n$ to $\mathcal{E}d(ACG)$
      Initialize the weight of $n' \curvearrowright n$ to 1 {This is because only one correlation is associated to it so far.}
    end for
  else
    m is associated with $root$ which is the root node of the ACG. {The event is not correlated to any existing node}
  end if
end for

3.4 Using Inverted Indexes for Efficient Message-Node Association

Inverted indexes are widely used in database systems to efficiently locate information [12]. For example, an inverted index for a collection of documents is a data structure that stores, for each term (word) occurring in the collection, information about the locations where it occurs. Such inverted indexes allow to make the location of items more efficient. In the following, we justify the need and we describe the use of inverted indexes in the proposed approach to ensure an efficient identification of nodes associated to a given message.

Correlating messages is based on the equality of their attribute's values. Therefore, creating an inverted index of attribute's values of all messages in the log is an obvious solution to make correlation identification more efficient. Having such an inverted index of values, every value refers to all its couples of message/attribute having the same value. The inverted index can be formalized as a function *InvInd* mapping values to couples of attributes and messages:

$$InvInd : \mathcal{V} \longmapsto P(\mathcal{A}, \mathcal{M})$$

such that $\forall v \in \mathcal{V}, \forall m_i \in \mathcal{M}, \forall a_j \in \mathcal{A}$, if $V(a_j, m_i) = v$, then $(a_j, m_i) \subset InvInd(v)$.

However, building the inverted index *InvInd* concerns all couples of messages and attributes of the log and makes parsing/updating the index inefficient when large logs are used. Moreover, such an inverted index is helpful to build a correlation graph of the entire event log instead of an aggregated correlation graph.

We propose in this section an inverted index handling correlation of message aggregations (nodes in the ACG) instead of the original messages themselves. We formalize such an index as a function $\mathcal{AggInvInd}$ mapping values to couples of attributes and ACG nodes:

$$\mathcal{AggInvInd} : \mathcal{V} \longmapsto P(\mathcal{A}, \mathcal{N}(ACG))$$
such that $\forall v \in \mathcal{V}, \forall n \in \mathcal{N}(ACG), \forall a \in \mathcal{A}$, if $\exists m \in \mathcal{M}(n)$ and $V(a, m) = v$, then $(a, n) \subset \mathcal{SumInvInd}(v)$.

The proposed algorithm is building the SCG and while processing each message sequentially, the index $\mathcal{AggInvInd}$ is incrementally populated and used. For every message $m \in \mathcal{M}$, its attribute's values are used to access the inverted index $\mathcal{AggInvInd}$, identify nodes correlated to the event and check if an existing node in the ACG is suitable to be associated with the message.

3.5 Using the ACG for Identifying Process Instances

In this section we discuss how the ACG can be used for identifying process instances and the proper use of graph filtering.

Starting from the root node, the ACG is read as different branches, each of them corresponding to a different sequence of correlation conditions revealing a bunch of similarly correlated process instances. The following example of ACG, illustrated in figure 3 (left) corresponds to an ACG generated from a log of 4000 messages. Such graph contains all sorts of correlations discovered from the log and is not easy to read as many of those correlations are non frequent (small node sizes). When the ACG contains a high number of nodes and edges, we propose to apply filtering techniques allowing to reduce the number of edges and nodes and make it clearer to read and easier to interpret.

Assuming that the relevance of a correlation condition depends on the weight of its associated edges in the ACG, applying graph filtering techniques to the ACG allows to discover relevant correlation conditions. Graph filtering techniques can consist of removing low weighted edges, and consequently the potential resulting orphan nodes, those removed edges are non frequent cases. After applying the filtering, it is then possible to read the filtered ACG and clearly identify relevant correlations.

The relevance of a correlation conditions depends on the number of process instances actually verifying that correlation condition in the log. Therefore, graph filtering can play an important role in highlighting potentially relevant correlation conditions from the ACG. An example of filtered ACG compared to an unfiltered ACG is illustrated in figure 3. The size of a node in the graph corresponds to the number of messages associated to it in the ACG. The thickness of an edge corresponds to its weight in the ACG.

3.6 Discussion

In some cases, correlator attributes in the log could have the same value for a large number of messages. This has a major consequence on the resulting ACG

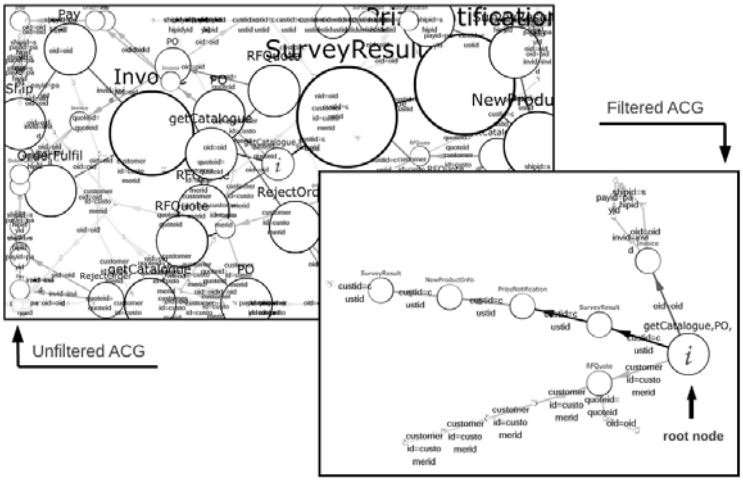

Fig. 3. Example of unfiltered ACG (left side) and filtered ACG (right side)

as two correlated messages cannot be associated with the same node in the ACG and therefore, if a large number of messages are correlated with each others, the ACG would have a large number of nodes and that makes its size inefficient. The relevance of an attribute depends on the diversity of its values within the service interaction log. A threshold is set to ignore attributes having low diversity meaning that most of the messages have the same value for the same attribute.

In some other cases, messages of iterative service calls (loops in a process) can occur in the log. The sequences of correlation conditions referring to a same process model may have various lengths as each execution introduces a variable number of loops, and thus includes a variable number of repetitive subsequences in the ACG graph. Therefore, such sequences are not aggregated together and lead to the identification of several processes.

Also, the approach works well if the log contains correlator attributes and every two consecutive messages belonging to a same process instance are actually correlated using some of those correlator attributes. Therefore, if the process is larger, it is more likely to have broken correlation chains. This is the case when two consecutive messages belonging to a same process instance are not correlated using any pair of correlator attributes. In such cases, a large process will be fragmented in the ACG as multiple smaller processes.

Finally, noise in service logs affects the result of correlation discovery as widely observed [5] [6]. Real-world logs are imperfect, i.e., they are incomplete (correspond to a subset of possible execution) and noisy (e.g., do not record some messages). A known approach to deal with noise in logs is to use a frequency threshold to filter noisy data [11]. In previous work [5], we have presented a quantitative approach for estimating a noise threshold used to filter noise from service logs. In this paper, we assume that the log is free of noise, or has been cleaned from noise in a pre-processing step.

4 Implementations and Experiments

A prototype has been developed to implement the proposed approach and offers 4 user driven steps:

Step 1 - Uploading Event Log Data
The format considered in this paper to represent service interaction log files is the Comma Separated Values (CSV). The user can upload a CSV file using a Web interface. For our experiments, we used a CSV log file generated using HP SOA Manager (SOAM) which is a monitoring tool for Web services. SOAM captures all the messages that are exchanged to/from the set of registered services. The resulting log represents a scenario generated based on WS-I (Web Service Interoperability Organization) for a set of services in the supply chain (Retailer services). Figure 1 illustrates a short sample of data generated by SOAM.

Step 2 - Indexing Messages and Selection of Correlator Attributes
This step allows to build indexes for each attribute in the service interaction log. Such indexes allow to speedup the algorithm for building the ACG. We use information about the size of each index to suggest which attributes are to be considered as relevant for message correlation.

Step 3 - Aggregating Messages to Build the ACG Graph
This step allows to build the ACG by executing algorithm 1. The inverted index described in section 3.4 is also built incrementally during this step. Once the ACG and the inverted index are created, they are stored on the back-end database.

Step 4 - Filtering and Visualizing the ACG Graph
This is the final step and it allows, optionally, the filtering of the ACG, and the visualization of the ACG graph highlighting the relevant sequences of correlation conditions.

An evaluation of the tool has been conducted using service interaction logs of various sizes, based on the Retailer services. Four logs containing respectively 1000, 2000, 3000 and 4000 messages have been processed to generate their associated ACG. A first analysis concerns the impact of the number of messages in the log on the size of the filtered ACG as illustrated in figure 4 (a). It shows the number of nodes in the resulting filtered ACG (with 50% threshold) for service interaction logs of various sizes. This experiment shows that the size of the filtered ACG is relatively the same regardless of the size of the service interaction log. This is due to the fact that filtered ACG shows the most frequent processes and therefore do not include all the ways messages are correlated.

A second analysis concerns the impact of the number of messages in the log on the size of the ACG (unfiltered) as illustrated in figure 4 (b). It shows the number of nodes in the resulting unfiltered ACG for service interaction logs of various sizes. The number of nodes in the unfiltered ACG starts stabilizing to a certain level at around 3000 messages in the log. This is due to the fact that process instances are usually repetitive in the log and the proposed algorithm aggregates similar sequences of correlation conditions. Therefore, when a large number of messages are placed in the ACG, most of the sequences of correlation conditions

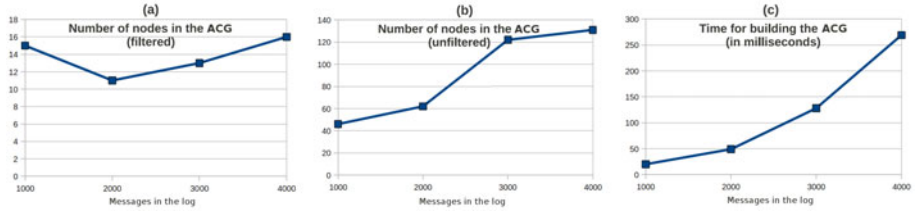

Fig. 4. Impact of log size on filtered ACG size (a), on unfiltered ACG size (b), and on processing time for generating the ACG (c)

become represented in the ACG. Thus, processing more messages would lead to increasing the size of existing nodes and the weight of existing edges but not increasing the number of nodes or edges. The stability in the number of nodes of the ACG makes the proposed approach effective in generating a useful ACG regardless of the size of the message interaction log.

The third and last analysis concerns the processing time needed to generate the ACG. Figure 4 (c) shows the time needed to build the ACG from service interaction logs of various sizes. Although the size of the ACG is stabilized over a certain number of messages, it is required to process all available messages from the log, just in case additional unexpected sequences of correlation conditions may occur. Therefore, it takes longer to process a larger log as illustrated in figure 4 (c).

5 Related Work

The need for automated approaches to message correlation in Web services has first been reported in [4] where a real situation on how to correlate service messages is presented. A categorization of various correlation methods in Web service workflows are presented in [1]. However, no automated support for message correlation is reported. The need for automated approaches for correlation of service messages in composite business applications is also raised in IBM Websphere platform [9]. This work presents an approach for the discovery of correlation identifiers in messages from the log of service interactions. This approach identifies the correlation between message types (e.g., PurchaseOrder and Invoice message types). This approach only considers atomic conditions, while we also consider composite correlation conditions. Also, this approach only concerns correlations between pairs of message types which does not allow to reason at the process instance level.

In related work also Web usage mining investigates the problem of session reconstruction [10]. A session represents all the activities of a user on a Web site during a single visit. Identification of users is usually achieved using cookies and IP addresses, if available, or through heuristics on the duration and behavior of the user. By contrast, when correlating messages, we assume that the correlation

information is in the content of events (their attributes), and the problem is that of discovering which attributes or combination thereof are correlators. In contrast, in time-based session reconstruction, the issue of how to decide on the boundaries of the session is often related to the specification of the time delays between user activities.

In [2], an approach for constructing process instances from sender-receiver information has been proposed in the context of Web services. However, the used correlation criteria does not look at the content of exchanged messages for understanding the correlation between messages.

As a complementary work to our paper, a probabilistic approach based on iterative model convergence is used in [3] for discovering process models from unlabeled event logs. In our approach, we consider the message payload for the purpose of correlation and building the set of process instances.

Finally, in our previous work [8,7] various types of CCs are identified and used to discover interesting CCs and build process instances. The interestingness of CCs are defined based on heuristic-based criteria and user input. The aim of that approach is to prune the space of all possible CCs to only visit the relevant ones by reducing the number of considered CCs depending on their evaluated interestingness. Such approach is favoring the exploration of space of possible CCs. However, the notion of interestingness of CCs is based on heuristics and assumptions leading to ignoring many non-interesting correlations. However, interestingness of correlations is subjective. For example, one can aim to discover rare or exceptional correlations for identifying how a process in an organization executes in exceptional circumstances. Heuristics may be adapted to do so, however, it implies re-executing CC's interestingness evaluation according to every new heuristic. In contrast, our approach allows to try heuristic-based filtering on the ACG without the need to re-generate it.

6 Conclusion and Perspectives

In this paper, we proposed an efficient approach that quickly discovers potentially relevant sequences of correlation conditions from large service interaction logs. The approach reads a service interaction log, and builds an Aggregated Correlation Graph (ACG), which is size-efficient and exhibits all the sequences of correlation conditions identified in the log in a single graph. Such ACG is visualized and refined by the user through filtering parameters to identify relevant sequences of correlation conditions which reveal process instances.

Existing process discovery algorithms require as input the list of all process instances. An important step in future work would be the discovery of process models directly from the ACG, taking advantage of its efficient size and properties. This step is currently under experiments and is based on a mapping between the ACG and a BPMN representation of the process model. The major challenge here is to identify control flow operators (AND join, AND split, etc.). Another future work would be to offer an open interface for process-aware information systems to log in real-time messages directly into the proposed system for immediate inclusion in the ACG. Finally, further experiments with much larger service

interaction logs would unveil the need of taking advantage of the inverted index compression techniques to further enhance the performance.

References

1. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. In: Proceedings of FASE Conference, pp. 245–259. Springer, Heidelberg (2007)
2. De Pauw, W., Lei, M., Pring, E., Villard, L., Arnold, M., Morar, J.F.: Web services navigator: visualizing the execution of web services. IBM Syst. J. 44, 821–845 (2005)
3. Ferreira, D.R., Gillblad, D.: Discovering Process Models from Unlabelled Event Logs. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 143–158. Springer, Heidelberg (2009)
4. Motahari, H.R., Benatallah, B., Saint-Paul, R.: Protocol discovery from imperfect service interaction data. In: Proceedings of VLDB Ph.D. Workshop (2006)
5. Motahari-Nezhad, H.R., Saint-Paul, R., Benatallah, B., Casati, F.: Deriving protocol models from imperfect service conversation logs. TKDE 20, 1683–1698 (2008)
6. Măruşter, L., Weijters, A.J., Aalst, W.M., Bosch, A.: A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. Data Min. Knowl. Discov. 13(1), 67–87 (2006)
7. Motahari Nezhad, H.R., Benatallah, B., Saint-Paul, R., Casati, F., Andritsos, P.: Peocess spaceship: Discovering process views in process spaces. Technical Report UNSW-CSE-TR-0721. The University of New South Wales, Australia (2007)
8. Nezhad, H.R.M., Benatallah, B., Saint-Paul, R., Casati, F., Andritsos, P.: Process spaceship: discovering and exploring process views from event logs in data spaces. PVLDB 1(2), 1412–1415 (2008)
9. Pauw, W.D., Hoch, R., Huang, Y.: Discovering conversations in web services using semantic correlation analysis. In: Proceedings of ICWS Conference, 639–646 (2007)
10. Spiliopoulou, M., Mobasher, B., Berendt, B., Nakagawa, M.: A framework for the evaluation of session reconstruction heuristics in web-usage analysis. Informs J. on Computing 15(2), 171–190 (2003)
11. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
12. Zhang, J., Long, X., Suel, T.: Performance of compressed inverted list caching in search engines. In: Proceeding of WWW Conference, pp. 387–396. ACM, New York (2008)